TRABAJO DE FIN DE MASTER EN SISTEMAS INTELIGENTES

MASTER EN INVESTIGACIÓN EN INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Creación de Juegos Serios en Unity 3D

Autor:

MATE ANTIČEVIĆ

Director:

Iván Martínez Ortiz

Curso Académico: 2013-14

Calificación Obtenida: Sobresaliente

Madrid, Junio 2014

This page is intentionally left blank

El/la abajo firmante, matriculado/a en el Master en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "*Creating Serious Games in Unity 3D*", realizado durante el curso académico 2013-2014 bajo la dirección de Dr. Iván Martínez Ortiz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

MATE ANTIČEVIĆ

Madrid, 23 de junio de 2014

Director: Iván Martínez Ortiz

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Iván Martínez-Ortiz for his support and effort put into this work, by providing corrections and suggestions. His experience in the field was essential for completing this work.

I would also like to thank the whole team behind the e-Adventure project for providing me with a high level game authoring tool perfect for use in this thesis.

ABSTRACT

This thesis addresses the problems of the development of serious games, focusing particularly on bridging the gap between of two separate communities: developers and domain experts, that is, educators. Nowadays, students are used to "consume" a large number of games for different purposes (mainly for leisure), and they are expecting a certain quality level in the games (e.g. media assets, story, effects, etc.) and affecting the attitude of playing the game if it does not pass a certain threshold. Hence, to avoid an extra barrier for the use of serious games, it is necessary to take into account the technology used by commercial leisure games. This work tackles these two issues, by fostering the collaboration of these two despair communities by providing both a workflow and supporting tools that are particularly tailored for each of the communities. On the one hand the educators need a simple to use high level authoring tool that allows to sketch the structure and to define the main elements of the game logic without having a great computer science background, and on the other hand developers require powerful game platforms that allow to exploit the technology available in commercial games.

The approach is exemplified by using an educator-friendly authoring tool such as the eAdventure platform and using the Unity 3D™ as the platform used by the developers. The applied solution consists of a tool that connects a high level game design editor eAdventure with a game development platform Unity3D. The approach and the developed tools have been validated through two study cases, where two eAdventure games have been ported to the Unity 3D platform.

RESUMEN

El presente trabajo aborda los problemas asociados al desarrollo de los juegos serios, haciendo particular hincapié en proponer mecanismos que faciliten la colaboración y tiendan puentes entre dos comunidades dispares: desarrolladores y expertos de dominio, es decir, educadores. En la actualidad los estudiantes están acostumbrados a "consumir" un gran número de juegos con diferentes objetivos, aunque mayoritariamente como entretenimiento, por lo que esperan un cierto nivel de calidad en los juegos (e.g. gráficos, historia, efectos, etc.) llegando a afectar incluso la predisposición del alumno a jugar con el juego si no se llega a un nivel mínimo. Por tanto, para evitar una barrera adicional en la adopción y uso de los juegos serios, es necesario tener en cuenta la tecnología y herramientas utilizadas por los juegos de comerciales de entretenimiento. Este trabajo aborda estos dos problemas, fomentando la colaboración de estos dos grupos dispares proponiendo un modelo de trabajo y proporcionando herramientas de soporte específicas para cada una de las comunidades. Por un lado, los educadores necesitan una herramienta simple de utilizar y que permita definir con un alto nivel de abstracción la estructura y los principales elementos del juego sin requerir un gran conocimiento informático y por otro lado los desarrolladores requieren de una plataforma de desarrollo potente que permita explotar las tecnologías disponibles en los juegos comerciales.

Esta aproximación se ejemplifica mediante el uso combinado de una herramienta simple de utilizar por parte de los educadores como es eAdventure y utilizando la herramienta Unity 3D como potente plataforma de desarrollo de videojuegos. La aproximación propuesta se apoya en el desarrollo de una herramienta específica que conecta la herramienta de alto nivel eAdventure y la herramienta de desarrollo de videojuegos Unity 3D. La aproximación y la herramienta han sido validadas mediante dos casos de estudio donde juegos desarrollados con eAdventure han sido portados a la plataforma Unity 3D.

KEYWORDS

- Educational video games
- Serious games
- Game design
- Unity 3D
- Human-Computer Interaction
- eAdventure
- Game engine

PALABRAS CLAVE

- Video juegos educativos
- Juegos serios
- Diseño de juegos
- Unity 3D
- Interacción persona-ordenador
- eAdventure
- Motor de videojuego

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

INDEX OF CODE SNIPPETS

This page is intentionally left blank

CHAPTER I. Introduction

This chapter offers an introduction to this thesis and presents its structure and goals.

1. Serious games

We are all witnesses that computer games are increasingly more popular in the last decade. It is estimated that in 2014th the industry will reach over 100 billion dollars in revenue[1] (Table 1 shows the video game industry revenue in the last four years).

Year	Video game industry revenue
2013	76 billion US dollars
2012	63 billion US dollars
2011	65 billion US dollars
2010	63 billion US dollars

Table 1 Video game industry revenue in past 4 years

Although computer games are often mentioned in a negative context, primarily for promoting violence (Anderson & Bushman, 2001), there are many positive aspects to mention. Even the violent games showed to contribute to better visual spatial abilities according to (Ferguson, 2007), while not showing correlation with player's aggressive behavior. Latest studies show that most effective learning techniques are active, problem based and provide feedback (Boyle, Hainey, & Connolly, 2011). Computer games offer exactly these kinds of activities.

One term that is mentioned often when talking about serious games is gamification. It refers to the concept of using game mechanics and thinking to engage users to solve problems in the context outside of games (O'Brien, 2010). The concept is based on people's natural desire for competition, status and achievements. By awarding users with points, badges or filling of the progress bar for achievements or tasks accomplished they get encouragement to continue, just like in any entertainment oriented game.

Educational video games, or serious games, although increasing in popularity (Hypergrid Business, 2012) and growing in numbers (Wexler, et al., 2007), still take a small part of video game industry, where commercial blockbuster type games dominate. With a significantly lower budget, serious games are not on a same level as commercial ones, regarding that developing high level games requires a big investment. Other than financial requirements, games with a purpose to educate

[1] http://www.gartner.com/newsroom/id/2614915

require people with a great deal of knowledge on the subject in question, often called domain experts. When high budget is not an option, domain experts use special high level tools that don't require great technical skills to use. These tools are designed to bridge the gap that game development makes for domain experts in delivering their knowledge by creating a serious game. Those are exactly the kind of problems which will be further explored in this thesis.

Serious games are often referred to as educational games, as their primary purpose is not to entertain, but to deliver knowledge, to educate the player. Although there are several definitions of the term "serious game" the definition that is closest to the goal of this work is the definition proposed by (Zyda, 2005).

"A mental contest, played with a computer in accordance with specific rules that uses entertainment, to further government or corporate training, education, health, public policy, and strategic communication objectives."

Serious games simulate real world situations and environments, teaching the player how to behave. While they often sacrifice the entertainment part of the game for the sake of delivering the knowledge, entertainment is important, especially if the targeted users are children.

The term and an idea of serious games was first mentioned in 1970 (Abt, 1970), long before modern video games. When talking about serious games, Abt had board and card games in mind, but his idea applies to modern video games as well. He explained that these games have explicit educational purpose and are not primarily for amusement.

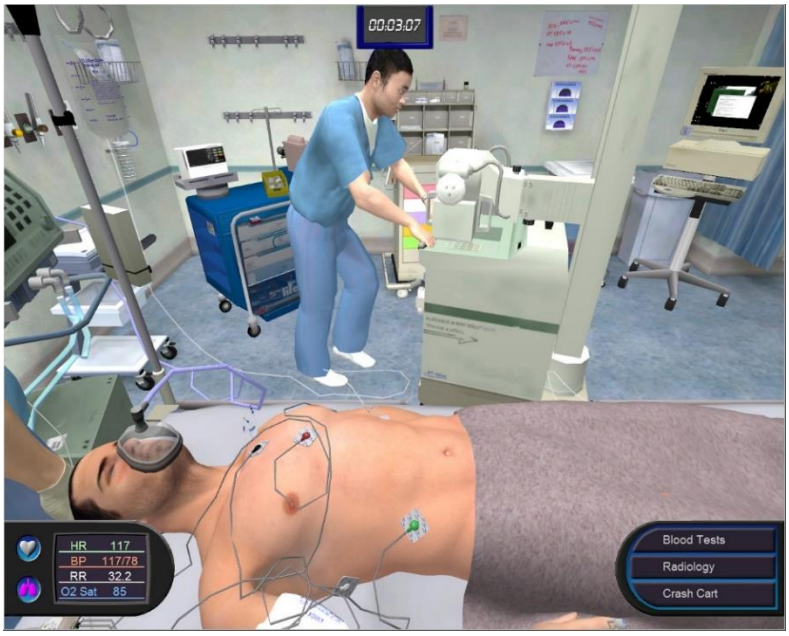

Figure 1 Screenshot from Pulse! – A serious game from the medical sector

Serious games have many useful applications, most notable being their use in professional training in various fields. For example, Figure 1 shows a scene from the game “Pulse!” which is used to train medical personnel. Military and aviation are fields where serious games are exceptionally useful (Proctor, Bauer, & Lucario, 2007), the reason being that training soldiers and pilots only in real world conditions is very expensive and potentially dangerous, while training using serious games is safe and not as much expensive in long term. Obviously these kinds of serious games have to simulate the real world to great detail for them to provide the trainee with useful experience, which inevitably requires big financial investments.

2. Limitations of the current approaches for the serious games development

The main obstacle in creating serious games is that current approaches mainly require the developers to have programming skills, requiring some or even substantial knowledge of the computer science field. To tackle with this limitation, there are new tools that focus on abstracting the development process and which do not require such programming skills. However, they are always limited in terms of features available. Depending on the size and requirements of the project they can be sufficient, but often they are simply not powerful enough.

The other, even bigger obstacle in creating serious games is the development cost. Developing a cutting edge serious game, comparable with a commercial entertainment oriented one, in terms of impressive graphics and advanced technologies, requires huge financial expenses. As Table 2 shows, development costs of video games in the last few years have skyrocketed, and when we take marketing costs into account they are now comparable with the biggest hits from the movie industry.

