Evolving the Cooperative Behaviour in Unreal^{TM} Bots

A.M. Mora, M.A. Moreno, J.J. Merelo, P.A. Castillo, M.G. Arenas and J.L.J. Laredo

***Abstract*— This paper presents an approach to the evolution of the cooperative behaviour of some bots inside the PC game Unreal^{TM}. We intend to create bots that cooperate as a team trying to beat other teams (composed of human players or bots). So, in addition to the improvement of the default artificial intelligence (AI) of bots, we have performed an improvement of the 'team AI'. We have applied an evolutionary algorithm which optimizes the parameters considered in the hard-coded states inside the bot AI code, mainly those related to the cooperation. Two different approaches have been tested inside some different battle arenas: one considering a different set of parameters for every bot in the team, and the other one considering the same set of parameters for all the teammates. The results show that both methods yield better teams than the standard ones. The teams which share the same behaviour parameters, get a higher score than those with bots playing with different parameters.**

I. INTRODUCTION

First Person Shooters (FPS) are action games, where the player can only see the hands and the current weapon of his character, and has to fight against enemies by shooting to them. These games arose at the end of the eighties to PCs as one of the new pseudo-3D games, evolving concepts previously seen in some others such as Maze Wars (1974). After the first and famous Wolfenstein^{TM} and DOOM^{TM} games, the FPSs began to be played by (millions of) videogamers in individual (or single) mode, until the appearance of the new ones which included multiplayer modes. Unreal^{TM} [1], launched for PCs by Epic Games in 1998, had a great success since it incorporates the best multiplayer mode to date. In that mode, up to eight players (in the same PC or connected through a network) fought among themselves, trying to defeat as many of the others (enemies) as possible (getting the so-called *frag* for each defeating), moving in a limited scenario (or arena), where some weapons and useful items appear frequently. The players can be human or automatic and autonomous ones, known as *bots*.

In addition, almost all these games let the programmers modify their source code to build new maps, weapons or characters, and even change the enemies' artificial intelligence (AI) schemes, to get new autonomous bots.

Following these ideas, we implemented (and presented) in a previous work [2], two approaches to evolve bots inside Unreal^{TM} : the first one was applied for tuning up a set of parameters, corresponding to some hard-coded values inside the bot AI code. The second method was implemented to change and improve the default set of rules (or states) that defines its behaviour.

Department of Architecture and Computer Technology, University of Granada, Spain, {amorag,jmerelo,pedro,maribel,juanlu }@geneura.ugr.es, fransi@correo.ugr.es

But these approaches were based on the improvement of the individual bots' behaviour, trying to get the maximum profit for each of them.

In the last FPSs, a change in the confrontation has been emerged: the *team battles*. There are plenty of team modes, such as: death match, conquer the hill, capture the flag or hunt and escape. The common aim in all of them is the cooperation of the individuals in each team to obtain a global gain. That is, the main objective is to get a good team behaviour, rather than an good individual conduct. But, in principle, it is difficult to predict how an improvement in the individual AI of a bot can profit the whole team. So, in the current paper, we have focused our work on the implementation of two approaches of team-based evolutionary methods (devoted to optimize the behaviour of the whole team), in order to obtain good results in a classical team battle (or death match): get the maximum number of frags against other teams.

Both approaches are based on the evolution of parameters, following the Genetic Algorithm based bot (G-Bot) methodology (introduced in [2]), so, an evolutionary algorithm is applied to evolve chromosomes representing a combination of values for these parameters, by playing a game and evaluating the fitness for each of them. The main difference between both methods is the consideration of a different chromosome or the same, for each bot in the same team. As stated, the objective is to get bots whose behaviour would be good for the team profit.

The rest of the paper is structured as follows. Firstly, the main features of the Unreal^{TM} environment are introduced in the next section. Then, the state of the art in this area and topics is commented is Section III. The approaches to solve the problem of improving the team bot's AI are described in Section IV. The experimental setup, runs and results are presented in Section V. Finally, in Section VI the conclusions and future work lines are exposed.

II. UNREAL^{TM} GAME FEATURES

As previously commented, Unreal^{TM} was a very famous FPS for PCs published in 1998. It presented a very good single mode, but the multiplayer possibilities gave it (and still give nowadays) a great success. Currently there are many games which include multiplayer modes against human or bots, but there are some features, which made Unreal^{TM} the best framework for developing our work:

- It includes bots with a high level AI, which was the best for a long time, since it introduced some novel techniques (such as predefined scripts or states and events)
- It includes a proprietary programming language, called *UnrealScript*, which combines the C and Java syntax,

with some useful features, such as a garbage collector. It is object-oriented and handles implicit references (every object is represented by a pointer).

- This language includes the declaration and handling of *states*, which are the most powerful feature of the Unreal bots' AI. They model one bot status and behaviour at a time, and are defined as classes. During the game (depending on the bot location and status), the current state of the bot is changed, and the functions defined in/for it are performed.
- In addition to the game, a programming environment, named *UnrealEditor* is included. It makes it easy the management of the hierarchy of classes, as well as the creation of new classes which inherits from the existing ones.

