Learning a Context-Aware Weapon Selection Policy for Unreal Tournament III

Luca Galli, Daniele Loiacono, and Pier Luca Lanzi

Abstract— **Modern computer games are becoming increasingly complex and only experienced players can fully master the game controls. Accordingly, many commercial games now provide aids to simplify the player interaction. These aids are based on simple heuristics rules and cannot adapt neither to the current game situation nor to the player game style.**

In this paper, we suggest that supervised methods can be applied effectively to improve the quality of such game aids. In particular, we focus on the problem of developing an automatic weapon selection aid for Unreal Tournament III, a recent and very popular first person shooter (FPS). We propose a framework to (i) collect a dataset from game sessions, (ii) learn a policy to automatically select the weapon, and (iii) deploy the learned models in the game to replace the default weapon-switching aid provided in the game distribution. Our approach allows the development of weapon-switching policies that are aware of the current game context and can also imitate a particular game style.

I. Introduction

Modern computer games are at the same time a fascinating application domain and a very convenient testbed for the methods of computational intelligence. *First Person Shooters*, FPSs in brief, are perhaps the most popular genre of computer game. They are three dimensional shooter games providing a first person viewpoint, i.e., the player sees with the eyes of the character controlled in the game. FPSs can take place in very different scenarios (e.g., fantasy worlds, sci-fi, or second world war) but they are characterized by a rather typical gameplay: the player has to explore a world and to survive while fighting against several enemies and solving different types of challenges. Although the gameplay of FPSs looks quite immediate, the players have to learn several skills to succeed even when playing at the basic difficulty level: the player in fact has to learn both reactive behaviors (e.g., dodging, aiming) and strategical behaviors (e.g., map navigation, collecting equipment and power-ups). Accordingly, to help inexpert players most of the commercial FPSs offer aids, like an automatic selection of the weapon or an automatic aiming system. Unfortunately, these aids usually consist of simple heuristics with poor capabilities of adapting to the current game context. In addition, they are not designed for the specific needs of each player but they are typically general purpose mechanisms which do not allow any customization or adaptation.

Luca Galli, Daniele Loiacono, and Pier Luca Lanzi, are with the Politecnitco di Milano, Dipartimento di Elettronica e Informazione, Milano. Contact authors: `{loiacono,lanzi}@elet.polimi.it`

Pier Luca Lanzi is also is also member of the Illinois Genetic Algorithm Laboratory (IlliGAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA; email: `lanzi@illigal.ge.uiuc.edu`

Computational intelligence can provide feasible solutions to support users in complex and challenging gameplay scenarios, helping less skilled players while not compromising their game experience. In this paper, we focus on *Unreal Tournament III* [1], a very popular *First Person Shooter*, and present an approach to develop an automatic tool to support users during the selection of weapons. Our approach replaces the default weapon selection policy with a learned policy that has been trained by applying methods of supervised learning to data collected from previous games. Initially, we logged several games, played by experienced users or very successful bots, and collected information about their weapon selection. Then, we applied methods of supervised learning to compute models of the target weapon selection policy which was previously logged. The experimental results we present show that by collecting basic information from the Unreal Tournament server and by applying rather simple supervised learning methods it is possible to learn rather accurate models of a target weapon selection policy. Finally, we deployed the models on the actual game so as to provide a way to customize an important part of the Unreal Tournament gameplay.

The paper is organized as follows. In section II, we briefly overview the works in the literature that are related to the work presented here. We describe *Unreal Tournament* and the weapon selection task respectively in Section III and in Section IV. In Section V, we introduce the proposed methodology while in Section VI we present the experimental results. Finally, in Section VII we draw our conclusions.

II. Related Work

In the recent years, several works have investigated the application of computational intelligence to FPS games. Bakkes et al. [2] introduced TEAM, an evolutionary approach to evolve a teamplay strategy for the *Capture The Flag* game mode in *Quake 3 Arena*. More recently Hefny et al. [3] combined reinforcement and supervised learning to *Cerberus*, another *Capture The Flag* game. In particular, reinforcement learning was applied to learn a high-level team strategy while supervised learning, implemented by a neural network, was applied to model the fight behavior of each bot of the team. Several other works focused on exploiting imitation learning approaches to develop believable non-player characters for popular commercial games. Thurau with colleagues applied Bayesian imitation [4], [5], [6] and other machine learning approaches [7], [8] to develop an NPC player for Quake II. In [9], [10], a rule-based evolutionary approach was applied to evolve an effective NPC player for Quake III.

