Controlling bots in a First Person Shooter game using genetic algorithms

Conference Paper · August 2010

DOI: 10.1109/CEC.2010.5586059 · Source: IEEE Xplore

5 authors, including:

Anna Esparcia-Alcázar
Universitat Politècnica de València
91 PUBLICATIONS **479** CITATIONS
SEE PROFILE

Antonio Mora
University of Granada
200 PUBLICATIONS **1,139** CITATIONS
SEE PROFILE

Juan Julián Merelo Guervós
University of Granada
541 PUBLICATIONS **4,936** CITATIONS
SEE PROFILE

Pablo García-Sánchez
Universidad de Cádiz
126 PUBLICATIONS **462** CITATIONS
SEE PROFILE

Controlling *bots* in a First Person Shooter Game using Genetic Algorithms

Anna I. Esparcia-Alcázar, *Senior Member, IEEE*, Anaís Martínez-García,
Antonio Mora, JJ Merelo and Pablo García-Sánchez

***Abstract*— In this paper we employ a steady state genetic algorithm to evolve different types of behaviour for bots in the Unreal Tournament 2004^{TM}computer game. For this purpose we define three fitness functions which are based on the number of enemies killed, the lifespan of the bot and a combination of both. Long run experiments were carried out, in which the evolved bots' behaviours outperform those of standard bots supplied by the game, particularly in those cases where the fitness involves a measure of the bot's lifespan. Also, there is an increase in the number of items collected, and the behaviours tend to become less aggressive, tending instead towards a more optimised combat style. Further "short run" experiments were carried out with a further type of fitness function defined, based on the number of items picked. In these cases the bots evolve performances towards the goal they have been aimed, with no other behaviours arising, except in the case of the multiple objective one. We conclude that in order to evolve interesting behaviours more complex fitness functions are needed, and not necessarily ones that directly include the goal we are aiming for.**

I. Introduction

Amongst the types of games that have recently deserved particular attention from the research community stand First Person Shooter games. A first person shooter, or FPS, is a kind of combat game through the first person perspective, i.e. one in which the player sees action through the eyes of a player character. An important characteristic of the game is the existence of characters other than the player. In particular, we are interested in *bots*, which can be defined as non-human players controlled by some form of artificial intelligence (AI).

A shortcoming of the AI used by bots in commercial games is that they employ methods that are hard-coded, and hence are static and require hand-tuning of parameters [1]. Among these methods we can cite finite state machines, expert systems and rule-based systems. Attempts to improve this situation with the application of various computational intelligence techniques, such as evolutionary algorithms, are relatively recent [2] and in general they employ two approaches: modifying the existing AI included in the game core or introducing alternative ways of controlling the bots.

A.I. Esparcia-Alcázar and A.I. Martínez-García are with Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain (phone: +34 96 3877069; email: {anna,amartinez}@iti.upv.es).

A. Mora, JJ Merelo and P García-Sánchez are with the Department of Computer Architecture, Universidad de Granada, Periodista Daniel Saucedo Aranda, 18071 Granada, Spain (phone: +34 958 243162; email: {amorag,jmerelo,pgarcia}@geneura.ugr.es).

For instance, in [3] the authors employ a genetic algorithm to tune the parameters of an expert system that controls the bots, achieving results which are equivalent to those obtained when it is a human expert that tunes the parameters. In [4] both approaches described above are taken. On the one hand, a genetic algorithm is employed to evolve the parameters of the Fuzzy Finite State Machine (FFSM) included in the game core that controls the bot's behaviour. On the other hand, the authors use genetic programming to evolve rules as a replacement for the FFSM, although the results reported are clearly inferior. The works in [5] and [6] also deal with rule-based systems, the former via an evolutionary algorithm and the latter by employing dynamic scripting. The latter refers to an online learning technique that uses an adaptive rulebase for the generation of game AI on the fly.

Other approaches related to computational intelligence, are based on bots controlled by neural networks (NNs). For instance, [7] train NNs by reinforcement learning and [8] evolve NNs to achieve multimodal behaviour in bots. In [9] the emphasis is placed on the integration of a neuroevolution learner with an off-the-shelf computer game.