So, it is possible to change an existing class by creating another one which inherits from it, and modifying the code for the desired functions or methods. It is known as a *mod* (since it is a modification in one of the components of the original game). These way, what we have created in the previous and the current works are some mods of the existing class for bots.

On the other hand, UnrealScript has some drawbacks that make it less powerful and flexible than would be desirable:

- It can handle one dimensional arrays only.
- The number of elements in each array is limited (just around 60).
- It is very difficult to debug a running program, since it is just possible to write some short messages in the game screen or in a log file, which is usually written once the game execution has ended.
- The number of iterations in a loop is limited too, so if more than around 100000 iterations are performed, the game crashes. This is a constraint included to avoid infinite loops which can delay or freeze the game.

All these limitations have to be taken into account when programming a native bot.

In the present work we have considered Unreal Tournament^{TM} (UT) as the environment to implement and test our algorithms, since it is one of the games in this series focused in the battlematch modes, being the teams one of the most important factors in it. It preserves the excellent bot's AI (they even present a better one), and it is also possible to use UnrealScript, with less flaws than in the original game. In addition it has some improvements such as the possibility to write in specific log files (but not read) or accelerate the run, but this option change the results so, it should be applied carefully.

In this game, the most traditional combat mode is "Death Match", in which the player must eliminate as many enemies as possible before the match ends, avoiding being defeated by other players. Depending on the combat mode, the player would belong to a team or play alone.

Each player interacts with other players, trying to kill or help them, if they are his teammates or not, respectively. These players can be either humans or bots. Everyone have a limited number of health points, which are decreased as he gets hurt. If the health counter goes down to 0, the player is defeated, and a frag is added to the last player who shot him. After being killed, the player is then respawned (usually in the same place or quite near) in the game, until his maximum number of lives is reached.

In order to aid players to have success in the match, some elements appear periodically in the arena: *weapons* to defeat the enemies (with a limited ammunition, and an associated power), and *items*, that provides the player with some advantages (such as extra health, high jump, invisibility or ammunition, for instance).

When there are some teams in the battle, the objectives can be very diverse, but the typical one is to get the maximum number of frags by adding the sums obtained by each member of the team.

A match ends when the termination conditions (typically a limit of frags or time) are reached.

With respect to the powerful bot's AI implementation which Unreal^{TM} and UT present, it is based on a quite complex Finite State Machine [3], where plenty of *states* (including several *substates*) are present. Each of them, models the behaviour of the bot when it has a specific status, location in the map, or relationship with the other players (enemies or teammates). The substates represent the different 'steps' that the bot's AI can follow inside a global state (inside which they are), determining if the flow continues in the state or changes to another one. They are implemented as functions in the bot's AI code, which are usually evaluated.

Figure 1 shows an example of the flow diagram of one of the states. Specifically it is the *Roammig State*, one of the main states which the bots follow when they are searching for items or moving around the arena, deciding the next state to pass to.

One bot, during a game, changes its current state depending on some factors present in its surroundings and depending on its own status and location. That is, the states transitions strongly depend on a number of parameters (considered in the functions to evaluate) which determine the final behaviour of the bot, since most of them are thresholds depending on which, the bot state changes (for instance, the distance to an enemy or the bot's health level). These variables can be related to the individual behaviour, but some of them are also devoted to model the behaviour of the bot inside a team.

In addition, the thresholds are typically compared with hard-coded values inside the source code of the bot's AI. This way, the state changing (and the power of the bot's AI) finally depends on some constant values.

So, the topic addressed in this work has been mainly the improvement of these constants using a Genetic Algorithm [4] (GA), following the idea presented in the previous study [2], but focused this time in the team behaviour and profit.

Thus, we have implemented bots with an *Evolutionary Team AI*, since they apply a Genetic Algorithm to improve the UT AI team parameters and thresholds (and also some other individual ones). We refer them as **Genetic Team Bots**

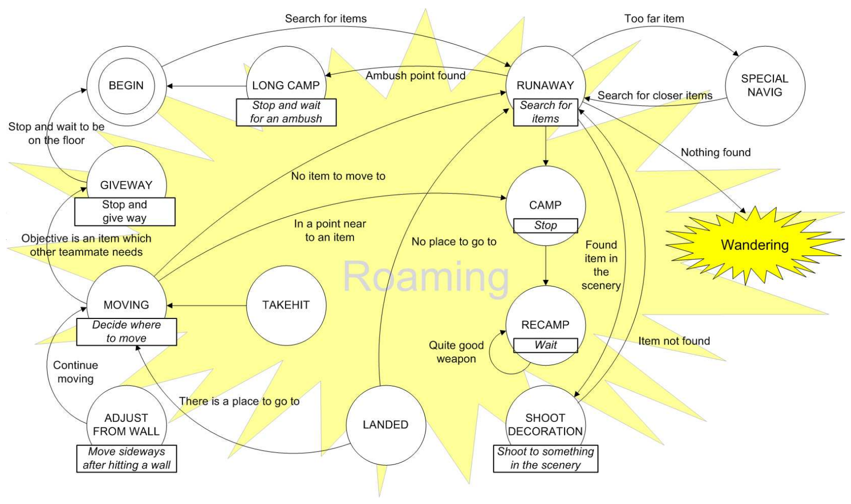

Fig. 1. Flow diagram of bot's Roaming state. The states are represented by stars, and the sub-states (or functions to evaluate) by circles.