Later, in [11], [12], the same approach was combined with imitation learning to evolve human-like NPCs. McPartland and Gallagher [13] applied reinforcement learning to solve combat and navigation tasks in a self-developed FPS. Genetic algorithms have been used in [14] to evolve NPCs for an open source FPS called *CUBE* and in [15] to tune the parameters of NPCs in *Counter Strike*. Parker and Bryant [16] exploited neuroevolution and genetic algorithms to evolve NPCs for Quake II that take only visual information as input. Most of the works that dealt with evolving or learning NPCs, focused on the strategical planning, on the navigation and on combat movement.

To our knowledge, only two works considered the problem of selecting the weapon to use. Zanetti and El Rhalibi [17] applied neuroevolution to learn several behaviors of NPCs bots for Quake III from a dataset. Their set of behaviors [17] included a weapon selection behavior. Their results showed that the evolved neural network was able to choose the most powerful weapon available. In [18], Bauckhage and Thurau applied a mixture of experts to learn a context-aware policy to select the weapon in a Quake II NPC; the results reported are promising, although the choice of the weapons have been limited among three of them (i.e., the *Blaster*, the *Rocket Launcher* and the *Railgun*) and a small dataset generated ad-hoc was used.

Both [18] and [17] consider the Quake engine, which provides a simpler environment than Unreal Tournament III (the focus of this work). In addition, previous works are focused on the performance improvement of non-player characters (NPCs) whereas, in this work, we focus on improving the user gameplay experience. Accordingly, to our knowledge, this work is the first to apply supervised learning to develop customized player-oriented game aids in the context of FPS games.

III. Unreal Tournament III

Unreal Tournament III (UT3) is the last title of a very popular series of commercial *first person shooters*. It is based on the *Unreal Engine*, a very powerful and popular game engine, used by more than 30 top commercial games in the last few years. Besides its impressive rendering capabilities (see Fig. 1), the Unreal Engine also exploits the NVDIA's PhysX engine, to simulate the physics and the dynamics of the game world accurately. UT3 was developed using the *Unreal Script* programming language, a java-like scripting language interpreted by the *Unreal Engine*. This two-tier architecture allows the decoupling between the development of the underlying engine and the actual gameplay: any modification to the engine does not require a change to the scripts implementing the gameplay.

Most of the scripts developed for UT3 are publically available and can be modified to change the game behavior. Therefore, although the source code of the Unreal Engine is not available, the game itself is still highly customizable through using the Unreal Script language. In particular, there are two major approaches to modify the game behavior of UT3 using the Unreal Script technology [19]: the *Mutators* and the *Game Types*.

Fig. 1. A screenshot of Unreal Tournament 3.

Mutators are the easiest way to modify a game and how it is played. They allow to change *almost everything*: the game rules, its goals, the available weapons, the available items and power-ups, etc. UT3 comes with some built-in mutators and the users can develop their own mutators to customize the game. Mutators are designed to be applied in a chain to combine their effects. Accordingly, there are limitations on the element of the game that can be modified with a mutator in order to guarantee the compatibility with other mutators.

Game Types are typically used to change the game behavior completely or when it is necessary to perform some operations (or to access data) that are not available using mutators. Accordingly, they allow to develop games that are completely different from the original one.

IV. Automatic Weapon Selection

Weapon selection is a key factor for success in Unreal Tournament III as well as in most first person shooters. In a typical game, there are several weapons available and it is important to choose the best weapon in each situation. Table I reports a brief description of the weapons available in UT3, we refer the reader to [19] for more detailed descriptions. Weapons mainly differ in their fire rate, the damage they cause to the opponents, their range, and wether they fire instant-hit shots or not. In addition, there are weapons which can cause explosions and might damage the player itself.

