Artif Intell Rev (2007) 27:223–244
DOI 10.1007/s10462-008-9078-1

Evolving team behaviours in environments of varying difficulty

Darren Doherty · Colm O'Riordan

Published online: 13 September 2008

Abstract This paper investigates how varying the difficulty of the environment can affect the evolution of team behaviour in a combative game setting. The difficulty of the environment is altered by varying the perceptual capabilities of the agents in the game. The behaviours of the agents are evolved using a genetic program. These experiments show that the level of difficulty of the environment does have an impact on the evolvability of effective team behaviours; i.e. simpler environments are more conducive to the evolution of effective team behaviours than more difficult environments. In addition, the experiments show that no one best solution from any environment is optimal for all environments.

Keywords Genetic programming · Team behaviours · Team evolution · Shooter games

1 Introduction

First-person and third-person shooter (FTPS) games belong to a popular genre of computer games known as shooters. Shooter gaming environments involve two or more sets of agents competing for survival in a hostile setting, where agents use some form of projectile weapon to attack their enemy. In recent years, there has been an emergence of squad-based FTPS games as agents competing in these games share the common goal of eliminating the opposing team(s). The AI element of squad-based FTPS games must be tactical as the team of artificial agents must work together to devise the most efficient and effective method to achieve their common goal. Team tactics can be defined as the set of methods used by a team for winning a small-scale conflict. As tactics are highly dependent on the situation (Thurau et al. 2004) (i.e. terrain, team supplies, enemy movement/concentration, etc) it is very difficult for game

D. Doherty (✉) · C. O'Riordan
Department of Information Technology, National University of Ireland, Galway, Ireland
e-mail: darren.doherty@nuigalway.ie

C. O'Riordan
e-mail: colm.oriordan@nuigalway.ie

developers not only to hardcode the team tactics themselves but also to decide when and where it would be effective to deploy certain tactics.

Rather than attempting to code the complex behavioural systems required to allow a team of agents to simulate intelligent tactical behaviours, we have chosen instead to evolve these tactical team behaviours using evolutionary computation (EC) techniques. In previous work (Doherty and O'Riordan 2006), effective teams have been evolved in a shooter gaming environment where agents have perfect vision through a field of 180°. In this current paper, the robustness of this approach is tested by exploring the ability of the team agents to evolve effective behaviours in eight different environmental settings of varying difficulty. Moreover, it is common for FTPS games to have a number of difficulty levels in which the game can be played. The environmental difficulty is varied by altering the field of view and viewing distance of the agents, thereby varying their ability to perceive information from the environment. In modern FTPS games, both the non-playable characters (NPCs) and the human player(s) have limited fields of view and viewing distances within which information can be perceived. To examine the effect varying the environmental difficulty has on the fitness of the evolved teams, the same genetic program is used in all experiments. The environments used in the experiments range from very restricted, with short viewing distances and narrow fields of view, to very unrestricted environments, with long viewing distances and wide fields of view.

We hypothesise that the shorter the viewing distance and the narrower the field of view, the more difficult it will be for the agents to evolve effective behaviours as their perceptual abilities are more limited. Additionally, larger viewing distances and wider angles enable the agents to perceive more of their environment, which should be more conducive to the evolution of effective behaviours. We believe that these latter experiments should produce results comparable to our previous experiments in which agents have perfect vision through a field of 180°. Altogether, eight experiments are undertaken over four viewing distances and two fields of view and the results analysed.

In the remainder of this paper, some related work, the game to be played and the genetic program to be used in the evolution will be discussed, together with the experimental setup and a discussion of the results of the experiments.

2 Related work

2.1 Artificial intelligence in FTPS games

In early FTPS games, such as *Doom* (ID-Software 1994) or *Quake* (ID-Software 1996), the player had very limited interactions with the NPCs in the game world. These early games were only concerned with the player on a solo mission through an unknown world, fighting wave after wave of NPCs. The NPCs in these early FTPS games were known as "cannon-fodder" NPCs because their behaviour was very repetitive and predictable as deterministic techniques, such as finite state machines and scripting, were used to define their behaviours. These techniques were used as they are easy to understand and implement. However, the use of deterministic techniques to implement the artificial intelligence of NPCs meant that the behaviour of all the NPCs would be identical, which allowed players to easily predict their actions. Since the early 2000s, developers have begun to realise the need for more intelligent NPC behaviours in order to capture and hold the interest and attention of players. Developers are striving to create NPCs that behave in a more humanlike and individual manner to push the suspension of disbelief of their NPCs and prevent their behaviour from being repetitive and predictable.

With the emergence of squad-based FTPS games in the early 2000s, game developers have struggled to create effective systems that allow for teams of NPCs to interact, cooperate and coordinate their behaviour intelligently. As such, developers have opted to use simple techniques to make it appear as if the NPCs are cooperating in an intelligent manner. For example, some game developers prevent two NPCs from shooting at the player simultaneously, causing them to appear to be taking turns attacking the player. This is combined with audio cues from the agents such as shouting "cover me" when an agent goes to reload its weapon to create the illusion of cooperative behaviour. However, using rudimentary or "cheating" mechanisms to simulate cooperative behaviour in squad-based FTPS games is less than ideal. Developers are still striving to create an AI architecture that allows a team of NPCs to communicate and coordinate their behaviour in an intelligent manner, such that they display effective group rational behaviour. This task is by no means trivial as achieving coordination among multiple autonomous agents in any domain can be a difficult task and FTPS games are set in very complex environments.

2.2 Evolutionary computation and games

In the past, evolutionary computation techniques have been comprehensively applied to board games, such as: tic-tac-toe (Fogel 1993), chess endgames (Hauptman and Sipper 2005; Lassabe et al. 2006), checkers (Fogel 2002), Go (Rosin and Belew 1995; Richards et al. 1997), Othello (Eskin and Siegel 1999) and Monopoly (Frayn 2005) in an attempt to find optimal playing strategies. However, EC techniques have not been used as extensively in the exploration and research of artificial intelligence for computer games (commonly referred to as game-AI). In general, the environment and range of agent behaviours in a computer game are very complex. This means the search space will most likely be large and it will take a long time to converge on a solution. In particular, online evolution (evolving while the game is being played) will be slow and undesired behaviours may emerge. Developers have also been hesitant to introduce EC techniques into their game-AI offline (during development) as there is no guarantee the EC technique will find desirable or intelligent behaviours because the search space is so large. However, a number of games that have incorporated EC into their game-AI have been very successful, e.g. Lionhead Studio's *Black & White* (Lionhead-Studios 2001) or GSC Gameworld's *S.T.A.L.K.E.R.: Shadow of Cherynobyl* (GSC-Game World 2006). A genetic algorithm (GA) has been used to evolve NPCs in a first-person shooter game to successfully dodge enemy fire (Champandard 2004). Ponsen (2004) have used a GA off-line to design tactics for a real-time strategy game. Cole et al. (2004) successfully tuned an NPC's weapon selection parameters in the game *Counter-Strike* (Valve 2000) using a GA. Genetic programming (GP) has been used to evolve agent behaviours for simple computer games, such as PacMan (Koza 1992) and Snake (Ehlis 2000). In addition, neuro-evolution has been successfully applied to computer games to evolve adaptive behaviours of NPCs (Stanley et al. 2005; Yannakakis and Hallam 2004).

2.3 Evolutionary computation and teamwork

Evolutionary computation techniques have been successfully applied to the evolution of team behaviours in different simulated domains. Neuro-evolution, genetic algorithms and genetic programming have all been used to successfully evolve team behaviours in the predator-prey domain (Haynes and Sen 1995; Luke and Spector 1996; Yannakakis and Hallam 2004). In addition, neuro-evolution has been used to allow a human player to evolve a team of

robots for military combat in real-time (Stanley et al. 2005) and an adapted genetic algorithm representation was used to develop an adaptive team-oriented AI mechanism for agents in a first-person shooter computer game (Bakkes et al. 2004).