(GT-Bots). Each of them improves its team AI by playing a game, evolving its values and getting a better team behaviour in time: defeating bots in other teams, being defeated as less as possible, letting items, weapons and ammunition to its teammates, and so on.

III. State of the Art

When they arose (became famous) in the last years of the 80s (with Wolfenstein^{TM} in 1987), the FPS games were devoted to single player modes. Later, most of them offered multiplayer possibilities but always against other human players, such as DOOM^{TM} in 1988. The first known game in including autonomous characters, the so-called bots (with a simple AI) was Quake^{TM} in 1992. It presented not only the option of playing against machine-controlled bots, but also the possibility to modify them (just in appearance or in other few aspects) or create new ones. In order to do this, the programmers could use a programming language named QuakeC, widely used in those years, but which presented some troubles, since it was strictly limited and hard-constrained. So, the bots created using it shown a simple AI (based on fuzzy logic in the best case), and it was not possible to implement more complex techniques as evolutionary algorithms, for instance.

Unreal^{TM} appeared some years later, being the first game in including (in the same pack) an easily programming environment and a more powerful language, so plenty of bots were developed. But just a few of them applied meta-heuristics or complex AI techniques, and most of these bots were based on predefined hard-coded scripts.

Nowadays, there are many games that offer similar possibilities, but almost all of them are devoted to the creation of new maps (arenas, battlefields) or characters, being mainly focused on the graphical aspect (such as modifications in their topology or appearance respectively).

The studies involving computer games and the improvement of some components of the characters' AI appeared some years ago [5], but the use of metaheuristics to study (and improve) specifically the behaviour of the bots inside FPSs, have arise in the last few years. We start our re-searching in this field in 2001, publishing our results in national conferences [6] (in Spain). We applied evolutionary techniques to improve the parameters in the bots AI core, and to change the way of controlling the bots.

Some other evolutionary approaches have been published, such as [7], where evolution and co-evolution techniques have been applied, or [8] in which an evolutionary rule-based system has been applied, and which has also been developed under the Unreal^{TM} framework (Unreal Tournament^{TM} 2004, UT2004)

In the first years, some other studies arose, such as [9], where the authors used self-organizing maps and multilayer perceptrons, or as [10], which applied machine learning, to achieve in both cases human-like behaviour and strategies, in Quake^{TM} 2 and 3 games respectively.

Recent studies related to computational intelligence, are based on bots controlled by neural networks (NNs). For instance, the authors in [11], [12] train NNs by reinforcement learning, and Schrum et al. [13] evolve NNs searching for multimodal behaviour in bots.

With respect to the evolution of cooperative bots, or team behaviour improvements, there are some studies such as the one by Bakkes et al [14], which presents TEAM, an evolutionary approach to improve the profit in the "Capture the Flag" mode in Quake^{TM}3. The key idea is that bots don't evolve by themselves, but as a team with a centralized agent control mechanism. Priesterjahn et al. [15] performed a study considering a team improvement in routing (or pathfinding), considering a communication mechanism using the scenery (simulating a stigmergic communication, like ants do).

Two years ago, Doherty et al. [16] explore the effects of communication on the improvement of team behaviour. They shown that using communication in difficult environments increases the effectiveness of the team.

Our work is devoted to study a different idea. The objective is to improve the team behaviour as a whole, by evolving the set of parameters which determines the behaviour of every team of bots in the FPS.

With respect to the possibilities of programing bots in Unreal, recently has appeared a project called Pogamut [17] which defines an interface to program these bots using Java. It is based on GameBots (GB) 2004 which is a *mod* for the game UT2004, written in UnrealScript and which provides a network text protocol for connecting to UT2004 and controlling in-game avatars (bots). So, with GB, the user can control bots with text commands and at the same time, receive information about the game environment (also in text format).

In the present work we have chosen the UT game, since it has a simple environment, being also presented in the first Unreal, but with some better features and less flaws than the previous game, so it is easier for us to improve our previous research inside the same scope.

IV. Genetic Team Bots

As previously stated, the objective in this work is to improve the behaviour of the standard bots in UT when they belongs to a team, that is, improve the whole team behaviour to get a higher profit. To do this, their default AI algorithm should be improved. It can be done by making better the values considered in the conditions assessed to change the current state (the thresholds which determine the movement to another state in the FSM that models the bot's AI).

Following the GA-Bot approach presented in our previous work [2], the idea is to perform a Genetic Algorithm which evolves (and optimizes) the values of a set of parameters, which represent some of the hard-coded constants included in the bot's AI code. These parameters determine the final behaviour of the bot, since most of them are probabilities, weights or thresholds considered in the behaviour functions (substates) of the bot; or depending on which, the bot state changes (for instance, the distance to an enemy or the bot's health level).

