Decision Tree-Based Algorithms for Implementing Bot AI in UT2004

Antonio J. Fernández Leiva and Jorge L. O’Valle Barragán

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
Campus de Teatinos, Universidad de Málaga,
29071 Málaga – Spain
afdez@lcc.uma.es, jlobarragan@gmail.com

Abstract. This paper describes two different decision tree-based approaches to obtain strategies that control the behavior of bots in the context of the Unreal Tournament 2004. The first approach follows the traditional process existing in commercial videogames to program the game artificial intelligence (AI), that is to say, it consists of coding the strategy manually according to the AI programmer’s experience with the aim of increasing player satisfaction. The second approach is based on evolutionary programming techniques and has the objective of automatically generating the game AI. An experimental analysis is conducted in order to evaluate the quality of our proposals. This analysis is executed on the basis of two fitness functions that were defined intuitively to provide entertainment to the player. Finally a comparison between the two approaches is done following the subjective evaluation principles imposed by the “2k bot prize” competition.

1 Introduction and Related Work

Until recently, research on videogames was mainly focused on having more realistic games by improving graphics and sound. However, in recent years, hardware components have experienced exponential growth and players demand higher quality opponents controlled by better artificial intelligences (AI). In this context AI plays an important role in the success or failure of a game and some major AI techniques have already been used in existing videogames [1][2] (e.g., evolutionary computation and neural networks are beginning to be considered with moderate success [3]). However, traditionally game developers have preferred standard AI techniques such as Artificial Life, Neural Networks, Finite State Machines, Fuzzy Logic, Learning, and Expert Systems, among others [4][5]. One technique used with success for implementing the game AI in First Person Shooter (FPS) games is Decision Tree (DT); a decision tree is basically a tree that takes as input a specific situation (e.g., a combination of values corresponding to a set of perceptions) and outputs a yes/no decision. In other words, a decision tree is a tree in which its internal nodes represent questions that can be answered with yes or no, and its leaves are actions to be executed. Decision trees have already been employed as decision-making techniques in successful AAA
videogames (e.g., Crysis). The reasons for their use are clear: DTs are based on a simple concept, can be easily interpreted, and helps to smooth the complexity of other techniques (e.g., finite state machines are also easy to understand and be interpreted but they introduce a high complexity if the model involves a high number of states as it is not always easy to manage the whole set of relations among all the states).

In FPS games, requiring higher quality opponents means obtaining enemies exhibiting intelligent behavior; however, it is not easy to evaluate what a ‘human-like intelligence’ means for a bot in these games. Generally speaking, it is well known that the Turing Test is a procedure proposed by Alan Turing to corroborate the existence of intelligence in a machine [6]. The basic fundament is that a machine that behaves intelligently might be considered as intelligent in the human sense. In this context, the “2k bot prize” is a competition that proposes an interesting adaptation of the Turing test in the context of the Unreal Tournament 2004 (UT2004), a multi-player online FPS game in which enemy bots are controlled by some kind of game AI.

This paper describes an on-going work to provide intelligence to the bots in the context of UT2004, and we propose here two different DT-based algorithms. These two different approaches are compared experimentally, first from an objective point of view considering two fitness functions, and second from a subjective point of view according to the “2k bot prize” competition.

2 Proposals to Control the Bot AI

We have tested two different ways to generate the bot AI in UT2004. The first implementation is manufactured via decision trees and was manually coded following our intuition under a process of trial, error, and debugging. In fact, this is basically the process followed in most of the existing commercial FPS games. The second of our proposals consists of automating the process of generating an adequate strategy for the bot AI and is based on genetic programming (GP) techniques [7]. In the following we describe both proposals.

2.1 A Hand-Made Decision Tree-Based AI

In a high level of abstraction the logic of the bot was manually coded as machine with the following 4 states:

- Combat (if the bot is under attack).
- Pick items (if the bot detects an item).
- Pursue (if the bot detects an enemy).
- Idle (otherwise).