2.3.1 Genetic programming and teamwork

Genetic programming has been a popular evolutionary computation technique for evolving tactics and strategies for teams of agents in a variety of domains. GP was first applied to team evolution by (Haynes et al. 1995b). With respect to team evolution, GP has been mainly used to solve multi-agent control problems using teams of cooperating agents.

Richards et al. (2005) used a genetic program to evolve groups of unmanned air vehicles to effectively and efficiently search an uncertain and/or hostile environment. Three different environments of varying difficulty are used in their experiments. The first two environments have rectangular search areas but the second also contains a single fixed hostile agent making it more difficult. The third environment has two irregular shaped search areas and a no-fly zone making it the most difficult environment of the three. Their results show a correlation between the fitness of the evolved flying strategies and the difficulty of the environment. Although, the work of Richards et al. (2005) has some similarities to our research, there are a number of notable differences. The hostile agent in Richards et al. (2005) is fixed and cannot navigate the map, whereas in our research the enemy agent actively searches the map to find and destroy the team. In Richards et al. (2005), the objective of the evolving team is not to destroy the hostile agent but to scan a search area by flying over it, whereas in our research, the goal of the team is to eliminate the enemy agent. In addition, because the environments used in Richards et al. (2005) are fixed and the hostile agent's location does not change the evolution can endow the team with instinctive knowledge about areas of the map that are dangerous. In our research, this is not possible as the enemy's desirability algorithms that govern its behaviour are created with random biases at the start of every game and when exploring the map agents navigate to a randomly selected node so the enemy's behaviour/movements varies from game to game. Moreover, the environmental difficulty in our experiments is varied by altering the agents' perceptual abilities rather than by using more complex maps.

Genetic programming has been used to enable a team of ants to work together to solve a food collection problem (LaLena 1997). The ants must not only cooperate in order to reach the food but must also work together to carry it as it is too heavy for one ant to carry alone.

Pursuit and evasion strategies for predator–prey domains have also been successfully evolved using GP techniques (Reynolds 1993; Haynes and Sen 1995; Luke and Spector 1996). Reynolds (1993) used GP to evolve prey that exhibit a herding behaviour when confronted by predators. Haynes and Sen (1995) used strongly typed genetic programming (STGP) to evolve a team of predators to hunt a single prey. STGP is a variant of GP where every node in the genetic program is constrained over the type of nodes it can create as child nodes depending on its own type. Luke and Spector (1996) evolved predator strategies that enable a pack of lions to successfully hunt gazelle (Luke and Spector 1996). In order to vary the difficulty of the environment, the lions were given different levels of sensing ability in each of the experiments. This is a similar method to the one used in this paper for varying the environmental difficulty. Luke and Spector (1996) show that heterogeneous teams perform better than homogeneous teams in their environment.

In addition, combat tactics for teams of armed forces (Doherty and O'Riordan 2006, 2007) and sporting strategies for a number of different sports (Raik and Durnota 1994; Luke et al. 1997) have been successfully evolved using GP techniques. Raik and Durnota (1994) conducted experiments to demonstrate evolution of cooperation and effective communication

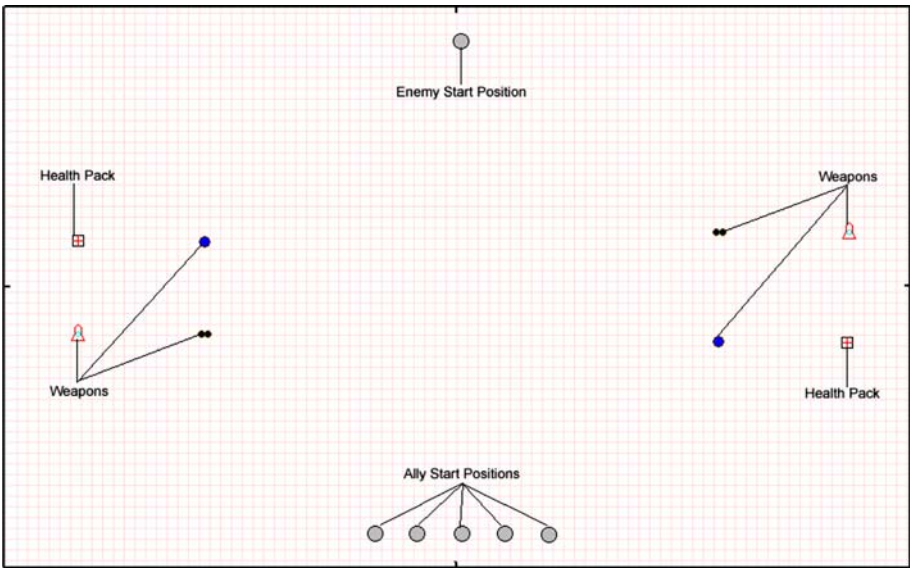


Fig. 1 Simulation environment map

amongst a team of volleyball players. Luke et al. (1997) used a strongly typed genetic program combined with a pseudo-heterogeneous team approach to successfully evolve team behaviours for a team of soccer agents.

3 The gaming environment

The environment used in this work is a shooter gaming environment similar to that used in previous research (Doherty and O'Riordan 2006, 2007) (see Fig. 1). The gaming environment is a 2-dimensional (2D) space, enclosed by four walls and is built using the *Raven* game engine[1] (Buckland 2005, Chap. 7). The evolved team (consisting of five agents) is pitted against a single powerful enemy agent. Items are 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 that respawn after a set time if collected by an agent. See Table 2 of Appendix for game specific parameters.

There are two types of agent in the simulation environment: team agents and the single enemy agent. Both types of agent use the same underlying goal-driven architecture to define their behaviour (Buckland 2005, Chap. 9). However, the methods used to decide which goal to pursue at any given time are different. The single enemy agent uses hand-coded desirability algorithms associated with each goal to decide on its course of action,[2] whereas team agents use an evolved decision-making tree to decide which goal to pursue at any given time. Team agents begin the game with the least powerful weapon in the environment but have unlimited

[1] The Raven game engine source code can be downloaded from the following url: http://www.wordware.com/files/ai/Buckland_AISource.zip

[2] If the enemy does not know the location of the team agents or items, its most desirable goal will be to explore the map. Thus, causing it to actively seek out their locations rather than wait. For an explanation of the desirability algorithms used see Appendix.

ammunition for it, so agents always have the ability to attack. The single enemy has five times the health of a team agent and the most powerful weapon in the environment with infinite ammunition. For a list of weapons and their properties see Table 3 in Appendix.

There is also a degree of randomness inherent in the gaming environment. Random biases are used when creating each desirability algorithm for the enemy agent, influencing the enemy's behaviour and a random degree of noise can be added to an agent's aim depending its aim accuracy. For these experiments, agents are given perfect aim accuracy to reduce the amount of randomness in the environment. However, bullets fired can still miss their target if the target moves before the bullet reaches it or if the weapon is fired outside of its ideal range. Note that weapons have different ideal ranges within which they are more effective and bullets for different weapons have different properties, such as travelling velocity, mass, spread, etc.

All agents have a limited range within which they can visually perceive information in their environment. An agent's visual capability is defined by a field of view and a viewing distance. For these experiments, the enemy can see twice as far as team agents but the fields of view of both agent types are equal.

Both types of agent have a memory which allows them to remember information they perceive in their environment. Any dynamic information, such as ally or enemy positions, is forgotten after a specified time (5 s for these experiments). However, static information, such as the location of items, is not forgotten if perceived, as their location does not change for the duration of the game.[3]

If the enemy is sensed by a team agent, the enemy is automatically selected as the team agent's target as there is only one enemy. However, as more than one team agent can be sensed by the enemy agent at any given time, the enemy chooses the closest team agent recently sensed as its target.

