The Evolution of Groups for Common Pool Resource Sharing

Applications of Genetic Programming to Groups for Computer Game Artificial Intelligence

Alan Cunningham

PhD. Dissertation
Supervisor: Colm O'Riordan

September, 2013

College of Engineering and Informatics
National University of Ireland, Galway

ABSTRACT

The scope and scale of computer games has increased such that, creating unique hand-made behaviours for each character becomes unfeasible. Throughout a typical computer game there are many AI characters with which the player will meet and interact, but only some of these characters will be central to the main story. There is a tendency to rely on template behaviours which are replicated throughout the game world.

This thesis concerns the creation of groups of characters which, through the use simple actions, cooperate and coordinate to survive together. These groups are created automatically using Evolutionary Computation (EC) methods. In order to apply EC algorithms to a domain, the problem being solved will be executed and evaluated a large number of times as solutions are created, altered and refined towards a good solution. As computer games tend to be resource intensive, running thousands of simulations using a game world is not feasible. An abstract representation of a game world is needed.

Selecting group based dilemmas from the social science and economic literature provides a suitable abstract representation. A Common Pool Resource (CPR) dilemma is chosen which models a group's use of a shared resource. Previous studies of human behaviours with this game environment allow for the comparison of the automatically generated solutions against expected behaviours and human performance. It is shown that by introducing irrationality into the solution creation, human-like play can be generated automatically.

By expanding the traditional CPR dilemmas by introducing notions of spatiality, the abstract games move closer to the domain of computer games. In the abstract game model proposed, the notions of character roles are also introduced. This provides several benefits including modelling a familiar convention within computer games as well as providing context based constraints for the evolutionary process. The group behaviours being created must now involve cooperation and coordination, as individuals within the group try to survive in the abstract game world.

A range of parameters and their effects are shown on the creation of group behaviours in this abstract game environment. A selection of parameters is chosen to illustrate the behaviours possible under various conditions. Simulated dynamic elements, derived from likely interactions or experiences an AI character may encounter in a computer game, are introduced. The ability of the EC algorithm to create groups under many different environmental pressures is assessed. The benefits of using a range of situations in the evolutionary process is shown as solutions become more robust to changes in the environment.

A simulated computer game environment is created to assess the performance of group behaviours created in an abstract world. The benefits of creating behaviours in an abstract environment is that behaviours and performances are much easier to evaluate in a quantifiable way and abstractions, typically, are much more efficient to execute. By applying the solutions in a simulated computer game environment, the usefulness of using abstractions to create behaviours is determined.

The performance in the abstract game world does not necessarily correspond to an equivalent performance in the simulated computer game world, due to various factors including a non-determinate action execution times as characters move around the world. By using various selection criteria when choosing the automatically created behaviours, it is possible to find solutions which produce good performance in the simulated game world.

It is shown that solutions created in an abstract environment can, without much modification or translation, be suitable for a computer game environment. This thesis also shows that, by using an appropriate environment, the application of simple individual actions can lead to group behaviours that exhibit both cooperation and coordination in order to survive in a changing world.

CONTENTS

Publications xii

1. Introduction 1

1.1 AI in Computer Games 3

1.2 Computer Games and Evolutionary Computation 7

1.3 Open Research Questions 10

1.4 Thesis Goals and Hypotheses 14

1.5 Thesis Structure 16

2. Background 18

2.1 Video Games 18

2.1.1 History 19

2.1.2 Genres 21

2.2 Research into Game-AI 23

2.2.1 Traditional Approaches 23

2.2.2 Approaches from Academia 25

2.2.3 Industry State of the Art 28

2.3 Evolutionary Computation 30
2.3.1 Genetic Programming 30
2.3.2 Evolutionary Approaches for Game-AI 32
2.3.3 Evolutionary Approaches for Teams 34
2.4 Game Theory 36
2.4.1 Common Pool Dilemmas 36
2.4.2 Evolutionary Approaches for Game Theory 38
2.5 Chapter Summary 39

3. Genetic Programming and Group Based CPR Dilemmas . 40
3.1 Common Pool Resource Dilemmas 41
3.1.1 Game Definition 43
3.1.2 Results of Human Trials 46
3.2 A Genetic Programming Approach 48
3.2.1 Genetic Programming for the CPR Dilemma 49
3.2.2 Evolutionary Process 52
3.2.3 Fitness Function 53
3.2.4 Crossover and Mutation 55
3.3 Experiment 1: Baseline CPR Dilemma 56
3.3.1 Experimental Setup 56
3.3.2 Results 57
3.3.3 Comparison of Evolved Behaviours 60
3.4 Game Variations, Towards Human-like Play 64

3.4.1 Evolving Against Naive Strategies 65
3.5 The Effect of Environmental Pressures 69
3.5.1 The Effect of Randomness or Uncertainty 70
3.5.2 Probabilistic Destruction of the Commons 73
3.5.3 Discussion 77
3.6 Co-evolution and a Group based fitness 78
3.6.1 Fitness Function with Group Performance 80
3.6.2 Results 81
3.7 Chapter Summary 83

4. Evolution of Groups for an Abstract Game 86
4.1 Role-Based CPR Environment 87
4.1.1 Game Definition 91
4.1.2 Agent Roles 94
4.1.3 GP Actions and Costs 96
4.2 GP Approach 97
4.2.1 GP Suitability 97
4.2.2 GP Algorithm 98
4.2.3 GP Parameters 101
4.2.4 The Genetic Program Fitness Function 103
4.3 Experiment 1 - Single Resource 105
4.3.1 Baseline Game 107
4.3.2 Limiting the Action Queue 115

4.4 Experiment 2 - Two Resources . . . 118

4.4.1 Varying the Multiple of Food . . . 125

4.5 Behavioural Differences in a Static Environment . . . 128

4.6 Chapter Summary . . . 132

5. Evolving Groups in Dynamic CPR Game Environments . . 134

5.1 Dynamic Environments . . . 135

5.1.1 Dynamic Elements . . . 136

5.2 Experimental Setup . . . 138

5.3 Health Dynamic Environments . . . 140

5.3.1 Health Dynamic Environment Results . . . 141

5.4 Coping with Death . . . 144

5.4.1 Death Results . . . 146

5.5 Outside Traders . . . 148

5.5.1 Resource Dynamic Environment Results . . . 150

5.5.2 Summary . . . 156

5.6 Round Robin Experiments . . . 157

5.6.1 Results . . . 159

5.6.2 Round Robin Summary . . . 162

5.7 Chapter Summary . . . 164

6. Applications of Abstract Evolved Behaviours . . . 165

6.1 Game Environment . . . 166

6.1.1 Action Translation 167
6.1.2 Game Update Loop 172
6.1.3 Integration of Generated Solutions 173
6.2 Static Environment - Abstract to Continuous 174
6.2.1 Varying Tick Length 175
6.2.2 Modifications and Expert Tweaks 181
6.2.3 Additional Abstract Requirements 183
6.2.4 Summary 186
6.3 Layout Experiments 187
6.4 Dynamic Environment 191
6.4.1 A Dynamic Continuous Environment 192
6.4.2 Continuous Dynamic Results 195
6.5 Discussion 199
6.6 Summary 200

7. Conclusions 204
7.1 Answers to the Research Questions 209
7.2 The Hypotheses 213
7.3 Future Work 216

Appendix 218

A. Evolution of Groups for an Abstract Game 219
A.1 Limiting the Action Queue 219

A.2 Previous Actions 221

B. Evolving Groups in Dynamic CPR Game Environments . . 224

DECLARATION

I, Alan Cunningham, that this thesis is entirely my own work. I have not obtained a degree in this University, or elsewhere, on the basis of this work.

Alan Cunningham

ACKNOWLEDGEMENTS

To my family, thanks for the unending support.

I would like to thank my supervisor Colm O'Riordan for his tremendous guidance and enthusiasm throughout this thesis. He is a constant inspiration and source of knowledge both on topic and off.

This work was completed with the support of the College of Engineering fellowship to which I am grateful for this opportunity.

Thank you to 311 and, to a lesser extent, 306.

PUBLICATIONS

Alan Cunningham and Colm O'Riordan. Evolution of Stable Societies and Social Structures for Computer Games: An Analysis of Dynamic Environments. *19th Irish Artificial Intelligence and Cognitive Science Conference (AICS)*, August, 2008

Alan Cunningham and Colm O'Riordan. An Analysis of Fitness Landscapes in the Evolution of Social Structures for Computer Games. *GAME-ON*, Valencia, November, 2008

Alan Cunningham and Colm O'Riordan. A Genetic Programming Approach to an Appropriation Common Pool Game. *European Conference on Artificial Life*, September, 2009

Alan Cunningham and Colm O'Riordan. Genetic Programming and Common Pool Resource Problems with Uncertainty. *GAME-ON*, Galway, Ireland, August, 2011

1. INTRODUCTION

Computer games are a popular form of interactive digital entertainment which now rival more traditional sources of entertainment, such as music and movies, in terms of revenue generated [Hughes, 2008]. As competition and expectations in this market continue to increase so do the scale of the projects necessary to produce AAA[1] game titles. As noted by DeLoura [2001]: “it's not uncommon these days to hear of development taking 3-5 years, nor to hear of projects with 200-person teams”.

Computer games span many genres, each with their own needs and specialisations. For example, in a racing game simulation the emphasis for the game designers may be a realistic reproduction of the driving experience of the cars. This may be approached from several points of view, ranging from detailed physics simulations to exact modelling of cars, tracks and accessories. A fantasy game on the other hand, does not focus on realism in a traditional sense, may choose to involve the player with an immersive story, challenging puzzles and interactions with other characters and people.

There are some common requirements across this range of genres and specialisations. Games designed to be a single player experience (i.e. one user against the computer), which will be the focus of study in this thesis, typically need to provide non-player characters (NPCs). These NPCs account for every character that is not the player, be they friend or foe, central to the story or a background character. Each genre of game will have specific re-

[1] The term AAA game refers to a large budget game release that typically gets positive reviews and sells well.

quirements for their NPCs and the techniques employed to create opponents for a racing game can be very different from those of controlling a squad of soldiers in a shooter game.

In the many instantiations of computer games, the common goal is to entertain the end-user (i.e. the player) by providing an immersive and entertaining experience. This is achieved in many ways across the various types of games for instance, by giving the player a challenge or by presenting an event or activity that otherwise could not have been experienced. NPCs in computer games provide a way to enhance this experience. They are the opponents that challenge the player's driving skills to be "world class" or allow the player to be the commander of a squad of elite soldiers.

The games industry is very successful with many examples of popular and critically acclaimed games. However, the performance of the NPCs has not traditionally been a selling point of the game. Games designers have been focused on the visual aspects, providing the player with increased levels of detail and realism in a graphical sense. These advancements in visual quality have been aided by the progress in computer hardware which has given more computational power to developers. The standardisation of computing platforms and hardware, such as the mass adoption of games consoles, and the reuse of game engines have encouraged developers to differentiate themselves in ways other than impressive graphics.

There have been many approaches that computer game developers have used to make their games stand out from the competition. Some games employ realistic physics simulation into their games. A testament to the progress that there has been in this area is the creation of complete middleware applications that deal solely with physics simulations (for example [Havok, 1998, Nvidia, 2008]). Other approaches have been to employ professional writers, or to adopt stories from novels and movies, to script storylines for the games (e.g Tom Clancy, Terry Pratchet). The other main differentiation for modern computer games lies in utilising more complex artificial intelligence techniques for providing actions for the NPCs in the games.

Providing more realistic and intelligent behaviours for NPCs in computer games has become more important to developers and to consumers. In older games, more simplistic actions or reasoning were necessary due the limited amount of resources available where poorer graphics rendering helped to disguise the sometimes nonsensical behaviours. The progression of the visual element and the scope of games has lead to the progression of NPC intelligence. Simplistic AI techniques can lead to shallow and unfullfilling experiences and these are now more apparent visually [Fairclough et al., 2001]. For this reason artificial intelligence techniques for computer games has attracted a large amount of interest and has become a large field of research in itself.

1.1 AI in Computer Games

Artificial intelligence techniques used in computer games are generally referred to as game-AI. Typically these techniques are referring to the behaviours that NPCs employ within the game environment. Using more rudimentary implementations of game-AI these behaviours can appear simplistic, repetitive and predicable. This predictability can lead to a reduction in entertainment of replaying a game or a section. Furthermore, these simple actions reduce the player's experience and is the reason why users may prefer to play against other humans where possible [Schaeffer, 2001]. There are, however, other applications of artificial intelligence techniques in computer games which include design authoring tools, procedural level design or texturing and online skill matching of opponents [DeLoura, 2001]. These are beyond the scope of the discussion in this thesis where the focus will be on behavioural AI.

There are several goals of the game-AI research community. The primary focus traditionally was to create NPCs in games with as much realism and believability as possible so that their behaviour in a particular situation simulates what a human player would do given a similar situation. However, game-AI must consider situations where human-like decisions would be detrimental to the player's experience in the game. Similarities can be drawn to work in the field of Game Theory (see Section 2.4), which inspires a large amount of research. Game Theory explores situations where there are difficult choices to be made about which course of action to take and typically, this is the study of decisions which affect an individual who is a part of a group. The individual's decision-making usually involves weighing up short term selfish gains at the expense of the group against long term gains possible from playing with the group.

This leads to the area of research of understanding entertainment and fun within games and then using these techniques to inform the game design such that the experience for the user is enhanced. As Nareyek [2004b] points out, algorithm design is multi-objective and there are trade-offs to be considered when striving for these features. For example, playing against an opponent using an optimal strategy presents the ultimate in challenge for a game player and in certain domains this may be the goal for the game's designers. However, this does not necessarily make the experience fun and in certain contexts the opponent should not be unbeatable, for instance at the beginning of a game where the player is usually learning how the game world works. Game designers and indeed game players have to choose between challenge and entertainment with the appropriate mixture of the two being game-dependent and subjective. A story driven game is unlikely to require the player to hone skills and create complex strategies, where as a shooter game can, but probably won't have the same level of complexity in the story. Of course there are exceptions to these rules, Max Payne[Remedy, 2001] and the Half-life[Valve, 1998] series being two prime examples of action, skill and story.

There are only a small number of commercial computer games successfully implementing more advanced AI techniques. Games like Black & White [EA, 2001] incorporated decision trees and neural networks into its game-AI, and S.T.A.L.K.E.R.: Shadow of Cherynobyl [THQ, 2007] uses an ALife system in its game-AI (a more complete review of game-AI is presented in Section 2.1). These games serve to highlight the possibilities of incorporating more sophisticated techniques but there is a divergence between the research community and the implementations of AI techniques within commercial games. Game-AI researchers typically deal with abstractions and can employ all their computational power to executing their techniques while game designers deal with highly complex worlds and limited shared computational power.

Computer games differ greatly from board games where, traditionally, research from academia has been applied. Computer games offer much more real-time, dynamic and complex environments compared with either complete knowledge (e.g. Chess) or other board games which have been extensively studied [Schaeffer, 2001]. Artificial intelligence solutions for these games have been centred around optimising search techniques within the particular domains. While this leads to increases in computation speed, Nareyek [2004b] argues that the result is extremely specialised which cannot be easily adapted to handle different environments. He refers to the "multi objective algorithm design" and that a shift is needed away from optimisation in an application oriented way.

The interest here is in bridging the gap between abstract solutions and computer game implementation, and the gap between sophisticated academic AI techniques and practical usage within the scope of a game. While it is not proposed that this research will solve all the differences between the two disciplines, it should act as a guide to utilising these techniques in a general way for computer games. The particular focus of this research is on team or group behaviours of agents in games with an emphasis on background characters within large computer game worlds. These characters, while being peripheral to the main story of the game, do add to the believability of the game world by creating a more rich and dynamic environment.

Cutumisu et al. [2006] observe in an example of a modern computer game that “the NPCs wake at dawn, walk to work, run errands, go home at night, and make random comments about the disposition and appearance of the PC (Player Character). However, the behaviours and comments are ‘canned’ and repetitive and NPCs never interact with each other”. They provide many examples of current game-AI and the negative effect poorly performing background characters have on the believability and immersion within the game world. Traditional approaches implemented in current commercial computer games are typically rule based systems and finite state machines [Tozour, 2002a]. However, as there are usually only a limited number of states, scripted AI can greatly reduce a game’s replay value. These methods are also expensive to use in order to fully script each and every character in a large game such that they are performing unique and interesting actions.

As the complexity of the worlds in computer games increases, a larger number of NPCs can be seen inhabiting this world. Shooter games, for example, have been largely responsible for a large portion of group behaviours within computer games. The need for groups of NPCs to be able to correctly assess the situation, cooperate and coordinate their actions to achieve a common goal is a necessity when squads of agents are involved, but these problems are very difficult to solve for developers. These problems are not exclusive to computer games and are found in all facets of coordinated multi-agent systems. There have been many approaches to implementing groups in computer games which include rule based systems, fuzzy and finite state machines, scripting and goal driven agent behaviour reasoning. Games that use goal-driven AI include: The Sims ([EA, 2000]), Empire Earth 2 & 3 ([Vivendi, 2005, Sierra, 2007]) and F.E.A.R. First Encounter Assault Recon ([Sierra, 2002]).

1.2 Computer Games and Evolutionary Computation

As mentioned in the previous section, the scope and scale of creating modern computer games has increased over time as competition in the market, continuing technological progression and the players' demands for more immersive experiences necessitate more ambitious projects. In order to solve this problem and try to understand what makes entertaining and human-like behaviours, work from various classical academic AI fields are brought together in the computer game domain. Academic research into AI techniques for games often try to create solutions that are not only effective at solving tasks but are also automatically generated. Much of this research comes from applying various machine learning techniques, in particular evolutionary computation and neural networks, to computer games.

Evolutionary computation involves the creation of solutions, decision trees or some other internal representation of the agent's cognition automatically [Holland, 1992]. Solutions are initially created randomly. These random solutions are then evaluated, ranked and subjected to alterations akin to those seen in Darwinian evolution. The end result are solutions that hopefully solve the intended problem after a number of iterations of this process. For instance, genetic algorithms create and modify bit-strings which can map to decisions to make in a given scenario for an agent. Cole et al. [2004] use Genetic Algorithms to evolve sets of parameters for AI characters in a video game with comparable performance to behaviours tuned by a human with expert knowledge. Another example, genetic programming, utilises a tree structure to represent the agent control (for a complete discussion of learning and its applications in computer games see chapter 2). Examples of using learning and genetic programming techniques for shooter games are described by Doherty and O'Riordan [2006a,b], where a squad of soldiers are evolved to fight an enemy in a coordinated way using automatically generated decisions trees.

These techniques provide the possibility of creating solutions automatically, which may have many advantages for creating AI behaviours in computer games. Machine learning techniques can generate solutions for a problem even if the solution is not known beforehand, that is, the algorithm only needs a method of evaluating a generated solution. When applying these techniques to games, this can simply be how well the computer agents did in a certain scenario, for example how fast did they complete a lap on a race track, what was the kill/death ratio in a shooter game, etc.

Beyond searching for the best solution for a given problem, this method of evaluation can be useful even in more subjective environments such as training an NPC to act like a human (possibly using a set of human experts for evaluation or using their behaviours for training). In this way, customised solutions can contain subjective measures of good play as opposed to optimal strategies. This area is promising when considered in terms of desirable features of a character, in terms of uniqueness or behaviours within a group, which might differ to the rational decisions based on selfish performance gains. The difficulty of this area lies in grading performance that is not optimal yet is desirable such as opponents that provide a challenge rather than being impossible to beat. There is also the task of distributing group success among the members which becomes a challenge when certain members sacrifice individual performance for the improvement of the group's. If these underperforming individuals are removed and replaced by a higher performing one, the group's performance may diminish.

Another potential benefit inherent in the evolutionary computational paradigm is the creation of a population of solutions. Evolutionary techniques typically employ populations of generated solutions in the process of finding an individual which solves the problem. Each solution is evaluated and allowed to reproduce (typically in proportion to their ranking or fitness score). As the solutions reproduce, they undergo changes in their makeup, such as mutation and crossover. Mutation is the alteration of a subset of a program into an alternative subset and crossover is the swapping of a part of the program with a part from a different individual in the population. These processes allow the population of random solutions to converge to a single solution or a range of equally ranked varying solutions. What this means for the game-AI is that, potentially, from the same evolutionary run multiple solutions of similar good fitness scores could be created while all having very different behaviours. These differences can be influenced by the specification of the fitness function for evaluating a member of the evolutionary population, by rewarding a variety of behaviours or outcomes all the while ensuring valid behaviours are created.

The search for computer game AI is multi-objective and involves of the design of optimal play, simulating human-like performance for a given situation, and trying to maximise entertainment in a domain. An area of inspiration for such multi-objective NPC behaviours lies in game theory. Game theory is the study of strategies of a player in games where their decision and performance is affected by the decisions and actions of all the others in the game. Game theory originates in the fields of Mathematics, Economics and Social Studies and is used as an umbrella term for games which model conflict of decisions in strategic games (see [Osborne, 2003] for a comprehensive introduction). These games have been abstracted and applied to model various aspects of human interaction, for example group based resource sharing. By utilising ideas from these games it may be possible to recreate a tension between players' self interest and what is good for the group, in order to produce a selection of behaviours through evolutionary computation.

Game theoretic dilemmas model many situations from human interaction, for example, the prisoner's dilemma [Rapoport and Chammah, 1965] is representative of what occurs in an arms race situation [Brams et al., 1979].

There are a variety of scenarios that are modelled in an abstract way in order to achieve a better understanding of the interactions between participants. These true abstractions could be expanded to include extra individual characteristics while maintaining the equivalent group dynamics and interactions. Extensions like providing roles, much the same as a job, or by adding extra dimensionality through continuous time environments would allow these dilemmas to occur within groups of NPCs in a computer game.

The domain of evolutionary computation can allow the creation of AI behaviours automatically. There are several restrictions on the application of EC techniques in games, such as time, to create solutions and the non-deterministic nature of evolutionary search. However, EC techniques can be useful if applied in an offline mode, creating the behaviours automatically, extracting and inserting them into the final product. If the correctness of solutions can be ensured during the creation process, the total time for creating a finished behaviour could be reduced.

1.3 Open Research Questions

A number of extensions have been made to traditional approaches to AI in games, such as, in [Cutumisu et al., 2006], a model is shown for scripting NPCs more easily. They develop a framework that allows non-programmers to write scenes with proactive and reactive characters. Spronck et al. [2006] have shown the successful application of online learning applied to scripts. This approach has been extended to include dynamic difficulty scaling within their on-line learning [Spronck et al., 2004]. Other approaches include adding social networks to NPCs to provide more realistic responses from interactions between the player and NPCs [Gruenwoldt et al., 2005, Darken and Kelly, 2008]. These approaches however, consider each individual as a single insulated entity.

The current research into team or group AI typically involves fully cooperative agents that share common goals. There have been many studies in abstract games such as the predator/prey game which involves agents cooperating to chase a prey, for example by Haynes et al. [1995c,b] (see Chapter 2 for more a comprehensive overview). In computer games, many of the early examples of group cooperation come in the guise of large group behaviours such as steering or flocking behaviours [Buckland, 2004]. These behaviours give the appearance that there is some large group based coordination, however, these techniques are based on proximity algorithms. The key difference in this case is that for practical application in a computer game simply giving the appearance of cooperation is enough. Orkin and Kelly [2004] describes the leveraging of information in pathfinding and sensory system as well as sharing information via a blackboard system, gave the perception of coordination, without the need for a dedicated group behaviour layer which was demonstrated in the successful and critically acclaimed game “No One Lives Forever 2” [Monolith, 2002].

Many of the examples of group cooperation in computer games focus on this notion that the group shares the same common goal. However, the idea that all agents are not in any way self interested, as they tend to appear in real life, has not been utilised within the context of computer games. There are many studies examining group behaviour in the field of sociology and economy where assumptions and predictions about players behaviours are made. The most studied of these games is the prisoner’s dilemma [Rapoport and Chammah, 1965] and examples of how cooperation changes with group size can be found in [Bonacich et al., 1976]. These games are abstract and simple when compared to the complex dynamics of a computer game world. They also make certain assumptions about the rationality of the players acting within the games which can be computationally intractable even in these simple environments. These games do, however, describe a framework through which cooperation can be achieved by self interested agents in simple games and under certain circumstances [Axelrod, 1984]. Can these notions of self interest be incorporated into group dynamics for complex worlds such as those found in computer games effectively?

It is possible to define iterated group based games such that self-interested agents may, under certain circumstances, cooperate. These dilemmas have potential to create interesting behaviours for agents for computer games and would do so in an appropriate context as many of the existing dilemmas are used for modelling real world human interactions (for a more in depth discussion see Section 2.4.1). By using these as a starting point is it possible, with modifications specifically for the context of computer games, to create complex interactions between individuals in groups using simplified abstracted actions? What modifications are needed to translate the abstract dilemmas into a computer game domain and are the properties of the group behaviours preserved with this translation?

Abstract social dilemmas are, by their nature and purpose, simplifications of a scenario that is being modelled. Computer games usually try to implement a rich, fully featured world which can result in an entertaining and immersive experience for the player. These features can make analysing behaviours difficult as the increase in complexity can obscure the effects of actions. In order to bridge the gap between social dilemmas and computer games, some features would have to be captured in an abstract representation. Can adding a spatial and temporal aspect to social dilemmas provide an appropriate environment to create behaviours for a game? Can an abstract spatial and temporal modelling be mapped to a game environment?

If agents were solely self interested, then situations where no agents work for the group could appear. These issues are discussed by Hardin [1968] who says that multiple individuals acting independently and solely and rationally consulting their own self-interest, will ultimately destroy a shared limited resource, even when it is clear that it is not in anyone's long term interest for this to happen. These situations could be detrimental to the enjoyment of the player especially if all the agents managed to kill or exploit each other such that none were alive. Examinations of enjoyment and other measures of game-AI would serve to add extra evaluation to these agents. Yannakakis and Hallam [2004a,b] propose several measures of entertainment and interesting behaviours that they incorporate into generating characters. These measures guide the character generation and are then user tested for evaluation purposes. If techniques from abstract games are to be utilised within a computer game setting it is not enough to say that these agents are merely cooperating or are self interested, further examination of the action of the individuals and the group dynamics within which they are acting are needed.

Existing studies involving humans taking part in abstract game theoretic dilemmas can be used to compare the performance of evolutionary algorithms with expected rational behaviour and actual human play. These games can inform the design of abstract computer game worlds which capture many of the same cooperation characteristics that they share. If evolutionary computation algorithms can create behaviours for the abstract games, what does it mean for computer games themselves? Is is possible to translate and utilise behaviours created in an abstract environment in a video game world? Can the performance in a computer game environment be predicted by the performance of behaviours in the abstract game?

The following open research questions have been identified from the literature that are addressed in this thesis:

1. Evolutionary computation can reveal rational strategies for agents in game theoretic dilemmas but often humans are irrational under the same circumstances. Can we create human like play without the input of humans in these game theoretic dilemmas?

2. If it is possible to create human-like play in game theoretic dilem-

mas, can behaviours for groups of characters in computer games be automatically generated by modelling the features of their interactions with shared resource problems? Do they have the same properties as the game theoretic games?

3. Can we automatically assign roles and behaviours to NPCs for games that require coordination and cooperation among roles? Can specific actions required for survival in individuals be evolved when creating behaviours from a group perspective? For the above scenarios, are the generated behaviours robust to environmental changes? How diverse are the generated behaviours?

4. Can behaviours be generated in an abstract environment and applied effectively in a continuous game? Do game elements modelled in the abstract environment create the performances expected in the continuous environment?

1.4 Thesis Goals and Hypotheses

This thesis examines the creation of group behaviours for NPCs in a selection of games using genetic programming techniques. The groups should consist of agents who are self-interested but who also cooperate to some degree to solve collective problems such as those from game theoretic dilemmas. The games include traditional social and economic game theoretic models and abstract computer game environments incorporating spatial and temporal elements. Two separate evolutionary techniques are utilised. One to create behaviours for individuals with whom groups are formed and a second to create behaviours for groups which contain distinct individuals.

We list our hypotheses and discuss the methodology we use to explore each.

H1 Genetic programming techniques are suitable for providing artificial intelligence solutions for groups of agents in computer games.

H2 Evolutionary computational approaches in game theory dilemmas yield human-like performance under certain circumstances. Human behaviours in games are typically suboptimal appearing at times random and reactionary to the environment, but tend to perform better than purely random behaviours.

H3 We can specify a class of game which captures cooperation and coordination and using evolutionary techniques find solutions for them.

H4 The introduction of dynamic elements into the evolutionary process allows for the creation of more robust behaviours across a range of changing environments.

H5 The creation of agent behaviours in abstract economic dilemmas provide suitable behaviours for groups of NPCs in computer games. These dilemmas, if they require extension, exhibit similar behavioural properties in the computer game domain as they did in their original context.

The exploration of genetic programming's suitability as a tool for the creation of game-AI will be a central part of every evaluation and experiment within this thesis. Through the experimentation and analysis of the generated solutions for the groups of characters, the genetic programming process will be evaluated and critiqued based on its suitability in this medium. Suitability will be measured in terms of GP's effectiveness to create useful solutions, the time taken to generate such solutions and finally the ease of application of these solutions to a game setting.

An analysis of genetic programming is performed using a classic group based common pool resource dilemma. This study is used to identify the ability of the GP process to find solutions to difficult games where the performance of human players is known. A comparison is made between the evolved solutions to see in what cases human-like behaviour is exhibited and how the predictions from other similar studies compare with the GP. Using a GP process also allows for the examination of the decision making process within the game, which is something not readily available from the trials done with human subjects.

A game is created which encompasses the properties of an economic dilemma but that has added complexity (in terms of a spatial and a temporal element). This game is used as a test bed for exploring the effectiveness of the GP process for finding solutions in a difficult learning environment. The solutions for this game are then analysed to provide insights into this method. Firstly, to determine the suitability of the generated behaviours to be used in a computer game, through adaption of the abstract to be relevant in the real game.

Secondly, an analysis of the various properties of the created groups, in terms of diversity, robustness and dynamics are determined to guide future behaviour in subsequent generations by adding an alternative fitness pressure. These new measures are analysed to determine if they provide a prediction of the quality of solutions in a real-time game versus their performance in an abstract one. Finally, through empirically testing various methods in a continuous computer game simulation and the abstract game, the appropriate methods are determined for using evolution to create groups of abstract agents for application in more complex environments.

1.5 Thesis Structure

The structure of the dissertation is as follows: Chapter 2 introduces computer games and outlines game-AI techniques, examining traditional techniques used for game-AI and advanced techniques currently being explored to solve game-AI problems. Chapter 3 introduces a common pool resource dilemma from the literature and the exploration of the solution space using genetic programming. A comparison with human performances and other evolutionary methods in the game is performed.

Chapter 4 introduces the notion of group coordination to the common pool resource dilemma by adding a spatial and temporal element. This chapter provides an analysis of this game in a static environment. Chapter 5 introduces dynamic environmental elements derived from potential computer game interactions an AI character may experience into the abstract game world. The group now has to survive in a changing world where outside influences on the game are modelled.

Chapter 6 moves from abstract to continuous games and analyses the application of behaviours created in an abstract environment to a computer game like world. Finally, a summary of the conclusions is provided in Chapter 7.

2. BACKGROUND

Game-AI research began as a field that mostly focused on optimising search techniques with specific application to board games. As computational power increased and the medium of computers became popular, computer games emerged providing entertainment in a more interactive and dynamic way than movies could, while providing a more diverse range of puzzles than those found in board games. As pressures increase to be competitive in this domain, there is a need for advancements in the way that video games are created. As the environments become more complex in video games, a platform emerges upon which technologies can be improved upon, including hardware and AI techniques.

Chapter 2 introduces computer games, providing a brief history and introducing some AI techniques that have been utilised within the domain. Evolutionary learning and genetic programming are then introduced and their applications to computer games are discussed. Following this is an introduction of common pool resource sharing problems from the field of game theory. Finally, examples of how real world problems from sociology and economics are modelled are provided.

2.1 Video Games

This section briefly introduces video games, their history and the genres which may be used to classify them. The terms of “computer games” and "video games" are used interchangeably from now on to mean the same thing for the purposes of this research. Video games are typified by using some electronic display to provide the game information to the player who may interact with this information through a controller, either of a generic or specialised variety. In this way, computer games are played on gaming systems which range from a standard personal computer to a purpose built gaming console such as an XBox[Microsoft, 2001]. This is one distinction between video games and classical board games, in so far as, classical board games come with playing materials specific to that game whereas a large range of video games are played through the same medium.

2.1.1 History

In 1948, the researchers Goldsmith, Grove and Mann received a patent for their Cathode-Ray Tube Amusement Device[Goldsmith Jr. et al., 1948]. By connecting a cathode ray tube to an oscilloscope and devising knobs that controlled the angle and trajectory of the light traces displayed on the oscilloscope, they were able to invent a missile game that, when using screen overlays, created the effect of firing missiles at various targets. As Cohen [2012] points out "although the Cathode-Ray Tube Amusement Device is indeed the first patented electronic game and is displayed on a monitor, many do not consider it an actual video game. The device is purely mechanical and does not use any programming or computer generated graphics, and no computer or memory device is used at all in the creation or execution of the game". In the space of a few short years, this technology saw the creation of the first examples of AI for computer games in an implementation of Noughts and Crosses [Winter, 2013].

The first game commercially became available in 1962 called SPACEWAR! by Russell [1962] which inspired the creation of 1500 arcade machines to play the game. Although the project was a commercial failure it lead to the founding of the games company Atari. However, Atari were not the first console, Magnavox having this honour, Atari released many important games which cemented gaming into the family home, like Pong [Atari, 1972], which are a defining part of the culture of the 1970s. For a comprehensive introduction to the history of video games the reader is directed to Bakkes [2010].

In many ways the progress and history of games is intrinsically linked with the development of computers and games consoles through the years. As hardware progressed, the possibilities of games broadened which helped to popularise gaming and thus increase the interest in further progression of the hardware. The modern state of the art in hardware is fascinating. PC gaming is typified by expensive, gaming specific hardware that pushes the technological limits as gamers make their general machines custom tailored for game playing. The configurations that PC gamers achieve influence both hardware manufacturers and console manufacturers, as components get tested to their limits. Games consoles began life as very specific limited pieces of hardware but the modern incarnation are multi-functional serving not only as high powered games machines but also media centres, web browsers, and a tool for socialising through multiplayer games.

As technology has been integrated more and more into everyday life, games as a form of entertainment, have been popularised and become an integral part of everyday life. Once luxury items, such as the mobile phone, are now common place and enable owners to play games on the move. Video games have become a way for people to socialise and have fun. This uptake has also spurred on the research community as they try to understand human behaviour as well as explore AI techniques.

2.1.2 Genres

There are many categorisations for computer games that, broadly speaking, fall into a set of genres. Indeed, there are many games that can fall between these classifications and some which do not fit in to any of general cases. However, for our purposes of an introduction to the types of games that exist and the main role NPCs take within the games, this list is sufficient. Fairclough et al. [2001] make the distinction between 'action games', 'adventure games', 'role-playing games', 'strategy games' and 'other games'. The other games section can grow quite large with distinctions within it as pointed out in [Schaeffer, 2001] as it can include 'god games', 'sports games', 'simulation games' and 'sports games' for example. Indeed this could be extended further to include 'serious games' which is the study of computer games for purposes other than entertainment, usually as a form of teaching aid.

Action Action games are those games whose primary gameplay mechanics are based on quick reflexes, accuracy, and timing to overcome obstacles. The games use various methods to test hand-eye coordination in an engaging way. The first and one of the most popular examples of such a game is Pong. There are many subgenres of the action game genres. For instance, *Beat-em Up* games usually feature a 2-player fighting game in which a series of button combinations results in the player's character performing a fighting action against their opponent. The game series Street Fighter is an example [Capcom, 1987]. The subgenre of the Shooter game is another very popular genre (so much so that it may well deserve its own distinction). Games in this style are usually played from a first or a third person perspective and feature mainly projectile based weaponry and combat. There are many popular examples such as the Quake [id Software, 1996, 1997, 1999] and Half-life [Valve, 1998, 2004] series of games.

Adventure Adventure games focus on narrative challenges where players solve puzzles and interact with other characters in order to further the story. The two main types of adventure games are text-based adventures or interactive fiction and graphical adventures. Many adventure games are known as point-and-click, which is the main way a player would interact with the game world. Unlike action games, there is no requirement to have quick reactions or good timing. Examples of this genre include Sam & Max Hit the Road [LucasArts, 1993] and Grim Fandango [LucasArts, 1998].

Role-playing Computer Role-playing games (CPRGs) derive their gameplay from traditional role-playing games like Dungeons & Dragons. The player assumes a role of one or more "adventurers" who specialise in specific skill sets while progressing through a predetermined storyline. Gameplay elements strongly associated with RPGs, such as character development through the acquisition of experience points, have been widely adapted to other genres such as action-adventure games. Though nearly all of the early entries in the genre were turn-based games, many modern CRPGs are in real-time. The games can be single, multiplayer or indeed, massively multiplayer (MMORPG). MMORPGs have become extremely popular, with World of Warcraft [Blizzard Entertainment, 2004] being the most famous example.

Strategy Strategy video games focus on gameplay requiring players to plan and manage a large number of units in order to achieve victory. Players typically have an overview of the game world and have to be able to both macro-manage and micro-manage the units under their command. The three main types of strategy games are puzzle games (e.g. Portal [Valve, 2007]), turn-based strategy games (e.g. Sid Meier's Civilization [MicroProse, 1991]) and realtime strategy games (e.g. Age of Empires [Ensemble Studio et al., 1997]).

In this thesis, the focus is primarily on role-playing games. The notion of groups of NPCs working together within a game is the particular aspect that will be explored and these groups mainly arise in modern role playing games in the guise of towns or villages. The methods discussed in the remainder of the thesis are not restricted to this genre in any way. In any case, these definitions merely convey a general classification of games and it is common for games to exhibit features common to multiple genres. This occurs when game designers attempt to incorporate the best elements of various genres into an original game [Slater, 2002].

2.2 Research into Game-AI

This section will introduce the methods that have been used within games throughout their history. The section begins with a brief summary of traditional approaches to game-AI and an introduction to the areas academic AI has been focusing on. The state of the art in the computer games industry is then introduced followed by a section summary.

2.2.1 Traditional Approaches

Many of the traditional approaches to game-AI concentrated on low level problem-solving in the game environment rather than more cognitive abilities of the agents acting within the game. Many early methods involved movement algorithms, that is, ways for the game agents to get around the world. Much of these techniques centred around optimisation of pathfinding algorithms such as that of Dijkstra's shortest path [Dijkstra, 1959] and A* [Hart et al., 1972].

Other types of movement algorithms were introduced based on steering behaviours of the characters. It has been shown that complex group behaviours can emerge when individual agents use simple steering behaviours such as seeking a target, wandering, group separation and cohesion and this is especially apparent when these behaviours are utilised together [Reynolds, 1987]. These techniques originated from the field of animation and were used initially to simulate flocking in birds. They are used in computer games for large scale movements of characters around a game world and range from simple Pac-Man ghost behaviours to the complex steering behaviours used for driving a racing car or piloting a spaceship in three dimensions. Millington [2009] has a full exposition of these behaviours and their applications to games.

Movement algorithms consisted of the majority of the computation needs for NPCs in computer games. However, these methods are not what a player typically thinks about when they are talking about game-AI. Many of the earlier techniques for creating the appearance of intelligence in characters consisted of creating rules that could be followed in a deterministic way. The appearance of intelligence can be created by using simple tricks like difficulty scaling. The player of a game may choose a difficulty which has a direct effect on how the NPCs behave. This may be something as simple as increasing the damage the NPCs can inflict to more advanced behaviours such as altering their accuracy or aggressiveness.

Rule-based systems are utilised to allow the NPCs to make decisions based on their current state within the game and the interpretation of their perceptions. Rule-based approaches are predictable and therefore easy to test and debug, and thus most favoured among game designers. They allow non-programmers to design scenarios for the A.I. system in a game. As there are a limited number of states and because the reaction is based on binary logic, these techniques become predictable. There are of course extensions to these including fuzzy logic and random state changes but the number of potential state transitions are limited. Rule-based techniques are used in many different varieties to implement game-AI including decision trees, finite state machines, and scripting.

Goal driven techniques are utilised to provide NPCs with a range of possible behaviours for a set of situations. Each NPC has an associated set of hierarchical goals which it will try satisfy. These goals can define a single action or be a composite tree of actions. These trees can grow quite large and therefore be quite complicated. This is a useful technique for programming NPCs to act when the player is not in view. The NPC can have a goal to complete until it sees the player at which point a higher ranking goal, such as attack the player, may take over.

Scripting provides the possibility for non-programmers to develop behaviour sequences for AI characters. Scripts are often developed in game independent languages, like Lua, and incorporated back into the game engine. Scripting does not typically lend remarkable intelligence to an agent but it does allow programmers to quickly implement AI solutions for game characters. Not only are scripts used for character prototyping but also for scripting the triggers and behaviours of levels and for programming the user interface. As Buckland [2004] points out, despite their many advantages, scripts can be difficult to develop and debug. Because of the scale and complexity of games resulting in long and complex scripts, manually developed scripts can contain design flaws and programming mistakes [Bakkes, 2010].

The main problem with applying these techniques to a large population of NPCs is the time consuming effort it requires to apply an original behaviour to each individual. The typical solution to this problem is to create a small number of individuals and simply replicate them, exactly or very closely, in order to give the illusion of a large number of unique characters.

2.2.2 Approaches from Academia

Many of the approaches in academia involve applying sophisticated search or learning techniques, such as neural networks or evolutionary computation, to the domain of computer games. There is sometimes a disparity of interest between what game developers want and what researchers in academia wish to achieve and upon meeting, both often acknowledge the disparity and the need to bridge it [Champandard, 2003]. Game developers typically wish to make an entertaining game which is facilitated and enhanced by some game-AI characters, whereas researchers often wish to create the best AI solution or the most cognitively proficient NPC. Game developers are constrained by the hardware they wish their game to run on, as well as many other components of the game which require CPU time, such as physics and graphics. Academics are typically free to utilise most of the computational power available to create advanced solutions without having to consider an average customer's computer specification or the limitation of a console's ability.

Many of the academic approaches to game-AI are based in the domain of traditional board games, card games or puzzle games. Many of these games have been solved including Go-Moku [Allis et al., 1996], Connect Four [Allis, 1992] and Nine Men's Morris [Gasser, 1996]. Van den Herik et al. [2002] provides a good overview on two-person zero-sum games with perfect information, of which these games are examples, that have been solved. More complex approaches to games that haven't been solved like Chess, with Deep Blue's victory over the champion Kasperov, or Jeopardy, with Watson, have been approached with supercomputer powered search successfully.

These supercomputer powered searches, while extremely impressive, cannot be used in all domains. Often times, the game-AI used in the computer game domain is not about winning or losing. As pointed out by Yannakakis [2012], "in contrast to the breadth and multifaced nature of AI research challenges met in game development advances in that field can only be algorithmic with respect to a particular aim (i.e. learn to play a board game) in constrained board game spaces". There are many goals for game-AI including trying to appear human-like by capturing elements of human behaviours within a game, e.g., through imitation learning [Thurau et al., 2004] or simply understanding what enjoyment in games means [Conati, 2002].

Often, the approaches to creating advanced AI solutions for games from academia come from applying advanced search or learning techniques to a particular domain in order to create a set of behaviours that is at least as good as hand crafted solutions. Other approaches offer assessments to subjective questions about the nature of computer games. The notion of defining how a game is entertaining has been studied to some extent [Tozour, 2002b, Nareyek, 2004a]. These notions can then be used to inform how one creates AI solutions using learning [Yannakakis and Hallam, 2007].

One specific field that is relevant for this work is that of creating background or ambient characters for computer games. Cutumisu et al. [2006] used generative behaviour patterns to generate ambient behaviour scripts that have been shown to be believable, entertaining and non-repetitive. This approach has been applied in the commercial game Neverwinter Nights [Bioware, 2002]. McNaughton et al. [2003, 2004] introduce a method to allow non-programmers to create scripts also for the Neverwinter Nights game. This work has subsequently lead to the creation of a freely available tool which may be used to create game stories using a high level menu driven programming model.

Mac Namee [2004] described a similar approach that not only allows agents actions to continue even though they are off screen, but also provides an intelligent agent architecture that allows for role passing and switching between various types of agents. As a form of evaluation for their experimental setup, in which they have agents using their technique play out a scene from a bar, they use human observation coupled with survey questions. This allows subjective feedback to inform the success of AI implementation, which is a typical testing tool of commercial game developers.

A significant effort has been made on the part of academics to bridge the gap between the advanced AI exploration in research with the practical application in actual computer games. There are many AI competitions in different domains such as those hosted by Togelius et al. [2008] and Loiacono et al. [2008] for racing games, which have demonstrated advanced learning techniques such as neural networks, reinforcement learning and genetic programming. The Ms. Pacman [Namco, 1981] game has been studied as an example of 2D arcade games, Super Mario [Nintendo, 1985] is used as an environment for platform games and Starcraft [Blizzard Entertainment, 1998] is used for RTS games.

2.2.3 Industry State of the Art

Techniques like hierarchical state machines (HSM) [Harel, 1987] are an example of a method applied in modern games. These represent an extension to the traditional state machine. Instead of only transitioning from state to state, a HSM allows the transitions between state machines themselves. For example, a certain behaviour could be implemented in a single state machine with a set of states and actions. If there was a need for an NPC to have multiple behaviours transitions could be designed such that an NPC would transition from one machine to another and back. This allows certain behaviours to be implemented and tested in their own state machine while allowing a HSM to use it as part of a more complex behaviour set.

There are many AI techniques that have been applied to games, some of which go through stages of popularity. One of the more popular modern techniques are behaviour trees and as Millington [2009] points out, "Halo 2 [Bungie Studios, 2004] was one of the first high profile games for which the use of behaviour trees was described in detail and since then many more games have followed suit". Behaviour trees break down the a behaviour into a subset of tasks which can range from simply looking up values to playing an animation. Tasks can be grouped together in subtrees which make up complex actions. These actions can be grouped together to form a high level behaviour. The strength lies in allowing each task to be self contained so that they can be built into hierarchies without having to worry about how each subtask in the hierarchy is implemented.

Blackboard systems, while not being a decision making tool itself, have been used to implement cooperation and coordination amongst a set of NPCs in a game. Each NPCs decision making can be implemented using any technique, but each technique could utilise the central storage of knowledge that has been built around each NPC's interactions with the game world.

Though these techniques are available to the AI programmer in computer games, most AI implementations are based on scripting. Developers for linear story-based games will want predictability when it comes to how an NPC acts in a certain place in a level. In this scenario, the game becomes almost like an interactive movie which plays out in an exciting fashion but in a very similar way each time. However, if the game is based in an open world, a more complex system might be necessary to achieve the appropriate AI behaviours that make the game enjoyable.