Each of these states is implemented with a specific hand-coded binary DT (i.e. internal nodes have two children at most). Internal nodes represent bot perceptions that involve a question with two possible answers: yes or no, and leaves

contain actions that have to be directly executed by the bot; note that a specific action associated to a leaf is executed when all the internal nodes in the path from the tree root to the leaf have been traversed. The perceptions that the bot perceives are of the type “have I an item close?”,” am I under attack?”, “do I see an enemy?”,” am I armed?”,” do I see an item?”, etc. The actions that a bot can execute are of the type “shoot”, “do nothing”, “re-equip with weapons”, “look for a specific item (i.e., health kit, weapon, armor, etc.)”, “pick item”, “turn (right, left)”, “jump”, . . . , etc. Due to the complexity of these trees we will not show them completely. Figure 1 shows a (very simplified) illustration of the tree associated to state “Pick Item” (here the perception “Have I chosen an item to look for?” and the actions “Continue looking for it”, and “Choose item to look for” are over-simplifications corresponding to more complex DTs.

2.2 A Genetic Programming Based Approach

Note that the approach previously described requires an important effort from the programmer point of view as the strategy that controls the behavior (i.e., the logic) of the bot has to be intuitively coded. In fact, this strategy was the result of directly applying our experience as players. We have also considered an automated process to generate bot AI. This process is based on genetic programming (GP) techniques [7]. More specifically, we have implemented a GP algorithm to optimize the bot AI. The basic idea here is that the individuals are represented as decision trees (in fact as 4-states machines as explained in the previous section) where the actions correspond directly with terminal symbols and the perceptions are associated with non-terminal symbols. Our automated algorithm is a standard GP method that uses the following parameters: random initialization via the classical “grow” method, binary tournament selection, crossover based on random node selection and interchange of the respective subtrees, mutation based on the replacement of a complete subtree, and replacement of the worst parent. Our GP algorithm manages the following set of symbols:

\[
\mathbf{T}(\text{Terminals}) = \text{Jump, RunToTarget, RunAway, Attack, Shoot, StopShooting, Disarm, Arm, TurnSomeDegrees, GoToPosition, Idle, PickWeapon, PickMedicalKit, PickArmor, PickAmmunition, NoOperation.}
\]
NT(Non-terminals) = ClosestWeapon, ClosestArmor, Distance, ClosestEnemy, ClosestMedicalKit, ClosestAmmunition, SeeEnemy, VisibleObject, WeaponLoaded, AmIShooting, WeaponType, AmIUnderAttack, AmmunitionAmount, HealthAmount.

The meaning of the symbols should be clear from their terms and we will not describe them in detail. To avoid the well-known problem of bloat in GP a maximum depth (i.e., 15 levels) was imposed for our decision trees.

3 Experimental Analysis

In this section we show the results obtained experimentally by considering separately the two proposals described previously and execute a comparative study in the context of the “2K bot prize” competition. The construction of the game AI was done for the videogame UT2004 and, although this provides its own programming language UnrealScript, our programs were coded in Java.

3.1 General Issues

In the experiments we have considered as opponent one of the bots predefined in Pogamut (i.e., a plug-in for Netbeans that is a free toolkit for development of bots in Unreal Tournament 2004); more specifically we have considered the hunter one. This election was based on our experience as players and with the aim of reaching a more exciting gaming experience. The hunter bot was tested against each of our proposals separately in a game. This experiment (i.e., this game) was executed 5 times in 5 different scenarios (i.e., maps). Each game was executed during 1 minute and a half (note that each game was executed in real time and this was very time-consuming as unfortunately we couldn’t ascertain how to execute the game off-line).

Two different variants of the fitness function where considered:

\[ f_1(x) = (id + 50 \times \text{frags}) - \frac{rd+50\times d}{2} \]

\[ f_2(x) = \frac{id+50\times \text{frags}}{2} - (rd + 50 \times d) \]

Here \( x \) represents a candidate solution, ‘id’ (resp. ‘rd’) is the damage inflicted (resp. received) by our bot to (resp. from) the opponent bot (in terms of health units), ‘frags’ is the number of enemies annihilated by our bot, and ‘d’ is the number of times that our bot was destroyed by the built-in bot. These definitions for the fitness were obtained according to our intuition and based on our gaming experience as players of FPS games.