Yet another approach aims at obtaining bots whose behaviour is more similar to that of humans, as this is considered more interesting for the player than evolving "superhuman" bots. This is the case of [10], which is inspired on mobile robotics. As opposed to standard bots, which have full information about the locations of enemies, items, enemy strongholds, etc., their bot perceives the environment through a number of sensors in a human-like fashion and hence it does not know a priori where items or enemies are. Using an evolving neural network, they try to improve those skills that a human player would improve if s/he played over a period of time, such as shooting or exploration. In [11] the emphasis is also placed on bots that are not omniscient. As a step further, in [12] the authors aim at evolving bots that imitate human behaviour which has been previously recorded from real (human) players.

In this work we aim at evolving "interesting" behaviours for a custom bot. Our approach is similar to that of [4] in that we evolve the parameters of a FFSM but differs from it in several aspects. Firstly, in that work a different version of the game is used (Unreal Tournament, as opposed to Unreal Tournament 2004, which is used here). Another main difference is that we are not interested in evolving a bot that can outperform the standard bots included in the game, but rather in studying the different behaviours that can be obtained with different fitness functions. So, while in

[4] one bot was evolved using a composite fitness function, we evolve three types of bots using three simple fitness functions. Also, in that work the individuals were evaluated using a limited time span, while here we limit the number of lives and not the time.

The rest of the paper is structured as follows. Section II describes Unreal Tournament 2004, the game we employed in our experiments. In Section III our evolutionary approach is described. Section IV provides the experimental setup and Section V presents some conclusions and outlines areas of future research.

II. The Unreal Tournament 2004^{TM} environment

Unreal Tournament 2004^{TM} (also known as UT2004 or UT2k4[1]) is a FPS in which the player moves within a virtual world in order to achieve the game goals, which can vary depending on the different kinds of playing, or combat modes. The most traditional combat mode is "Death Match" in which the player must eliminate as many enemies as possible before the match ends and avoid being killed by other players. In other combat modes, goals can range from assaulting enemy territory to capturing the enemy's flag. Goals can also be achieved by killing enemies. Depending on the combat mode, the player will belong to a team or play alone.

In the course of the match the player interacts with other players (generally trying to kill them) which can be either humans or bots, that is, characters generated by the game itself and whose behaviour can be considered equivalent to a human player behaviour. A bot is programmed to help other bots or players in its own team and attack those in a different team.

In order to allow players (either humans or bots) to attain the goals of the match, some elements appear periodically in the arena (or map). These elements are *weapons* and *items*. A *weapon* has a given capacity to hurt, or power, and a limited number of shoots. An *item* provides the player with useful characteristics, such as a temporary shield, health points or invisibility.

Each player begins its life with 100 points of health, which are decreased as it gets hurt or grow as it picks up a health item. When its health counter goes down to 0 the player dies. If the cause of death is an enemy's shot, the player which fired the shot will increase its kill counter. However, if the reason of its death is a 'suicide' (i.e. the player died due to its own action, for instance, if affected by the expansive wave of a weapon shot by himself, or if it falls in a lethal area such as a lava pool) its own kill counter will be decremented.

A match ends when the termination conditions, which are specific of the combat type, are met. In a mode such as Death Match these conditions can be a certain number of dead enemies or a time limit. Players reappear (are *respawned*) in the game after being killed, until their number of lives limit is reached.

[1]For more information about the game, see the official game web: `http://www.unrealtournament2003.com/ut2004/index.html`

In Death Match mode, the human can be a spectator, i.e. s/he does not participate in the match as a player and can only see what the rest are doing. This way of playing, known as Only-Bots, will be used here to evolve our bots.

If we want to evolve behaviours, we will have to take into account how these behaviours are represented. The behaviour of a bot in UT2004 is controlled by a Fuzzy Finite State Machine (FFSM), for which Figure 1 shows all possible states.