4 The genetic program

The genetic program used to evolve the decision-making trees for the team agents is similar to that used in our previous research (Doherty and O'Riordan 2006). The entire team of five agents is viewed as one chromosome, so team fitness, crossover and mutation operators are applied to the team as a whole. The decision making trees defining the behaviour of each of the five agents are derived from different parts of the team chromosome, so evolved teams are heterogenous. For a list of parameter values specific to the genetic program see Table 4 in Appendix.

A strongly typed genetic program comprising five node sets is adopted. In STGP, the initialisation process and genetic operators must only allow syntactically correct trees to be produced. The node sets are similar to those in previous research (Doherty and O'Riordan 2006). However, a number of new nodes have been added to accommodate the limited viewing capabilities of the agents. For example, a *can_see_enemy* node is added to the condition node set to allow an agent to make decisions based on when the enemy has become visible. For a complete list of nodes used in the experiments see Table 5 in Appendix. In total, there are 50 nodes across the five node sets that can appear in a GP tree.

[3] It is not possible for evolution to endow agents with knowledge of where to look for items as the genetic program does not contain directional nodes such as "go left" or "go right". Moreover, if an item's location is unknown, the "explore" node sends the agent to a random map location.

4.1 Fitness calculation

The fitness function takes into account: the remaining health of both the enemy agent and team agents, the length of time the games last and the length of the chromosome (to prevent bloating of the trees).

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

$$StdFitness = (MaxRawFitness - RawFitness) + \left(\frac{Length}{LengthFactor}\right)$$

where $Games$ is the number of games played per evaluation (set to 20 for these experiments), $TSize$ is the number of agents in the evolving team (i.e. five), $MaxHealth$ is the maximum health a team agent in the game can have, EH and AH are the total amount of health remaining for the enemy agent and for all five team agents respectively, $AvgGameTime$ is the average duration of the evaluation games (where game duration is counted as the number of game updates), $Scaling$ is a variable to reduce the impact of game time on fitness (set to four), $MaxGameTime$ is the maximum duration of a game (set to 5,000 game updates for these experiments), $MaxRawFitness$ is the maximum value possible $RawFitness$ can hold, $Length$ is the length of the chromosome and $LengthFactor$ is a parameter used to limit the influence the length of the chromosome has on the fitness (set to 5,000).

More importance is attached to the change in the enemy agent's health than the corresponding change in the team's health as the tactics are evolved to be capable of defeating the enemy. EW is a weight that accounts for this (set to five for these experiments). Longer games are also favoured to prevent teams who are capable of surviving the enemy's attack from dying off in the earlier generations. Once the $RawFitness$ of the team is calculated, the chromosome length is taken into account and the fitness is inverted such that fitness values closer to zero are better. Using this fitness function encourages evolution of aggressive behaviour where teams actively seek out to eliminate the enemy rather than simply avoid it.

4.2 Selection

There are two forms of selection used in these experiments. The first is a form of elitism where m of the best n chromosomes from each generation are reproduced unchanged into the next generation. Retaining 2–5% of the population using this form of elitism has been known to give good results (Buckland and Collins 2002, Chap. 5). For these experiments, three copies of the best individual and two copies of the next best individual are retained in this manner. The second method of selection is roulette wheel selection, which selects chromosomes from the current generation probabilistically based on the fitness of the chromosomes. Any chromosomes selected in this manner are subjected to crossover and mutation (given some probability of crossover and mutation respectively). To increase diversity, there is also a 2% chance for new chromosomes to be created and added to the population each generation.

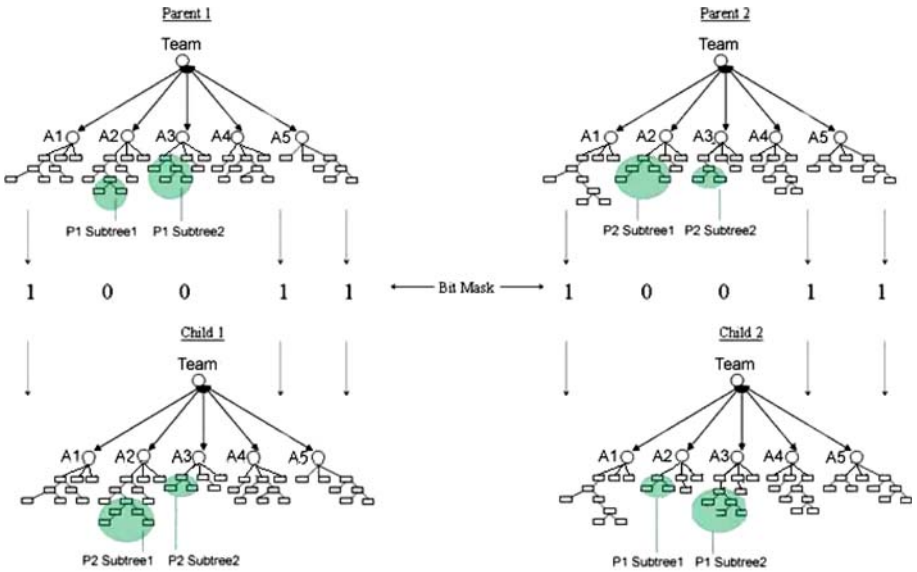


Fig. 2 Sample crossover operation between two team chromosomes

4.3 Crossover

The crossover operator used is specifically designed for team evolution (Haynes et al. 1995a). At the start of each crossover operation a random $Tsize$ bit mask is selected that decides which of the team agents in the parent chromosomes are to be altered during the crossover. A '1' in the mask indicates that the agent at that position is copied directly 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 (see Fig. 2). A random crossover point is then 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.

4.4 Mutation

Two forms of mutation are used in these experiments, one to allow good subtrees to spread within a team and the other to help maintain diversity in the population. The former mutation method, known as intra-team mutation, randomly chooses two agent trees from the same team chromosome and swaps two randomly selected subtrees between the agents (see Fig. 3). Similar to the crossover operation, the root nodes of the subtrees must be from the same node set in order for the mutation to be valid.

The second form of mutation is known as swap mutation and is used to help maintain diversity in the population. It involves randomly selecting a subtree from the team chromosome and replacing it with a newly created random tree (see Fig. 4).

5 Experimental setup

These experiments investigate how varying the difficulty of the environment can affect the evolution of team behaviour in a 2D shooter game setting. The difficulty of the environment is