This wide range of features requires a careful choice of the weapon to use depending on the current situation including the distance and the position of the opponents, the inventory, the current life points, the weapon used by the opponent, and its life points, the position in the map, etc. The choice of the most adequate weapon must also be done quickly since a mistake in the weapon used might easily lead to the player character getting killed. Weapon selection can easily become an issue for inexperienced players as they are generally too focused on moving and shooting to be able to deal with the weapon selection properly and quickly. Accordingly, UT3 (as done by commercial FPSs) offers an automatic-switching-weapon policy to help players with less experience. Unfortunately, this automatic-switching-weapon policy is extremely simple as it is based only on the ammunition levels and on a generic ranking of the weapons. UT3 has an absolute ranking of the weapons and the players weapons are sorted according to this rank. As soon as the current weapon runs out of ammunition, the next weapon in the rank is automatically selected: independently from the current situation, of the possible damages occurring to the player, and of the opponent's weapon. So for example, in a small environment, a beam weapon might be automatically switched to a missile-based weapon which causes damages to the player when operated in small spaces. However, the choice of the weapon should take into account the characteristic of the current situation and possibly the current user preferences.

TABLE I
BRIEF DESCRIPTION OF THE WEAPONS AVAILABLE IN UT3.

Weapon	Description
Impact Hammer	It is able to deal an huge amount of damage but it requires direct contact with the opponent to be effective.
Enforcer	It is a rapid fire instant hit weapon but deliberately not so accurate.
Bio Rifle	This weapon fires a big slow arcing gob of sludge that can instantly take out a heavily armed opponent.
Shock Rifle	It shoots an instant-hit beam or an explosive ball of energy that is slow but quite powerful.
Link Gun	This gun fires small and fast moving plasma balls that deal a fair amount of damage
Stinger Minigun	It fires projectiles at very high rate but do not result in high damages.
Flak Cannon	It is a very powerful weapon that fires non instant-hit chunk of metals.
Rocket Launcher	It fires seeking or non-seeking rockets. It is powerful but it is a non instant-hit weapon.
Sniper Rifle	It is a long-range rifle that fires instant-hit high caliber round.
Redeemer	It is an extremely powerful weapon that fires a small nuclear warhead. However it has only one ammunition.

V. OUR APPROACH

In this paper, we propose an approach to develop weapon selection strategies that take into account (i) several features of the current game state and (ii) the user preferences. Our methodology consists of four main steps. At first, we collect the game data of one or more matches that involve one or more players using the target weapon selection policy which we wish to learn. Then, we preprocess the collected game data to generate a dataset suitable for applying supervised methods, in the third step, to learn one or more models of the target weapon selection strategy. At the end, the model learned is deployed to the actual game engine to replace the default auto-switch-weapon mechanism.

A. Logging the Game Data

Initially, we collect all the game data that might be useful to learn a target weapon selection policy. These data include information about (i) the position of the players, which combine with the environment map provides information about the context; (ii) the inventory (i.e., weapons and ammunition) of the players; (iii) the status of the players (e.g., the health); (iv) the actions taken by the players and (v) the sensory information perceived by the players. To gather all this information during the game, an ad-hoc *game type* (see Section III), called *Logging Deathmatch*, was developed. Besides logging the game information, the *Logging Deathmatch game type* has also additional features in that it allows to select what information should be gathered, the game speed, the initial inventory of the players, which weapons will be available to the players, the initial ammunition for each weapon, etc. In this work, we configured the game to let all the players begin with all the weapons and full ammunition so as to rule out possible biases due to the strategies used for gathering the weapons and ammunitions which, otherwise, might affect the weapon selection policy. In addition, we focused on the task of learning the weapon selection for one opponent. Accordingly, the data were collected using deathmatch rounds involving only two bots. Finally, to speed up the data collection process, we did not employ human-players but publically available bots of different types so as to be able to increase the game speed. Note however, that the approach can be applied *without any modification* to game data collected from human players.

B. Generating the Dataset

In the second step, we preprocess the raw game data collected during the first step to generate a dataset suitable for the application of supervised learning techniques. For this purpose, the log is searched for weapon change events and, for each weapon change, we add an instance to the dataset containing the following information:

- the position of the player according to the game coordinate system;
- the relative position of the opponent with respect to the player;
- the ammunition levels for each weapon of both the player and the opponents;
- the life points of both the player and the opponents and also the difference between their life points;
- the weapon chosen

Overall, each instance consists of the 31 attributes and a label representing the chosen weapon.

