A Phenotypic Analysis of GP-Evolved Team Behaviours

Darren Doherty
Department of Information Technology
National University of Ireland Galway
Galway, Ireland
darren.doherty@nuigalway.ie

Colm O'Riordan
Department of Information Technology
National University of Ireland Galway
Galway, Ireland
colm.oriordan@nuigalway.ie

ABSTRACT

This paper presents an approach to analyse the behaviours of teams of autonomous agents who work together to achieve a common goal. The agents in a team are evolved together using a genetic programming (GP) [8] approach where each team of agents is represented as a single GP tree or chromosome. A number of such teams are evolved and their behaviours analysed in an attempt to identify combinations of individual agent behaviours that constitute good (or bad) team behaviour. For each team we simulate a number of games and periodically capture the agents' behavioural information from the gaming environment during each simulation. This information is stored in a series of status records that can be later analysed. We compare and contrast the behaviours of agents in the evolved teams to see if there is a correlation between a team's performance (fitness score) and the combined behaviours of the team's agents. This approach could also be applied to other GP-evolved teams in different domains.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and Coordination, Multiagent Systems* ; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program Synthesis, Program Verification*

General Terms

Experimentation

Keywords

Genetic programming, phenotypic analysis, artificial intelligence, cooperative agents, tactical team behaviour

1. INTRODUCTION

Combative computer games are a specific genre of computer games where two sets of agents are competing in a hostile environment with the primary goal of eliminating the opposition. These games usually take the form of "shoot-em ups" as the agents use some form of projectile weapon to attack their enemies.

The artificial intelligence (AI) used in computer games to control the non-playable characters[1] is often implemented using deterministic techniques such as finite state machines (FSMs) and scripting as these are easy to understand and implement [11]. However, these techniques lead to predictability in the game AI which is not desirable in modern games as the behaviour of the non-playable characters (NPCs) appears unrealistic and thus reduces the game's playability.

For combative computer games, the AI tends to be squad based as each set of agents share a common goal (i.e. to eliminate the other team). The AI in these games can be viewed as being tactical as the actions of each team can be defined as a set of methods used for winning a small-scale conflict against the opposing team. To imitate a tactical behaviour however, a broad understanding of the situation is needed [12]; thus it is very difficult for game developers to manually code the tactics for the NPCs so that their combined group behaviour is coordinated and effective.

Genetic programming has been shown to be successful at evolving behaviours for teams of agents in a number of different domains. Luke et al. used genetic programming to evolve coordination between teams of agents in a virtual soccer environment [9] and Richards et al. successfully evolved cooperative strategies for a group of unmanned aerial vehicles (UAVs) using GP techniques [10]. These examples provide evidence that GP can be used as a effective technique for the automated development of rational team behaviours in different application domains.

Doherty and O'Riordan [4, 5] created an architecture for developing effective team tactics using genetic programming techniques. The evolved team behaviours that emerged from the simulations outperformed those of a team of intelligent, individually rational, manually coded agents developed by a game designer. In this paper, we analyse the phenotypes of these emergent team tactics and compare and contrast the behaviours of the individual members of the teams in an attempt to extract a set of features inherent in teams that perform well and see what combination of individual agent behaviours constitute a good tactical team behaviour. We also wish to analyse the less fit teams to determine what fac-

[1]Non-playable characters, or NPCs for short, are any characters in the gaming environment that are controlled by the computer.

GECCO'07, July 7-11, 2007, London, England, United Kingdom.

tors lead to their poorer performance. We identify a number of different behavioural types commonly found in the combative gaming environment. For each agent we monitor their behaviours and identify if their behaviour describes one or more of these common behavioural types. This results in a vector of probabilities representing the likelihood that an agent's behaviour adheres to the common types. The probability vectors of all agents on a team are combined into a single team vector that can be compared with other team vectors to analyse the behaviours of different teams. For an in depth discussion of team performance assessment and measurement see [2].

2. DEVELOPMENT

In [5] it is shown that a number of teams of game agents can be successfully evolved to defeat a single intelligent agent with infinite ammunition[2] and a health level equivalent to that of the entire team of agents. The single intelligent agent here can be thought of as the human player in a single-player game. Thus, the tactics evolved should be effective for use by teams of enemy NPCs in single player combative computer games. Each team consists of five agents that must work together as a collective group and display tactical team behaviour in order to outwit and overcome the single intelligent enemy agent. As a control for these experiments, a team of five individually rational agents were pitted against the same single enemy agent where all agents have the same reasoning abilities. In this control experiment, the team of five only defeat the single, more powerful agent 225 times out of 500 due to their individually rational behaviour, whereas the genetic program managed to evolve a team that can defeat the single agent 495 times out of 500. This paper focuses on extending their work by analysing the resulting behaviours of each of these evolved teams to ascertain which combinations of individual agent behaviours within a team constitutes a good (or bad) tactical team behaviour.