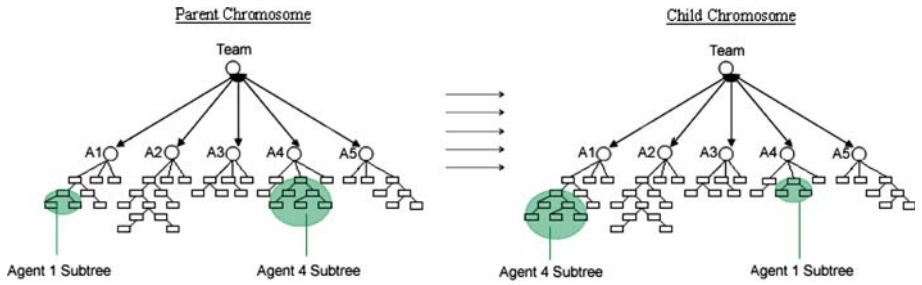


Fig. 3 Sample intra-team mutation operation between two team agents

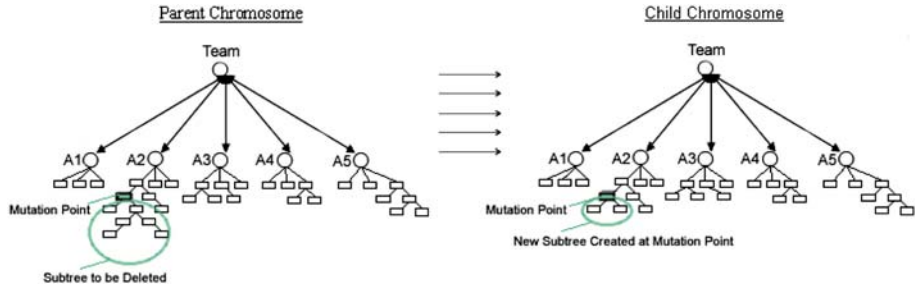


Fig. 4 Sample swap mutation operation

varied by altering the evolving agents' perceptual capabilities. Experiments are set up for two fields of view (90 and 180°) and four viewing distances (50, 200, 350 and 500 pixels). Eight experiments are used to investigate the evolution of team behaviours in these environments. Maximum and minimum viewing distances are based on the size of the grid. The maximum viewing distance of the agents is set to 500 pixels and the minimum viewing distance is set to 50 pixels for these experiments. Intermediate distances are then chosen as equally spaced distances between 50 and 500 pixels. With a viewing distance of 500 pixels, the team agents can see the location of the enemy agent as well as the locations of a number of items from their starting positions. This makes the game easier as agents do not have to spend time searching for game objects. In the most restrictive environments, where viewing distance is 50 pixels, agents can only see what is directly ahead, making it very difficult for them to locate game objects in their environment.

The enemy viewing distance is scaled relative to the viewing distance of the team agents. As there are five team agents and only the one enemy agent, the collective viewing range of the team covers a much larger portion of the map than that of a single agent. Therefore, it was decided to allow the enemy's viewing distance to be twice that of a team agent.[4] Figure 5 shows the eight environmental setups used for these experiments.

Twenty separate evolutionary runs are performed in each of the eight environments. In each of the runs, 100 team chromosomes are evolved over 100 generations. Each team evaluation comprises 20 games, whose results averaged in order to obtain a more accurate measure of performance as there is some randomness in the environment (see Sect. 3). The best performing team from each of the runs is recorded.

[4] For a viewing distance of 1,000 pixels, the enemy will have perfect vision within its field of view as 1,000 pixels is greater than both dimensions of the map. Note that an enemy viewing distance of 700 pixels is greater than the height of the map but not the width.

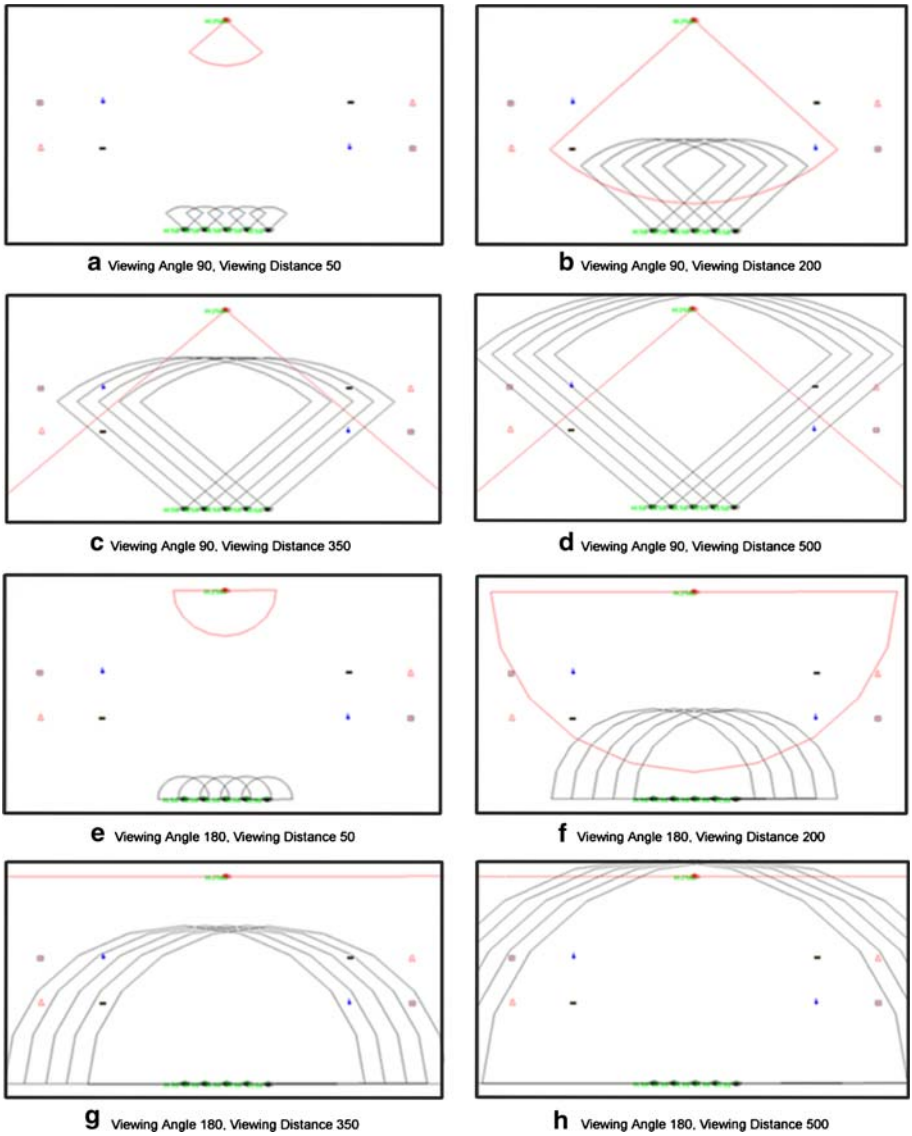
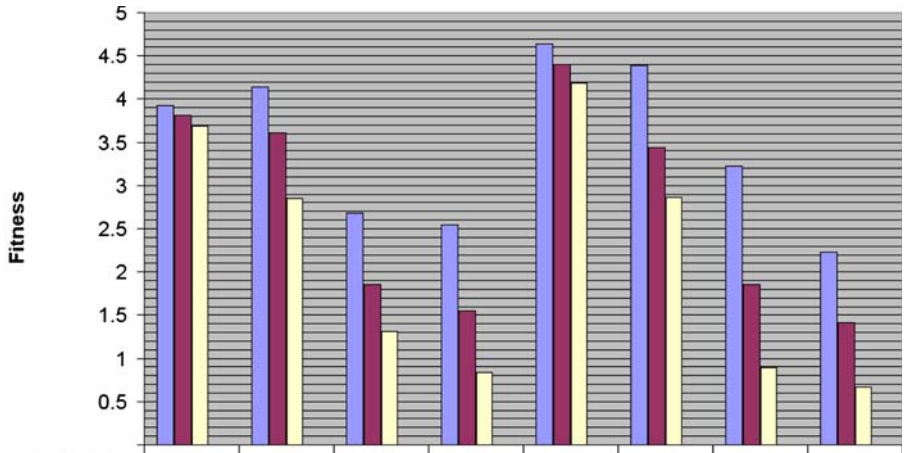


Fig. 5 Experimental setups for various fields of view and viewing distances

Each recorded team is then tested more extensively using a larger number of games to obtain a more accurate measure of its performance. The validation tests involve evaluating each recorded team's performance over 1,000 games and basing the team's fitness score on their average performance over the 1,000 games. The number of games won, lost and drawn are also recorded. Once these tests are performed for each of the recorded teams, the maximum, minimum, average and standard deviation of the team's fitness and the maximum, minimum and average of the number of wins, losses and draws are found for the 20 recorded teams in each of the eight experiments. These results are then analysed to see if environmental difficulty has a significant impact on the evolution of effective team behaviour.



