Programming by Demonstration in a Complex 3D Game[⋆]

Ismael Sagredo-Olivenza, Marco Antonio Gómez-Martín, Pedro Pablo Gómez-Martín and Pedro A. González-Calero

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
email: {isagredo, marcoa, pedrop, pedro}@fdi.ucm.es

Abstract. We have extended behavior trees to support a form of case-based reasoning by retrieving the best action to be performed according to the current state of the game. The case based was filled by programming by demonstration techniques.
In this paper we demonstrate how this extension can be used to program the behavior of a non-player character in a complex 3D video game. We have also measured the perception of control that game designers have on the authoring process and conclude that designers are confident with the final result and they do not have the feeling of having lost control.

1 Introduction

Game AI development is hard. Having characters behave as intended is a challenge for both programmers and designers. On the one hand, game programmers deal with low-level algorithms, perception systems or behavior representation. On the other hand, designers envision the way their characters behave, and are used to express themselves using high level descriptions such as "high general alertness (very nervous, thoroughly investigates every little disturbance, hard to circumvent, hard to shake off once he sees the target)"[1]. Moreover, designers have also in mind the player experience and whether the final behaviors are enjoyable or not.

In order to successfully join both sides of the same problem, programmers and designers must work together. As designers do not usually have programming skills, building the final AIs is an iterative process where designers try to specify the behaviors needed, programmers develop them and designers test them. The process repeat itself once and again until designers are satisfied (or budget and/or time vanish).

In order to alleviate the problem, programmers have been developing tools for designers for years. Their goal is to ease the task of behavior development in such a way that people without technical knowledge (as usually the designers are) can build them, ensuring, ideally, the stability of the final system.

[⋆] Supported by the Spanish Ministry of Science and Education (TIN2014-55006-R)

[1] This is the description of one of the non-player character (NPC) of FarCry.

Rule based systems, finite state machines (FSMs) and scripting languages are examples of alternatives that game studios have been using. Over the last decade, however, behavior trees (BTs) have gained momentum. BTs take ideas from FSMs and visual programming. With specialized tools, anyone is able to create a behavior building a BT without writing a line of code. Though they were initially seen as a tool for designers, the experience demonstrated that building AIs with BTs is not so easy.

On the other hand, an emergent technique that is also actively being studied is *programming by demonstration* [1]: instead of having designers to code behaviors with FSMs, BTs, rules systems or any other method, let them just play the game as they would like their NPCs to behave and have an expert system to infer the underlying rules. The main drawback of this method is that usually designers do not want to trust the final behavior of the characters to a system that has not been fully specified or, in other words, Their do not want loss of control.

This paper put both techniques together and demonstrate that they can coexist keeping the designers confidence high. Using an extension of BTs called Trained Query Nodes (inspired on [2]) and it was described previously in [3] when we allow designers to play the game and specify the actions an NPC should perform at every moment. In the training phase, data is collected to train the query node that is later used in the BT of the NPC. When playing, the BT execution will reach the query node and it will select the appropriate action depending on the state of the game and the training data.

In order to check if the query node replicates the behavior, we have used the system and measure programmatically the difference between the behavior of the NPC during the training phase (when controlled by the designer) and the test phase (when controlled by the query node). More important, we have also measured if the designer is confident with the final result and does not fear that loss of control anymore. As we will see, the results demonstrate that it is possible to keep the designers satisfied and with their feeling of control intact while using programming by demonstration.

The rest of the paper runs as follows. Next section explain programming by demonstration as an approach to do NPC AI. Section 3 describes the game we have used for the experiments and how we have tested the approach, while section 4 presents our experimental results. And finally presents related work and concludes the paper.

2 NPC programming by Demonstration

As explained in section 1, designers envision and decide the behaviors that the NPCs must follow, and programmers make those ideas real. This requires an iterative process where implemented behaviors are refined until designers are satisfied.

But this is not always easy. Designers and programmers usually have opposing interests and concerns, that affect even the way they communicate between them. A way to reduce the communication error and development time consists on easing the way behaviors are specified. Ideally, designers would be able to define those behaviors using *editors* reducing the need for programmers intervention. But behaviors editors such as BT require certain technical knowledge and experience [4] that not all designers posses.