Table 2 Video games with highest development cost

Game	Year	Estimated development cost
Grand theft auto V	2013	115 million $[2]
Destiny	2014	140 million $[3]
Star Wars: The Old Republic	2011	200 million $[3]
Call of Duty: Modern Warfare 2	2009	200 million $[3]

The irruption of Unity3D has been a breakthrough, particularly related to the licensing costs, proposing a flexible model pricing, a licensing model that allows developers to minimize the upfront costs. Unlike other game engines which require other components to provide various features, Unity3D (simply known as Unity) is an all-in-one solution. Rendering, scripting, asset importing and physics are all

[2] http://www.businessweek.com/articles/2013-09-18/grand-theft-auto-v-is-the-most-expensive-game-ever-and-it-s-almost-obsolete

[3] http://kotaku.com/how-much-does-it-cost-to-make-a-big-video-game-1501413649

included. Other main advantage is its pricing, which is free for private users and almost negligible for professional if compared to alternative solutions. With these perks Unity provides comparable features as its competitors at a considerably lower price, thus presenting a good choice for serious games development.

Even if the high development costs are not the problem, collaboration inside of a heterogeneous team can have its obstacles. Since domain experts more often than not don’t poses programming skills, they have to find a “common language” with game developers, to successfully deliver their ideas.

3. Goals

For this work we have proposed the following goals:

- Make an analysis of the process of serious games development and point out the obstacles and difficulties.
- Make a critical analysis of game development platforms and game authoring tools suitable for serious games.
- Propose a collaborative workflow between domain experts and game developers to facilitate serious game development.
- Create a software tool that will facilitate collaboration between domain experts working in a high level tool and game developers working in a powerful and professional game development platform. Tool works in one direction translating the high level tool to low level and powerful tool.
- Test the proposed workflow and tools using eAdventure and Unity3D as a case study.

4. Structure of this Document

The rest of this document is structured as follows:

- CHAPTER II – Introduces the concept of serious games, their history and their usage today. Then, serious games from various fields of application are presented. Finally an overview of existing professional game development tools and game authoring tools is done.
- CHAPTER III – Difficulties in serious game development are explored, focusing mainly on collaboration between domain experts and game developers. The approach in serious game development is proposed and described.
- CHAPTER IV – Applied approach from the previous chapter is presented. Applied approach involves connecting eAdventure as a high level authoring tool to Unity, a game development platform. Therefore bridging the gap between domain experts and game developers.
- CHAPTER V – Developed tool described in the previous chapter is tested on two example games.
- CHAPTER VI – Conclusions from the work and future work.

CHAPTER II. Related work and applications

In this chapter different approaches to serious game development will be presented. From lower level game development platforms that can be used to create any type of game, through authoring tools made for rapid game development, to specialized high level tools that require no programming knowledge.

1. Educational Video Games

It is hard to say when first serious video games appeared, some argue it could even be before entertainment video games (Djaouti, Alvarez, Jessel, & Rampnoux, 2011). For example, first commercial home video game console, the *Magnavox Odyssey*, was shipped with both entertainment and educational games.

One of notable mentions is a 1971 text based game designed to teach school children about 19th century pioneer life on Oregon Trail. *The Oregon Trail* (Minnesota Educational Computing Consortium, 1974) was developed by three historians on a *HP 2100* minicomputer. Today, the game has various editions and is supported on the majority of popular platforms.

A more modern application of serious games is a military training based game *America's Army*[4] (see Figure 2), developed by United States Army. The idea is to provide the future soldier with virtual army experience, without exposing him to actual danger. Over the years, the game gained commercial popularity and is played online as a multiplayer game. This game is a great example how entertainment oriented and serious games don't have to be separated, on the contrary, combining the two can prove very useful.

Figure 2 America's Army multiplayer mode

[4] https://www.americasarmy.com/

The rest of this chapter provides a brief analysis of a sample of the myriad of serious games available and applied to different aspects of both formal and non-formal education and provides a critical analysis of the game development platforms that are more relevant to this work.

Untrusted

Untrusted[5] is a very unusual web browser game written in JavaScript with a purpose to teach you the same language it's written in. The game is suitable for any user interested in JavaScript, no matter how skilled he is. Beginners can learn some basics, while more experienced users can consolidate what they already know and have fun in the process.

Figure 3 Untrusted game interface

Untrusted has 21 levels and in each the player has to get to the exit portal, meanwhile satisfying specific goals. As seen in Figure 3 the game interface is split in two parts. Left part is the graphical display of the game level, while the right part displays the underlying script that creates it. User is allowed to change certain parts of the script to be able to pass the level. Every level provides multiple ways to pass it, challenging players to find the most creative way to solve the puzzle.

[5] http://alexnisnevich.github.io/untrusted/

Scratch platform

A platform game or a platformer is usually 2D type game that involves an avatar which is controlled by the player. The avatar jumps between platforms and over obstacles often killing or avoiding various types of enemies. Scratch[6] is a free tool developed by MIT (*Massachusetts Institute of Technology*) designed to enable users to easily create platformer type games and doing so learn the very basic concepts of algorithm design and acquiring basic programming skills.

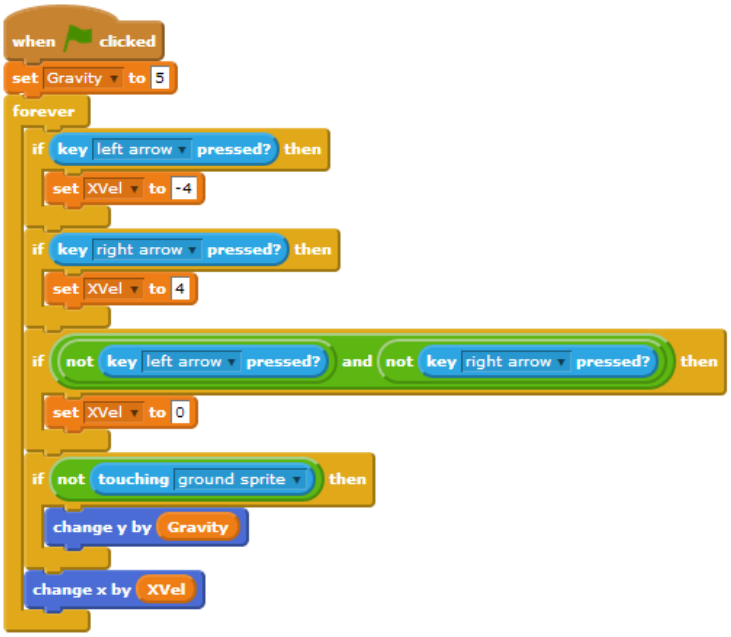

Figure 4 Scratch platform - game logic

Creating new games in Scratch is easy, intuitive and does not require any programming skills. An example of game logic designed in Scratch is shown in Figure 4. It's clearly visible how it resembles a programming language, with loops, conditions, and variable setting.

SimSchool

As the title says simSchool[7] is a classroom simulation (see Figure 5). It is like a "flight simulator" for educators, a place where they can explore teaching strategies, classroom management techniques and practice building relationships with students that will help them learn more efficiently. The teacher learns about diverse

[6] http://scratch.mit.edu/
[7] http://www.simschool.org/

styles of learning, and can practice designing tasks for the students and get the feedback.

Figure 5 simSchool gameplay

The game is browser based and free to use. Users can register for a simSchool account at any time by visiting their website. They offer a number of research articles and videos to help the user get the working knowledge of features available within the environment.

e-Bug

e-Bug[8] is place to play games and learn about microbes in the same time. Depending on the student age, there is a junior and a senior version. The platform offers different types of games teaching the same material from different aspects. One of them is a simple quiz type game to test student's knowledge. In other games, the player is shrunk down to microbe size and explores how human body interacts with viruses, bacteria and fungi. There is also quite a lot of textbook style reading material accompanied with images. The final objective of this game is to have an impact in the habits of the players that can affect their health, showing them during the game and solving the puzzles what are the risks of the different microbes.

SimCity Edu

SimCity is a well-known game brand with a long history of various editions. Since the first installment that was published in 1989, the game was a city-building

[8] http://www.e-bug.eu/

simulation video game (see Figure 6), where the player has no ultimate and final goal besides advancing the city. The player has to build city infrastructure, residential objects, schools, police departments and etc.

Figure 6 First installment of SimCity game series (1989)

Based on original SimCity games, SimCity Edu[9] focuses more on the education part rather than entertainment. It is a regular SimCity game customized for learning and assessment purposes (Higgins, 2013). While students try to solve game tasks, game play data is gathered and processed, providing feedback to their teachers and parents.

The Mystery of Taiga River

This game[10] sets the player in the Taiga National Park environment. Player's acts as an investigative reporter and has to determine the water quality and park health in general using scientific concepts and methods.

[9] http://www.simcityedu.org/
[10] http://science.asu.edu/node/1995

Figure 7 The Mystery of Taiga River gameplay

Figure 7 shows how the game teaches a scientific concept. The existing hypothesis can be proved logical or inconclusive through a chain of facts, which can have different chain strength. Repeated testing on the pre and post learning gains revealed a significant increase in student's score (Taiga River).

iCivics

The iCivics games[11] place the player in different civic roles to address real-world problems and issues. Teachers around the world have successfully implemented iCivics into their curriculum. Students who use the program are more knowledgeable, engaged, and eager to participate in civic actions and discussions (LeCompte, Moore, & Blevins, 2011).

FoldIt

FoldIt[12] is a computer video game which enables players to contribute to important scientific research. The research field is protein folding. Proteins are an essential building material in human body. They come in thousands of different varieties, but

[11] https://www.icivics.org/
[12] http://fold.it/portal/

have same building blocks, amino acids that make a chain. The amino acid chain, can be folded in various ways, and that is the player's objective. Solving game puzzles by folding the amino acid chain, player contributes to better understanding of the protein in question (see Figure 8).