The first step is the selection of the set of parameters, which is quite different to the one considered in the previous work, since there are a lot of new values to consider in the UT AI core (the previous study was performed in the first Unreal$^{\text{TM}}$). In addition, this set should be focused on the improvement of the bot's AI (as previously), but taking into account the benefit of the team to which the bot belongs. Again, and due to the UnrealScript flaw related to the arrays maximum size and to the high amount of generations needed to evolve a big array, a set of 40 parameters has been chosen (instead of the ideal 60-80 set). After a deep analysis of the code, some of them where redefined as a function of others, and the less relevant ones did not have been taken into account.

The set of considered parameters is shown in Table I. They model the behaviour of the bots, and can be classified as devoted to an exclusive team profit, a both team and individual benefit (those devoted to move to or to attack the enemies in offensive states), or just an individual profit.

The GA-Bot defines each individual in the population as an array with a set of values, corresponding each of them to one of these parameters. This models, in such a way, one approach to the bot's AI. So, the GA evolves 'the behaviour of the bot' (optimizing the trigger values to change between states).

Thus, each chromosome in the population is composed by 40 genes represented by normalized floating point values (it is a real-coded GA). This way, each parameter moves in a different range, depending on its meaning, magnitude and significance in the game. The limits of the range have been estimated, conforming a width interval if the parameter is just a modifier (it is added or subtracted), and a narrow one if it is considered as an important factor in the bot's decision scheme. But previously to the evolutionary process, all of them are normalized to the [0,1] range.

The *two-point crossover* and the *simple gene mutation* (change the value of a random gene (or none) by adding or subtracting a random quantity in [0,1]) have been applied as genetic operators (see [18] for details). The GA follows the *generational + elitism* scheme, and apply the *lineal order selection* method to calculate the *selection probability (SP)* for every individual.

So, this value is calculated considering its rank in the population (ordered depending on the fitness values), instead of calculating it directly considering the value of the fitness function. This way, the domination by superindividuals (those with a very high fitness value) is avoided in the next population, to prevent a premature convergence. Once all the SPs have been calculated, a probability roulette wheel is used to choose the parents in each crossover operation. The elitism has been implemented by replacing a random individual in the next population with the global best at the moment. The worst is not replaced in order to preserve the diversity in the population.

In the previous work, the evaluation of one individual was performed by assigning the correspondent set of values in the chromosome as the parameters for a bot's AI, and placing the bot inside a scenario to fight against other bots (and/or human players). This bot was fighting until a number of global frags was reached, since it was not possible to set a time play for a bot in Unreal$^{\text{TM}}$.

In the present study, two different approaches related to the chromosome and bot's AI relationship have been implemented, since the goal is the whole team behaviour improvement:

- *one chromosome per bot (**cr-Bot**)*, where every bot in the team has associated a different chromosome in the algorithm, it is a 'different behaviour'. When a bot finishes its playing, the next available chromosome in the population is assigned to the bot. It's a kind of co-evolution.
- *one chromosome per team (**cr-Team**)*, where there is just one chromosome which is shared (is the same) by

TABLE I

PARAMETERS TO CONSIDER IN THE BOT'S AI, GROUPED DEPENDING ON THEIR INFLUENCE ON THE TEAM OR INDIVIDUAL BEHAVIOUR

Parameter	Description	Type
WeaponAIhelp	Power of the current weapon to decide if can help a teammate	Team
BestWWeightHelp	Maximum distance to help a teammate	Team
RoamFollowDist2	Maximum distance to follow a teammate	Team
MaxShareLocation	Maximum distance to share an item (let it go) with a teammate	Team
DiffHealthShare	Health points difference with a teammate to decide let him a health recovery pack	Team
pondSkill1	Used to weight the state changing, considering the bot's skills	Team/Indiv.
WeaponAI2	Used to decide if the current weapon is enough powerful or the bot needs another one	Team/Indiv.
HealthPickDest	Considered to decide if the bot has enough health to attack	Team/Indiv.
NumHuntPathPickDest	Considered in the state Hunting to decide if attack an enemy	Team/Indiv.
MaxDistAttack	In the state Attacking, maximum distance to attack an enemy	Team/Indiv.
distItemRetreat1	In the state Retreating, minimum distance to pick up an item	Team/Indiv.
probAttRetreat1	Probability of attack in the state Retreating	Team/Indiv.
BestWeightAttack	Weights the best moment to attack	Team/Indiv.
pondSkillCharg1	Probability to move from the state Charging to RangedAttack	Team/Indiv.
probRangedAttackPro	Probability to move from the state TacticalMove to RangedAttack	Team/Indiv.
SkillAdvancedTactics	Weight of the bot's skill, considered to decide if the bot can apply advanced tactics	Team/Indiv.
LocationKeepingAttack	Distance to enemy, considered to decide if attack him and move to state Hunting, to Tactical Move or nothing	Team/Indiv.
WeightTacMov1	Weights the value of an item to decide if go to pick it in the state TacticalMove	Team/Indiv.
probTMTacMov1	Considered to decide if in the state TacticalMove, the bot perform a 'StrafeMove' or a 'DirectMove'	Team/Indiv.
pondSkillTacMovPickDest	Considered to decide what TacticalMove do, according to the time passed since it took the last item	Team/Indiv.
MaxDistTacMov1	Maximum distance to enemy, considered to decide the movement in the state TacticalMove	Team/Indiv.
pondSkillTacMovPickRegDest	Weights the skills to consider in the decision of the tactical move	Team/Indiv.
probSkillTacMovPickRegDest	Probability to consider in the decision of the tactical move	Team/Indiv.
distEnemyTacMov	Distance to enemy to consider in the decision of the tactical move	Team/Indiv.
pondAgressionTacMov1	Weights the type of aggression to an enemy	Team/Indiv.
pondCombatStyleTacMov1	Weights the combat style to fight with an enemy	Team/Indiv.
pondLocRoWaChTMH	Distance to have fear from an enemy or concrete area	Individual
probPickWandering	Probability to move from state Roaming to Wandering	Individual
probChargTacMov1	Probability to move from state Tactical Move to Charging	Individual
LastTimeInvPickDestAttack	Used in the state Hunting to note the last time it takes an item	Individual
pondCollisionRadiusRoam1	Weights the collision radius	Individual
pondWHuntAttack1	Weights the distance considered to move from the state Attacking to Hunting	Individual
probWarnRetreat1	In the state Retreating, it is considered to dodge shots	Individual
pondLocationFallB1	Weights the changing of enemy when the bot is in the state Falling	Individual
VSizeAttackStakeout	Distance to enemy to change the state Hunting to Stakeout	Individual
lastSeenTimePickDestAttack	Time stamp marking the last time the bot saw an enemy (to decide if the bot tries to attack him)	Individual
probAdvancedTacticsCharg	Probability of perform an advanced tactic in the state Charging	Individual
distFindNewStakeOut	Maximum distance to a point to be watched over by the bot in the state StakeOut	Individual
pondFindNewStakeout	Weights the time stamp of the last time an enemy was seen to decide a change in the watching over point in the state StakeOut	Individual
probChangTacMov1	Probability to move from the state TacticalMove to Charging	Individual