Programming by demonstration (PbD) can be used to avoid designers for specifying the complete BT. But, how do we do it? In a BT, the leaf nodes are charged to execute the primitive actions and other behaviors previously made. As described in [3], those leaf nodes can be replaced with a special node named Trained Query Node (TQN). This node is able to operate in two modes: training mode and execution mode. In training mode, the node saves the traces generated using the human guidelines, trying to preserve the logic environment that produced them and the action executed. In the execution mode, the node search the stored trace more similar to the current environment state, and retrieves which action was executed in that situation.

Unfortunately, designers are not comfortable with non-deterministic behaviors because they have a feeling of control lost for these behaviors. And PbD usually can generate non-predictable behaviors. Our TQN aims to solve this situation, letting designers choose specific areas of the complete behavior where they are confident enough to use PbD.

But, which it is the methodology to use this tool to make games? The designer must design the behavior of the game NPCs using a high-end specifications, for example, natural languages, tables, rules, diagrams, etc. Finished this step, designers and programmers must identify and select the basic actions of the NPC, because these actions must be implemented by programmers. On the other hand, designers must identify the actions that must be running in a determinist way. These actions o sub behaviors must be implemented in the traditional way, using the BT language, to make sure that they run are controlled. And finally, designers must identify those actions that can be changed in the future, because they are not clear at this moment in the design process or because these depend on which strategy will take the NPC. These actions can be implemented by programmers like a part of the BT, but designers can also replace this set of actions with a Query Node and train it. In this way, the designer is in charge of generating this part of behavior and can modify itself.

In the training phase, the designer must select the most representative and relevant subset of BT attributes that will compose the training environment. This subset may change in the future if the training no produce a good result. Furthermore, each attribute is weighed with a parameter that describes its relevance in the decision making. This weight may be modified in the future in order to adjust the behavior. In addition, in order to avoid disturbances by different ranges to calculate the similarity, the parameters are normalized taking into account its range of values.

Once configured the node with the corrects parameters and its weights the training phase begins as an iterative process that consists of the following sub-phases: On one hand, in the demonstration sub-phase, the designer play a cropped version of the game created specifically for training. In this sub-phase, the designer interrupts the NPC to change the action was running, taking into account the game situation, the value of the selected attribute (which we recommend to be visible to help the designer) or anything else that the designer take in consideration. On the other hand, in the validation phase, the designer must play the game like an observer and decide if the training have been correct or not. If it is correct, then the process will finish, but if is not enough satisfied, the designer can repeat the process, adding new situations and examples, modifying the attribute weights or the CBR parameters, etc.

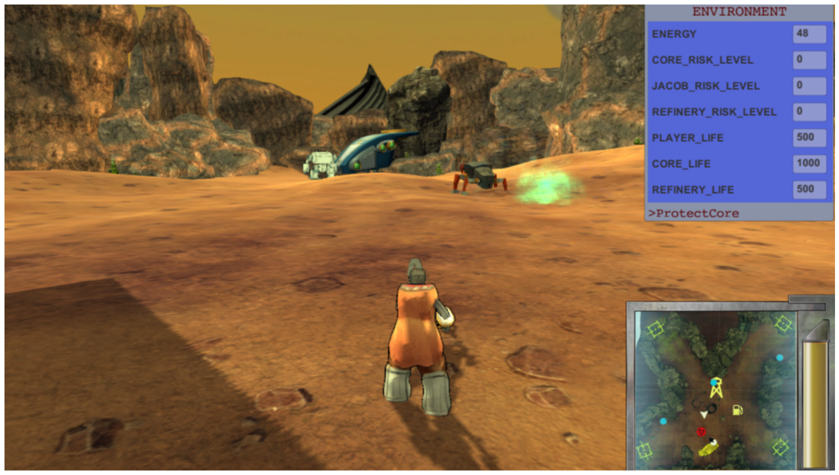
Once we have created some behaviors with different strategies, designers can combine them to create new behaviors. For example, we can train a soldier with a defensive and offensive behavior and later to use both to build a mixed strategy. Another possibility is to use these different behaviors with different strategies in other BT with other Query Node and training to select when must be using one or another. In this way, we can reuse the behaviors in order to create other behaviors easier or change the behavior dynamically.