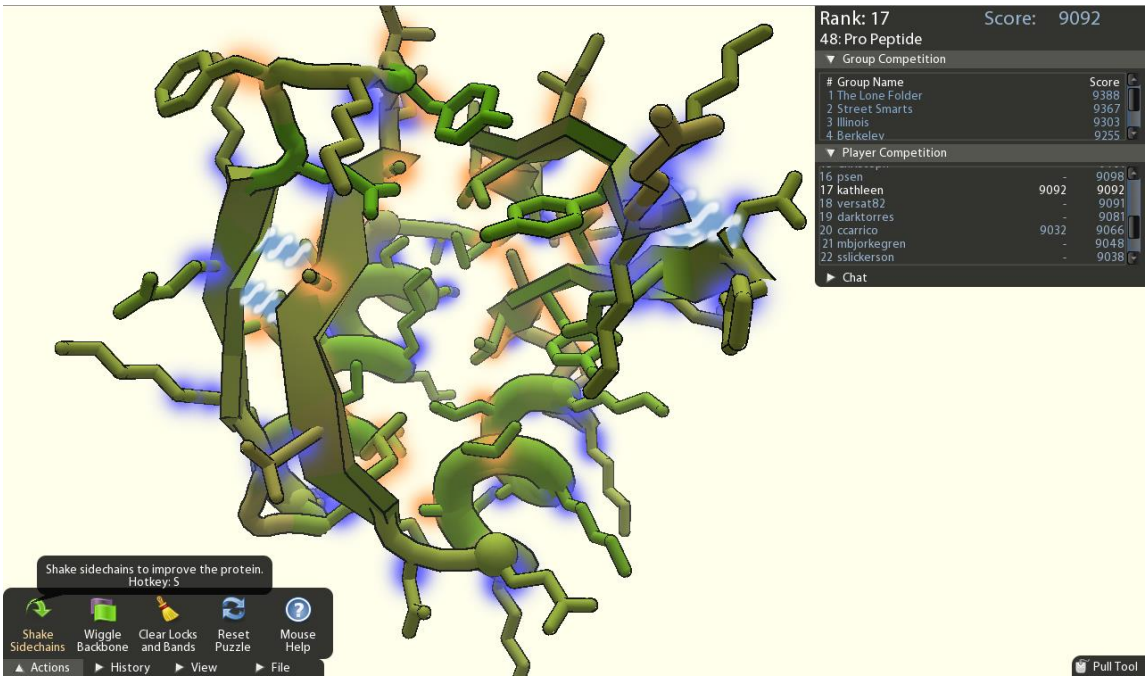

Figure 8 FoldIt puzzle gameplay

2. Industrial solutions

While designing a video game, a developer will find himself frequently having to realize very similar behaviors. Creating a game from the scratch every time would therefore not be very productive. To solve that problem, there are development platforms that offer a library of common behaviors, making it easier for a developer to reuse repeated functionality. These platforms also often offer exporting to multiple platforms, saving the developer time programming the same game multiple times.

Unity3D

Unity 3D[13] is a game engine used to develop complex games. It is cross platform, and supports exporting to various platforms like Windows, Android, iOS, Linux, Flash, PlayStation and others. Latest version of Unity is 4.3.3 and it will be discussed and used in this project, while Unity 5 is coming very soon. There are two editions available, professional edition with all functionalities included, and a free one with some advanced features locked[14].

Figure 9 Unity IDE

[13] http://unity3d.com/
[14] http://unity3d.com/unity/licenses

Unity was created primarily for creating 3D games, but as of 4.3 release it has introduced native 2D support. It supports model importing for a variety of popular file formats like 3ds Max, Cinema 4D, Maya and others. Graphics rendering engine uses Direct3D and OpenGL depending on the targeted platform. Unity itself was written in C++, and for scripting purposes supports C#, UnityScript (JavaScript in essence) and Python inspired language Boo. It also offers an intuitive IDE (see Figure 9) and a tool MonoDevelop for script debugging and an intuitive editor which enables a quick start even for programmers beginning with game development.

The Unity3D framework also includes a powerful 3D physics engine nVidia PhysX Physics[15], so developers can concentrate on the game itself rather than waste time on implementing physics.

Advantages

- Built-in physics engine
- C# support
- Cross platform
- Free and easy to use
- Simple yet powerful

Disadvantages

- Programming skills required
- Some features only available in professional version

Although games made with Unity are not comparable in size and complexity with commercial blockbuster games, some are worth mentioning:

- ARMA Tactics
- Deus Ex: The Fall
- Family Guy Online
- Surgeon Simulator 2013
- Temple Run

[15] https://developer.nvidia.com/gameworks-physx-overview

Unreal Engine

The Unreal Engine[16] is one of the most powerful game engines today. It is developed by Epic Games[17], and was first introduced in 1998 with a first person shooter (FPS) game *Unreal*. Engine is developed using C++, it features a high degree of portability and is used by many game developers today. Unreal engine is a very powerful tool and requires considerable programming knowledge.

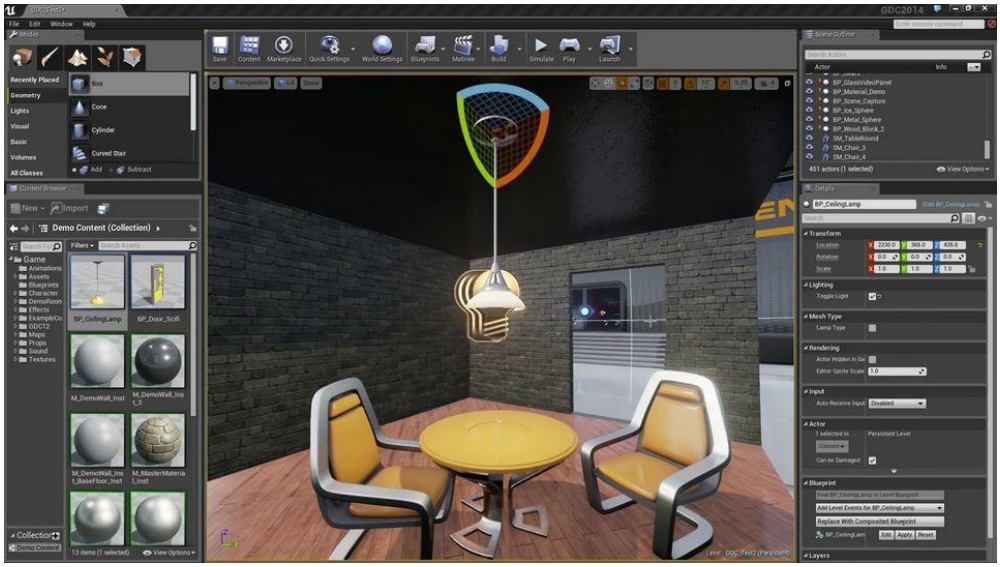

Figure 10 Unreal Engine 4 development tool

There is a huge number of commercial games based on the Unreal Engine, some of them are:

- America's Army
- Batman: Arkham Asylum
- BioShock
- Deus Ex
- Duke Nukem Forever
- Thief

[16] https://www.unrealengine.com/
[17] http://epicgames.com/

Current version of the engine is 4.1, which brings support for Microsoft's Direct X 11[18]. Another great feature of the new version is that they are doing the first steps of porting the game engine to the web. Using Emscripten[19] they managed to translate C/C++ low level virtual machine (LLVM) code to JavaScript and by that running the game engine on web (Mozilla, 2014).

All previous versions of the engine used UnrealScript for scripting purposes. Version 4 removes UnityScript, and focuses only on C++ instead.

Advantages

- One of the most powerful game engines today

Disadvantages

- High hardware requirements
- Only C++ supported
- Requires considerable programming skills

[18] http://windows.microsoft.com/es-es/windows7/products/features/directx-11
[19] https://github.com/kripken/emscripten/wiki

Microsoft XNA

XNA[20] is based on Microsoft's .NET Framework. It is a framework designed to develop games for multiple platforms, mainly for Windows, Windows Phone and Xbox 360. The toolset was first released in 2006. Most current version is 4.0, released in 2011, and since then Microsoft has stopped developing the framework. Games for XNA can be written in any language supported by .NET, and mostly using Visual Studio as developing environment.

XNA is a good place to start for programmers beginning with game development. For people familiar with .NET Framework it is fairly easy to start developing games with XNA. However, it requires a lot of programming, especially because it lacks a built-in physics engine. The other downside is that only Windows platforms are supported. In conclusion, it's a good place to start and to get an idea how game development works, but nothing more.

Advantages

- Cross platform – Windows, Windows Phone, Xbox
- Easy to get started
- Supports .NET languages
- Supports 2D and 3D

Disadvantages

- No built-in physics engine
- No longer being developed
- Requires considerable programming skills
- Windows only

[20] http://msdn.microsoft.com/en-us/library/bb200104.aspx

Source Engine

Source[21] is a 3D video game engine developed by Valve Corporation[22]. It was first introduced with a FPS game *Counter Strike: Source*. Although it was initially designed for FPS games, it was later used for role-playing, puzzle and other game types. Source Engine doesn't have any meaningful version numbering system, instead it uses constant incremental updates distributed by Valve Steam[23] service.

Source is a very powerful engine which includes a number of advanced built-in technologies. One of the most important is the built-in physics engine *VPhysics* which was derived from Havok[24]. Engine simulates real world physics taking into account mass, gravity, friction, air resistance, inertia and buoyancy (Valve, 2011). Other notable technologies supported include high dynamic range rendering (HDR), scalable multiprocessor support, 3D bump mapping and a facial animation system.

Advantages

- Built-in physics engine (derived from Havok)
- Cross-platform (Android, Linux, OS X, PlayStation 3, Xbox, Windows)
- Scalable multiprocessor support
- Very powerful

Disadvantages

- Requires considerable programming and game development skills

Majority of games based on Source engine are developed by Valve themselves, some of them are:

- Counter-Strike: Source
- Left 4 Dead 2
- Half-Life 2
- Portal 2
- Team Fortress 2

[21] http://source.valvesoftware.com/
[22] http://www.valvesoftware.com/
[23] http://store.steampowered.com/
[24] http://www.havok.com/products/physics

3. Low cost solutions

LibGDX

The LibGDX[25] is an open source Java game development framework. It provides a unified API for all supported platforms, which include Windows, Linux, OS X, Android, BlackBerry, iOS and of course Java. Its goal is to provide an environment for rapid prototyping and fast iterations without having the developer worry about low level details.