3.2 The Hand Made Strategy

Our first proposal based on the direct encoding of decision trees is referred here as ‘Hand Made bot’ (HM). 5 different maps (i.e., TrainingDay, Sulphur, etc and
decision tree-based algorithms for implementing bot AI in UT2004

Table 1. Summary with different fitness strategies

<table>
<thead>
<tr>
<th>Fitness</th>
<th>Training Dodging Sulphur Metallurgy Morpheus Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Day</td>
</tr>
<tr>
<td>+aggressive</td>
<td></td>
</tr>
<tr>
<td>Hunter/Best</td>
<td>0.8205</td>
</tr>
<tr>
<td>Hunter/Worst</td>
<td>0.4590</td>
</tr>
<tr>
<td>HandMadeBot/Best</td>
<td>0.5535</td>
</tr>
<tr>
<td>HandMadeBot/Worst</td>
<td>0.0000</td>
</tr>
<tr>
<td>−aggressive</td>
<td></td>
</tr>
<tr>
<td>Hunter/Best</td>
<td>0.9150</td>
</tr>
<tr>
<td>Hunter/Worst</td>
<td>0.3780</td>
</tr>
<tr>
<td>HandMadeBot/Best</td>
<td>1.0000</td>
</tr>
<tr>
<td>HandMadeBot/Worst</td>
<td>0.4395</td>
</tr>
</tbody>
</table>

5 different executions on each of them were done) and Table 1 shows the best and average values obtained by our bot as well as the opponent one; the two different fitness functions mentioned above were also considered. Observe that the HM behaves acceptably well and beats the hunter bot in most of the games.

We have also evaluated our hand-made proposal according to the bases of the “2k bot prize”, that is to say, via a subjective evaluation involving a number of human judges. To do so, we have asked five different persons (i.e., undergraduate students that wanted to participate in the experience) to evaluate which of the bots behaved in a more human-like way (even though, as mentioned in the introduction section, it is not easy to define what this means). In this experiment we considered our HM bot facing other two different bots (i.e., the built-in ‘hunter’ and ‘simple’ bots). The judges were informed about the rules of the “2k bot prize” competition and were said that at least one human was playing (i.e., they were misled). The mission of each judge was to select the bot with a ‘more human-like behavior’. Results are shown in Table 2 in which ‘B’ denotes ‘hunter’ or ‘simple bot’ indistinctly. We can observe that HM is chosen by a majority of the judges. The main reason for it might be that our HM bot exhibits a less mechanical behavior than those of the predefined bots. This is not surprising as HM is based on four complex decision trees (one for each of the states mentioned previously) that consider more combinations of perceived values than the predefined bots.

3.3 The GP Approach

We have also executed our GP based algorithm in the map TrainingDay. 16 runs were done with the following parameters: number of generations: 20, Offspring size: 10, crossover probability: 0.8, mutation probability: 0.1, and fitness function: the less aggressive one. Initial population was randomly generated (a

Note that these experiments are not really like in the “2k bot prize” competition since no human players are on the game.
Table 2. Subjective evaluation according to the “2K bot prize” rules; ‘B’ denotes predefined ‘hunter’ or ‘simple’ bot indistinctly

<table>
<thead>
<tr>
<th>Judge 1</th>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
<th>Round 4</th>
<th>Round 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>HM</td>
<td>HM</td>
<td>B</td>
<td>HM</td>
<td>HM</td>
<td></td>
</tr>
<tr>
<td>Judge 2</td>
<td>HM</td>
<td>B</td>
<td>B</td>
<td>HM</td>
<td>HM</td>
</tr>
<tr>
<td>Judge 3</td>
<td>HM</td>
<td>HM</td>
<td>HM</td>
<td>HM</td>
<td>HM</td>
</tr>
<tr>
<td>Judge 4</td>
<td>B</td>
<td>B</td>
<td>HM</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Judge 5</td>
<td>HM</td>
<td>HM</td>
<td>B</td>
<td>B</td>
<td>HM</td>
</tr>
<tr>
<td>Total HM</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Total B</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

test to avoid non-factible candidates was also used). The reason because we only considers one scenario is because this algorithm is time-consuming as the candidate evaluation has to be done in real time; note that in each generation 10 new individuals are produced and this means to dedicate 15 minutes per evaluation in each generation (i.e., globally this algorithm takes 5 hours). Precisely for this reason no tuning of the parameters were done.