In the majority of cases, the main problem of this technique is designers are not friends of the indeterminism and this technique can produce it, but in certain situations it may be interesting somethings emergence behavior that can surprise the player, with which we can take advantage of this, but in general, if the training fail to gain a acceptable and reproducible behavior, the designer or programmer must create an equivalent BT using the editor. With this technique the designer has control what actions can be trained by demonstration and that others must be edited with a BT editor.

3 Experimental Setup

TowoT is a tower defense game prototype developed at Complutense University of Madrid. It combines tower defense with third-person action elements. As a tower defense, every level of the game consists of two stages: tower deployment, first, and defense, afterwards. First, the player has to place a number of towers that will serve to protect certain places in the level once the enemies deploy. As a third-person action game, the player can fight the enemies in the fight phase, so instead of keeping on placing towers as new enemies arrive (usual in pure tower defense games such as "Plants vs Zombies"®) the player concentrates on actually fighting the enemies ("Orcs must Die"® is the best known game combining those elements). In addition to the static towers with diverse defensive and offensive abilities, the player is helped by TowoT. TowoT is a big robot that moves slowly but fires heavily.

The goal of the game is to collect Uridium, a mineral which is crucial for the *Zaphod Federation* to accomplish their dark plans. Jacob, our hero, moves from mining world to mining world harvesting Uridium with the help of his loyal robotic friend, TowoT. They arrive in their small spaceship and have to move quickly collecting as much Uridium as possible, before the insect droids, defending every mining world, destroy them or their spaceship. Figure 1 shows a screenshot from the game, with Jacob, at the foreground looking towards TowoT defending the spaceship from an approaching arachnid droid.


ENVIRONMENT

ENERGY	48
CORE_RISK_LEVEL	0
JACOB_RISK_LEVEL	0
REFINERY_RISK_LEVEL	0
PLAYER_LIFE	500
CORE_LIFE	1000
REFINERY_LIFE	500

>ProtectCore

Fig. 1: TowoT

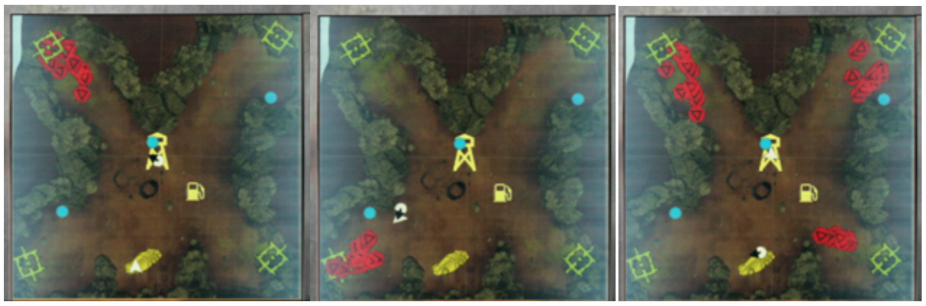


Fig. 2: Enemy waves

The goal of the experiment was to have a number of game designers (game design students, actually) programming TowoT by demonstration. We asked designers to train TowoT and then test whether it behaved as trained.

After a short presentation of the game and the structure of the experiment we had the designers play a simplified version of the game where a mining tower and a charging station have been already put in place. In the actual game, the player has to place mining towers close to the mineral, and choose a place to put a tower for the TowoT to recharge. In the mini-map of Figure 2 the charging station is the petrol pump icon, while the mining tower is represented as an oil extraction tower in the center of the mini-map. A yellow spot in the middle of the bottom shows the position of the spaceship.

In the experiment, the designers played the same simplified level several times. A level in our game, as usual in tower defense games, defines a number of enemy waves with increasing number of enemies as the level progresses. Figure 2 shows the three waves included in this particular level. First wave comes from the generator in the upper left corner of the map, second wave from the generator in the bottom left, and third wave simultaneously spawns enemies form upper left and right, and bottom right. Enemies, with a certain random component, will attack, with decreasing probability: Jacob's spaceship, the mining tower, or the charging station.