Yannakakis [2012] explores the most modern games and discusses the more advanced AI techniques that have been employed. This paper does an excellent job at categorising many of the goals for modern computer game-AI as well as providing examples of work in those areas.

Beyond general purpose techniques that may be used to implement game-AI, often the best example behaviours come from games that have been programmed with context specific tricks and cheats [Laursen and Nielsen, 2005]. For example in the game Half-life [Valve, 1998], only two opponents are allowed to attack the player at a time. Lidèn [2004] compared this to the concept of Kung Fu fighting borrowed from the Kung-Fu movie genre, where only a couple of opposing martial arts combatants are battling at any given time. Lidèn [2004] also suggests many other techniques like having AI characters move before firing and miss the first time to improve the player's enjoyment.

Throughout the progress of AI techniques in computer games there has been a pressure for the game-AI developer to be able to rapidly create AI for characters, who can act correctly in the game context while remaining not too computationally expensive. With rapid creation as a motivation, the next section introduces some techniques which allow the automatic creation of solutions for complex problems. These techniques will be used in the contributions of this thesis.

2.3 Evolutionary Computation

Evolutionary Computation (EC) is the general field of search techniques that are inspired by evolutionary processes in nature. An initial population is created and, over a number of generations, successive populations of hopefully better solutions are generated. There are several specific implementations of EC including, among others, Genetic Algorithms (GAs) and Genetic Programming (GP).

2.3.1 Genetic Programming

Holland's work on adaptive systems is based on a biological metaphor and resulted in Genetic Algorithms [Holland, 1992]. Learning is performed through competition among individuals in a population of solutions to a problem. Each solution is represented as a binary string expressing that solution's genes. At each generation, the solutions are evaluated and assigned a fitness. These genes are selected and combined over a number of generations until an optimal solution is hopefully found. The fitness of the individual is used to rank the solution and usually, proportionally select them for subsequent generations. The stopping condition for the process can be a fixed number of generations, a fitness score or some other evaluation of the solutions.

Koza's work on Genetic Programming [Koza, 1992] was motivated by the representational constraint in traditional Genetic Algorithms [Haynes et al., 1995a]. Genetic Programming adopts a different way to represent the members of a population. A tree is used to represent the program where each node is represented by

- An element of the function set (some function that takes one or more arguments)
- An element of the terminal set (one of the possible leaf nodes)

Prior to evolution, the initial population of programs is randomly generated. Such programs are comprised of functions and terminals in the problem domain. The developer needs to specify all of the functions, variables and constants that can be used as nodes in the parse tree. The root is chosen randomly, then symbols are chosen for each argument and the tree is recursively built until a terminal is selected. In traditional GP, all of the function and terminal set members must be of the same type [Haynes et al., 1995a]. There are many genetic operators used in GP including crossover (selecting subtrees from two members of the population and switching them) and mutation (altering some or all of a subtree of a member), but others also include reproduction, permutation and editing.

Strongly Typed Genetic Programming (STGP) is a variation on standard GP that provides for adding extra constraints to the evolutionary process [Haynes et al., 1995c]. Constraints can be added to the creation of the trees to ensure that only sensible combinations of functions and terminals occur. In a similar fashion the reproduction methods of STGP are also constrained to ensure correctness of trees. The crossover points that may be chosen are limited by type and mutation occurs in a more controlled way. The constraints are problem specific and based on the programmer's domain knowledge. The advantages of this system are the reduction of the possible parameter space, insofar as not every possible combination of functions and terminals needs to be explored. There is also a performance improvement that offsets the extra cost of constraining the trees where no evaluation time is wasted on trees that are logically incorrect or programmatically invalid for the particular problem domain. When the term GP is mentioned in the following chapters, it is referring to this constrained version.

2.3.2 Evolutionary Approaches for Game-AI

There are a small number of examples of learning within commercial games but these methods have proven to be quite important unique selling factors in these games. Learning techniques can appear in many different forms while sharing many similar characteristics. Learning can be achieved either online, offline, be direct or indirect, supervised or unsupervised and either human or computer controlled.

Online learning means that the AI that the NPC will use is learned as the game is being played whereas offline means that the AI is trained beforehand and is then fixed into the final shipping product. Indirect processes implies that the agent extracts information from the game world that is used by "conventional" AI to steer behaviour with the AI designer specifying which information to use and how to change the behaviour. Direct learning means that an agent's behaviour is adapted by testing modifications against their effect on the game world. Supervised learning involves immediate feedback on success of a behaviour and is usually followed by indirect learning. Unsupervised learning only gets feedback on success of a behaviour after observation and is usually followed by direct learning. Human controlled simply means that a human player is directing the learning through some feedback and thus this learning is typically both online and direct and is usually an element of the gameplay. Computer controlled refers to all other techniques which do not require feedback from a human for evaluation.

Black & White [EA, 2001] used learning decision trees to achieve new behaviours for the characters. The human player provides feedback on their creature's actions by giving rewards or punishments to the creature and thus it was able to learn which behaviour the human player wished to see. This was achieved by using reinforcement learning which was online, indirect, supervised and human-computer controlled.

The game series Creatures [Creature Labs, 1996] used evolutionary algorithms to develop new creatures. It also used a neural network for decision making. The creature creation had forms of online, direct and supervised human controlled learning. The behaviours were defined with direct, unsupervised, computer controlled learning. The main selling point of these games was the fact that learning techniques were applied and partially human controlled.

Colin McRae Rally 2.0 [Codemasters, 2004] used a neural network trained in an offline manner for its driving behaviours. This was an example of offline, indirect, supervised, computer-controlled learning. The game Max Payne [Remedy, 2001] used automatic difficulty scaling which is an example of online, indirect, supervised, computer-controlled learning.

Although these games contain learning or an application of learning algorithms, they have been limited in their scope, especially with online learning for a number of reasons. Restrictions are incurred due to the amount of processing power available to the learning algorithm and are applied to ensure that appropriate or entertaining solutions are generated automatically. A strength of the learning process is its non-deterministic nature which can allow many diverse solutions to be generated but this non-deterministic nature is also a weakness. A game would be frustrating, rather than interesting, if the NPCs had learnt nonsensical behaviours. A similar restriction exists to maintain the balance between a fun gameplay feature that requires human supervised learning and the repetitive task that a human will endure in evaluating actions.

In contrast, Doherty [2009] points out that “GP has been shown to be successful at evolving strategies for games that have a clearly defined rulebase of possible moves, such as board games (Chess endgames [Hauptman, 2005] and Othello [Eskin and Siegel, 1999]) and simple computer games (Pac-Man [Koza, 1992, Collin and Eglen, 1998], Tetris [Siegel and Chaffee, 1996] and Snake [Ehlis, 2000])”.

2.3.3 Evolutionary Approaches for Teams

Teams are typically evolved as a cooperative unit where the goal of the team is also the goal of each team member. Under these circumstances teams can consist of homogeneous, heterogeneous or hybrid agent behaviours. As Doherty [2009] states, “in a homogeneous team, all agents use the same logic controller to define their behaviour, whereas in a heterogeneous team, each agent has its own logic that controls how it behaves. Hybrid teams have also been suggested that encapsulate aspects of both homogeneous and heterogeneous teams. The choice of team approach is important as the wrong choice could significantly affect the quality or capabilities of the emergent teams”.

There have been several studies into the creation of teams or groups of agents using evolutionary programming methods. Haynes and Sen [1997] explore several variations of crossover operators for teams and rank them in terms effectiveness for a given problem. Most of the work in this area is in the field of cooperative multi agent systems. For a comprehensive review see [Panait and Luke, 2005].

Doherty and O’Riordan [2006a,b] create an example of these cooperative techniques applied to games where the authors evolve team tactics for agents in a shooter game using genetic programming. The team of agents are pitted against one large enemy and through evolved team work are able to defeat this enemy. Doherty [2009] conducts a full analysis of the evolved behaviours as well as the effects of communication on the performance on this group of agents. The cooperative agents are evolved in teams where agents hold a fixed position, from an evolutionary point of view, such that they may crossover with other members in the same position. This evolutionary strategy encourages team-specific roles to emerge.

Although there are not many examples of GP applied in computer games, GP has been shown to be successful in a number of simulated domains. This is relevant for the research in this thesis, as abstractions of environments and agents form the basis of behaviours for a computer game environment. Examples include evolving food collection behaviours for artificial ants [Lalena, 1997] and the creation of sporting strategies for teams of volleyball players [Raik and Durnota, 1994] and teams of soccer players [Luke, 1998, Ciesielski et al., 2002]. Sumo-wrestling strategies for real robots [Sharabi and Sipper, 2006] and fighting strategies for robot tanks [Shichel et al., 2005] have also been evolved using GP.

In these examples, the teams are created to fight a common enemy or try to achieve a common goal. The teams are typically fully cooperative insofar as each agent's priority is the team's performance. In the next section, game theory is introduced. This field of study combines self interested agents into groups and studies their interactions. The notion of evolved teams containing self interest agents is of interest to this thesis.

2.4 Game Theory

Game theory is the study of strategic interactions among rational players. The main idea of game theory is to model the behaviour of interacting agents under conditions of uncertainty with a finite number of strategies. The basic notions of game theory include players (decision makers), actions (strategies), payoffs (benefits, rewards) and preferences over payoffs (objectives).

The players strategies can change over time through various techniques such as copying others, individual and population learning [Macy and Flache, 2002] or evolutionary computation [Axelrod, 1984]. Macy and Flache [2002] also explore the changes in the agents' behaviour as the relationship between the payoffs for actions change i.e. as the game changes from prisoner's dilemma to the stag hunt to a game of chicken. For a good introduction to these games and the search for cooperation within them see [Kollock, 1998].

Various extensions have been considered to these games. For example, the most popular of the game theoretic games is the prisoner's dilemma. Extensions to this game have considered extending from a 2 to n player game and the implications of this. Bonacich et al. [1976] explore group size and its effects on cooperation rates. Other aspects including allowing variable and continuous rates of cooperation have been studied [Wahl and Nowak, 1999].

2.4.1 Common Pool Dilemmas

Common pool dilemmas concern themselves with groups of agents who share some common resource. The study originates from Hardin [1968] describing a situation in which multiple individuals, acting independently, and solely and rationally consulting their own self-interest, will ultimately deplete a shared limited resource even when it is clear that it is not in anyone's long-term interest for this to happen. In economics the problem has been dubbed a common pool resource dilemma and typically consists of a resource (e.g. water or fish), to which a group has access. There may be a cost for access, from which a return is determined and is usually based on the amount of resource extraction by the group. The resources are usually protected by rules or agreements in order to allow for its continuous exploitation. Members of the group can attempt to gain extra from the resource at the cost to the remainder of the group and at a risk to the resource itself.

There is a great deal of literature available on specific field based CPRs such as the fishery [Gordon, 1954]. Field based studies are carried out on real-life occurrences of CPRs. For a review of the literature see [Furubotn and Pejovich, 1972]. The focus is on studies that are more comparable to the computer agent simulations that are employed. Ostrom et al. [1994] discuss CPR problems from the point of view of lab experiments with human players and a study of common-pool scenarios in the field. Their main focus from the experiments is the appropriation game with a view to discover how changes to the rules of the games alters the rate of cooperation. They find that both, the ability to communicate and to have sanctioning in games, increases the players' yield. This game will be discussed further in Chapter 3.

Schlüter and Pahl-Wostl [2007] model both, centralised and decentralised mechanisms for controlling water usage in a river basin as a CPR. They show that when there is only a single usage for the water the centralised system works best. Diversification of resource use, for example irrigation and fishing, increases the performance of the decentralised regime and the resilience of both. Steins and Edwards [1999] introduce collective action in multiple-use common pool resources and using examples from the field, discuss the problems with managing such resources.

2.4.2 Evolutionary Computation Approaches to Common Pool Resource Dilemmas

The research into the El Farol dilemma is rooted in a game theory problem set in a bar in Santa Fe, California [Arthur, 1994]. The problem is this: there is a small bar that is quite popular, but because of its size it can lead to overcrowding. So the scenario arises that if less than 60% of the population go to the bar, they'll all have a better time than if they stayed at home. If more than 60% of the population go to the bar, they'll all have a worse time than if they stayed at home.

This problem, and variations on it, have been explored with the one of the more prominent examples being the Minority Game [Challet and Zhang, 1998]. This game has similar characteristics to the El Farol problem. The game consists of an odd number of players (>1) where each player is given a choice of either A or B. The game then rewards the players who are in the minority group. The cost/payoffs are similar to other dilemmas, where players are trying to maximise their own profits. This game is well studied in the fields of physics where it originated with the interest being in the dynamics of players strategies with varying payoffs and even in evolutionary computation [Greenwood, 2009] where deception and collusion were studied.

Common pool problems have been discussed in the field of artificial life. Epstein and Axtell [1996] show how a common resource is consumed in an agent based modelling paradigm under various conditions. Bousquet et al. [2001] discuss the use of game theory and agent modelling as an approach for simulating resource management issues.

There are various CPR dilemmas that are modelled in this field. There are not many applications of GP to this domain where there is potential success for this algorithm. Firstly, it provides a less constrained cognitive approach over GAs, as the tree structure allows for a more diverse variety of solutions

[Laramée, 2004]. Secondly, because the GP process creates decision tree structures, the reasoning used by agents can more easily be understood.

2.5 Chapter Summary

This chapter has outlined the main research themes of this work as well as introducing many of the background materials that the work relies upon. The domain of video games was introduced, with a brief introduction to the classic approach to problem solving that both industry and academia have employed. The concept of evolutionary computation was discussed as well as the specific implementation of Genetic Programming which will be the main application focus of this research. Genetic Programming provides a mechanism to generate solutions for problems automatically without knowing the correct solution in advance while also maintaining readability in a tree based representation. An introduction to Game Theory was then provided with the more specific subfield of common pool dilemmas being discussed. CPR dilemmas are used to model real world scenarios where agents share resources. Some evolutionary computational approaches to these problems were then discussed.

3. GENETIC PROGRAMMING AND GROUP BASED CPR DILEMMAS

In this chapter, the usefulness of Genetic Programming to the application of solving a group based Common Pool Resource dilemma is explored. Firstly, a game from the literature is selected and the results of studies from previous work in varying fields are discussed. This previous research presents a study conducted with human subjects. These human trials are used to compare the behaviours generated in an evolutionary approach.

Secondly, an application of evolutionary computation to the CPR game is presented. Genetic Programming is chosen for a number of reasons: GP allows less constrained search than other methods, decision making is an inherent part of the GP algorithm which enables the possibility of human like reactive behaviours and it allows the understanding of the resulting behaviours by creating decision trees that are easily readable.

This chapter addresses hypothesis **H2** described in Section 1.4. This hypothesis states that evolutionary computational approaches in game theory dilemmas yield human-like performance under certain circumstances. GP is applied to the same game that the human subjects played to establish a baseline for comparison with the human behaviours and to investigate the claims of the game theoretic predictions for the players.

Experiments are designed to test the GP's ability to create reactive behaviours in order to counter many types of strategy. A list of extensions to the game are discussed along with their implications for behaviours. These extensions study the effect of the environment on the evolved behaviours. Where applicable, a comparison to human players's and other evolutionary approaches' performance in this domain is conducted. Finally the chapter is summarised providing a discussion of the results of the experiments. [1]

3.1 Common Pool Resource Dilemmas

Common pool resources (CPR) are essentially a shared resource from which people or agents, who are sharing it, may extract some of the resource at a cost. The resource can typically be consumed by any agent, either freely or under regulations, and as such this affects all other agents wishing to share the resource. A characteristic of these resources is that they are not infinite and as such, they may be damaged or destroyed by over extraction. An example of such a resource is a fishing grounds. The fishing grounds is shared among fishing vessels who agree to fishing quotas and to fish using certain nets. However, if a vessel was to violate these regulations in order to increase their catch it would have several effects. Firstly, there is an initial economic gain for that fishing vessel. Secondly, in the short term, there is a reduction in fish for others sharing the grounds to catch. Finally, there is the long term effect of potentially destroying future fish stocks as there are now less fish to reproduce. This is an example of how acting in one's own interest can provide a short term gain at the expense of others and at the

[1] This chapter is based on the following publications :
Alan Cunningham and Colm O'Riordan, A Genetic Programming Approach to an Appropriation Common Pool Game, *European Conference on Artificial Life*, Budapest, Hungary, September, 2009
Alan Cunningham and Colm O'Riordan, Genetic Programming and Common Pool Resource Problems with Uncertainty, *GAME-ON*, Galway, Ireland, August, 2011

expense of future possible gains. In this case, if everyone sticks to the quota the group as a whole does better and the fish stocks are preserved.

CPR dilemmas are ones which try to model this interaction between short term selfish behaviour and long term group aspirations. These games model a variety of real world economic and social scenarios where many individuals are sharing resources. There are many different types of common pool dilemma that can be categorised as either appropriation or provision problems. In appropriation problems, the levels of return for a given level of input are assumed to be known. The problem becomes one of excluding potential beneficiaries and allocating the returns from the pool. Provision problems are related to creating a resource, maintaining or improving the production capabilities of the resource or avoiding destruction of a resource. In provision problems, the group benefits from the pool resource regardless of whether or not they paid into it. For example, the provision of a public good like a bridge benefits the entire group once enough resources have been allocated by at least some of the group. Those who do not contribute do not incur the cost of the production of the public good and yet, still enjoy the benefits.

There are many variations of sub-problems but for the purposes here, this classification will suffice. Even with these two categories, there exist a large range of games with different properties. The construction of the public resource of a bridge is a form of the provision problem. This particular instance of the game can be modelled using a step-wise function, which states that unless sufficient participants in the pool contribute the correct amount no one will benefit from the resource. In this case, the pay off is an all-or-nothing situation, where a partially constructed bridge, no matter how close to completed, is of no benefit to the contributors of the resource. In most real-world settings, both provision and appropriation problems exist and are often interrelated.

In this chapter, appropriation problems will be the main focus. In order to measure performance in these games, concepts of economic efficiencies are used. A comparison with optimal solutions is used to gauge how well any group or individual has performed. Predictions about the behaviours of the members of the group are also made on the basis of establishing the levels of performance based on Nash equilibria for the games [Nash, 1950].

"A Nash equilibrium, named after John Nash, is a set of strategies, one for each player, such that no player has incentive to unilaterally change her action. Players are in equilibrium if a change in strategies by any one of them would lead that player to earn less than if she remained with her current strategy. For games in which players randomize (mixed strategies), the expected or average payoff must be at least as large as that obtainable by any other strategy" [Shor, 2005].

3.1.1 Game Definition

Ostrom, Gardner, and Walker [1994] developed a series of laboratory experiments utilising human subjects in order to investigate the correlation between the behaviours of the human players and the behaviours predicted by non-cooperative game theory. The subjects in the original experiments were all undergraduate students recruited primarily from a 'Principles of Economics' class. The students are familiar with these type of games, are briefed on each of the experiments beforehand, and all have the necessary skills to calculate returns and outcomes within the game.

The experiments designed all had multiple participants acting simultaneously in repeated rounds. In each round, the participants received an allotment of tokens and then decided separately where to invest the tokens. The tokens can be seen as a contribution of money or effort towards receiving a reward in return. In these games, there are two markets into which tokens may be invested. The first offers a fixed return based on each individual's

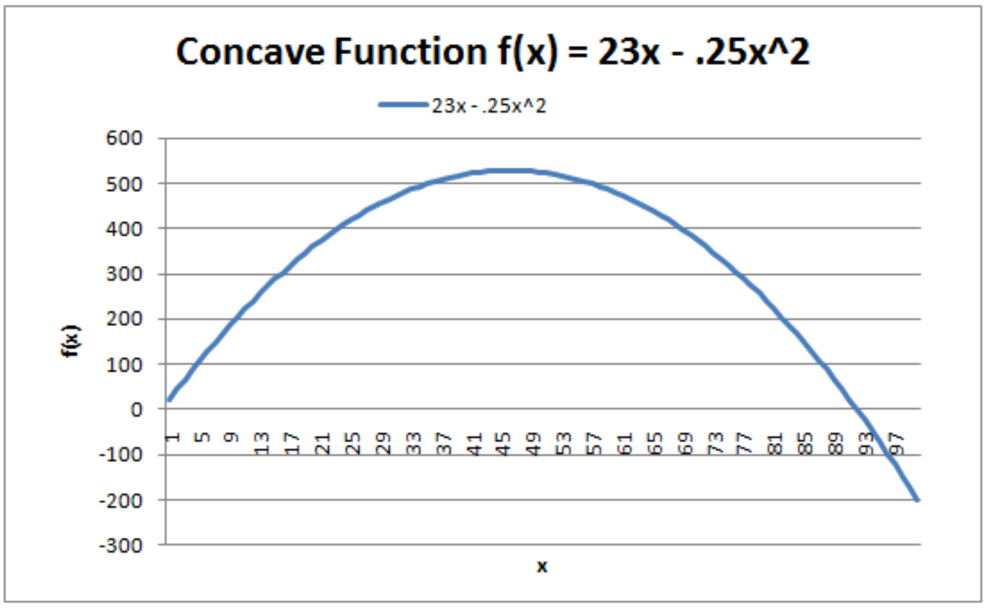


Fig. 3.1: A concave function example

investment. The second is the common pool resource which offers a return based on the amount of total investment by the group in proportion to each individuals investment. The pay off function for the pool, in this experiment, is determined by a quadratic production function which is concave in form, that is, the amount the pool pays out increases with investment to a point, after which the return decreases. An example of a concave function is shown in Figure 3.1.

A series of experiments are carried out varying the effects of investment in the pool. The initial baseline experiment explores the performance of the humans in a game with minimal constraints. As Ostrom et al. [1994, chap. 5] states "it allows for a close examination of individual and group behaviour under conditions designed to parallel those of noncooperative complete information game theory and it provides a benchmark for comparison to behaviour under alternative physical and institutional configurations".

The baseline experiment comprised the following: eight human participants made finitely repeated investment decisions regarding an amount of tokens with which they were endowed at the beginning of each round. The tokens are then invested in Market 1, offering a fixed return, or Market 2, the common pool offering a return based on the level of investment, or some combination of both. Participants know the number of other players, their own token endowment, their own past actions, the aggregate past actions of others, the payoff per unit for output produced in both markets, the allocation rule for sharing Market 2 output, and the finite nature of the game's repetitions. Participants also know the mapping from investment decisions into net payoffs.

The primary purpose of this base line experiment is to evaluate the difference in performance of a set of rational agents playing according to theoretical game theoretic predictions and the performance of humans playing the dilemma. In particular, the experiments were conducted to explore whether or not the group investment in the common pool resource will approximate the Nash equilibrium. In the experiments, the performance of the group is measured as a percentage of optimum rent (see Ostrom et al. [1994, chap. 5] for the full calculations of the Nash equilibrium and optimal levels of investment). Rent is defined as the return the group receives from Market 2, minus the opportunity costs of investing in Market 1, that is, the earnings from the pool market minus the fixed returns that could have been earned if the same investment was made in the fixed market. This is compared against the optimum rent which is calculated from the maximum return possible when all agents are investing equal amounts.

If the group plays to the Nash equilibrium prediction, the group earns rent at 39% of the optimum. The experiments are run with both 10 and 25 token endowments at the beginning of each round which does not affect the optimum and Nash equilibrium levels of investment. Table 3.1 shows the parameters for the two baseline experiments undertaken which are also used for the evolutionary experiments. The number of tokens predicted by Nash equilibrium is 8 for each agent, or 64 for the group, for both experiments. Neither the Nash equilibrium nor the optimum group investment depend on the number of tokens allotted so long as the allotment is sufficiently large. In this respect, the Nash equilibrium fails to capture the concept of over-appropriation caused by high endowments from which most real world CPRs suffer, ie., big mistakes are more likely and they are more harmful to the resource with high endowments.

This particular dilemma is not widely studied in evolutionary computational research. However, there is another problem that has been extensively researched. The El Farol bar problem, as discussed in Section 2.4.2, is an example. The difference between these games is that the CPR dilemma allows partial membership to the pool and the fixed market. The El Farol bar problem allows you to choose to go to the bar or not and a strict winning or losing payoff is assigned depending on what the rest of the participants choose. This difference warrants exploration especially in terms of studying groups of non-cooperative agents.

3.1.2 Results of Human Trials

The following is a summary of the results obtained by Ostrom et al. [1994] after conducting the baseline experiments with human subjects. Averaged across several experiments the results were as follows: the average net yield as a percentage of the maximum yield was rent at 37% in the 10-token game and at -3% in the 25-token game of the optimum. To put these figures into perspective, recall that investment at the Nash equilibrium level results in a return of rent that is 39% of the optimal. The main conclusion from this baseline experiment was that even as the players reach the equilibrium point, net yield decays toward zero and rebounds as the subjects alter their investment strategies. In low endowment settings, aggregate behaviour results tend

	Type of Endowment	
	Low (10 tokens)	High (25 tokens)
Number of Subjects	8	8
Individual token endowment	10	25
Production function (x_i is the investments by player i)	$23(\sum x_i) - .25(\sum x_i)^2$	$23(\sum x_i) - .25(\sum x_i)^2$
Market 2 return/unit of output	\$.01	\$.01
Market 1 return/unit of output	\$.05	\$.05
Earnings/subject at group maximum	\$.91	\$1.65
Earnings/subject at Nash equilibrium	\$.66	\$1.40
Earnings/subject at zero rent	\$.50	\$1.25

Tab. 3.1: Parameters used in the Human Experiments

toward Nash equilibrium. This however, appears to be as a function of the allotment of tokens. Careful analysis of the investments made, on a round-by-round basis, show that no subject consistently invests the Nash amount or within one token of the Nash amount of tokens. In the low endowment experiments, many subjects invest all of their tokens into the pool, a situation which does not occur in the high endowment experiments.

In the high endowment setting, aggregate behaviour in early rounds is far from Nash equilibrium but does approach it in later rounds. At the individual decision level, however, the behaviour is inconsistent with the Nash prediction. The group has the appearance of acting as if they are playing at predicted levels of investment as the total invested in the CPR, in the later rounds of the game, approaches 64. Each player is actually investing different amounts with some having high investments and some low investments.

In general, these experiments show that the game-theoretic predictions for rational agents do not hold for human participants, even in these simple scenarios where the subjects have the tools to make rational decisions. The rebound of aggregate investments from zero rent is as a result of reactionary decisions about investment levels. The human participants typically need several rounds in order for the group level of investment to even out.

Although the Nash predictions do not hold true for these experimental settings, investments by humans in similar experiments do follow the Nash predictions. Cox et al. [1988] discuss the case of single-unit, sealed bid auctions. However, as observed by Ostrom et al. [1994], "institutional changes, such as the change to multiple unit auctions, can lead to subject behaviour that is no longer consistent with the Nash model based on expected utility maximisation [Cox et al., 1984]".

3.2 A Genetic Programming Approach

An investigation into the performance of agents co-evolved using genetic programming techniques to play the baseline experiments (detailed in Section 3.1.1) is presented here. In this section, the application of Genetic Programming to this CPR dilemma is discussed, including algorithm structure, the function and terminal nodes available to the GP process and the representation of the agents during evaluation. The results of the agents' performance in this game are then analysed.

The goal of these experiments is to evaluate the behaviours which result from evolving a group of agents to play this particular CPR game. The results for human participation in this game have been established and a comparison is made between how the humans and the evolved behaviours play. This comparison is important if these games are to be the inspiration for computer game-AI.

As stated in Chapter 1, game theory dilemmas may provide an appropriate base upon which to create environments specifically tailored for computer game worlds. These dilemmas model complex interactions between individuals in groups using simple actions. Simple actions are necessary for groups of background characters who, in a commercial game, may not be allocated many computational resources. By taking aspects of the implicit relationships between self-interested individuals in groups from these dilemmas, it may be possible to create interesting groups for computer games. If the behaviours can be generated automatically, this may be a useful method for creating groups of background characters.

By first applying the GP algorithm to the traditional game theoretic dilemma, the GP algorithm can be validated by comparing generated behaviours against the expected rational behaviours. There are also benefits to applying evolutionary computation approaches to these dilemmas. The GP behaviours should provide an insight into the decision making process of an agent as the evolved tree exposes the game parameters that the agent is taking into account as it makes a decision.

The effects of evolving groups in this environment allow for the study of the effects of evolutionary parameters and game parameters on the behaviours of the agents. This can inform the application of genetic programming to more complex games which will be discussed in the following chapters.

3.2.1 Genetic Programming for the CPR Dilemma

Genetic Programming (GP) utilises a tree-based system to evolve decision strategies for agents. The potential reasoning ability of the created behaviour tree is limited by the function and terminal nodes available in the creation, selection, crossover and mutation phases of evolution. This section discusses the nodesets and also includes the constraints imposed by the Strongly-Typed Genetic Programming process [Montana, 1994, Haynes et al., 1995c] which serves to limit the combination of nodes in trees.

The GP process is used to create a tree for each agent representing its investment strategy for the CPR. The GP tree created makes a decision about the agent's token allotment and outputs an integer which represents the investment amount into the pool market. As all tokens must be invested at each round, the remainder are put into the fixed market. The GP creates trees from the set of functions and terminals contained in Table 3.2, which are divided into different *nodesets* for the STGP (for determining creation and crossover points for the trees). The functions are listed in Table 3.2 denoted by $f(n)$, where n is the number of arguments the function takes.

The description of the GP nodesets is as follows: The *Environment* nodeset contains functions and terminals that represent the agent's perception of the environment. Where the node says that it is believed, for this game that belief is with 100% certainty. The function BELIEVEDPROFITFROMM1 returns the amount the agent would receive based on the level of input as a given argument. BELIEVEDPROFITFROMM2 acts the same but also requires a figure for total group investment as an argument as well as the level of individual investment.

The *Decision* set has functions which take environmental or constant nodes and compare them (greater, less, equal). There are also *And* and *Or* functions which take other decisions. This allows for complex if then statements to be created.

The *Constant* nodeset contains terminals in the form of constants. It also contains some standard mathematical functions which take two arguments to be operated on (multiply, divide). It also contains the *If* function which takes 3 arguments, the first is a node from the *Decision* set, the second is the true branch and the third is the false branch. The *Decision* node is evaluated and returns either true or false which triggers the following of the true or false branches. The arguments to these functions can either be terminal or functions, except in the case where the growth factor of the tree has been

Environment	Constants	Decisions
BELIEVED ROUND	Multiply $f(2)$	Or $f(2)$
BELIEVED TOTAL GROUP TOKENS	Mod $f(2)$	Greater $f(2)$
BELIEVED AMOUNT OF AGENTS	Plus $f(2)$	Less $f(2)$
BELIEVED ROUND ENDOWMENT	Divide $f(2)$	Equal $f(2)$
PROFIT LAST ROUND	If $f(3)$	And $f(2)$
PROFIT M1 LAST ROUND	$\{0, 1, 2, \ldots\}$	
PROFIT M2 LAST ROUND		
TOTAL GROUP INVESTMENT M2 LAST ROUND		
INVESTED M2		
CUMULATIVE PROFIT		
AVERAGE INVESTED M2		
AVERAGE CUMULATIVE PROFIT		
AVERAGE TOTAL GROUP INVESTMENT M2		
AVERAGE PROFIT EACH ROUND		
AVERAGE PROFIT M1 EACH ROUND		
AVERAGE PROFIT M2 EACH ROUND		
BELIEVED PROFIT FROM M1 f(1)		
BELIEVED PROFIT FROM M2 f(2)		

Tab. 3.2: Node Sets for GP

exceeded. This parameter prevents from growing too deep thus maintaining a minimum level of efficiency.

The groupings of the nodesets are chosen to logically separate nodes based on their functionality. When performing evolutionary operations such as crossover and mutation, the groupings of the nodes allow for easy identification and alteration of nodes, without comprising the correctness of the tree. The groupings are also based on the return types from each node within the set. Nodes in the *Decision* set return Boolean values and allow conditional navigation of the tree. *Constants* and *Environment* return integers based on the results of some mathematical operations or information from the game state. The structure of the tree is therefore shaped by the necessity to not have nodes from the *Decision* set as the root of the tree.

3.2.2 Evolutionary Process

The following evolutionary parameters were arrived at through experimentation and comparison with typical values used in similar research like Doherty [2009], Doherty and O'Riordan [2006a,b], where another instance of the STGP algorithm is applied. The parameter values applied here are not exclusively the best for the problem set. The parameters do, however, have a large enough population to avoid converging on a suboptimal solution due to a low range of random solutions created in the first generation. The generational length, along with the values for crossover and mutation, are large enough to allow convergence to be disrupted if the solutions are suboptimal for an evolutionary run.

In each evolutionary run, two hundred and fifty trees are evolved over one hundred generations. The fittest individual at the final generation is chosen as a representative of that run. A complete list of the GP parameters used in the experiments is shown in Table 3.3. Random trees are initialised according to the constraints in the nodesets and each tree must begin with any of the functions from the *Constants* set. These trees are then evaluated using the fitness function outlined in Section 3.2.3. Once evaluated they undergo tournament selection in order to be chosen for the next generation. A tournament size of 5 is used in order to avoid a rapid convergence from selection pressure or a premature convergence towards a solution type due to an over-weighted fitness based selection.

The selected members of the population are then subjected to crossover and mutation with the probabilities 0.9 and 0.1 respectively. The crossover and mutation operators are outlined in Section 3.2.4. The creation of individuals at the first generation is initiated by first choosing a function from the *Constants* nodeset and by then choosing appropriate functions and terminals at random. The tree growth is limited by an initial maximum depth of 6 which ensures that very large trees are not created. The maximum depth

Parameter	Value
Population Size	250
Number of generations	100
Games per Evaluation	At least 20 $f(2)$
Creation type	Ramped half and half
Creation Probability	0.02
Crossover Probability	0.9
Swap mutation Probability	0.1
Maximum depth for creation	6
Maximum depth for crossover	17
Tournament Size	5

Tab. 3.3: Parameters for GP

for growth in crossover is limited to 17. This ensures that the trees in the population do not become very large and therefore very inefficient to process and evaluate. The fitness function reflects the desire to have more concise decision making processes by maintaining a small penalisation for tree length. It works on the following principle: if two trees have the same strategy the one who expressed it more succinctly will have a better fitness score. The fitness function tries to encapsulate the search for general solutions to the problem and is detailed in Section 3.2.3.

The main computational expense is the execution of the trees generated. The GP process is non-deterministic in terms of the structure an individual will have. The initial tree size is kept at a low number in order to curb the cost of evaluating deep but useless branches. This can become problematic as trees start to grow in the later generations. The run-time of the evolutionary process is tied to the average length and depth of the trees in the population.

3.2.3 Fitness Function

Each individual is evaluated by playing a series of games with random opponents from the population. For every individual, seven opponents are chosen at random and the game begins. The individual from the evolutionary population represents the investment strategy of the players in the game. For each round, the tree is evaluated returning the number of tokens which the agent will invest into the CPR and the remainder of the tokens are invested into the fixed market. The game is played for a fixed 20 rounds as opposed to a minimum of 20 as in the human experiments. The agents are not, however, aware of the finite nature of the game. The use of a non-deterministic game length for the human subjects was to avoid final round exploitation of the game. A fixed length is used for the evolutionary experiments for efficiency reasons. The game length is not direct knowledge and would have to be learned.

The fitness of the individual is the cumulative profit of the agent over the 20 investment rounds with a small penalty for length of the tree. Invalid trees are penalised by assigning a default poor fitness score which makes it unlikely that the tree will be chosen for subsequent generations through natural selection. An invalid tree is one that returns a value of tokens to invest in the CPR greater than the round endowment or less than zero. When a strategy like this occurs during evaluation, it is replaced by one investing a random percentage of the tokens endowments each round. The frequency of this occurring is in the order of less than 10% of the population in early generations dropping to near 0% by the final generation.

At the end of each game the fitness of the agent is saved for the agent and at the end of the evaluation process the fitness of each individual in the population is the average fitness obtained from these games. The sampling size is sufficient to represent the individual correctly in terms of the current population. A steady state population is not utilised and at every generation each individual must be re-evaluated against the new generation created. This ensures that no legacy strategies that were successful in early generations maintain their high fitness scores in subsequent generations against better individuals unless they are actually good strategies.

There is no single good solution to this problem and that at every generation a different strategy could be more successful. For example, in a population of cooperators, an exploitative strategy could be rewarded by over-investing in the common pool. However, if this strategy is adapted en masse, then strategies that invest most of their tokens in the fixed market will have a better cumulative profit, as returns from the pool will be destroyed by the over-investors.

3.2.4 Crossover and Mutation

Crossover is performed on the individual of the population after evaluation when the next generation of individuals is being created. An individual is chosen for crossover with a predefined probability. If crossover is to be performed, a second individual is chosen from the population for the operation to take place. The crossover algorithm then chooses a random point on the tree of the first individual and then randomly tries to find a node of the same nodeset type from the second tree. The node on the second tree is chosen at random until either a match is found or the process fails to find a match after twenty random node selections. If no match is found then crossover does not occur. If there is a match, the points are then swapped on both trees such that the complete subtree of the node chosen is also moved to the new tree.

Mutation is performed on an individual by selecting at random a point in its tree. A new node is then chosen from the same nodeset and inserted in its place. This new node is then grown and created much like in the creation process of the random population of the first generation. If the node is a function, appropriate nodes are chosen to branch out and grow using the same depth limitations for new trees. If the node is a terminal, it is simply exchanged with a random terminal from the same nodeset.

3.3 Experiment 1: Baseline CPR Dilemma

In this section, the experimental setup for the application of Genetic Programming to the baseline CPR dilemma, as outlined in Section 3.3, is discussed. The results of the application of the GP algorithm to the CPR dilemma are presented and compared with the results from human trials and other evolutionary studies. A comparison of the evolved GP trees' decision-making processes by playing the created solutions against the same fixed strategies is detailed.

3.3.1 Experimental Setup

A population of solutions is created and evolved using the rules from Section 3.2.2. The tree with the best fitness at generation 100 is selected to be a representative of that evolutionary run. As is stated in Section 3.2.3, it is not guaranteed that there is one best strategy due to the competitive co-evolutionary problem. The success of each strategy depends on the composition of the population, against which it is evaluated. Solutions which perform well in early generations are re-evaluated at every generation and must maintain performance against the newest population of solutions if they are to be preserved. The evolutionary run is believed to be sufficiently long enough to allow some convergence in the population, and this individual is most likely to be a good representative of the solutions. This conclusion was arrived at by examining evolutionary fitness progressions in a number of sample solutions. Although there can be no guarantee of a cycle-free population, the game-theoretic predictions for self-maximising agents suggests the population should converge towards the Nash equilibrium.

In order to mitigate the chance of evolving an individual that is just a lucky successful solution, eight separate populations of solutions are evolved to sample the search space. Each of the eight selected strategies are played in a game against each other. This game is used as the comparison with the human players' game from Section 3.1. As a benchmark, each evolved strategy is played against a group of pre-coded fixed strategies.

This process is repeated three times for each of the experimental setups. Each of the experiments for both 10 and 25 token endowments per agent per round are repeated three times to give a sample of solutions. The results of the aggregated play of the group, as well as individual investments in each round, are used as the basis of comparison between the evolved strategies and the human players.

3.3.2 Results

Using the tree with the best fitness at the end of 100 generations of the evolutionary run as a representative, eight of these candidates are used to play the game. It is found that, regardless of the token amount, the chosen agents play close or equal to the Nash equilibrium amount of tokens.

Recall from Table 3.1 that the Nash equilibrium individual investment is 8 tokens (64 tokens for the group). In the three 10-token scenarios, the agents play fixed strategies where, in every round, the same amount is invested into the pool. The average group token investment for the games is 65.68 with a standard deviation of 1.7 over the three sets of twenty rounds. This is the same performance as the human players had at an aggregate level. At an individual level, however, we see a different trend from that of the human players. The evolutionary pressure is on the individual to perform at the Nash equilibrium and this is the result we see from our evolved agents. The human players' performance is different, showing a changeable strategy that tends towards a group aggregate Nash in later rounds, but at the individual level, it is quite varied. The evolved agents individually play a typical non-varying strategy, of an average 8.21 tokens with a standard deviation of 0.21,

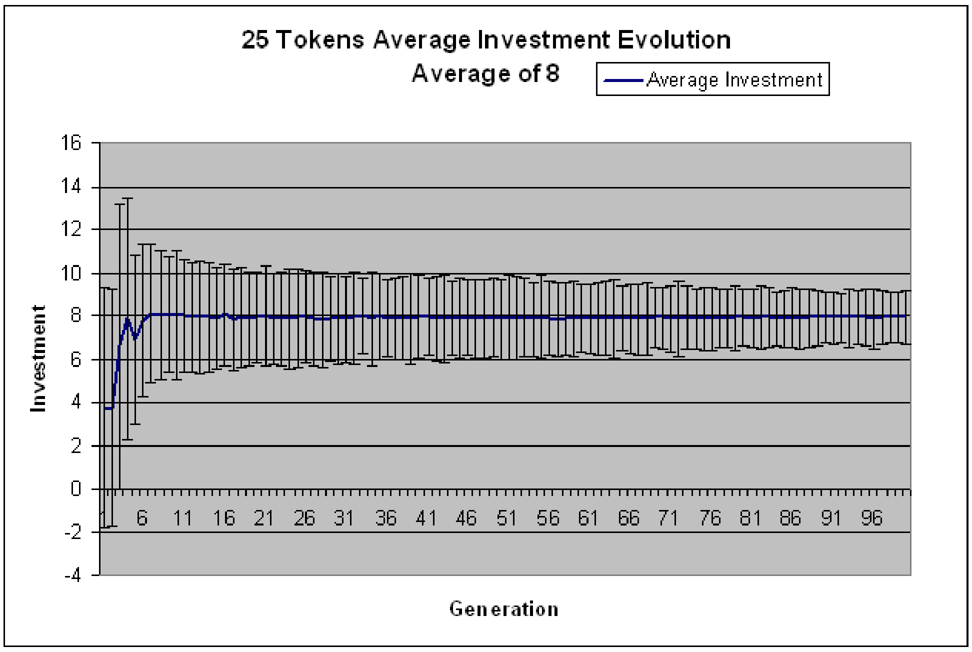


Fig. 3.2: Average Investment for 25 tokens

over the three sets of twenty rounds.

In the 25-token game, taking a candidate agent from each of the eight different evolutionary runs, we see a perfect Nash play in the three chosen instances. During the game between the evolved agents, each agent plays an unchanging strategy of 8 tokens per round with zero deviation over the three sets of twenty rounds.

As can be seen from Figure 3.2 the evolutionary process converges to strategies that contribute the Nash-predicted amount of tokens within the first ten generations. The standard deviation of the entire evolutionary population's investment decreases for the run. The results seen here are averaged over eight separate evolutions of the problem. This evolution pattern is typical for both token endowment levels.

The results here differ from both previous studies, using human subjects and a swarm modelling approach, as discussed in Section 3.1. Using genetic programming, the agents in these games converge to the Nash equilibrium point of investment, at both an individual and a group level. The population converging to the same investment level repeatedly indicates that this is the dominant strategy in this game. This is in contrast to the previous work in Swarm modelling, which found no dominant strategy while using the predefined strategies [Deadman, 1999]. Agents have been created that play as the game theoretic predictions suggest but not as human players do.

Throughout the evolutionary process, the behaviour of the GP generated trees moves towards the Nash equilibrium amount of investment that would be used by rational self-interested agents. The GP behaviours start out as random instantiations and therefore the average investment into the CPR will be approximately half of the endowment. Agents, who invest some but not all of their tokens, will do best in this scenario and as such will be propagated forward into subsequent generations. The core principal that the Nash equilibrium amount of tokens is based on is that it is the best amount for agents to invest if the whole group share equal profits and therefore have equal investment.

The evolutionary landscape has an attractor at the Nash equilibrium point for the agents in the group, especially when other agents are investing at this point. The agents converge to this behaviour, investing the Nash-predicted amount of tokens, and have no incentive to change their strategy. The human play on the other hand, is characterised by irrational play at an individual level. Even through the humans are able to work out what the most appropriate strategy is to play, they are often selfish in short term and over-invest. This leads to retaliation and eventually under-performance in the market-place.

3.3.3 Comparison of Evolved Behaviours

A set of fixed naive strategies is used to assess the similarity in performance between each representative strategy chosen from the evolutionary runs in Section 3.3.2. The cumulative profit of each strategy should show the degree of variance between them and also how well each strategy performs. Even though the strategies played against each other during evolution, the play that they encountered was that of rational self-interested agents. Fixed strategies allow the evolved solutions' performance to be assessed against a range of opponents. The fixed strategies are composed of some behaviours which run counter to the fitness goal of self-maximisation, which means that the evolved solutions will play against a range of behaviours, including against both under-investing and over-investing strategies.

To perform this evaluation of the co-evolved strategies from Section 3.3.2, each strategy is played against a set of naive strategies that invest 0%, 20%, 40%, 60%, 80% or 100% of their token endowments into the pool respectively. These opponents have a uniform and unchanging strategy throughout where by, in the first game, all 7 players playing with the evolved tree would all invest 0% of their tokens in the CPR. This is repeated for each value specified and the results are analysed on a game-by-game basis. This experiment is performed to investigate, not only the similarity in investment behaviours of the evolved strategies but also, if the evolved strategies contain the ability to exploit when there is low pool investment and to counteract over-investment. In other words, as a result of the convergence in the co-evolutionary population, do the agents blindly follow their investment strategies or are do the strategies take into account the rest of the players in the game?

Playing the agents evolved in the 25-token game against these naive strategies results in a mix of fixed and slightly exploitative strategies being played. An exploitative strategy is one that invests over the Nash equilibrium level, taking advantage in a lower than Nash aggregate level investment in the CPR.

Average	Std. Dev.	Max	Min
-542.598	10.0566	-496.385	-545.15

Tab. 3.4: 25 Token Evolved Strategies playing naive strategies. Figures indicate cumulative profits over a sample of 24

When an agent plays a fixed strategy, the resulting behaviour is to invest 8 tokens or 8 in the first round and then 9 tokens in subsequent rounds into the CPR. This indicates that the evolutionary strategies have learned to play at the Nash level but in an uncalculated way. The agents are not altering their strategies to maximise their returns by failing to respond to investments into the pool by the fixed strategies.

Some exploitative strategies gradually increased the amount of tokens from 8 to 10 when playing with the low investment naive strategies. Once the high-investment naive strategies were in play, no evolved agents reduced their usage of the pool from close to the Nash point and as such suffered net losses. The average cumulative profits for the 25 token allotment evolved strategies are summarised in the Table 3.4. The figures in the table represent 24 evolved solutions, each playing in games against the six different fixed strategies. The negative returns display the level of exploitation the evolved behaviours suffer as a result of not altering their strategies. The low standard deviation shows that all 24 solutions had approximately the same behaviours and results.