all the bots in the team, so all of them have the 'same behaviour'. The assignment is made as a time in order to ensure that all the bots share the same chromosome.

In addition, in UT it is possible to define a time limit for playing each bot, so they compete for a similar number of seconds. It is not strictly the same because once the time limit has been reached, the bot can play until it is defeated again, which means they can stay for a different additional time, but they compete for a similar amount of seconds.

The *fitness function* has been defined considering the main factors to score the played game by a bot, and also by a team. These factors have been obtained through experimental observation, and are:

- *frags*, the number of defeated enemies by the bot
- *W*, the number of weapons the bot has picked up
- *P*, the associated power to these weapons
- *I*, the number of items the bot has collected
- *d*, the number of times the bot has been defeated
- *t*, game time the bot has been playing

So, the fitness function equation for a bot i is:

$$F_i = \frac{frags_i + [\frac{P_i}{d_i} + (\frac{W_i \cdot 10}{d_i})^{-1}] + \frac{I_i}{10} - \frac{d_i}{10}}{t_i} \quad (1)$$

where the constant values are used to decrease the relative importance of each term. So, $frags$ is the most important term in the formula. The next factor (inside square brackets) is related to the weapons and is composed by two terms: the first one considers the importance of the associated power of the picked up weapons, on average, since a player loses all the weapons once it is defeated; the second term weighs the number of weapons collected by the bot, but it is again an average. This term is inverted since it should take low values when the bot has collected lots of weapons in a life. The objective of this whole factor is to assign a higher weight to a bot which has picked up less but powerful weapons, since searching for them is a risky task, and takes some extra time. The other two factors are devoted to weigh the collected items and the number of times the bot has been defeated (with a negative weight). All the terms are divided by the time the bot has been playing to normalise the fitness of all the chromosomes (they play for a different time).

The fitness function associated to a team is obtained by adding the fitness of every bot in that team, following this equation:

$$F_T = \sum_{i \in 1..N_T} F_i \qquad (2)$$

where N_T is the number of bots in the team T.

V. Experiments and Results

We have performed several experiments to test the GT-Bots. Each of them consists in launching a game match in the *Death Match* mode for eight to sixteen players, grouped into a different number of teams, having teams with two players, or teams with four players. The players are all of them bots[1]. The bots in one (or more) of the teams are GT-Bots, so its AI is modelled using a chromosome, that can be the same or different for all the bots in the team (depending on the method which is being tested). The experiments considered (in all the runs for the GA) the parameters showed in Table II, which have been defined starting from the standard GA ones, and tuning up them through systematic experimentation.

TABLE II
Parameters of GA (used in GT-Bots).

Number of individuals	27
Mutation probability	0.01
Crossing probability	0.6
Number of generations	27
Time limit per chromosome	90 seconds

Each run takes around 20 hours, since every individual in the algorithm is playing for 90 seconds (plus the remaining time of the last life). The match is played in 'pseudo' real-time, since it is possible to increase the run speed in a game. The time limit is an advantage with regard to the number of defeats limit considered in the previous work, since the runtime also depended very much on the map where the bots were fighting so, if it was a big map, it took longer to reach this number (and change to the next individual).