Note that, in principle, this dataset might be generated while the game data were logged. However, we separate the logging of the data from the preprocessing to keep the methodology as general as possible. In fact, we did not want to customize the logging step with a preprocessing which usually needs to be tailored on the supervised method used. Accordingly, it is possible to use the game data logged also for other analyses based on different representation of the game state (e.g., by adding more information about the map, the opponent, etc.).

C. Learning the Weapon Selection Policy

In the third step, supervised learning is applied to the dataset previously generated to compute a model of the target weapon selection policy. For this purpose, we implemented a learning framework based on Weka [20], a well-known open source data-mining tool. In the experiments reported in this paper, we compared four methods of supervised learning: (i) Naive Bayes classifiers [21], (ii) decision trees [22], (iii) Breiman's random forests [23], and (iv) neural networks, trained using backpropagation [24]. Naive Bayes classifiers compute probabilistic classifiers based on the assumption that all the variables (the data attributes) are independent. Decision trees are a well-know approach which produce human-readable models represented as trees. In particular, in this work, we used *J48* [20], the Weka implementation of Quinlan's C4.5 [22]. Random forests [23] are ensembles of decision trees. They compute many decision trees from the same dataset, using randomly generated feature subsets and boostrapping, and generate a model by combining all the generated trees using voting. Finally, Neural Networks are a widely used [24] supervised learning method inspired by the early models of sensory processing by the brain.

D. Model Deployment

Once a model is learned, we deploy it to the actual game by replacing the existing default weapon-switching aid of UT3 with our learned model. For this purpose, we developed a *game mutator* (Section III), called *Weapon-Switcher*, that replaces the usual routine used in UT3 to change automatically the weapon. We decided to develop a *game mutator* instead of a *game type* to make it possible to include it in any types of game and to integrate it with other available mutators. The mutator we developed collects the current game state, as done for logging the game data, it preprocess the data, and use the outcome as the input for the model which then outputs a prediction about the weapon to be used. In principle, the model could be embedded in the mutator and used to predict the weapon, without the need of configuring anything. However, our goal is not to obtain an optimal weapon-selection policy to replace the one provided with the game, instead, we are interested in a policy which can be easily customized according to the player preferences. Accordingly, our framework allows the user to choose a weapon-switching policy from a library of models developed for different players, for different types of games, and possibly for different types of maps.

E. Implementation

We wanted to use state-of-the-art model implementations and therefore we used the Weka library both to compute, to load, and to apply the models. We designed a client-server architecture and developed a Java application (called *WeaponPredictor*) which acts as a server. WeaponPredictor is launched before the start of a match and it allows the user to select what model to use for the weapon-switching policy. At each game tic, the WeaponSwitcher mutator, which acts as a client during the game, collects the information about the current game state and sends it to the WeaponPredictor. The WeaponPredictor applies the classification model selected to the input received from the WeaponSwitcher and sends back the label of the instance, i.e., the weapon to choose. Finally the weapon is changed in the game by the WeaponSwitcher accordingly to the suggestion of the WeaponPredictor.

VI. Experimental Results

We performed a set of experiments to test our framework. All the experiments were performed using the implementations available in Weka [20] with the default parameter settings. Initially, we performed an exploratory analysis of the dataset generated from the logged data to check for noise, class distribution and, generally, to get some insight about the problem. Then, we applied four supervised methods to compute accurate models of the weapon switching behavior traced by the data and compared them in terms of predictive accuracy. Finally, we repeated the same process including an initial resampling step so as to get rid of class imbalance.

Experimental design. To collect the game data for our experiments we ran a deathmatch with two *Godlike* bots of UT3, i.e., the bots with the highest skill level. We collected the data in the *DM-Deck* map because it is a map available in every copy of UT3 and it has well-balanced areas with lifts, bridges, open space and tunnels. For these reasons this map is able to represent the vast majority of situations that a player can encounter during a match. Game data was collected for approximately 12 hours of simulation and resulted in a dataset of 19095 instances.

Performance Measures. To compare the predictive performance of the four supervised methods considered, we applied a 10-fold stratified crossvalidation using classification accuracy and Cohen's κ coefficient. The former gives us a basic measure of the performance of the classification model learned, while the latter is generally used to estimate the classifying agreement in categorical data, i.e., it provides an estimate of how significant the results are: the more the κ coefficient is close to one, the more significant is the correlation between the prediction and the target values; while, a negative κ coefficient indicates that there is no agreement.