Table 3. Improvement percentage: 16 runs; GP approach

<table>
<thead>
<tr>
<th>Run</th>
<th>Worst</th>
<th>Best</th>
<th>% improvement</th>
<th>Run</th>
<th>Worst</th>
<th>Best</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0500</td>
<td>0.2335</td>
<td>18.35%</td>
<td>9</td>
<td>0.1242</td>
<td>0.5568</td>
<td>43.26%</td>
</tr>
<tr>
<td>2</td>
<td>0.2755</td>
<td>0.3155</td>
<td>4.00%</td>
<td>10</td>
<td>0.2885</td>
<td>0.2885</td>
<td>0.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.6375</td>
<td>0.6375</td>
<td>0.00%</td>
<td>11</td>
<td>0.1650</td>
<td>0.1650</td>
<td>0.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.0950</td>
<td>0.1570</td>
<td>6.20%</td>
<td>12</td>
<td>0.0820</td>
<td>0.1475</td>
<td>6.55%</td>
</tr>
<tr>
<td>5</td>
<td>0.2671</td>
<td>0.7786</td>
<td>51.15%</td>
<td>13</td>
<td>0.0350</td>
<td>0.3805</td>
<td>34.55%</td>
</tr>
<tr>
<td>6</td>
<td>0.2065</td>
<td>0.2065</td>
<td>0.00%</td>
<td>14</td>
<td>0.3970</td>
<td>0.4485</td>
<td>5.15%</td>
</tr>
<tr>
<td>7</td>
<td>0.1400</td>
<td>0.5926</td>
<td>45.26%</td>
<td>15</td>
<td>0.0000</td>
<td>0.6790</td>
<td>67.90%</td>
</tr>
<tr>
<td>8</td>
<td>0.2345</td>
<td>0.6323</td>
<td>39.78%</td>
<td>16</td>
<td>0.0070</td>
<td>0.4172</td>
<td>41.02%</td>
</tr>
</tbody>
</table>

Table 3 displays the improvement percentage that this algorithm provided in each of the 16th runs. The improvement percentage is measured as the difference in fitness values between the best initial random candidate generated in the initial population and the best final solution. Several considerations can be done here: (1) the improvement seems to be not appreciable if the initial solution is acceptably good (as for instance in runs 3, or 6); (2) the best overall solution obtained has a fitness value of 0.7786 (see run 5) and was evolved from an initial value of 0.2671 representing an improvement of about 52% (Figure 2 illustrates this evolution). Note however that this solution is worst (according to its fitness value) than the best solution obtained by the HM bot (i.e., 1.000); initially this might indicate a clear superiority of HM with respect to the GP algorithm. However this does not necessarily mean that HM behaves in a more human-like way than the evolved bot. The next section tries to shed some light on this issue.
3.4 A Subjective Comparison between the Two Proposals

We have conducted a comparison between the HM bot and the best solution (denoted in the following as EV = ‘evolved bot’) obtained with our (non-optimized) GP algorithm according to the rules imposed in the “2k bot prize” competition. To do so, we have again requested 5 judges (i.e., undergraduate students) to evaluate which of these two bots (i.e., the HM and the EV) behaves in a more human-like way and the results are illustrated in Table 4. Surprisingly, EV was the best evaluated in this subjective process. One reason to justify this fact might be that a bot AI that a priori is not too good surely behaves erroneously in certain situations and this represents a more human-like behavior with respect to an optimal strategy. In any case, this result should be taken carefully again as this evaluation is totally subjective and no empirical data were analyzed. This still remains as an issue of further work.

4 Related Work

Evolutionary algorithms (EAs) (we use this term in a broad sense to refer to any kind of evolutionary procedure, including genetic algorithms and genetic
programming) offer interesting opportunities for creating intelligence in strategy or role-playing games and, on the Internet, it is possible to find a number of articles related to the use of evolutionary techniques in videogames. For instance, [8] shows how EAs can be used in games for solving pathfinding problems; also [3] focused on bot navigation (i.e., exploration and obstacle avoidance) and proposed the use of evolutionary techniques to evolve control sequences for game agents. However, till recently most of the work published on the use of EAs in games was aimed at showing the important role EAs play in Game Theory and, particularly, their use in solving decision-taking (mainly board) games [9][10][11][12][13][14].