2.1 Simulation Environment

The simulator is built on the 2D *Raven* game engine created by Matt Buckland [3]. The environment consists of an open 2-dimensional space, enclosed by four walls. The team of agents and the single enemy agent begin the game from opposite ends of the map. Agents can navigate from their current position to any other position on the map by using the A* algorithm to find the shortest path. Items are also placed on the map at locations that are equidistant from both the team starting points and the enemy starting point. These items consist of health packs and a range of weapons, all of which can be collected and used by both the team agents and the enemy agent during the course of the game. If an item is picked up by an agent during the course of the game it disappears from the map for a short time before it respawns and can be used again. The game ends when either the team of agents have defeated the single enemy agent, the enemy agent has defeated all five team agents or a predefined maximum game time has elapsed.

[2]All the agents have infinite ammunition for the weakest weapon (i.e. the blaster) but the single enemy agent has infinite ammunition for the strongest weapon (i.e. the railgun) also.

2.2 Agent Artificial Intelligence

The behaviour of all autonomous agents within the gaming environment (i.e. both the team of evolved agents and the single enemy agent) is based on the goal-driven agent architecture as described by Buckland [3]. However, the way the agents decide on which goal to follow at any particular time is different. The enemy agent decides on which goal to pursue at any given time based on intermittent desirability checks. Each goal has a desirability algorithm associated with it that results in individually rational behaviour. The evolving team agents decide on what goal to pursue based on a decision-making tree evolved using a genetic program.

Within the goal-driven agent architecture the goals can be either atomic (define a single task or action) or composite (made up of several subgoals). Composite goals are broken down into subgoals of a simpler nature, hence a hierarchical structure of goals can be created for any game agent to define its behaviour.

3. THE GENETIC PROGRAM

Genetic programming is chosen as the technique used to evolve the behaviours as it has the potential for uncovering novel team tactics for the NPCs by searching the solution space of all possible behaviours. It also allows for the automatic creation of the NPCs' decision-making code and the GP tree representation means that the behaviours of the teams can be later analysed and reused. Each GP chromosome comprises five separate trees, one for each agent in the team that defines the manner in which the agent decides what actions to perform when following the tactic (i.e. each team of five agents are represented as a single chromosome). The population consists of 100 chromosomes (teams) and each simulation is run for 100 generations or until the population converges. As there is a degree of randomness within the gaming environment, every team (chromosome) plays five games in each generation of the simulation and the results are averaged so as to give a more accurate representation of a team's performance. An example of a team chromosome can be seen in Figure 1.

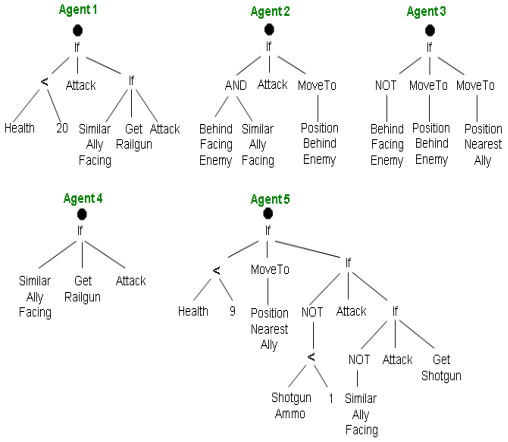

Figure 1: Sample Evolved Team Chromosome.

3.1 GP Node Sets

A strongly-typed GP is used in order to constrain the type of nodes that can be used as children of other nodes. Strongly-typed genetic programming (STGP) is able to evolve programs that perform significantly better than GP evolved programs. Furthermore, programs generated using STGP are easier to understand [7]. There are five node sets in total:

Action node set: The nodes that constitute this set define the goals the agent can pursue or actions it can perform (e.g. attack the enemy) but also include the *IF statement* node. All goal nodes are terminals bar the *MoveTo* goal, which takes in a position from the positional node set to which the agent should navigate to.

Conditional node set: There are 7 conditional nodes in this set that can be combined to form the conditions under which an action is to be performed.

Positional node set: Nodes in this node set are all terminal nodes that represent vector positions on the map to which the agents can move. These positions are relative to the positions of other agents on the map (e.g position of nearest ally).