In this study, several experiments have been performed, as a summary:

- *cr-Bot based team of two*: one team of two GT-Bots, considering a different chromosome per bot, versus three standard teams of two bots.
- *cr-Team based team of two*: one team of two GT-Bots, considering the same chromosome for all the bots, versus three standard teams of two bots.
- *cr-Bot based team versus cr-Team based team of two*: One team of each type fighting in the same battlefield as another two teams of two standard bots.
- *cr-Team based team of four*: one team of four GT-Bots, considering the same chromosome for all the bots, versus three standard teams of four bots.

We have performed three runs per experiment, and considered four maps (battlefields) in order to test the value of the results.

[1]although it is possible to include also human players

Firstly, Figure 2 shows the team fitness function evolution during the running of the GT-Bots, considering the *cr-Bot* approach. Both plots (Best and Mean) in every case, correspond to the average value of the fitness yielded in the three runs. There is a two GT-Bots team, fighting against three teams of two standard bots, in each one of the maps.

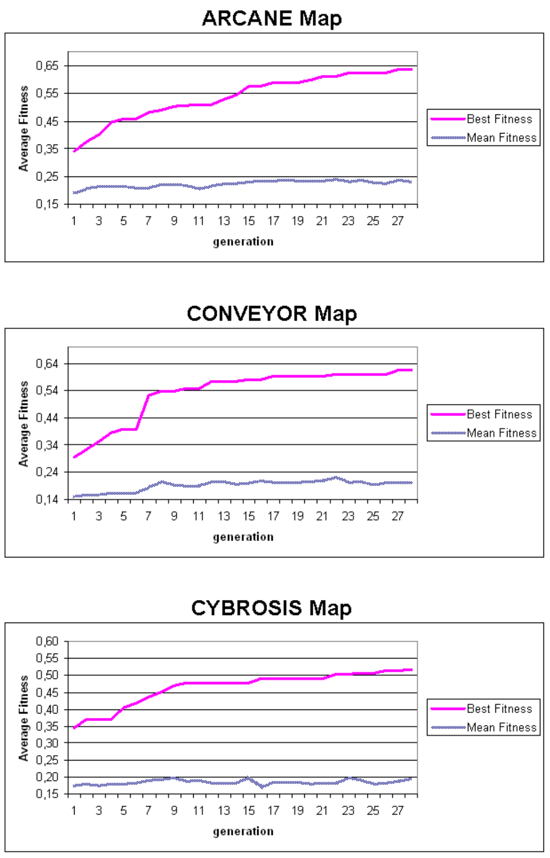

Fig. 2. Team fitness evolution with *cr-Bot* approach in three different maps. Both fitness plots are the average (of best and mean) of three runs. There are four teams of two bots (one including GT-Bots).

In this figure, a clear evolution in the best and mean team fitness are shown in all cases. The best fitness function follows a clear progression, while the mean fitness function fluctuates due to the stochastic component present in the genetic algorithms, which let to yield a worse team fitness value in any individual (and generation), expecting to get better results in the following searches. It leads to include diversity in the population. This evolution is as good as expected, which means the algorithm behaves as it is desired in all the runs and all the maps. It can be noticed a very 'soft' progression in the best fitness function due to the bots cooperation to evolve (and improve) it; since each of them consider a different chromosome in a generation, and all of them are devoted to improve the team profit (modelled with the fitness function), there is an implicit *co-evolution*, which makes the fitness better in every generation.

The team fitness function evolution considering the *cr-Team* approach is presented in the Figure 3. Again, both plots

(Best and Mean) in every case, correspond to the average value of the correspondent fitness in the three runs. The experiment performed in the DECK16 map includes a four GT-Bots team, and three standard bot teams with four bots in each one. In the rest of the maps, there is a two GT-Bots team, fighting against three teams of two standard bots.

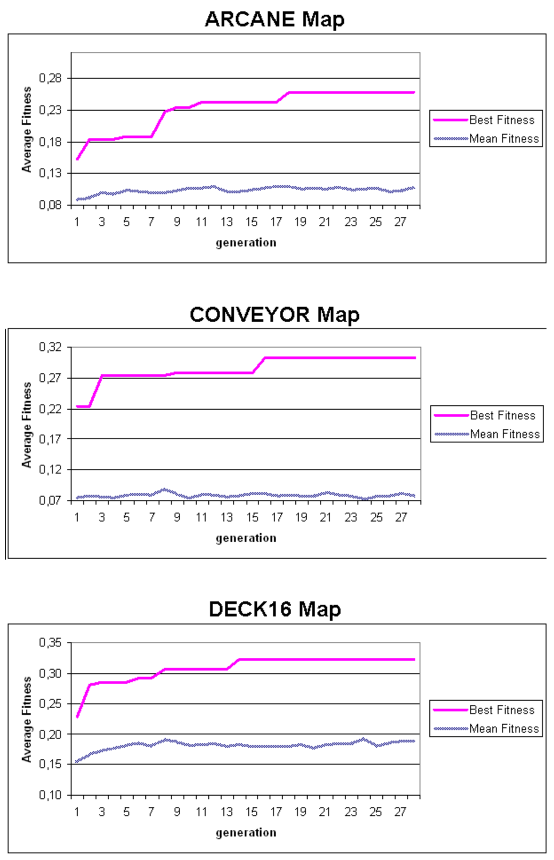

Fig. 3. Team fitness evolution with *cr-Team* approach in three different maps. Both fitness plots are the average (of best and mean) of three runs. There are four teams of two bots in the first two maps and four teams of four bots in the last one. In every case, one of the teams includes GT-Bots.