Game designers trained TowoT two times in the experiment. For every training they first played in *training mode*, and then evaluated the results of the training in *test mode*. In training mode, in addition to control the player, they had to give orders to TowoT, while in test mode TowoT behaved according to the last training and they only had to control Jacob. Again, in order to simplify the conditions of the experiment, TowoT has only four possible behaviors: protect the spaceship; protect the mining tower; protect Jacob; and recharge. When protecting the spaceship or the mining tower, TowoT goes to the vicinity of the spaceship or the tower and stays there. When protecting Jacob, TowoT follows the player's avatar, and when recharging it moves to the charging station for energy refill. In any of those states, TowoT will fire any enemy entering its firing range.

Since we wanted to obtain comparable results, we asked designers to train their TowoT in the same way, following the same strategy. In the first training episode, they should play using a defensive strategy: place the TowoT to defend the spaceship, make the player defend the mining tower, and take care to make TowoT recharge between enemy waves. For the second training episode they should play using a more aggressive strategy: ask TowoT to protect Jacob, and use Jacob to wait for the insect droids close to their generators, since combined fire from Jacob and TowoT will destroy most of them as they emerge from the generator; for the final wave, they have to split forces, having TowoT defend the spaceship and Jacob the mining tower.

The rationale for the two play strategies used in the experiment is to first train TowoT with a simple strategy, since it remains most of the time defending the spaceship and only needs to move for recharging between waves. The second strategy is more complex for the TowoT and there are more chances that the result is not exactly what the designer is intending. It has to be noticed that in the experiment we asked each designer to play just one training session for every strategy while in actual use of this technology several training sessions would be required, and some knowledge about the inner workings of query nodes in behavior trees would be needed. The whole experiment was run in just one hour by designers without prior exposure to the game or the program by demonstration technique. The details of the metrics used and results are described in the next section.

4 Experimental Results

The experiment was realized by thirty eight game design students that trained the two strategies described in the previous section. Each training phase lasts about 4-5 minutes (which coincides with the time of a game session in the arena shown in Figure 1) where the traces were saved in order to be used in the test phase. This second phase can be seen as a pure game session where the user acts as a player and let the TowoT move autonomously using the trained query node.

Towot is ruled by a BT containing just a repeat node at its root with two tasks running in parallel, one in charge of the perception that continuously set parameters in the shared blackboard and other with the TQN (figure 3).

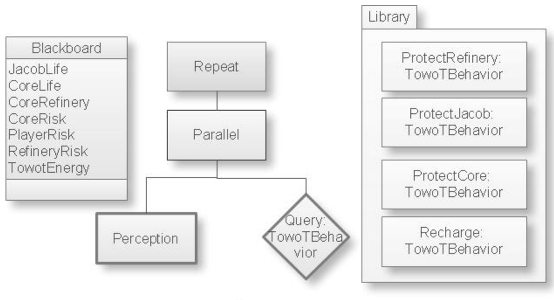


Fig. 3: BT trained in the experiment with one trained query node

As explained in section 2, TQN monitorizes the status of the game and selects the next action to be done by TowoT based on the action took by the designer on similar scenarios in the training phase.

The description of each scenario consists of the values of the set of relevant game parameters and the action TowoT was performing of that particular moment. During the training phase, designers know the values of all these attributes, as they are shown in the HUD (upper-right of figure 1). The actual set of attributes stored during the training phase:

- TowotEnergy: The TowoT energy has (a value in the range [0,100]). The weight used in the evaluation of the distance between cases is 2.
- RefineryRiskLevel: Number of enemies round the Refinery. Its Weight is 1 and its range is [0,4] (it saturates when the fourth enemy is reached).
- CoreRiskLevel: Number of enemies round the Core or base. As the previous one, its weight is 1 values are between [0,4].
- JacobRiskLEvel: Number of enemies round the Refinery. Weight and range are equal to the other risk levels.
- CoreLife: The Core remaining. The weight of this parameter is significantly lower than the previous one (only 0.25) and its range are between [0,1000].
- RefineryLife: The Refinery life level. Its weight is 0.25 and its range is 0-500

– JacobLife: The player's life. Its weight is and its range is the same as the previous.