Field of View	90	90	90	90	180	180	180	180
Viewing Distance	50	200	350	500	50	200	350	500
Worst	3.92	4.14	2.68	2.54	4.64	4.39	3.22	2.23
Average	3.81	3.60	1.85	1.55	4.40	3.44	1.85	1.42
Best	3.68	2.85	1.31	0.84	4.18	2.86	0.89	0.67
Std Dev	0.06	0.34	0.37	0.55	0.09	0.33	0.65	0.45

Fig. 6 Validation results for team fitness over 1,000 games

An additional test is then carried out to observe how effectively the best recorded teams from each environment perform in all other environments. This is implemented by taking the best performing team from each environment and evaluating their performance over 1,000 games in each of the other seven environments. The number of wins that each team achieves out of the 1,000 games is recorded for each environment and the results are analysed.

6 Results

The results of the experiments show that the level of difficulty of the environment does have an impact on the genetic program's ability to evolve effective team behaviours. Figure 6 shows the maximum, minimum, average and standard deviation of fitness for the twenty recorded teams in each of the eight environments over 1,000 games. Note that fitness values closer to zero are better. As is evident from Fig. 6, there is a correlation between the agent's viewing range and the genetic program's ability to evolve effective team behaviours. The results for the environments where the field of view is 90° follow a similar trend to those of environments where the field of view is 180°. This is understandable as the field of view of both the enemy and team agents are equivalent in each of the eight experiments. Thus, varying the field of view does not have as much of a bearing on the difficulty of the environment as varying the viewing distance. The *Best* bar shows the fitness of the best performing team in each environment. A steady improvement can be observed in the performance of the best evolved team as the environmental difficulty decreases. Similarly, the average fitness of all twenty recorded teams in any one environment (shown here by the *Average* bar) improves as the environments become less difficult. The worst fitness of any of the 20 recorded teams from each environment is shown by the bar labelled *Worst*. Although, the worst fitnesses

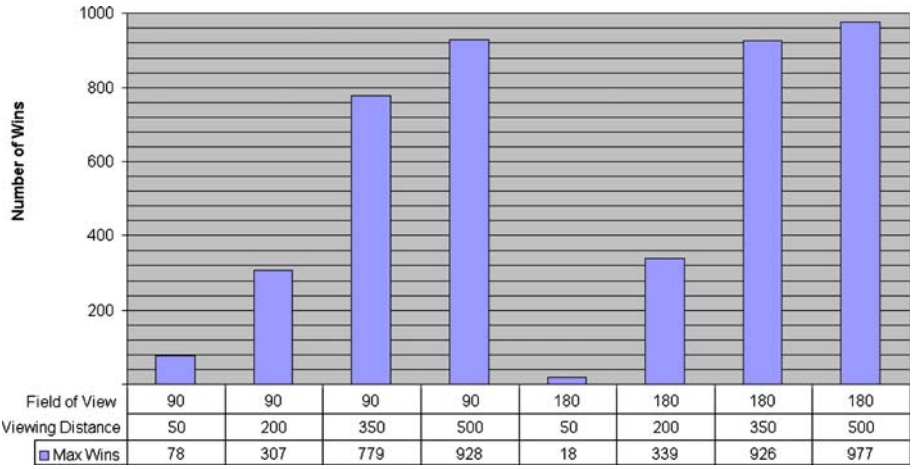


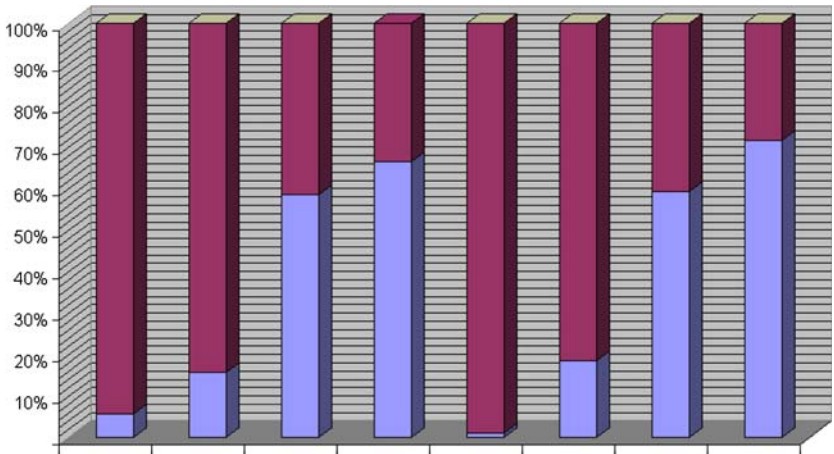
Fig. 7 Maximum number of games won by a team in each environment

generally follow a similar trend, some of the worst teams in the less difficult environments perform more poorly than some from the more difficult environments. This can be attributed to outliers in the data. Statistical tests show a statistically significant increase in fitness as the viewing distance of agents increase for a confidence interval of 95% (see Fig. 9 in Appendix).

Figure 7 shows the number of games won by the best performing recorded team in each of the eight environments. The results from Fig. 7 show that environments where the field of view is 90° follow a similar course to those for environments where the field of view is 180°. Generally, as the level of difficulty of the environment decreases, the number of games won by the best evolved team increases. This follows a similar trend to the fitness values shown in Fig. 6. The best fitness in each of the environments gradually improves as the environment becomes less difficult, i.e. as the viewing distance of agents increase. In the most difficult environments, where the viewing distance of agents is 50 pixels, the maximum number of games won by any of the recorded teams is less than 8% of games played. In contrast, the maximum number of games won by the best evolved team in the least difficult environments, where the viewing distance of agents is 500 pixels, is over 92% of games played.

As mentioned earlier, the results are comparable for both environments with 90 and 180° fields of view, as there is a steady increase in the number of games won as the viewing distance of the agents in the environment increases. Overall, the results of the experiments with the 180° field of view are generally slightly better, with the exception of the experiment where the viewing distance is 50 pixels. We believe this is because 50 pixels is too small a distance to benefit from the larger viewing angle but the enemy's viewing distance of 100 pixels in this environment does benefit, making it easier for the enemy to dispose of the team. Hence, this environment can be regarded as the most difficult of all eight environments.

Figure 8 shows the average number of wins, losses and draws, out of 1,000 games for all twenty recorded teams in each of the eight environments. There is a steady trend in the proportion of wins, losses and draws of teams as the environmental difficulty changes. In both, environments where team agents have a field of view of 90° and environments where the agents' field of view is 180°, there is a steady increase in the proportion of wins of the evolved teams as the level of difficulty of the environment decreases. In the most difficult environment, where agents have a field of view of 180 and viewing distance of 50, the proportion of wins of all 20 recorded teams is only 1.15% of games played. If the viewing



Field of View	90	90	90	90	180	180	180	180
Viewing Distance	50	200	350	500	50	200	350	500
Draws	1.05	0.45	0.05	0.00	0.25	0.35	0.15	0.05
Losses	942.35	843.05	414.10	333.50	988.25	815.15	405.90	283.90
Wins	56.60	156.50	585.85	666.50	11.50	184.50	593.95	716.05

Fig. 8 Average number of games won, lost and drawn by all recorded teams

distance is increased to 200 or 350, the proportion of wins improves to 18.45% of games played or 59.4% of games played, respectively. In the least difficult environment, with field of view 180 and viewing distance 500, the proportion of wins of the twenty recorded teams is 71.6%, which is a huge improvement on the average win percentage of teams recorded from the most difficult environment (i.e. 1.15%). This shows that the environmental difficulty has a significant bearing on the ability of the genetic program to evolve effective behaviours capable of consistently defeating the enemy.