Framework includes math and physics libraries to simulate real world physics. Streaming music and sound effect playback is also supported for WAV, MP3 and OGG. Input handling provides abstractions for mouse, touch-screen, keyboard, accelerometer and compass. OpenGL ES 2.0[26] takes care of all graphics rendering.

Advantages

- Active user community on forums
- Built-in physics
- Cross platform and open source
- Input handling

Disadvantages

- Lack of documentation and information
- Quite low-level

Cocos2D

Cocos2D[27] is an open-source framework for building cross platform 2D games. Development is only available with XCode and Objective-C while the targeted platforms are iOS and Android. Visual editor *SpireBuilder* is included enabling easy and intuitive game development. Coscos2D is flexible and works with any third-party library and all of Apple's native APIs.

[25] http://libgdx.badlogicgames.com/
[26] http://www.khronos.org/opengles/2_X/
[27] http://www.cocos2d-iphone.org/

There was also an Android version of Cocos2D, all in Java, which is no longer used and supported. It's recommended to use Cocos2D-x for Android development.

Advantages

- Active user community on forums[28]
- Cross platform and open-source

Disadvantages

- Development only in C++ and Objective-C

Some notable games for iOS and Android developed with Cocos2D are:

- Badland
- Farmville
- StickWars
- ZombieSmash

Quintus

Quintus[29] is an easy to learn and use JavaScript HTML5 game engine for mobile and desktop platforms. Development is limited to 2D games, and most suitable for designing platformer games. It's most suitable for rapid development of simple web browser games (see Figure 11), rather than more complex games.

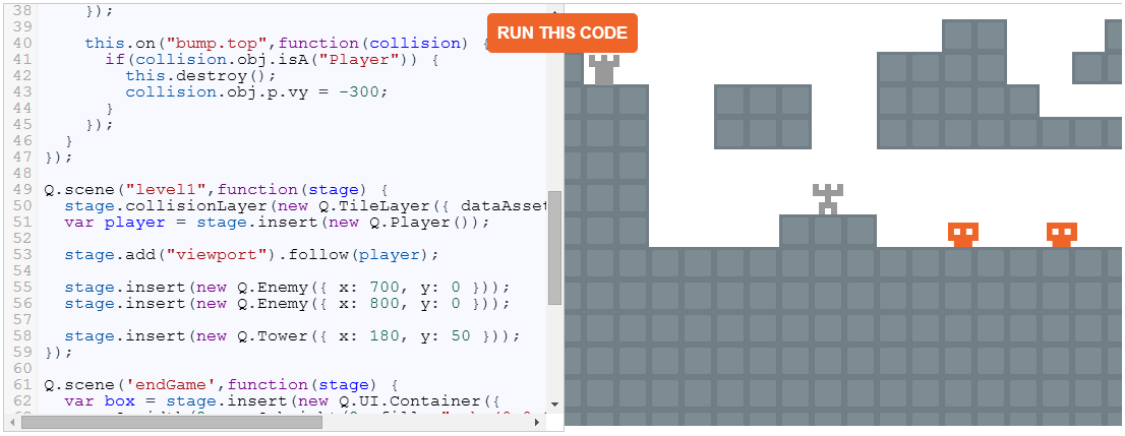

Figure 11 Quintus editable demo game level

[28] http://discuss.cocos2d-x.org/
[29] http://html5quintus.com/

Advantages

- Easy to use
- Fast game development
- Free and open-source

Disadvantages

- Only web based
- Very limited

GameSalad

GameSalad[30] is an authoring tool aimed primarily for non-programmers. Rather than programming, game is built using drag-and-drop actions inside a visual editor and combining it with behavior-based logic system. Since programming is not necessary, GameSalad has a lot of features built-in, like the integrated physics engine which uses a rigid-body simulator for handling motion and collision.

Advantages

- Cross platform (iPhone, Android, Windows Metro, HTML5)
- No programming required

Disadvantages

- Very limited

ImpactJS

ImpactJS[31] is another JavaScript Game Engine that allows you to develop stunning HTML5 Games for desktop and mobile browsers. All major browsers are supported. It is one of the first professional engines to run inside the HTML5 canvas element (Szablewski, 2011). Framework includes a flexible level editor (see Figure 12) which allows you to create versatile game worlds with ease. ImpactJS is not free but it offers a powerful game engine for a reasonable price.

[30] http://gamesalad.com/
[31] http://impactjs.com/

One of great features of Impact game engine is the module system. Since JavaScript doesn't feature an "include" concept, module system helps you organize game code into separate files which in the end get joined into a single game file.

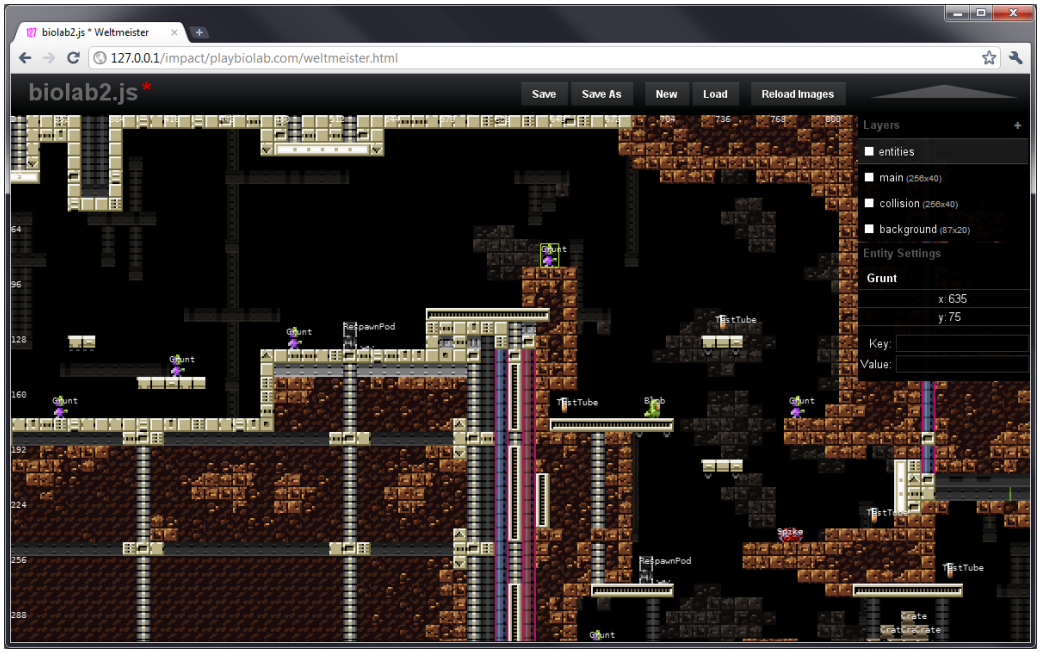

Figure 12 Weltmeister - level editor running inside Chrome web browser

Advantages

- Great documentation and support
- Intuitive level editor
- Powerful game engine

Disadvantages

- No official Android support
- No trial license available

4. Video Game authoring tools

Working with tools like Unity and Microsoft XNA requires user to have considerable knowledge of both the software itself and programming to design games. For beginners or users who don't have the time to learn using them, there are more simpler tools for rapid game development, often requiring very little or no programming knowledge. These kinds of tools are very limited, but appropriate for smaller projects and game developing beginners.

Construct2

This is a tool for rapid game development. Using HTML5 it targets multiple platforms: Windows Phone, Mac, Linux, iOS, and Android. Construct2[32] supports only 2D and is primarily intended for beginners. It includes an IDE (see Figure 13) which makes game creation easy and intuitive using the drag and drop method, and it doesn't require learning a programming language. Construct2 comes in three editions: Free, Personal and Business.

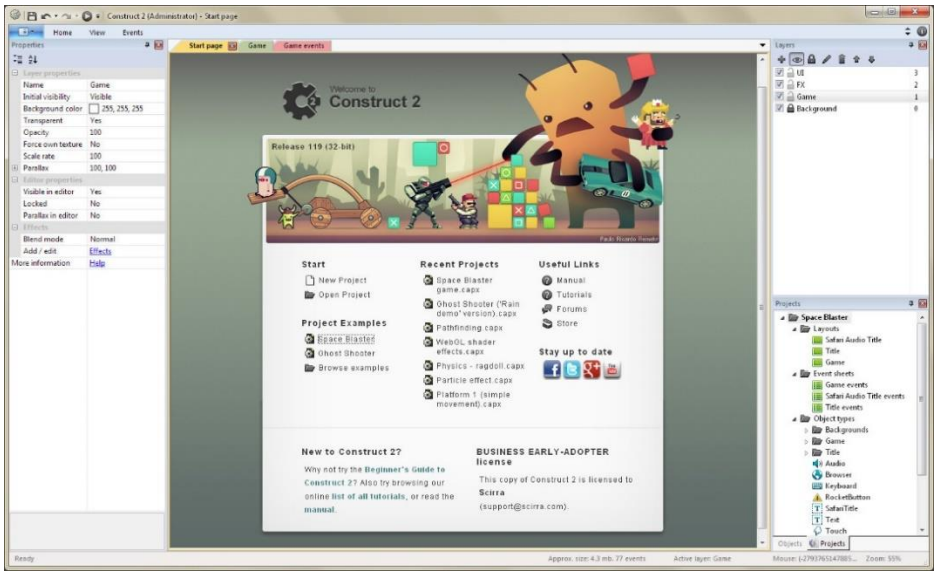

Figure 13 Construct2 game editor

[32] https://www.scirra.com/construct2

Advantages:

- Cross-platform
- Doesn't require programming
- Easy to use

Disadvantages:

- Limited features
- No 3D support

Game Maker Studio

This is another tool for rapid game development, according to their web site[33] they promise 80 percent faster development. Editor (see Figure 14) supports standard drag and drop method and they provide a scripting language called GML (*Game Maker Language*). Game Maker Studio can export games for Android, iOS, Windows and web (HTML5). This tool is intended for intermediate beginners and comes in various editions with free edition also available.

