Supporting the Collaboration between Programmers and Designers Building Game AI

Ismael Sagredo-Olivenza, Marco Gómez-Martín, Pedro González-Calero

Supporting the collaboration between programmers and designers building Game AI *

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

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

Abstract. The design of the behavior of non-player characters (NPCs) in a game is a collaborative task between programmers and designers. Nevertheless this collaboration is an open problem since the limits, responsibilities and competences are not well defined.
Behavior trees are the technology of choice nowadays for programming the behavior of NPCs, and they are first and foremost a programmers tool. In this paper we describe an experiment that shows that with the right division of labor and a reduced background in Programming, designers can also build behavior trees and thus find a principled way to collaborate with programmers in that task.

1 Introduction

Game development is a multidisciplinary task that involves professionals from different areas with different knowledge and sensibilities. The three main roles involved in the development of the artificial intelligence (AI) of non-player characters (NPCs) in a game are: artists (that are beyond the scope of the paper), which make the models and animations; programmers that implement behaviors; and designers, responsible for designing those behaviors.

Designers define how the characters must behave and programmers implement these behaviors, therefore, they should communicate between them and try to reach an agreement. This relation produces an iterative cycle of changes, because programmers can deliver an incomplete or inaccurate version of the behaviors to designers. Accordingly, designers need to validate these behaviors and then require the appropriate changes to programmers. This process ends when the behavior is accepted by designers.

To get a more fluid process, programmers usually develop authoring tools for designers. The goal is to get designers to become as autonomous as possible so they can modify certain parameters in order to configure behaviors to their needs. Different solutions have been explored (state machine, scripting, behavior trees...) but the most behaviors of the NPCs are stored in data files. These data file formats are seldom easy to understand or edit, hence, designers and

* Supported by the Spanish Ministry of Science and Education (TIN2014-55006-R)

developers must have tools to edit or visualize them. If designers have a certain degree of autonomy, then they can test, modify or create parts of behaviors without the intervention of a programmer.

In this paper we present a methodology together with a tool (Behavior Bricks) for designing behavior trees that distinguish high-level behavior from low-level ones. We can think of these high-level behaviors as a simple description of the general behavior of an NPC, similar to the ones used in the early stages of design [1]. Our hypothesis is that, with the proper tool, designers can implement high-level behaviors while programmers implement the low-level ones in parallel.

The rest of this paper is structured as follows. Next Section describes behavior trees, Section 3 describes Behavior Bricks and our methodology for using it, and Section 4 describes the experiment. Finally Section 5 presents some conclusions and future work.

2 Behavior Trees

Behavior trees (BTs) are a modeling technique of behavior of an NPC. They were popularized for their utilization in Halo 2 [2] and Halo 3 [3], and they have similar representation capabilities of the traditional Finite State Machines (FSMs) [4,5]. The main problem of the FSMs and the Hierarchical FSM (HFSM) are the scalability of their transitions. When the problem grows, the number of relationships between states grows much faster than states and they become uncontrollable soon.

BTs improve scalability over FSMs thanks to remove these transitions, replacing them for internal nodes, which select the tasks that will run. By definition, BTs are hierarchical, therefore, it is very easy to have multiple abstraction-layers which improve their re-usability. Behavior trees have become popular thanks to the need to make increasingly complex behavior and the needs of the developers that were not comfortable with the existing techniques. Designers wish full control of the behavior but not all time. They need to delegate the most complex tasks to programmers [6]. In addition, designers want that the behaviors are driven by goals, but also that they are reactive to unexpected events and finally, they want that the chosen model is easy to use, easy to comprehend and, if possible, easy to draw.

3 Behavior Bricks and its methodology of use

As we said before, designers must create behavior independently. For that reason designers need tools to simplify this task. In the market, there are some tools with this purpose like *NodeCanvas*[1], *Behavior Designer*[2] or *Behave*[3] among others.

[1] http://nodecanvas.com/
[2] http://www.opsive.com/assets/BehaviorDesigner/
[3] http://angryant.com/behave/

After analysing these editors, we have concluded that neither of them is adapted to our methodology because none of them has a correct abstraction system that allows to encapsulate the behaviors and the primitive tasks for reusing them. For those reasons, the authors have created a new BT framework named *Behavior Bricks*, a tool designed to be used by designers that are able to create, view and modify behavior trees and that, in addition, can be easily extended by programmers.