A. Exploratory Analysis

Figure 2 reports the distribution of selected weapons in the dataset. The weapon distribution is extremely skewed and three weapons are the most used: the rocket launcher, the flak cannon and the redeemer (see Table I for details). In particular, the rocket launcher and the flak cannon are by far used more often than the Redeemer which can be used

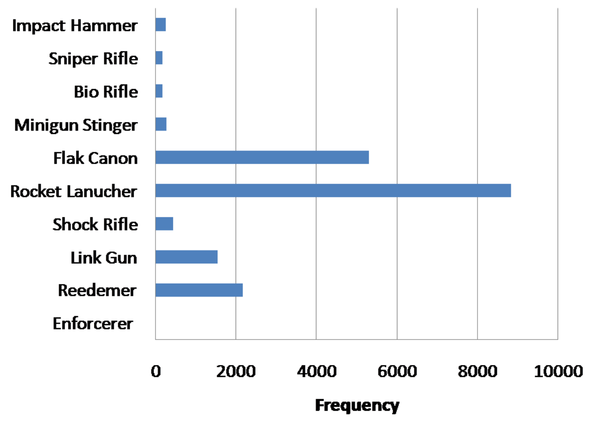

Fig. 2. Distribution of the weapon selected in the dataset.

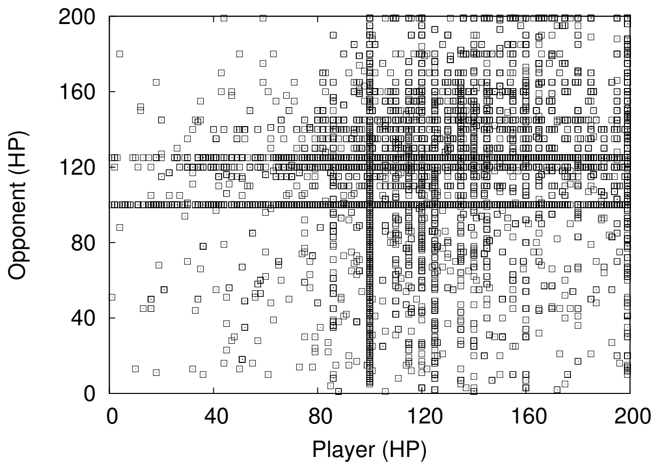

Fig. 3. Distribution of the health points of the player and of the opponent in the dataset.

only once since it has only one ammunition. Furthermore, the Enforcerer is never used during the game. In our opinion, the distribution in Figure 2 suggests that the weapons of UT3 have not been carefully designed as some of them are too powerful and convenient to use with respect to others. This is also confirmed by many comments of several expert players of the UT3 community. Figure 3 reports the distribution of health points of the player and of the opponent. The dataset covers almost all the range between 0 and 200 points (the highest number of health points that can be achieved through a power-up). In particular, it can be noticed that the most frequent situation is when at least one of the characters has 100 health points, as it is the initial value. However, there are also many situations where at least one of the characters has more than 100 health points. This is probably due to the experimental setup which involves only two characters allowing enough time between fights for the players to collect several power-ups.

B. Model Performance

In the first set of experiments, we applied the four supervised methods (Naive Bayes classifiers, decision trees, random forests, and Neural Networks) to learn a model of weapons-switching policy from the collected dataset which is subject to a huge class imbalance (Figure 2). The experiments were performed using Weka with the default parameter settings distributed with the software.

Table II compares the accuracy obtained by the four classifiers. As can be noted, all the classifiers perform rather poorly and they are not very accurate but, as suggested by the positive κ coefficients, they are significantly better than a random guess: in a 9-class problem the accuracy of a random guess would be roughly equal to 11%. The low accuracy of Naive Bayes classifiers suggests that the problem variables are not independent as this approach assumes. In contrast, decision trees and random forests achieve better accuracy than Naive Bayes and the Neural Networks. Figure VI-B shows an example of a simplified tree learned applying J48 to the dataset. The model first checks whether the player has ammunition available for the Redeemer. It is interesting to note that even when the ammunition for the Redeemer is available, the Rocket launcher is still preferred if the player has more health points with respect to the opponent, i.e., the redeemer is used only when the situation for the player is somehow dangerous.