Environmental parameter node set: This node set consists of parameters taken from the gaming environment that are checked during the decision making process of the evolving agent. Such nodes include an agent's current health, the distance to an agent's nearest ally, the distance to its enemy and the agent's ammunition supplies for each weapon in its inventory.

Numerical node set: This node set defines arithmetic operators and constants.

There are a total of 39 different types of node across the five node sets that can be combined to describe an agents decision-making code. The decision-making trees created from the evolutionary process can reach a maximum depth of 17. Hence, the search space of possible trees is vast.

3.2 Fitness Evaluation

To evaluate the performance of a team in a given simulation, the fitness function must take a number of factors into account: the length of time a game lasts, the remaining health of both the enemy agent and ally team and the length of the chromosome (to prevent bloating of the trees). To obtain an accurate measure, five games are used. The basic fitness is calculated as follows:

$$RawFitness = \frac{AvgGameTime}{Scaling * MaxGameTime} + \frac{(5 * (Games * TSize * MaxHealth - EH) + AH)}{Games * TSize * MaxHealth}$$

where $AvgGameTime$ is the average duration of the five games in a simulation, $Scaling$ is a variable to reduce the impact the game time has on the fitness of a team, EH and AH are the total amount of health remaining for the enemy agent and for all five ally agents respectively (averaged over five games), $TSize$ is the number of agents in the evolving team (set to 5 for these experiments), $Games$ is the number of games played (i.e. five) and $MaxHealth$ is the maximum health an agent in the game can have.

As a general rule, the longer a game lasts, the better the team are at surviving the enemy's attack. As the single agent in these experiments is likened to a human player so as to evolve team tactics for single player combative computer games, shorter games would mean either the human player dies quickly (making the games too difficult) or the team dies quickly (making the games too easy). More importance is attached to the change in the enemy agent's health than the corresponding change in the ally team's health as the tactics are evolved to be capable of defeating the enemy.

The length of the chromosome is taken into account in the fitness calculation to prevent trees from bloating.

$$StdFitness = (MaxRF - RawFitness) + \frac{Length}{LengthFactor}$$

where $MaxRF$ is the maximum value possible $RawFitness$ can hold and $LengthFactor$ is a parameter used to limit the influence the length of the chromosome has on the fitness. The fitter the team the closer the value $StdFitness$ is to zero. Once fitness scores are calculated for all teams of a generation, they, then, are used to probabilistically select chromosomes (i.e. teams) from that generation to be used to make individuals for the next generation of the population.

3.3 Selection Process

Selection is performed in two phases. The first is a form of elitism where m of the best n individuals from each generation are retained by reproducing them into the new generation unaltered. For these experiments, three copies of the best individual and two copies of the next best individual are retained in this manner. This ensures that the fittest members of the population are not destroyed or lost.

The second method of selection is roulette wheel selection, which selects chromosomes from the current generation probabilistically based on the fitness of the chromosomes. Each individual is assigned a section of the roulette wheel proportional to its fitness in relation to all other individuals in the population. Any chromosomes selected in this manner are subjected to crossover and mutation operators before being inserted into the next generation.

In order to add more diversity and prevent premature convergence of the population, there is also a 2% chance for completely new chromosomes to be created and added to the population each generation rather than selecting from the current population.

3.4 Team-based Crossover

The crossover operator used here was first proposed by Haynes [6]. It involves selecting a random five bit mask that decides what agents of the parent team chromosomes are to be altered during crossover. A '1' indicates that the agent is copied directly over into the child chromosome and a '0' indicates that the agent is to take part in crossover with the corresponding agent of the other parent chromosome, before being placed in the child chromosome.

Following the selection of the bit mask, a random crossover point is chosen within each agent to be crossed over. The node at the crossover point in each corresponding agent of the two parents must be from the same node set in order for a valid crossover to take place (e.g. a subtree with its root as a conditional can only be swapped with a subtree whose

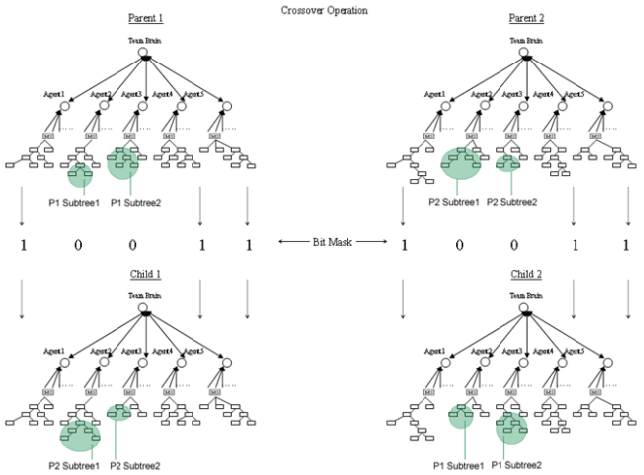

Figure 2: Crossover Operation between two Teams

root is also a conditional). Figure 2 illustrates the crossover operation between two sample team chromosomes.