In the 10-token game, the agents are not as badly exploited as they are in the 25-token game. This is due to the fact that the naive agents are bound to the maximum of 10 tokens so the negative returns in the 25-token game are not seen here. Once again, the agents display an inability to change strategy as the pool is over exploited with agents playing almost uniform fixed strategies that invest either 8 or 9 tokens. The twenty-four evolved samples (eight evolved strategies over three iterations) have very similar behaviours as can be seen from the low standard deviation in cumulative profits in the Table

Average	Std. Dev.	Max	Min
682.4877	2.8101	687.16	678.1

Tab. 3.5: 10 Token Evolved Strategies playing naive strategies. Figures indicate cumulative profits over a sample of 24

3.5.

To put these figures into perspective, the total that a group would earn, if a Nash strategy was implemented for each game played (6 games at 20 rounds of Nash Investment) for the 10-token game is 633.6. The 10-token strategies exceed this figure against the fixed strategies due to the fact that the group earns more with the lower fixed investment strategies than it loses against the exploitative strategies. The low endowment prevents the fixed strategies from creating negative returns for the evolved strategies. The majority of the fixed strategies invest an amount lower than the Nash equilibrium amount (all except the 100%) and as such the evolved strategies benefit from a return is better than would be expected from Nash equilibrium play. The CPR pays out a higher proportion of a better return to the evolved strategies when there is under-investment from the others in the game.

For the 25-token game, the group would have earned 1353.6 from a Nash equilibrium level of investment instead of the average negative return of -542.6. With the 25 endowment in place, the evolved solutions fail to compensate for the high level of exploitation and as a result the returns from the CPR are destroyed. The fixed strategies invest more than a Nash level of tokens in four out of the six sample strategies (40%, 60%, 80% and 100%) causing the unresponsive evolved strategies to incur negative returns.

The failure of the evolved strategies to respond to the behaviours of the fixed strategies causes them to under-perform. The reason for this lies with the evolutionary process and the evaluation of solutions during evolution. Towards the end of the evolutionary process there is a good chance that the representative strategies do not meet any irrational behaviours. As the co-evolved population of agents converge to play the Nash amount of tokens, the evolutionary population will contain an increasing proportion of opponents playing similar strategies. This in turn, leads to agents not evolving very dynamic strategies as the strategies of the population are increasingly predicable. With the pressure of penalisation for length of solution during evolution, the ability of a strategy to adapt becomes lost from their decision tree in favour of simple Nash playing strategies.

This results in very fixed strategies, and as seen above, typically the evolved agents do not normally take into account the actions of others. The low deviation between the agents' performances across the 24 samples for each endowment suggest that the behaviours seen are typical of the strategies that can be expected when evolving in this environment under the evolutionary parameters specified.

The findings in this experimental setup, offer a confirmation of the game theoretic predictions for rational agents for this baseline dilemma. The application of GP to the baseline CPR dilemma has provided several insights. Firstly, the GP process is suitable for creating solutions to this dilemma. The solutions created act according to their fitness functions and they perform as rational self-maximising individuals. Secondly, the co-evolutionary process, in this instance, is not suitable for providing solutions capable of maximising returns against a range of solutions outside of the evolutionary process. Finally, the solutions generated are not similar to the human behaviour described in Section 3.1.1.

3.4 Game Variations, Towards Human-like Play

This section introduces a variation to the evolutionary setups discussed in Section 3.3.1. This alteration is included to try and capture some additional properties of the strategies that the process of co-evolution in the original game failed to create. This includes, the ability of the GP process to retain its decision-making ability in the face of irrational play by other members of the group.

In Section 3.3.2, the evolved GP behaviours converged to the Nash equilibrium point in the CPR dilemma as is expected by rational agents. The co-evolutionary process ensured that most of the population tended toward this point and as a result the agents do not get exploited. Even if completely selfish agents are entered into the game, through the process of creation, crossover or mutation, there are enough agents playing at the Nash point to counteract any poor fitness scores from these games. The agents do best at this investment level and as such there is no incentive to change from it. This results in the loss of reactive behaviours and provides no insight into the decision making processes of the agents as they generally play fixed strategies.

Changes are introduced to the evolutionary set up to attempt to create and preserve reactionary quality in strategies that are being evolved. This is seen as a feature that all human players would have and as such an attempt should be made to emulate. When a strategy is behaving irrationally in the group, the members should be able to alter their strategies to counteract this scenario.

By varying the members of the group, to include purposefully disruptive strategies, the influence of the group composition on the agents can be analysed. The parameters of the game are varied so as to test whether the evolutionary approach allows the creation of reactive strategies. Further comparisons are then made between the evolved behaviours of the agents and those of humans in the baseline game. An analysis is performed as to how the behaviours react to changes in their environment.

3.4.1 Evolving Against Naive Strategies

To explore the possibility of creating strategies that maintain their ability to be reactive, naive strategies are used in the evaluation of our agent trees. For these experiments, evolutionary parameters from the baseline experiments are reused and are detailed in Table 3.3. The main difference in this case is that during the evaluation each agent plays six different games instead of using a competitive co-evolutionary process. Each of the six games consists of opponents all of whom use one of the naive strategies as described above. For instance, in the first game, seven of the agents in the group invest 0% of their token investment into the CPR resource. In the next of the six games, seven of the agents in the group invest 20% of their token investment and so on. The payoffs for this game are also the same as the baseline experiments and the fitness of the agents is the cumulative profit from the six games.

Eight evolutionary runs are conducted with the best individual at generation 100 chosen as a representative of the run. This is repeated for both the 10-token and the 25-token endowments. In order to assess the solutions, each played in six games with groups composed of the naive strategies from the evolutionary evaluations. The profit for each agent every round is saved and used to compare the approaches of the eight evolved solutions. Following this, the eight evolved solutions play games against each other and the results are recorded. This game will allow for a direct comparison of behaviours for created solutions with those from the baseline game.

The results for trials are as follows: for the 10-token game, the eight evolved behaviours playing each other yielded similar performance to those of the co-evolved baseline game. As the behaviours in 10-token game do not get receive a poor return when all tokens are invested by the 8 players, the strategies learned are consistently close to, or at, a Nash level of investment. The evolved behaviours for the 25-token game display a different set of results than those in the 10-token game.

For the 25-token games, the evolved strategies vary. Some play a uniform investment strategy while others play adaptive strategies. The play of the evolved strategies is displayed against each of the naive strategies in Figure 3.3. From this figure, the different average profit per round can be seen across the evolved strategies. As the rest of the group is playing fixed strategies, each of the evolved strategies' performance can be compared. There are varying approaches from each of the representative individuals as opposed to the general trend of fixed Nash level investments seen from the co-evolved strategies.

The evolved solutions were played in a game against each other which served to further emphasise the differences between the strategies. Figure 3.4 shows the token investment for each of the eight evolved strategies. The average individual investment per round over the twenty rounds is 10.54 tokens with an standard deviation of 9.73. The best performing strategy is one that does not invest in the pool at all while the two next best are adaptive strategies. The worst performing strategies are ones that constantly invest the total amount of tokens. The profit for each of the evolved behaviours are shown in Figure 3.5. The average efficiency of the investments as a percentage of the maximum return is 53.5%.

The amount of variance between the evolved strategies for the 25-token game is a different result than that of the co-evolved results in the baseline game. When the new evolved strategies play each other, the performance is similar to human play. Both these strategies and the human play resulted in pool

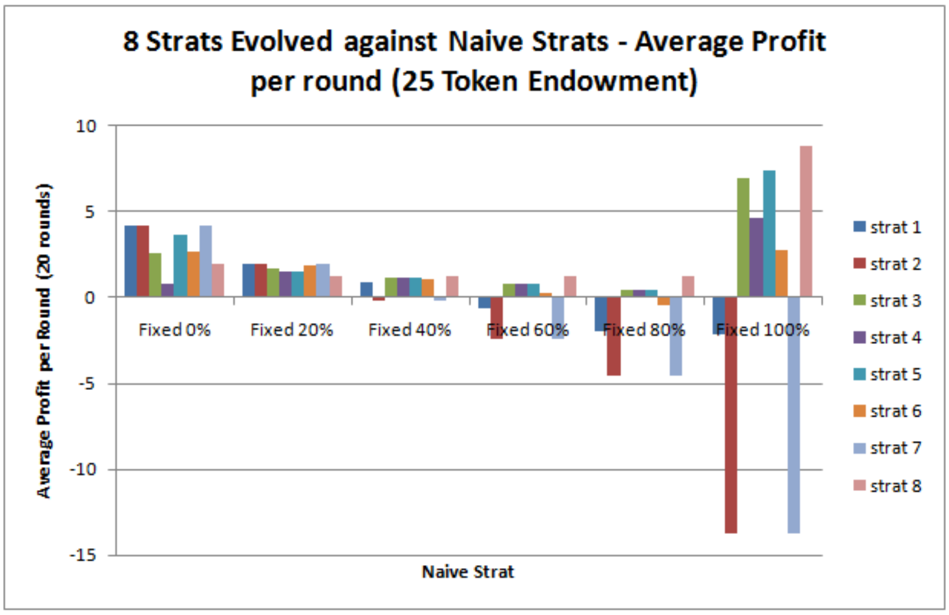


Fig. 3.3: Evolved Strats playing against Naive Strategies

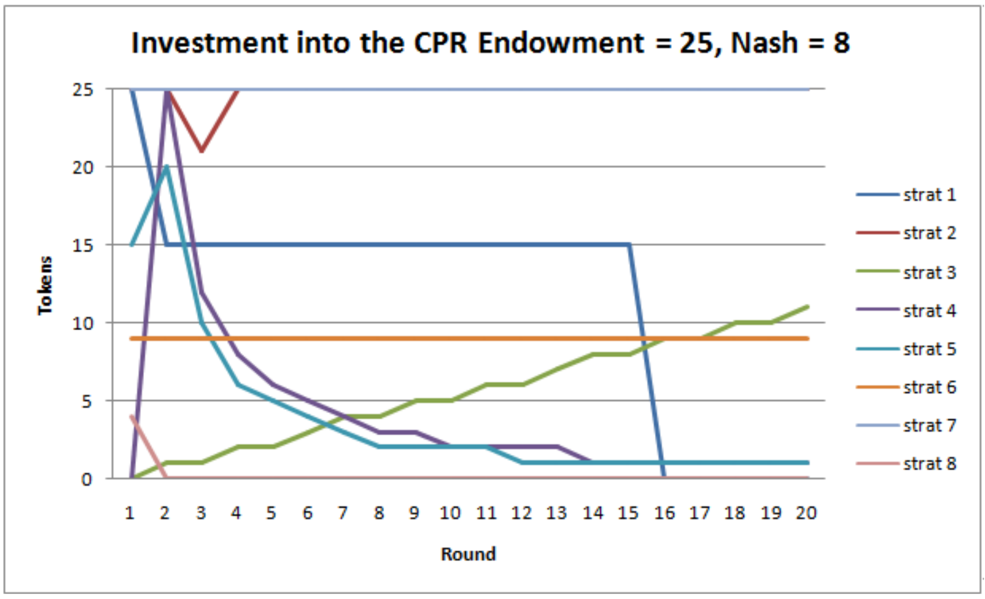


Fig. 3.4: Evolved Strategies playing each other (Investment Patterns)

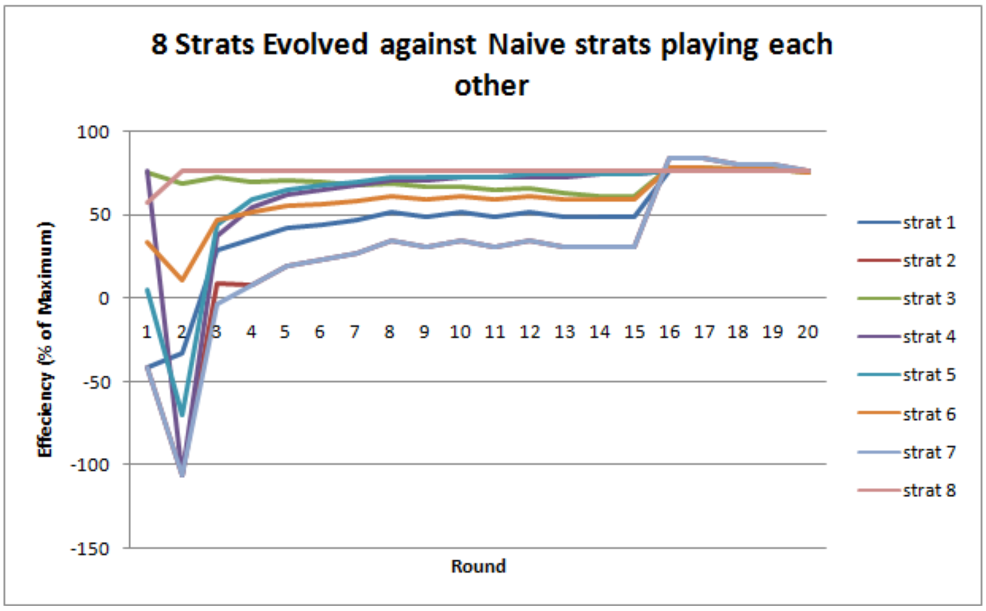


Fig. 3.5: Evolved Strats playing each other

utilisation which refined itself close to Nash at the aggregate level but, from an individual level, the investments are wildly varying. The efficiency of earnings for the evolved strategies of 53.5% is much higher than that earned by human players. The efficiency of a group playing at the Nash point is 85%.

Human-like play has been created, that is, play by the agents that on a group level comes close to the group Nash equilibrium but that is varied at the individual level. The process of evolving against a set of fixed or irrational players has also maintained the GP created strategies ability to adapt in these environments.

An example of previous approaches to this problem using evolutionary approaches was the application of Swarm modelling to simulate the common pool resource dilemma (for an explanation of Swarm modelling see [Minar et al., 1996]). Deadman [1999] report almost identical performance from their agents, in terms of efficiency, when compared to the human players. In this model, 16 strategies are predefined some of which are derived from play in the human trials and others which attempt to maximise return round-by-round by increasing or decreasing investment. Agents are endowed with a subset or all of these strategies at the beginning of the game and the strategies remain fixed. A model of adaption is provided for the agents such that they may choose to use one of the strategies with which it has been endowed. These predefined strategies could be the reason that the performance was very similar to the human play. They also show that no strategy becomes dominant and even though the agents may have access to all of the strategies, their individual performances vary.

3.5 The Effect of Environmental Pressures

In previous sections, the behaviours of individuals within the groups have been measured. A demonstration of the effects of irrational play, that is, play that runs counter to the fitness goal of self-maximising profits, was provided for the cases of irrationality introduced during and after the evolutionary process. If the behaviours have not encountered such play during evolution, they are unable to respond by altering their strategy and extract a good return from the markets. When the irrational behaviours are introduced into the evolutionary process, several types of behaviour emerge which shares similarities with human-like play.

In this section, the effect of environmental pressures on both group and individual behaviour is explored. The game rules are altered to account for external forces on the agents and the results of their actions within the CPR dilemma. Firstly, an investigation into the effects of uncertainty or unpredictability of the actions of group members on the evolution of the behaviours is conducted. Randomness is introduced into the play of individuals in the group at different levels and the changes in behaviour of the evolved agents are studied. This will allow comparison with work from a similar field to see if these behaviours conform to their findings (see Section 3.5.1 for an introduction to this work).

Secondly, experiments are conducted to establish how the introduction of a finite resource, with probabilistic destruction of the pool resource, affects the behaviours of agents. Two experiments are undertaken: one with a safe zone which permits some level of safe investment into the pool and the other without this safe zone. The probability of destruction of the pool is increased linearly by the amount of investment in the pool (outside any safe zone).

These experiments will allow for further comparison between the behaviours of the evolved strategies and humans. By introducing variations into the environment, the suitability of the GP algorithm for creating behaviours in a range of settings is also tested.

3.5.1 The Effect of Randomness or Uncertainty

As is pointed out by Jager et al. [2002], the introduction of uncertainty into a CPR dilemma leads to over-harvesting which subsequently leads to under performance for the group or, where applicable, destruction of the resource. A series of evolutionary runs are conducted which introduce an amount of randomness (in order to simulate uncertainty as to the behaviour of the group) to the group of individuals playing the game. As before, the individual trees are co-evolved, with each one representing the strategy for investment. The decision tree created dictates the amount of tokens to be invested into the pool with the remainder (from the allotment at the beginning of the round) being invested into the fixed return market. For evaluation, each member of the population plays against a random collection of other members 20 times. The return from each game is recorded for each group member. Any individual can be chosen multiple times to be part of the evaluation of other solutions and so, along with the twenty games for their own evaluation, can end up playing more games than this. The fitness of the individual for that generation is averaged over these games.

In order to introduce randomness into the game, a number of players playing a purely random token investment into the pool are involved in the evaluation. The number of players ranges from one to seven, to represent each of the other members of the group. When there are multiple random players, one random investment is chosen and all other random players play this amount. A different investment is chosen at every round. For each number of random players the experiment is repeated 8 times with the best individual from generation 100 taken as a representative of that evolutionary run.

The results of this on the evolved behaviours are displayed in Figure 3.6. Each line on the graph represents the average investment made in the CPR by the agents. Each point on the graph is the average of 8 runs for the same problem. For example, the $1RandomMember$ line represents the average investment at every generation for all the members of the population, averaged over 8 separate evolutionary runs, in an environment where one group member is playing randomly.

From the graph, it is evident that the populations are converging to specific investment points, which changes depending on the level of random investment by the members of the group. With only one random member, the group achieves close to the original Nash equilibrium investment point. When six or seven of the members of the group are playing randomly, the investment by evolved strategies drops to zero.

In Figure 3.7, the predicted average investment by the random members of the group is plotted against the average investment by the evolved members and the group Nash investment point. In this scenario, the evolved players reduce their investment such that each of the evolved strategies obtains an

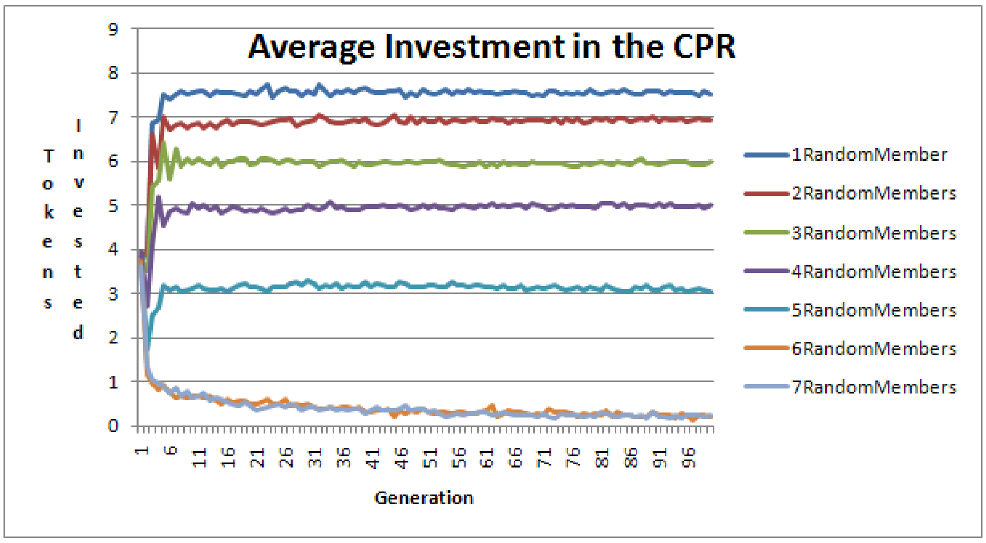


Fig. 3.6: Average Investment in the CPR with varying degrees of randomness

equal share of return on investment. As uncertainty increases, there is over-exploitation, as the amount that the evolved strategies invest is, on average, above the remaining group Nash investment level.

Jager et al. [2002] use agents which they say are equipped with human-like cognitive processes in their simulations. These agents can use deliberation, social comparison, imitation and reputation of previous behaviour when making investment decisions. They show that increased uncertainty may stimulate an imitation effect that promotes over-harvesting. The increased uncertainty also lead to an increased optimism about future returns and lessened the ability of agents to adapt during resource depletion.

In this experiment, the increase in uncertainty leads to a decrease in investment by the evolved agents in the CPR in proportion to the increase in average random investment. This behaviour is the expected one by rational agents as opposed to one with human cognition.

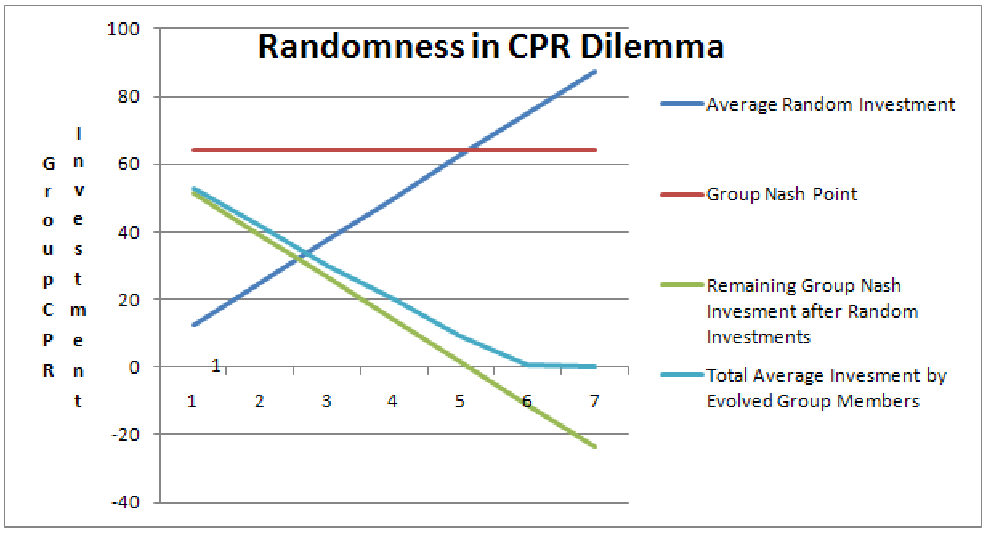


Fig. 3.7: Average Investment by Evolved Members of the Group vs Random Members

3.5.2 Probabilistic Destruction of the Commons

It has been shown that co-evolution in the baseline game will lead the agents to conform to the Nash equilibrium predictions for individual behaviour. The introduction of a set of fixed strategy opponents into the evolutionary process yield a selection of varying suboptimal human-like behaviours. Then, by introducing varying degrees of randomness into the game, agents opted out of investing in the CPR in proportion to the uncertainty of investing in it. In the experiments presented in this section, the uncertainty relates to the resource about which the agents are making decisions.

The notion of a limited resource is introduced into the CPR dilemma. Up until this point, the games which we have been discussing simply continue for a set number of rounds regardless of the behaviours of the agents within the game. This extension examines what happens when there is the potential to destroy the resource which is being shared. Both the other agents and the

	Experiment :	
	No Safe Zone	Safe Zone
Number of Subjects	8	8
Individual token endowment	25	25
Production function (x_i is the investments by player i)	$23(\sum x_i) - .25(\sum x_i)^2$	$23(\sum x_i) - .25(\sum x_i)^2$
Market 2 return/unit of output	\$.01	\$.01
Market 1 return/unit of output	\$.05	\$.05
Earnings/subject at group maximum	\$1.65	\$1.65
Earnings/subject at Nash equilibrium	\$1.40	\$1.40
Earnings/subject at zero rent	\$1.25	\$1.25
Safe Zone	0	40

Tab. 3.6: Parameters used in the Probabilistic Destruction Experiments

environment in which the game is being played influences the behaviours of the agents.

Probabilistic destruction in CPR dilemmas is useful for studying real-world CPR problems. Often, in the real-world, CPRs are fragile, and human exploitation can lead to destruction. Examples of such scenarios are fishery grounds, forest resources or groundwater basins. Ostrom et al. [1994] provide real-world examples from social science case studies and also define an abstraction to be played by humans. This abstraction is utilised to study the effects of environmental pressures on the evolved agents and once again, there is the opportunity to compare the GP evolved behaviours with those of the humans. The experimental setup is similar to the baseline problem previously studied. For these experiments, the 25-token endowment experiment set is concentrated on solely, as this is sufficient to illustrate the effect of a destructible resource. The parameters are outlined in Table 3.6.

Probabilistic destruction exists in two forms: the first with a safe-zone of investment and the second with the safe zone removed. A safe zone is used to model resources where there is an excepted level of extraction that can occur while maintaining the performance of the resource, with an example of such a safe zone being water levels based on rainfall in shared irrigation systems. A resource without a safe zone is a particularly volatile one, with any extraction increasing the chance that the resource will be destroyed.

In this case, destruction occurs when (outside the safe-zone) each token invested in the pool increases the chances that the pool will cease to exist (and the game terminates) by 0.5%. Therefore if, in single round, all agents invested their allotment of tokens into the CPR, this would result in a 100% probability of destruction of the pool resource (8 agents with a 25 token allotment). Once either twenty rounds have been played or the pool has been destroyed, the game ends. Eight separate evolutionary runs are completed and the evolutionary trajectories are plotted as averages of these. For comparison, the best agent at generation 100 of each run is chosen as a candidate solution.

Investment behaviours are evolved in the same fashion as for the baseline experiment as described in Section 3.3. Co-evolution is utilised to create behaviours using the same GP nodesets from the baseline experiments. In the first experiment the safe-zone is set at 40 for the group which is 20% of the group's total endowment. After this point, the probability of destruction increases linearly with the group's investment.

The results of the evolutionary runs for the destructible CPR with a safe zone are detailed first. The evolved results displayed in Figure 3.8, show the population converging to a point where each member invests exactly five tokens each into the pool, for a group total of 40, with the remainder being invested in the fixed market. The GP process converges to the rational strategy of maximising the return from the CPR whilst ensuring that the resource is not destroyed. An analysis of the candidates from each evolutionary run reveals that, at an individual level, this rational behaviour is indeed the case.

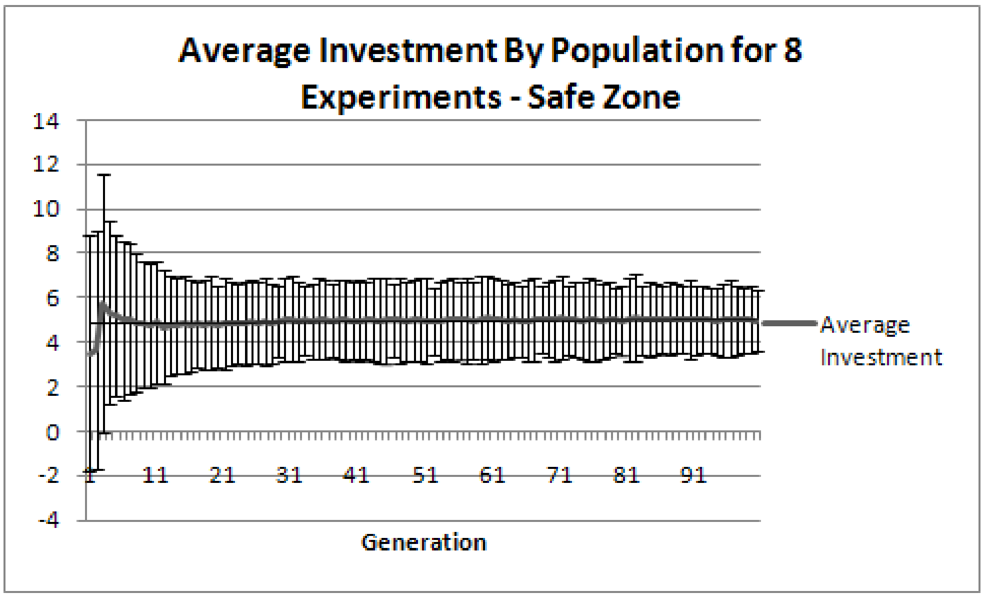


Fig. 3.8: Probabilistic Destruction with a 20% Safe-zone

This behaviour differs from that of the human-play. Even though the same information was available to the humans playing the game, they were unable to avoid destroying the resource with a safe zone.

In the second experiment, the safe-zone is removed leaving only an increasing probability of destruction of the resource. In this case, the evolved solutions are unable to avoid destroying the resource. By repeating the game between the candidate solutions from the eight evolutionary runs 100 times, it is revealed that the agents preserve the resource for an average of 4.04 rounds. One interesting feature of their behaviour, is that the agents play close to the Nash predicted level of investment (average of 7 tokens for the 8 individuals). This indicates that, while the resource is being prematurely destroyed, whatever income is garnered from it, is distributed almost equally among the participating agents. The human trials also destroy the pool quickly although their investment pattern differs from the evolved behaviours by varying at an individual level.

In these experiments, the evolutionary processes produce rational strategies that are predicted by the Nash equilibrium when there is a safe-zone. This play is unlike the humans, who are unable to preserve the pool with the safe-zone in place. When the safe-zone is removed, the evolved strategies do not preserve the resource as they invest into the pool. This play is similar to the humans at a group level. At an individual level however, the evolved investment strategies still preserve some notion of equality, by converging close to the Nash predicted investment point.

3.5.3 Discussion

In this section, the effects of environmental pressures are studied and the behaviours of the agents analysed. Through evolutionary computation methods, the population of co-evolved agents tends to converge towards the Nash predicted equilibrium points. The introduction of individual irrationality within the group evolution changes the behaviour of the agents at an individual level such that there is no longer convergence to a single point in the behaviour space.

The introduction of the probability of destruction of the pool resource has two effects on the group, depending on whether or not there is a safe-zone of investment. When a safe-zone of investment is established, the agents evolve to play within bounds of the safe-zone, preserving the pool and maximising the return for the game. The agents do this using an equilibrium strategy, with each agent investing an equal amount into the pool. This behaviour is unlike the human behaviour discussed in previous trials, where the participants were unable to preserve the pool. With the removal of the safe-zone, the agents are no longer able to maintain the pool and games typically last a short amount of time. At a group level, this evolved behaviour is the same as the human behaviour. However, at an individual level, the population of agents still tends to converge to an investment point close to the Nash prediction.

So far in this chapter, a variety of results have been presented. The rationality of the evolved agents is displayed when agents are co-evolved in the baseline game. Similarity to human play can emerge from the evolutionary process in two different ways: firstly, at an individual level, if irrational agents are introduced to the evolutionary process; secondly, at a group level, when enough disruption to the environment occurs because of the behaviours of individuals.

3.6 Co-evolution and a Group based fitness

In Section 3.3, a co-evolutionary strategy was used to create behaviours for individuals in groups. Using this method, it was observed that rational agents (interested in maximising self-profit) would act according to the Nash equilibrium predictions. This results in agents who all profit equally but the group performs below its potential.

Most applications of evolutionary computation involve the search for the optimal solution for a given problem. Complications arise when the notion of groups is introduced. Individuals taking part in game theoretic dilemmas are easy to rank, in so far as, their performance is objectively assessed and can be compared to all other individuals. When creating group based behaviours for evolutionary computation, each behaviour must be assigned a fitness score. When creating individuals that form part of a group, each contribution must be scored. This means that the contribution of each individual to the performance of the group must be assessed but this can be a difficult task.

An individual could perform an action which is sub-optimal for themselves but this could allow the group to achieve a higher fitness overall. The performance of each individual is linked with how each other member of the group plays. The evolutionary pressure for an individual is towards the Nash equilibrium which is sub-optimal for the group. If an agent tries to alter its strategy to help the group achieve a better score, the individual will receive a lower fitness and, as a result, may be lost in the evolutionary process.

In the CPR dilemma posed in Section 3.1.1, the strategy to achieve the optimal for the group is different to the best strategy for an individual. For the CPR dilemma, self interested agents, when co-evolved, will never achieve the group optimum. The agents need to have a sense of group performance in order to achieve greater than Nash equilibrium levels of return. In this section, the fitness function for the individuals is changed such that they are no longer solely interested in maximising self profit. A range of fitness functions are outlined below, in which, the performance of the group is taken into account.

The motivation for this work lies in the assessment of the evolutionary process' ability to create solutions for groups. Throughout this chapter, competitive co-evolution has been used to create individuals that take part in group based activities with the self-profit maximising. In this section, the competitive co-evolutionary algorithm is used to create groups with better performance while still creating individuals in isolation of the group. The CPR dilemma provides a suitable domain for assessment of the GP algorithm in this respect, as it allows the easy assessment of an individual's and a group's performance. This provides the groundwork for the application of evolutionary computation to more complex environments in later chapters.

3.6.1 Fitness Function with Group Performance

The GP fitness function is the only part of the algorithm that is changed from the baseline experiments in Section 3.3. The original function was specified as follows: The evolutionary individual is chosen to play in many games with groups composed of random individuals from the remainder of the population. The fitness of the individual is the cumulative profited averaged over the number of games in which the solution played (with a small penalisation based on tree size).

This fitness function is altered with the following addition: the individual profit in any game is scored as a proportion of the maximum profit possible for an individual. The group's performance for each game is also calculated as a proportion of the maximum potential group performance. These two proportions are then combined to provide the score for a game. The final fitness of an individual is the average of the scores for all the games played. The fitness function is detailed formally in Equation 3.1

$$(\frac{IProf}{Max.IProf} * IProp) + (\frac{GProf}{Max.GProf} * GProp) \tag{3.1}$$

where:

$IProf$:	is the individual's profit for the round.
$Max.IProf$:	is the maximum potential profit for an individual.
$IProp$:	is the weight given to the Individual's profit.
$GProf$:	is the group's profit for the round.
$Max.GProf$:	is the maximum potential profit for a group.
$GProp$:	is the weight given to the group's profit.
	and $IProp$ = 100 - $GProp$

The proportions of the Group's performance that an individual uses in their fitness are 25%, 50%, 75% and 100% Group. These figures are selected in order to provide a spectrum of behaviours across the range of the individual-group proportions. The 0% Group proportion is equivalent to the experiments in Section 3.3 where the individuals are solely self-interested. For the 25% Group proportion, an individual's fitness is composed of the individual's performance and the Group's performance. The individual's profit is calculated as a proportion of the maximum possible individual profit multiplied by 75. The Group profit is calculated as the group profit earned as a proportion of the maximum possible group profit multiplied by 25. The individual proportion and the group portion are summed together to provide the fitness score where the maximum possible fitness is 100. The results of the evolutions are presented and discussed below.

3.6.2 Results

Presented in Figure 3.9, is the average investment into the common pool resource by individuals in evolutionary populations for the varying fitness functions. The data points are the average of each investment into the CPR made in that generation, averaged over eight evolutionary runs. Figure 3.9 shows the changes in investment patterns for individuals with varying levels of consideration given to the group's performance.

Taking the best individual from the final generation from each of the 8 evolutionary runs and playing them in a group together provides an indication of the behaviours produced by each fitness function. Table 3.7 details the average investment for each of the groups for comparison.

By taking into consideration the performance of the group into the fitness of the individual, the average performance of the group and individuals increases as expected. The Nash equilibrium strategy earns nearly 84% of the group's maximum earnings. By giving the group's performance the same weight as the individual's performance, the group's performance increases

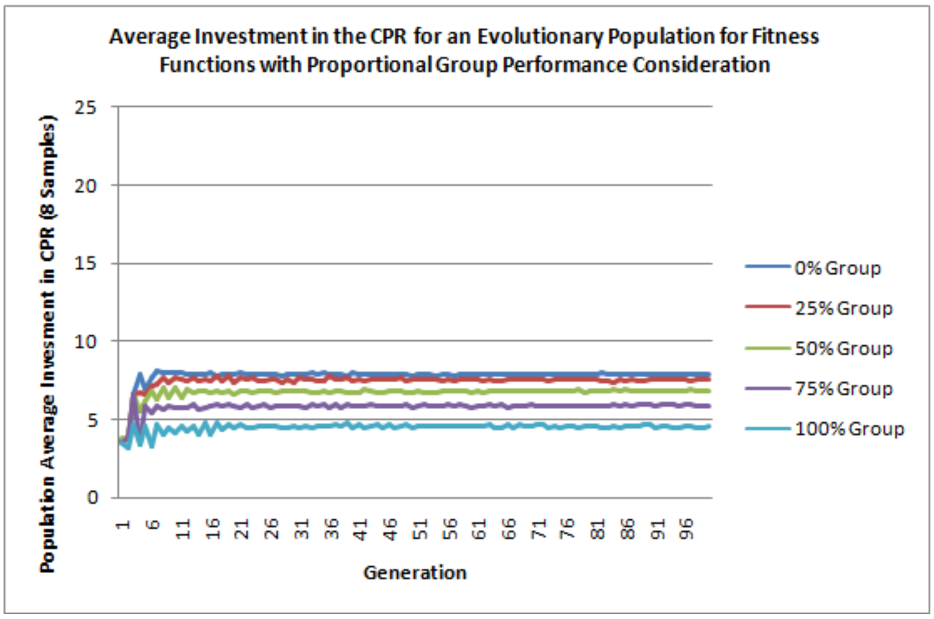


Fig. 3.9: Average Investment in the CPR for Varying Fitness Functions

	Max	100% G	75% G	50% G	25% G	0% G
Group Ret	13.34	13.23	12.88	12.24	11.68	11.2
Avg Indiv Ret	1.655	1.65	1.61	1.53	1.46	1.4
% of Max	100%	99.18%	96.55%	91.75%	87.54%	83.95%
Avg Investment	4	4.73	6	6.99	7.63	8

Tab. 3.7: Average Investments for Individuals with varying proportions of Group (G) Consideration

earning nearly 97% of the maximum. When only taking the group's performance into consideration the group achieves over 99% of the maximum.

By changing the fitness function for individuals it is possible to achieve better overall group performance while still using the competitive co-evolutionary paradigm. From the example solutions chosen, it results in a collection of individuals which evolve to invest slightly varying amounts with an average that is required to achieve the higher group performance.

The group's performance and individuals' performances are bound by having individuals with the same preference for the overall outcome. When these 8 solutions are tested against naive strategies (from Section 3.3.3), they fail to respond to the irrational play, i.e, play that is contrary to the fitness pressure of the evolved individuals. Indeed, any individual with a different fitness preference would change the group's performance and a self interested agent could take advantage by over-investing in the CPR.

3.7 Chapter Summary

This chapter discussed a traditional CPR dilemma in which a group of agents make repeated investment decisions resulting in a payoff for each agent based on how all other agents played. The baseline experiment, which was defined by Ostrom et al. [1994], is used to compare the behaviours of humans to the behaviours of genetic programming evolved agents. Human play was shown to be erratic on an individual level but conforms to the game theoretic predictions of a Nash equilibrium at an aggregate level. The evolved behaviours were shown to conform to Nash predictions at both individual and group levels. The resulting behaviours had, however, lost their ability to react against strategies which may exploit their behaviour or which may be naively investing in the common pool market.

A series of experiments were conducted whereby the GP strategies are evolved to play against a set of naive strategies. The resulting behaviours were diverse and did not converge to a single strategy while maintaining their ability to react to poor play by other agents. When a set of these evolved behaviours are played against each other the results were very similar to the play utilised by the human players.

The effects of changes in the environment on the evolved behaviours were then explored. It was shown that the effects of randomness or uncertainty in other agents' investing patterns caused the agents to invest less in the CPR. This rational response differs from results for a similar game in a previous study, which pointed out that the introduction of uncertainty into a CPR dilemma leads to over harvesting which subsequently leads to under performance for the group. This finding is a demonstration of the rational versus irrational response to the situation.

Probabilistic destruction was introduced to the game to understand the effects of environmental pressures on the evolved behaviours of the agents. When there was a safe zone of investment, the agents were able to evolve to investment patterns that ensured that group level investment stayed within the safe zone. This in contrary to the findings of human players who disregarded the safe zone for short term gains. When no safe zone exists and any investment in the CPR increases the chance of destruction, the GP is unable to find a sustainable solution. This is the same behaviour that the human players exhibit.

The search for the group optimal performance using co-evolution was discussed. When agents took the group performance into account, the group's performance was increased. The higher the consideration given to the group performance the greater the group performance. The individuals created were however, unable to generalise their strategies and would under perform against agents not evolved with the same fitness function.

Throughout the chapter, it has been shown that GP can be an appropriate mechanism for evolving group behaviours in this domain. The resulting behaviours will act rationally under certain circumstances, while under others, that is ones which contain either random or poor play by other agents, the behaviours will be like those of humans. In all cases, the behaviours are naturally the result of both, fitness pressures and environmental pressures.

This chapter examines the hypothesis that evolutionary computational approaches in game theory dilemmas yield human-like performance under certain circumstances. Creating behaviours that acted the same as the game-theoretic predictions proved relatively straightforward for the GP algorithm. Creating more general solutions proved more difficult and required additional fitness pressures. Human-like play proved to be even more challenging, requiring different approaches to automatically create human-like behaviours at an individual level to those at a group level.

4. EVOLUTION OF GROUPS FOR AN ABSTRACT COMPUTER GAME ENVIRONMENT

In this chapter, an abstract environment is defined to simulate a simple village scenario that could be potentially from a computer game. The environment is defined such that it resembles a game theoretic dilemma, requiring cooperation and coordination within the group. Genetic programming is used to create a decision tree for the group acting in this environment.

This chapter addresses the hypothesis **H3** as outlined in Section 1.4 that states that we can specify a class of game which captures cooperation and coordination and using evolutionary techniques find solutions for them. This game is an abstraction of a computer game scenario and is designed so that, in order for the agents to survive, they should adopt traits which would be desirable in a computer game. This environment should be suitable for the automatic creation of the behaviours of the agents.

Firstly, the environment to be studied is introduced and the motivations and features of the model are outlined. A comparison with traditional CPR dilemmas and computer games is made. The environment contains spatial and temporal elements which begin to bridge the gap between computer games and abstract dilemmas.

Secondly the GP algorithm to be used is introduced and discussed. In previous chapters, co-evolution was used to create individuals to compete in a game. In these experiments, a cooperative approach is utilised with a move towards creating group behaviours rather than focusing on the individuals. A single GP tree is created to provide the behaviours for the group. In this case, the fitness function concerns itself with the performance of the entire group rather than the individuals.

Following this, a set of experiments are conducted to explore the effect of various game parameters on the GP's ability to create solutions in a fixed environment. The GP parameters remain fixed while the environment's generosity and the number of agents in the game are varied.

Certain parameter sets are then chosen for each configuration. The behaviours of the individuals in each of the evolved groups is examined. The difference in behaviours of the groups acting in the static environment is demonstrated. [1]

4.1 Role-Based CPR Environment

The motivation for this work stems from the computer games domain and the need for more suitable Artificial Intelligence (AI) approaches. The increasing complexity of games and the technologies that drive them, both in terms of graphics and resources available for game processing, push the advancement of AI. If, the Non Player Characters (NPCs) appear to be acting nonsensically, then the user is left with an unsatisfactory experience. The advancement of computer game graphics has the user exposed to more detail than ever before and increased the expectations on game developers to provide extra detail. As computer hardware develops, game makers have more

[1] This chapter is based on the following publication: Alan Cunningham and Colm O'Riordan, An Analysis of Fitness Landscapes in the Evolution of Social Structures for Computer Games, *GAME-ON*, Valencia, November 2008

access to create larger and more complex games.

In a typical computer game, such as a Role Playing Game, there may exist a town or a large collection of NPCs. Usually each person not central to the story will have a static role assigned to them and this role may have no impact on the game. However, upon revisiting this place multiple times during the course of a game, the characters will all be performing the same actions as they did in the first encounter. These characters, even though they are not central to the plot, contribute to the believability of the game. This is identified as a current issue in AI approaches in computer games which exists due to the large amount of time-consuming effort that is needed to fully script each and every character in a large game such that it is performing interesting actions.

The problem of creating complex, active and robust ambient characters is addressed. The goal is to provide a group of agents with simple actions with the hopes of creating interesting group behaviours automatically. An investigation into the application of GP to solve the problem of creating these groups is carried out (for an introduction to GP see Section 2.3.1). GP is a form of evolutionary computation that can allow the automatic creation of solutions for specific problems. GP is chosen as its representation of solutions as trees is particularly suited to the decision making needs of an NPC.

It is believed that abstract economic or social problems mapped into the game can provide such opportunities for diverse and important character roles and will provide a tangible framework for building a town upon. In order to specify suitable environments for these populations to be evolved in, social dilemma problems are used for inspiration, where the optimal behaviour for the group is not necessarily the optimal for each individual.

A fantasy world, on which an environment can be based, is abstracted. The characters are given roles which tie into the context of the game world and provide them with a set of actions with which to act. Small groups of individuals are created and each one is posed a dilemma that they must act in (cooperate or defect), in order to survive. The belief is that interesting groups of agents can be generated by evolving behaviours to solve a dilemma in order to maintain the characters' existence. This would create an automatic relationship and dependency between characters and any changes can be reflected through the whole group. The relationships of the agents which have been implicitly created through the evolution process, both with other agents and the environment itself, are explored.

In order to move the game theoretic dilemma into the computer game domain, it is necessary to introduce extra complexity into the environment, for example the notion of continuous actions. It is this leap in complexity that makes the study of computer game-AI challenging, mainly because the relationship between parameters becomes more difficult to understand. As the complexity increases, it also becomes more difficult to apply evolutionary computation techniques to the games. The required fitness functions become more complicated and the evaluation time increases. The effort required to evolve solutions is one of the main reasons that EC has not had widespread usage in games. For the same reason, these techniques cannot be used to provide real-time solutions.

The environment for the agents is restricted in order to help alleviate these efficiency problems. In order to model the continuous environment, agents have a location and the ability to move around the world. Moving is abstracted to a single unit of distance with a fixed cost and each location is fixed in the world. Time occurs in discrete units, in which an agent's action is executed in full. This allows the addition of temporal and spatial elements to the environment without too much computational overhead. This allows the game to be executed in a timely fashion and means that, for this environment problems like navigation, for which excellent techniques already exist, can be ignored.

To help constrain the game from an evolutionary point of view, the selection of nodes that the GP can choose for an agent is reduced. This is achieved by introducing roles for the characters into the game. The notion of roles can be compared to traditional scripting, where a type of character can be created with a set of behaviours and several of these characters instantiated throughout the game world. Once an agent has adopted a role, the number of potential actions that agent can carry out is reduced. This is reflected in the GP process, by restricting the combination of nodes with respect to the role chosen for the agent.

An agent's actions are kept simple and discrete, taking a single unit of time and having a cost to the agent in terms of health. An action can be regarded as a simple single action in a computer game or as a complex set of behaviours that the agent has access to, such as a behaviour that is part of a behaviour tree implementation. For these experiments, it is enough that actions have a semantic meaning, an effect on the environment, an associated cost and a time to execute. In a continuous game, one of these abstracted actions might be implemented using a complex set of techniques for example, use navigation algorithm to go to some location, play animation to pick up some object and effect some world object in some manner. For the abstract game, the actions need only have an appropriate time and cost in order for them to have meaning in a real game.

Compared to a modern computer game, this is a major simplification. However, the goal here is to create behaviours automatically and not the complete simulation of a real world event. If behaviours can be generated for the group in the simplified game, perhaps the method will work for more complex environments but it is very difficult to start with a fully featured game world and say something quantitatively definitive. The abstractions exist to represent some agent behaviour that might exist in a computer game. Can a small number of actions and restrictive individual reasoning be combined in some way to produce interesting groups of agents? Further to this, can these groups be generated automatically?