On the other hand, the atomic actions that TQN (or designer in the training phase) has available are expressed in form of subbehaviors implemented by AI programmers using also BTs.

The training phase ends with about 4,000-6,000 cases in the case based that save the actions that designer specified when the designer believes that are suitable. TQN will use these examples to select the action to will be performed later. The TQN implementation is agnostic regarding the method used, but in the experiment we retrieve the $k = 10$ most similar scenarios and, using KNN to select the action to be executed. As the game state does not dramatically change between consecutive ticks, TowoT behavior is stable and does not suffer oscillation problems.

In order to determinate if the TowoT had learned, we needed that the users play the games the same way in training and test steps (with the same strategy) because we want to compare both and compute its similarity. Obviously, The behaviors will not match exactly among executions, because the game has some random behavior. For example, the enemy can attack the base or the refinery with some probability or the player actions are not exactly the same at the same time.

In order to measure if TowoT have learned the behavior trained, we have used a double check. On the one hand, we use the behavior similarity between the training and test phase. To calculate this measure, we build a vector with the training and test actions and compare both of them using the Levenshtein distance. On the other hand, the users rated the behavior in the test phase with a value from 1 to 5, therefore we can compare both measures.

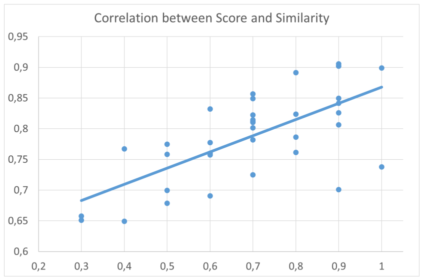
As you can see in the Figure 4 exist a correlation between the similarity (Levenshtein distance between the actions executed in training and testing phase) and the users scoring in the experiment. Therefore, the similarity between the actions executed in both phases is a good measure to known if the Towot has been learned successfully.

Our initial hypothesis was if the training is correct and the examples supplied are enough, the NPC will be able to reproduce in the test step the behavior learned in the training phase. The results shown in the Figure 5 prove that the NPC can repeat a similar behavior training in first and second strategy, in the majority of cases.

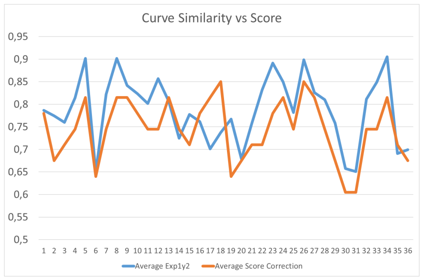
In the Table 1 it can see the average of results obtained with the two strategies trained, using as a measure the similarity and the user score.

	Strategy 1	Strategy 2
Similarity	0.86	0.71
Score (1-5)	3.77	3.19

Table 1: Average results using similarity and users score

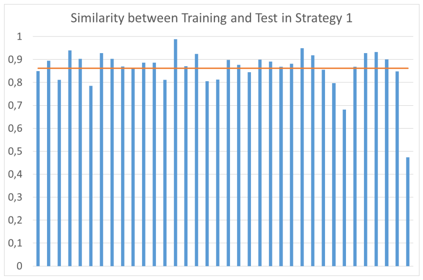


(a) Correlation between the score and similarity

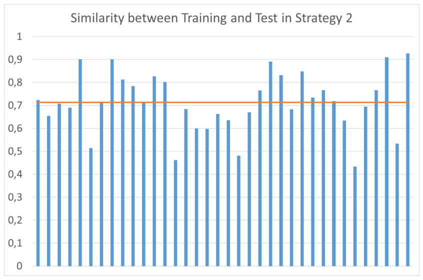


(b) Curve Similarity vs the Score

Fig. 4: Comparing similarity and the Score



(a) Strategy 1



(b) Strategy 2

Fig. 5: Similarity between training and test actions executed in each strategy

Figure 6 shows a condensed view of the similarities for each strategy using a box diagram. As stated before, the first strategy has by far the best results with a similarity greater than 0.75. The dispersion of the second strategy is greater, because the behavior is more complex but the users had learned to train them better.