3.5 Team-based Mutation

There are two kinds of mutation operators used in the evolutionary process.

The first form is known as swap mutation. It involves randomly selecting a mutation point in one of the five agent trees of the chromosome and deleting its subtree. The node at the mutation point is replaced with a new random node from the same node set and a new random subtree is grown from this new node.

The second mutation operator randomly selects two of the five agents of the chromosome to take part in the mutation. A random point is chosen in each of the selected agent trees for the tactic being evolved and the subtree at that point is swapped between the two agents.

4. PHENOTYPIC ANALYSIS

The GP described in the previous section is used to evolve a number of solutions capable of defeating the single enemy agent. A total of 21 solutions are evolved over a series of runs. All of the teams evolved are capable of defeating the enemy agent, but not all of them have the same fitness as some can defeat the enemy more effectively than others. We suspect this is due to a variance in the behaviours of the individual agents on each of the teams. We hypothesise the agents of the teams with the better fitness scores work more cooperatively together in order to defeat the enemy and that the interplay between the individual behaviours of the group members has a significant impact on the overall team's performance. We wish to discover what factors constitute a good team behaviour or what combination of specific agent roles are important to have on a team to ensure good overall team performance. Although it is possible for there to be multiple ways of defeating the enemy agent effectively, we suspect that commonalities exist between teams of similar fitness and that the agents of these teams of similar fitness fulfill similar team roles. We believe an investigation of these evolved team behaviours will result in the formation of clusters of teams with both similar behavioural properties and fitness scores. For this we have put forward a method for analysing and classifying the phenotypic behaviour of a group of evolved agents, such as those evolved using this genetic program.

Table 1: Sample Status Record for Team Agent

Update	1	2	3	...
GoalFollowed	14	14	7	...
WeaponSelected	2	3	3	...
RocketsCollected	0	0	0	...
RailAmmoCollected	0	15	0	...
ShellsCollected	0	0	0	...
BlasterAmmoUsed	0	0	0	...
RocketsUsed	0	0	0	...
RailAmmoUsed	0	0	1	...
ShellsUsed	0	0	0	...
Health	50	48	48	...
DistanceAlly1	1485	1587	1720	...
DistanceAlly2	9540	8563	7458	...
DistanceAlly3	14000	14102	14537	...
DistanceAlly4	14870	14982	15010	...
DistanceEnemy	5250	4580	4325	...
EnemyTarget	0	1	1	...

4.1 Gathering Behavioural Information

In order to investigate team behaviours, we must first specify a method to formally capture behavioural information from the gaming environment for each individual team agent. We implement this by periodically taking snapshots of each agent's game state throughout its lifetime. These periodic snapshots are accumulated throughout the game for each team agent and are stored as a vector of status records that can later be analysed. Each status record (snapshot) holds information on the identification number of the current goal being pursued by the agent, the identification number of current selected weapon, any ammunition collected for each weapon type and any ammunition used for each weapon type (since last snapshot was taken), relative distances to each of the agent's four allies and the enemy agent, the agent's current health level and whether or not the agent is the current target of the enemy agent. One of the most difficult tasks is choosing the most suitable factors to extract from the gaming environment. Although other factors could be captured from the gaming environment (e.g. distance to items), we believe the set of agent parameters we have chosen is sufficient for the classification of a game agent's behaviour in this environment. The set of parameters that can be extracted and stored in these status records could be extended to deal with more complex environments. A sample record is illustrated in Table 1. The values in the cells of the table refer to: update number, goal and weapon identifiers, amounts of ammunition supplies and health, distances from other agents and a Boolean value stating whether or not the agent is the current target of the enemy agent.

A vector of these status records is kept for each agent on a team. Records are periodically taken every n game updates for each team agent until either the game has ended or the team agent has been killed by the enemy agent. This vector of snapshots of the agent's state throughout the game is later used to classify its behaviour in ascertaining how likely it is to be a specific agent type.