4.1.1 Game Definition

The model consists of a group of agents, each of whom must adopt a role and use their available actions in the process of resource collection, to survive in the environment. The game resembles a provision problem from common pool resource dilemmas. The agents must choose to cooperate in order to provide a resource that maintains their health. Cooperating in this case, involves the agents expending health points in order to generate the resource. If cooperation happens, the agents' sacrifice will yield a resource that will replenish their health and maintain their life time. If it does not, then the agents' sacrifice will have been for nought, and they will die more quickly than if they had done nothing.

The food created is a shared resource. All agents have access to the food regardless of their contribution to its creation. The dilemma lies in cooperating and contributing to the production of food with a cost to your health points and trusting that other agents will also contribute or opting out of the food production process and reducing the cost to your health, but having the possibility of using the food if it is created.

The game is broken up into time periods called *Ticks*, with each action in the game taking a single tick to perform. Each agent has a starting value of health and each action that an agent performs has some cost to their health. Agents can replenish health by eating food. Each agent has a set of conditions to reason about the environment and a small set of role-specific high level actions.

The focus is on one scenario that has been defined for the agents acting in this environment. The dilemma posed to the agents is as follows: there are n resources that need to be combined in order to create food for the agents.

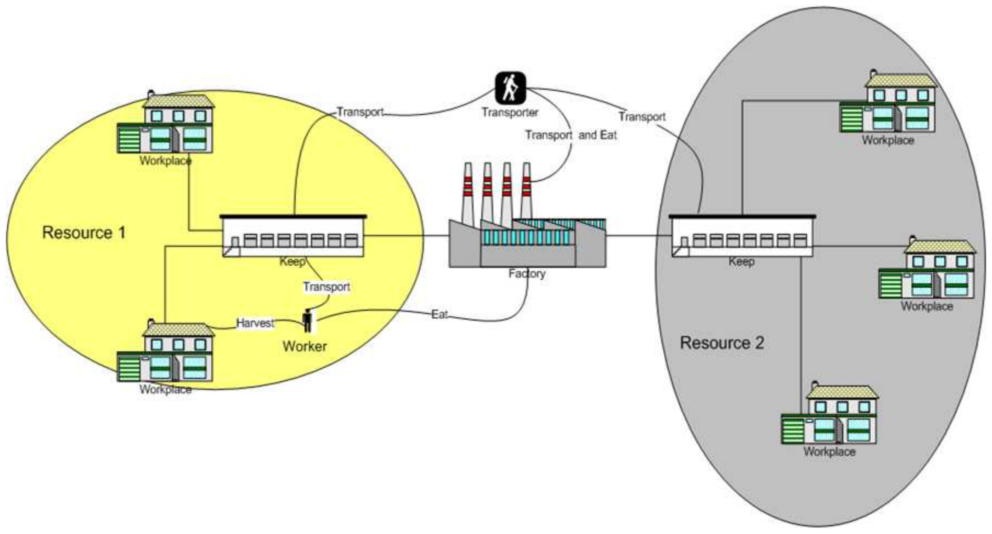


Fig. 4.1: The Game Layout

Each resource must first be harvested and then transported to a refinery to be refined. Once this has been completed, the refined resource can be brought to the factory where it can be used for the production of food. Where $n > 1$, all resources are needed in some proportion, to create the food. Figure 4.1 shows the layout as it might be in a video game with a group size of six and where two resources are required to generate food.

Harvesters gather the resource at their workplace which can be stock piled to a predefined limit. The resource is then brought to the keep for refinement. A *Transporter* can then bring the refined resource to the factory to be used for food production. All resources are necessary for food production to occur. In order to solve this specific problem, enough agents must harvest the required amount of each resource in order to create sufficient food to sustain all the agents involved. There must also be enough agents transporting goods between the keeps and the factory such that the food can be created in a timely fashion.

There exists a coordination problem with respect to the role adoption for the group. There must be a sufficient number of agents fulfilling each role to satisfy the demand the group has for the food. This coordination problem is further complicated, as all agents *types* are required for the creation of the food and roles are chosen at the beginning. The agents do not have any means to communicate but they have to be able to synchronise their actions in such a way that allows this coordination to happen. In this case, this problem is made easier by having discrete time intervals although, agents do not have complete knowledge of the actions of others and can only reason in general about the world.

Agents must cooperate in order for the group to survive. However, not all agents may be needed to contribute. This could lead to instances of individuals exploiting the generosity of the group. This idea stems from the notion that, in simple environments it is easy to calculate and generate the optimal solution. The optimal solution is, from a computer game's perspective, rarely used usually because they are either too difficult to beat or are no fun. In the previous chapters, it was shown that for simple solutions creating the game-theoretically predicted behaviours was straightforward but creating human like or interesting behaviours proved more challenging. In this game, there can be more group members than are necessary, an easy problem to solve and a fitness function that only wants the group to survive. Opportunities may exist for some agents to do no work to help the group but nevertheless, the group can maintain a high fitness. Can a environment such as this produce interesting behaviours using an evolutionary process?

The notion of roles is adopted as a convenient way to constrain the GP process. Strongly-typed genetic programming is used to limit the nodeset combinations and ensure correctness of the evolved trees. From a computer game developer's perspective, it is important that the automatic generation of AI solutions is fast and error free. The constraints on the evolutionary process mean that it is quicker as less time is spent on evaluating incorrect or nonsensical trees. The trees also meet at least a minimum standard of correctness, insofar as no functions or nodes which cannot be used together are combined, and this allows them to be used in a game with less testing.

A discussion of the roles an agent can adopt is provided and a description of the GP algorithm follows, explaining how they both fit together.

4.1.2 Agent Roles

The roles are created for the agents in such a way that allows for the expansion of both, group size and resource amount, as is necessary. The actions created for each agent are kept at high level descriptions which would typically be decomposed into several smaller tasks if applied in a commercial game. The number of actions is small for each agent, allowing for easy expression and evaluation during the evolutionary process. The small number of actions allow the investigation of whether or not it is possible to create interesting group behaviours with a simple set of individual actions. The actions defined here are also constrained by the requirement for agents to cooperate in order to generate food.

Harvester

Each *Harvester* is specific to one resource and does not change role during its lifetime. The available actions are:

Harvest: The agent must be at their workplace, or must travel there incurring the time and cost of moving, in order to harvest. The agent then takes one tick to produce one unit of resource which is stored in the workplace.

Transport: The agent must be at their workplace, or travel there incurring the time and cost of moving, in order to Transport. Once the agent is at the workplace they take one unit (the *Harvester*'s carrying capacity) from the workplace stock if available, and bring it to the keep. This process takes one tick.

Eat: The agent must be at the factory in order to eat. Once at the factory the agent can consume one unit of food, if available, per *Eat* action.

Idle: The agent performs no action for the tick. This is the default action for an agent if the conditions are not met for the other actions, for example if after arriving at the factory there is no food to eat, the agent idles.

Transporter

The *Transporter* is responsible for bringing the refined resources from the keeps to the factory. In this scenario we will assume that two resources are needed to produce food.

TransportRandomResource: The agent travels to one of the keeps, chosen randomly, incurring the time and cost for moving. If a refined resource is there, the agent can transport one unit to the factory. Once the agent is carrying the resource they incur the cost of transport.

TransportResource1 and TransportResource2: The same mechanics as "TransportRandomResource" but corresponds to a specific resource.

Eat: The agent must be at the factory in order to eat and incurs the time and cost to get there, if necessary. Once at the factory, the agent can consume one unit of food, if available, per *Eat* action if it is available.

Idle: The agent preforms no action for the tick. This is the default action for an agent if the conditions are not met for the other actions, for example if after arriving at the factory there is no food to eat the agent idles.

4.1.3 GP Actions and Costs

Table 4.1 shows the associated costs to health for an agent to perform an action. However, each action is dependant on the location of the agent for its true cost, as the agent may have to move in order to complete the action. If an action fails for some reason, for example, trying to transport a good which does not exist, the agent incurs an *Idle* cost.

The cost to produce a unit of food depends on the number of resources in the game. All resources are needed to create a unit of food, but a *Harvester* can usually only harvest one type of resource. The cost for food increases as the number of resources increase, but so does the number of agents that must share the food replenishment.

Equation 4.1 shows the minimum cost of food production in a two resource game. However, this cost increases after the first units are created when incorporating the return journey to the workplaces. In this case, two units of refined resources are needed to create one unit of food. Also of note is that three agents are needed for the production of a single unit of food which is now available for the whole group to consume.

$$Food = \underbrace{(H + T + M) * 2}_{2Harvesters} + \underbrace{(2 * T) + M}_{Transporter} \tag{4.1}$$

Action	Health
Harvest	-1
Transport	-1
Idle	-0.5
Move	-0.5

Tab. 4.1: Action Costs for Agents

4.2 GP Approach

In this section, the GP approach taken to create group behaviours in the defined environment is outlined. First, a discussion of GP's suitability to the problem domain is presented. Secondly, the specific algorithm implementation for the GP process is described. Following this are the detailing of the GP parameters and the fitness function to be used in the application of GP to this domain.

4.2.1 GP Suitability

Firstly, evolutionary computation techniques were chosen for the purpose of creating group strategies automatically. The cost of creating AI solutions for background characters in computer games is the number of hours it takes humans to code those solutions. If this process can be assisted using automatic methods then it may contribute to a saving in development and an advancement in the level of detail that an AI solution has.

The nodesets chosen for the GP process were purposefully restricted and kept small to investigate the notion of creating interesting group behaviours from simple individual behaviours. The functions and terminals are deemed sufficient to solve the problem while remaining simple enough to be implemented quickly in an actual computer game. The investigation of the application of GP to static environments for large sets of parameters in this chapter will reveal whether or not the nodesets are sufficient to provide desirable solutions.

The structure of the GP solution, to create a single tree to represent the group, was chosen to simplify the definition of a group and allow for the specific evolution of the group behaviours. The roles of the individual remain simple enough such that they are not really of interest, whereas the group behaviour is the primary concern and as such is evolved in a cohesive way. This may not be suitable in domains where more complex individual behaviours are necessary. Trees would grow too large to make the evolutionary approach ineffective.

The group created could be classified as heterogeneous. This form of GP group creation has been shown to be successful, for example, by Haynes and Sen [1997] and Luke and Spector [1996], amongst others. Murata and Nakamura [2004] observe that heterogeneous teams can be difficult to evolve as the search space is so vast, but also state that the heterogeneous model allows for the generation of complicated systems. In their model, Murata and Nakamura [2005] create agents and then automatically assign roles based on their abilities. The approach adopted in this chapter is simpler, first selecting the role and using constrained creation techniques to create agents for those roles.

Although heterogeneous models tend to have larger search spaces, the convention of adopting an unchanging role for the agents and the use of STGP helps to constrain the search space in this application. The nodes used in this research are high-level abstractions of computer game actions and as Ciesielski et al. [2002] observe that high-level functions result in more effective team behaviours.

4.2.2 GP Algorithm

The GP approach to this scenario is a cooperative one. In Chapter 3, a competitive co-evolutionary method is used to create individuals to play in a group. The objective of this chapter is to create group behaviours to play in an environment. It is no longer necessary to measure individuals against each other, rather, the focus is on how well the group can survive the environment. In Section 3.6, the fitness function for individuals was altered in order to create the optimal Group performance. This method could be applied here, but the addition of *Roles* complicates the implementation.

Not all agents or actions are interchangeable, which means that the evolutionary process is more complicated if focusing on an individual level. Selecting a *Transporter* to undergo a crossover operation with a *Harvester* could create an invalid tree. Additionally, once individuals are giving full preference to the group's performance over their own, treating the group as an individual that must be evolved, is equivalent.

A single tree is evolved to represent the behaviours of the entire group. From the root of the tree, an agent role branch is defined for each agent in the game. Each member of the group has a single branch with their role chosen from the *Role* set. By checking this level of the tree at the start of the game, it allows the required agents to be found and loaded into the game easily.

Once the role has been selected for an agent, that agent will retain that role for its entire lifetime. The role chosen also limits the potential actions that the agent can have. Each agent's subtree is composed of the nodes from their respective nodesets. Tables 4.2 and 4.3 show the GP node sets that are used to create the trees of agents. Not shown is the full list of constants which may be used in comparison operations. The *Role* set is a selection of functions which takes one argument from the specific node set of that role.

The function *If* for each agent is combined with nodes from the *Decision* set and the *Environment* set and a set of constants (not shown) for the agents to make judgements about their world. The notation $f(i)$ specifies that the function takes i arguments. The first argument is the condition to be checked and must be selected from the decision set. The second argument is the true branch and the third is the false branch both of which must be selected from the role set. Once an *If* node is executed, it should eventually return an action for the agent. In order to get to the action, a series of *If* functions may have to be evaluated each with their own decision comparator.

The *Environment* set nodes are based on the agent's current knowledge of

$RoleSet$	$Harvester_N$	$Transporter$
$Harvester_1$ f(1)	Eat	Eat
$\vdots$	$TransportRaw_N$	$TransportRefined_N$
$Harvester_N f(1)$	$HarvestRaw_N$	$TransportRandomResource$
$Transporter f(1)$	$Idle$	$Idle$
	If f(3)	$If f(3)$
	And f(2)	$And f(2)$

Tab. 4.2: Agent Roles and Role-specific Nodes

the environment. For nodes like *Health* and the *Distance*, these values are up to date every *Tick*. The stock level nodes are updated every time the Agent visits those locations. There is no information sharing between the agents and any agent wishing to use stock information must discover it for themselves.

The *Decision* set takes two arguments, both of which can be from the environment set or the set of *Constants*. The two arguments are then compared in a left to right fashion, i.e. the *Greater* function returns true if arg(0) > arg(1). The function *And* allows the agents to add multiple actions to their action queues. This function takes two arguments which have to be other functions. When the agent's tree is executed, an action is placed onto the agent's action queue. This action is executed on the following game tick. Once the action has completed the decision tree is executed again to get the next action.

Each agent in the game contains an action queue. As actions are completed by the agent, they are removed from the agent's action queue. When an agent's action queue is empty, their subtree within the group representation is executed. The agent's functions and terminals are constructed such that when the tree is executed, actions get added to the end of the queue. Once an action has completed, either successfully or otherwise, it is removed from the queue and the next action becomes the current action.

$Decision$	$Environment$	$Constant$
$Greaterf(2)$	$Health$	$\{1, 2, 3, \ldots\}$
$Lessf(2)$	$WorkplaceStock$	
$Equalf(2)$	$DistancefromWorkplace$	
	$DistancefromFactory$	
	$FactoryStock$	
	$KeepStock$	

Tab. 4.3: Common Nodes for GP

When an agent's subtree is executed, it can be constructed in a way such that it adds several actions to the agent's queue at one time. When the number of actions allowed to be added to the queue is limited to a small number, the problem becomes more difficult to solve for the GP algorithm using the nodesets defined here. The effects of this limit are discussed later in this chapter.

4.2.3 GP Parameters

GP parameters dictate various conditions of the evolutionary process, from population size to the number of generations that a population undergoes to the way individuals are created in the population. The following evolutionary parameters were arrived at through experimentation and comparison with typical values used in similar research like Doherty [2009], Doherty and O'Riordan [2006a,b], where a similar STGP algorithm is applied. Many configurations of population size, generation length and crossover and mutation probabilities were evaluated, with the ones chosen determined to be the best performing all round configuration for different variations of the game environment.

In each evolutionary run, n trees are evolved over m generations. A full list of the GP parameters is outlined in Table 4.4. The fittest individual at the

Parameter	Value
Population Size	500
Number of generations	100
Creation type	Ramped half and half
Creation Probability	0.02
Crossover Probability	0.90
Swap mutation Probability	0.10
Maximum depth for creation	6
Maximum depth for crossover	17
Elitism enforced	1

Tab. 4.4: Parameters for GP

final generation is chosen as a representative of that run. Random trees are initialised according to the constraints in the nodesets and each tree must begin with any of the functions from the *Role* set. These trees are then evaluated using the fitness function outlined in Section 4.2.4. The selected members of the population are then subjected to crossover and mutation with the probabilities 0.9 and 0.1 respectively. The crossover and mutation operators are outlined below. Once evaluated, the trees undergo tournament selection in order to be chosen for the next generation. A tournament size of 5 is used in order to avoid a rapid convergence from selection pressure. Elitism is enforced so that the best individual solution is propagated through each generation.

The creation of individuals at the first generation is initiated by first creating a subtree for each agent in the group. The root of each subtree is taken from the *Class* nodeset. The argument that the class node takes must be a function from that agent's *Role* set on creation. The tree growth is limited by an initial maximum depth of 6 which ensures that very large trees are not created. The maximum depth for growth in crossover is limited to 17. This ensures that the trees in the population do not become very large and therefore very inefficient to process and evaluate.

Crossover is performed on the individuals in the population after evaluation when the next generation of individuals is being created. An individual is chosen for crossover with a predefined probability. If crossover is to be performed, a second individual is chosen from the population for the operation to take place. The crossover algorithm then chooses a random point on the tree of the first individual and then randomly tries to find a node of the same nodeset type from the second tree. The node on the second tree is chosen at random until either a match is found or the process fails to find a match after twenty random node selections. If no match is found then crossover does not occur. If there is a match, then the points are then swapped on both trees such that the complete subtree of the node chosen is also moved to the new tree.

Mutation is performed on an individual by selecting a point at random in its tree. A new node is then chosen from the same nodeset and inserted in its place. This new node is then grown and created much like in the creation process of the population of the first generation. If the node is a function, appropriate nodes are chosen to branch out and grow using the same depth limitations for new trees. If the node is a terminal, it is simply exchanged with a random terminal from the same nodeset.

4.2.4 The Genetic Program Fitness Function

The fitness of a member of the GP population is determined by the performance of the entire group of agents that the tree represents. A game length of 1000 ticks is chosen to be long enough for a representative of the behaviours over a continuous amount of time while still remaining tractable. The game has completed once this length has been reached or when all the agents have no health points remaining. The score for the group is then calculated as follows:

*(Sum of the Worker's Health *1.5)+ Sum of the Worker's Ticks Alive - (depth of tree/4) - (tree length/4)*

The main score for the fitness of the group is the number of ticks for which the agents are alive. This directly corresponds to how well, as a group, the agents performed. A bonus is added to this number for the amount of health an agent has at the end of the game. The amount of remaining health at the termination of the game is given a bonus multiplier to provide an incentive to promote survival in group. If the solution found cannot sustain all the agents intended, there is a preference to have some agents alive. The bonus was chosen through experimentation and provides a small compensation when comparing agent health and game length (an agent's maximum health is one tenth of the maximum lifetime).

The number of ticks the agents are alive, represent a measure of progression throughout the evolution process. In the early stages, it is highly unlikely that any agent will survive the specified game length. If there are not enough agents contributing for the survival of the group, the best the agents can hope for is exploitation, that is, agents tending towards idling, an action which incurs the lowest cost and thus, survival for longer.

The length and depth of the GP tree being evaluated serves as a penalisation of the fitness score. The rationale behind this thinking is that if two solutions have the same group performance, the one which expressed it most succinctly should be favoured. More succinct solutions are preferred in the general application of these techniques as it makes them easier to understand, quicker to execute and saves resources during evaluation and in the final applications of the trees. Both the length and depth are normalised so as not to limit the ability of the GP process to create solutions. As the group is created as a single tree, with each member getting a distinct subtree, the length and depth can grow large. The distinction between length and depth is that length is the total number of nodes in the tree, where as depth is the maximum distance from the root node to a leaf.

The value of 4 chosen to normalise both length and depth was arrived at through experimentation. If the penalisation is too harsh, the solutions fail to ever survive the game and resort to idling. Without cooperation in the group, this is the way agents prolong their lifetime. If, on the other hand, the punishment is too low the trees suffer badly from bloat and because a single tree represents an entire group and accordingly can have a large width, this is especially problematic.

The amount of time each evolution takes is linked to how the fitness score is progressing. The game length is non-deterministic as the agents may die out quickly or all survive until the end. As the fitness of the population increases, the game will take longer to evaluate as the agents survive for longer.

4.3 Experiment 1 - Single Resource

In this section, a series of experiments are presented which demonstrate the potential for the GP algorithm to create behaviours for groups in the simplest environment. The goal of this experiment is to perform a preliminary investigation the performance of the GP algorithm in the simplest version of the newly defined environment. An exploration into the effects of varying the parameters on the performance of GP algorithm's ability to create solutions is undertaken. A comparison is made between the performance of the GP when the agents are constrained by roles and when they agents can choose any action.

The environment remains static, that is, there are no outside factors affecting the state of the game world. The actions of the agents are the only thing that can change the world. This setup will act as a baseline for establishing the parameters for the game environment that allow good solutions to be

Harvester	Transporter
Eat	Eat
$TransportRaw_1$	$TransportRefined_1$
$Harvest_1$	Idle
Idle	If(2)
If f(2)	And(2)
And f(2)	

Tab. 4.5: Two Roles for a Single Resource

generated and can be used for comparison with more complex environments later.

Firstly, the GP algorithm chooses a role for each agent and builds a decision tree for each one. The group is then pitted against the environment for a set time. The fitness of the GP tree is based on the performance of the group after this time. A second approach is then explored where the GP algorithm can build a generic worker and provide it with any of the available actions. This allows for an exploration of the advantages and disadvantages involved in introducing roles to the evolutionary process.

This game consists of a single resource which must be harvested and transported to create food. The node sets from Table 4.2 are restricted to two types of agents as detailed in Table 4.5. The GP tree consists of first, choosing the various classes needed and then for each class, assigning decisions and actions. This problem, when translated to the game, has fishermen going fishing on boats and bringing their fish to the harbour. The *Transporters* then bring the fish to the fishmongers where it they are turned into consumable food.

The environmental variables, comparison operators and a set of constants which both class of agents can use are detailed in Table 4.3. With these functions and terminals, the agents can make decisions about the game world by comparing their state and the state of the world as it appears to them.

4.3.1 Baseline Game

The game environment variables that remain fixed are as follows:

- 1000 Ticks per game
- The resource must be harvested, transported to the keep for refinement and finally, transported to the factory where it becomes food. The rate of food creation is one to one with the amount of refined resource in the factory. That is, each unit of refined resource in the factory will be converted into a unit of food. The process of food creation takes one tick.
- Resources are located one unit of distance away from keeps and two units of distance from the factory. A keep is located one unit of distance from both the workplaces and the factory. All agents take one tick to travel one unit of distance at a fixed cost of moving.
- All agents start with 100 units of health.
- The costs to the agents' health for performing an action remains fixed. They are detailed in Table 4.1

In the this game, the following parameters are modified and explored:

- The amount of health an agent receives when a unit of food is eaten.
- The number of Agents in the group
- The rate of transforming a refined stock into food
- The starting value of a resource stock

In the single resource game, two agents are required to create a unit of food – a *Harvester* to generate and refine the resource and a *Transporter* to move the refined resource to the factory for food creation. Based on the action costs that have been chosen, the table below shows the minimum cost to the agents to produce and consume two units of food. From this table, it can be seen that the minimum cost to the health of the agents is 7 for the *Harvester* and 5 for the *Transporter*. The sequence for the agents is:

Tick	1	2	3	4	5	6	7	8	9	Cost
Harvester	H	T	M	H	T	M	E	M	M	7
Transporter	M	I	T	E	M	T	I	I	I	5

Where for the *Harvester*:

H:	is *Harvest* the raw resource.
T:	is *Transport* the raw resource to the Keep.
M:	is *Move* to another location.
E:	is *Eat*.

and for the *Transporter*:

T:	is *Transport* the refined resource to the Factory.
M:	is *Move* to another location.
E:	is *Eat*.

The results of the evolutionary runs are organised by the number of agents in the group and presented. Each point on the graph is the average of the fitness of the best individual at generation 100 over twenty separate evolutionary runs which provide a sample of the solution space. The first parameter modified is the food replenishment. This is the amount of health an agent receives after eating one unit of food. The second variable examined is the rate of food production. It takes two agents to create a unit of food so it might make sense to create more than one unit of food when converting a refined resource. Finally, the influence of having a starting stock amount is examined. Each of these variables are connected and impact the agents' potential to sustain themselves for the lifetime of the game.

The results of the trials with varying group sizes can be compared by examining how well the average agents performed. The total amount of ticks the agents are alive for is a good indicator of the performance of the group in the game. By taking this number and dividing by the number of agents in the group we get a normalised figure with which to compare the various agent amounts. What this provides is an insight into the effects of adding extra complexity, in terms of tree width, on the performance of the GP algorithm. As the group gets larger, additional subtrees are added to the solution. The effect this has on the GP is that individuals are larger and therefore slower to execute and evaluate. It also means that single point crossover and mutation may be less effective leading to slower or even, ineffective refinement of solutions. Are these symptoms revealed in the results of the group behaviours?

Secondly, the evolutionary runs are conducted without the constraints of the agent roles. Are the agent roles a help or a hinderance to the GP process in this simple environment? By comparing the average of all the variables in the per-agent runs, it should be possible to see whether or not having roles helped the GP create better solutions.

The parameters that are available for the role and generic single resource game are:

Workplace Starting product This value represents the initial stock of raw resource in each workplace. The values explored are 0, 2, 4, 6, 8 and 10.

Factory Food Multiplier This value represents the rate at which a unit of refined resource is turned into food in the Factory. The multipliers explored are 1x and 2x.

Food replenishment This value represents the amount of health points an agent receives from consuming a unit of food. The replenishment values examined are 1, 3, 5, 7 and 9.

Agents This value represents the group size for the game. The group sizes are 2, 4, 6, 8 and 10.

For evaluation each combination of the parameters is averaged over twenty evolutionary iterations.

The configurations are compared using average agent lifetime which is the number of ticks all agents were alive for, divided by the number of agents in the group. The survival rate is also used which is the games where all agents live for the entire duration of the simulation divided by the number of iterations.

Evolutionary trials in the role-based game give the following results:

- The value of starting stocks is found to have no effect on the average life span of an agent in the evolved group. That is, if an agent's workplace has stock existing which didn't need to be harvested, it doesn't improve the performance of the groups in terms of average agent lifetime.
- A group is more likely to survive when there are less agents in that group. With the food multiplier at 1x, the results of trials are arranged by agent number with the results of trials for the parameters for food replenishment (1, 3, 5, 7, 9) and starting stock (0, 2, 4, 6, 8, 10) combined. The average ticks alive for two agents is 724 and the probability of survival is 0.6 across all these games (600 trials). The average ticks alive falls to 696 with a 0.36 probability of all agents surviving when there are ten agents. In these trials, when the replenishment for food is 1 or 3 it is impossible for all the agents to survive, but they may be able to do better than just idling. If the agents do some work they can prolong their life beyond what would be expected from *Idling*. However, this is difficult to evolve as penalisation is harsh for incorrect trees with the low replenishment and the Idling behaviour provides an

easy and effective alternative. They are included to see if there are any differences when group number or starting stock is varied.

- The minimum cost for two units of food is 11. When the replenishment is 5 the two agent groups can survive in each of the twenty trials. The loss of health for the agents over the nine ticks is offset by the starting health of the agents and the number of the ticks in the game. When the number of agents is 10 the probability of survival, in 120 trials, drops to 0.03. When only the replenishment numbers 5, 7 and 9 are considered, there is only a 5% difference in the average agent lifetime between 2 agents and 10 agents. However, the survival rate for the entire group falls by 39% when the number of agents is increased from 2 to 10.

- When the value for food replenishment is at 7 or 9 per unit, all agents survive when there are two agents in the group. The probability of all agents surviving drops to 0.81 (20 trials) when the group size is increased to ten. This demonstrates that the GP algorithm can create cooperating and coordinating groups when there are favourable conditions. It also shows that there is increased difficulty for the GP algorithm when agent numbers increase. The probability of survival for four agents is 0.94, for six agents is 0.92 and for eight agents is 0.87.

The GP algorithm has demonstrated that when using the convention of role-based agents in the single resource game, solutions can be found where all agents survive. When there is enough food replenishment to cover the cost of the actions needed to create the food, the agents work together and can survive for the game lifetime. When the food replenishment is 5, it is just enough to allow the agents to survive for the game if they sacrifice some of their health. In this scenario, however, the GP algorithm performs poorly when there are more than two agents. This happens, presumably, because the agents' strategy have to almost be perfect and this perfection is more difficult to achieve through evolutionary means due to an increase in tree width.

Generic Workers

Evolutionary runs are performed for the same parameters as above with the notion of roles removed. This experiment is performed in order to investigate the effect of including roles for the agents on the GP algorithm's performance and the performance of the generated behaviours.

With the roles removed, the GP algorithm can create a generic worker, who has a workplace where they can create the raw material. This raw material can be brought to the shared keep for refinement and then brought to factory to be turned into food. By performing this step, the effect of constraining the GP using roles is evaluated. In this simple single resource game, the generic workers have a small number of actions and only one more than a role-based *Harvester*.

The principle of creating a group behaviour is the same as for the role-based game. The root node has N-subtrees where N is the number of agents in the group. An agent's subtree is executed when their action queue is empty. The constraints remain on the functions and terminals during evolutionary operations. Only nodes from the same nodeset can be swaped in crossover or mutation (environmental, constants, decision, action).

This worker can harvest a resource, transport the raw resource to the refinery, transport the refined resource to the factory, eat and idle. Each worker has their own "workplace" where they harvest a single type of resource. A single keep exists in the game to refine the resource. A single factory exists where the refined resource is converted into food. Any agent may transport the refined resource if it is in the keep and any agent may eat food if it is in the factory.

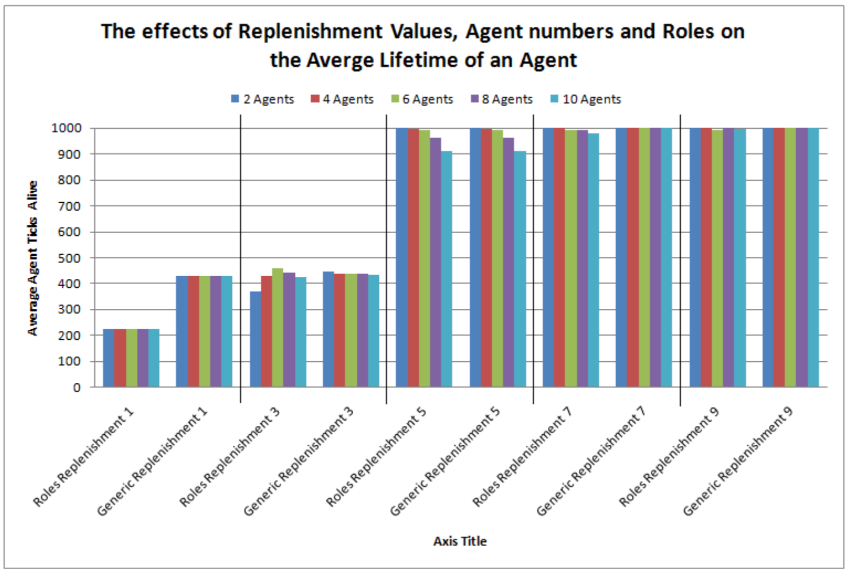


Fig. 4.2: Single Resource - Comparison of Average Agent Lifetime

The cost of the actions remain the same as the role-based game (specified in Table 4.1). The minimum number of required agents drops to one from two with the removal of the role.

Figure 4.2 presents a comparison of the effect of group size on the average agent lifetime between the role-based and the generic agents. Figure 4.3 illustrates the effect of replenishment values and agent numbers on the probability of survival rates between role-based and generic evolution.

Evolutionary trials in the role-based game give the following results:

- Starting stocks continue to have no significant effect on the average life span of an agent.
- A group is more likely to survive when there are less agents in that group. With the food multiplier at 1, the results of trials are separated by agent number with the trial of the parameters for food replenishment

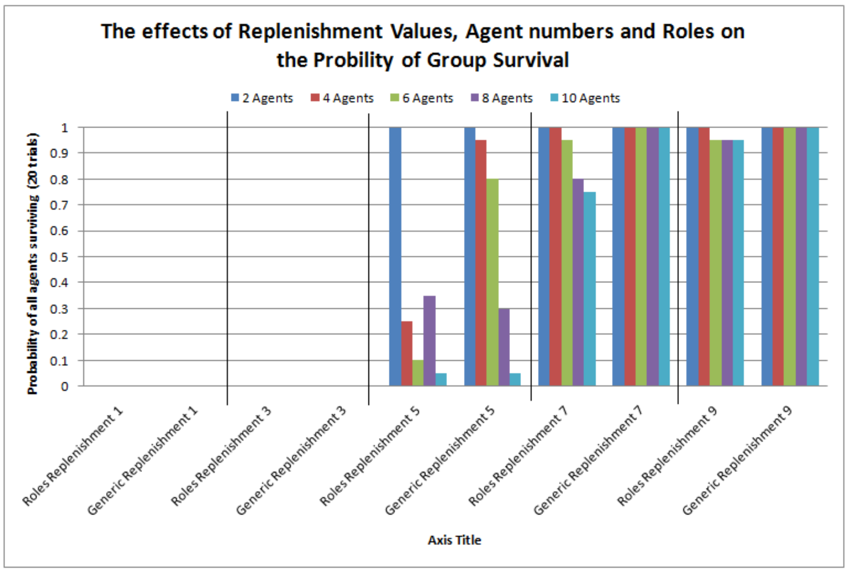


Fig. 4.3: Single Resource - Comparison of Survival Rates

(1, 3, 5, 7, 9) and starting stock (0, 2, 4, 6, 8, 10) combined. The average ticks alive for two agents is 771 and the probability of survival is 0.6 across all these games (600 trials). The average ticks alive only falls 14 ticks to 757 with the probability of all agents surviving falling to .42 when there are ten agents. In these trials, when the replenishment is 1 or 3 it is impossible for all the agents to survive, but they can do better than just defecting. They are included to see if there are any differences when group number or starting stock is varied.

- When the value for food replenishment is at 7 or 9 per unit, all agents survive regardless of group size or starting stock value.

The main observable differences between the role-based and the generic runs was in the execution time of the evolutionary runs and the lifetime that agents can attain. The total experiment took 59% longer to run when the roles were removed. This is attributed to trees growing larger due to the less constrained nature of evolving the generic workers. The maximum lifetime that is attained when the food replenishment is too low for sustainment increases from 225 ticks in the role-based game to 429 ticks in the generic game (Figure 4.2). This is attributable to the fact that all agents in the game can now harvest the raw resource and create food without relying on others.

The generic setup out performs the role-based game when there are more agents. When the replenishment value is seven or greater, the amount needed to cover agent costs for making food, the agents in the generic groups all survive. In the role-based game, only the two agent setup survive all the trials. Once the agent number is increased to 10, all agents survive in 80% of games.

Regardless of the game type, agent numbers or food replenishment value, providing the agents with some starting stock does not increase their chance of survival in the game. In the game where replenishment is 5, the minimum needed to sustain the agents for 1000 ticks (requiring some sacrifice of their own health), the generic game is better able to create groups that survive on this low endowment when agent numbers are greater than two. The generic approach has a 31% higher probability of all agents surviving, when replenishment is 5 and the number of agents is 4 or more (4, 6, 8 and 10).

4.3.2 Limiting the Action Queue

In the baseline game the action queue size was unlimited, meaning that the GP tree representing an agent's strategy could append on several actions at a time to that agent's action queue. If the action queue size is restricted, does it change the performance of the evolved behaviours? The purpose of this experiment is to investigate the GP algorithm's ability to create specific decisions for the agents within the group that choose the correct action for the agent to perform for specific game contexts.

A set of experimental runs are conducted to investigate the effect of placing a limit on the action queue length can have on the performance of the role-based and generic agent setups. A subset of the above parameters is chosen to compare the effects of limiting the action queue in both the role-based game and the generic worker game.

The evolutionary runs are conducted in the same fashion to the unlimited queue runs in Section 4.3.1. The same evolutionary parameters and operators are used and each parameter uses the average of twenty best individuals at generation 100. The only difference in this game is the length of each agent's action queue, which is set to two.

An agent's action queue is empty for two reasons, either it is the start of the game or the agent has, successfully or otherwise, completed all its queued actions. When this occurs, its subtree in the GP solution is executed which makes decisions about the environment and adds actions to the agent's queue. Multiple actions can be added at any one time. In this instance, the number of actions that can be added at any one time is limited to two. This provides two pieces of information. Firstly, can the GP create strategies which can respond to the immediate state of the game and secondly, if the GP has sufficient environmental variables with which to perform this task?

The summary of the findings is as follows:

- The value of starting stocks do not have an effect on the average lifetime length of agents regardless of roles or generic based evolution, with or without a limited action queue.
- As the number of agents increases, there is a decrease in the average lifetime of agents (Figure 4.4) and a decrease in the survival rates of agents (Figure 4.5).
- The Generic unlimited evolution performs best with the unlimited role-based evolution a close second. When the action queue is limited, even

if there exists enough replenishment for agents to survive, there is a large drop in average agent lifetime length (Figure 4.6).

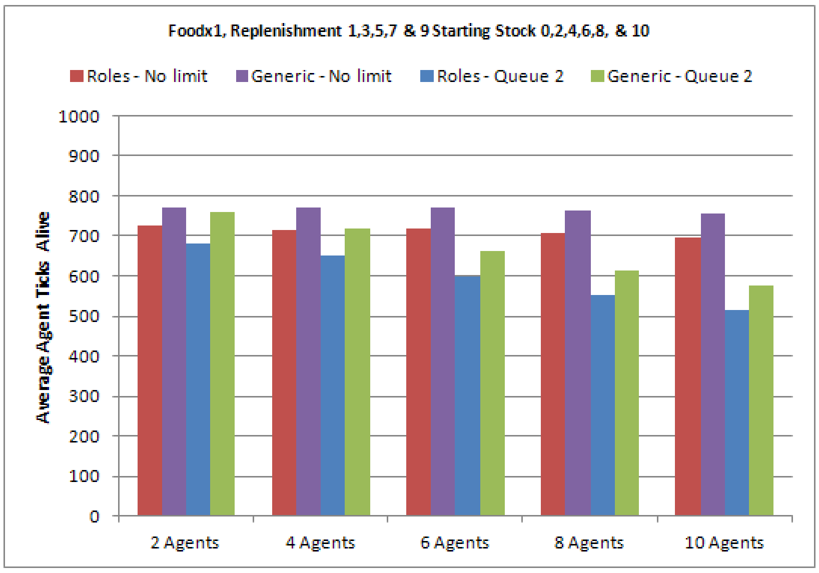


Fig. 4.4: Single Resource - Effect of Agent numbers

By placing limits on the action queue, both the average agent lifetime and the survival rates decrease. The greater the group size, the greater the effect of limiting the action queue is felt on the average agent lifetime and survival rate. This indicates that the GP algorithm has more trouble refining exact decision making as trees representing groups grow wider. In Sections 4.3.1 and 4.3.2, a range of parameters were established for which the GP algorithm can create groups of agents that successfully survive the defined game environment

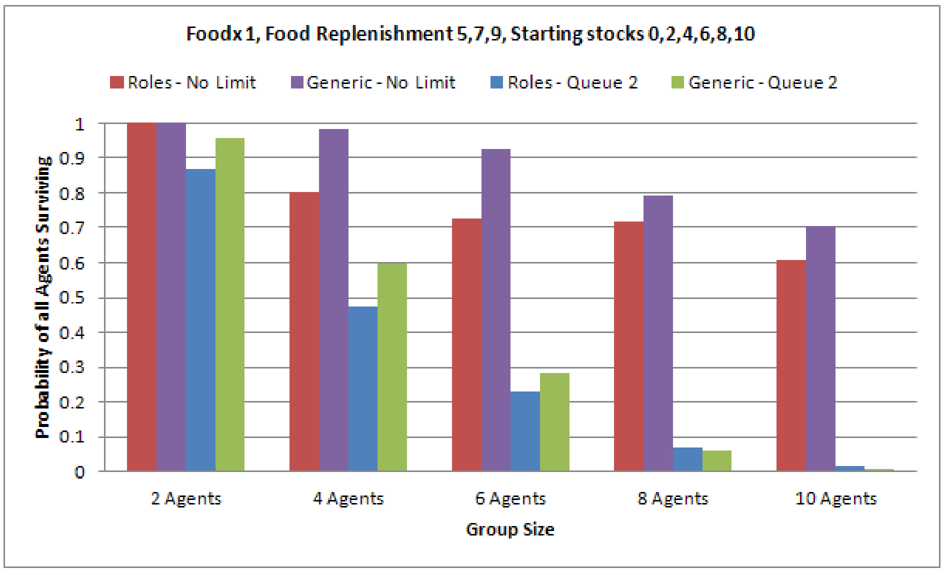


Fig. 4.5: Single Resource - Probability of Survival

4.4 Experiment 2 - Two Resources

In Section 4.3, it was shown that it was possible for the GP process to find solutions in the single resource environment under many parameters. When there was a high enough value for the replenishment of food, solutions were found in which all agents were able to survive. The performance difference between the *Role-based* and the *Generic* evolutionary approaches were similar however, the *Generic* typically took longer to evolve.

In this section, the complexity of the environment is increased by introducing a second raw resource. The addition of a second resource adds extra relationships between the agents. The role-based system employed means that, a *Harvester* may only create one type of raw resource. This additional resource is harvested in the same way as previous experiments, requiring both a harvesting and refinement stage to be completed by an agent with a new role for the job. This additional resource now means that at least three

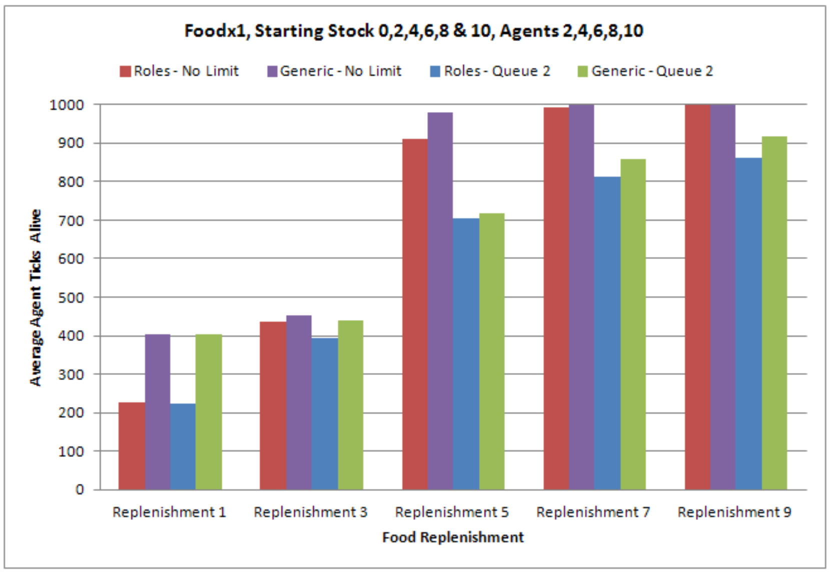


Fig. 4.6: Single Resource - Replenishment Values

agents are now needed to fulfill roles to produce the food. In this case, both resources are needed in equal measure to produce food.

Adding a second resource is the convention adopted to add complexity into the game world. Because two resources are needed to create the food, the cooperation and coordination effort now involves at least three agents. Sharing the food also becomes more problematic, as three agents are required to make efforts in order to create a single unit of food.

In this game, the replenishment value of food is varied, along with the number of agents playing the game. The ability of the GP algorithm to create solutions for this environment is assessed. The nodeset for this experiment is specified in Table 4.2. When a factory conversion rate is specified, it is the multiple of food that is produced for combining a unit of each refined resource. For example, a 1x multiplier or conversion rate means that one unit of food is produced for combining a unit of each refined resource.

In the two-resource game, three agent roles are required. The first for harvesting the first resource, the second to harvest the second and the third to transport the refined resources to the factory so that food can be created. The table below shows the minimum sequence of actions required to create three units of food, one consumed by each agent.

Tick	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Cost
H1:	H	T	M	H	T	M	H	T	M	E	M	M	I	I	9.5
H2:	H	T	M	H	T	M	H	T	M	E	M	M	I	I	9.5
Trans:	I	M	T	M	T	M	T	M	T	M	T	M	T	E	9.5

Where for the *Harvester* (H1 and H2):

H:	is $Harvest$ the raw resource.
T:	is $Transport$ the raw resource to the Keep.
M:	is $Move$ to another location.
E:	is Eat.

and for the *Transporter* (Trans.):

T:	is $Transport$ the refined resource to the Factory.
M:	is $Move$ to another location.
E:	is Eat.

For comparison, a generic agent is created using the GP process. The generic agent is not directly comparable with the role-based agent with the addition of a second resource. A role-based agent is reliant on others to do some work and therefore must share the rewards of the units of food created. A generic agent is able to harvest all resources, transport them for refining, and finally transport the refined resources for food. The generic agent has an advantage in both the ability to create the final unit of food, but also incurs less cost to health per unit of food created. The generic solution is included both, to compare the benefits of employing the agent-role convention, as well as, examining the penalisation on group performance by including the extra complexity of roles.

To perform this comparison, the average agent lifetime and the agent survival rate are used. The average agent lifetime, is the the total number of ticks a group of agents was alive for divided by the group size averaged over the number of game iterations. The agent survival rate, is the number of times all agents survived the entire game length, averaged over the number of solutions for that configuration.

In this instance the parameters that are available for both, the role and generic two-resource game, are:

Workplace Starting product This value represents the initial stock of raw resource in each workplace. The values explored are 0, 5, and 10.

Factory Food Multiplier This value represents the rate at which a unit of refined resource is turned into food in the Factory. The multiplier explored is 1x.

Food replenishment This value represents the amount of health points an agent receives from consuming a unit of food. The replenishment values examined are 9, 11, 13, 15, 17 and 19.

Agents This value represents the group size for the game. The group sizes are 3, 5, 7 and 9.

Action Queue Length This value represents the number of actions that an agent can queue when evaluating their decision tree from within the GP group behaviour. Both an unlimited and a limited action queue are investigated (limit = 2).

Each combination of parameters is averaged over twenty evolutionary iterations in order to provide a sampling of the solution space. The best individual from the final generation of each run is chosen as a representative of that run.

The average performance of each of the twenty representative solutions are used for comparison.

In Figure 4.7, the starting stock is shown to have no effect in the unlimited action queue game. When the queue is restricted, the increase in starting stock leads to an increase in average agent lifetime length. In the role game, the average agent lifetime increases by 15% and 25% when starting stocks are 5 and 10 respectively, over 0 stock. In the generic game, the increases are 10% and 12% for the 5 and 10 stock respectively.

In the less complex single-resource game, the value of starting stocks did not effect the performance of the evolved solutions. The GP algorithm finds better performing solutions in this more complex game when there is some starting stock. The value of starting stock can provide half formed behaviours the opportunity for higher fitness by reducing the number of actions required for food in the short term. This can lead to useful components of behaviours being created and improved upon through the evolutionary process.