C. Model Performance with Class Resampling

The exploratory analysis (Figure 2) shows that the class distribution in the dataset is highly unbalanced resulting in a generally low classification performance. Accordingly, we added a resampling step (using the corresponding operator provided with Weka [20]) to rebalance the classes so as to generate a new dataset with a uniform distribution of weapons. Table III compares the predictive performance obtained by the four supervised methods on the resampled

TABLE II
COMPARISON OF THE FOUR CLASSIFIERS ON THE ORIGINAL DATASET: ACCURACY AND κ COEFFICIENT.

Classifier	Accuracy	κ
Naive Bayes	36.78%	.1113
J48	54.55%	.3210
Random Forest	60.68%	.3882
Neural Network	49.68%	.1784

```
Redeemer Ammo <= 0
| deltaY <= -1846.76
| | deltaX <= 527.24: Link Gun (434.0/263.0)
| | deltaX > 527.24: Rocket Launcher (394.0/229.0)
| deltaY > -1846.76: Rocket Launcher (4941.0/2599.0)
Redeemer Ammo > 0
| deltaHP <= 0
| | deltaY <= 1723.88
| | | deltaX <= 136.41: Rocket Launcher (4230.0/2058.0)
| | | deltaX > 136.41
| | | | deltaZ <= -848.55: Rocket Launcher (1613.0/859.0)
| | | | deltaZ > -848.55
| | | | | deltaHP <= -13: Redeemer (437.0/263.0)
| | | | | deltaHP > -13: Rocket Launcher (523.0/242.0)
| | deltaY > 1723.88: Redeemer (312.0/164.0)
| deltaHP > 0: Rocket Launcher (6211.0/3450.0)
```

Fig. 4. An example of decision tree learned with J48.

dataset. Naive Bayes classifiers still have a poor accuracy and this is probably due to the underlying assumption that the problem variables are independent. Neural networks also obtain a relatively low accuracy, although their performance has been improved with the balanced dataset. In contrast, tree-based approaches have significantly improved their performance and the κ coefficient confirms that the results are statistically significant.

TABLE III
COMPARISON OF THE FOUR CLASSIFIERS ON THE RESAMPLED DATASET: ACCURACY AND κ COEFFICIENT.

Classifier	Accuracy	κ
Naive Bayes	33.38%	.2506
J48	83.3150%	.81
Random Forest	87.58%	.8603
Neural Network	58.64%	.5346

We added resampling to improve the performance of supervised methods and, as shown by the results in Table III, it was effective in improving the performance of decision trees and random forests. However, resampling introduces a bias in the evaluation of the results in that it assumes that misclassification errors have all the same costs, although, during a match some errors in the weapon switching might be worse than others. Accordingly, the proposed approach might be improved by introducing a cost matrix to weight each misclassification error according to the its importance, e.g., choosing a sniper rifle instead of a rocket launcher is generally worse than choosing a flak cannon instead of the rocket launcher. The definition and the analysis of a cost matrix, based on player preferences and specific domain knowledge, is beyond the aim of this work but it is the subject of our current investigations.

D. Deployment

The models obtained during the previous steps have been deployed to the actual game engine and analyzed in qualitative terms using human-players (students, colleagues, and friends). Our qualitative analysis suggests that the learned policy are generally better than the default automatic weapon selection aids provided with the game, as they are able to deal effectively with the game context. To validate the models we also played against the same UT3 bots used to collect data for the analysis. During the games it was possible to observe that our weapon (which was automatically selected using our deployed models) was the same one used by the opponent bot in the same situation. Therefore the deployed model was coherent with the bot used to generate the model itself.

VII. CONCLUSIONS

In this paper, we applied supervised learning to design an automatic weapon selection aid for *Unreal Tournament III*. The proposed methodology involves four steps: (i) collect data from one or several game sessions; (ii) generate a dataset from the data collected representing the weapon selection policy and (if needed) apply resampling to obtain a uniform distribution of weapons; (iii) apply supervised methods of preference to compute a model of the weapon selection policy; (iv) deploy the model learned in the game.