4.2 Identification of Agent Types

Having discussed how the behavioural information is collected from the game environment, we must now consider how to best utilise this information to classify the different behaviours of the team agents. The first step is to identify the different agent types to be used in the classification. Each agent type has their own specific behavioural characteristics but the agent types are not mutually exclusive and an agent's behaviour can be categorised as simultaneously belonging to different types. A total of eleven different agent types that capture specific agent behaviours have been identified. In order to classify these behaviours, algorithms have been developed to calculate the degree of membership that an agent has to each specific agent type. Based on an agent's behaviour we calculate values indicating the degree to which this satisfies a particular role (e.g. attacker). The value returned from these calculations is used to calculate the probability of being a particular role according to:

$$FinalProbability = \frac{1}{1 + c * e^{((-value)*s)}}$$

where c and s are constant values and $value$ is the original value returned from the agent type classification equations.

4.2.1 Agent Type Classification

In order to classify an agent's behaviour, we must define classification equations for each of the roles identified that determine to what degree a particular agent's behaviour describes that agent type.

- Decoys: In our environment, a decoy agent is any agent who deters enemy fire from its teammates. This is generally done when a team agent deliberately exposes itself to the enemy so as to make itself the prime target of enemy fire and hence deter fire from the other team members. The first factor we use to calculate the decoy score for an agent is the average distance from the team agent to the enemy. For this, the smaller the distance, the better, as the closer an agent is to the enemy the more likely it is to influence the enemy's behaviour (i.e. cause the enemy to retreat, panic, focus its fire on the agent, etc.). The second factor used to calculate the decoy score for an agent is the number of times throughout its lifetime that the team agent was the focus of the enemy fire. Both these factors are weighted equally for these experiments.

$$\frac{DecoyBoundaryDist}{AverageDistEnemy} * 0.5 + \frac{\sum EnemyTarget}{AgentLifetime} * 0.5$$

- Ammunition Gatherers: An agent is an ammunition gatherer if one of its primary activities is to collect ammunition packs. As there are three different types of ammunition packs that can be collected within the gaming environment, we define three different types of ammunition gathering agent types, one for each of the railgun, shotgun and rocket launcher weapons[3]. In order to ascertain the likelihood an agent is an ammunition gatherer, we must take account of how much ammunition the agent has gathered for that particular weapon over the course of its lifetime and how much of its lifetime it has spent looking for that ammunition type (i.e. the proportion of status records in which its goal is to retrieve ammunition for that weapon). For these experiments, both the factors are weighted evenly. The following equation is used to classify a railgun ammo gatherer:

$$\frac{RailAmmoGathered}{RailAmmoPerPack} * 0.5 + \frac{\sum GoalGetRailgun}{Lifetime} * 0.5$$

- Attackers: An attacker is an agent who aggressively attacks the enemy. We define four different types of attacking agents for our classification, one for each weapon type. To calculate the probability of an agent being a specific attacker type three factors need to be taken into account: the amount of ammunition fired for that weapon during the course of the agent's lifetime, the number of times the agent's goal was to attack the enemy and the number of times the weapon in question was the agent's weapon of choice. For these experiments, we have viewed the amount of ammunition used by the agent as the most important factor in determining its probability of being an attacker and have thus weighted this factor more heavily than the other two. The equation to classify the rocket launcher attacker type is shown below:

$$\frac{RocketsFired}{Lifetime} * 0.5 + \frac{\sum GoalAttack}{Lifetime} * 0.25$$

$$+ \frac{\sum WeaponSelectedRocketLauncher}{Lifetime} * 0.25$$

- Evaders: Evaders are agents who do their best to keep their distance from the enemy agent. Their behaviour can be thought of as being cautious as they choose to stay back rather than engage the enemy. For these experiments we calculate the probability of an agent being an evader based on its average distance from the enemy over the lifetime of the agent. The larger this distance the more likely the agent is an evader.

$$\frac{AverageDistanceEnemy}{EvaderBoundaryDistance}$$

- Health Preservers: An agent can be categorised as a health preserver if it has a high average health level over its lifetime and it actively seeks health packs within the game. Therefore, the probability of an agent being a health preserver is relative to its average health level over its lifetime and how often it actively tries to retrieve health packs to replenish its health. An agent's ability to preserve its health should also take into account how long it is able to stay alive; therefore the lifetime of the agent relative to the lifetime of the longest living agent on the team (i.e. $MaxLifetime$) is also taken into account when calculating the agent's health preserving ability. Although this method is not ideal, as it only calculates the probability relative to the longest living agent on the team, it does eliminate

[3]Agents have infinite ammunition for the blaster weapon.