6.1 Testing generality of evolved solutions

An additional experiment is carried out to test the generality of the evolved teams. It involves taking the very best performing team from each environment and evaluating their performance over 1,000 games in each of the other seven environments so as to assess how effectively the best teams perform in all environments. The number of wins that each team achieves in each environment is recorded (out of the 1,000 games) and the results are analysed. Table 1 shows the number of wins achieved by the best evolved team from each environment when placed in each different environment.

In Table 1, *Team090050* refers to the best genetic program (i.e. team) evolved in the environment where the agents' field of view is 90 and team agent viewing distance is 50 and so forth for the other teams. Similarly, *Env180200* refers to the environment in which the teams are tested where agents' field of view is 180° and agents' viewing distance is 200 pixels and so on. The results displayed in Table 1 show that the best performing team in each environment is the one that has been evolved in that environment (following the diagonal from top left to bottom right of table). This illustrates two things: one, that the genetic program is succeeding at finding an optimal solution in each environment and two, that no one evolved solution is optimal for all environments. This essentially means that teams cannot be evolved in the

Table 1 Number of games won by the best evolved teams in all environments

	Env090050	Env090200	Env090350	Env090500	Env180050	Env180200	Env180350	Env180500	Total
Team090050	78	70	179	33	11	51	78	11	511
Team090200	21	307	761	765	11	256	683	866	3,670
Team090350	17	112	779	758	4	79	461	802	3,012
Team090500	9	136	679	928	3	130	472	833	3,045
Team180050	62	119	390	20	18	115	225	12	961
Team180200	6	190	748	899	0	339	803	913	3,898
Team180350	16	125	618	621	2	221	926	799	3,328
Team180500	5	86	528	742	1	128	676	977	3,143
Total	214	1,145	4,682	4,766	50	1,319	4,324	5,213	

simpler, less difficult environments and used as effective teams in the more difficult environments. The last row in the table shows the total number of wins that all eight teams managed to achieve in each environment. The values in this row are a crude indicator of environmental difficulty as the lower the total number of wins achievable by all the best performing teams in any environment, the greater the difficulty of that environment. For example, the highest total number of wins are achieved in the least difficult environment, where teams manage to win 5213 out of a total of 8,000 games. Table 1 also supports our earlier assumption that the most difficult environment is the one where agents' field of view is 180 and viewing distance is 50 pixels, as all the best teams perform very poorly in this environment, winning only 50 of the 8,000 games played.

Most teams evolved in the more difficult environments perform better in the less difficult environments, as would be expected. However, the solutions evolved in the most difficult environments (i.e. where agent viewing distance is 50) do not perform well in the least difficult environments (where agent viewing distance is 500). One possible reason for this is that the solutions evolved in the more restrictive environments do not evolve to take advantage of the larger viewing distances offered by the least difficult environments and the enemy (having perfect vision in these environments) can easily overcome them.

The most general teams across all environments are those evolved in the environments where agents' viewing distance is 200 pixels. Table 1 shows that these teams win 3,670 and 3,898 games out of a total of 8,000 across all environments for the 90° field of view and 180° field of view environments, respectively. This can be attributed to these teams evolving in a fairly restrictive environment (having a viewing distance of 200 pixels), and when played in the less difficult environments where agents' viewing distance is 350 pixels and 500 pixels, they perform even better than when in their native, more difficult environment. Although their evolved tactics perform better in the less difficult environments, they do not perform as well as the native teams evolved in these environments. This is intuitive as the teams evolved in any particular environment would be evolved to specialise in that environment, and thus should perform better.

7 Conclusions

This paper explores the effect that varying the environmental difficulty has on a genetic program's ability to evolve effective team behaviours in a 2D shooter gaming environment. The gaming environment in which the game is played and genetic program to be used in the evolution are discussed, together with the experimental setup and the results of the experiments.

The results displayed in Figs. 6, 7 and 8, clearly indicate that the level of difficulty of the environment affects the evolution of effective team behaviours. The performance of evolved teams improves as the difficulty of the environment decreases. In the very difficult environments, where agents' viewing range is very restricted, teams perform very poorly. The genetic program is unable to evolve good solutions capable of consistently defeating the enemy as the environment is too difficult. In contrast, in the less difficult environments, the evolved teams perform very well as the genetic program is able to evolve effective team behaviours. As mentioned earlier, Table 1 shows that the best evolved team from each environment is at worst a local optimum solution. The fact that the best evolved solution in the more difficult environments perform so poorly leads us to believe that they are not the global optima for these environments.

In future experiments, we wish to use a technique such as fitness distance correlation (Jones and Forrest 1995) or negative slope coefficient (Vanneschi et al. 2004, 2006) to more

formally analyse the level of problem difficulty inherent in each of these environments. The fitness distance correlation technique requires the global optimum in the solution space to be known a priori in order to calculate the difficulty of the problem space. Although the global optimum is not known in advance for these environments, the best known solution can be used. However, this is not ideal, especially for the more difficult environments where the best known solution is still very poor. Negative slope coefficient, on the other hand, is a technique that does not require the global optimum to be known beforehand to calculate the difficulty of a problem. Additionally, negative slope coefficient has been shown to be successful at correctly analysing the difficulty of a number of well known benchmark problems (Poli and Vanneschi 2007; Vanneschi et al. 2004, 2006).

In addition to formally analysing the problem difficulty, we also wish to explore the effects communication between agents could have in the evolution of team behaviours, as communication would be potentially very useful in environments where agents' viewing range is more restricted. Agents could share information and perceive the environment as a team rather than individually. The position of the enemy or other game objects could be communicated amongst team agents, which could allow for much more effective team behaviours to evolve. We also wish to analyse the actual behaviours evolved to see if there are common properties in the behaviours of teams evolved in similar environments.

Acknowledgements The primary author would like to acknowledge the Irish Research Council for Science, Engineering and Technology (IRCSET) for their assistance through the Embark initiative. The authors would also like to thank the reviewers of the paper for their insightful comments and recommendations.

Appendix

The following is a list of the desirability algorithms used by the single enemy agent to determine its course of action:

–

$$Desirability_{GetHealth} = a * \left(\frac{1 - Health}{Distance\_to\_Health} \right)$$

where a is a random bias in the range 0.5 to 1.5, $Health$ is the current health of the agent and $Distance\_to\_Health$ is the distance the agent is to the nearest health pack (if this is known). If the agent does not know the location of any health packs the desirability returned from this equation will be zero. Note that if an agent's health becomes low and the location of health packs is unknown, the agent's highest desirability will be to explore the map.

–

$$Desirability_{Attack} = b * Total\_Weapon\_Strength * Health$$

where b is a random bias in the range 0.5 to 1.5, $Total\_Weapon\_Strength$ is the proportion of ammunition the agent has in its inventory for all weapons as a fraction of the total amount of ammunition it can carry for the all weapons and $Health$ is the current health of the agent. The agent will only want to attack its enemy if the agent's attacking ability (i.e. $Total\_Weapon\_Strength$) and its health level are both relatively high.