At the end of the experiment, the users filled out a survey where we asked about the global satisfaction using this methodology. The most relevant questions and their answer were:

- *Would you have found easier to implement this behavior using a Behavior Tree editor?*
  - 57.1% No
  - 42,9 Yes.
- *Do you think you could train a behavior in a game using this tool?*
  - 76.2% Yes
    - 33.3% they would use it, but prefer using a behavior editor
    - 28% prefer using this tool before behavior editor
    - 83% of non programmers prefer using this tool before behavior editor.
  - 23,8 No

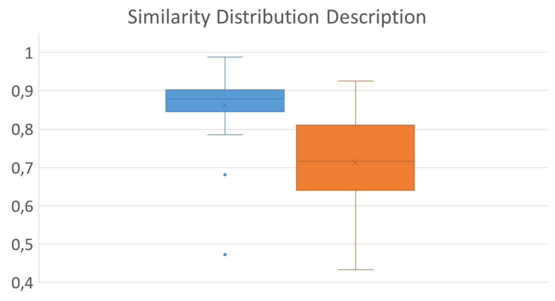


Fig. 6: Similarity between training and test

All users that answered the survey, **57%** had good programming skills and **43%** medium or low programming skills.

With all this in mind, we can say that the main fear of designers, to lose control of the behavior, does not occur with a proper training. As the results show, the average results training both strategies are high. Therefore, we can say that this tool allows training a behavior using demonstrations, simplifying the task edition to designers that do not have good programming skills. And we can measure if this training is correct using an automatic technique comparing the training and test behavior. This allows to create a high level behavior with a few training sessions and get a similar to the expected behavior.

5 Related Work and Conclusions

Significant work done under the label of learning from demonstration has emerged recently in the Case-Based Reasoning community. Floyd et al. [5] present an approach to learn how to play RoboSoccer by observing the play of other teams. The different approach is clear in this example. The Robosocer is an interesting environment to test in an experimental way, but the complexity of this game is not comparable with commercial games, fun is not a priority and no designers impose their requirements for the behaviors. Our system allows to designers more control about the behavior, since the part of behavior that the designer wishes to ensure can be modeled using a BT. Ontañón et al. [1] use learning from demonstration for real-time strategy games in the context of Case-Based Planning. In this approach, the complexity of the game using to test (Wargus is a real time strategy game created for research purposes) produces an acceptable behavior in high level task, but very poor results in low level units control. Ulit Jaidee et al. [6] uses a case base driven by goals. In this work the authors use Case-Based Reasoning to move and coordinate multiple agents in Wargus with similar results.

Other works have also combined BT with other learning techniques like Q-learning or genetic algorithms. For example G Robertson et al. [7] use computational biology and observations of expert humans to produce a behavior tree for strategic-level actions in the real-time strategy game StarCraft and Rahul Dey et al. [8] use Q-learning technique to Enhancing Behavior Tree for designers in order to assist them and identifying the most appropriate moment to execute each branch of a BT. The condition nodes were changed in this work for a special node named Q-Condition Nodes. This node stores a Q-Table that it builds itself automatically in a previous offline Q-learning phase. This approach is another way to enhanced a BTs and help designers to build behaviors. These techniques are very interesting, but require a previous offline training and for designers is more intuitive training with certain interactivity. For that reason we exclude those techniques in our system.

And finally [9] have already explored the interrelation between the Online Case Base Planning (OLCBP) that was described in Ontañión et al [10] and the Behavior Trees. In this work, the authors used BTs to create the behavior of a low level units control in a RTS game, which had not retrieved good results in previous works using only OLCBP. The idea is similar to explained in this paper, but in our approach the problem is seen from another point of view. In Ricardo Palma's work, the planning layer selected an action to execute in the game using expert knowledge. When the action is selected, the system tries to retrieve a BT that implements this selected action among a list of BTs allowed in the system. Each BT is described by rules which show which is the best scenario to use them. The system selects the BT with the description more similar to the current context in the game or execute the primitive action in the system if the similarity of the best BT is too low. In contrast, in our approach, we delegate the main responsibility to run the behavior of the NPC in a BT instead of doing in a Case Base planner, and we use CBR to retrieve the BT candidate to run the action described in the TQN. This approach simplifies the designers work because the description of the scenario where the BT can be executed is hard to make and to train the BT while the designer plays the game is a more natural way to do it. The main reason of this different point of view is because our system is intended to be a support tool for the designer, and designers need more control of the behavior.