the chance of agents who die off quickly receiving a high health preserver score. Furthermore, the chance of all five team agents dying off early is very slim as the enemy agent can only target one team agent at a time (although it can hit many with a single shot if they are very clustered). For these experiments the factors that determine the probability score are weighted differently as we believe the average health and relative lifetime of an agent are better indicators of an agent's ability to preserve health than the number of times it seeks health packs.

$$\frac{AverageHealth}{MaxHealth} * \frac{Lifetime}{MaxLifetime} * 0.75$$

$$+\frac{\sum GoalGetHealth}{Lifetime} * 0.25$$

- Cohesive Agents: Cohesive agents are agents who stay close to at least one of their teammates during the course of their lifetime. The average distance from the agent to its nearest teammate is checked to see if it is within some arbitrarily chosen distance.

$$\frac{CohesiveBoundaryDistance}{AverageClosestAllyDistance}$$

4.2.2 Storing Team Agent Type Probabilities

These probabilities are stored in a vector of probabilities for every game played by each agent. In order to get an accurate measurement of an agent's behaviour a number of games must be played per evaluation and the results averaged, as there is a degree of noise or randomness inherent in the gaming environment. In total, for these experiments, 100 games are played per team evaluation so 500 probability vectors are calculated, 100 for each of the five agents on any one team. For each agent, the probabilities in all 100 probability vectors for that agent are averaged to give a single probability vector that describes the agent types that agent most likely belongs to. So we end up with five probability vectors in total, one for each agent on the team describing the behaviour of that team agent.

4.3 Comparison of Team Behaviours

In order to compare overall team behaviours, we must define a method to combine the five team agent probability vectors into a single team probability vector that describes the behaviour of the team as a whole. As the probabilities in the probability vector are not mutually exclusive, we use a simple thresholding system to decide to which agent types a particular agent most likely belongs. A threshold of 0.6 is chosen for these experiments. Every agent probability above 0.6 is raised to 1 and everything below lowered to 0. Although this method is not perfect and very lossy, it is sufficient to distinguish between different types of team agents for these experiments. Once this thresholding technique has been executed for a team agent's probability vector, a 0 should be in place for every agent type it is unlikely to be and a 1 should be in place for every agent type it is likely to be. When this is done for all five team agents, the probability vectors can simply be added up to give a final team vector of whole numbers describing how many agents of each type are on the team.

4.3.1 Decision Tree Classification

In order to compare the team behaviours of the 21 evolved teams, a team vector must be calculated for each evolved team using the method described above. Once these are found, we put the team vectors together with the relative fitness of each evolved team into a decision tree classification algorithm and observe the results. The target variable of our decision tree is, of course, team fitness as we wish to see what team behaviours or combination of agent types on a team lead to similar team fitness. As our data set is relatively small we chose our minimum node split size to be 5 so leaf nodes can be a maximum of size 4. We believe that the path from the root node of the tree to the leaves should tell us something about the phenotypic behaviours of the agents of teams who perform similarly and the combination of agent types on a team that allow for better (or worse) team performance.

5. RESULTS

The decision tree classification algorithm used to analyse the phenotypic behaviours of the evolved teams resulted in the decision tree shown in Figure 3.

As the target variable for the decision tree is a team's fitness score, the teams are grouped based on similar fitness values, so the leaf nodes of the tree contain groups of teams that performed alike over their evaluation, given some small degree of variance within the group. By the nature of the decision tree, these groups are formed by finding behavioural properties common in the team vectors of all teams in a group. Although the training set of data is relatively small for these experiments, the decision tree classification algorithm did manage to extract behavioural properties inherent in good, bad and average teams by splitting the teams into their corresponding groups based on their performance during the classification.

From the tree in Figure 3, we can see that the most differentiating factor between good and bad teams is the number of railgun attackers on a team. Teams with at least one railgun attacker on their team perform better than those that do not. This is understandable as the railgun is the most powerful weapon available in the environment. Hence, the team will generally have a higher fitness if at least one agent on a team uses a railgun to attack the enemy, as more damage is likely to be inflicted on the enemy agent.

Following the left branch of the tree we can establish the features that constitute a poorly performing team. Firstly, as mentioned earlier, these teams have no agents who attack using railguns which severely limits their attacking ability. The number of evaders on these teams is also an important factor in determining a teams performance. A team with two or more evaders tends not to be as fit as a team with zero or only one evader. This can be explained by the fact that there are no railgun attackers on the team. The distance an agent is away from the enemy has little or no influence on the damage inflicted by or the accuracy of a railgun. However, distance has a great impact on the effectiveness of the other three weapon types. Hence, the greater the number of evaders on a team with no railguns, the more limited its attacking ability will be, leading to an overall lower team fitness.