The FFSM differs from a standard Finite State Machine [13] in that the transitions between states follow fuzzy logic and depend on a number of variables related to the environment and the state of the bot. In particular, we are interested in three types of variables:

- Personality traits, such as aggressiveness, reaction time and combat style. These values do not directly affect the transitions between states in the FFSM, but they influence other values that do.
- Threshold values used by events and functions. These thresholds affect the behaviour of functions and events, which in turn affects the flow, and cause flags to change, which determine the current bot situation.
- Other values affecting decisions, e.g. distance to enemy.

Our aim is to employ a genetic algorithm to evolve variables of these types, as will be explained below.

III. Evolving bots for UT2004

The main objective of this work is to evolve different behaviours of a custom bot during a Death Match combat with unlimited time but with a limited number of lives (as explained later).

We consider three types of behaviour:

- Survival: The objective will be to maximise the life time of the bot, expressed in clock ticks (SurvivalBot)
- Killing efficiency: The objective is to maximise the number of enemies killed (KillerBot)
- A combination of survival and killing efficiency: This is a bi-objective problem that involves maximising the two objectives above. (MultiBot)

A further objective is to find out how skills influence one another. That is, if the improvement of some of these features improves or worsens some of the other skills. For instance, one could expect a longer lifetime to involve a greater number of kills.

To achieve these objectives we employ a steady-state genetic algorithm in order to evolve a set of characteristics for the FFSM, which at run time will control the bots' behaviour. The configuration of the algorithm is given in Table II. For the multiobjective bot we have employed NSGA-II [14], an non-elitist multiobjective evolutionary algorithm (MOEA) which was developed in order to overcome the problems of previous MOEAs, such as the high computational complexity of sorting non dominated solutions.

During the course of evolution a single match of the game is played, in which the participants are the evolved bot plus other bots spawned by the game. In the maps we are using

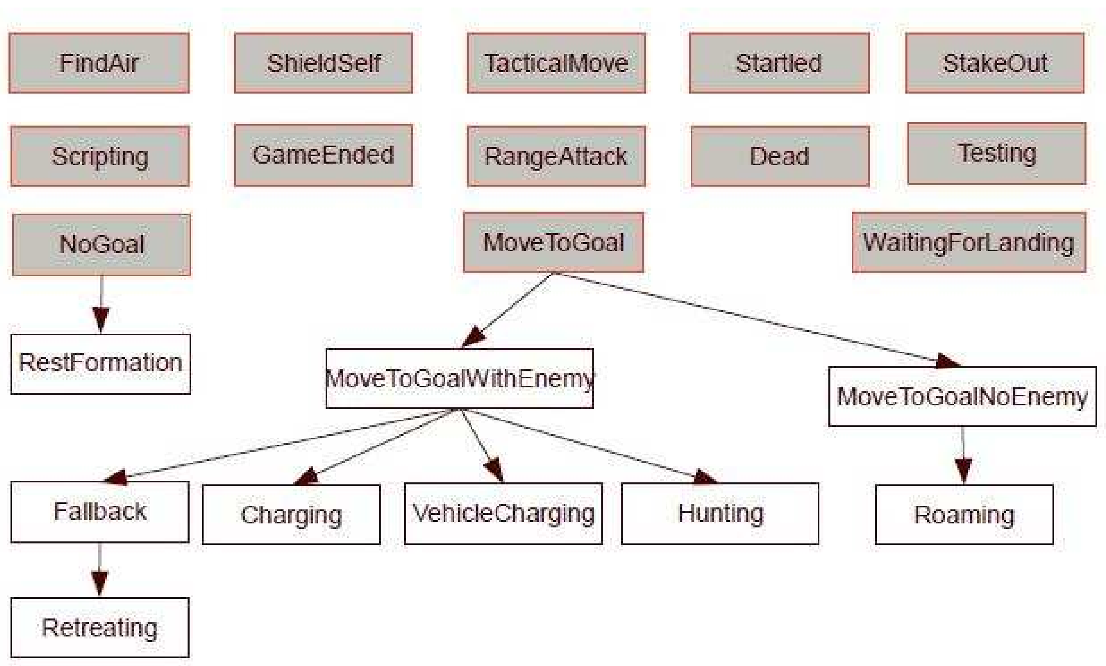

Fig. 1. Hierarchy of possible states of the game's Fuzzy Finite State Machine. Grey boxes indicate the super-states and white ones the sub-states. The transitions between them are complex and are not depicted here.