Evolutionary techniques involve considerable computational cost and thus are rarely used in on-line games. One exception however, published in [15], describes how to implement a genetic algorithm used on-line in an action game (i.e. a first/third person shooter). In fact, the most successful proposals for using EAs in games correspond to off-line applications, that is to say, the EA works on the user’s computer (e.g., to improve the operational rules that guide the opponent’s actions) while the game is not being played and the results (e.g., improvements) can be used later online (i.e., during the playing of the game). Through offline evolutionary learning, the quality of opponent intelligence in commercial games can be improved, and this has been proven to be more effective than opponent-based scripts [16].

Related with the work described here, [17] applies GP techniques to evolve bot AI in Unreal™. Although there are similarities with our second proposal described here, there are also evident differences: the main distinction is that bot AI is coded in [17] as finite state machines (FSM) that do not match our decision trees (even though these FSM were internally coded as trees); also only 2 states were considered and the rules governing the bot AI just admitted two inputs (translated to our work described here this would represent decision trees with only two levels of depths). Moreover, a 8-players game was used for the experimentation whereas we have considered a version of two/three players; in addition no subjective evaluation was considered and the fitness function was different to ours. In any case, the results obtained in [17] support our conclusions that GP is a promising approach to evolve bot AI in FPS games. The same authors also explored the employment of genetic algorithms to controlling bots in FPS games [18].

5 Conclusions and Further Work

This paper has dealt with the problem of providing artificial intelligence to non-player characters (i.e., the bots) in first person shooter games, and more specifically in the context of the game Unreal Tournament 2004. Two different proposals based on Decision Trees to code bot AI in UT2004 have been described. The first proposal represents the approach that currently is followed in the development of existing commercial games and consists of manually coding the bot AI. This way, our manufactured bot AI has been pre-programmed as a decision tree with multiple rules. The second approach is based on genetic
programming, and consists of evolving automatically with no human intervention) a set of candidate solutions represented as decision trees.

We have also conducted an experimental analysis and have compared our two proposals, firstly from an objective point of view according to two specific fitness functions (defined from our gaming experience), and second from a subjective point of view according to the basis of the “2k bot prize” competition. Our two proposals have both advantages and drawbacks. While it is evident that game industry is demanding automated processes to automatically generate the so-called game AI, it is also clear that this is a difficult task as it is to be very dependant on the evaluation phase (and more specifically on the definition of the fitness function). The difficulty of establishing a good fitness function in FPS games comes from considering the user satisfaction as the value to optimize and this represents a ‘hard’ obstacle as this component is not easy to quantify mathematically; there exist however some interesting papers that open interesting research lines - see for instance [19,20].

In this paper our hand-coded strategy provides better fitness values than the best evolved strategy generated automatically via a standard genetic programming (GP) algorithm. In addition the GP algorithm is time-consuming as the evaluation function requires simulation in real time. However, there are other issues that favor the automated process. For instance, the hand-coding is also a costly process that requires many hours of coding; in addition this process of trial, error and debugging is also very expensive measured in human resources. Surprisingly the evolved strategy was better evaluated than the hand-coded one when we considered a subjective evaluation close to that proposed in the “2k bot prize” competition. A more detailed analysis on this issue should be conducted in the future although we have already pointed out some reason for it.

As further research we plan to optimize our GP algorithm by a simple tuning of its parameters. Also, alternative (and more complex) definitions for the fitness functions, that take into account user’s satisfaction, surely would improve the evolution of strategies. In this sense, human-guided interactive evaluation might be useful and thus interactive optimization will be considered in the future. It would be also interesting to see how the two approaches work in a larger scale experiment.

Acknowledgements. This work is supported by project NEMESIS (TIN-2008-05941) of the Spanish Ministerio de Ciencia e Innovación, and project TIC-6083 of Junta de Andalucía.

References