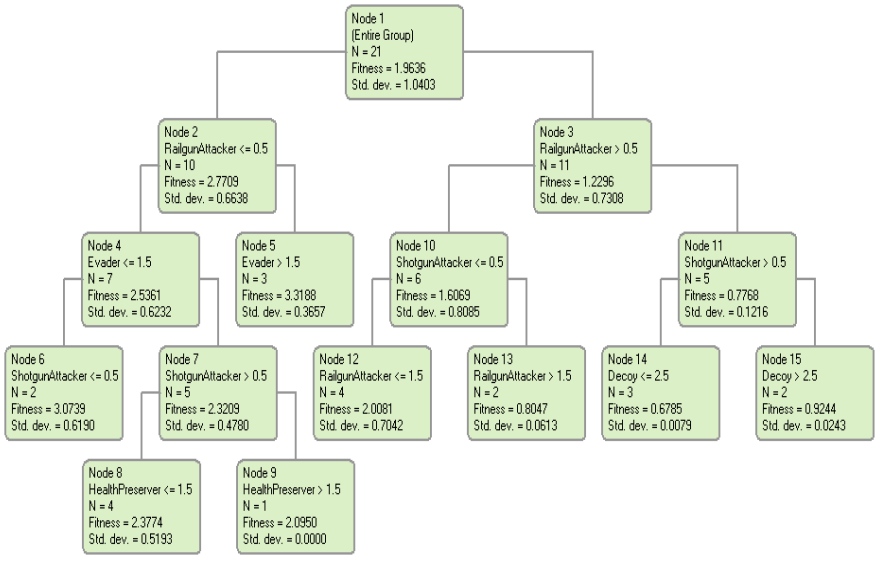

Figure 3: Decision Tree for Classification of Team Behaviours.

Teams with just one (or no) evader(s) are further divided into those with no shotgun attackers and those with at least one shotgun attacker. Those with no shotgun attackers perform worse which is understandable as their attacking ability is even more restricted with no railgun attackers nor shotgun attackers on the team. The teams with at least one shotgun attacker are additionally split based on the number of health preservers on a team. Understandably, the teams who are better at preserving their health perform better as agents in a team need to conserve their own health and stay alive in order to inflict more damage on the enemy agent.

Looking at the right branch of the tree we can see that the main factor that constitutes a team that performs well is its ability to attack the enemy. All teams on the right half of the tree have at least one railgun attacker, which is the most powerful weapon available in the simulation environment. Moreover, the decision tree shows that teams who have at least one shotgun attacker on their team as well perform even better. This is intuitive as the shotgun would be the second most powerful weapon after the railgun[4] and if a teammate is already going for the railgun it makes sense to go for the next most powerful weapon[5]. It seems that if there is at least one railgun attacker and at least one shotgun attacker on a team, then the team has sufficient firepower to perform well. However, if one or two of the other agents on a team are decoy agents[6] that deter the enemy fire away from the attackers, then the team performs even better as a whole. If all three of the other team members are decoys, then the team performs more poorly. We believe this is because one to two decoys per team is sufficient to distract the enemy and if more decoys are present more damage than is necessary gets inflicted on the team as a whole. Furthermore, the other team member(s) could be another attacker(s) which could inflict even more damage on the enemy agent[7].

Teams that have at least one railgun attacker but no shotgun attacking agents do not generally perform as well, however teams with two or more railgun attacking agents perform much better than teams with just one railgun attacker. This finding is justifiable as a team with no shotgun attackers and two railgun attackers has approximately twice the firepower of a team with no shotgun attackers and just one railgun attacker, thus able to inflict more damage on the enemy.

From these results we can clearly see that it is the team's combined ability to attack the enemy and hence inflict damage on the enemy that is the primary factor in determining the performance of the team. The best performing teams are those that have a combination of both railgun attackers and shotgun attackers whilst also having one or two decoy team members to draw enemy fire away from the team's attacking agents.

6. CONCLUSIONS AND FUTURE WORK

The purpose of this paper is to define an approach to analyse the phenotypic behaviours of teams of autonomous agents evolved using a genetic program. For these experiments, a number of such teams are evolved for a combative computer game environment and an analysis is done on their behaviours in an attempt to see what combination of individual agent behaviours make up a good (or bad) team behaviour. From the above results we can conclude that the behavioural interactions among the agents on a team significantly affect the team performance. Although the data

[4]Note that the rocket launcher is also a powerful weapon but it is not as effective as the railgun or shotgun as it is much slower and if it is used in close proximity to the target it can damage the agent firing it also.