Our approach takes the advantage over the previous works that we can combine both behaviors by demonstration and behaviors by programming using BT. As trained query node is a part of the tree, there may be other nodes created explicitly. The idea behind our technique is to help designers to make behaviors with more autonomy, but without replacing the programmer role. In addition, the training process using this technique is clear and interactive which doing the training more bearable for the designer. With the experimental result, we can say that designers can train the BT to achieve behaviors easily. A correct combination of traditional BT node and our trained query node can create complex behaviors and reduce the development time. Taking all the above into account, we can say that this technique and methodology allow designers cooperate with programmers in the behavior authoring, even if the designer does not have great technical knowledge. Likewise, designers do not have the feeling to loss control of the behavior because the results show that the behaviors are similar in execution and tanning. Finally, in the most sensitive parts of the behavior can always be modeled with traditional nodes.

As a future work, we want to analyze the information saved in the training phase to generate a BT description automatically and explicitly to help both programmers and designers to understand the BT trained and thus, they will be able to train or create betters BTs in the future. Another interesting improvement of this work would be, in the testing phase, the designer would be able to correct the bad decisions taking by the NPC with the aim to polish and clean the behavior database of possible incorrect examples or adding new examples to complete the database.

References

1. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: On-line case-based planning. Computational Intelligence **26**(1) (2010) 84–119
2. Flórez-Puga, G., Gómez-Martín, M.A., Gómez-Martín, P.P., Díaz-Agudo, B., González-Calero, P.A.: Query enabled behaviour trees. IEEE Transactions on Computational Intelligence And AI In Games **1**(4) (2009) 298–308
3. Sagredo-Olivenza, I., Puga, G.F., Gómez-Martín, M.A., González-Calero, P.A.: Implementación de nodos consulta en árboles de comportamiento. In: Proceedings 2st Congreso de la Sociedad Española para las Ciencias del Videojuego, Barcelona, Spain, June 24, 2015. (2015) 144–155
4. Sagredo-Olivenza, I., Gómez-Martín, M.A., González-Calero, P.A.: Supporting the collaboration between programmers and designers building game AI. In Chorianopoulos, K., Divitini, M., Hauge, J.B., Jaccheri, L., Malaka, R., eds.: Entertainment Computing - ICEC 2015. Volume 9353 of Information Systems and Applications, incl. Internet/Web, and HCI., Springer International Publishing (2015) 496–500
5. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating robocup players. In Wilson, D., Lane, H.C., eds.: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA, AAAI Press (2008) 251–256
6. Jaidee, U., Muñoz-Avila, H., Aha, D.W.: Case-based goal-driven coordination of multiple learning agents. In: Case-Based Reasoning Research and Development. Springer (2013) 164–178
7. Lemaitre, J., Lourdeaux, D., Caroline, C.: Towards a resource-based model of strategy to help designing opponent ai in rts games. In: 7th International Conference on Agents and Artificial Intelligence (ICAART 2015). Volume 1. (2015) 210–215
8. Dey, R., Child, C.: QL-BT: Enhancing behaviour tree design and implementation with q-learning. In: Computational Intelligence in Games (CIG), 2013 IEEE Conference on, IEEE (2013) 1–8
9. Palma, R., González-Calero, P.A., Gómez-Martín, M.A., Gómez-Martín, P.P.: Extending case-based planning with behavior trees. In: Twenty-Fourth International FLAIRS Conference. (2011)
10. Ontanón, S., Mishra, K., Sugandh, N., Ram, A.: On-line case-based planning. Computational Intelligence **26**(1) (2010) 84–119