the maximum number of bots allowed by the game is four: one will be the evolved bot and other three will be spawned by the game. These have a standard behaviour given by the fuzzy finite state machine (FFSM) included in the game core. Note that all bots are controlled by the same FFSM, but final variations in bots' behaviour are given by their default characteristics.

A chromosome in our evolving population is composed of 56 genes, each representing a characteristic (or component) belonging to one of the three types described in the previous section. Each gene is structure consisting of: the component's range (the maximum and minimum values the component can take), a real value in the range $[0, 1]$ and a boolean value indicating whether to evolve the upper limit (the lower limit is not evolved because it does not make sense to have, for instance, a negative health value). The actual value of the component is then given by a linear interpolation between the range limits.

More specifically, the personality traits we evolve are:

- Accuracy (in shooting), taking values in the range [-2, 2].
- BaseAggressiveness $\in [0, 1]$, a low value indicates low aggressiveness
- CombatStyle: $\in [0, 1]$ A value close to 0 means the bot tends to snipe, while if it is close to 1 it tends to body fight (melee)
- Jumpiness $\in [0, 1]$. A high value indicates the bot can do big jumps
- ReactionTime $\in [-2, 2]$
- StrafingAbility $\in [-1, 1]$
- BaseAlertness $\in [0, 1]$, indicates the alert level of the bot, i.e. the attention it pays to external signals such as noises

The remaining characteristics evolved are thresholds to certain functions, events and states; these are listed in Table I.

In order to calculate a chromosome's fitness value, its evaluation spans over seven bot's lives. That is, when the bot dies (i.e. its health level goes down to 0), it is respawned using the characteristics given by the current chromosome for up to a total of 7 lives. When this limit is reached, the characteristics of the bot change to those given by the next chromosome to evaluate. On the other hand, when one of the regular bots dies, it is substituted by an identical one. Hence, the game is played continuously and only ends at the end of the evolution process.

IV. Experiments and Results

In our first experiment the main aim was to show the different behaviours that can be obtained. In order to do this we carried out 3 runs of the algorithms in steady state , for a total of 4800 individual evaluations each. Each individual evaluation consists of allowing the bot to play with the given chromosome characteristics for a total of seven lives.

Each run aimed at evolving a different type of behaviour, as explained above. We have termed the resulting bots as KillerBot, SurvivorBot and MultiBot. We will also refer to the bot supplied by the game as StandardBot. Although the aim is not to outperform the StandardBot, in most of the cases this is what actually happens from early in the run.

KillerBot: The behaviours of the best individual found for KillerBot and the best StandardBot of the match are

TABLE I

EVENTS, FUNCTIONS AND STATES FOR WHICH THRESHOLDS ARE EVOLVED.

Event *NotifyHitWall*	Function *CanComboMoving*	Function *YellAt*
Event *MayDodgeToMoveTarget*	Function *FightEnemy*	Function *NotifyLanded*
Function *TryWallDodge*	Function *TryToDuck*	Function *ShouldStrafeTo*
Function *AlternateTranslocDest*	Function *FaceActor*	Function *WanderOrCamp*
State "RestFormation"	State "Charging"	State "TacticalMove"
State "StakeOut"	Function *CanImpactJump*	Function *ImpactJump*
Function *RelativeStrength*	Event *MayDodgeToMoveTarget*	Event *NotifyMissedJump*
Function *FightEnemy*	Function *ChooseAttackMode*	Function *NotifyLanded*
Event *DelayedWarning*	Function *ReceiveWarning*	Function *ShouldStrafeTo*
Function *FaceActor*	Function *NeedWeapon*	Function *DefendMelee*
State "FallBack", function *FireWeaponAt*	State "VehicleCharging"	State "Hunting"
State "StakeOut", function *NotifyTakeHit*	State "TacticalMove", function *PickDestination*	

TABLE II

CONFIGURATION OF THE EVOLUTIONARY ALGORITHM. ALL NUMERICAL PARAMETERS WERE CHOSEN AFTER TRIAL AND ERROR.