Behavior Bricks has two different modules: *Runtime module* and *visual editor*. Both runtime and editor are distributed as a extension of the game engine Unity3D[4].

3.1 Methodology

BTs can run sub-behaviors easily. These sub-behaviors can be used by designers within other behaviors like a simple primitive task. This mechanism may have different abstraction layers in a BT. We can define two different levels of details the *high-level* detail and the *low-level* detail. The low-level behavior can be defined as the behavior that can be reused in other behaviors. This behavior manages the concepts closer to scripting languages such as: target selection, follow a route, find a coverage, etc. The high-level behaviors can be defined as the behavior that describe the general behavior of the NPC. One of the purpose to create Behavior Bricks is to introduce this new methodology to develop the video games AI.

The responsibility of programmers in Behavior Bricks is two-fold. On the one hand, they must create the low-level primitives using the scripting language of the underlying engine and, on the other hand, we must implement with Behavior Bricks editor the low-level behavior.

A primitive task is defined establishing his name (that must be unique in the hierarchy) and optionally a collection of input and output parameters. The input parameters read their values from a blackboard stored in the behavior executor and the output parameters write its values in same blackboard.

The responsibility of designers in Behavior Bricks is make the high level behaviors using the visual editor. The designers using a subset of the internal nodes to simplify the BT complexity. This subset is formed by the following nodes: Sequences, selector, priority selector and the repeat decorator.

4 Experimentation

The experiment that we describe below demonstrates that it is possible and effective that non-technical designers implement high-level behavior with similar result to programmers, if they have been trained enough and they have a visual tool like Behavior Bricks. We have carried out this experiment with the students from the master program on Game Development at Complutense University of

[4] http://unity3d.com/

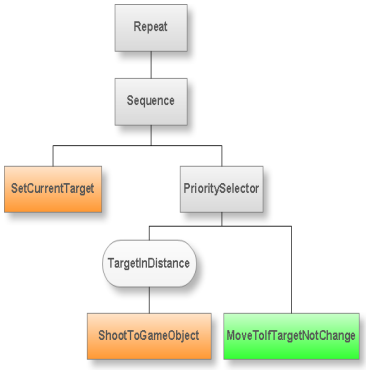


Fig. 1: The expected solution of the basic enemy behavior

Madrid, which includes two different itineraries: one for programmers and one for designers. These students have a diverse background, including: Computer Science, Mathematics, Journalism or Arts, and therefore is complex enough to validate our hypothesis. Before the experiment, the students had a class about BTs where they learned the concepts and how to use our tool.

4.1 The experiment environment

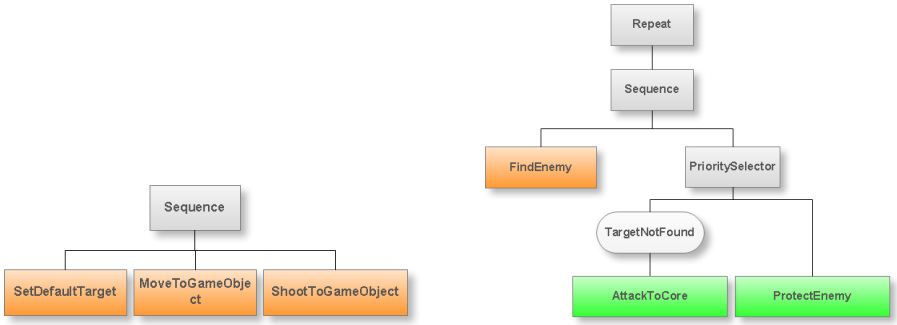
To carry out the experiment, we have used a prototype of a tower-defense video-game. In this prototype, the player's mission is to defend the base (or core) of a space mine. In this version, the player only can move his avatar and shoot against the enemies. Enemies appear in different spawning points and go to the base to destroy it.

Designers have described in a design document two types of enemies in the game: the basic enemy and the shield.

The *basic enemy* receives a target to its perception system, in this case only are available the core and the player. The selection of this target is random with different probabilities for each type of target. Perception is in charge of selecting the target that the NPC must consider. This perception is already done in the exercise and the students should not worry about it. To explain the behavior, we can see the expected solution in the figure 1.

The second exercise is divided in two parts: In the first part, the students must implement the low-level behavior *AttackTheCore*, when the initial target of the enemy is the core. The enemy should move next to the core and shoot it. This low-level behavior is very simple and it does not require the intervention of a programmer. The expected solution of this behavior can be seen in Figure 2a.