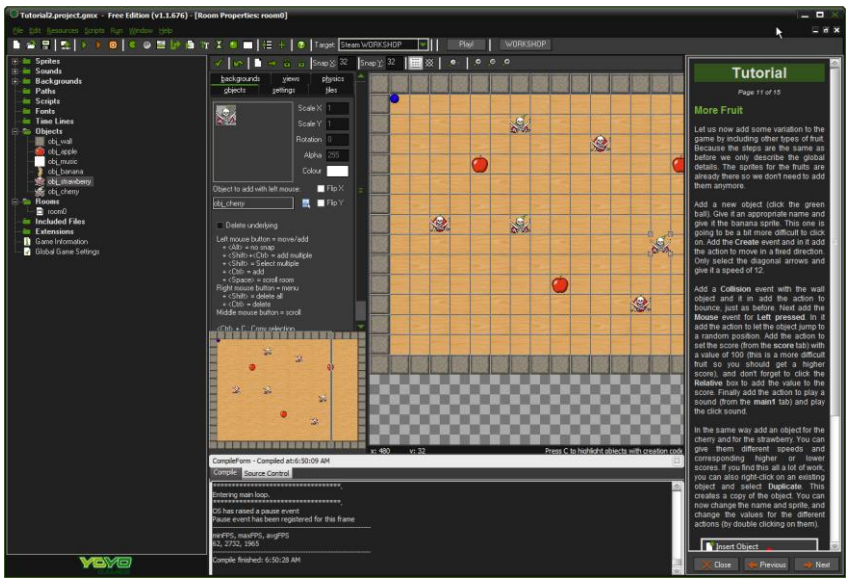

Figure 14 Game Maker Studio editor

[33] http://www.yoyogames.com/studio

Advantages:

- Exporting to multiple platforms
- Rapid game development

Disadvantages:

- Uses a special scripting language (GML)

Sploder

This is a tool for total beginners[34] and makes game development very easy. Of course, it is not nearly as powerful as other tools mentioned. Sploder is web based, and created using Flash technology. There are several predefined types of games you can create, which makes it very limited, but on the other hand very easy to use, without any programming or game design knowledge.

Advantages:

- Fast game development
- No programming required
- Very easy to use

Disadvantages:

- Only web based
- Very limited in features

Thinking Worlds

This tool is specifically designed to create educational games and simulations[35]. It doesn't require programming knowledge and it's free to try. Created games can be exported as a Java or an Android application. It offers an interactive 3D editor, which makes game development simple and fast.

Advantages:

- Specialized for serious games

Disadvantages:

- Somewhat limited in features

[34] http://www.sploder.com/
[35] http://www.caspianlearning.co.uk/products-2/thinking-worlds

5. Other tools based on narrative or story writing concepts

Adobe Captivate

Captivate[36] is an electronic learning tool developed by Adobe, designed to author software demonstrations, simulations and randomized quizzes. The software has a long history, it started as a simple screen recorder under a different name, to eventually get under Adobe and changing the name to Captivate. It supports output to HTML5, video and flash, also conversion between Microsoft PowerPoint is possible. The main advantage of this tool is a powerful interface (see Figure 15) which is highly interactive and easy to learn.

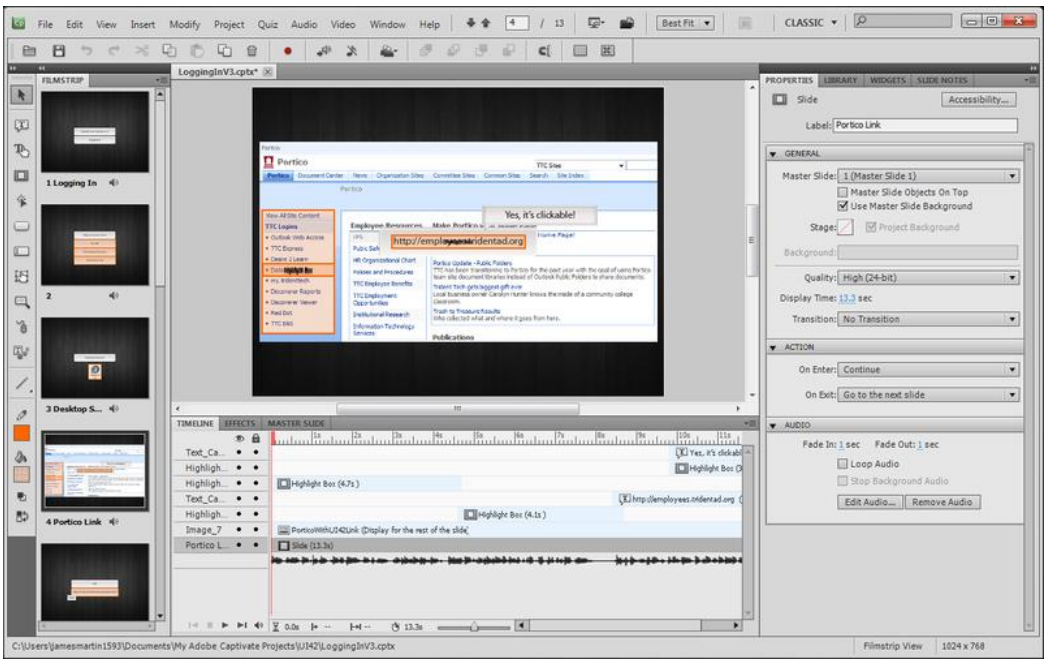

Figure 15 Adobe Captivate interface

Advantages:

- Very easy to use

Disadvantages:

- Not free
- Not really suited for games

[36] http://www.adobe.com/es/products/captivate.html

Articulate Storyline

Storyline[37] is an alternative to Adobe's Captivate. It offers an extremely user friendly and intuitive interface, requiring no training before getting started. Storyline includes a number of predefined templates and characters, enabling users a rapid development. The application interface (see Figure 16) is simple and even users with lower computer skills will find it easy to understand.

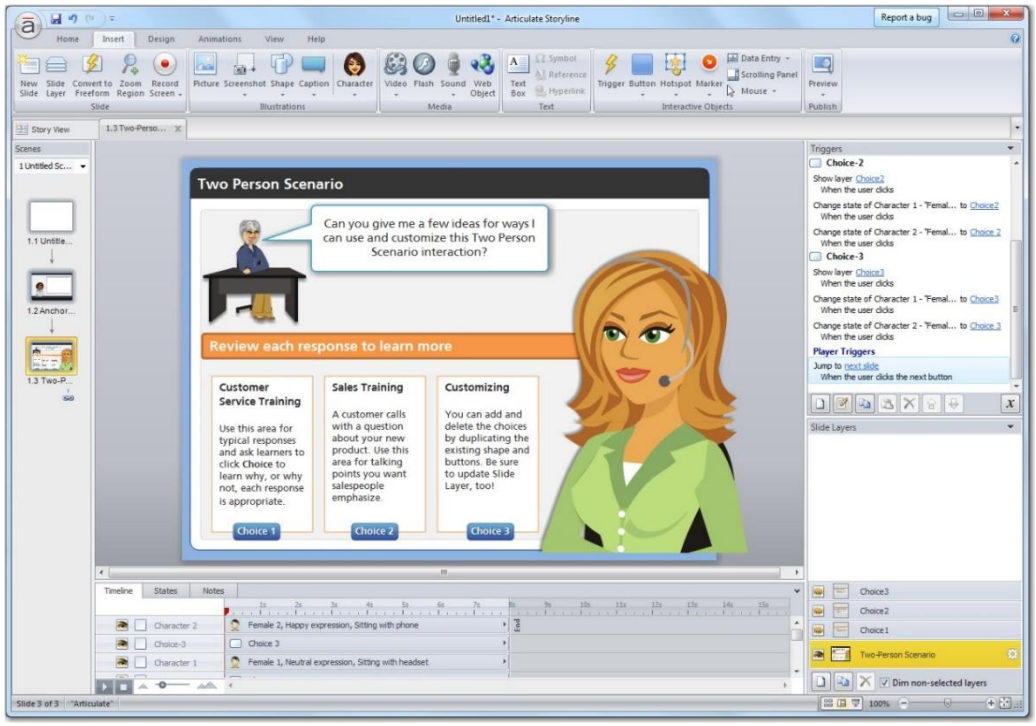

Figure 16 Articulate Storyline interface

Advantages:

- Very easy to use

Disadvantages:

- More expensive than Adobe Captivate
- Not free
- Not really suited for games

[37] https://en-uk.articulate.com/products/storyline-overview.php

6. The e-Adventure Educational Video Game Development platform

The eAdventure platform[38] is a research project developed by e-UCM Research Group at the Faculty of Informatics of the Complutense University of Madrid. It provides an environment to create educational games and simulations. The project is open source[39] and written in Java.

The development of the eAdventure platform started in 2005 (Torrente, Del Blanco, Marchiori, Moreno-Ger, & Fernandez-Manjon, 2010) conceived as a tool to ease the creation of educational 2D point-and-click adventure games like the Myst© or Monkey Island© saga that were very popular in the 90's. The main objectives for the eAdventure platform were:

- To reduce the development cost of serious games.
- Minimize the technical knowledge required to use the tool. No programming should be needed to create an educational game or simulations and educators should be able to create or to modify a game.
- Games should be easy to deploy and maintain. That is, should be ease to deploy them in schools and should be feasible to modify and customize them by other educators.
- Games should be interoperable with other educational and e-learning tools like Learning Management Systems.

Graphical editor (see Figure 17) of eAdventure does not require any programming and the concept is based on the creation of a virtual world by defining multiple scenes, and then adding interactive elements such as characters and objects, as well as the game rules and game story.