Algorithm	Real-coded Steady-state Genetic Algorithm.
Encoding	The gene i represents the i component.
	Chromosome length $= 56$
Selection	Tournament in 2 steps. To select each parent, we take $tSize$ individuals chosen randomly and select the best.
Evolutionary operators	BLX-α crossover between two parents, with $\alpha = 0.05$ (see [15]). The new individual created replaces the worst individual in population if it is better than this individual
	The mutation operator changes a maximum of two genes in the chromosome.
Termination criterion	Terminate when the total number of individuals evaluated equals 5000.
Fixed parameters	Population size, $popSize = 100$
	Tournament size, $tSize = 7$
	Mutation probability: $pM = 0.1$ per chromosome, $pMg = 0.036$ per gene
	Crossover probability: $pC = 0.6$ per gene

compared in Table III. It can be seen that although KillerBot slightly outperforms the StandardBot in number of kills, the latter has a longer lifespan. In any case, the differences are not big enough to allow us to conclude that one is superior to the other.

TABLE III

COMPARISON OF THE BEST KILLERBOT AND STANDARDBOT IN THE SAME MATCH. TIMES ARE IN GAME CLOCK TICKS AND PERCENTAGES REFER TO THE LIFESPAN OF EACH BOT.

	KillerBot	***StandardBot***
Kills	15	14
Lifespan	280.19	304.81
Weapons collected	11	11
Items collected	29	39
Time in RangedAttack	(31.12%)	(33.98)
Time hunting	(15.01%)	(12.80%)
Time roaming	(29.42%)	(10.91%)
Time Retreating	(18.43%)	(4.64%)

Analysing the chromosome characteristics of KillerBot, we find that the shooting accuracy has evolved to the maximum and its reaction time is smaller. KillerBot has evolved to become a sniper (given by the "combat style" gene of its chromosome) that is, towards more effective attacks and it is perhaps the one who starts the attack rather than waiting to suffer damages.

SurvivorBot: The behaviours of the best individual found for the survival objective and the best StandardBot of the match are compared in Table IV. In this case, we can see how SurvivorBot outperforms the StandardBot in all aspects, most significantly in the lifespan. The best individual has managed to extend its life by the collection of more elements (items in particular) and the avoidance of combat. In the combat situation it has also become more efficient; this is shown in the chromosome because the accuracy has evolved to the maximum, the aggressiveness is low and the bot tends to become a sniper.

MultiBot: Table V shows the results for the two non-dominated individuals present in the final population compared to the StandardBot that obtained best results both in terms of killing and survival abilities.

The results show that if we aim at improving simultaneously the number of kills and the lifespan the tendency is to evolve towards two strategies: that of KillerBot (efficiency

TABLE IV

COMPARISON OF THE BEST SURVIVORBOT AND STANDARDBOT IN THE SAME MATCH. TIMES ARE IN GAME CLOCK TICKS AND PERCENTAGES REFER TO THE LIFESPAN OF EACH BOT.

	SurvivorBot	*StandardBot*
Kills	13	7
Lifespan	539.63	342.38
Weapons collected	11	9
Items collected	72	35
Time in RangedAttack	(26.87%)	(42.13%)
Time Hunting	(15.82)	(17.56)
Time Roaming	(29.65%)	(32.71%)
Time Retreating	(21.75%)	(5.77%)

TABLE V

COMPARISON OF THE NON-DOMINATED MULTIBOTS AND THE BEST STANDARDBOT IN THE SAME MATCH. THE ***K*** AND ***S*** DENOTE THE BEST KILLING AND SURVIVING BEHAVIOURS RESPECTIVELY. TIMES ARE IN GAME CLOCK TICKS AND PERCENTAGES REFER TO THE LIFESPAN OF EACH BOT.