In Figure 4.8, the effect of increasing the group size can be seen. In all configurations, except the restricted generic game, the average agent lifetime decreases as the group size increases. Figure 4.9 shows that the small decrease in average agent lifetime actually corresponds to a dramatic fall in the probability of all agents surviving in the game. Interestingly, the Generic setup with the restricted queue has, on average, better agent lifetime lengths across the various group sizes and starting stocks than the unlimited role setup. However, the unlimited Role setup outperforms the restricted Generic setup 209 to 120 in number of solutions where the entire group survived (out of 1440 games).

The generic solutions have better performance as the number of restrictions on agents' actions is reduced. This has the effect of removing the constraint of relying on other agents for parts of the supply chain for food. Each agent now has the capability of producing food themselves. This is demonstrated

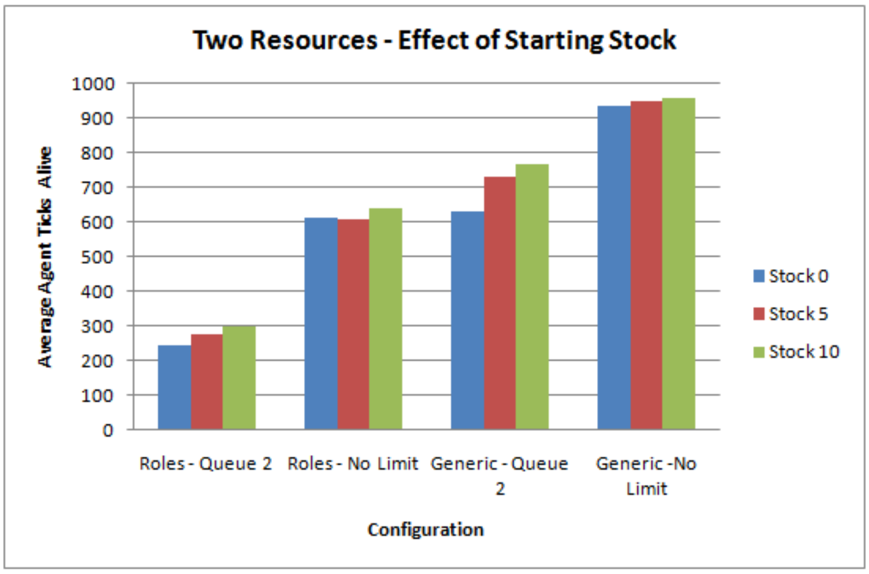


Fig. 4.7: Two Resources - 1xFood - Starting Stock

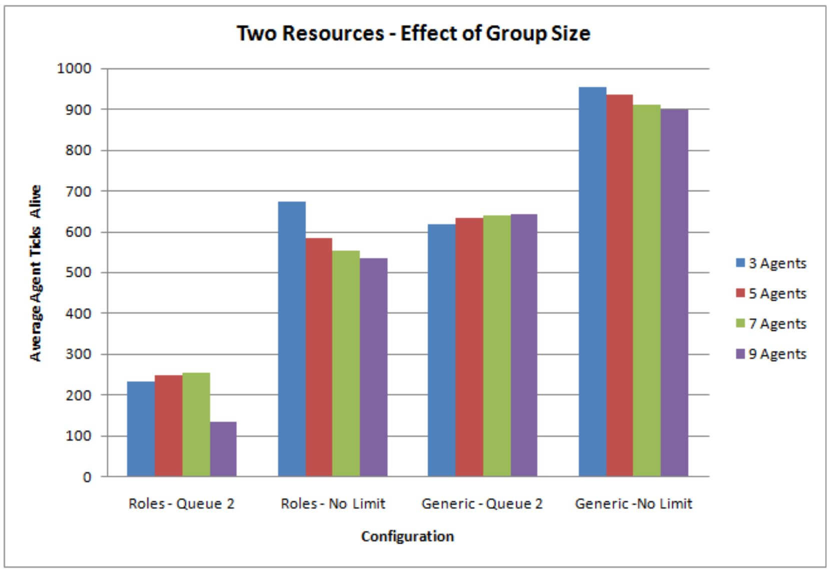


Fig. 4.8: Two Resources - 1xFood - Effect of Agent numbers

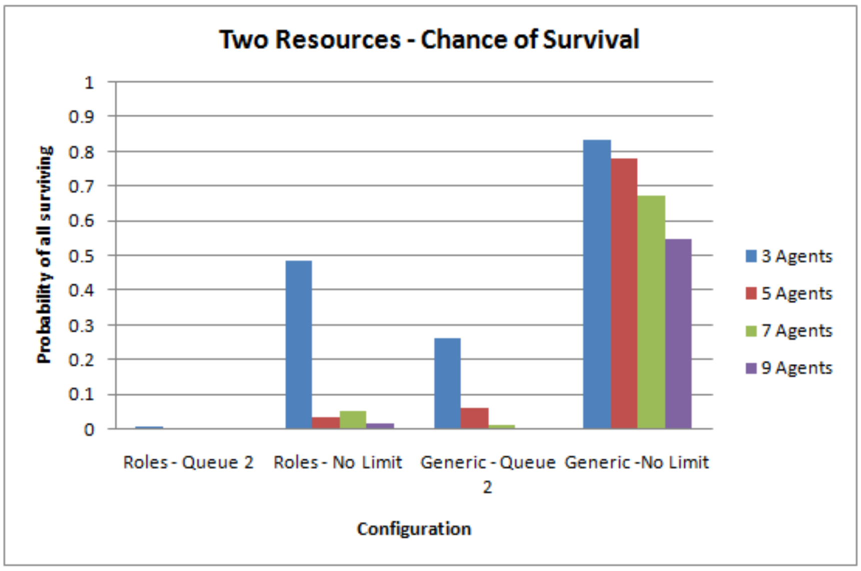


Fig. 4.9: Two Resources - 1xFood - Probability of Survival

by the high average agent lifetime and the reduction of the effect of group size when compared to the role game.

Figure 4.10 shows the profile of the effect of replenishment values on the performance of the various configurations. The minimum value needed for food replenishment in order for the role-based agents to survive is 9.5. In this game, the complexity is such that the GP algorithm does not find solutions where all agents survive until food replenishment is 13 for the unrestricted (3 agents 3/20 trials) and 15 for the restricted (3 agents 1/20 trials). The unrestricted Role solution required a value of 19 for food replenishment before a solution was found where all agents survived in every trial. The restricted setup is unable to find such a solution in the twenty trials conducted.

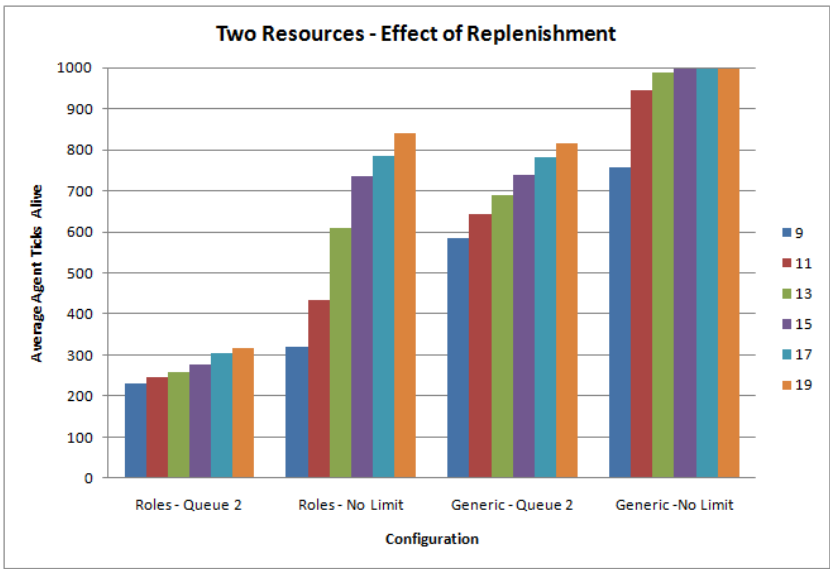


Fig. 4.10: Two Resources - 1xFood - Replenishment Values

4.4.1 Varying the Multiple of Food

In this section, the effect of varying the conversion rate with which refined resources are converted into units of food is explored. As many agents are needed to create food, it may be easier for the agents to survive when the food is split into multiple pieces, rather than a single unit that only one agent can benefit from. Having the refined resources converted into food at higher multiples should make cooperation easier for the group.

A unit of each refined resource is needed to created the food. A single unit of each resource is combined and then multiplied by the food production multiplier to create the unit of food. In this case, experiments will be conducted with a multiplier of two and a multiplier of three. For instance, in the two-resource game and with a multiplier of three in place, combining a unit of both refined resources results in three units of food. The replenishment value for the food still works the same, that is, when an agent consumes a unit of food they receive health to the value of the replenishment amount up to a maximum of 100.

The following parameters were explored for an unlimited role-based game:

- 0 starting stock
- Group size 3, 5, 7 and 9
- Food multiplier 1x with replenishment 6, 12, 18, 24, 30
- Food multiplier 2x with replenishment 3, 6, 9, 12, 15
- Food multiplier 3x with replenishment 2, 4, 8, 8, 10
- 1000 ticks

The best individual is taken from the final generation of each of the twenty evolutionary runs conducted for each configuration. The the average performance of these best individuals is used for comparison between the configurations.

The food replenishment figures were chosen in order to maintain equal total replenishment across multipliers while examining the benefits of the food sharing between the agents. This allows for the direct comparison of the configurations and an examination of whether the quantity of food units is more important than the value of food.

A summary of the results is as follows. When the food multiplier is 2x and 3x, there is a 8% and 9% percent increase respectively, to the agent average lifetime across all parameters (Figure 4.11). For the 2x multiplier, this results in a 60% increase in group survival over the 1x multiplier across all parameters. In the 3x scenario, there is an increase of 57% in group survival over the 1x multiplier. The survival figures can be seen for group size in Figure 4.12.

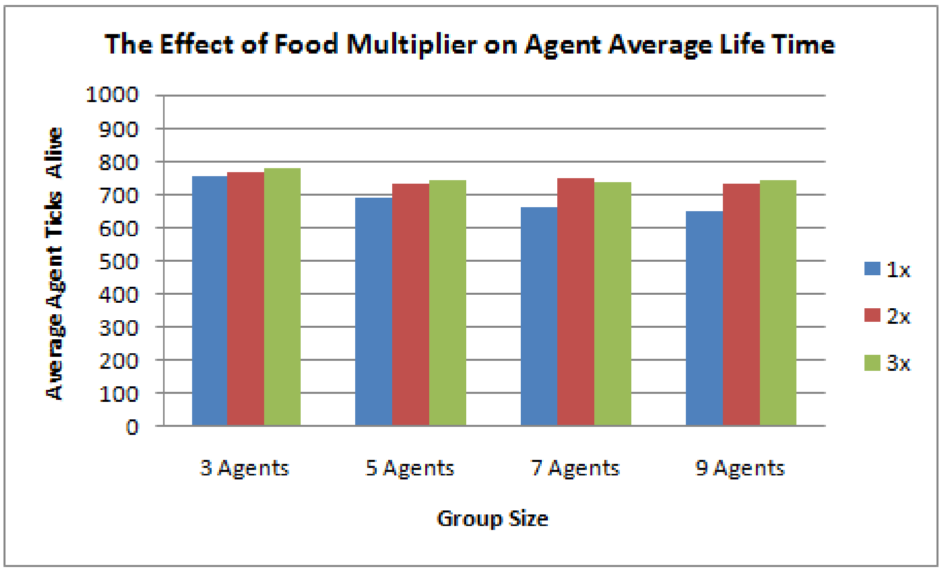


Fig. 4.11: Roles - Unlimited Two Resources - Effect of Food Multipliers on Agent Lifetimes

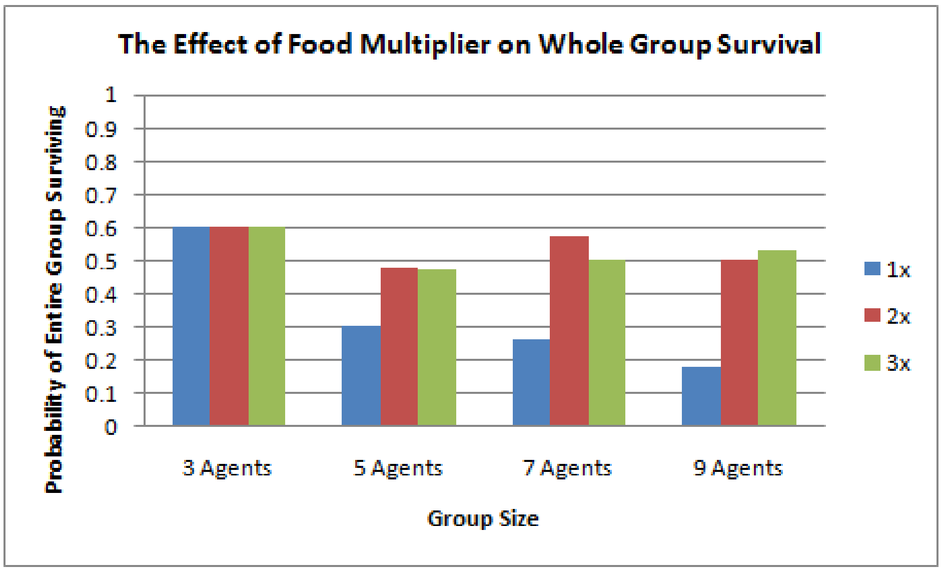


Fig. 4.12: Roles - Unlimited Two Resources - Effect of Food Multipliers on Group Survival

These experiments demonstrate that the easier it is to share resources, the better larger groups can perform. By increasing the number of units of food produced from the refined raw resources, the survival rate for larger groups is on a par with smaller ones, without increasing the total replenishment. The GP algorithm can consistently create small groups able to share low volumes of expensive food but requires a larger volume of less valuable food to achieve success with larger groups.

Some additional parameters are explored through experiments detailed in Appendix A.

4.5 Behavioural Differences in a Static Environment

In this section, the actual behaviours generated are discussed. So far in this chapter, the performance of the GP algorithm and its potential for finding solutions under certain parameters has been discussed. Now a sample of the generated solutions are used for comparison to ascertain if there are any behavioural differences between evolved groups in a static environment.

New solutions were evolved for a set of parameters, for which it had been determined from previous sections that all agents in the group could survive. The chosen parameters were ones which were deemed most influential on the performance of the GP algorithm in finding solutions. The parameters of the games were as follows:

- Zero starting stock
- Group sizes of 3, 5, 7 and 9
- Food multiplier and replenishment of 1x 18, 2x 9 and 3x 6

- Game length of 1000 ticks
- Configurations: Roles and Generic

Solutions are generated using the GP parameters specified in Table 4.4. The best individual was taken at generation 100 and used as a representative of that evolutionary run. An extra constraint was added whereby the entire group had to survive for 1000 ticks. If the group did not survive, another iteration of this process was undertaken. In most cases, the parameters chosen allowed the groups to survive in the first attempt but for certain configurations of the game many repetitions were needed.

For example, the role-based game with a food multiplier of 1 and a replenishment of 18 took 1 iteration for 3 agents, 10 iterations for 5 agents, 6 iterations for 7 agents and 50 iterations for 9 agents. Most other configurations took 1 or 2 iterations. Without seeing the behaviours, this information could inform the selection of parameters if the method is being applied in time restricted settings. There may be a trade off between environment difficulty or behavioural properties and the execution time to generate solutions.

The tick by tick performances of the resulting groups were then analysed and compared. The results were not very dramatic but the performances did display some variance in approaches to the game. Some groups were much more biased to one resource and would build up a large stock while others generated the resources equally. Some groups were able to maintain a high average agent health and others saw a gradual decline in the average health over the course of the game.

For each of the solutions where the entire group survives 1000 ticks, the behaviours are then used in games that last for 50000 ticks. This tests the group's ability to survive in an environment differing to that in which they were evolved, albeit only in terms of longevity. It was hypothesised that for the evaluations, that if a group survived for 1000 ticks it would survive indefinitely, i.e, if no outside influences acted on the group or environment the group would maintain its existence. By testing the group in an environment whose length is much longer than the evaluation period, the evaluation period can be verified and the performances of the groups over longer periods can be compared.

If a group was able to last for 50000 ticks it is presumed to last indefinitely, although it is not impossible for the group to die out. For the twenty four solutions evolved, half lasted for the full 50000 ticks. The same amount of solutions survived the 50000 ticks for both the Role and Generic configurations. Groups lasting for the full amount of ticks also tend to have a behaviour that is relatively the same, that is, they are able to maintain a high average agent health.

The role-based games in which the group did not survive to 50000 were selected to give a demonstration of the behaviour of the group during the game for two reasons. The first is that these are the games in which there may be a noticeable difference in average agent health during the course of a normal 1000 tick game. Secondly, is to ascertain whether or not it is possible to tell, from the 1000 tick profile, if the group will indeed die out.

From Figure 4.13, it can be seen that in some cases it is indeed very obvious that the group is on the downward trajectory by simply plotting the average agent health on a per tick basis. However, not all games offer an immediate indication that they would not simply persist. The average of the average agent health per tick at the end of the game could be a useful indicator. For example, for the role-based games, achieving a final average health of greater than 83 after 1000 ticks meant that 6 out of 7 solutions which managed this, survived for 50000 ticks. In order to obtain a fully predictive metric, further experimentation would be necessary.

Although these runs are just a single point in the solution space, it is clear that a range of behaviours is possible even in the static environment. Figure

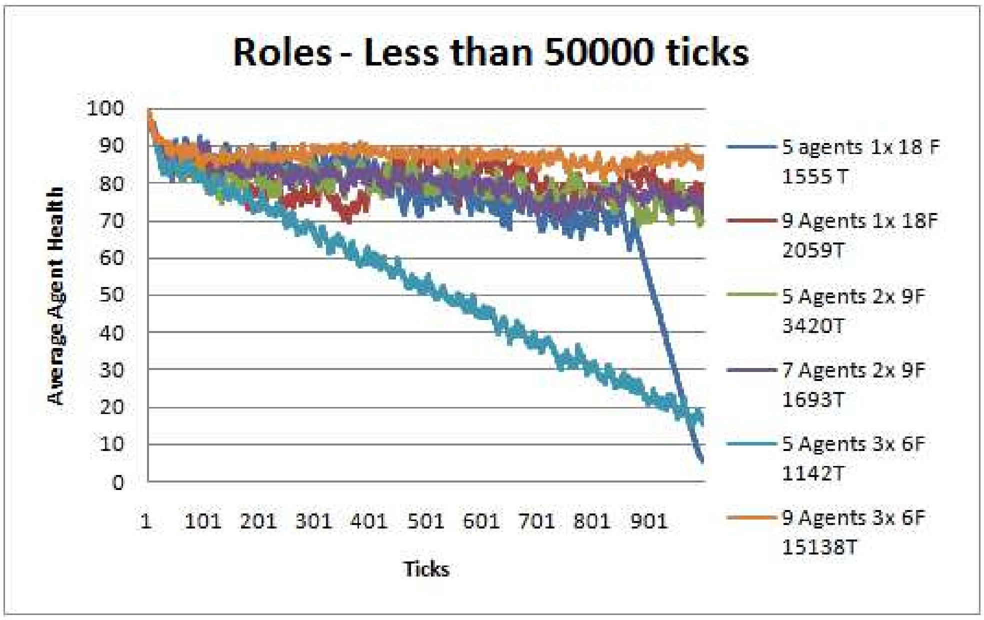


Fig. 4.13: Role-Based Configurations lasting less than 50000 Ticks

4.13 also highlights some of the short comings of the evolutionary evaluation. When solutions are evolved in one environment and put into a differing environment it may have unexpected results. If these behaviours were used in a game, an entire group of people would no longer exist leaving behind a ghost town. This could be both, frustrating to a game creator or story writer and an interesting and naturally occurring feature of life in groups.

Throughout the chapter, a range of game parameters was examined and the GP algorithm's ability to create group behaviours for those parameters presented. In this section, an introduction to post-evolutionary evaluation was provided. By making extra assertions about the behaviours of the solutions created, e.g, all members of the group must survive 1000 ticks and some of group should last for 50000 ticks, the quality of the solutions can be verified automatically. By only including this check when selecting candidate solutions, the cost of performing the evaluation on each member of the genetic population can be avoided. This feature was applied here when selecting representative solutions after a fixed number of generations but it could be incorporated into the stopping conditions of the evolutionary process. Extra checks on the quality of solutions is an important feature required when applying these techniques in video games.

For this game environment, slight differences occur in the evolved group behaviours. However, the majority of solutions tested here for extra longevity have no obvious differences when compared on a tick by tick basis. The game environment and action sets for the agents are too simple to facilitate very diverse approaches to survival. The agents only have one goal: survival for as long as possible. The environment has only a limited number of ways of facilitating this: cooperate and make food or don't and hope to live off the efforts of others. And yet, even in this simple environment, for solutions of similar fitness and with the ability to maintain a group for the evaluation length, some diversity of approaches remains.

4.6 Chapter Summary

In this chapter, an environment has been introduced which encapsulates the notions of cooperation and coordination through a resource gathering and survival mechanism. Various configurations of the environment were introduced, ranging from the simplest small group with a single resource, to a more complex environment with a large group and two resources.

The Genetic Programming algorithm used to find behaviours for the groups was introduced. The algorithm consists of a cooperative solution which evolves a single tree for the group with each member of the group getting their own distinct subtree. The fitness of the solution is then the success of the group at surviving in the environment. A role-based paradigm is used to restrict the available actions to agents in the game and to enforce reliance on other members of the group.

A thorough examination of the game parameters was conducted and a range suitable for creating sustainable groups was demonstrated. A brief investigation was conducted into the generated behaviours of groups capable of survival. Solutions that can survive for the evolutionary evaluation period do not, necessarily, perpetually survive even though resources are infinite. The GP algorithm is, however, able to create solutions for groups in this environment.

This chapter explored the hypothesis that we can specify a class of game which captures cooperation and coordination and using evolutionary techniques find solutions for them. An abstract environment based on a computer game world was presented. The abstract environment is inhabited by agents who have to coordinate to adopt roles and cooperate in generating a shared resource in order to survive. A method for generating behaviours automatically for groups in this environment was shown to be successful for specific sets of parameters. Even though the environment and agents are simple, a range of diverse group behaviours can be generated.

A foundational set of game parameters, for which GP can create surviving group behaviours in a static environment, are demonstrated. In Chapter 5, this work is expanded to introduce simulated dynamic interactions that an AI character may encounter in a computer game. Chapter 6 applies the behaviours created in the abstract game to a simulated continuous computer game environment.

5. EVOLVING GROUPS IN DYNAMIC CPR GAME ENVIRONMENTS

In this chapter, the game discussed in Chapter 4 is expanded upon by introducing more aspects of computer game environments. Dynamic elements are introduced into the environment to increase the complexity of the world. The dynamic elements will help bring the defined abstract environment closer to the complexity of a computer game environment.

The goal of this chapter is to introduce the types of dynamic interactions that a group in a computer game may experience into the abstract game environment. This will allow the verification of the hypothesis **H4** put forward in Section 1.4 that states that the introduction of dynamic elements into the evolutionary process allows for the creation of more robust behaviours across a range of changing environments.

Firstly, dynamic elements from computer games are discussed with a selection being chosen to be applied to the abstract game. Secondly, the impact on the performance of the GP algorithm is shown, as well as what effect this dynamic world has on the behaviours of the group. A round robin tournament is conducted which compares the various behaviours created in different environments in their own environment and across the set of dynamic environments. Finally the chapter is summarised providing a discussion of the results of the experiments.[1]

5.1 Dynamic Environments

Part of what makes the computer game domain so compelling is the immersion in a world where your character can change the world and where the world changes around your character. If a game is truly dynamic, it allows a player to play through it multiple times without losing its entertainment appeal. These dynamic environments are however, much more complicated and present a greater challenge for creating AI solutions.

The ability of the AI characters to interact with player characters, other AI characters and the environment itself can lead to a non-deterministic setting in which to design or script AI behaviours. The aim of this chapter is to introduce elements of a computer game's dynamism into the resource game that was previously explored.

In this section, dynamic elements from computer games are discussed. A selection of these dynamic elements are then chosen and applied. The practical application of the dynamic elements is discussed and the experiments to analyse the effects of changing the game environment are outlined.

Firstly, the selection of dynamic changes to the environment is introduced. These will describe their effect on the environment as well as how they relate to computer games. A summary of the performance of the GP algorithm's performance at creating solutions for the groups with the various dynamic elements acting on the environment is then presented. Finally, a demonstration is given of the variance of the group's behaviour when the changing

[1] This chapter is based on the following publication:
Alan Cunningham and Colm O'Riordan, Evolution of Stable Societies and Social Structures for Computer Games: An Analysis of Dynamic Environments, 19th Irish Artificial Intelligence and Cognitive Science Conference (AICS), 2008

environments are introduced.

5.1.1 Dynamic Elements

For AI characters in computer games, dynamic elements of the game world exist in several forms. They include interactions with the player character and other non-player characters, interactions with changing resources and navigating in a world whose topology and layout can change.

For the purposes of the work in this chapter, changing the layout or topology of the game world will not be addressed. In the abstract implementation of the game, navigation is one factor which is not dealt with when developing AI behaviours. When using the evolved behaviours in an application, navigation behaviours would have to be created specifically for that environment.

The most direct way AI characters experience dynamic elements in a game environment is through interactions with the player character or other NPCs. Depending on the amount of freedom given to a player and the type of game involved, the AI character may not survive an interaction should the player choose to be hostile. In games where the particular AI character is seen as central to the story, it is a common convention to disable the player's ability to attack that particular character.

An AI character must be able to recover from attack where they have incurred an irregular loss to their health. If an AI character dies and they are a part of a group or team, it is important, from a gameplay point of view, that the group can continue. What this means for the abstract resource game is that the group should be able to absorb the extra drain on food caused by attacks on group members. It also means that the group should be able to continue to survive even if members have been killed.

In order to cope with attacks, a likely strategy is to stockpile the food resource until needed by an attack victim. This extra cost to the group is slightly different than raising the costs for actions or reducing replenishment for food as the entire group shares a potentially infrequent cost. This will require the agents to have a specific trait for their approach to consuming food. The agents should only consume the units of food when necessary if they are to create a stockpile of the resource. The GP process will be tasked with creating very specific decision making qualities amongst generating the group behaviours.

A strategy to deal with the loss of a group member is to include redundancy for each of the roles required in the group to make food. Depending on the group size, redundancy is not always possible. The restriction of not being able to change roles throughout the game on agents also is a factor in the difficulty of dealing with a change in the group structure. Creating specific allocations of roles within the group is the challenge to the GP process in this case. The GP process must be able to create group structures containing redundancy that survive evolutionary processes.

Another type of dynamism that an AI character can experience is that of changes to the resources that the agent is interacting with. With regards to the specific case of the abstract role-based game, if other agents or player characters were also able to manipulate the created resources, there may be a change in the group behaviours. For instance, certain behaviours may no longer be effective if a player was purchasing stock from a refinery instead of having that resource available to create food. Outside influences on the stocks, in the form of trading, is a likely outcome of having a resources in a game. In order for player to receive the full entertainment benefit of witnessing a group create and refine resources throughout a game they should be able to partake in the process.

Some computer games have trading features as a main part of the gameplay, for example Elite [Braben and Bell, 1984]. In some games, resource trading can be carried out without an effect on the inhabitants of the game. In the case of the abstract resource game, changes to the resource availability could lead to the destruction of the agents within that group as the rate of production of the food may be changed. The GP process will be required to create group behaviours that can cope with the changes in availability of food. This could mean that the production of food is no longer sequential, as an outside agent may have taken the resource. The group behaviours should contain individuals capable of adjusting to new knowledge as it becomes known.

5.2 Experimental Setup

In this section, the methodology for conducting experiments in this chapters is described. The abstract game is the same as the one used in the previous chapter and a description can be found in Section 4.1. The GP algorithm remains unchanged and is detailed in Section 4.2.

An *Interval* is specified at which a dynamic element affects the game environments. The frequency of the *Interval* is defined relative to the group size, i.e., the larger a group the more the group will encounter the dynamic element throughout the game lifetime. The interval at which a dynamic element is performed is defined as:

$$CurrentTick \bmod (GameLength/GroupSize) = 0 \tag{5.1}$$

The word *Interval* is used to describe the dynamic elements that occur at the variable frequency defined in Equation 5.1. Some dynamic elements are affect the environment at specific ticks and are clearly labelled the their frequencies.

The following game configurations are used for all the dynamic experiments:

Parameter	Value
Population Size	250
Number of generations	100
Creation type	Ramped half and half
Creation Probability	0.02
Crossover Probability	0.90
Swap mutation Probability	0.10
Maximum depth for creation	6
Maximum depth for crossover	17
Elitism enforced	1

Tab. 5.1: Parameters for GP

- Group Size 3, 5, 7, 9
- Food multiplier and replenishment of 1x 24, 2x 12 and 3x 8
- Game Length 1000 Ticks
- Game Iterations 10

The value of food has been determined based on the performance of the GP algorithm for experiments in static environments previously in Chapter 4. A number of *Game Iterations* are used to better approximate the performance of the behaviours. *Game Iterations* are needed as dynamic game elements may have random components to them and as such, the performance of the group could vary. The average performance over ten game iterations is determined to be enough to rank the behaviours within a given dynamic environment while still being low enough to feasibly perform evolutionary computation.

For each configuration combination, twenty separate evolutionary runs are conducted using the evolutionary parameters detailed in Table 5.1. For each of the twenty runs, the evolutionary individual with the best fitness at the final generation is chosen as a representative of that run. The results of the group's behaviour in each dynamic environment is based on the average performance of these twenty individuals.

The two parameters used to compare results for each of the trials are average agent lifetime and survival rate. The average agent lifetime is the total number of ticks agents in a group are alive for, normalised by the group size. The survival rate is the number of games where all agents in a group survives divided by the total number of games played. For both of these parameters, the figures used are the average results over the number of evaluation iterations, in this case, ten game iterations.

5.3 Health Dynamic Environments

The first of the dynamic elements introduced to the game are simulated attacks on, and gifts to, the AI characters. Attacks on, and gifts to, an agent serve to represent scenarios where the AI characters come into contact with other characters and their health is affected, but the AI characters survive. When an agent comes under attack, they incur an unexpected loss to health and conversely, when an agent receives a gift it is modelled as an unexpected increase to the agent's health. The interactions with other characters are abstracted into a single tick and the net outcome at the completion of the tick is seen as the effect on the agent or the agent's environment.

In order to explore the affects on the GP performance to create solutions when there are unexpected alterations to agents' health, a set of dynamic elements are chosen. The *Interval* defined in Equation 5.1 is used to schedule the dynamic events throughout the course of the game. The following dynamic elements are added to the environment:

1. *Health 1* - Every *Interval* choose one agent randomly and remove 50% of their health.

2. *Health 2* - Every *Interval* each agent has 50% chance of losing 50% of their health
3. *Health 3* - Every *Interval* one agent gains 50% health up to a maximum of 100
4. *Health 4* - Every *Interval* each agent has 50% chance of gaining 50% of their health up to a maximum of 100

For this set of dynamic elements there are two positive and two negative scenarios, with one affecting only one agent at a time and the other affecting multiple agents. The negative interactions explore the group's ability to deal with adversity. On a small scale can the group support a weakened individual, and on a larger scale can the group sustain itself in spite of the outside pressures? It is believed that these negative interactions should cause a change in the agents behaviour to create more food but not to consume it until an agent really needs to. The reason for choosing a loss of 50% health in the negative scenarios is this ensures that the attack never kills an agent and should leave them with enough time to recover given the right conditions.

The positive interactions should promote different behaviours. There may not be much change to the group's behaviour when there is a small amount of positive interaction but larger amounts of positivity may lead to a reduction in the amount of food a group creates. In this case, positive interactions could encourage or lead to scenarios where the agents are idle. These behaviours, if applied in a non-dynamic environment, may not retain the ability to survive.

5.3.1 Health Dynamic Environment Results

Firstly, the results of the negative health interactions are discussed. The Negative interactions are ones which periodically remove some health from the agents. The results can be seen in Figure 5.1. Two things become

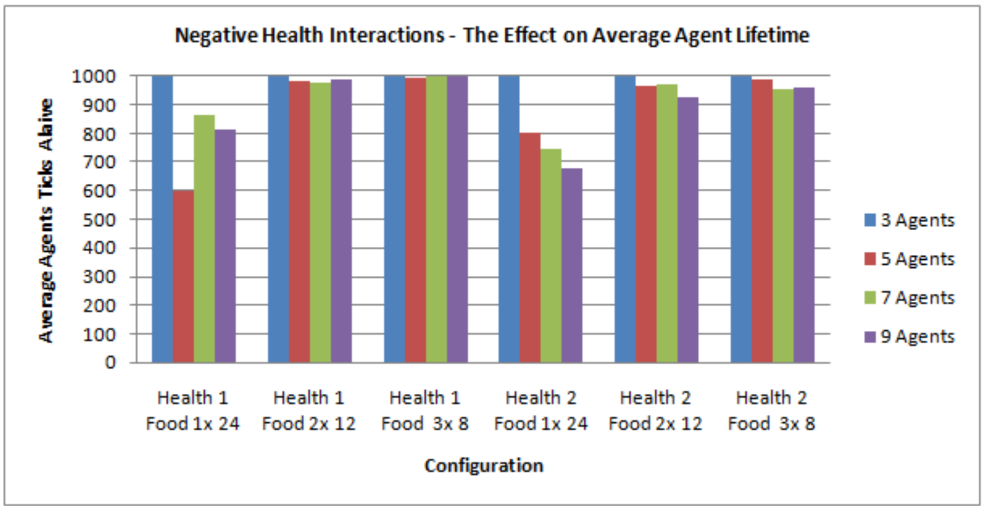


Fig. 5.1: Negative Health Interactions - Average Agent Lifetime

apparent from these results. Firstly, as the food multiplier increases, even though the total value of the food remains the same, the agents' average lifetime increases, by 17% and 18% for the 2x and 3x multipliers respectively over 1x. Secondly, as the group size increases, the agents' average lifetime decreases. In *Health 1*, the average agent lifetime decreases by 6% when moving from 3 agents to 9 agents. In *Health 2*, the average agent lifetime decreases by 14% as the group size increases from 3 agents to 9 agents.

As expected, as the interaction becomes more negative, both the average agent lifetime and the survival rates decrease. The survival rate for the two dynamic environments where agents lose health can be seen in Figure 5.2. When averaged across the two negative environments, the survival increases, by 58% and 64% for the 2x and 3x multipliers respectively when compared to the performance of the 1x multiplier. Overall, the average survival rate for Health 1 and Health 2 is 0.71 and 0.59 respectively. For these environments, the survival rate falls as group size increases and as the environment becomes more negative.

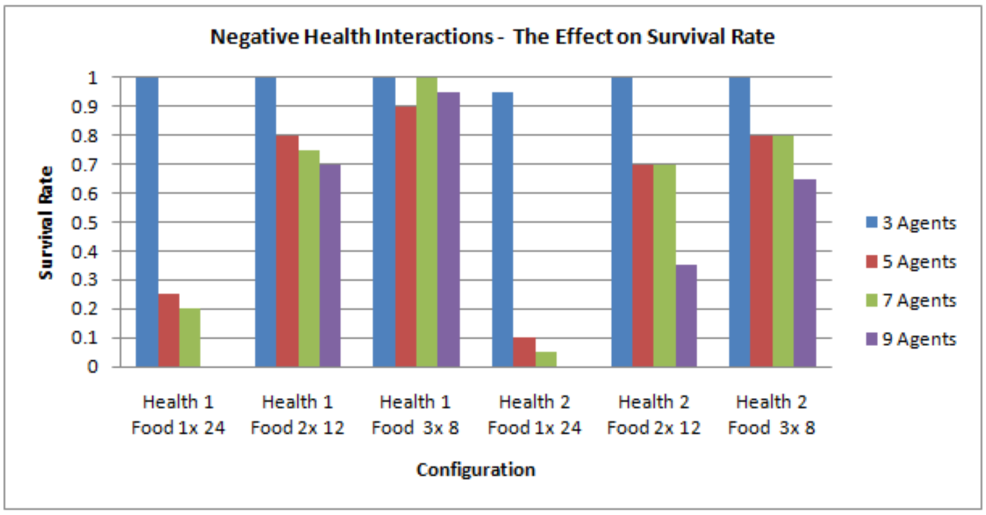


Fig. 5.2: Negative Health Interactions - Survival Rate

The Positive interactions are ones which periodically add some health to the agents. Averaged over all configurations for each positive Health dynamic environment, *Health 3* and *Health 4* had an average agent lifetime of 981 and 988 ticks respectively (maximum 1000 ticks). Similarly to the negative interactions, as the food multiplier increases so does the agent's average lifetime in the positive interactions, but in this case the increase is marginal as all configurations perform well.

The average survival rate is also increased (as expected), when comparing the negative health interactions to the positive ones, up from 0.65 to 0.81 across all configurations. The survival rates for the positive Health interactions can be seen in Figure 5.3. When there are multiple units of food, i.e., when the multiplier is 2x or 3x, the agents survive even in large groups (0.95 survival rate across all group sizes). The 3 agent group size survives all games in every positive configuration. Then the food multiplier is 1x, the larger groups all suffer a reduction in survival rate.

There is no difference between the average survival rate for *Health 3* and

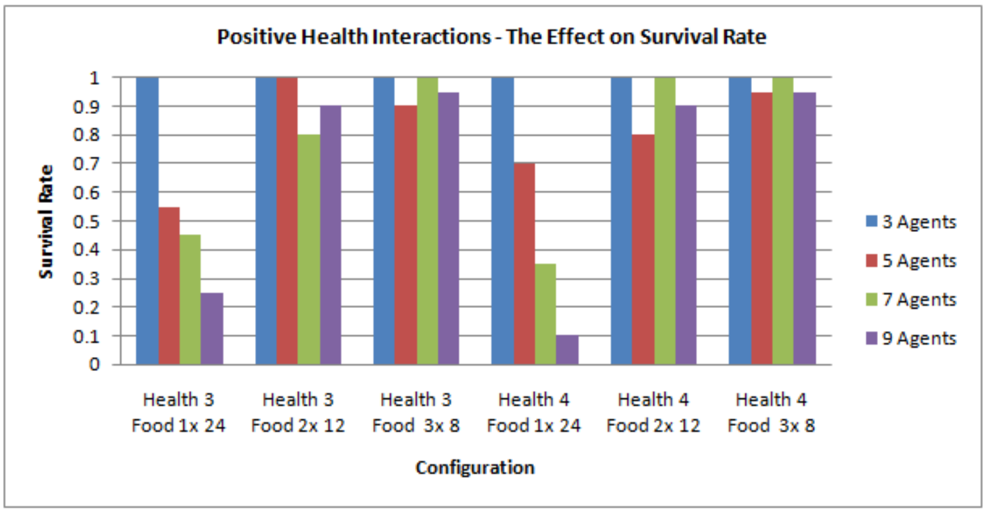


Fig. 5.3: Positive Health Interactions - Survival Rate

Health 4 across all solutions. This means that by making the environment more positive in this way does not necessarily mean that groups will survive at a higher rate.

5.4 Coping with Death

In this section, a set of experiments are conducted which introduce the notion of death to the group caused by an external factor. The aims of this section are as follows:

- Firstly, from a computer game perspective, groups of agents should be robust enough to continue even if an agent is killed. As this may be a common occurrence within a game environment, the entire fate of the group should not rest upon a single agent.
- Secondly, these experiments test the ability of the GP algorithm to evolve to contain built-in redundancy, in terms of roles and actions, for

the agents in the group. It is a test of the feasibility of the structure of the GP solution for creating groups of individuals.

The following dynamic environments are defined to explore the effects of death on the groups:

- *Death 1* - In the first tick, select one agent at random and set their health to zero
- *Death 2* - In tick 500, select one agent at random and set their health to zero

By setting an agent's health to zero it essentially removes them from the game as they no longer participate or perform actions. By setting an agent's health to 0 in the first tick, the agent has no influence on the game whatsoever. The remaining agents must then be able to support the group for the entire game length. Setting the agent's health to zero at the halfway point in the game simulates an interaction during the normal course of the game which causes the agent to die. The agent can contribute for some of the game, however, there are still enough ticks remaining to require redundancy within the group.

Because these experiments deal with the death of an agent, the calculations for average agent lifetime and survival rates require a small change. The survival rate and average agent life of the group are now calculated using $groupsize - 1$. This means that, even though an agent has died, if the rest of the agents survive until the end of the game it counts as a successful survival. The value used for each of the twenty samples is still the result of 10 game iterations. The group must survive through each of the 10 random selections made when choosing an agent to die, in order to count towards the survival rate. This prevents groups surviving with lucky configurations and helps to ensure that true redundancy is present within the group structure.

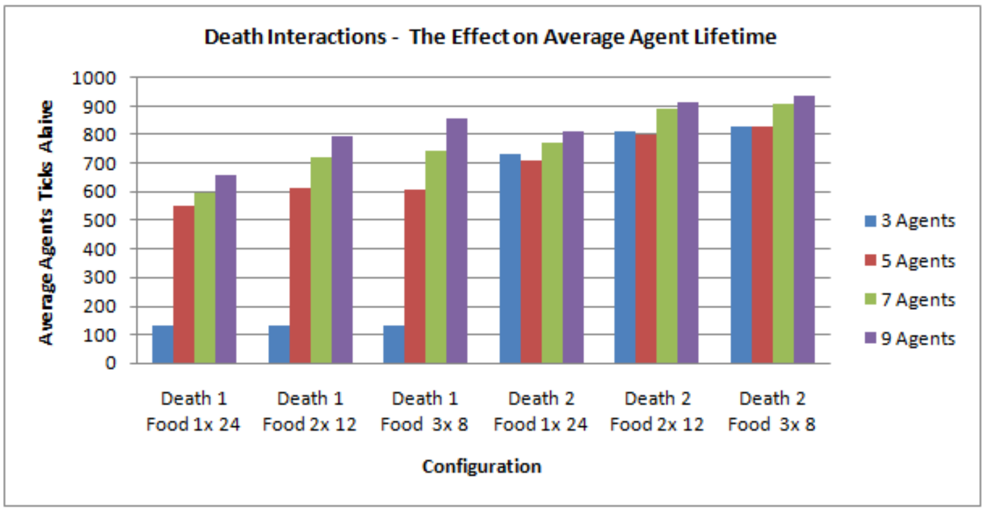


Fig. 5.4: Death Interactions - Average Agent Lifetime

5.4.1 Death Results

In this section, the result of the dynamic environments dealing with death are presented. Firstly, the average agent lifetime is discussed; the results of these experiments can be seen in Figure 5.4. For the Death 1 environments, the group of three agents does poorly and for good reason. It is impossible for the three agent group to survive when one is taken out at the beginning of the game. This is due to the fact that it is a two resource game and all three agents are required to create food.

In both the *Death 1* and *Death 2* environments, in terms of average agent lifetime, generally the larger the group the better. Group sizes 7 and 9 outperform have the highest average agent lifetime. This result can be contributed to a larger group size naturally providing better redundancy of agent roles and actions when a death occurs. For all cases, a larger multiplier for food will improve average agent lifetime.

The results of the survival rates can be seen in Figure 5.5. The first thing

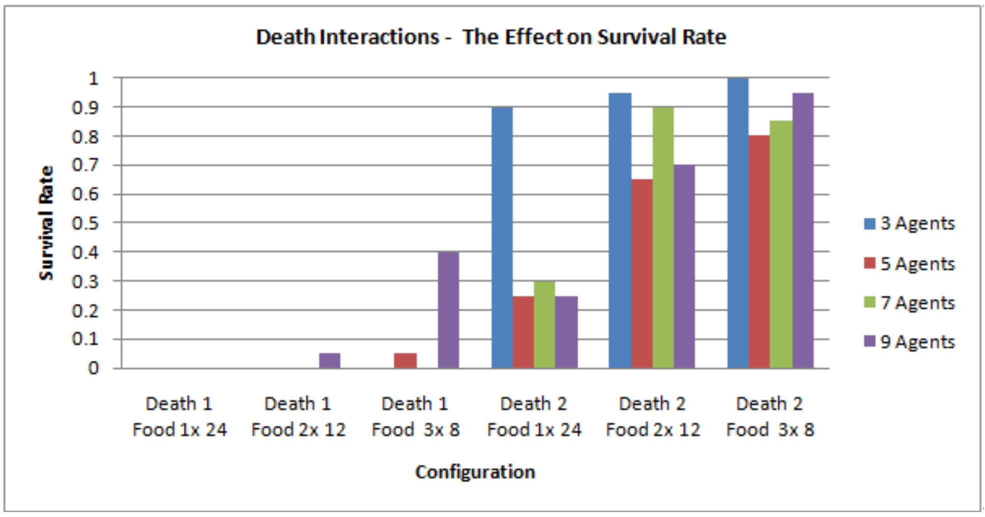


Fig. 5.5: Death Interactions - Survival Rate

that becomes apparent is that the *Death 1* environment is a very difficult one for the groups to survive in. With a 2x multiplier, only the 9 agent group survives and that is only in one sample from twenty. As the multiplier increases to 3x there are more survivals with 0.4 for 9 agents and a 0.05 for 5 agents.

The *Death 2* environment has a much higher survival rate, with 0.7 overall compared to 0.04 in the Death 1 environment. In the Death 2 environment, the 3 agent group has the highest survival rate at 0.95. The ability of agents to contribute for some of the game, coupled with a generous environment allow the remaining two agents to survive even though they no longer can create food. The next best performing group size is 7 agents with a Death 2 overall survival rate of 0.68, followed by 9 agents and 5 agents with 0.63 and 0.57 respectively.

In these environments, larger group sizes have less of a handicap when compared to other results, e.g., those in Section 5.3. The larger groups can absorb the loss of an agent with more ease than smaller ones because, due to their size, they have better redundancy for the required roles, The GP process has shown the ability to create behaviours with the provision of redundancy so that the groups can survive if an agent is removed from the game.

5.5 Outside Traders

In this section, the notion of outside traders are introduced to the abstract game environment and are simulated through manipulations of the resources within the game environment. These resource interactions effectively change the cost of creating resources and as such change the payoffs that the agents encounter. Resource modification can occur in two ways, through implicit or explicit modification. Implicit modification, like the taxation of a resource, could be implemented through a modification of the cost of creating the resource itself, where one resource costs the agent more health to produce than the other.

Explicit modification, as it used in this case, is implemented by changing the actual levels of stock in the game. These interactions simulate other agents or human players either trading or stealing stocks from the group. Altering the stocks of resource in this way allows for a temporary increase in the production costs of the resource.

