Supporting the Construction of a GUI Component for Specifying the Behavior of Non-Player Characters in Unity

Ismael Sagredo-Olivenza, Complutense University of Madrid, Madrid, Spain

Gonzalo Flórez-Puga, Complutense University of Madrid, Madrid, Spain

Marco Antonio Gómez-Martín, Complutense University of Madrid, Madrid, Spain

Pedro A. González-Calero, Complutense University of Madrid, Madrid, Spain

ABSTRACT

Unity 3D is a widely used middleware for game development, primarily by small studies. In addition to the core technology, a number of extensions or add-ons have been created by third-party developers to provide additional functionality. Nevertheless, the support Unity 3D provides for building GUI for such add-ons is at a very low level, and for that reason the authors have designed and built UHotDraw, an extensible framework in C# that simplifies the development of GUI and draw applications and editor extensions in Unity 3D.

Keywords: Artificial Intelligence, Framework, Interfaces, Unity3d, Video Games

1. INTRODUCTION

The Computer revolution has brought mankind a plethora of tools that ease tasks that in the pre-digital era were made by hand. Nowadays, those tedious tasks that required several hours of hard work are done by machines driven by pieces of software in such a way that are able to complete the same job in less time and with minimum human supervision.

Computer engineers themselves have profited of this process and we have developed a wide range of tools whose unique goal is to ease the task of building other software. In the list of technologies and tools, we can mention the development of high-level languages, integrated development environments (IDEs), code-generator tools and model-driven architecture (MDA) (Pastor, Molina, 2007), version control systems, automated tests, and so on.

DOI: 10.4018/IJCICG.2015010103

The Game development area has not been immune to this tendency. In a typical game studio we find professionals of different areas such as game designers (they are in charge of defining the game rules and usually lack of technical skills), programmers (they implement the game code and tools) and artists (they create the visual assets of the game), not to mention those roles not related to the development itself such as marketing. Traditionally, programmers have had two different responsibilities: in one hand they have to provide tools to the rest of the team (being the level editor, the main one); in the other hand, they develop the code for the game itself.

Over the last decade, however, a different approach has arisen, allowing programmers to be partly relieved of those responsibilities. They are known as game engines: a piece of software that integrates both development tools and game code.

The entry point to the game engine is the level editor, where designers build the different scenes that conform the game. The level editor itself is usually responsible of creating the game executable for the target platform which usually incorporates levels, assets created by artists, game engine code and that additional code created by the studio developers which includes the specific behaviours that their game needs. The significant cost savings in development time and team size compensate for the cost of licensing the game engine.

According to a survey done to game developers by Gamasutra[1], a website specialized on games, the two more valuable aspects of a game engine are the rapid development time and its flexibility and easy extendability. Regarding this last point, it involves two different aspects: how easy is to extend the game engine creating new behaviours and code to be added to the final game and the ability to extends the tools themselves, incorporating functionality that is used by designers and artists at development time, but that is not seen by final users.

One of the most used game engines is Unity3D[2] . According to the same survey mentioned above, up to 53.1% of independent game studios are using it, mainly for game development over mobile platforms based on both Android and IOS.

Though those numbers support its quality to a large extent, it lacks of an adequate support for authoring the behaviours of non-player characters (NPC). Unity completely relies on programmers that should sculpt these behaviours in the code, instead of using visual editors. To create these behaviors, successive iterations are needed. First, the designers are in charge of defining the high level behaviours of the NPCs, then the programmers write the code to implement them and finally the designers validate the results and propose the necessary modifications that programmers must perform. This cycle will repeat over and over to get the behaviour that the designer had in mind.

It is usual that parts of some of the behaviours are reused in the other behaviours assigned to different NPCs in the game. Hence reusing is an important property of behaviours that helps to reduce the development time (Flórez-Puga, Díaz-Agudo and González-Calero, 2013).

For these reasons, if the designers have available visual tools that allow to design and reuse the behaviours, without the intervention of programmers, the number of iterations to complete the game and the development time will probably decrease. Nevertheless, as we will see in the Section 4, there are no tools in Unity to successfully address these needs. This drawback must therefore be compensated with an external Unity add-ons.

With all these in mind, we pursued the integration of our previous work on building authoring tools in Unity (Flórez-Puga et al., 2005; Flórez-Puga, González-Calero, Jiménez-Díaz and Díaz-Agudo, 2013; Flórez-Puga, Jiménez-Díaz and González-Calero, 2012). Our main motivation was the creation of a visual editor of non-player character (NPC) behaviours using mainly Behaviour Trees (BT) (Rabin, 2006; Rabin, 2008). By creating a BT editor as part of the Unity interface we have the intention of providing a complete set of tools that are tightly integrated into Unity 3D. With a visual editor like the one we envision, the game

Related Content

Soniferous Architecture: From Archaeo-Acoustics Towards the Soundsculpture Aural Era

Mostafa Refat Ismail (2014). *International Journal of Art, Culture and Design Technologies (pp. 42-62).*
www.igi-global.com/article/soniferous-architecture/116023?camid=4v1a