	MultiBot K	*MultiBot S*	*StandardBot*
Kills	16	12	3
Lifespan	287.91	298.81	45.45
Weapons collected	11	11	8
Items collected	36	53	8
Time in RangedAttack	(39.922%)	(26.458%)	(27.437%)
Time Hunting	(15.501%)	(4.9998%)	(5.721%)
Time Roaming	(23.261%)	(35.601%)	(34.676%)
Time Retreating	(21.100%)	(30.267%)	(28.140%)

in combat) and that of the SurvivorBot (avoiding combat); also the number of items and weapons collected increases. Interestingly, MultiBot S has 25% less kills than Multibot K, but they are achieved in a much shorter time in combat. In summary, in the three cases studied evolution has been able to find a good strategy which involves increasing the number of items collected, avoiding combat where possible and if there is combat, increasing the efficiency in it. Finally, we can conclude that working towards the objective of an increased lifespan works sufficiently well to improve the global bot behaviour both in terms of number of kills or lifespan; the number of kills alone does not seem to make for a good fitness function.

In the next experiment we brought all the best bots into combat for 10 minutes in two different sceneries: the same one used for evolution (map *Calandras*) and a different one (map *Boopgod*). The results obtained are in Table VI and refer to the KillerBot that kills most and to the SurvivorBot that lives longer. Interestingly, the StandardBot does not do too badly here, outperforming the rest in number of kills and being only beaten in lifespan by the SurvivorBot. The two multiple-objective bots are clearly the losers in all aspects. However, it is in the new map where the paradox arises, with the best life of the KillerBot achieving the longest lifespan and the best life of the SurvivorBot the highest number of kills. On average, however, the SurvivorBots have both longer lifespans (as expected from their lower number of lives) and higher number of kills. Again, the multi-objective strategy does not seem to provide any advantage regarding the maximum values, although in the averages the MultiBot S shows a performance similar to the Killerbot in number of kills and superior in lifespan. The StandardBot shows a slightly worse performance than in the previous map, but still outperforming that of the MultiBot K. We must point out that the AI included in the UT games has always been regarded as one of the best in commercial videogames.

In our final experiment, we performed 5 shorter runs[2] per type of bot and map in two different maps, named *Insidious* and *Leviathan*. The termination criterion was 714 new individuals generated (after the initial population), which amounts to 714*7 = 4998 life evaluations.

To the types of bots used earlier we have added a new one, whose objective is to maximise the number of weapons and items picked, and which we have termed *PickerBot*. The results are given in Table VII. In this case there are no surprises, as we can see that each bot outperforms the rest in their own objective. Interestingly, the MultiBots compare favourably to the KillerBots, plus they have evolved a "picker" behaviour, which has been abandoned by their Killer and Survivor counterparts.

V. CONCLUSIONS AND FUTURE WORK

We have presented a steady-state genetic algorithm that evolves different bot behaviours by using two very simple single objective fitness functions and a biobjective one that combines both. This is in contrast to the work of [**?**], where a single-objective fitness function combined several of the desired characteristics.

We have shown how at the evolution stage (that is on the one-to-one comparison), the bots that incorporate the lifespan in their fitness are capable of outperforming the standard bots supplied by the game in all aspects considered (lifespan, number of kills and items collected) and from the very early stages of evolution, while the one that only includes the number of kills is only superior in this particular aspect, and that only slightly. The results obtained when the game was played after evolution are somewhat inconclusive. In this case the bots evolved incorporating the lifespan in their fitness function achieve better results in general, but they do not clearly outperform the StandardBot. Finally, in the shorter runs experiment, we can see that each fitness evolves bots suited to their specific purpose, except in the case of the MultiBot, where the evolved bots are a sort of middle term of the rest, having also developed a picker behaviour which they are not aimed for, but which can compete with those who have (at least as far as weapons are concerned).

We can see that there is an equilibrium between the different characteristics and that they are interrelated, with a

[2]These "shorter" runs lasted for about 12 hours in the computers used.