In these interactions, both the effects of positive (adding extra resources without any cost) and the negative (removing resources that have been created) interactions are explored. The interactions were chosen to represent a range of potential experiences, ranging from infrequent large resource modifications to very frequent small resource modifications. They are outlined as follows

- *Resource 1* - At every *Interval* add 20 to one of the *Keep's* raw materials (chosen at random)

- *Resource 2* - At every *Interval* remove 20 from one of the *Keep's* raw materials (chosen at random)
- *Resource 3* - At every *Interval* either add or remove (50/50 chance) 20 to or from one of the *Keep's* raw materials (chosen at random)
- *Resource 4* - Every 100 ticks either add or remove (50/50 chance) 20 to or from one of the *Keep's* raw materials (chosen at random)
- *Resource 5* - At every *Interval* either add or remove (50/50 chance) 20 to or from both of the *Keep's* raw materials
- *Resource 6* - Every 100 ticks either add or remove (50/50 chance) 20 to or from both of the *Keep's* raw materials
- *Resource 7* - Every 50 ticks either remove 5 from both of the *Keep's* raw materials
- *Resource 8* - Every 50 ticks either remove 5 from one of the *Keep's* raw materials (chosen at random)

The implementation of the *Keep's* stocks is changed slightly to facilitate negative resource interactions. If there is not enough stock when a negative resource interaction occurs, the stock goes into minus inventory. The agent can still add to it but obviously cannot remove stock for their own purposes while there is negative stock levels. The agent cannot put the stock level into negative inventory when removing stocks.

In the actual stocks that are manipulated include the two Keep's raw materials. This is the raw resource that agents have harvested and transported to the keep, but which must be refined there. This is a critical point in the creation of food as a change not only affects the agent harvesting that resource, but also can change all other agents' food production schedules. This can occur when there is no refined resource for the Transporter to bring to the

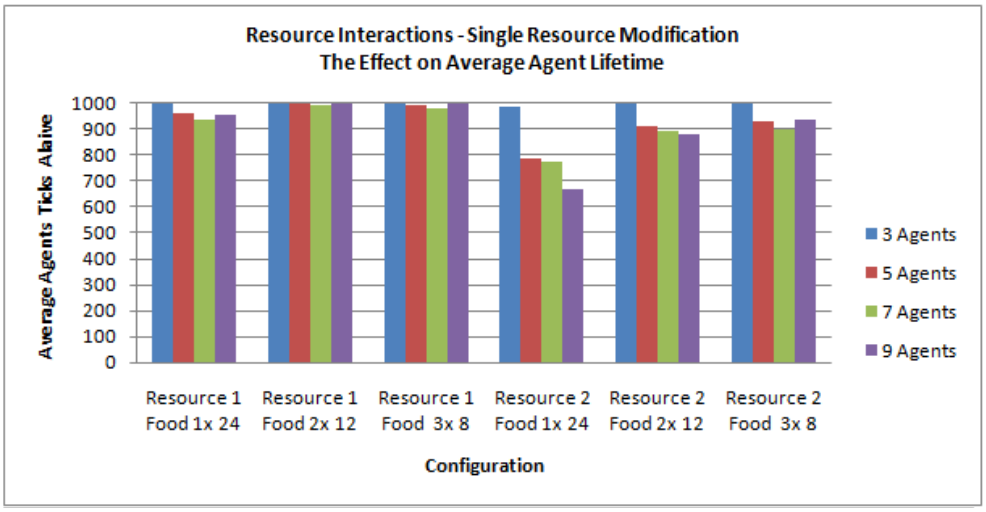


Fig. 5.6: Single Resource Infrequent Interactions - Average Agent Lifetime

Factory and the agents harvesting the other resource can no longer depend on the production of food.

The resource dynamic environments can be broadly grouped into categories of infrequent large manipulation of a single resource, infrequent large manipulation of both resources, frequent large manipulation of the stocks and very frequent small manipulation of the stocks.

5.5.1 Resource Dynamic Environment Results

The results of the resource dynamic environments as presented in this section. The first results discussed are the *Resource 1* and *Resource 2* environments which change a single resource at a time every interval. *Resource 1* adds 20 to a stock while *Resource 2* removes 20 from a stock. The effect on the average agent lifetime can be seen in Figure 5.6. There is a 10% drop in average agent lifetime between the *Resource 1* and *Resource 2* environments.

The effect on survival rate can be seen in Figure 5.7. There is a larger drop

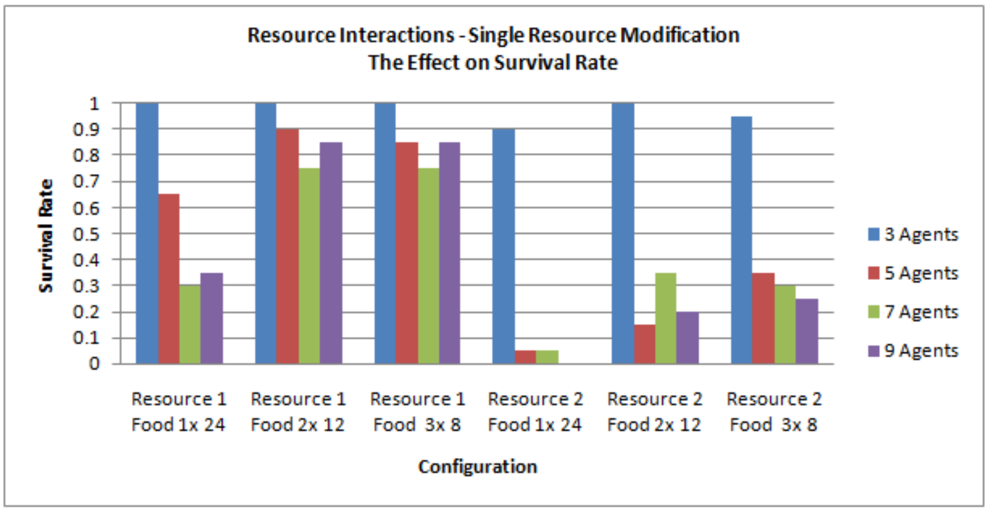


Fig. 5.7: Single Resource Infrequent Interactions - Survival Rate

in survival rate when comparing *Resource 1* and *Resource 2* than in average agent lifetime. The survival rate is 0.77 for *Resource 1* and drops to 0.38 for *Resource 2*. The group size of 3 agents is less effected than the larger groups. The 3 agents group drops from 1 to 0.95 while all other groups drop from 0.69 to 0.19.

Resource 3 and *Resource 4* also manipulate a single stock but can either add to the stock or take away from it. Resource 3 uses the *Interval* and as such, is infrequent, while *Resource 4* manipulates the stock at a fixed frequency of 100 ticks. When comparing the stock manipulation of *Resource 1*, *Resource 2* and *Resource 3*, all of which share the same frequency, the results are as expected. With just positive additions to stock, *Resource 1* is best, with both positive and negative stock manipulation, *Resource 2* is next, and finally *Resource 3*, with only negative stock manipulation performs worst in terms of average agent lifetime. The average agent lifetime for *Resource 3* and *Resource 4* can be seen in Figure 5.8.

Increasing the frequency for the resource dynamic event causes a further

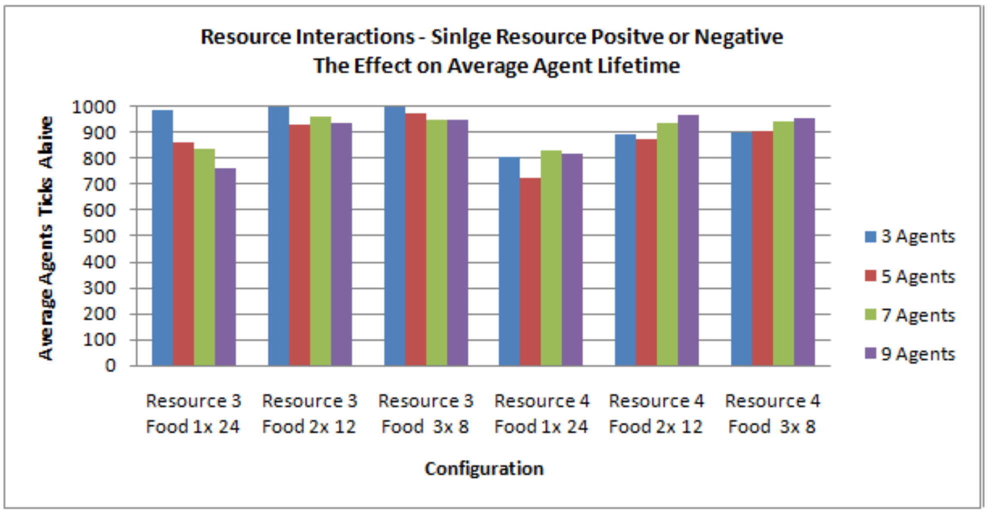


Fig. 5.8: Single Resource Infrequent Mixed Manipulation Interactions - Average Agent Lifetime

decrease in average agent lifetime, falling 5% from *Resource 3* to *Resource 4*. The drop in average agent lifetime is mirrored in survival rate which can be seen in Figure 5.9. Once again, the more frequent the resource manipulation, in this case means that the groups find it more difficult to survive.

In the *Resource 5* and *Resource 6* dynamic experiments, both resources are manipulated either positively or negatively at a certain frequency. The effects on the average agent lifetime can be seen in Figure 5.10. By manipulating both resources the average agent lifetime falls when compared to manipulating just one. The average agent lifetime for *Resource 5* and *Resource 6* is 901 and 812 ticks respectively.

The survival rates for *Resource 5* and *Resource 6* are presented in Figure 5.11. This is a more difficult environment for groups to survive when compared to single resource manipulation. When the resource manipulation takes place more frequently, the larger groups generally perform better. The 3 agent group is unable to survive the *Resource 6* environment in any of the twenty

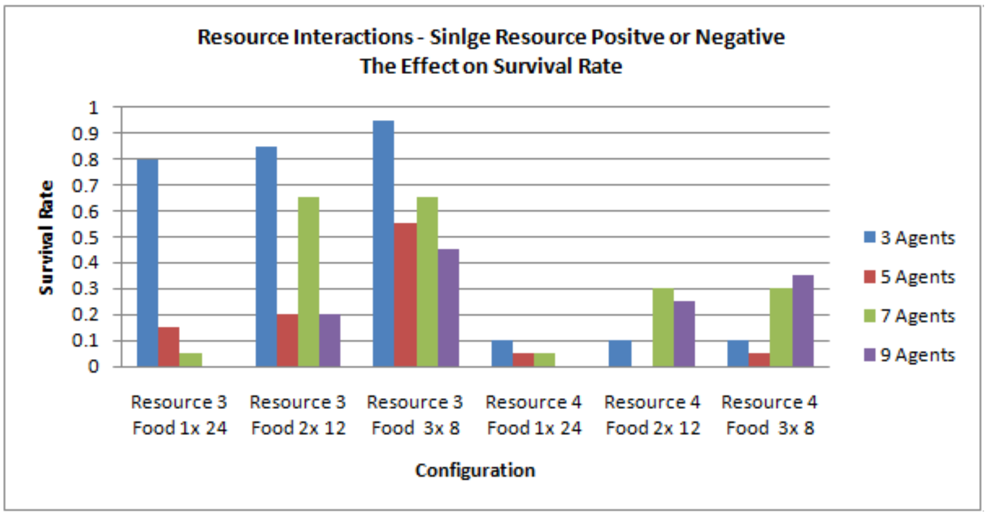


Fig. 5.9: Single Resource Infrequent Mixed Manipulation Interactions - Survival Rate

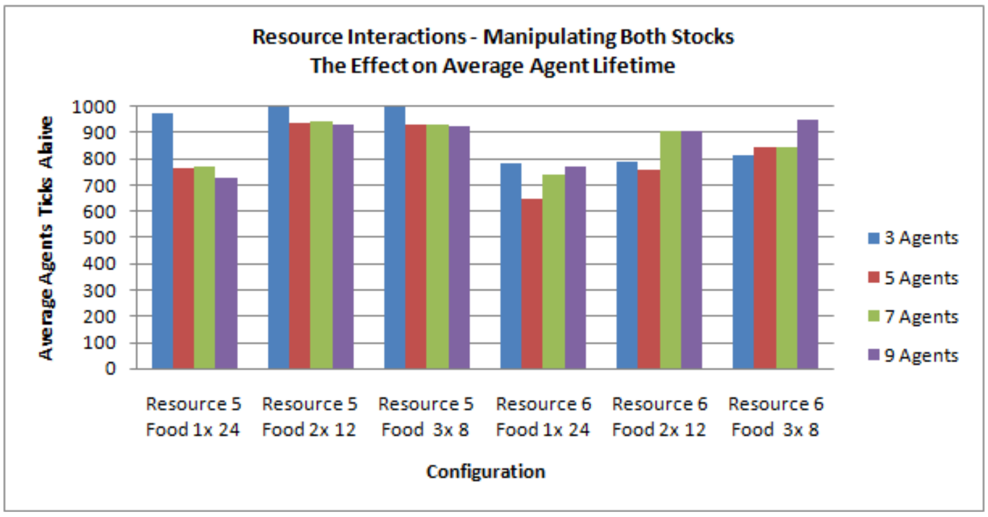


Fig. 5.10: Both Resources Infrequent Mixed Manipulation Interactions - Average Agent Lifetime

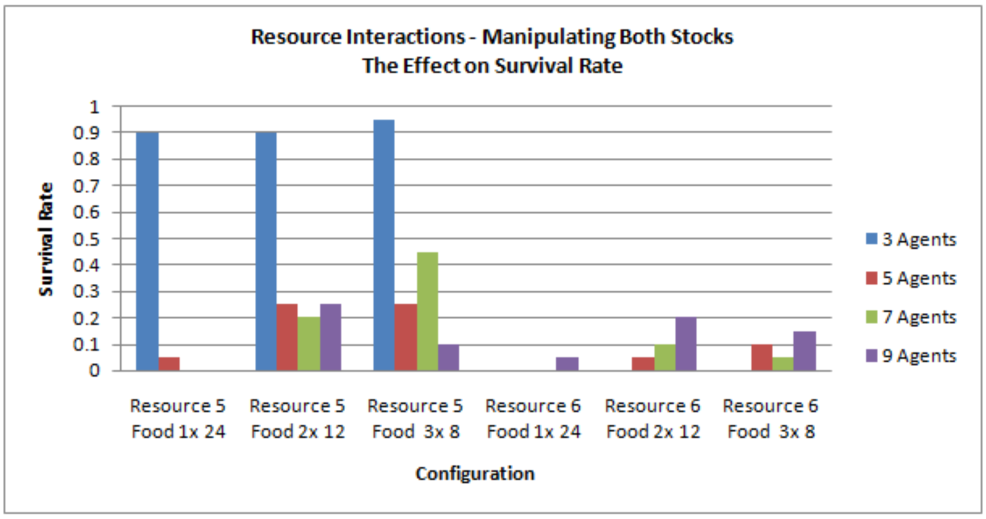


Fig. 5.11: Both Resource Infrequent Mixed Manipulation Interactions - Survival Rate

samples.

The frequency of the resource manipulation is increased for the *Resource 7* and *Resource 8* dynamic environments. Small negative manipulations occur at a fixed rate throughout the game with *Resource 7* modifying both resources and *Resource 8* modifying just one. The average agent lifetime results can be seen Figure 5.12. In both environments, the larger groups perform better overall in terms of average agent lifetime. This can be attributed to their ability to absorb the small fixed manipulation every 50 ticks, the increase in the cost of creating food is smaller when averaged over a larger group.

The survival rates for the *Resource 7* and *Resource 8* environments can be seen in Figure 5.13. As the food multiplier increases, the rate of survival increases. The average survival rate for *Resource 7* and *Resource 8* is 0.28 and 0.46 respectively. There is a 39% increase in survival rate when reducing the modification of two resources to one resource at this frequency.

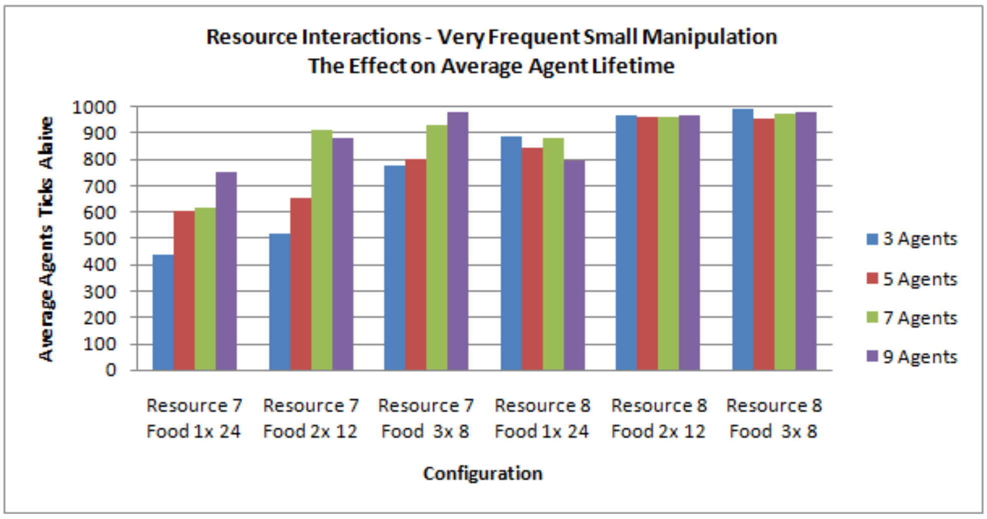


Fig. 5.12: Both Resources Infrequent Mixed Manipulation Interactions - Average Agent Lifetime

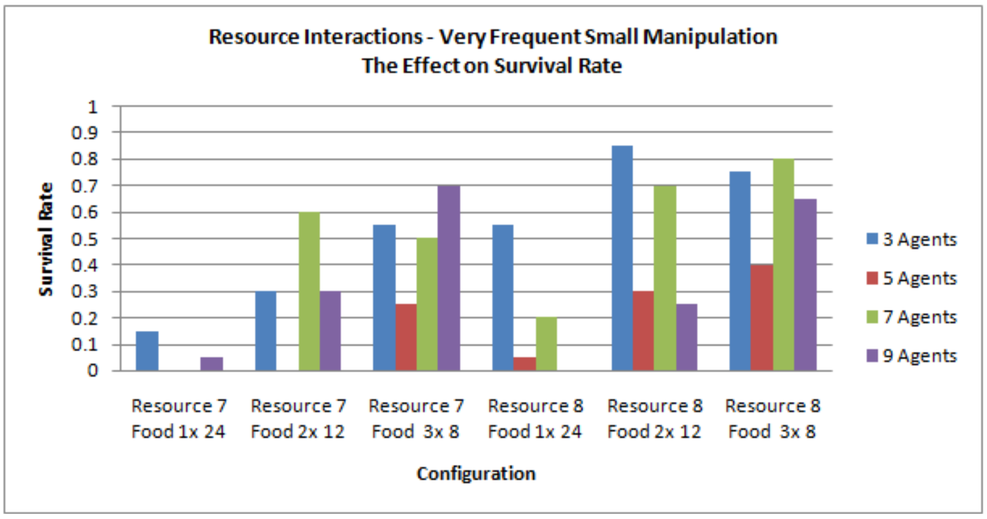


Fig. 5.13: Both Resource Infrequent Mixed Manipulation Interactions - Survival Rate

For comparison, Table 5.2 displays the average agent lifetime and average survival rates for each of the configurations and includes the results for static evolved behaviours evaluated in a static environment. As expected, the environments with positive manipulation do better than the static environments. Negative manipulation leads to poorer performances when compared to those of the static behaviours. Importantly, for each of the configurations, the GP had the ability to create groups that could cope with and survive the dynamic environments.

5.5.2 Summary

In this section, the results of the experiments conducted using various interactions are discussed. In the previous sections, several types of dynamic interactions were defined. These interactions were inspired by potential experiences AI characters may have when they are subjected to outside influences in a computer game.

The interactions included direct modification of the agents' health, death of agents within the group and modifications of the resources that the agents are working with. These interactions happen with both negative and positive effects from an agent's point of view. They also happen at varying frequencies throughout the game duration.

The GP algorithm is shown to be able to create groups which can survive

	Static	R. 1	R. 2	R. 3	R. 4	R. 5	R. 6	R. 7	R. 8
L:	970	985	889	927	877	902	812	737	930
S:	0.74	0.77	0.38	0.46	0.14	0.36	0.06	0.28	0.46

Tab. 5.2: Comparisons of the Resource Interactions - where L: is the average agent lifetime in ticks; S: is the average survival rate and R. 1 is the Resource 1 dynamic environment.

these environments which have added external pressures. Some negative environments, like the ones where an agent gets killed in the first tick, affect the performance of the group to a large extent. In others, the group is able to absorb the change to the resource or to the health of the agents without much impact. Some positive interactions, where resources get added to, improve the lifetime and survival rates of the agents.

For this work, the ability to recreate an event from a computer game in an abstract form and create groups of agents automatically that can survive in the environment, is important. The ability to survive outside influences to the environment gives a degree of confidence to the success of using these behaviours in a computer game. In Chapter 6, this concept will be explored in more detail. In Section 5.6, the behaviours created in the various dynamic environments are evaluated in the complete range of dynamic environments to investigate their applicability across multiple environments.

5.6 Round Robin Experiments

In this section, a round robin tournament is conducted using the dynamic elements introduced previously in the chapter. The aim of this section is to determine the usefulness of behaviours evolved with a specific dynamic game element across the range of dynamic environments.

For each dynamic environment, 14 in total, and for a static environment, evolutionary runs are conducted with the individual with the best fitness selected at the end of generation 100 as a representative of the run. Each representative group behaviour then plays ten games in the static environment as well as ten game in each dynamic environment. An average is taken for each of environments with the survival of the group and the average ticks alive recorded.

For each configuration of the game, twenty separate runs are conducted and the results from each run are averaged to provide an indication of behaviours from each configurations across a range of environments. A selection of behaviours created in a static environment are used for comparison. The dynamic environments are:

- *Resource 1* - At every *Interval* add 20 to one of the *Keep's* raw materials (chosen at random)
- *Resource 2* - At every *Interval* remove 20 from one of the *Keep's* raw materials (chosen at random)
- *Resource 3* - At every *Interval* either add or remove (50/50 chance) 20 to or from one of the *Keep's* raw materials (chosen at random)
- *Resource 4* - Every 100 ticks either add or remove (50/50 chance) 20 to or from one of the *Keep's* raw materials (chosen at random)
- *Resource 5* - At every *Interval* either add or remove (50/50 chance) 20 to or from both of the *Keep's* raw materials
- *Resource 6* - Every 100 ticks either add or remove (50/50 chance) 20 to or from both of the *Keep's* raw materials
- *Resource 7* - Every 50 ticks either remove 5 from both of the *Keep's* raw materials
- *Resource 8* - Every 50 ticks either remove 5 from one of the *Keep's* raw materials (chosen at random)
- *Health 1* - At every *Interval* set one agent's health (chosen at random) to half its current value
- *Health 2* - At every *Interval* 50/50 chance of setting each agent's health to half its current value

- *Health 3* - At every *Interval* set one agent's health (chosen at random) to twice its current value (max 100)
- *Health 4* - At every *Interval* 50/50 chance of setting each agent's health to twice its current value (max 100)
- *Death 1* - In the first tick, select one agent at random and set their health to 0
- *Death 2* - In tick 500, select one agent at random and set their health to 0

The configurations used in this round robin tournament are:

- Group size 3, 5, 7 and 9
- Food multiplier and replenishment of 1x 24, 2x 12 and 3x 8

For each pairing of group size and food multiplier and replenishment, twenty representative group behaviours are created in each of the dynamic environments. The average performance of the twenty behaviours across each environment are used to determine performance. Group size performances are discussed as averages of three separate sets of twenty results for each of the food configurations, e.g., 3 agents would have twenty runs in each 1x 24, 2x 12 and 3x 8 for each environment. Food configurations are averaged across all group sizes. The results are presented in the following subsection.

5.6.1 Results

In this subsection, the results of the round robin tournament for behaviours created in dynamic environments are presented. The average agent lifetime and group survival rate are used to compare results. The average agent lifetime is the total number of ticks all agents were alive for in the group divided by group size. The group survival rate is the number of games where all agents survived the entire game divided by the number of samples, in this case, twenty.

Averaged across all configurations, behaviours evolved in the *Resource 4* dynamic environment do best on average across all dynamic environments in terms of both average agent lifetime and group survival, 866.77 tick and 0.37 survival respectively. A reason for the robustness of the behaviours from this environment is that it is one of the more changeable environments.

The resources are modified often, every 100 ticks, with a possibility of a large reduction or addition to one of the resources each time. *Resource 6* is a more harsh environment with both resources being modified each time and *Resource 3* alters the resources in the same way as *Resource 4* but at a less frequent interval. Both of these environments perform better than average but have a 5% and 3% drop in average agent lifetime and 18% and 2% drop in survival rates respectively. Learning to cope in this environment has created behaviours that are successful across a range of varying environments.

The environment in which all behaviours had, on average, both the highest average agent lifetime and group survival was the *Health 4* environment with 919.57 ticks and 0.59 survival respectively. The environment in which the behaviours performed the worst was *Death 1* environment with an average agent lifetime of 462.97 ticks and an average survivability of zero.

The *Health 4* environment is the most generous environment as it potentially rewards multiple agents with extra health throughout the game. This leads behaviours from all environments doing well with this dynamic element with 12 out of 14 of the sets having their highest average agent lifetime scores with *Health 4* as the environment.

The *Death 1* environment is the most difficult for the GP algorithm to create solutions for. Solutions created in this environment perform the worst, on average, across all environments. All solutions have their lowest scores for average agent lifetime and survival rates in this environment. A random member of the group is killed at the start of each game which makes it hard for the GP algorithm to provide solutions in this environment. Other behaviours, having never experienced this before, may no longer have key members contributing.

When compared to the behaviours created in a static environment, 12 out of 14 dynamic behaviours have better average agent lifetime scores and 10 out of 14 have better survivability scores when averaged across all environments. For the 12 dynamic environments that were better that the static environment, the average increase was 7% in average agent lifetime with a maximum increase of 15%. For the 10 dynamic environments that were better than static in terms of survivability, the average improvement was 17% with a maximum improvement of 33%. Including the two omitted behaviours from the survivability results to use the same 12 behaviours as the lifetime scores, the average survival rate increase drops to 13%.

Only two out of 14 of the dynamic behaviours had a better average agent lifetime than the static behaviours when evaluated in the static environment. The *Resource 4* dynamic behaviours have the highest average agent lifetime in four out of the 15 test environments with the *Health 4* dynamic behaviours having the highest in three. In terms of survivability, the *Health 3* dynamic behaviours have the highest rate in three environments and the *Resource 4* dynamic behaviours have the best in two.

In terms of both average agent lifetime and survival rate, 12 out of 14 of the dynamic behaviours had the highest rate for the environments they were evolved in. The 12 dynamic behaviours that were best in their environment were not the same 12 that had a better than static behaviour overall average agent lifetime result. The static behaviours were best in none of the trials performed.

If the environment that a set of behaviours was evolved in is removed from the average overall results, the dynamic behaviours performance increases over those of the static behaviours. That is, if the behaviours performance is only measured in environments with which they are unfamiliar, the average improvement of dynamic behaviours over the static behaviours increases from 2% (max. 33%) to 10% (max. 58%) in terms of survival rate. By contrast, the difference between the dynamic and static behaviours in average agent lifetime increases from 4% to 5%.

In Figure 5.14, the survival rate for the arranged by configuration can be seen. The three configurations of food multiplier and replenishment and the four configurations of group size are included. The *Food 3x 8* configuration has the highest survival rate of 0.37 which is 61% higher than the *Food 1x 24* rate of 0.14. The 3 Agents group size has the best survival rate at 0.44 which is 52% better than the worst performing 9 Agent configuration which has a rate of 0.23.

When combining the survival rate results of the two Death environments, the 9 Agent configuration performs better than all other group sizes. This occurs, presumably, because the more agents there are in a group, the more likely it is for the group the have redundancy in the roles and actions and thus increase the survival of the group. The overall average agent lifetime is present in Appendix B.

5.6.2 Round Robin Summary

In this section, through the use of a round robin tournament, the performance of various abstract game configurations have been assessed and ranked, as well as, exploring the GP algorithm's usefulness to create solutions fit for dynamic environments.

If the group behaviours are likely to encounter dynamic elements in their en-

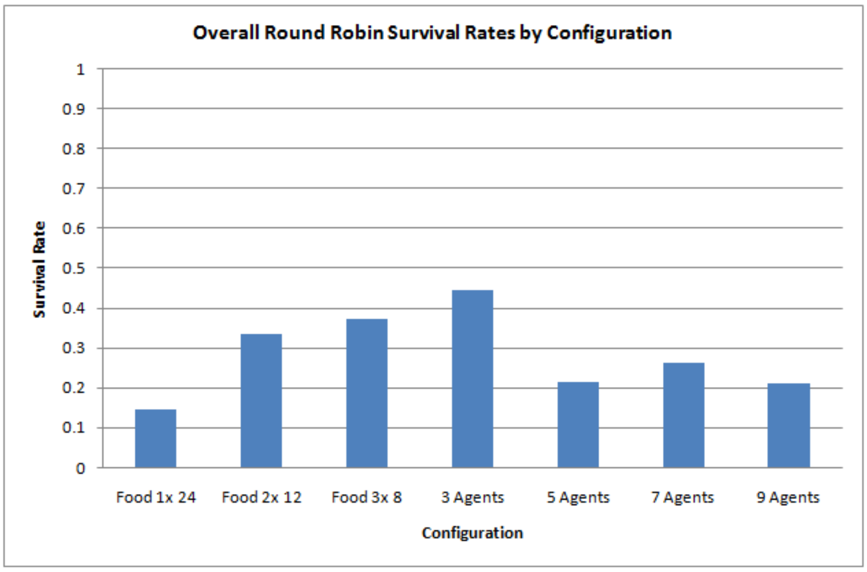


Fig. 5.14: Round Robin Survival Rates by Configuration

vironment, the performances of those behaviours can be increased by evolving in a dynamic environment. The use of a dynamic environment during evolution can increase both the average agent lifetime and the ability for the group to survive across a range of dynamic elements. If a specific dynamic element can be expected, including that in the evolutionary evaluation will improve the performances of the behaviours.

Although behaviours created in static environments have not encountered dynamic elements before, they can have the ability to perform well and survive given the right environment. In general, including dynamic elements in the evolutionary evaluation makes the group behaviours more robust, especially if the dynamic environments have not been encountered in the evolutionary process.

5.7 Chapter Summary

In this chapter, the notion of dynamic environments are introduced to the abstract game environment of Chapter 4. A selection of potential dynamic environments are highlighted from a computer game domain and are simulated within the abstract game environment. This chapter explored the hypothesis that the introduction of dynamic elements into the evolutionary process allows for the creation of more robust behaviours across a range of changing environments.

The dynamic environments are grouped by the way they affect the environment. A selection of *Health* dynamic environments test the group's ability to withstand interactions with outside characters resulting in the unexpected loss or gain in health for agents. The environments that incorporated *Death* test the group's robustness in the face of an agent being killed by external forces. And finally, *Resource* environments assess the ability to implicitly support resource trading within the structure of the game by modelling it as an external modification of the game's stock.

For each of the above environments, the GP algorithm is able to provide solutions where groups perform well and survive for a range of configurations. The performance of the groups is shown as the various dynamic environments affect the game.

Finally, a round robin tournament is conducted to explore the performance of behaviours in environments that are not the same as the one in which they were created. This reveals that, in general, including dynamic elements in the evolutionary evaluation makes the group behaviours more robust, especially if the dynamic environments have not been encountered in the evolutionary process.

6. APPLICATION OF ABSTRACT GENETIC PROGRAMMING SOLUTIONS FOR A CONTINUOUS ENVIRONMENT

In this chapter, the application of solutions evolved in abstract environments to a continuous computer game environment is discussed. In Chapters 4 and 5, an abstract game is outlined and a set of solutions are created in it for a number of different scenarios. This chapter explores the possibility of using an abstract environment to create solutions for a continuous game. The advantages and disadvantages of such a method are also discussed.

This chapter addresses the hypothesis **H5** as described in Section 1.4, namely that the study of agent behaviours in abstract economic dilemmas provide appropriate templates upon which to create groups of NPCs in computer games. These dilemmas, if they require extension, exhibit similar behavioural properties in the computer game domain as they did in their original context.

Firstly, this chapter outlines the continuous environment which simulates properties of a commercial computer game. Secondly, the integration of the GP generated solutions into the continuous environment is described. Finally, the performance of the behaviours created in the abstract environment in the continuous game is assessed. A discussion is provided of the factors which influence the performance of the GP generated solutions.

6.1 Game Environment

The game environment is described in this section. Many of the aspects of the environment are strongly influenced by Millington [2009], which is a good source for traditional and advanced implementations of AI for computer games across a large set of topics. The main features of the game environment are a continuous time-line, physical locations, independent evaluation of each NPC's decision making and the use of traditional methods for navigation. The game is drawn using Open GL and GLUT [Kilgard, 2012]. The game is created in C++ for windows.

The environment is represented as a 2D world projected in 3D space. The environment remains abstract in a sense, eschewing the use of texturing or other graphical enhancements to focus on a more analytical approach in the application. The environment recreates the abstract game such that agents, buildings and resources all have a physical location. The agents have a maximum velocity that they can travel and have the ability to move around the world as necessary. The physical objects move using a kinematic mechanism whereby, at every update the force of the movement of the object is calculated.

In addition to the features that the agents have in the abstract game (health, roles, actions, etc.), each is modelled with a velocity and an orientation as well as a circular size for the motion calculations. This allows the game engine to provide kinematic steering behaviours with which the agent's navigate around the game world closely resembles a computer game.

Each agent's decisions and actions are evaluated in a similar fashion to the abstract game. An agent decides to perform an action based on some decisions in the decision tree. The action gets added to the agent's action queue which are then executed in sequence. The number of actions on the agents action queue are variable, depending on the composition of the decision tree.

Each action that is available in the abstract game must be translated into a continuous action for the new environment. Each action is decomposed into subtasks which are completed in sequence with the final result changing the state of the game objects in the same way as the abstract action.

The environment is laid out in a way to represent the abstract game. The abstract environment consisted of buildings and locations which were equidistant from each other however, this is difficult to represent in a 2D or 3D game engine. The game is laid out with the factory in the centre of a game world. The two resource games requires two refineries which are located in opposite corners of the world equidistant from the factory.

The number of workplaces varies depending on the roles chosen within the group. Some groups may favour harvesting one resource over the other, resulting in more workers allocated to that role. Each worker has their own workplace which is located along a semi-circle on the opposite side to the factory, making each workplace equidistant from the refinery.

This implementation is necessarily different from the abstract game. Not all locations are equidistant from each other as in the abstract game. The time to travel between locations will now vary for the agents. It is hoped, however, that the group behaviours can be created in the abstract environment and applied to the continuous with a degree of success. A screenshot of the continuous environment can be seen in Figure 6.1 showing a game layout with nine agents.

6.1.1 Action Translation

The following outlines the actions available to the agents and the translations into the continuous environment.

Eat **if** Agent is at the Factory **then**

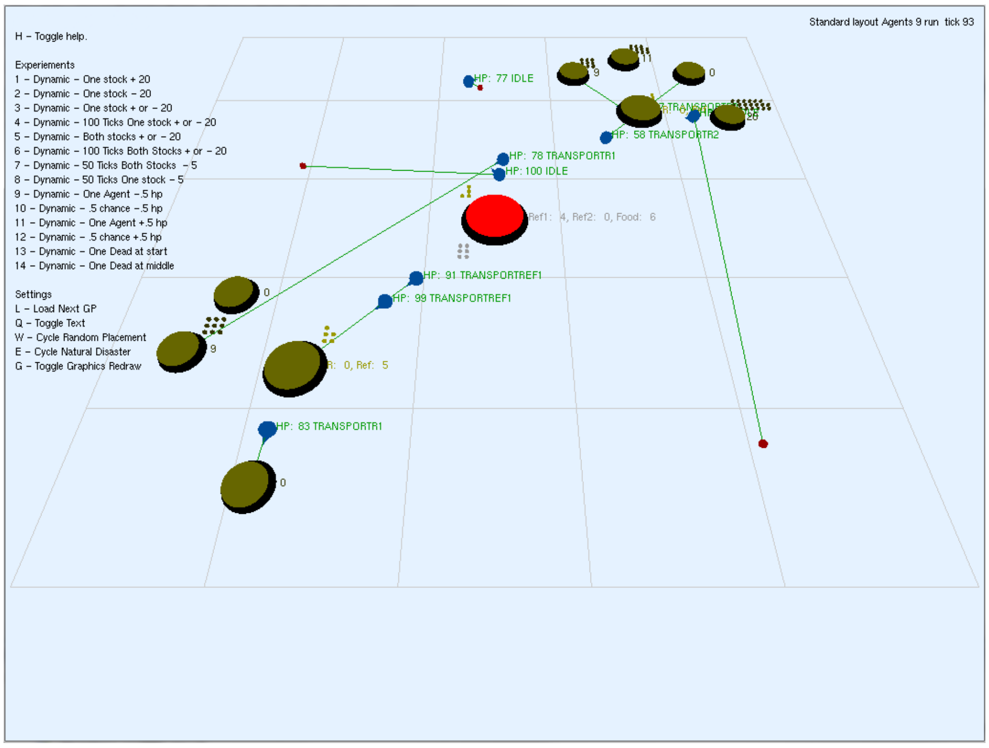

Standard layout Agents 9 run tick 93
H – Toggle help.
Experiements
1 – Dynamic – One stock + 20
2 – Dynamic – One stock – 20
3 – Dynamic – One stock + or – 20
4 – Dynamic – 100 Ticks One stock + or – 20
5 – Dynamic – Both stocks + or – 20
6 – Dynamic – 100 Ticks Both Stocks + or – 20
7 – Dynamic – 50 Ticks Both Stocks – 5
8 – Dynamic – 50 Ticks One stock – 5
9 – Dynamic – One Agent –.5 hp
10 – Dynamic – .5 chance –.5 hp
11 – Dynamic – One Agent +.5 hp
12 – Dynamic – .5 chance +.5 hp
13 – Dynamic – One Dead at start
14 – Dynamic – One Dead at middle
Settings
L – Load Next GP
Q – Toggle Text
W – Cycle Random Placement
E – Cycle Natural Disaster
G – Toggle Graphics Redraw
HP: 77 IDLE
HP: 58 TRANSPORTR2
HP: 78 TRANSPORTR1
HP: 100 IDLE
Ref1: 4, Ref2: 0, Food: 6
HP: 91 TRANSPORTREF1
HP: 99 TRANSPORTREF1
R: 0, Ref: 5
HP: 83 TRANSPORTR1
9
11
0
26
0
9
0

Fig. 6.1: Continuous Game Environment

```
        if Food Stock at Factory > 0 then
            $FoodStock--$
            $AgentHealth+ = FoodReplenishment$
        else
            The action is complete, remove it from the agent's queue if the
            minimum action time has elapsed
        end if
    else
        if $NavigationTarget \neq Factory$ then
            $NavigationTarget \leftarrow Factory$
        end if
        Use Seek to $NavigationTarget$
    end if
IDLE  Begin Wander
    if IDLE time elapsed then
        $Health--$
        The action is complete, remove it from the agent's queue if the
        minimum action time has elapsed
    end if
HARVEST  if Agent is at its Workplace then
        $Workplacestock+ = 1$
        $AgentHealth- = CostofHarvest$
        The action is complete, remove it from the agent's queue if the
        minimum action time has elapsed
    else
        if $NavigationTarget \neq AgentWorkplace$ then
            $NavigationTarget \leftarrow AgentWorkplace$
        end if
        Use Seek to $NavigationTarget$
    end if
```

```
TRANSPORT  if Agent is Carrying a Unit of Resource then
    if Agent is at Refinery then
      $RefineryRawResource++$
      $AgentHealth -= CostofTransport$
      The action is complete, remove it from the agent's queue if the
      minimum action time has elapsed
    else
      if $NavigationTarget \neq Refinery$ then
        $NavigationTarget \leftarrow Refinery$
      end if
      Use $Seek$ to $NavigationTarget$
    end if
  else
    if $NavigationTarget \neq AgentWorkplace$ then
      $NavigationTarget \leftarrow AgentWorkplace$
    end if
    Use $Seek$ to $NavigationTarget$
    if Agent is at Agent Workplace then
      if $AgentWorkResourceStock > 1$ then
        Agent Workplace Resource Stock -= 1
        Agent Carrying Resource += 1
      else
        The action is complete, remove it from the agent's queue if
        the minimum action time has elapsed
      end if
    end if
  end if
```

The behaviours introduced into the continuous world are mainly concerned with agent navigation. These behaviours use the implementation provided by Millington [2009] and are detailed below:

SEEK A seek behaviour calculates the direction from the character to the target and requests a velocity along this line.

WANDER A kinematic wander behaviour moves in the direction of the character's current orientation with maximum speed. A steering behaviour modifies the character's orientation which allows the character to meander as it moves forward.

These two behaviours combine to provide a simple continuous environment for testing the generated behaviours. Other subcomponents of behaviours are built in to the implementation like *Independent Facing* which, allows the character to look in separate directions to movement and *To Face* which, rotates the character so it looks where it is going.

As the behaviours were created in an abstract environment with distinct time ticks, the notion of a minimum action duration is introduced to provide an equality to the length of time an action can take. The variability of the action length in the continuous environment is part of the complexity which makes planning and coordination difficult. This concept is discussed in more detail in Section 6.2.1.

The actions now have a minimum duration that they must spend executing until they can be considered complete regardless of the status of the game effecting components of that action. Some actions can take longer, like those with long travel distances, which can change the performance of the group in the continuous world compared to that of the abstract environment. The minimum action time is determined through experimentation and is counted in term of updates to the AI/Kinematic system. The game tick is determined from this minimum action length and all buildings perform their updates at this point (for example the factory converting the refined resources into food).

6.1.2 Game Update Loop

The game environment gets updated at the computer's draw frequency, usually determined by the monitor (usually in the range 60 Hz – 120 Hz). The refresh frequency of the drawing is controlled by the GLUT system and the remainder of the update cycles available are used to update the characters' physical behaviours as well as determining the AI decisions and handling input to the game. Each draw update renders each agent, building, resource and game world object.

Each game update, the agents' actions are evaluated with their kinematic targets and motions being updated. Every time an agent is updated, their action queue is checked to make sure it has an action to perform. If it has an action, the corresponding movement updates (such as setting the workplace as the target and seeking towards it) are executed. If the action queue is empty, the GP subtree that describes the actions for that agent is evaluated and as a result, actions are added on to the queue.

Depending on a number of factors including the interval length and the distance to be travelled by an agent, the actions can take a varying time to complete, which is in contrast to the fixed and uniform action lengths of the abstract game. A discussion on choosing the best interval for the continuous environment is conducted in Section 6.2.1.

Health is deducted from an Agent when an action has completed, either successfully or unsuccessfully, in order to keep the cost of actions simple. This saves creating and evaluating a continuous function to reduce health every update as well as ensuring a simplistic mapping of whole number costs from the abstract game to the continuous.

A useful feature of the game engine for evaluation is the ability to not update the graphics on screen. In this case, the kinematic motion's update timing, which is usually derived from the time between frames, is fixed at 75Hz (which is the refresh rate of the monitor on the test machine). This saves update cycles and allows games to be executed more quickly with the same behavioural outcomes as if the graphics have been rendered.

6.1.3 Integration of Generated Solutions

A group's behaviour tree is saved in an output file from the GP evolution. Each file has information on the nodesets and the functions and terminals contained within. The file also contains a tree structure, which provides the instructions for combining the functions and terminals together.

A semantic parser is then needed for the tree. This parser resembles the function that is called when a GP tree is being evaluated during the evolutionary process. Each of the nodes that are in the abstract evaluation need to be recreated for the continuous environment. Most nodes can however, be duplicated without any modification. The nodes that require modification are those that deal with placing the actions on to the agent's action queue. The full list of GP nodes can be seen in Tables 4.2 and 4.3.

Generally, a selection of solutions will be created using the abstract environment. Typically, the best individual is chosen at the end of each run but this is not necessary. Any criteria, such as minimum fitness or ranking in the final generation perhaps would be as valid for choosing sample solutions.

In this case, the concern is with the general performance of the GP process and its ability to create solutions automatically and so the individual with the best fitness from a run is chosen. This means that entire separate evolutionary runs must be conducted which is more time consuming but leaves a better impression about the GP's performance in general.

Each sample solution is then placed in a population of sample solutions which is, conveniently, a GP population structure. This population can then be saved to a file, with its nodeset information, to be reused at game time. Once this has been loaded from the file, it is trivial to access any member of that population which is an entire group's behaviour tree. The individual subtrees of the group behaviour can also be accessed easily in order to determine the behaviour of an agent.

6.2 Static Environment - Abstract to Continuous

In this section, the application of the abstract created behaviours in the continuous environment is explored. Specifically, the effects of some of the features of the continuous environment on the performance of the abstract behaviours are examined. The main feature that is studied is the transition from an abstract tick to a continuous environment. In the abstract game, the tick allows behaviours to execute actions, move around the environment and coordinate roles in a timely fashion.

Cooperation within the abstract solutions may be linked with the strict execution time of actions. In the continuous environment, simply through navigating the world, it may effect the speed of completion of actions as agents can have different distances to travel. As such, the performance of the abstract behaviours may not be consistent with expectations from the abstract environment.

This section firstly seeks to determine, whether the abstract behaviours can be used in the continuous environment or is the abstract environment too simple to capture the group coordination and cooperation needed. Secondly, additional features are added to the continuous environment that are deemed essential in order for the abstract generated behaviours to be useful, specifically the minimum action time and fixed locations of objects. Thirdly, modifications and allowances are made for the behaviours in the continuous environment in a bid to increase performance. Finally, a more stringent selection process is used in order to preselect better performing behaviours at the abstract level.

In order to conduct this examination, a set of abstract behaviours are created. Evolutionary runs are conducted in the abstract static environment as defined in Section 4.4. The following game parameters are used:

- Action Cost: 1 (Harvest, Transport, Move)
- Food replenishment: 18, Food multiplier: 1
- Game length: 1000 ticks
- Agent starting and maximum health: 100

The individual with the best fitness score at generation 100 is chosen as a representative of the evolutionary run with the following additional condition: all the agents must survive for the entire length of the game. This allows an easy comparison for the performance of the abstract behaviours in the continuous environment. Twenty individuals are selected from separate runs and saved for each group size of three, five, seven and nine agents. The saved group behaviours are used for the experiments in the following subsections.

6.2.1 Varying Tick Length

In this section, the tick length in the continuous game is examined. The continuous game differs from the abstract in the agent update. Agents must be updated repeatedly while performing the same action in order to move them around the game world. The actions for an agent only need to be updated when they are finished. The abstract game's notion of a *tick* is necessary to provide a measure of equality for the duration of action, given that the running time is no longer deterministic.

Group Size	3	5	7	9	Overall
Action Length	165	180	171	179	174
Standard Deviation of Length	99	93	99	106	99
Avg. Agent Lifetime	199	427	324	305	314

Tab. 6.1: Baseline - average Tick length and average agent lifetime for a zero minimum tick length

In the continuous game, an action may be executed over many update cycles. A count of the game updates is maintained and at certain intervals, the buildings and resources are updated. The actions an agent can perform all conform to having the same minimum time as this interval. This provides a way to make the duration of actions more uniform while maintaining the flexibility of the continuous environment. This interval is known as the *Tick Length*.