–

$$Desirability_{GetWeapon} = c * \left(\frac{Health * (1 - Weapon\_Strength)}{Distance\_to\_Weapon} \right)$$

Table 2 List of game specific parameters used in experiments and their values

Game parameter	Value
Team agent max health	50
Enemy agent max health	250
Health recovered from health pack	25
Team agent starting weapon	Blaster (unlimited ammunition)
Enemy agent starting weapon	Railgun (unlimited ammunition)
Map grid width	41 nodes
Map grid height	26 nodes
Map edge length	20
Weapon selection frequency	100 per second
Goal appraisal update frequency	200 per second
Targeting update frequency	100 per second
Trigger update frequency	400 per second
Vision update frequency	200 per second
Update interval time	0.0003333 s
Frame rate	3000 per second
Agent reaction time	0.004 s
Agent aim persistence	0.02 s
Agent aim accuracy	0.0 radians
Agent memory span	0.1 s
Separation weight for steering force	10.0
Wall avoidance weight for steering force	10.0
Wander weight for steering force	1.0
Seek weight for steering force	0.5
Arrive weight for steering force	1.0
View distance for separation	15.0
Wall detection feeler length	20
Waypoint seek distance	5
Default giver trigger range	10
Health respawn delay	0.2 s
Weapon respawn delay	0.3 s

Note that all game parameters relating to time and frequency in Tables 2 and 3 have been scaled in order to speed up the games so the genetic program runs efficiently. Games must be run at 60 frames per second and all parameters relating to time and frequency must be adjusted accordingly in order for games to be visualised.
Note also, in Table 5, if the position of the enemy or nearest ally is unknown then any environment nodes that require these by default will return a very large number (i.e. $MaxDouble$) and any position nodes that require them will return a marker to tell the agent to explore the map

where c is a random bias in the range 0.5 to 1.5, $Weapon\_Strength$ is the proportion of ammunition the agent has in its inventory as a fraction of the total amount of ammunition it can carry for the current weapon and $Distance\_to\_Weapon$ is the distance the agent is to the nearest ammunition pack for that weapon (if this is known). If the agent does not know the location of any ammunition packs for this weapon the desirability returned from the equation will be zero. Note that each weapon that ammunition can be retrieved for

Table 3 List of weapon parameters used in games and their values

Weapon parameter	Value
Blaster firing frequency	150 per second
Blaster ideal range	50
Blaster sound range	100
Blaster bolt max speed	5
Blaster bolt damage	1
Rocket launcher firing frequency	75 per second
Rocket launcher default rounds	15
Rocket launcher max rounds carried	50
Rocket launcher ideal range	100
Rocket launcher sound range	200
Rocket blast radius	20
Rocket max speed	3
Rocket damage	10
Railgun firing frequency	50 per second
Railgun default rounds	15
Railgun max rounds carried	50
Railgun ideal range	200
Railgun sound range	150
Railgun slug max speed	5,000
Railgun slug damage	10
Shotgun firing frequency	50 per second
Shotgun default rounds	15
Shotgun max rounds carried	50
Number of pellets in shotgun shell	10
Shotgun spread	0.05
Shotgun ideal range	100
Shotgun sound range	150
Pellet max speed	5,000
Pellet damage	1

Table 4 List of genetic program specific parameters used in experiments and their values

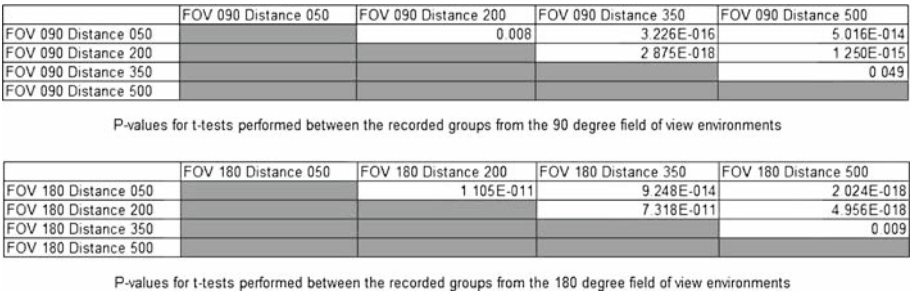
GP parameter	Value
Population size	100
Number of generations	100
Games per evaluation	20
Creation type	Ramped half and half
Creation probability	0.02
Crossover probability	0.8
Intra-team mutation probability	0.1
Swap mutation probability	0.1
Maximum depth for creation	10
Maximum depth for crossover	17

Table 5 Complete list of nodes used in experiments

Node	Node set	Description
If(C, A_1, A_2)	Action	If C is non-zero execute action A_1 else execute A_2
Move_To(P)	Action	Move to position P
Hunt_Target()	Action	Search for enemy by first trying last known position
Attack_Target()	Action	If enemy recently sensed attack else explore map
Flee()	Action	Continue to track enemy's position while retreating
Dodge_Sideways()	Action	Strafe side to side to avoid enemy fire
Get_Health()	Action	Search the map for a health pack & retrieve it
Get_Shotgun()	Action	Search map for shotgun ammunition & retrieve it
Get_Railgun()	Action	Search map for railgun ammunition & retrieve it
Get_Rockets()	Action	Search map for rocket launcher ammo & retrieve it
Explore()	Action	Explore map by wandering to a random location
Explore_Cautiously()	Action	Keep looking around while move to random location
Wait()	Action	Wait at the current location
Wait_Cautiously()	Action	Wait at the current location while looking around
Greater(N_1, N_2)	Condition	Return 1 if number N_1 greater than N_2 else 0
Smaller(N_1, N_2)	Condition	Return 1 if number N_1 less than N_2 else 0
And(C_1, C_2)	Condition	Return 1 if both conditions C_1 and C_2 true
Or(C_1, C_2)	Condition	Return 1 if either condition C_1 or C_2 true
Not(C_1)	Condition	Return 1 if condition C_1 is false, 0 otherwise
Behind_Enemy()	Condition	Return 1 if agent is behind and facing the enemy
Similar_Ally_Facing()	Condition	Return 1 if facing same direction as nearest ally
Can_See_Enemy()	Condition	Return 1 if agent can currently see the enemy
Enemy_Can_See_Me()	Condition	Return 1 if agent can see the enemy looking at it
Target_Present()	Condition	Return 1 if agent can remember sensing enemy
Distance_Enemy()	Environment	Return distance to enemy if known, else MaxDouble
Distance_Ally()	Environment	Return distance to nearest ally if known
Health()	Environment	Return current health of the agent
Enemy_Health()	Environment	Return current health of the enemy if known
Shotgun_Ammo()	Environment	Return agent's ammunition supplies for shotgun
Railgun_Ammo()	Environment	Return agent's ammunition supplies for railgun
Rocket_Ammo()	Environment	Return agent's ammo supplies for rocket launcher
Recent_Damage_Inflicted()	Environment	Returns amount of damage recently incurred by agent
Position_Behind_Enemy()	Position	Return position directly behind enemy if known
Position_Enemy()	Position	Return position of enemy if known
Position_Ally()	Position	Return position of nearest ally if known
Add(N_1, N_2)	Number	Add number N_1 to N_2 and return result
Subtract(N_1, N_2)	Number	Subtract number N_2 from N_1 and return result
Multiply(N_1, N_2)	Number	Multiply number N_1 by N_2 and return result
Divide(N_1, N_2)	Number	Divide number N_1 by N_2 and return result
Modulus(N_1, N_2)	Number	Divide number N_1 by N_2 and return remainder
Power(N_1, N_2)	Number	Return N_1 raised to power of N_2

Table 5 continued

Node	Node set	Description
Constant1()	Number	Returns the number 1
Constant2()	Number	Returns the number 2
Constant3()	Number	Returns the number 3
Constant5()	Number	Returns the number 5
Constant10()	Number	Returns the number 10
Constant100()	Number	Returns the number 100
Constant200()	Number	Returns the number 200
Constant500()	Number	Returns the number 500
Constant1,000()	Number	Returns the number 1,000



	FOV 090 Distance 050	FOV 090 Distance 200	FOV 090 Distance 350	FOV 090 Distance 500
FOV 090 Distance 050		0.008	3.226E-016	5.016E-014
FOV 090 Distance 200			2.875E-018	1.250E-015
FOV 090 Distance 350				0.049
FOV 090 Distance 500				

P-values for t-tests performed between the recorded groups from the 90 degree field of view environments

	FOV 180 Distance 050	FOV 180 Distance 200	FOV 180 Distance 350	FOV 180 Distance 500
FOV 180 Distance 050		1.105E-011	9.248E-014	2.024E-018
FOV 180 Distance 200			7.318E-011	4.956E-018
FOV 180 Distance 350				0.009
FOV 180 Distance 500				

P-values for t-tests performed between the recorded groups from the 180 degree field of view environments

Fig. 9 T-test results for recorded teams from environments with similar fields of view

(i.e. the shotgun, railgun and rocket launcher weapons) each have their own $GetWeapon$ desirability algorithm associated with them. Moreover, if the agent does not know the location of ammunition packs for any of the weapons and if its health is ok, the agent's highest desirability will be to explore the map.