In the second part, the students must implement the high level behavior of the *Shield*. This NPC try to protect an enemy. While protecting the enemy, it should maximize the number of enemies protected by its shield. If the target dies, it try to find another enemy to protect. If not found an enemy to protect,



(a) The expected solution of the AttackTheCore behavior

(b) he expected solution of the Shield behavior

Fig. 2: Second exercise formed by two parts

then it must attack the core. To explain the behavior, we can see the expected solution in the figure 2b.

For both, the basic enemy and the shield, we provide the students a set of primitive tasks and behaviors to be used. We can assume that this tasks would be developed by programmer in a professional environment.

4.2 The experiment

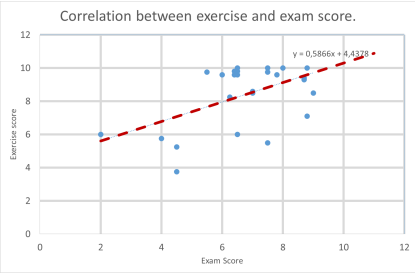
With this experiment, we want to evaluate if a correlation exists between the knowledge of programming and their results using Behavior Bricks. To do this, we have compared the marks obtained in a programming test and the marks obtained in the practical exercises with Behavior Bricks.

In our hypotheses, we expected that the programming knowledge will help to better solve the exercises, although it is not a barrier for designers that can also create high-level behaviors with little training.

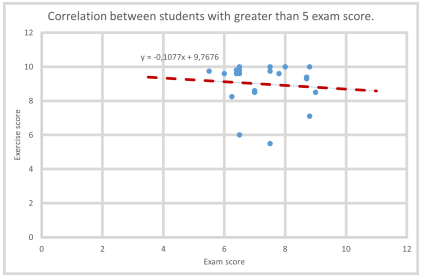
The sample of this experiment is of 25 students. They had 2 hours and 10 minutes to resolve the two exercises. Once completed the experiment, we have evaluated these exercises. The score of each exercise has been as follow: 4 points for the first, 2 points for exercise 2.1 since it is similar to the first one, and 4 points to exercise 2.2.

The plot that we can see in Figure 3a shows an evident correlation between the result of the programming exam and the result of the exercise using Behavior Bricks. In addition, if we calculate the Pearson product-moment correlation coefficient (PCC) we obtain a result of 0.525, which indicates that a correlation exists (its value is greater than 0.5) but the correlation is not very pronounced. Furthermore, the result agrees with the hypothesis that the technical knowledge is important to comprehend correctly BTs, but simplifying the model and applying it in high-level behaviors, the designers can use it too.

If we carefully analyze the plot we find that given a basic programming knowledge, the differences between exercise results and exam do not follow any



(a) Correlation between programming knowledge and exercise result



(b) Correlation with an exam score greater than 5

Fig. 3: Experiment results

kind of correlation. Figure 3b shows the distribution of the students who have passed the exam. In this figure we can observe this fact since there are no obvious correlation between both and the dispersion of the exercise result is very large. Even those students without technical skills have been able to complete at least the first exercise.

5 Conclusions and Future Work

Given the results from our experiments, we can conclude that, using a tool such as Behavior Bricks a designer with little training in the tool and little previous knowledge about Programming, can design high level behaviors of good quality within a reasonable time. In addition, we can conclude that given a minimum programming knowledge we have not found any differences in the results using Behavior Bricks, therefore, is not necessary to have high programming skills to create high-level behaviors using Behavior Bricks with our methodology.

In the future, we want to have more experiments with more subjects in order to validate other aspects of our methodology. We plan to compare our methodology with other in order to assess its comparative competitiveness, both in terms of developing effort, collaboration support and quality.

References

1. Hudson's, K.: The ai of bioshock 2: Methods for iteration and innovation. In: Game Developers Conference. (2010)
2. Isla, D.: Handling complexity in the Halo 2 ai. In: Game Developers Conference. (2005)
3. Isla, D.: Halo 3 - building a better battle. In: Game Developers Conference. (2008)
4. Rabin, S.: 3.4. In: Implementing a State Machine Language. Volume 1 of AI Game Programming Wisdom. Cengage Learning (2002) 314–320
5. Bourg, D.M., Seemann, G.: AI for Game Developers. O'Reilly Media, Inc. (2004)
6. Champandard, A.J.: Behavior trees for next-gen ai. In: Game Developers Conference. (2005)