[38] http://e-adventure.e-ucm.es/
[39] http://github.com/e-ucm/

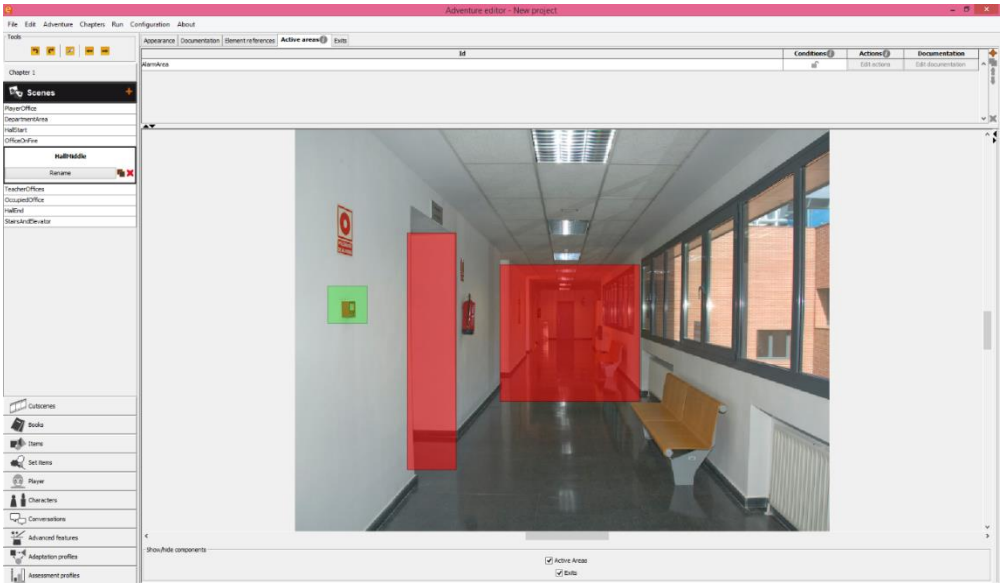

Figure 17 eAdventure game editor

Advantages

- Free and open source
- Intuitive and easy to use
- Includes predefined features ready to use
- Requires no programming skills

Disadvantages

- Limited possibilities
- No 3D support

7. Conclusions of this Chapter

Serious games are growing in numbers and in popularity. Still, they have a long way to go, as they are often not on the same level as other commercial entertainment oriented games. This is due to the very expensive development process which is unavoidable for cutting edge games, for example, the development of the new addition to Grand Theft Auto series was a project worth 115 million $ (Brustein, 2013). Seeing that budgets for serious games are not nearly as high as for entertainment oriented ones, quite often the only solution is to resort to an alternative development process.

Solutions like game authoring tools enable people with no programming skills to create decent looking games. Limitations of this approach are often not very impressive and feature limited games, but depending on the goal and application of the game in question it can be sufficient.

Powerful commercial game engines offer almost endless possibilities, but require considerable programming and game development skills. Even so, when a game requires higher complexity and impressive graphics, this is the only way to go.

CHAPTER III. Description of the approach

This chapter will present the proposed approach for better collaboration in heterogeneous teams which are unavoidable for serious game development.

When creating a serious game, on one side there are domain experts who have the knowledge on what to do but lack knowhow on how to do it, on the other side there are programmers who have the knowhow, but lack domain knowledge.

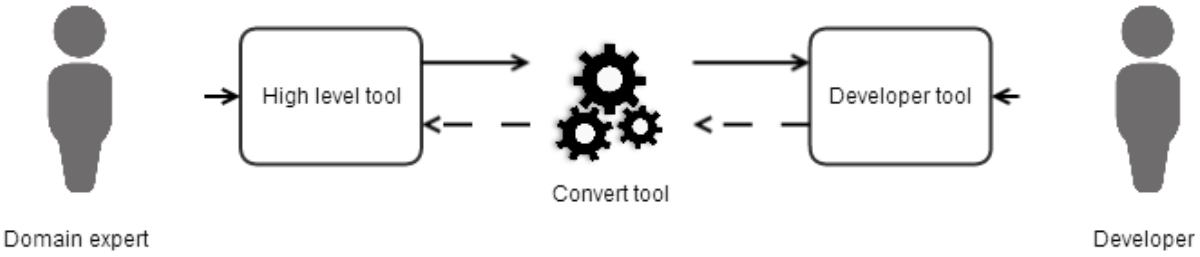

Figure 18 Abstract concept of the approach to advance collaboration between domain experts and developers

Abstract concept of the approach is presented on Figure 18. First step is to provide domain experts with relatively simple tools they can use without too much technical knowledge to design the game, that is, an authoring tool. To connect domain expert's authoring tool with the developer's tool, some kind of conversion tool is necessary, which works at least in one direction, preferably in both. Since the main goal is delivering domain experts ideas to developers, that direction is from the high abstract level authoring tool to the developer's tool. Other direction of conversion would also be useful, but may not be possible to implement seeing that a developer's platform is more low level and offers a bigger range of possibilities which may not be translatable to the high level tool.

1. Collaboration Workflow

Serious games are a result of multidisciplinary work. When experts from different fields with different backgrounds work together certain problems can arise. Experts from different fields usually use different terminology, and their project goals may differ drastically. For example game programmers are mainly focused on reducing complexity and increasing maintainability of the code, while domain experts want the game to include every detail of their field.

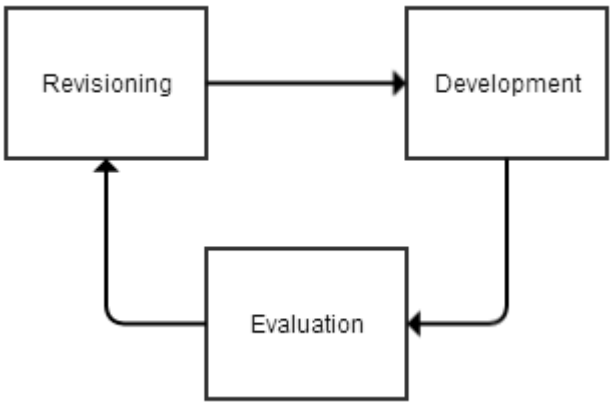

Figure 19 Cyclic development process

One of the ways to develop in such environments is proposed on Figure 19. The development process is iterative and incremental. It produces an improved product with every iteration. The cycle contains three steps. Revision consists of planning what to implement in current iteration, the domain experts start to work with the authoring tool (the tool that is simple to use) whereas developers evaluate and, if needed, improve the conversion tool. This is a crucial step where developers and domain experts have to work together. Development part depends exclusively on developers, just like the evaluation depends on the experts. During the development step, developers use the converted game as a starting point on which they build upon and improve the game. As a result of evaluation, they provide feedback back to developers starting a new iteration. Their feedback regards the quality of the final product and reviews that the main elements of the game are not missing or modified.

Figure 20 proposes an approach to collaboration between experts in a similar way as proposed by Marchiori et. al. (Marchiori, Serrano, del Blanco, Martínez-Ortíz, & Fernández-Manjón, 2012). These steps are presented on Figure 20.

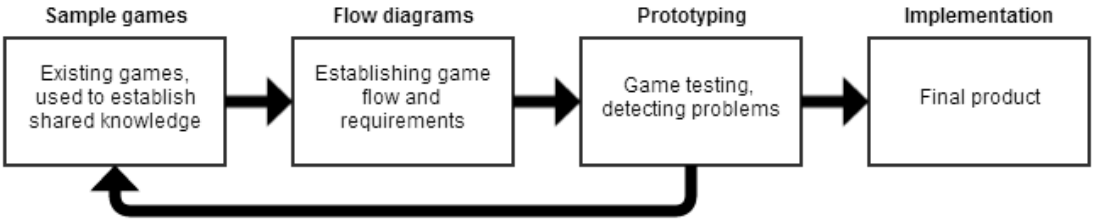

Figure 20 Steps to develop a game through collaboration

1. First step is studying existing similar games and sample games that have been created with the authoring tool, although the developers has access to complete arsenal of tools and features the main objective of the proposed approach is to let the domain experts drive the development process, hence the game will be restricted by the features available in the authoring tool. This study builds shared knowledge and familiarizes the team with both the domain of discourse (the topic or topic that will be covered in the game) and the game rules, mechanics, etc. that are available to develop the game.
2. Second step is to construct flow diagrams to give a high-level overview of the game and to point out the key elements. It's serves as a common ground for all experts to exchange ideas and concepts about the main game elements and how they interact.
3. Third step is the rapid prototyping which is the best way to discover potential errors or oversights. Based on the knowledge acquired in previous steps a straw man prototype is built in order to foster the discussion between the team around the prototype and to assess how it addresses the proposed objectives. This prototype is built using the authoring tool to shorten the iteration cycles because the domain experts will implement the prototype assessed by the developers. However developers can translate the prototype to the game platform in order to show the domain experts a quick overview of the final product or to let them evaluate if the introduced polishing details meets their requirements.

4. These first three steps are organized in an iterative way in order to facilitate the design en prototyping of the game. Each iteration of this process improves the game prototype including new game design elements and also testing and evaluating final details of the final game.
5. Building upon prototypes and testing results in the fourth step the final version of the game is implemented.

In the first three steps domain experts act as a main workforce driving the development process whereas game developers act as mentors and assessors. On the other hand, they change roles in the last step where the game developers develop the final game and the domain experts evaluate and suggest improvements in the final game.

2. Model conversion

Development of a game in an abstract high level tool results in a model which can be in a various kinds of formats. It is a requirement to have a clearly defined definition of the high level game model in order to achieve the model translation. In order to convert the source model into something a developer can work with, it's necessary to import it and create an internal model, objects representing the elements inside the source model.

Since both the source and the target model can be in various formats, certain problems can arise when converting one to the other. For example the object-relational impedance mismatch concerns difficulties in translation of relational database management system[40] (RDBMS) models to models suited for object-oriented (OO) programming. One example of the mismatch between models is data type differences. RDBMS models strictly prohibit pointer types, while OO languages embrace this concept. Accessibility wise, concepts like *private* and *public* which are known in OO languages, are not recognized in relational models. These and other difficulties make it hard to design an automatic conversion between the source and the target model, so the best solution in many cases is doing the conversion manually. While this approach is obviously more time consuming it offers better control over the creation of the target model, and therefore better results in the end.

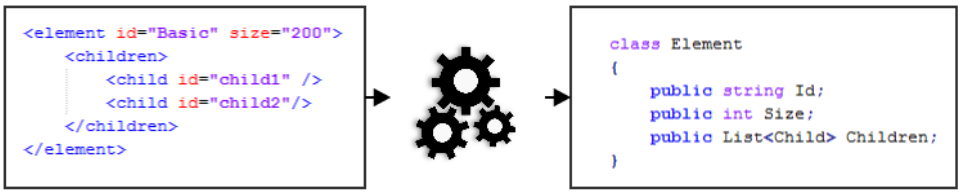

Figure 21 Example of the model conversion from the source model to the target model

Figure 21 shows an example of the model conversion. High level game model is represented by an XML document while the target is a C# class. Attributes specified in the source model are mapped to properties like *Id* and *Size*, while the element *children* which contains multiple instances of child elements is mapped to a generic collection.