Looking at this figure, it can be noticed again the progression in both team fitness functions, but it is less 'soft' this time. The reason is that all the bots share the same chromosome at a time, so the evolution (and improvement) is slower than in the previous case (there is no co-evolution). Even so, the changes in the fitness functions are the expected, showing again some fluctuations in the mean fitness, due to the classical diversification in the evolutionary algorithms.

At the end of the run of every experiment, the best individual is considered as the set of parameters which models the final behaviour for the bot which has been evolving in the team. It continues playing until the game is stopped (by the user), since there is no stop criteria in the match.

In the next tables (III and IV), the final scores (number of frags) of one run for each of the performed experiments are presented, to show the teams value. In can be seen the number of frags got by every bot in the teams, in addition, the global score and the mean per team is showed.

So, in the Table III, the final scores yielded by the *cr-Bot* based teams in three maps, are displayed.

TABLE III

FINAL SCORES (NUMBER OF FRAGS) FOR THE THREE STANDARD AND THE *cr-Bot* BASED TEAMS IN THREE DIFFERENT MAPS.

	ARCANE Map			
	Team1	Team2	Team3	*Team GT-Bots*
Bot1	1615	1725	1787	1901
Bot2	1573	1682	1755	1858
total	3188	3407	3542	**3759**
mean	1594.00	1703.50	1771.00	**1879.50**
	CONVEYOR Map			
	Team1	Team2	Team3	*Team GT-Bots*
Bot1	1074	1236	1162	1308
Bot2	965	1196	1052	1242
total	2039	2432	2214	**2550**
mean	1019.50	1216.00	1107.00	**1275.00**
	CYBROSIS Map			
	Team1	Team2	Team3	*Team GT-Bots*
Bot1	1402	1388	1432	1575
Bot2	993	1198	1267	1555
total	2395	2586	2699	**3130**
mean	1197.50	1293.00	1349.50	**1565.00**

As can be seen in the table, the GT-Bots always beat their rivals, both at a bot level comparison, and at a team level comparison.

In the Table IV, the scores for the *cr-Team* based teams are presented, showing again a clear supremacy of the GT-Bots, even in the case of having four bots per team, where the best bot (with respect to the score) belongs to a standard team, but the best global score is again for the evolutionary team.

TABLE IV

FINAL SCORES (NUMBER OF FRAGS) FOR THE THREE STANDARD AND THE *cr-Team* BASED TEAMS IN THREE DIFFERENT MAPS.

	ARCANE Map			
	Team1	Team2	Team3	*Team GT-Bots*
Bot1	2831	2783	2931	3147
Bot2	2298	2526	2904	3064
total	5129	5309	5835	**6211**
mean	2564.50	2654.50	2917.50	**3105.50**
	CYBROSIS Map			
	Team1	Team2	Team3	*Team GT-Bots*
Bot1	1448	1616	1557	1723
Bot2	1267	1356	1412	1650
total	2715	2972	2969	**3373**
mean	1357.50	1486.00	1484.50	**1686.50**
	DECK16 Map			
	Team1	Team2	Team3	*Team GT-Bots*
Bot1	1898	2054	1962	2036
Bot2	1813	1857	1866	2013
Bot3	1776	1804	1854	2009
Bot4	1680	1769	1834	1927
total	7167	7484	7516	**7985**
mean	1791.75	1871.00	1879.00	**1996.25**

Finally, Table V shows a comparison between the final scores of both approaches (*cr-Bot* and *cr-Team*).

Looking at this table, it can be noticed that the *cr-Team* based team yields much more better results than the *cr-Bot* based team. It is just a slight comparison (just in one map),

TABLE V

Comparison of the final scores (number of frags) between the *cr-Bot* and the *cr-Team* based teams in one map.

CYBROSIS Map				
	Team1	Team2	Team *cr-Bot*	Team *cr-Team*
Bot1	4530	4573	5251	5612
Bot2	3997	4243	5102	5513
total	8527	8816	10353	**11125**
mean	4263.50	4408.00	5176.50	**5562.50**

but the high differences in the results show that *cr-Team* is a better approach, as was expected, since it evolves a whole team, while the *cr-Bot* evolves individual bots (which belongs to a team). These results have been checked long time after the algorithms runs have finished, since *cr-Bot* evolves faster (as showed in the previous figures) and get better scores in less time, so a comparison in the first steps of the algorithms, or even just when they have finished their process, would be different.

VI. Conclusions and Future Work

In this work, an evolutionary algorithm has been implemented to improve the team AI of the default bots in a PC game named Unreal Tournament^{TM}. It is a genetic algorithm, applied to optimize the decision parameters in the bots, and mainly focused in those related with the team behaviour and performance. Two different approaches have been studied: the first one (named *cr-Bot*) considers a different set of parameters (chromosome) for each bot in the team; the second one (called *cr-Team*) works with the same set of values for all the bots in the team.