In order to create a baseline for comparison for the *Tick Length*, the game is run, using the generated strategies, with a zero minimum tick length. The number of update cycles required to execute each action is recorded and then averaged over the number of actions that the agent has executed. A final average is taken across all the agents in that group. After each action has completed, the agent's tick counter is incremented. As each agent is completing actions at different rates the group's tick counter is the highest tick by any alive agent. The games attempt to last for 1000 ticks or until all agents are dead. The findings are listed in Table 6.1.

There is poor synchronisation between the agents, that is, the agents' actions are completing at different rates, as denoted by the high standard deviation of action length compared to the average action length. Some actions may complete very quickly, like the action *Eat* when the agent is already at the *Factory*. Actions which can involve moving to multiple locations, like *Transport Raw Resource*, have varying durations depending on the position of the agent and the stock levels of the resource. The large standard deviation of the action length compared to the average length indicates that, within the group, agents are executing actions at different frequencies.

The agents' actions are executed and completed at a varying rate and this results in a poor overall average life time for the agents, especially when the behaviours can survive in the abstract environment. If, from the abstract behaviours, the agents are relying on a particular world state after the completion of their action, the variance in duration may mean that the behaviours are no longer useful. The poor performance of the behaviours indicate that a direct translation of actions from the abstract environment to the continuous environment is not possible. None of the evolved solutions survive in the continuous environment under these conditions. Based on these findings a selection of *Tick Lengths* are chosen and compared.

The purpose of introducing a minimum action duration as specified by *Tick Length*, is to try and improve the performance of the abstract behaviours in the continuous environment. The minimum action duration helps to bridge the gap between the uniform fixed length and highly synchronised abstract environment in which the behaviours are created and the continuous environment where the agents get updated, both in terms of behaviours and physical characteristics, across multiple game update loops.

The game tick in the continuous game is examined at four different update frequencies: 150, 200, 250 and 300 update cycles. The range of frequencies is determined programmatically. If the *Tick Length* is set shorter, few, if any, actions will have completed before the tick arrives. If the *Tick Length* was any longer, it would remove the benefits of a continuous environment as agents would be waiting after most of their actions had completed.

Figure 6.2 shows the average agent lifetime for each of the tick lengths. From this figure, it is clear that even though the behaviours created in the abstract environment survived for the game duration, the groups do not always survive in the continuous environment. That is not to say that some of the groups

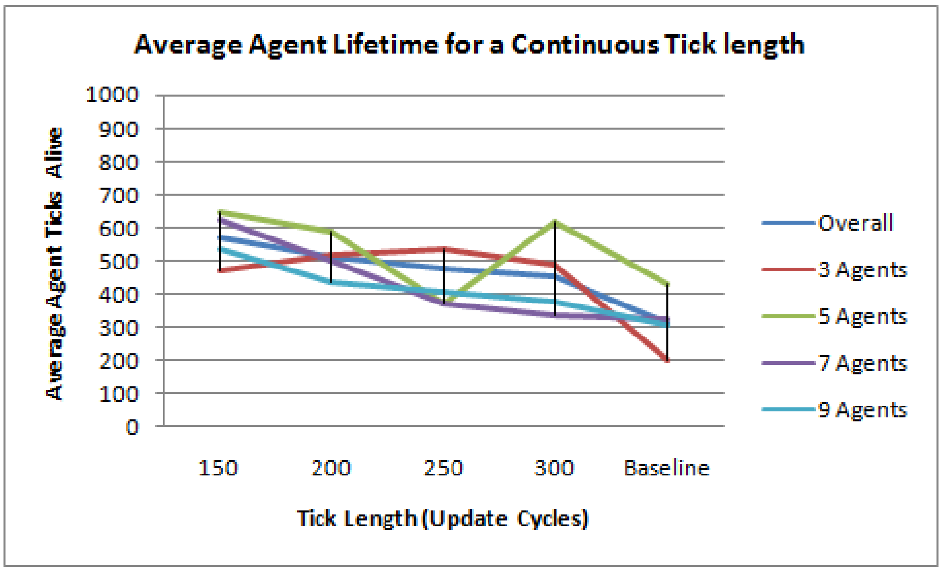


Fig. 6.2: Average Agent Lifetime with various Tick Lengths

could not survive in the continuous environment.

Figure 6.3 show the survival rates of the groups. The three agent groups have the best survival rate (where all of the agents survive 1000 ticks) with an average rate of 0.33 over the four tick lengths. The 250 *Tick Length* has the best rate of 0.4 (8 solutions out of 20) and the 150 tick length has the worst. Interestingly, in spite of having the worst survival rate the 150 *Tick Length* has the highest average agent lifetime out of all the other solutions. This demonstrates that a high average agent lifetime is not always indicative of a high survival rate.

The results of these runs revealed that, for this selection of behaviours and tick lengths, no one tick length provided the best performance regardless of group size. The performance of the introduced *Tick Length* is not close to the performance of the behaviours in the abstract game but there is an improvement over the ad-hoc approach of the baseline zero length described

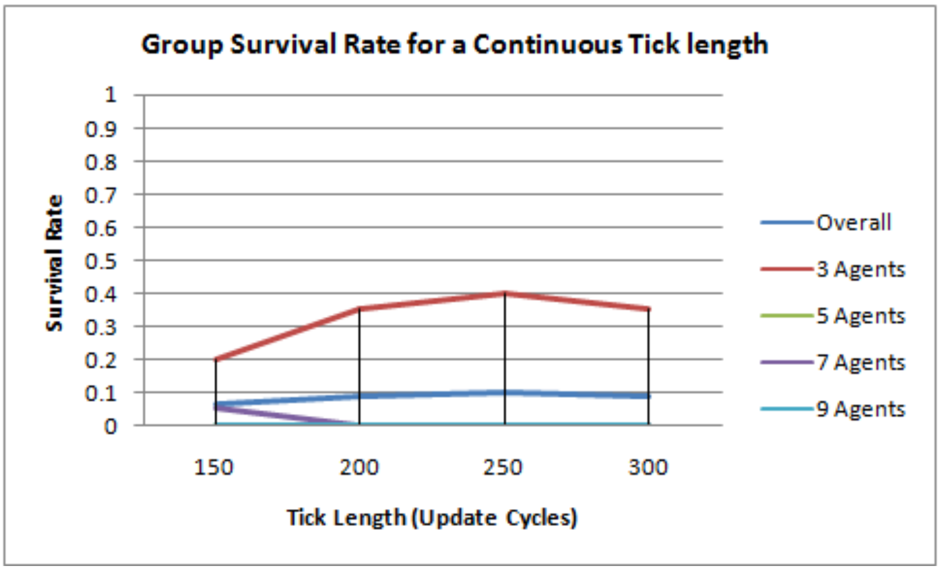


Fig. 6.3: Average Group Survival rate with various Tick Lengths

above. For each update frequency, there was at least one group size that under-performed compared to the average for that tick length, indicating that there is not one optimal value for *Tick Length*.

Even though the average agent lifetimes do not appear much greater than the baseline solutions, all except one group size/tick length setup exceeded the baseline average agent lifetime. Importantly, in each of the minimum *Tick Length* runs, groups were able to survive for the length of the game. This indicates that solutions can be created in an abstract environment for a continuous game.

Although, for this sample of behaviours, the group's performance decreased in terms of average agent lifetime, it was shown that behaviours created in abstract environments can work in a continuous environment. Even though there was only a 0.55 overall survival rate for the best tick length, creating a selection of behaviours in the abstract environment and then evaluating in

	Abstract	Continuous
Total time for 300 games (ms)	1123	221650
Avg. time for 20 iterations (ms)	74.87	14776.66667
Std. Dev. for 20 iterations (ms)	83.38	4401.95
Avg. time for one game (ms)	3.74	738.83

Tab. 6.2: Time to evaluate behaviours in an abstract and continuous environment

the continuous environment would be faster than evolving in the continuous environment.

To investigate this matter, a behaviour was chosen and loaded into both the abstract and the continuous environments. The behaviour was evaluated in a static environment and a number of dynamic environments for twenty iterations in each. Fifteen different environments were timed for the twenty iterations using the total milliseconds elapsed as a measure of the execution overhead. The results of the trials are presented in Table 6.2. The results of this trial indicate that executing behaviours the abstract game is in the order of hundreds of times faster than in the continuous environment.

The length of time needed to run solutions in the continuous environment is prohibitive when it comes to the evolutionary process. As each individual needs to be evaluated every generation, there are simply too many games to run. Even though there is no guarantee of an effective solution, by using the abstract approach the number of solutions created is far greater than the continuous environment. Given the survival rate of abstract solutions, even with multiple evolutionary runs it is faster to create a solution using the abstract method over the continuous method.

	Move				No Move			
	150	200	250	300	150	200	250	300
Overall	0.0625	0.0875	0.1	0.0875	0.425	0.6	0.6875	0.6875
3 Agents	0.2	0.35	0.4	0.35	0.6	0.65	0.75	0.7
5 Agents	0	0	0	0	0.6	0.6	0.75	0.6
7 Agents	0.05	0	0	0	0.25	0.7	0.75	0.65
9 Agents	0	0	0	0	0.25	0.45	0.65	0.8

Tab. 6.3: Comparison of survival rates between environments with and without a cost for the Move action

6.2.2 Modifications and Expert Tweaks

In order to see if increased performance could be obtained from the evolved behaviours, some tweaks are made to the continuous environment. In these runs, the effect of tweaking game action costs can be seen. The continuous game environment settings, like that of action costs, are loaded through a config file which allows for easy alteration. Even though the costs for action in the continuous environment may be equivalent, the differences between the abstract and continuous executions lead to differences in behaviours. Behaviours verified in the abstract game can still be effective in the continuous environment with some expert tweaking.

The cost of *Move* is set to zero for a run of varying game tick lengths. This has the effect of making all actions slightly cheaper, as all actions can contain a Move action. The Figures 6.4 and 6.5 demonstrate the effect of a simple reduction in the cost of Move from one to zero on the average agent lifetime and the survival rate respectively.

A comparison of the survival rates for the environments with and without the cost to the *Move* action can be seen in Table 6.3. The same underlying behaviours are used with only once cost altered. The 250 tick length from the original run and the reduced cost run are the best performing in terms

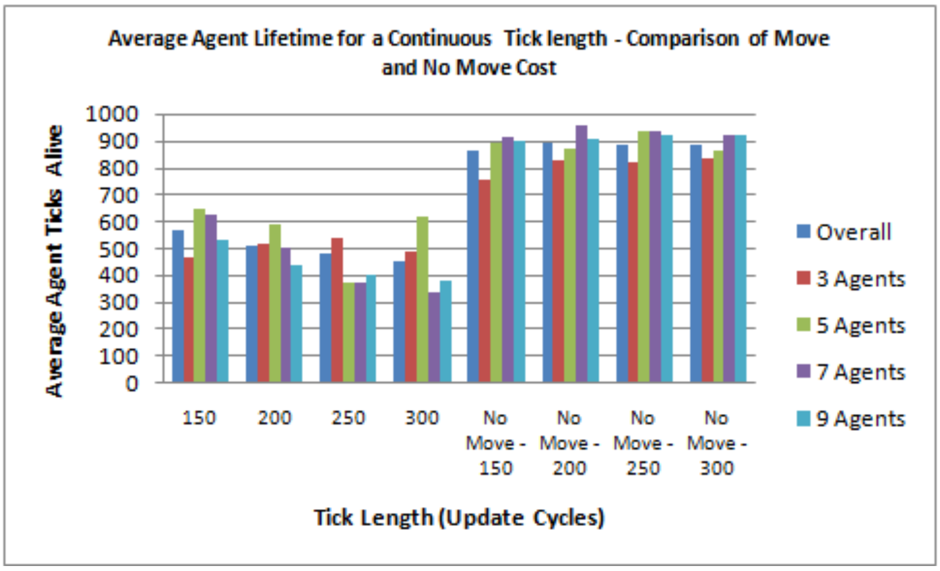


Fig. 6.4: No Cost for Move Comparison - Average Agent Lifetime with various Tick Lengths

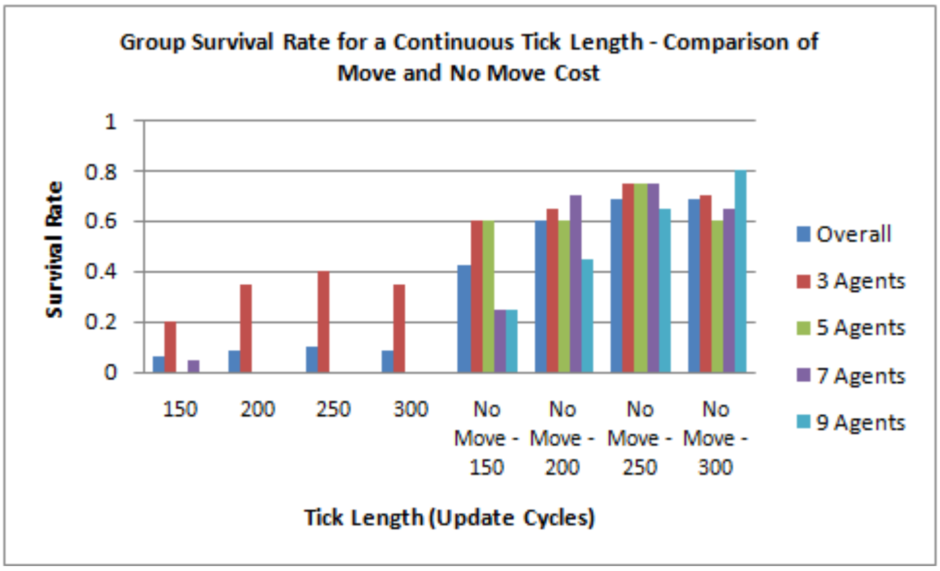


Fig. 6.5: No Cost for Move Comparison - Average Group Survival rate with various Tick Lengths

of numbers of groups surviving 1000 ticks. For 80 samples (20 runs for each group size), the number of groups surviving is 8 (10%) for the original and 55 (69%) for the reduced cost. Although these solutions did not perform at the same level as in the abstract environment, they were quickly adopted and tuned for good performance in a continuous environment.

By applying changes stemming from expert judgements of the game environment, the performance of the behaviours of the groups can be greatly improved. Even though the cost of the actions have been altered, the behaviours are the same as they were evolved in the abstract environment. With a minor tweak to the continuous game environment, the abstract behaviours can perform successfully.

6.2.3 Additional Abstract Requirements

Ideally, the application of evolutionary computational techniques to the creation of AI behaviours would reduce the amount of time developers would spend programming AI solutions. By imposing some additional constraints on the selection of evolved solutions, it is hoped that the performance of the generated solutions will increase without requiring any tweaking post-evolution. The use of the abstract environment, while not capturing all features of the continuous game, allows for fast evaluation during the evolutionary process.

In this experiment, new behaviours are generated using the GP algorithm and a more stringent selection is used to choose the sample behaviours. The solutions are created as before, but at the end of each generation the solution with the best fitness is tested. If the chosen group behaviour survives for 50000 ticks in an additional evaluation, it is selected and a new evolutionary run is commenced. If no solution is found by generation 100, the population is discarded and a new one is started. The running time of this process is not deterministic, as the yielding of a solution which matches the specified criteria is not guaranteed.

Firstly, a comparison is conducted to see what, if any, effects are seen by using the *50k Tick* behaviours on the natural synchronicity of the behaviours. Using the method for determining a baseline score for the strategies from Section 6.2.1, the *50k Tick* behaviours have very similar scores for average action length (also exhibiting a high standard deviation in length) to those of the *1000 Tick* behaviours from previous experiments. The *50k Tick* solutions do survive, on average, 24% longer than the *1000 Tick* behaviours in this baseline, however, no solutions survive for game length.

Introducing the minimum Tick values as presented in Section 6.2.1, across the four values the *50k Tick* behaviours had a 23% increase in average agent lifetime over the *1000 Tick* behaviours. Comparing the 200 *Tick Length* for each of the sets of behaviours, the differences can be seen for average agent lifetime in Figure 6.6. The differences in group survival rates can be seen in Figure 6.7.

By removing the cost of move from the *1000 Tick* behaviours the solutions perform very strongly. However, by evolving the *50k Tick* behaviours and removing the cost for Move the performance of the solutions is improved further. An increase in average agent lifetime is good but an increase in survival rates is more important. The increasing survival rates means that a higher proportion of the solutions survive the game and are usable in the continuous environment. In this case, usable would mean that the abstract behaviour could be applied in the continuous environment with the expectation that the group could sustain itself for the specified 1000 ticks.

In summary, solutions with some modifications and solutions created with a more stringent selection process improve the performance of the abstract solutions in the continuous environment. By using modification alone, the survival of the groups across all solutions increases from 0.09 to 0.6. By using

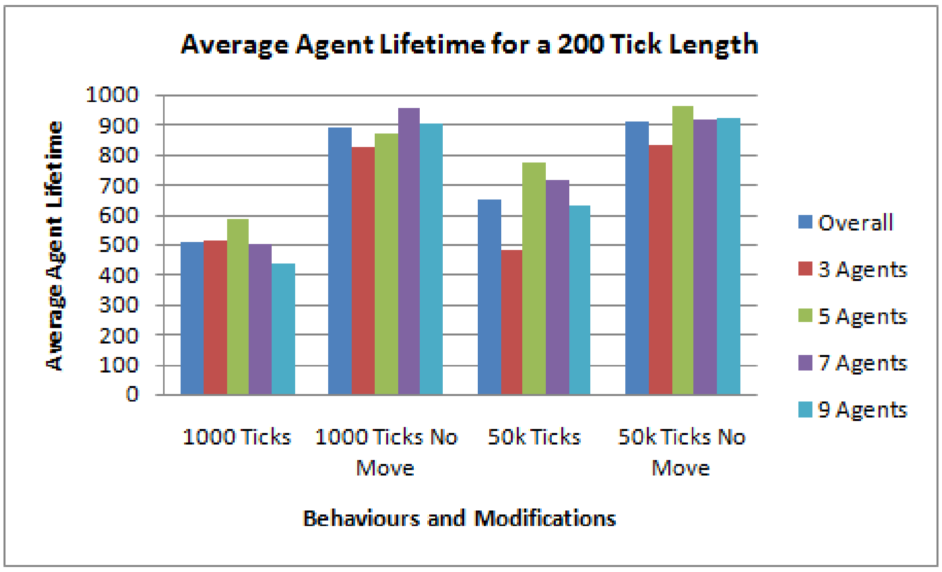


Fig. 6.6: 200 Tick Length - Performance of Behaviours - Average Agent Lifetime

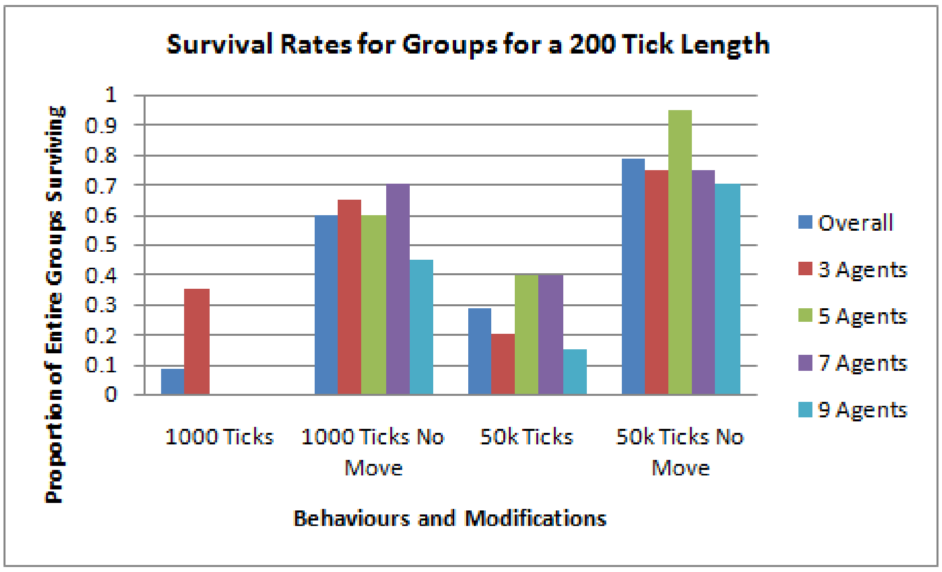


Fig. 6.7: 200 Tick Length - Performance of Behaviours - Group Survival

extra selective techniques the survival rate is 0.29 but by combining this with modification, the survival rate increases to 0.79.

In this case, it is possible to create solutions for a continuous game using an abstract environment. In order to get the best results, some modification of the environment rules is needed when translating from the abstract to the continuous. However, by using very selective techniques for choosing behaviours from the abstract environment, solutions were found which could perform well.

6.2.4 Summary

In this section, parameters which have an effect on the performance of abstract created behaviours in a continuous game environment are analysed. The first parameter examined is the mapping of the abstract tick onto the continuous game. Even though actions incur the same costs in each environment, the non deterministic running time of actions in the continuous environment leads to a lack of synchronisation. Even though all the behaviours created in the abstract game were able have the whole group survive for 1000 ticks, in the continuous game this is not the case. The performance is worse, although this can be compensated for somewhat by the introduction of a minimum action time.

By using this minimum action time, and by altering other parameters, the performance of the abstract behaviours could be improved in the continuous environment, without recreating or altering the generated behaviours. By removing the cost of move, a larger amount of the twenty behaviours for each group size survive for the entire game length. Removing this cost effectively makes the continuous environment less harsh than that of the abstract but it does not alter the behaviour of the agents. Even though the initial mapping of the abstract behaviours did not yield many good behaviours, with expert tweaking the solutions were made usable.

The avoidance of the necessity for an expert user is the motivation for using evolutionary computation to create behaviours. Additional constraints were placed on the chosen abstract generated solutions. In addition to being the best at the final generation of the evolutionary run, the solution must also produce behaviours which allow the group to survive for 50000 ticks. As the abstract game allows for rapid prototyping and evaluation of solutions, it is not a large performance hit to add this additional step and certainly faster than using the continuous environment for evaluations. The performance, over the original generated solutions was increased greatly. While still not providing a perfect mapping between the abstract and the continuous there is a great improvement and by removing the cost, the performance is also improved.

In this section, automatically generated abstract behaviours have been shown to usable as AI characters in a static simulated continuous computer game environment. The abstract environment is easily mapped in a continuous world and the integration of the automatically generated solutions was shown to be straightforward. These behaviours require no modification in order to provide surviving groups of cooperating agents within the continuous environment.

6.3 Layout Experiments

In this section, the effect of game layout on the translation of behaviours from abstraction to continuous is explored briefly. This experiment serves to highlight a potential drawback of applying behaviours created in an abstract environment to a continuous one. If the behaviours are to maintain their original properties, the continuous environment should reflect, to a certain extent, the properties of the abstract game. Variations of locations for buildings and resources are used to highlight changes in the continuous implementation that may be desired, but are not taken into account in the abstract environment.

The movement of agents around the game world is one of the larger differences between the continuous and the abstract games. This introduces the importance of distance between game objects and the distance has the greatest effect on the action duration. If all game objects were equidistant, as they are in the abstract game, the action duration is much more easily controlled.

This section serves to highlight one of the many aspects of a continuous environment that can affect the performance of the abstract behaviours, even after careful planning of parameters like minimum action duration (Section 6.2.1). Introducing variation to the placement of objects may have the effect of disrupting the coordination of the group, by producing non-deterministic movement durations (across several instances of the same behaviour). In the continuous environment, changing the layout may effect the speed of completion of actions and as such, the performance of the abstract behaviours may not be consistent over all layouts.

Varying degrees of randomness will be introduced into the placement of game objects. A selection of high performing strategies, identified from trials run in Section 6.2.1, are chosen (one for each group size) and used to compare the various layouts of the objects in a static environment. In this case, the static environment means that there are no outside influences on the environment once the game starts.

The game will be executed in the continuous environment with the same parameters as in Section 6.2.1. The tick length for the continuous environment will be fixed at 250, which was the best performing length tested in Section 6.2.1.

Each behaviour will be played 20 times for each random layout configuration. The random layouts are defined as follows:

- A baseline of the static environment layout from Section 6.2.1 where each workplace is equidistant from each refinery and each refinery is equidistant from the factory. Furthermore, the keeps and factory positions are fixed for the duration of the game. This baseline layout will not only allow for comparisons with the random layouts but will allow for the identification of any other random factors that effect the survival of the agents and the mapping of behaviours from the abstract environment.

- Random Factory placement - in this layout the workplaces and refineries are placed as per the fixed layout. The factory is then placed at random around the game world. This may have the effect of skewing the effort needed to get one of the stocks to the factory in order to make food.

- Random Refinery placement - in this layout the workplaces and factory are placed as per the fixed layout. The refineries are placed at random around the game world with the workplaces around them.

- Random placement for all buildings. All buildings are placed at random throughout the world. This layout placement should have the most disruptive effect on the performance of the behaviours.

Table 6.4 displays the overall results from the layout experiments. For these experiments, the same four candidate behaviours (one behaviour to represent each group size) were run with four different layout approaches in the continuous environment. The *Static* layout environment, where the buildings and resources are set out in the same way each time, had the worst average agent lifetime but also had the lowest standard deviation of agent lifetime and the second best survival rate.

By placing all the buildings and resources randomly each time, the survival rate drops by 35% compared to the static placement. All the configurations for random placement increase the standard deviation of the average agent lifetime when compared to the *Static* layout (as expected). With the *Random Refinery* placement, there is an improvement in survival rate over the *Static* layout by 20% which was unexpected.

The purpose of this experiment is to highlight one of the many factors that can influence the performance of the abstract behaviours in the continuous environment. The abstract game is conducted in a largely fixed environment. If the game world is laid out in at fixed fashion in the continuous environment, it will most closely resemble the environment in which the behaviours were created.

However, in a computer game, it may be desirable to have a range of game environments which, not only vary the layout of buildings and resources, but also vary the value of the resources and attributes of the agents. There is potential to model each configuration in an abstract environment and create behaviours for it automatically. This section highlights that any change in the continuous environment will have either a positive or negative impact on the performance of the generated behaviours.

	Static	All Rand.	Rand. Factory	Rand. Refinery
AL	692.25	719.11	762.55	738.96
Std. Dev. AL	178.45	255.81	261.65	211.73
SR	0.45	0.34	0.44	0.55

Tab. 6.4: Comparisons of various layout configurations. Average (Avg.) and standard deviation (Std. Dev.) agent lifetime (AL) in Ticks. The survival rate (SR) for each configuration is included

6.4 Dynamic Environment

Computer games allow a user to interact with the game world and be part of the action unlike most entertainment media which provide passive consumption for enjoyment. The player can change the world by buying and selling stocks and by fighting and killing AI characters, all while following a story arc or embarking on a quest. These unpredictable interactions by players can cause difficulties for AI characters. AI characters may have tightly scripted behaviours or are designed in such a way that their performance is dependant on specific environmental factors. Any changes in the environment or dynamic elements introduced by external parties can cause a loss of performance and unexpected behaviours.

In Section 5.1.1, a selection of dynamic elements are described. The inspiration for the dynamic elements lie in potential interactions within a computer game environment that an AI character may experience. A simulation is drawn up to explore the effects of each one on an abstract game environment. These dynamic elements try to simulate times when an AI character would get into a fight, would die or would encounter resource traders.

The behaviours created in Section 5.6 for an abstract environment are assessed in a continuous environment. The performance of the generated behaviours will provide two insights. Firstly, the ability of the GP algorithm to create behaviours for a computer game with a dynamic environment using abstract game environments for the evolutionary process can be assessed. Secondly, the introduction of random elements into the abstract environment in Section 5.6, showed that behaviours could better survive across a range of environments. An investigation is conducted to see if the improved characteristics of the behaviours hold in a continuous environment.

In this section, the incorporation of the dynamic elements into the continuous game environment are discussed and the experimental setup is described.

Following this, a presentation of the results and a discussion of their meaning is provided.

6.4.1 A Dynamic Continuous Environment

The dynamic elements from Section 5.1.1 are incorporated into the continuous game environment. In the abstract environment, the dynamic elements are executed at a certain fixed frequency and are completed in one tick. For the continuous environment, the same approach is used. The reason that this is appropriate, is that the intention of these experiments is to capture the effects on the performances of the generated behaviours. For this, only the resulting cost or effects of the dynamic interactions on the agents or the resources are needed.

This is a simplification of some interactions. For example, the interaction of an AI character with a player which results in the loss of health for the agent may take some time to play out. For these simulations, it is assumed that the resulting change to the agents and environment can be captured with an appropriate cost.

To apply the abstract dynamic elements to the continuous environment is straightforward. Every game update tick, simply check if a dynamic element should execute. A record of the current dynamic interactions that should take place is maintained and at the correct interval, the environment and agents are altered appropriately. Suitable dynamic environments are described in Chapter 5, with an assessment of their performance in Section 5.6. Each environment is applied in the continuous game for comparison

The full list of environments, used to create behaviours are:

- *Static* - No dynamic elements

- *Resource 1* - At every *Interval* add 20 to one of the *Keep's* raw materials (chosen at random)
- *Resource 2* - At every *Interval* remove 20 from one of the *Keep's* raw materials (chosen at random)
- *Resource 3* - At every *Interval* either add or remove (50/50 chance) 20 to or from one of the *Keep's* raw materials (chosen at random)
- *Resource 4* - Every 100 ticks either add or remove (50/50 chance) 20 to or from one of the *Keep's* raw materials (chosen at random)
- *Resource 5* - At every *Interval* either add or remove (50/50 chance) 20 to or from both of the *Keep's* raw materials
- *Resource 6* - Every 100 ticks either add or remove (50/50 chance) 20 to or from both of the *Keep's* raw materials
- *Resource 7* - Every 50 ticks either remove 5 from both of the *Keep's* raw materials
- *Resource 8* - Every 50 ticks either remove 5 from one of the *Keep's* raw materials (chosen at random)
- *Health 1* - At every *Interval* set one agent's health (chosen at random) to half its current value
- *Health 2* - At every *Interval* 50/50 chance of setting each agent's health to half its current value
- *Health 3* - At every *Interval* set one agent's health (chosen at random) to twice its current value (max 100)
- *Health 4* - At every *Interval* 50/50 chance of setting each agent's health to twice its current value (max 100)

- *Death 1* - In the first tick, select one agent at random and set their health to 0
- *Death 2* - In tick 500, select one agent at random and set their health to 0

A selection of behaviours are chosen from the set of abstract behaviours in Section 5.6. The configuration of *Food 3x 8* was chosen to be the candidate. The *3x* multiplier provides three units of food for every unit combination of both refined resources with a replenishment value of eight being ascribed to the food. For this configuration, the group sizes three and nine were selected. In the abstract tests, the three agent group is the most successful across all environments in terms of survival and the nine agent group is the least successful. Whereas, the nine agent group was the best in terms of average agent lifetime. The behaviours for these two group sizes should provide insight into the range of performances that can be expected in the dynamic continuous environment.

For both group sizes, twenty example behaviours for each of the fifteen different environments are applied in the continuous environment. Each behaviour is run in each dynamic environment for twenty iterations. The average agent lifetime and the survival rates are recorded over the iterations with the average of each being used as the score for that behaviour in that environment.

The average agent lifetime is the total number of ticks the agents in a group are alive for divided by the number of agents in the group. The survival rate is the number of games where all agents survived to the end divided by the total number of games. For a group's survival rate to be counted, they must have survived the environment in all iterations.

Each of the example set of twenty behaviours have been created either in the static environment or in one of the fourteen dynamic environments. Applying each behaviour across the range of environments allows the assessment of the behaviours in its own environment and its suitability as a general behaviour. These results also allow the comparison of behaviours in the static environment and with the performances of the behaviours in the same ranges of games in the abstract environment.

6.4.2 Continuous Dynamic Results

In this section, the results of the application of the abstract generated behaviours in the continuous dynamic environments are presented.

In Table 6.5, the overall average agent lifetime and survival rates are shown for both the abstract and the continuous environments for the configuration of *Food 3x 8* with group sizes of three and nine. There is a decrease of 14% for the three agent group and a decrease of 27% in the nine agent group in the overall average agent lifetime when comparing the abstract and continuous environments. This results in a decrease of 18% and 88% for the three and nine agent group respectively in survival rate.

For the three agent group the behaviours evolved in the *Resource 1* environment performed best overall in terms of average agent lifetime and in survival rate. The behaviours evolved in the *Resource 3* environment performed best in terms of average agent lifetime, with the behaviours from the*Death 1* environment performed best in terms of survival rate for the nine agent group.

For the three agent group, the best dynamic environment behaviours were

	Abs. AL	Cont. AL	Abs. SR	Cont. SR
3 Agents:	756.43	652.18	0.47	0.38
9 Agents:	892.38	648.02	0.35	0.04

Tab. 6.5: Comparisons of the Abstract (Abs.) and Continuous (Cont.) Dynamic round robin experiments. The overall average agent lifetime (AL) in ticks and the overall survival rate (SR) are shown

able to increase the average agent lifetime by 15% and the survival rate by 25% over the static behaviours. For the nine agent group, an increase of 36% for the average agent lifetime and 46% for the survival rate over the static behaviours was achieved. The only environment that did not have a group survive in either configuration was the *Death 1* environment, which is impossible for the three agent group to survive in and was shown to be the most difficult environment to create surviving agents for other group sizes in the abstract experiments. In the other fourteen environments, there was at least some groups that could survive.

There is a drop in the performance when moving the generated behaviours from the abstract to the continuous environment. Even with the drop in performance in both the three and nine agent groups, behaviours were created that could survive in the dynamic environments. Evolving the behaviours in a dynamic environment was shown to be able to create solutions with a better performance than those created in static environments. Overall, the three agent abstract behaviours perform more closely to the abstract behaviours in the continuous environment than the nine agent groups.

The expert tweak from Section 6.2.2 was applied to two candidate behaviour sets for the dynamic environments. The twenty behaviours generated in the *Static* and *Resource 1* environments are reevaluated in each of the test environments. The purpose of this test is to ascertain whether or not the behaviours generated in a dynamic environment can be used without particularly stringent selection criteria.

This experiment is conducted using behaviours evolved in a *Food 3x8* configuration which is the best performing configuration, in general, for the abstract environment. This configuration makes it easier to share the food created by the agents as the number of units of food created is the same as the number of agents required to make the food.

A comparison of the average agent lifetime can be seen in Figure 6.8 for each

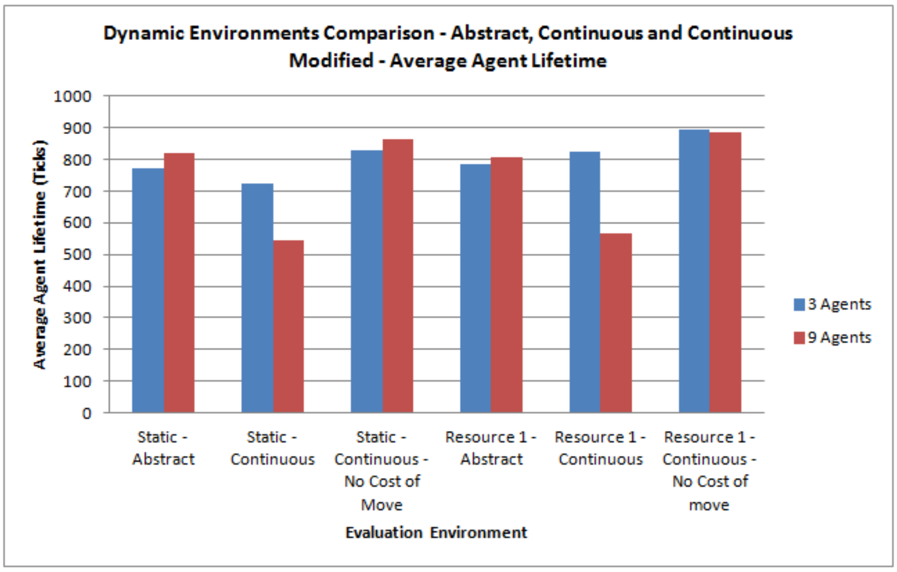


Fig. 6.8: Average agent lifetimes compared over different configurations for the Dynamic environments

of the configurations explored using the dynamic environments. The overall performance for each group size is included for two sets of behaviours, one generated in the *Static* environment and the other generated in the *Resource 1* environment. Figure 6.9 presents the comparison of the survival rates for each configuration.

By using the expert tweak of removing the cost of move from the continuous environment, the abstract generated behaviours can be applied to the continuous environment with more success than the abstract environment itself. Overall, the behaviours in the modified environment do better than both the abstract environments and the original continuous environment in terms of both, average agent lifetime and survival rate.

For both sets of behaviours, the three agent behaviours have almost equal performance in the abstract and the original continuous environments which indicates that the *Food 3x8* configuration is useful for using the some be-

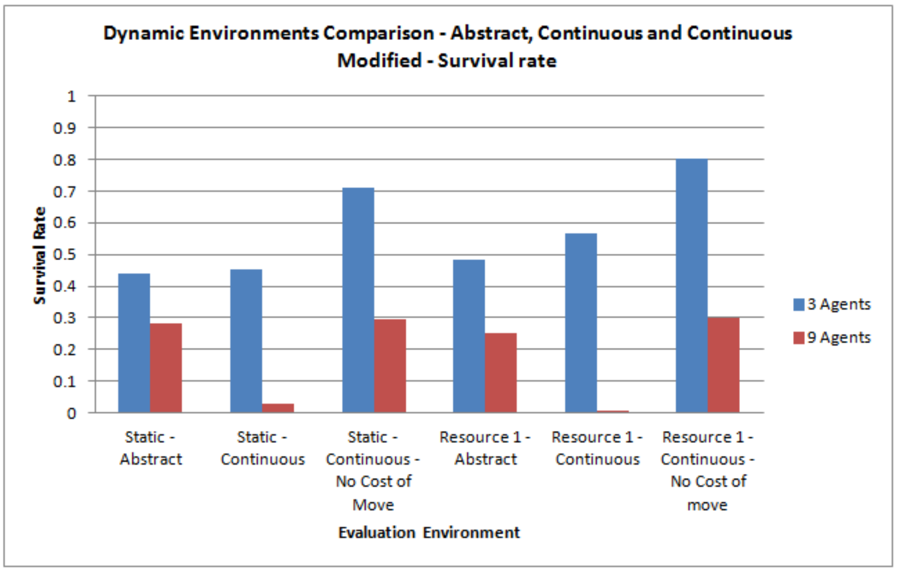


Fig. 6.9: Survival rates compared over different configurations for the Dynamic environments

haviours directly. The nine agent behaviours perform badly in the original continuous environment with a survival rate of 0.01 but improves to 0.3 in the *Resource 1* dynamic behaviours with the removal of the cost for move.

These improvements demonstrate the applicability of behaviours generated in a particular environment, in this case a static and a dynamic environment, across a range of unseen environments. The GP algorithm is able to create group behaviours that can survive in most of the dynamic environments designed to simulate interactions between the AI characters and a player. These behaviours can be used in a continuous environment successfully with little or no modification.

6.5 Discussion

Throughout this chapter, the continuous environment has been used as a representation of a simulated computer game world. The behaviours automatically created in the various abstract environments are incorporated into the continuous environment and their performance is reported on. The performance of the behaviours is considered from two points of view. Firstly, did the behaviours perform as well in the continuous environment as they did in the abstract ones and secondly, are the behaviours created in the abstract game able to survive in the continuous environment?

Both of these measures are empirically easy to measure and, as such, were selected as appropriate values to assess the performance of the behaviours. These measures provide an answer to the question of whether or not abstract environments are suitable for creating group behaviours in computer games. They also demonstrate that GP algorithms can be suitable for creating these group behaviours even though, the application of the created solutions is in a environment differing from the one the behaviours were evolved in.

However, the domain of computer games is rarely empirically measurable although steps have been made to bridge this divide by Yannakakis and Hallam [2007]. For the work in this chapter to be suitable for computer games two further steps should be taken in order to verify the methodology. Firstly, a commercial computer game should be selected and its environment abstracted so that the GP algorithm can be applied. Secondly, after the incorporation of the abstractions into a commercial game, human trials should be conducted.

The use of human trials would allow the verification of the method with the end-users of computer games. It is for the player's enjoyment and entertainment that computer games are developed and as such, any AI characters should help to enhance this quality. Conducting surveys is the traditional way that academic AI contributions are assessed from a human player's point of view and used effectively by Mac Namee [2004], for example.

The use of a commercial game allows for a comparison between implementation times of hand crafted AI agents versus automatically generated behaviours. Using predefined action sets, the performance of the GP algorithm could be evaluated against a human programmer. If human players are unable to discern the difference between the outputs of a human programmer and the automatically generated behaviours the approach could be deemed a success and a useful tool for large applications.

The use of a commercial game environment and human trials is beyond the scope of this thesis. The abstract environment defined in Chapter 4 was adequate for capturing the cooperation aspects of a traditional common pool resource dilemma. By introducing a spatial and temporal element into the abstract environment it helped move from the game theoretic domain into the computer game domain. This is shown to be true through the successful incorporation of the behaviours created in an abstract domain to the continuous environment. By introducing the notion of roles into the creation of agents, implicit relationships were built and cooperation and coordination behaviours were achieved within the group using a simple and limited action set.

6.6 Summary

In this chapter, a continuous environment which is designed to simulate a computer game environment is detailed. The environment displays and updates game objects like a computer game might, while remaining suitably simple to perform analysis on the performance of behaviours and to allow rapid prototyping of various environments. The abstract game environment from Chapter 4 is imported and represented in this continuous environment.

The mapping of behaviours automatically generated in the abstract game environment into the continuous environment is discussed. The incorporation of the abstract behaviours is relatively straightforward. A mapping is created for each of the actions that an agent can perform and a semantic parser extracts the GP tree from generated behaviours.

A selection of behaviours generated in the abstract environment are assessed in the continuous environment. In the abstract environment, agents execute actions in an abstracted unit of time called a tick. All actions have a fixed and uniform duration. This presents an immediate difficulty within the continuous environment, as actions may now have varying durations owing to the resource and building locations around the game world. What could be expressed as an equidistant game world in the abstract environment no longer holds.

This problem is alleviated by an exploration of a minimum action duration for the agents in the continuous environment. This is the minimum duration an action must be executing for until it is completed and removed from the agent's action queue. A range of values for the minimum duration are explored and are shown to improve the performance of the abstract behaviours in the continuous environment.

The abstract behaviours perform worse in the continuous environment than the environments in which they were created. In order to improve this, a series of approaches were introduced. Expert tweaks were used to alter the continuous environment based on inputs from the game designer. By removing the cost of moving around the continuous game world to the agent's health, the performances of the abstract behaviours increased. More stringent selection criteria were used when choosing candidate behaviours from the abstract environment. This approach also improved the abstract behaviours performance in the continuous environment and was most effective when combined with the expert tweaks.

A brief demonstration was then provided of the impacts that altering the continuous environment can have on the performance of the abstract behaviours. By simply changing the layout of the game world, the generated behaviour's can be altered. There is almost no restriction to the range of continuous environments that could be created within this game's framework, but each would have an impact on the performance of the abstract behaviours. This is especially true if the continuous environment differs greatly from the abstract environment in which the behaviours are created.

A selection of behaviours generated in dynamic environments are evaluated. Dynamic environments try to simulate the types of interactions that AI characters may experience if outside influences can act within the game environment. There is simulated trading, fighting and killing within the dynamic environments. The GP algorithm is shown to be capable of creating behaviours that can survive in most environments, even if the environment was not the one in which agents were created. The introduction of dynamic environments into the evolutionary evaluation of the behaviours was able to increase the group's performance in the continuous environment.

Finally, a general discussion of the method of creating behaviours in an abstract environment for computer games is provided. There are many advantages to creating behaviours in abstract environments, including the ability to more easily analyse the effects of actions and environmental variables. A selection of extensions to the evaluation and implementation of the current work are suggested which would help the method be more readily incorporated into a commercial game.

This chapter also explores the hypothesis that the study of agent behaviours in abstract economic dilemmas provide appropriate templates upon which to create groups of NPCs in computer games. Groups of cooperating and coordinating agents were created from a simple abstraction and applied successfully in the continuous computer game simulation. The dilemma, in some cases, could be tweaked slightly to increase the performance of the groups. Various approaches to behaviour selection demonstrated that abstract behaviours could be applied in this environment without the need for modification.

7. CONCLUSIONS

Chapter 7 provides a summary of the answers from the previous chapters to each of the research questions and hypotheses proposed in Chapter 1. In this chapter we first restate and answer the research questions. Subsequently, we examine each of the hypothesis and discuss to what extent the work in this thesis has shown them to be true or false. Finally, we discuss possible future work stemming from this research.

Chapter 1, introduces the domain of computer games and sets out the motivations for the work in this thesis. The costs of creating AI characters for computers is high and as such there is a need to automate as much of the process as possible. The area of background characters is the focus of this work. Background characters are often neglected parts of a game as they are seen as less important to the implementation of main storylines, graphics and other elements. We argue that, although background AI characters may not be key to the main story, they do add a level of believability of the game. Any nonsensical interaction a player has in the game will ultimately detract from the enjoyment of the game.

Several properties are identified as desirable for background characters. The first is the requirement for simple actions. Simple actions mean that anything that a background character performs should be relatively computationally inexpensive. This arises due to the fact that background characters may exist in large numbers in the game world. Following from this constraint is the requirement for a simple implementation of the agent's reasoning. Preferably, the background agents would use simple instructions which can be created and verified automatically. Finally, it is desirable to create interactions between the background characters and the game environment through the use of the simple actions.

Evolutionary computation and specifically, genetic programming, are presented as possible approaches to tackle the problem of creating an AI character's reasoning automatically. The regular use of decision trees in Game-AI implementations and the fact that GP uses a tree structure as it representation of solutions makes it an appropriate approach. Further, the ability to constrain the evolutionary processes like creation, crossover and mutation by using Strongly Typed GP meant that solutions, once created, would be in a correct format.

The research area of game theory is introduced and is presented as a possible source of inspiration for the scenarios potential background characters may find themselves. Game theory is the study groups used for modelling economic and social dilemma situations. Game theory demonstrates how individual decisions effect outcomes for both the individual and the group. Typically, game theoretic dilemmas are encapsulated as abstract games which require a limited number of decisions but model complex group dynamics.

Chapter 2, introduces the concepts dealt with in the thesis in more detail and expands on the related work conducted in the various research areas.

Chapter 3, introduces a common pool dilemma from the social science literature. A CPR dilemma involves a group sharing a resource to create an income. Typically, if individuals act in their own self-interest then the group does worse over all and if individuals sacrifice their own gain, the group does better. CPR dilemmas model many shared resources from real world scenarios e.g., a fishing grounds. An abstract game is chosen from previous studies that had both, a game-theoretic analysis and performed human trials.