To test our approach we compared the performance of four supervised methods on a dataset generated using, for the sake of simplicity, the weapon selection policy of programmed bots. The analysis of the dataset collected showed that the distribution of the weapons selected is highly unbalanced, suggesting that, perhaps, the weapon balance in UT3 has not been carefully designed (a hypothesis confirmed by the comments of several UT3 expert players that have been published online). Accordingly, we resampled the dataset to improve the performance of the supervised methods considered. Our results show that random forest and decision trees outperform the other approaches and can achieve reasonable performance on the original dataset and very good performance on the resampled dataset (reaching respectively the 83.31% and 87.58% accuracy). Finally, we deployed the weapon selection policies learned and performed a qualitative analysis during the game involving human-players. Our analysis suggested that the weapon switching aids based on our approach are generally better than the usual automatic weapon selection mechanism provided with Unreal Tournament. Therefore, although our results are still preliminary, this research direction appears to be very promising. Future works will include the introduction of cost matrices during model building using both domain knowledge and on the players preferences, so as to improve the policies, to provide a better customization, and to improve the quality assessment process.

REFERENCES

[1] "Unreal tournament 3 webpage," http://www.unrealtournament3.com/. [Online]. Available: http://www.unrealtournament3.com/

[2] 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.

[3] A. S. Hefny, A. A. Hatem, M. M. Shalaby, and A. F. Atiya, "Cerberus: Applying supervised and reinforcement learning techniques to capture the flag games," in *AIIDE*, C. Darken and M. Mateas, Eds. The AAAI Press, 2008.

[4] C. Thurau, T. Paczian, and C. Bauckhage, "Is bayesian imitation learning the route to believable gamebots?" in *GAMEON-NA*, H. Vangheluwe and C. Verbrugge, Eds. EUROSIS, 2004, pp. 20–24.

[5] B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys, "Bayesian imitation of human behavior in interactive computer games," vol. 1, IEEE. IEEE, 2006, inproceedings, pp. 1244–1247. [Online]. Available: files/papers/Gorman2006-BIO.pdf

[6] ——, "Believability testing and bayesian imitation in interactive computer games," vol. LNAI, Springer. Springer, 2006, inproceedings. [Online]. Available: files/papers/Gorman2006-BTA.pdf

[7] 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.

[8] ——, "Learning human-like movement behavior for computer games," 2004, inproceedings. [Online]. Available: files/papers/ Thurau2004-LHL.pdf

[9] S. Priesterjahn and A. Weimer, "An evolutionary online adaptation method for modern computer games based on imitation," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 344–345.

[10] S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels, “Evolution of human-competitive agents in modern computer games,” 0-0 2006, pp. 777–784.

[11] S. Priesterjahn, “Imitation-based evolution of artificial players in modern computer games,” in *GECCO ’08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 1429–1430.

[12] S. Priesterjahn, A. Weimer, and M. Eberling, “Real-time imitation-based adaptation of gaming behaviour in modern computer games,” in *GECCO ’08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2008, pp. 1431–1432.

[13] M. McPartland and M. Gallagher, “Creating a multi-purpose first person shooter bot with reinforcement learning,” in *CIG*, 2008.

[14] R. M. Young and J. E. Laird, Eds., *Adding Smart Opponents to a First-Person Shooter Video Game through Evolutionary Design*. AAAI Press, 2005.

[15] N. Cole, S. J. Louis, and C. Miles, “Using a genetic algorithm to tune first-person shooter bots,” in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2004, pp. 139–145.

[16] M. Parker and B. Bryant, “Visual control in quake ii with a cyclic controller,” in *CIG*, 2008.

[17] 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.

[18] C. Bauckhage and C. Thurau, “Towards a fair ’n square aimbot – using mixtures of experts to learn context aware weapon handling,” in *GAME-ON*, A. El-Rhalibi and D. van Welden, Eds. EUROSIS, 2004, pp. 20–24.

[19] “Unreal wiki,” http://wiki.beyondunreal.com/. [Online]. Available: http://wiki.beyondunreal.com/

[20] “Weka wiki,” http://weka.wiki.sourceforge.net/. [Online]. Available: http://weka.wiki.sourceforge.net/

[21] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[22] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[23] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[24] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: Macmillan, 1994.