TABLE VI

COMPARISON OF THE BEST INDIVIDUALS OF ALL RUNS FOR 10 MINUTES OF PLAY IN THE SCENERY USED FOR EVOLUTION (MAP *Calandras*) AND A DIFFERENT ONE (MAP *Boopgod*). MAXIMUM VALUES REFER TO THE BEST LIFE OF EACH BOT (WITH RESPECT TO THEIR OBJECTIVE). TIMES ARE IN GAME CLOCK TICKS AND PERCENTAGES REFER TO THE LIFESPAN OF EACH BOT.

MAP CALANDRAS	KillerBot	SurvivorBot	MultiBot K	MultiBot S	Standard
# of lives	15	16	25	26	18
Kills Max.	4	3	2	3	**5**
Average	1.73	1.63	0.48	0.31	1.39
Lifespan Max.	58.22	**108.34**	40.48	64.77	91.18
Average	42.68	40.31	24.77	23.92	35.54
Weapons collected	8	11	6	8	6
Items collected	7	28	2	9	10
Time in RangedAttack	33.82%	11.17%	43.97%	30.86%	58.80%
Time Hunting	2.30%	8.54%	51.11%	22.85%	8.32%
Time Roaming	15.49%	2.45%	0.15%	11.56%	26.82%
Time Retreating	48.28%	74.85%	2.00%	34.03%	3.30%

MAP BOOPGOD	KillerBot	SurvivorBot	MultiBot K	MultiBot S	Standard
# of lives	12	8	19	10	15
Kills Max.	3	**4**	3	3	3
Average	1.17	1.75	0.53	1.10	0.60
Max. Lifespan	**208.11**	168.68	56.95	138.27	87.81
Average	53.49	74.08	33.21	63.26	41.29
Weapons collected	11	9	8	9	7
Items collected	18	17	4	20	9
Time in RangedAttack	23.71%	17.79%	26.34%	37.99%	27.82%
Time Hunting	20.05%	11.97%	5.71%	11.51%	24.92%
Time Roaming	19.72%	49.78%	15.52%	37.88%	45.23%
Time Retreating	35.68%	17.43%	52.45%	12.27%	1.59%

high performance in one aspect meaning a low performance in another, so optimising for one single aspect does not seem to be the best way to go to generate interesting behaviours. This is a complex system and optimising directly for one desired behaviour can be counterproductive; also, one can get interesting interactions between behaviours: for instance, in scenarios *Leviathan* and *Insidious* the MultiBots realise a similar performance in the weapon picking ability as the PickerBots, which are specifically aiming at that purpose. As a conclusion, we should aim to optimise several things simultaneously and let evolution come up with solutions.

As a future line of work we are interested in the simultaneous evolution of a team of bots, where each one has different skills and is capable of cooperative behaviour. This would be more suited for combat modes other than Death Match, for instance *Capture the flag*. A logical further step would be to go beyond the FFSM and directly control the bots' behaviour using a genetic program or an evolved neural network. Also, in order to make the game more entertaining we want to make the behaviour of bots closer to human behaviour. This would involve eliminating those bots characteristics that represent "omniscience", allowing for a "surprise" factor in the behaviour and being able to modify the behaviour in real time in order to adapt to new situations (including the opponents' gaming strategies).

ACKNOWLEDGMENT

This work was partially funded by project NoHNES (Spanish Ministerio de Educación y Ciencia - TIN2007-68083-C02) and projects P06-TIC-02025 and P07-TIC-03044 of the Junta de Andalucía (Spain).