–

$$Desirability_{Explore} = d$$

where d is a constant, set to 0.5 for these experiments. An agent will only explore the map if all other desirability algorithms return low values.

References

Bakkes S, Spronck P, Postma EO (2004) Team: the team-oriented evolutionary adaptability mechanism. In: Rauterberg M (ed) Proceedings of the third international conference on entertainment computing (ICEC 2004). Lecture notes in computer science, vol 3166. Springer, pp 273–282

Buckland M (2005) Programming game AI by example. Wordware Publishing, Inc., Plano, TX

Buckland M, Collins M (2002) AI techniques for game programming. Premier Press, Portland, OR

Champandard AJ (2004) AI game development: synthetic creatures with learning and reactive behaviours. New Riders Publishing, Thousand Oaks, CA

Cole N, Louis S, Miles C (2004) Using a genetic algorithm to tune first–person shooter bots. In: Congress on evolutionary computation 2004, vol 1, pp 139–145

Doherty D, O'Riordan C (2006) 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, pp 121–126

Doherty D, O'Riordan C (2007) A phenotypic analysis of gp-evolved team behaviours. In: GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation. ACM Press, New York, NY, USA, pp 1951–1958

Ehlis T (2000) Application of genetic programming to the "snake game". GamedevNet 1(175), http://www.gamedev.net/reference/articles/article1175.asp. Accessed 9th October 2003

Eskin E, Siegel E (1999) Genetic programming applied to othello: introducing students to machine learning research. In: SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on computer science education. ACM Press, New York, NY, USA, pp 242–246

Fogel DB (1993) Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe. In: Proceedings of the American power conference, IEEE, pp 875–879

Fogel DB (2002) Blondie24: playing at the edge of AI. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

Frayn C (2005) An evolutionary approach to strategies for the game of monopoly. In: Proceedings of the 2005 IEEE symposium on computational intelligence and games (CIG05)

GSC-GameWorld (2006) S.T.A.L.K.E.R.: Shadow of Chernobyl. http://www.stalker-game.com/en/

Hauptman A, Sipper M (2005) Gp-endchess: using genetic programming to evolve chess endgame players. In: Proceedings of the 8th European conference on genetic programming, pp 120–131

Haynes T, Sen S (1995) Evolving behavioral strategies in predators and prey. In: Sen S (ed) International joint conference on artificial intelligence-95 workshop on adaptation and learning in multiagent systems.. Morgan Kaufmann, Montreal, Quebec, Canada pp 32–37

Haynes T, Sen S, Schoenefeld D, Wainwright R (1995) Evolving a team. In: Siegel EV, Koza JRWorking notes for the AAAI symposium on genetic programming. AAAI, Cambridge, MA

Haynes T, Wainwright R, Sen S (1995b) Evolving cooperation strategies. In: Lesser V (ed) Proceedings of the 1st international conference on multi-agent systems. MIT Press, San Francisco, CA, p 450, citeseer.ist.psu.edu/haynes94evolving.html

ID-Software (1994) Doom. http://www.idsoftware.com/

ID-Software (1996) Quake. http://www.idsoftware.com/

Jones T, Forrest S (1995) Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Eshelman L (ed) Proceedings of the 6th international conference on genetic algorithms. Morgan Kaufmann, San Francisco, CA, pp 184–192

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA

LaLena M (1997) Teamwork in genetic programming. Master's thesis, Rochester Institute of Technology, School of Computer Science and Technology

Lassabe N, Sanchez S, Luga H, Duthen Y (2006) Genetically programmed strategies for chess endgame. In: GECCO '06: Proceedings of the 8th annual conference on genetic and evolutionary computation. ACM Press, New York, NY, USA, pp 831–838

Lionhead-Studios (2001) Black & White. http://www.lionhead.com/bw/

Luke S, Spector L (1996) Evolving teamwork and coordination with genetic programming. In: Koza JR, Goldberg DE, Fogel DB, Riolo RL (eds) Genetic programming, 1996: Proceedings of the 1st annual conference. MIT Press, Stanford University, CA, USA pp 150–156

Luke S, Hohn C, Farris J, Jackson G, Hendler J (1997) Co-evolving soccer softbot team coordination with genetic programming. In: International joint conference on artificial intelligence-97 first international workshop on RoboCup. Nagoya, Japan

Poli R, Vanneschi L (2007) Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms. In: GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation. ACM Press, New York, NY, USA, pp 1335–1342

Ponsen M (2004) Improving adaptive game-AI with evolutionary learning. Master's thesis, Delft University of Technology

Raik S, Durnota B (1994) The evolution of sporting strategies. In: Stonier RJ, Yu XH (eds) Complex systems: mechanisms of adaption. IOS Press, Amsterdam, The Netherlands pp 85–92

Reynolds CW (1993) An evolved, vision-based behavioral model of coordinated group motion. In: Proceedings of the 2nd international conference on from animals to animats 2: simulation of adaptive behavior. MIT Press, Cambridge, MA, USA, pp 384–392

Richards N, Moriarty D, McQuesten P, Miikkulainen R (1997) Evolving neural networks to play Go. In: Proceedings of the 7th international conference on genetic algorithms. East Lansing, MI

Richards MD, Whitley D, Beveridge JR, Mytkowicz T, Nguyen D, Rome D (2005) Evolving cooperative strategies for uav teams. In: GECCO '05: Proceedings of the 7th annual conference on genetic and evolutionary computation. ACM Press, New York, NY, USA, pp 1721–1728

Rosin CD, Belew RK (1995) Methods for competitive co-evolution: finding opponents worth beating. In: Eshelman L (ed) Proceedings of the 6th international conference on genetic algorithms.. Morgan Kaufmann, San Francisco, CA pp 373–380

Stanley KO, Bryant BD, Miikkulainen R (2005) Evolving neural network agents in the nero video game. In: Proceedings of the IEEE 2005 symposium on computational intelligence and games (CIG05)

Thurau C, Bauckhage C, Sagerer G (2004) Imitation learning at all levels of game–AI. In: Proceedings of the international conference on computer games, Artificial Intelligence, Design and Education, pp 402–408

Valve (2000) Counter strike. http://www.counter-strike.com

Vanneschi L, Clergue M, Collard P, Tomassini M, Verel S (2004) Fitness clouds and problem hardness in genetic programming. In: KD et al (ed) GECCO '04: Proceedings of the 6th annual conference on genetic and evolutionary computation. Lecture notes in computer science, vol 3103. Springer-Verlag, Seattle, WA, USA, pp 690–701

Vanneschi L, Tomassini M, Collard P, Vérel S (2006) Negative slope coefficient: a measure to characterize genetic programming fitness landscapes. In: Collet P, Tomassini M, Ebner M, Gustafson S, Ekárt A (eds) Proceedings of the 9th European conference on genetic programming. Lecture notes in computer science, vol 3905. Springer, pp 178–189

Yannakakis GN, Hallam J (2004) Evolving opponents for interesting interactive computer games. In: Proceedings of the 8th international conference on simulation of adaptive behaviour (SAB04), pp 499–508