[40] http://www.agiledata.org/essays/impedanceMismatch.html

3. Conclusions of this Chapter

Out of all the difficulties that arise in the development of serious games, one of the biggest and the one which is the main focus of this thesis is the gap between domain experts and game developers. Domain experts are the ones who have the knowledge, and the goal of a serious game is to deliver that knowledge in a proper form to the player.

The proposed approach enables closer collaboration between experts and developers. By translating games made with high level tools to professional game development platforms, game developers can build upon the main concepts and ideas delivered by domain experts.

In the next chapter this approach will be further expanded and applied to connect The e-Adventure Educational Video Game Development platform as a high level tool with Unity3D as a professional game engine. Connection will be achieved by designing a tool to translate eAdventure games to an editable Unity project.

CHAPTER IV. Application of the approach

The proposed approach from the previous chapter will be further expanded and applied in this chapter.

Using abstract concepts from the previous chapter, a practical application will be built in this one[41]. The goal is to connect eAdventure as a high level tool made for domain experts to Unity 3D as a game developers tool (see Figure 22). The eAdventure model is represented in a XML format, while the target model is represented by a C# script library. The abstract model and tools proposed in the previous chapters are put into work in this chapter.

The experimental work done as a part of this work will focus only on one way translation, from the high level authoring tool eAdventure to Unity game development platform. Translation in other direction carries even more complexities and will not be discussed further in this work.

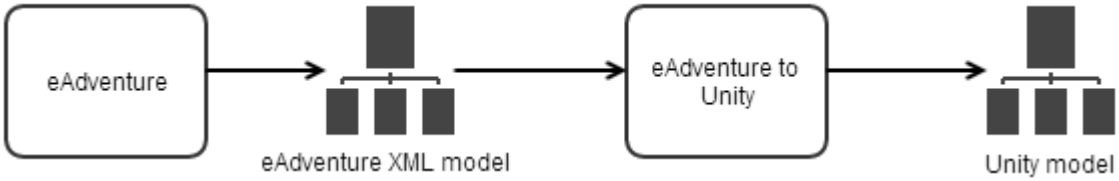

Figure 22 Applied approach of translating an eAdventure game to Unity

The idea is to create a standalone application that receives eAdventure's XML files and generates C# scripts that can be used to recreate and run the eAdventure games inside Unity. Application requires users only to provide XML file location, therefore there is no need for a graphical user interface, and a simple console user interface is sufficient, however a simple graphical user interface will be used just to make application usage easier and to give the users feedback about conversion errors.

[41] https://github.com/mateanticevic/sgsp

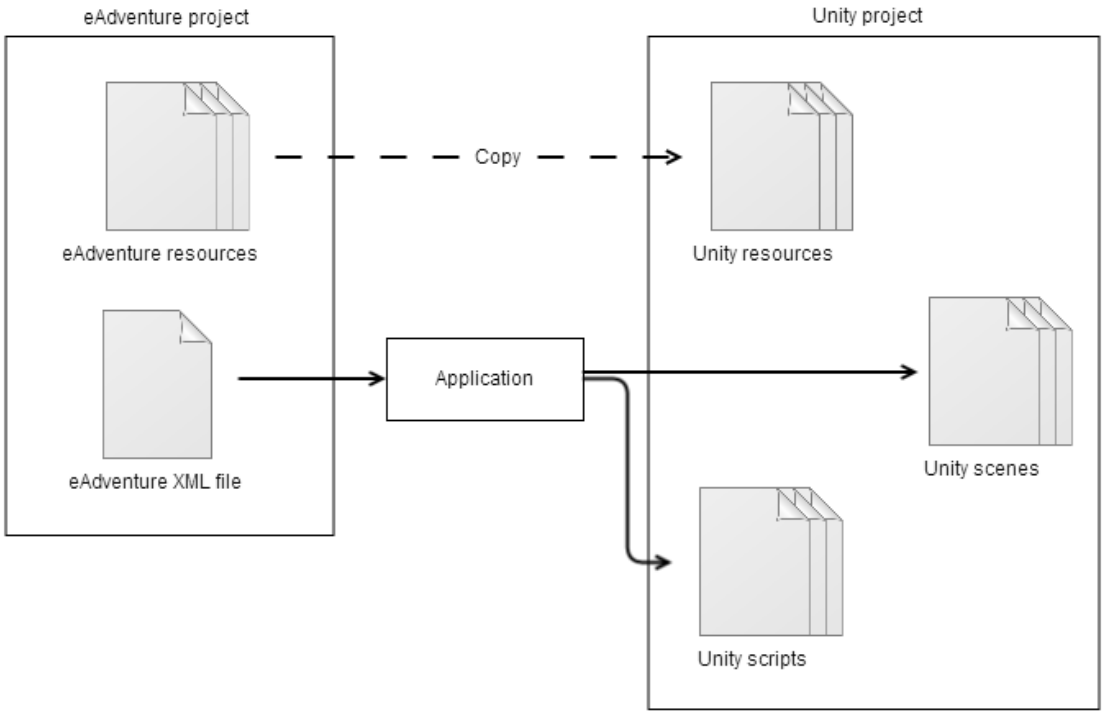

Figure 23 Inputs and outputs of the proposed application

As shown on Figure 23 and said before, application input consists only of eAdventure's XML file location. Application output is a Unity project which consists of generated scenes and scripts, while resource files are simply copied to the destination Unity project folder, without any modification. User is asked to provide the input location, and he can optionally select the destination Unity project folder, if not it will be created in the application root folder.

After creating an eAdventure game, XML file is produced and is ready to be parsed by the application. Next step is creating a model which can be saved in computer memory.

1. Application concept

Important elements of the application that should be highlighted are:

1. Source model – model from the high level eAdventure authoring tool.
2. Model converter – algorithm which converts the source model to the internal model.
3. Internal model – internal representation of the source model as C# classes.
4. Code template library – library of various template scripts which provide game behaviors and functionality.
5. Transformation rules – logic which produces the final target model.
6. Target model – Unity project, the final result of the conversion.

Like it was mentioned before, the required input for the application is the source model, an eAdventure XML file in this case. In order for the application to be able to work with and identify different elements of the model it is necessary to convert it into a model more suitable for an OO language. In this case, using C#, the suitable model is a library of C# classes which represent the internal model. Transformation rules read the internal model and using the code template library they generate the Unity project which is the target model and the final goal of the conversion process. Figure 24 visualizes how different application elements work together through the conversion.

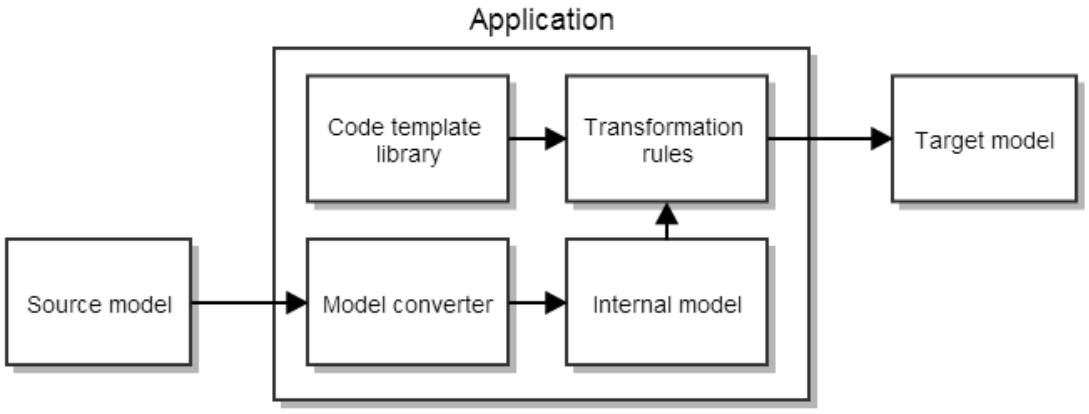

Figure 24 Applied approach - abstract concept of the application

2. Application architecture

Application is divided in three main parts or layers:

- **eAdventure model** – class library representing eAdventure elements (see Creating the internal representation of the source model, page 48)
- **Converter logic** – core of the application, script snippet library and logic which generates the Unity project
- **Graphical user interface** – simple interface to facilitate usage of the application

Figure 25 explains the dependencies between application layers. Application interface only enables user interaction and therefore depends on other layers. Converter logic layer is the application core and depends on the model layer which provides data. Model layer on the hand serves only as data storage and doesn't include any logic, therefore is a standalone layer which has no internal dependencies.

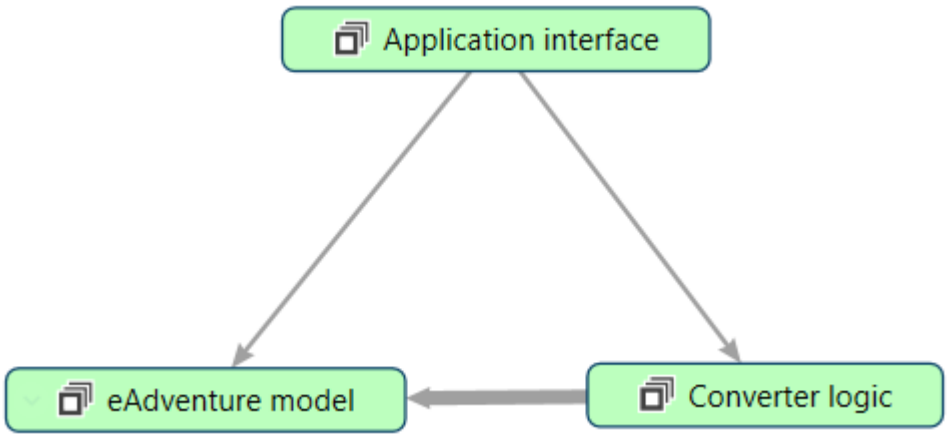

Figure 25 Application architecture

3. Creating the internal representation of the source model

To be able to process even the most complex scenes in eAdventure it is necessary to parse the input XML file and store it in a suitable model in computer memory. The idea is to create a separate class for every sub object in the eAdventure model. Figure 26 shows a partial mapping of the eAdventure model to C# class library. Objects like *Scene, Object, Conversation* and others are main model nodes and contain other sub objects. Main nodes can also be sub objects, either way, every object and sub object has a corresponding C# class.