Looking at the results, both approaches work as expected, reaching a clear improvement, and yielding final team bot's AI configurations which get the best scores in the matches against the teams of standard bots.

The *cr-Bot* method application implies a co-evolution in the search for the best team fitness function, while the *cr-Team* one has demonstrated to be a better option to improve a team AI, since it evolves the team as a whole.

This is our first approach to the team improvement problem, so there are many future lines of work starting from this point. The first one is the implementation of some different methods to evolve the team AI bots, in order to compare the results with those yielded by the presented approaches.

In addition we have to perform some studies to find the best parameter setting for the genetic algorithm, in order to improve its performance.

Another task to address is the implementation of these approaches (and some other previous) inside a newer engine or using different tools, in order to avoid the constraints which obstruct a better problem definition and solving (such as limited arrays and number of iterations in loops). This way would be possible to study some other team level problems, such as communication and coordination of bots.

The third line of improvement is related to the fitness function which is currently an aggregative function, so it could (or should) be separated into different functions, transforming the problem into a multi-objective one, closer to the real problem to address for getting a good bot's and team AI.

The last idea is related to the performance study of a 'pure' co-evolutionary approach, rather than a co-evolution based on teams with different individuals. The consideration of a team level co-evolution, including for instance heterogeneous teams would be also fruitful.

Acknowledgements

This paper has been funded in part by the Spanish MICYT projects NoHNES (Spanish Ministerio de Educación y Ciencia - TIN2007-68083) and the Junta de Andalucía P06-TIC-02025, P07-TIC-03044 and P08-TIC-03928.

References

[1] Wikipedia, "Unreal — wikipedia, the free encyclopedia," 2009, http://en.wikipedia.org/wiki/Unreal.

[2] A. Mora, R. Montoya, J. Merelo, P. García-Sánchez, P. Castillo, J. Laredo, A. Martínez, and A. Esparcia, "Evolving bot AI in Unreal," in *Applications of Evolutionary Computing, Part I*, ser. LNCS, C. D. C. et al., Ed., vol. 6024. Springer, May 2010, pp. 170–179.

[3] T. L. Booth, *Sequential Machines and Automata Theory*, 1st ed. New York: John Wiley and Sons, Inc., 1967.

[4] D. E. Goldberg, *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[5] J. E. Laird, "Using a computer game to develop advanced AI," *Computer*, pp. 70–75, 2001.

[6] R. Montoya, A. Mora, and J. J. M. Guervós, "Evolución nativa de personajes de juegos de ordenador," in *Actas primer congreso español algoritmos evolutivos, AEB0́2*, E. Alba, F. Fernández, J. A. Gómez, F. Herrera, J. I. Hidalgo, J.-J. Merelo-Guervós, and J. M. Sánchez, Eds. Universidad de Extremadura, Febrero 2002, pp. 212–219.

[7] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels, "Evolution of human-competitive agents in modern computer games," in *IEEE World Congress on Computational Intelligence 2006 (WCCI'06)*, 2006, pp. 777–784.

[8] R. Small and C. Bates-Congdon, "Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games," in *IEEE Congress on Evolutionary Computation 2009 (CEC'09)*, May 2009, pp. 660–666.

[9] C. Thurau, C. Bauckhage, and G. Sagerer, "Combining self organizing maps and multilayer perceptrons to learn bot-behaviour for a commercial game," in *GAME-ON*, Q. H. Mehdi, N. E. Gough, and S. Natkine, Eds. EUROSIS, 2003, pp. 119–123.

[10] S. Zanetti and A. E. Rhalibi, "Machine learning techniques for FPS in Q3," in *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2004, pp. 239–244.

[11] B. H. Cho, S. H. Jung, Y. R. Seong, and H. R. Oh, "Exploiting intelligence in fighting action games using neural networks," *IEICE - Trans. Inf. Syst.*, vol. E89-D, no. 3, pp. 1249–1256, 2006.

[12] B. Soni and P. Hingston, "Bots trained to play like a human are more fun," in *IEEE International Joint Conference on Neural Networks 2008 (IJCNN'08), in WCCI'08*, June 2008, pp. 363–369.

[13] J. Schrum and R. Miikkulainen, "Evolving multi-modal behavior in NPCs," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium On*, Sept. 2009, pp. 325–332.

[14] S. Bakkes, P. Spronck, and E. O. Postma, "Team: The team-oriented evolutionary adaptability mechanism," in *ICEC*, ser. Lecture Notes in Computer Science, M. Rauterberg, Ed., vol. 3166. Springer, 2004, pp. 273–282.

[15] S. Priesterjahn, A. Goebels, and A. Weimer, "Stigmergetic communication for cooperative agent routing in virtual environments," in *International Conference on Artificial Intelligence and the Simulation of Behaviour (AISB'05)*, 2005.

[16] D. Doherty and C. O'Riordan, "Effects of communication on the evolution of squad behaviours," in *AIIDE*, C. Darken and M. Mateas, Eds. The AAAI Press, 2008.

[17] "Pogamut 2," http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php.

[18] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Springer, 1996.