TABLE VII

COMPARISON OF THE RESULTS OF RUNS OF THE BEST INDIVIDUALS OF FIVE RUNS, WITH A TERMINATION CRITERION OF 714*7 EVALUATIONS IN THE SCENERY (MAPS *Insidious* AND *Leviathan*). VALUES REFER TO THE BEST LIFE OF EACH BOT (WITH RESPECT TO THEIR OBJECTIVE). IN THE MULTIOBJECTIVE CASE, THESE REFER TO THE NON DOMINATED INDIVIDUALS

MAP INSIDIOUS					
		KillerBot	SurvivorBot	PickerBot	MultiBot
Kills	Average	12,80	7,80	5,00	9,38
	Max	17	13	7	15,00
	Min	10	5	3	6,00
	stdev	2,59	3,27	1,58	2,33
		KillerBot	SurvivorBot	PickerBot	MultiBot
Lifespan	Average	244.37	380.69	220.12	221.61
	Max	373.75	636.25	246.19	327.20
	Min	185.39	295.31	195.83	126.63
	stdev	74.70	143.44	20.99	52.51
		KillerBot	SurvivorBot	PickerBot	MultiBot
Weapons collected	Average	0.00	0.00	7.80	7.85
	Max	0	0	9	9
	Min	0	0	7	7
	stdev	0.00	0.00	1.10	0.99
		KillerBot	SurvivorBot	PickerBot	MultiBot
Items collected	Average	0.00	0.00	28.60	6.15
	Max	0	0	37	9
	Min	0	0	19	3
	std	0.00	0.00	6.73	1.99

MAP LEVIATHAN					
		KillerBot	SurvivorBot	PickerBot	MultiBot
Kills	Average	9.0	8.2	3.8	7.57
	Max	12	10	7	10
	Min	7	6	2	6
	stdev	1.87	1.64	1.92	1.40
		KillerBot	SurvivorBot	PickerBot	MultiBot
Lifespan	Average	151.50	180.47	135.73	147.68
	Max	179.44	248.67	152.72	186.80
	Min	128.91	136.17	120.08	123.00
	stdev	19.60	46.05	14.51	19.89
		KillerBot	SurvivorBot	PickerBot	MultiBot
Weapons collected	Average	0.00	0.00	9.80	9.43
	Max	0	0	**11**	**11**
	Min	0	0	9	7
	stdev	0.00	0.00	1.10	1.62
		KillerBot	SurvivorBot	PickerBot	MultiBot
Items collected	Average	0.00	0.00	25.80	8.00
	Max	0	0	34	14
	Min	0	0	22	4
	std	0.00	0.00	4.82	3.70

REFERENCES

[1] M. McPartland and M. Gallagher, “Creating a multi-purpose first person shooter bot with reinforcement learning,” in *Computational Intelligence and Games, 2008. CIG ’08. IEEE Symposium On*, Dec. 2008, pp. 143–150.

[2] J.-H. Hong and S.-B. Cho, “Evolving reactive NPCs for the real-time simulation game,” in *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05)*, 4-6 April 2005.

[3] N. Cole, S. Louis, and C. Miles, “Using a genetic algorithm to tune first-person shooter bots,” in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 1, June 2004, pp. 139–145 Vol.1.

[4] Y. e. s. this is us, “Bot evolution in Unreal Tournament with genetic algorithms,” in *A Great Congress, AGC’2010*, A. Editor, Ed. A Publisher, 2010, pp. xx–yy.

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

[6] R. Thawonmas and S. Osaka, “A method for online adaptation of computer-game ai rulebase,” in *ACE ’06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2006, p. 16.

[7] 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.

[8] 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.

[9] I. Karpov, T. D'Silva, C. Varrichio, K. Stanley, and R. Miikkulainen, "Integration and evaluation of exploration-based learning in games," in *Computational Intelligence and Games, 2006 IEEE Symposium on*, May 2006, pp. 39–44.

[10] N. van Hoorn, J. Togelius, and J. Schmidhuber, "Hierarchical controller learning in a First-Person Shooter," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, 2009.

[11] S. Jacobs, A. Ferrein, and G. Lakemeyer, "Controlling Unreal Tournament 2004 bots with the logic-based action language GOLOG," *Proc. AIIDE-05*, 2005.

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

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

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[15] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata." in *Foundations of Genetic Algorithms 2*, D. L. Whitley, Ed. San Mateo, CA: Morgan Kaufmann., 1993, pp. 187–202.