[5]Weapon packs can only be picked up by one agent at a time and once picked up they do not respawn until a time delay has elapsed.

[6]All evolved teams contained at least one decoy agent.

[7]Although possible for a decoy agent to also be an attacker it is very unlikely.

set used for these experiments is small for classification purposes it does still show similarities in the behaviours of teams with similar fitness scores. These similar behaviours show phenotypic features that are common to teams with comparable fitnesses. As such, the phenotypic behaviours common to the teams who receive the higher fitness scores can be extracted as those behaviours inherent in good team behaviours. From these experiments, the features common to high performing teams are namely, having one or more shotgun attackers and one or more railgun attackers on a team whilst also having one to two decoy team members to draw away enemy fire from the team's attacking agents. It is interesting that the decision tree is able to classify the best performing teams as those who have a combination of the two most powerful attacking agent types as well as decoys to divert enemy fire away from the attackers. This combination of agents shows a group rationality among the agents as the decoys are rather selflessly making themselves targets to protect their teammates and the attackers are resourceful of the weapons they choose to attack with[8].

We can clearly see that there is a correlation between a team's performance (fitness score) and the combined behaviours of the team's agents (phenotypic behaviours). For future experiments, we will explore using different methods to combine the team agent vectors together when calculating the team vector, such as using two or more thresholds instead of just one and giving a different score for each threshold an agent's behaviour describes.

In an analysis conducted by Barlow et al. [1], it was found that the three most important factors in team performance in a squad environment are the assignment of roles to team agents, coordination amongst the team agents and communication between the team agents. The GP used in these experiments allows for automatic role assignment and coordination of team agents but there is no explicit communication between team members. In the future, we would like to integrate a communication element into our GP when evolving the behaviours by giving the agents the capability to send and receive messages and respond accordingly to messages received from teammates. It would be interesting to analyse the effect this explicit messaging would have on the evolved behaviours and to compare them to the behaviours evolved when no messaging is present. We hypothesise that the behaviours will be more varied with the communication element as agents will have the capability to warn allies of threats or request backup and their teammates should react accordingly.

Our approach to phenotypic analysis could also be used in other domains that contain teams of agents working cooperatively together to analyse their team behaviours and see what it is that makes them perform well (or poorly) in their environment.

7. ACKNOWLEDGMENTS

The primary author would like to acknowledge the Irish Research Council for Science, Engineering and Technology (IRCSET) for their assistance through the Embark initiative.

[8]The attacking agents of the fitter teams do not all go for the most powerful weapon (i.e. the railgun) as this would be disadvantageous to the team due to the limited resources available in the simulation environment.

8. REFERENCES

[1] M. Barlow, M. Luck, E. Lewis, M. Ford, and R. Cox. Factors in team performance in a virtual squad environment. In *SimTecT 2004 Simulation Technology and Training Conference*, 2004.

[2] M. T. Brannick, E. Salas, and C. Prince. *Team Performance Assessment and Measurement: Theory, Methods, and Applications*. Lawrence Erlbaum Associates, 1997.

[3] M. Buckland. *Programming Game AI by Example*, chapter Goal-Driven Agent Behaviour, pages 379–415. Wordware Publishing, Inc, 2005.

[4] D. Doherty and C. O'Riordan. Evolving agent-based team tactics for combative computer games. In *AICS 2006 17th Irish Artificial Intelligence and Cognitive Science Conference*, 2006.

[5] 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*, 2006.

[6] T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright. Evolving a team. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, Cambridge, MA, 1995. AAAI.

[7] T. Haynes, R. Wainwright, S. Sen, and D. Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 271–278, Pittsburgh, PA, USA, 15-19 1995. Morgan Kaufmann.

[8] J. R. Koza. Genetic programming: On the programming of computers by means of natural selection. *Statistics and Computing*, 4(2), 1994.

[9] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler. Co-evolving soccer softbot team coordination with genetic programming. In *Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

[10] M. D. Richards, D. Whitley, J. R. Beveridge, T. Mytkowicz, D. Nguyen, and D. Rome. Evolving cooperative strategies for uav teams. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1721–1728, New York, NY, USA, 2005. ACM Press.

[11] P. Sweetser. Current AI in games: A review. Technical report, School of ITEE, University of Queensland, 2002. http://www.itee.uq.edu.au/ penny/_papers/Game AI Review.pdf.

[12] C. Thurau, C. Bauckhage, and G. Sagerer. Imitation learning at all levels of game-ai. In *Proc. Int. Conf. on Computer Games, Artificial Intelligence,Design and Education*, pages 402–408, 2004.