The GP algorithm for application to this domain was then introduced. Solutions were created for individuals to make decisions as part of groups taking part in CPR dilemmas. The performance of the evolved agents was inline with game theoretic predictions about rational self-interested agents in the domain. The evolved behaviours were different from those of the humans, whose behaviours are varied and sub-rational. However, the evolved behaviours lose their ability to optimise their returns against a range of varied opponents.

To investigate the ability of the GP process to create solutions capable of playing different strategies against varying opponents, a set of experiments were devised which introduced irrational behaviours into the evolutionary evaluation. The GP evolved behaviours increased their performance against the varied opponents when compared to the co-evolved agents. These new behaviours, when played against each other, appeared to have human-like properties. The behaviours exhibited a high level of variance at an individual level and approached Nash equilibrium point at an aggregate level towards the end of the game.

Further studies with CPR dilemma investigate the effect of environmental pressures on the behaviours of the individuals and groups. An increase in uncertainty about the investment patterns of others in the group lead agents to have less confidence in the common resource. In scenarios where the CPR was destructible, the evolved agents were unable to extend the game by avoiding investing the resource which was similar to the human play. However, if there was a safe-zone, or an amount of investment in the pool which would not destroy the resource, the evolved agents could play to preserve the resource. This was contrary to the play of humans in the same game.

Chapter 4 introduced an abstract computer game world with some of the properties of traditional game theoretic dilemmas. The game involved groups of agents adopting roles and working together to harvest and refine raw resources to create a consumable food. Actions cost the agents health points which could be then replenished by eating the food.

In order to create food, several agents are required across different roles, with each contributing at some stage of the food production. Regardless of an agent's contribution, if there is food they could consume it. Food is a resource that is shared among the group which members of the group can choose to provide. This game expands on the abstract CPR dilemma concept by including extra constraints and features from a video game domain. Locations and roles are introduced to add extra complexity into the environment.

The introduction of roles into the game provides two things. Firstly, when deciding the composition of a group, the type of agents must be considered. Each role is necessary in order to have a successful group within the environment. This creates an implicit relationship between the members of the group and requires an amount of cooperation to create the food. Secondly, roles provide a way of limiting the number of actions an agent can do in both, the game and in the GP algorithm. This constraint helps to ensure the automatic creation of valid trees.

Chapter 4 outlined the GP algorithm and performance for a range of scenarios within the role-based game. A number of parameters are studied for different configurations of the game. The environment remains static as suitable game parameters, including group size and food replenishment value, are assessed. The GP algorithm is deemed suitable for creating groups in this environment. The group's performance in the environment is used to compare the evolved behaviours. An analysis of the evolved behaviours shows that, even though the environment is static and the GP parameters unchanging, the GP process can create several distinct groups for a given configuration.

Chapter 5 expands on the environment of the role-based game. Video games are examined for dynamic elements that effect the performance of AI characters. A selection of appropriate dynamic interactions are selected and applied in the role-based game. This serves to close the gap between the abstract environment and an application scenario in a computer game. This chapter puts the role-based game into a game context by simulating exchanges with other agents that AI characters may experience.

Attacks by other agents are simulated through experiments where members of the group lose health at different intervals. The group must be able to handle an agent requiring food after a sudden loss. The GP process must evolve agents who can create and preserve the food until it is required.

The GP process' ability to create groups with redundancy and balance is tested by introducing unexpected death into the game. An agent is killed at different intervals during the game and the effect is observed in the group. As all roles are needed within the game, if an agent dies their role could go unfulfilled to the detriment of all others within the group. In a computer game, the death of an agent should be seen to effect the rest of the group but should not mean that the group cannot survive or cope with the loss.

Trade is simulated by manipulating the stocks that the agents create. Rather the effect the agents directly, the outside influences buy or sell stocks changing their availability in the food creation process. By changing the stock levels, the agents will need to use the stock level information to make decisions about what their next action should be.

For all dynamic variations of the environment, the GP algorithm is able to create behaviours for groups to survive. By using dynamic environments in the evaluation during the GP process, the performances of groups is increased across a range of dynamic environments. Even if the group did not encounter the environment during creation, the behaviours created in dynamic environments perform better than those from static environments.

The group behaviours created have been shown to be robust to the abstracted computer game elements which the AI characters may have to encounter. If a dynamic element is likely to occur in a computer game it can be modelled and it may be possible for behaviours to be automatically created that survive the dynamic encounter. Even if an unpredicted dynamic element is encountered, by using some dynamic elements within the creation process, it increases the chance of success for the behaviours.

Chapter 6 describes a continuous environment that is implemented using techniques from computer game design. This environment implements the game world described in the abstract role-based game. In this chapter, the design of the continuous game is detailed including the changes required for the transition to a continuous environment.

Experiments are conducted to assess the performance of solutions created using an abstract environment and applied to a continuous one. Additional game features are needed to apply the abstract behaviours directly in the continuous environment, e.g., a minimum action duration for the agents. The minimum action duration tries to redress the balance of average action duration, which can vary due to locations of the agents in the game world and the availability of stocks.

Studies are conducted of how both, the elements of the abstract game and elements of the continuous environment effect the performance of the group. Varying the placement of game buildings in the world and thus varying the distances agents must travel and changing the length of action duration affects the performance of the abstract generated behaviours. How the abstract environment captures a particular game component, like the locations of buildings and resources, can influence the range of diversity that can be supported within a continuous environment.

7.1 Answers to the Research Questions

In Section 1.3, four research questions were formulated. This section provides an answer to each of these questions based on conclusions from previous chapters.

> **Research question 1:** Evolutionary computation can reveal rational strategies for agents in game theoretic dilemmas but often humans are irrational under the same circumstances. Can we create human like play without the input of humans in these game theoretic dilemmas?

The answer to the first research question is derived from Chapter 3. In the chapter, a game theoretic dilemma, with which human trials had been conducted, is introduced. An evolutionary computational approach is introduced to create behaviours for agents in this dilemma. Genetic Programming is shown to create rational behaviours as expected but differs from the play of the human subjects. The resulting behaviours had, however, lost their ability to react against strategies which may exploit their behaviour or which may be naively investing in the common pool market.

A series of experiments were conducted where by the GP strategies were evolved to play against a set of naive strategies. The resulting behaviours were diverse and did not converge on any one strategy while maintaining their ability to react to poor play by other agents. When a set of these new evolved behaviours played against each other the results were very similar to the play utilised by the human players.

The introduction of irrationality into the evolutionary evaluation caused the behaviours to appear human-like. Further studies of the factors which influenced the behaviours of the evolved strategies were then conducted. These studies included introducing varying levels of random uncertainty of the strategies of group members and environmental pressures caused by the probabilistic destruction of the shared resource. Genetic Programming is able to provide appropriate behaviours in each situation but does not always appear human like.

> **Research question 2:** If it is possible to create human-like in

> game theoretic dilemmas, can behaviours for groups of characters in computer games be automatically generated by modelling the features of their interactions with shared resource problems? Do they have the same properties as the game theoretic games?

This second research question is addressed in Chapter 4. In the chapter, a game environment is introduced in which groups of agents must adopt roles and harvest resources to create food and survive. Certain properties of the game theoretic games are maintained. The food resource that is created by the group can be consumed by any member of the group regardless of their contribution to its creation. This allows freeriding and exploitative behaviours within a group to exist.

A cooperative evolutionary solution is shown to create groups automatically which can survive in this environment. The members display both cooperation, as they contribute to the creation of the food resource, and coordination, as actions must occur in the correct order with other members' actions. If the environment is generous, that is, the food resource is abundant, freeriding by some members can be supported by a cooperative group. If too many agents perform exploitative behaviours, the defective strategies take over and the entire group tends towards exploitation.

This environment contains some aspects of game theoretic dilemmas by forcing individuals to consider between self interest and group aspirations. The environment also introduces extra coordination constraints, moving the game closer to an abstraction of a computer game.

> **Research question 3:** Can we automatically assign roles and behaviours to NPCs for games that require coordination and cooperation among roles? Can specific actions required for survival in individuals be evolved when creating behaviours from a group perspective? For the above scenarios, are the generated

> behaviours robust to environmental changes? How diverse are the generated behaviours?

The answer to the third research question is found in Chapter 5. The role-based game introduced in Chapter 4 is expanded upon, by including dynamic elements into the environment. Experiments with the group structure revealed that the GP process is able to assign roles within the group especially when redundancy is required in the group's composition.

Experiments simulating external influences on both the agent's health and the resources they create demonstrate the GP algorithm's ability to evolve specifically required individual characteristics from a group perspective. By evolving the group in the evolutionary process, we acknowledge the increase in tree complexity as the group size increases. By successfully searching for specific traits in group members, the GP process is shown to be suitable for creating group behaviours in more complex environments.

The dynamic elements introduced in this chapter test the group's ability to survive when there is environmental change. The effects of altering the agents' health, the resources and the group structure itself are demonstrated by the changes in the group's performance. The GP process is able to create behaviours to withstand the environmental changes. These various outside influences also change the diversity of the generated behaviours. By changing the group structure, diversity of roles is promoted. By changing the health of the agents or their resources, the behaviours of the groups is changed from the static environment.

> **Research question 4:** Can behaviours be generated in an abstract environment and applied effectively in a continuous game? Do game elements modelled in the abstract environment create the performances expected in the continuous environment?

The fourth research question is addressed in Chapter 6. A continuous environment is introduced which models the abstract role-based game using techniques from computer game engines. An exploration of the translations of solutions created in the abstract environment and applied in the continuous environment reveals that a number of extra parameters are required. The groups created in the abstract environment work in the continuous environment without any modification to the behaviours themselves. However, their performance in the continuous environment is reduced when compared to their performance in the abstract environment.

A number of extra selection criteria when choosing behaviours in the abstract environment show that the performance in the continuous environment can be increased. By incorporating judgements made by AI designers, poor performance by the abstract behaviours can be compensated for.

7.2 The Hypotheses

In this section, an answer is provided for each of the hypotheses put forward in Section 1.4. The answers are based on the answers to the research questions discussed in the previous section.

> **H1:** Genetic programming techniques are suitable for providing artificial intelligence solutions for groups of agents in computer games.

Throughout this thesis, the usefulness of GP has been demonstrated for creating group behaviours. In Chapter 3, behaviours for individuals with self-interest and group interest are evolved for agents playing in a group based common pool dilemma. GP was able to create behaviours that conformed to game theoretic predictions when searching for rational agents. The GP process was able to create play that appeared human-like when sub-rational behaviours were introduced into the evolutionary evaluation process.

In Chapters 4 and 5, groups composed of distinct individuals are evolved using cooperative techniques for a role-based game. The GP process was shown to be useful when creating behaviours from a group perspective with a number of internal and external pressures on the group. The behaviours which were evolved in an abstract environment were applied in a continuous computer game simulation. In Chapter 6, the results show that the performance of the agents in the continuous environment was an effective implementation of automatically generated game-AI.

> **H2:** Evolutionary computational approaches in game theory dilemmas yield human-like performance under certain circumstances. Human behaviours in games are typically suboptimal appearing at times random and reactionary to the environment, but tend to perform better than purely random behaviours.

In Chapter 3, if agents are co-evolved using a fitness function that rewards self-profit maximisation then the performance of the agents will conform to game theoretic predications for rational agents. By introducing sub-rational naive strategies into the evolutionary evaluation of the agents, varied and human behaviours were created. The agents shared the same characteristics that humans had in similar research, that is, their behaviours were varied at an individual and over the course of the game, approached Nash equilibrium at an aggregate level.

> **H3:** We can specify a class of game which captures cooperation and coordination and using evolutionary techniques find solutions for them.

Chapter 4 introduces an abstract role-based common pool dilemma that extends traditional provision CPR dilemmas with elements of computer game environments. Agents must adopt a role, create and modify resources at a cost to their health in order to make food to sustain their lives. This game requires cooperation across the group as multiple agents are required to create the food. Coordination is required to synchronise the creation and modification of resources as well as, the sharing of the food to which all group members have access. GP is shown to be able to create group behaviours in this game with a number of internal and external pressures acting on the group.

> **H4:** The introduction of dynamic elements into the evolutionary process allows for the creation of more robust behaviours across a range of changing environments.

In Chapter 5, several dynamic elements are introduced into the abstract game environment designed to simulate typical interactions that an AI character from a computer game may experience. The GP algorithm is tasked with creating specific individual behavioural traits within the group behaviours in order to survive the environment. It is shown that the introduction of certain dynamic elements into the evolutionary evaluation of solutions increases the performances of groups across a number of environments.

> **H5:** The study of agent behaviours in abstract economic dilemmas provide appropriate templates upon which to create groups of NPCs in computer games. These dilemmas, if they require extension, exhibit similar behavioural properties in the computer game domain as they did in their original context.

In Chapter 6, behaviours created in an abstract CPR dilemma inspired by a game from the game theory literature are shown to be effective when applied in a computer game simulation. The abstract game is shown to encapsulate cooperation and coordination in the group behaviours. This game can be mapped into a game environment with minor tweaks. By using appropriate selection techniques to chose candidate solutions from the abstract environments, the behaviours of the groups in the continuous environment are effective in a number of scenarios.

7.3 Future Work

There are a number of open research questions that stem from the research outlined in this dissertation.

GP and CPR Dilemmas There are several strands on which to expand the application of GP to CPR dilemmas. CPR dilemmas are group based by nature and as such the representation of the group plays an important role. The structure of representing groups could be explored and compared in terms of efficiency of evolution and of the performances of groups. The performance of individuals depends on the evolutionary evaluation. In this research, several scenarios were explored including naive group members, random group members and environmental pressures. Other scenarios could be included such as the ability to communicate and the ability to police the commons. Additionally, combinations of scenarios in the evaluation stage, may provide a suitable mechanism to create individuals who play successfully against a varied set of opponents.

Co-evolution and Role Switching In the abstract computer game environment defined, agents adopt a single role once and never change. This restricts the complexity of the search and allows for a comprehensive study to be conducted. It does, however, limit the agents in two main ways. Firstly, the group's future is vulnerable, especially if agents can die. Allowing agents the ability to adopt new roles throughout the game could lead to more robust groups. Secondly, because the group chooses a role for the agent and doesn't allow agents to choose their role-based on the group they are in, the usefulness of a single agent is reduced. Agents cannot integrate themselves into foreign groups successfully. The method of evolution is more constrained, as the agent behaviour generated is specific rather than general.

Human Evaluation Ultimately, computer games are designed as entertainment and as such, should be evaluated through human trials and user experience testing. Any automatic programming technique for application in the game-AI domain will only be useful if end users cannot differentiate the generated behaviours from hand-crafted ones. Beyond this evaluation, the structure of the groups and the behaviours of the individuals within it are only as good as the setting they are applied in when considering from a computer game point of view. The application of the role-based game in an existing commercial offering would allow direct comparison with state of the art AI techniques. The introduction of the implicit relationships between the agents using the role-based game could be evaluated in terms of player interaction, believability and enjoyment.

APPENDIX

A. EVOLUTION OF GROUPS FOR AN ABSTRACT GAME

In this section, additional experimental runs assessing the performance of the GP algorithm in the static abstract game environment are presented. These experiments, while not supporting the central hypotheses of this research, nonetheless offer insight into the applicability of GP to this environment.

A.1 Limiting the Action Queue

Results are presented of the effects of varying the multiplier of the food variable in limited action queue environments. In Figure A.1, the effect of increasing the number of units of food without increasing the total replenishment is demonstrated on the average agent lifetime in the limited action queue environment. With three units of less valuable food compared to one more valuable, the agents live over twice as long on average overall.

In Figure A.2, the effects of the increased units of food on the survival rate of the groups is demonstrated. In this configuration, increasing the number of units of food without increase the total replenishment has the effect of increasing the average agent lifetime. This is especially affective in the environment with the limit placed on the action queue, which is the worst performing set of group behaviours.

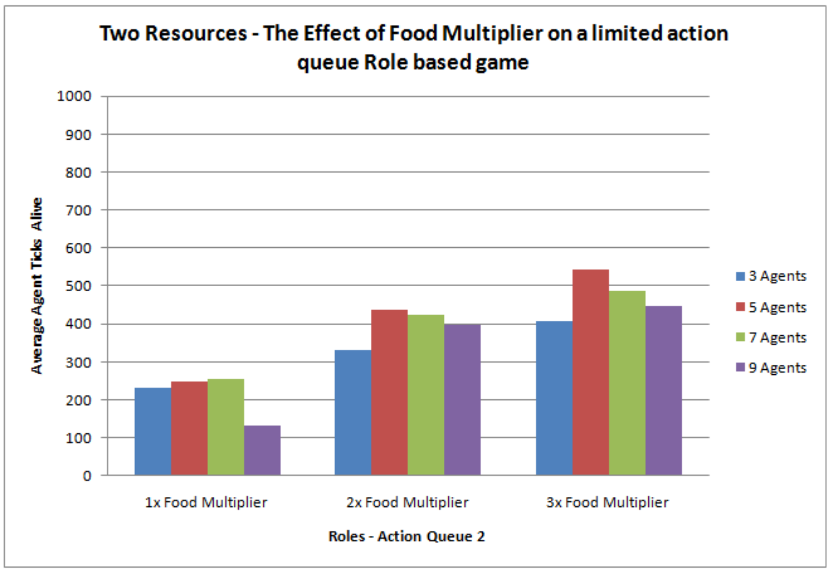


Fig. A.1: Two Resources - Role Based Evolution with Limited Action Queue - The Effects of Food Multiplier on Average Agent Lifetime

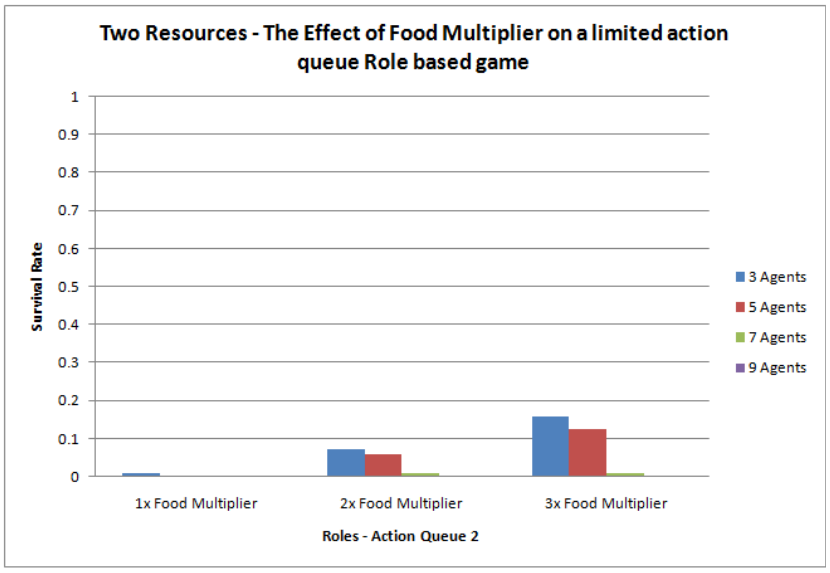


Fig. A.2: Two Resources - Role Based Evolution with Limited Action Queue - The Effects of Food Multiplier on Group Survival

The results show that by increasing the number of units of food created, it increases the survival rate of groups created with limited action queues. The gains in this environment when increasing the units of food are very large relative to the poor performance of the behaviours when there is only a single unit of food.

A.2 Previous Actions

In Sections 4.4.1 and A.1, the role based configuration with an action queue of constrained length was the worst performing configuration. In order to aid the agents in making decisions in this restricted setup, another node was added. This node allows an agent to make decisions based on their last action.

For the agents, the addition of the ability to reason using their previous action is implemented simply by modifying the environment nodeset for the agents which are detailed in Table 4.3. By adding a *Last Action* node to this set, the agent's previous action can be taken into account when making decisions. The reason for the addition of this node is to evaluate whether enhancing the limited reasoning that the agents can possess would yield better group performances.

The game configurations from Section 4.4.1 are used for the analysis. The best individual is taken from the final generation of twenty evolutionary runs conducted for each configuration. The the average performance of these best individuals is used for comparison between the configurations. The results were as follows:

- When comparing the lifetime of agents across group size, in the restricted queue game there was a 5% improvement with a 4% standard deviation. In the unrestricted game, there is less than a 1% difference between including previous actions and omitting them.

- The greatest increase to average agent lifetime was for the 3x multiplier of food when the group size was 7 or 9 (13% and 12% respectively).
- The survival rates in the unrestricted games are unaffected by incorporating the previous actions nodeset. In the restricted game, there is a 14% increase in survival rates when averaged across group size.

Figure A.3 presents data illustrating the various configurations of the role based evolution and the effect they have on the average agent life time. It should be noted, that all the 1x configurations are run over identical parameter sets but should not be directly compared to the 2x or 3x games. Instead, a these should be used as merely a guide for the trend of actual behaviour of the configurations. Figure A.4 shows the survival rates for the agents in the same configurations. Once again, the 1x, 2x and 3x configurations must be compared individually.

Adding the ability for agents to take their last action into account did not improve the average agent life or survival rates of the groups overall. When there is a limited action queue and multiple units of food, there is a non-significant increase in performance in some cases. The conclusion from this experiment is that providing the agents with the ability to sense their previous action does not improve the performance of the groups.

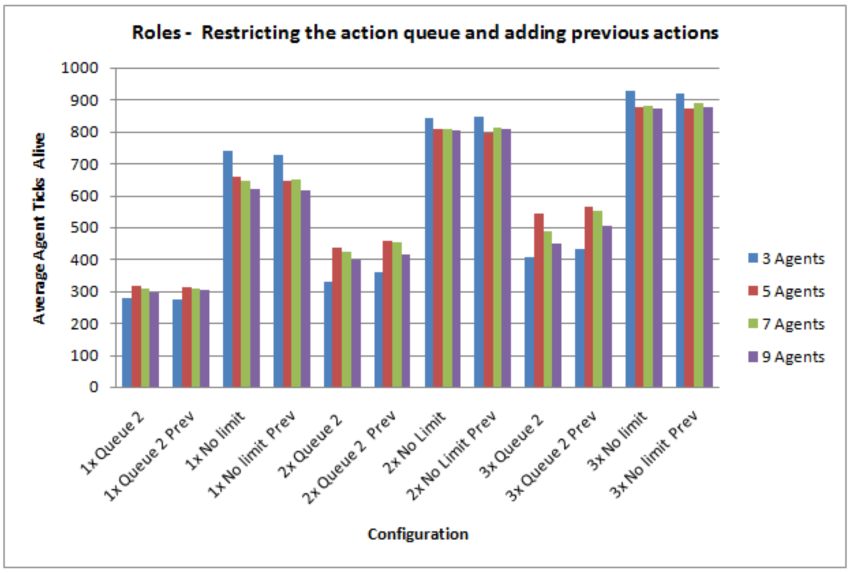


Fig. A.3: Two Resources - Role Based Evolution - Average Agent Lifetime across various configurations

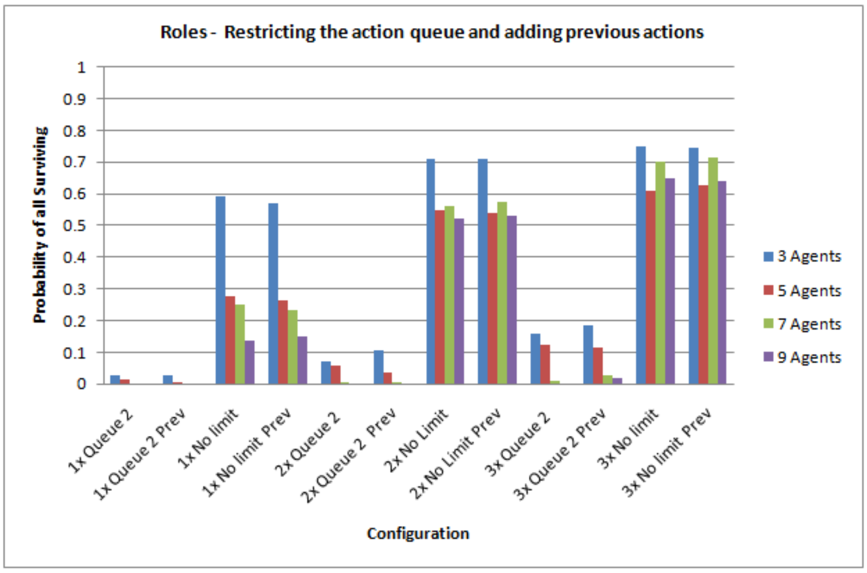
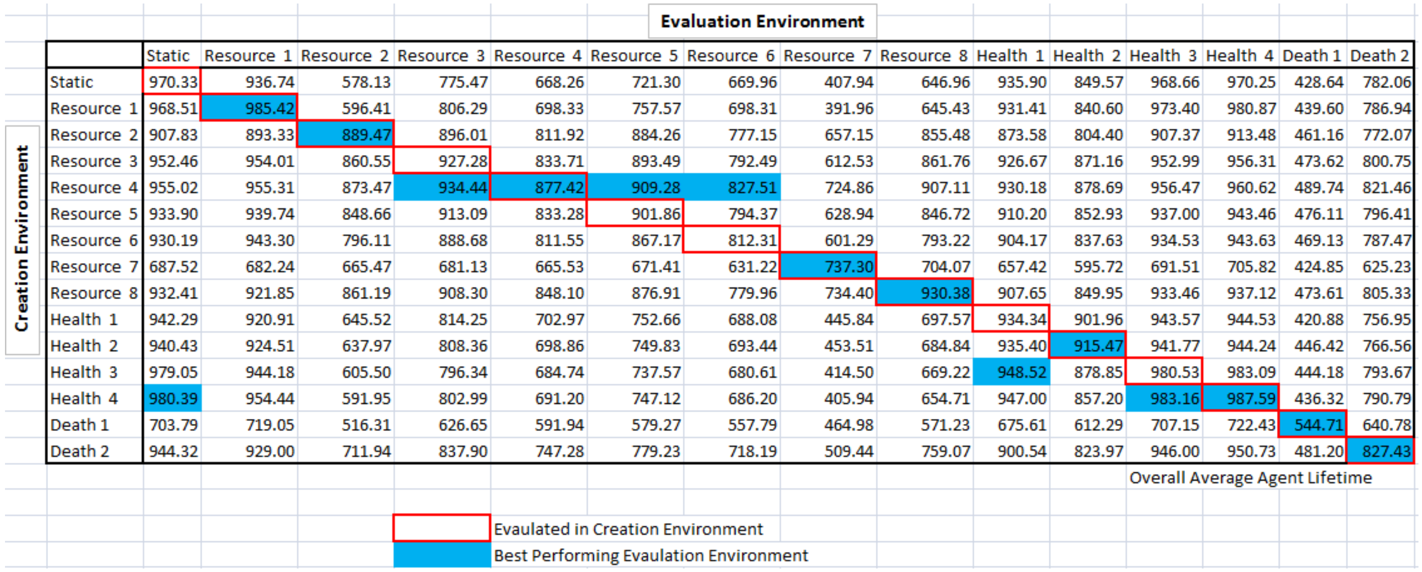


Fig. A.4: Two Resources - Role Based Evolution - Average Agent Lifetime across various configurations

B. EVOLVING GROUPS IN DYNAMIC CPR GAME ENVIRONMENTS

Figure B.1 displays the overall results of the round robin experiments with dynamic environments for the average agent lifetime.



Evaluation Environment

Creation Environment	Static	Resource 1	Resource 2	Resource 3	Resource 4	Resource 5	Resource 6	Resource 7	Resource 8	Health 1	Health 2	Health 3	Health 4	Death 1	Death 2
Static	970.33	936.74	578.13	775.47	668.26	721.30	669.96	407.94	646.96	935.90	849.57	968.66	970.25	428.64	782.06
Resource 1	968.51	985.42	596.41	806.29	698.33	757.57	698.31	391.96	645.43	931.41	840.60	973.40	980.87	439.60	786.94
Resource 2	907.83	893.33	889.47	896.01	811.92	884.26	777.15	657.15	855.48	873.58	804.40	907.37	913.48	461.16	772.07
Resource 3	952.46	954.01	860.55	927.28	833.71	893.49	792.49	612.53	861.76	926.67	871.16	952.99	956.31	473.62	800.75
Resource 4	955.02	955.31	873.47	934.44	877.42	909.28	827.51	724.86	907.11	930.18	878.69	956.47	960.62	489.74	821.46
Resource 5	933.90	939.74	848.66	913.09	833.28	901.86	794.37	628.94	846.72	910.20	852.93	937.00	943.46	476.11	796.41
Resource 6	930.19	943.30	796.11	888.68	811.55	867.17	812.31	601.29	793.22	904.17	837.63	934.53	943.63	469.13	787.47
Resource 7	687.52	682.24	665.47	681.13	665.53	671.41	631.22	737.30	704.07	657.42	595.72	691.51	705.82	424.85	625.23
Resource 8	932.41	921.85	861.19	908.30	848.10	876.91	779.96	734.40	930.38	907.65	849.95	933.46	937.12	473.61	805.33
Health 1	942.29	920.91	645.52	814.25	702.97	752.66	688.08	445.84	697.57	934.34	901.96	943.57	944.53	420.88	756.95
Health 2	940.43	924.51	637.97	808.36	698.86	749.83	693.44	453.51	684.84	935.40	915.47	941.77	944.24	446.42	766.56
Health 3	979.05	944.18	605.50	796.34	684.74	737.57	680.61	414.50	669.22	948.52	878.85	980.53	983.09	444.18	793.67
Health 4	980.39	954.44	591.95	802.99	691.20	747.12	686.20	405.94	654.71	947.00	857.20	983.16	987.59	436.32	790.79
Death 1	703.79	719.05	516.31	626.65	591.94	579.27	557.79	464.98	571.23	675.61	612.29	707.15	722.43	544.71	640.78
Death 2	944.32	929.00	711.94	837.90	747.28	779.23	718.19	509.44	759.07	900.54	823.97	946.00	950.73	481.20	827.43

Overall Average Agent Lifetime

▭	Evaulated in Creation Environment
🟦	Best Performing Evaulation Environment

Fig. B.1: Abstract Dynamic Environments - Overall Average Agent Lifetime Round Robin Results

BIBLIOGRAPHY

L.V. Allis. *A Knowledge-based Approach of Connect-four: The Game is Solved: White Wins.* Technical reports in computer science. University of Limburg, Department of Computer Science, 1992.

L.V. Allis, H.J. Van den Herik, and M.P.H Huntjens. Go-moku solved by new search techniques. *Computational Intelligence*, 12(1):7–23, 1996.

W.B. Arthur. Inductive reasoning and bounded rationality. *The American Economic Review*, 84(2):pp. 406–411, 1994.

Atari. Pong, published by atari, 1972.

R. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.

S. Bakkes. *Rapid Adaption of Video Game-AI.* PhD thesis, Universiteit de Tilburg, Netherlands, 2010.

Bioware. Neverwinter nights, published by bioware, game website: `http://nwn.bioware.com/`, 2002.

Blizzard Entertainment. Starcraft, published by blizzard entertainment, game website: `http://us.blizzard.com/en-us/games/sc/`, 1998.

Blizzard Entertainment. World of Warcraft, published by blizzard entertainment, game website: `http://eu.battle.net/wow/en/`, 2004.

P. Bonacich, G.H. Shure, J.P. Kahan, and R.J. Meeker. Co- operation and group size in the n-person prisoners' dilemma. *Journal of Conflict Resolution*, 20:687–706, 1976.

F. Bousquet, R. Lifran, M. Tidball, S. Thoyer, and M Antona. Agent-based modelling, game theory and natural resource management issues. *Journal of artificial societies and social simulation*, 2001.

D. Braben and I. Bell. Elite, published by acornsoft, game website: `http://www.iancgbell.clara.net/elite/`, 1984.

S.J. Brams, Morton D.D., and P.D. Straffin Jr. The geometry of the arms race. *International Studies Quarterly*, 23(4):pp. 567–588, 1979.

M. Buckland. *Programming Game-AI by Example*. Jones & Bartlett Publishers, 1 edition, September 2004.

Bungie Studios. Halo 2, published by microsoft game studios, game website: `http://www.bungie.net/Projects/Halo2/`, 2004.

Capcom. Street Fighter, published by capcom, game website: `http://www.streetfighter.com/`, 1987.

D. Challet and Y. Zhang. On the minority game: Analytical and numerical studies. *Physica A: Statistical and Theoretical Physics*, 256(3-4):514 – 532, 1998.

A.J. Champandard. *AI Game Development*. New Riders Games, 2003.

V. Ciesielski, D. Mawhinney, and P. Wilson. Genetic programming for robot soccer. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Computer Science*, pages 37–55. Springer Berlin / Heidelberg, 2002.

Codemasters. Colin McCrae Rally 2.0, published by codemasters, game website: `http://www.codemasters.co.uk/`, 2004.

D.S. Cohen. Cathode-ray tube amusement device - the first electronic game, 2012. URL `goo.gl/oPqRTW`.

N. Cole, S.J. Louis, and C. Miles. Using a genetic algorithm to tune first-person shooter bots. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 139 – 145 Vol.1, june 2004.

C. Collin and S. Eglen. An introduction to natural computation. *Trends in Cognitive Sciences*, 2(6):235 – 236, 1998.

C. Conati. Probabilistic assessment of users emotions in educational games. *Applied Artificial Intelligence*, 16:555–575, 2002.

J.C. Cox, V.L. Smith, and J.M. Walker. Theory and behavior of multiple unit discriminative auctions. *The Journal of Finance*, 39:983–1010, September 1984.

J.C. Cox, V.L. Smith, and J.M. Walker. Theory and individual behavior of first-price auctions. *Journal of Risk and Uncertainty*, 1:61–99, 1988.

Creature Labs. Creatures, published by creature labs, game website: `http://www.gamewaredevelopment.co.uk/creatures_index.php`, 1996.

M. Cutumisu, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko, and M. Carbonaro. Generating ambient behaviors in computer role-playing games. *IEEE Intelligent Systems*, 21(5):19–27, October 2006.

C.J. Darken and J.D. Kelly. *Individualized NPC Attitudes with Social Networks*. AI Game Programming Wisdom 4. Charles River Media, 2008.

P. J. Deadman. Modelling individual behaviour and group performance in an intelligent agent-based simulation of the tragedy of the commons. *Journal of Environmental Management*, 56(3):159–172, 1999.

M.A. DeLoura. *Game Programming Gems 2*. Charles River Media, Inc, 2001.

E.W. Dijkstra. Ta note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

D. Doherty. *Evolving Tactical Teams for Shooter Games using Genetic Programming*. PhD thesis, National University of Ireland, Galway, 2009.

D. Doherty and C. O'Riordan. Evolving agent-based team tactics for combative computer games. *17th Irish Artificial Intelligence and Cognitive Science Conference (AICS)*, 2006a.

D. Doherty and C. O'Riordan. Evolving tactical behaviours for teams of agents in single player action games. *In CGAMES 2006 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games*, 2006b.

EA. The Sims, published by electronic arts, game website: `http://thesims.ea.com/`, 2000.

EA. Black & White, published by electronic arts., game website: `http://www.lionhead.com/bw/`, 2001.

T. Ehlis. Application of genetic programming to the "snake game". *Gamedev.Net*, (175), 2000.

Ensemble Studio and Big Huge Games and Robot Entertainment. Age of Empires, published by microsoft studios, 1997.

J. Epstein and R. Axtell. *Growing Artificial Societies, Social Science from the Bottom Up*. Brookins Institution Press/ The MIT Press, 1996.

E. Eskin and E. Siegel. Genetic programming applied to othello: introducing students to machine learning research. *SIGCSE Bull.*, 31(1):242–246, March 1999.

C. Fairclough, M. Fagan, B. Mac Namee, and P. Cunningham. Research directions for ai in computer games. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence & Cognitive Science*, pages 333–344, 2001.

E.G. Furubotn and S. Pejovich. Property rights and economic theory: A survey of recent literature. *Journal of Economic Literature*, 10(4):1137–62, 1972.

R. Gasser. Solving Nine Men's Morris. *Computational Intelligence*, 12:24–41, 1996.

T. T. Goldsmith Jr., C. Grove, and E. R. Mann. Cathode-ray tube amusement device usa patent#2,455,992, 1948.

H.S. Gordon. The economic theory of a common-property resource: The fishery. *The Journal of Political Economy*, 62(2):124–142, 1954.

G. W. Greenwood. Deceptive strategies for the evolutionary minority game. In *Proceedings of the 5th international conference on Computational Intelligence and Games, Milano, Italy*, pages 25–31, 2009.

L. Gruenwoldt, S. Danton, and M. Katchabaw. Creating reactive non player character artificial intelligence in modern video games. *In Proceedings of the 2005 Game-On North America Conference*, 2005.

G. Hardin. Tragedy of the commons. *Science*, 162:1243–1248, 1968.

D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

P.E. Hart, N.J. Nilsson, and B. Raphael. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Bull.*, (37): 28–29, 1972.

A. Hauptman. Gp-endchess: Using genetic programming to evolve chess endgame players. In *Proceedings of 8th European Conference on Genetic Programming (EuroGP2005*, pages 120–131. Springer, 2005.

Havok. Havok Physics `Http://www.havok.com/content/view/17/30/`, 1998.

T. Haynes and S. Sen. Crossover operators for evolving a team. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 162–167, 1997.

T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright. Evolving multiagent coordination strategies with genetic programming. Technical report, Proc. Artificial Intelligence, 1995a.

T. Haynes, R. Wainwright, and S. Sen. Evolving cooperation strategies. In V. Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, San Francisco, CA, 1995b. MIT Press.

T. Haynes, R. Wainwright, S. Sen, I. Sen, and D. Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 271–278. Morgan Kaufmann Publishers, Inc, 1995c.

J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 2nd edition, 1992.

E. Hughes. Computer games rise up, November 2008. URL `goo.gl/FyDbDk`.

id Software. Quake, published by gt interactive, game website: `http://www.idsoftware.com/games/quake/quake`, 1996.

id Software. Quake II, published by activision, game website: `http://www.idsoftware.com/games/quake/quake2`, 1997.

id Software. Quake III Arena, published by activision, game website: `http://www.idsoftware.com/games/quake/quake3-arena`, 1999.

W. Jager, M.A. Janssen, and C. A. J. Vlek. How uncertainty stimulates over-harvesting in a resource dilemma: Three process explanations. *Journal of Environmental Psychology*, 22:247–263, 2002.

M. Kilgard. GLUT, opengl utility toolkit, website: `http://www.opengl.org/resources/libraries/glut/`, 2012.

P. Kollock. Social dilemmas: the anatomy of cooperation. *Annual Review of Sociology*, 24:183–214, 1998.

J.R. Koza. *Genetic Programming, On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

M. Lalena. Teamwork in genetic programming. Master's thesis, Rochester Institute of Technology, 1997.

F.D. Laramée. *Advanced Genetic Programming: New Lessons from Biology.* AI Game Programming Wisdom 2. Charles River Media, 2004.

R. Laursen and D. Nielsen. Investigating small scale combat situations in real-time strategy computer games. Master's thesis, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 2005.

L. Lidèn. *Artificial Stupidity: The art of making intentional mistakes.* AI Game Programming Wisdom 2. Charles River Media, Inc., 2004.

D. Loiacono, J. Togelius, P.L. Lanzi, L. Kinnaird-Heether, S.M. Lucas, M. Simmerson, D. Perez, R.G. Reynolds, and Y. Saez. The wcci 2008 simulated car racing competition. *in Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

LucasArts. Sam & Max Hit the Road, published by lucasarts, 1993.

LucasArts. Grim Fandango, published by lucasarts, 1998.

S. Luke. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 214–222, University of Wisconsin, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann.

S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 150–156, Cambridge, MA, USA, 1996. MIT Press.

B. Mac Namee. *Proactive Persistent Agents - Using Situational Intelligence to Create Support Characters in Character-Centric Computer Games.* PhD thesis, University of Dublin, Trinity College, 2004.

M.W. Macy and A. Flache. Learning dynamics in social dilemmas. *Adaptive Agents, Intelligence, and Emergent Human Organization: Capturing Complexity through Agent-Based Modeling, PNAS*, 2002.

M. McNaughton, J. Redford, J. Schaeffer, and D. Szafron. Pattern-based ai scripting using scriptease. *In Proceedings of the 16th Canadian Conference on Artificial Intelligence*, pages 35–49, 2003.

M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. Script-ease: Generative design patterns for computer role-playing games. *Proceedings of the 19th IEEE Conference on Automated Software Engineering (ASE 2004)*, pages 88–99, 2004.

MicroProse. Sid Meier's Civilization, published by microprose, 1991.

Microsoft. Microsoft xbox console, website: `http://www.xbox.com/`, 2001.

G. Millington. *Artificial Intelligence for Games*. Morgan Kaufmann., 2nd edition, 2009.

N. Minar, R. Burkhart, and C. Langton. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations, 1996.

Monolith. No one lives forever 2: A Spy in H.A.R.M.'s Way, monolith productions/sierra entertainment inc., 2002.

D.J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3:199–230, 1994.

T. Murata and T. Nakamura. Multi-agent cooperation using genetic network programming with automatically defined groups. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation GECCO 2004*, volume 3103 of *Lecture Notes in Computer Science*, pages 712–714. Springer Berlin / Heidelberg, 2004.

T. Murata and T. Nakamura. Genetic network programming with automatically defined groups for assigning proper roles to multiple agents. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1705–1712, New York, NY, USA, 2005. ACM.

Namco. Mrs. PacMan, Published by Bally/Midway/Namco, 1981.

A. Nareyek. Ai in computer games. *Queue*, 1(10):58–65, 2004a.

A. Nareyek. Computer games - boon or bane for ai research? *KI*, 18(1): 43–44, 2004b.

J.F. Nash. Equilibrium Points in N-Person Games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, January 1950.

Nintendo. Super Mario Bros., published by nintendo, game website: `http://mario.nintendo.com/`, 1985.

Nvidia. PhysX SDK. `Http://www.nvidia.com/object/nvidia_physx`, 2008.

J. Orkin and J.D. Kelly. *Simple Techniques for Coordinated Behaviour*. In AI Game Programming Wisdom 2. Charles River Media Inc., 2004.

M.J. Osborne. *An Introduction to Game Theory*. Oxford University Press, USA, August 2003.

E. Ostrom, R. Gardner, and J. Walker. *Rules, Games and Common-Pool Problems*. Michigan, 1994.

L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

S. Raik and B. Durnota. The evolution of sporting strategies. In *Complex Systems: Mechanisms of Adaption, 8592. IOS*, pages 85–92. Press, 1994.

A. Rapoport and A.M. Chammah. *Prisoner's dilemma A study in conflict and cooperation*. University of Michigan Press, Ann Arbor :, 1965.

Remedy. Max Payne, published by rockstar games, game website: `http://www.rockstargames.com/maxpayne/main.html`, 2001.

C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, July 1987.

S. Russell. Spacewar! website: `goo.gl/vLfuaw`, 1962.

J. Schaeffer. A gamut of games. *Artificial Intelligence Magazine*, 22(3):29–46, 2001.

M. Schlüter and C. Pahl-Wostl. Mechanisms of resilience in common-pool resource management systems: an agent-based model of water use in a river basin. *M. Ecology and Society*, 12(2):1137–62, 2007.

S. Sharabi and M. Sipper. Gp-sumo: Using genetic programming to evolve sumobots. *Genetic Programming and Evolvable Machines*, 7:211–230, 2006.

Y. Shichel, E. Ziserman, and M. Sipper. Gp-robocode: Using genetic programming to evolve robocode players. In *Proceedings of 8th European Conference on Genetic Programming (EUROGP2005)*, pages 143–154. SpringerVerlag, 2005.

M. Shor. Nash Equilibrium Definition, Dictionary of Game Theory Terms, 2005. URL `http://www.gametheory.net/dictionary/NashEquilibrium.html`.

E. V. Siegel and A. D. Chaffee. Genetically optimizing the speed of programs evolved to play tetris. In P.J. Angeline and K.E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 14, pages 279–298. MIT Press, Cambridge, MA, USA, 1996.

Sierra. F.E.A.R. First Encounter Assault Recon, published by sierra studios, game website: `http://www.whatisfear.com/us/`, 2002.

Sierra. Empire Earth III, published by sierra entertainment, game website: `http://www.empireearth.com/`, 2007.

S. Slater. Essential elements of immersive game play. In *GAME-ON*, 2002.

P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Difficulty scaling of game ai. *In: Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON)*, pages 33–37, 2004.

P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game-ai with dynamic scripting. *Machine Learning*, 63(3):217–248, June 2006.

N.A. Steins and V.M. Edwards. Synthesis: Platforms for collective action in multiple-use common-pool resources. *Agriculture and Human Values*, 16: 309–315, 1999.

THQ. S.T.A.L.K.E.R.: Shadow of Chernobyl, published by thq, game website:`http://www.stalker-videogame.com`, 2007.

C. Thurau, G. Sagere, and C. Bauckhage. Imitation Learning at All Levels of Game AI. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education*, 2004.

J. Togelius, S.M. Lucas, H. Duc Thang, J.M. Garibaldi, T. Nakashima, C.H. Tan, I. Elhanany, S. Berant, P. Hingston, R.M. MacCallum, T. Haferlach, A. Gowrisankar, and P. Burrow. The 2007 ieee cec simulated car racing competition. *Genetic Programming and Evolvable Machines*, 2008.

P. Tozour. *The Perils of AI Scripting.* AI Game Programming Wisdom. Charles River Media, 2002a.

P. Tozour. *AI Game Programming Wisdom*, chapter The evolution of game AI. Charles River Media, Inc., 2002b.

Valve. Half-Life, published by sierra studios, game website: `http://www.sierra.com/`, 1998.

Valve. Half-Life 2, published by sierra entertainment, game website: `http://www.valvesoftware.com/games/hl2.html`, 2004.

Valve. Portal, published by valve, 2007.

H.J. Van den Herik, J.W.H.M. Uiterwijk, and Van Rijswijck J. Games solved: Now and in the future. *Artificial Intelligence*, 134(12):277 – 311, 2002.

Vivendi. Empire Earth II, published by vivendi universal, game website: `http://empireearth.sierra.com/`, 2005.

L.M. Wahl and M.A. Nowak. The continuous prisoner's dilemma: I. linear reactive strategies. *Theoretical Biology*, 200:307–321, 1999.

D. Winter. Noughts and crosses - the oldest graphical computer game, 2013. URL `http://www.pong-story.com/1952.htm`.

G.N. Yannakakis. Game ai revisited. In *Proceedings of ACM Computing Frontier Conference*, 2012.

G.N. Yannakakis and J. Hallam. Evolving opponents for interesting interactive computer games. In *Proceedings 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB'04)*, pages 499–508, 2004a.

G.N. Yannakakis and J. Hallam. Interactive opponents generate interesting games. In *International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 240–247, 2004b.

G.N. Yannakakis and J. Hallam. Towards optimizing entertainment in computer games. *Appl. Artif. Intell.*, 21(10):933–971, 2007.