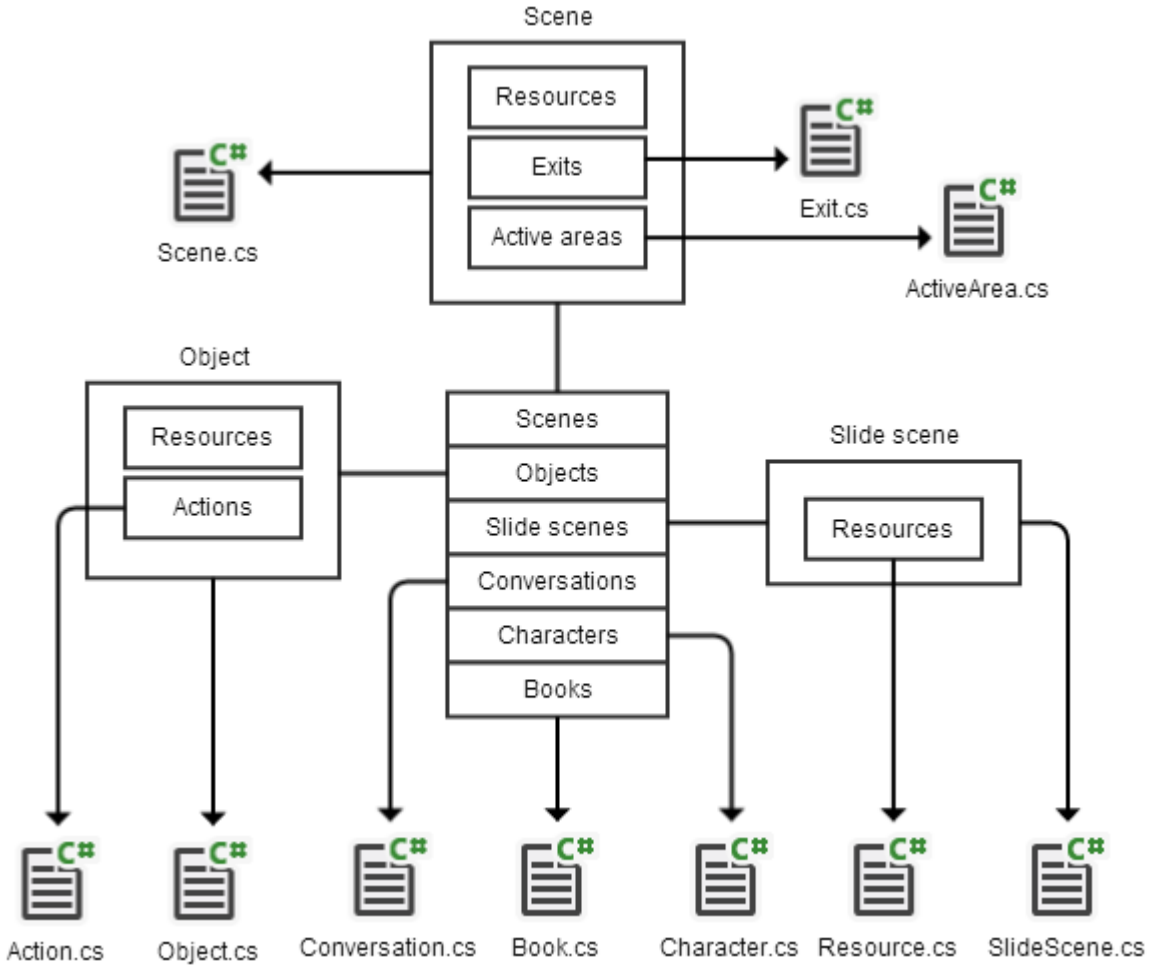

Figure 26 Excerpt from the mapping of eAdventure model to C# classes

Main game elements

Every main game element is uniquely identified with an *id*. Since this is a common property, every class representing a main game element is inherited from the class *BaseElement* which contains a string type property *Id* (see Figure 27).

- ***BaseElement*** – abstract element, from which all main elements are inherited
- ***Book*** – book which a player can read
- ***Character*** – character that is not controllable by the user
- ***Chapter*** – container element, from which every other element is accessible
- ***Conversation*** – conversation between the player and a character
- ***Macro*** – contains a sequence of various actions executed when called
- ***Object*** – an item that the player can interact with
- ***Player*** – player controlled by the user
- ***Scene*** – essential game element, connects all other elements
- ***SlideScene*** – cut scene made of one or more images
- ***VideoScene*** – cut scene that contains a video

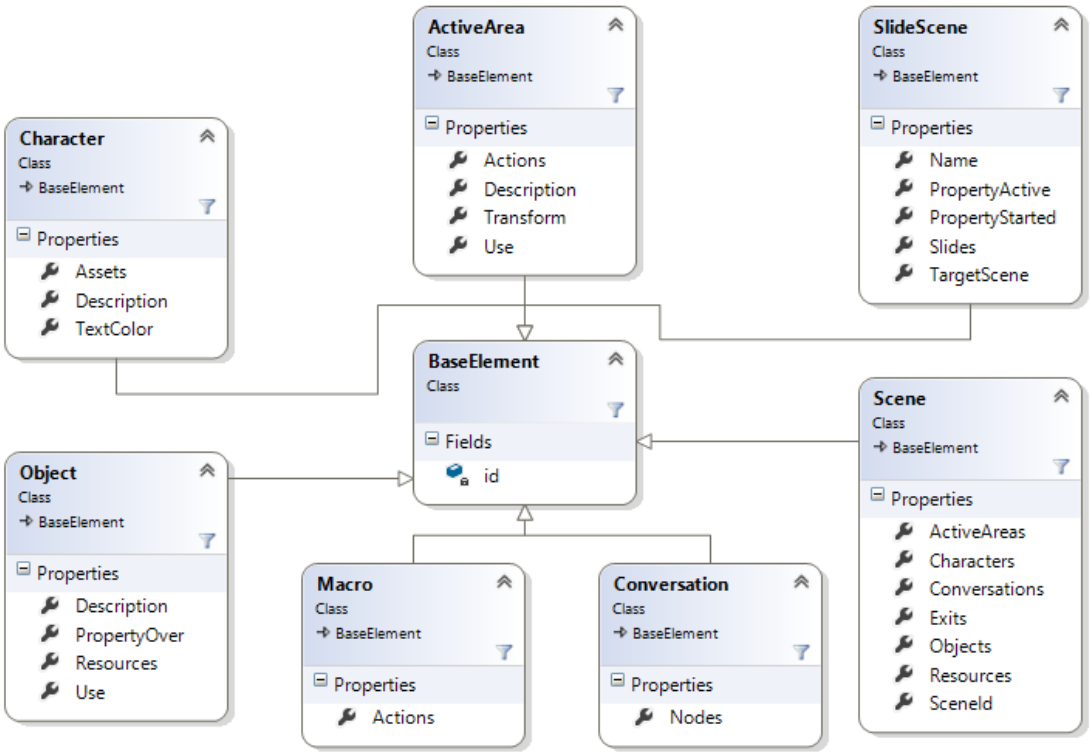

Figure 27 Class diagram of main elements

Common elements

Some elements appear in various places and therefore are classified as common. For example, element *effect* can be called from a conversation, on scene transition or as a result of a *use action*. Other good example of a common object is a *condition*. Conditions can be attached to various kinds of elements. *ResourceList* can have a condition determining whether it is active. An *exit* can have a *condition* determining if exit is enabled or not. Basically, almost any action or statement can be conditioned. For another example of common element usage see Figure 28.

- ***Action*** – action which can be performed on an object or a character.
- ***Condition*** - presents a condition evaluating to a logical expression. Contains active and inactive flags.
    - ***Active*** – flag which has to be active in order to be true.
    - ***Inactive*** – flag which has to be inactive in order to be true.
- ***Description*** – contains the name, brief and detailed description of the element to which is attached.
- ***Effect*** – contains a sequence of actions that get executed when the effect is triggered.
- ***Resource*** – abstract definition of a resource.
    - ***Asset*** – resource type, can be an image or an audio file.
- ***ResourceList*** – presents a collection of resources.
- ***TextColor*** – presents a text style.
- ***Transform*** – contains position coordinates, height and width of an object attached.
- ***Use*** – type of an action which can be performed on an object or a character.

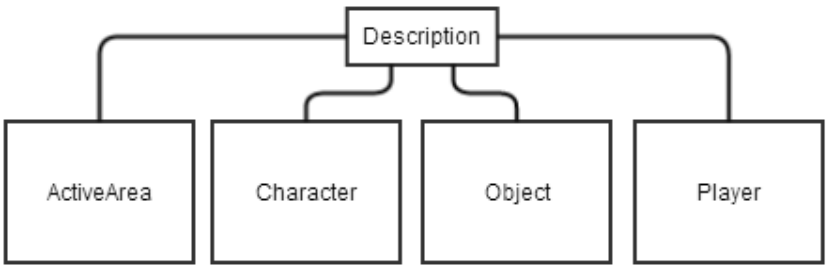

Figure 28 Common element usage example

Conversation elements

Conversation activity consist of one or more nodes. There are two type of nodes, one representing a simple text being spoken either by player or a character and the other type, an option node enabling player to choose one of the options offered. Figure 29 contains a UML diagram of conversation elements and their relationships.

- ***ConversationNode*** – abstract definition of a conversation node.
- ***DialogueNode*** – inherited from *ConversationNode*, represents a sequence in dialogue.
- ***Option*** – represents a single option contained in an option node. Aggregation relationship with *OptionNode.*
- ***OptionNode*** – inherited from *ConversationNode*, contains multiple options.
- ***Speak*** – abstract definition of spoken text. Aggregation relationship with *DialogueNode*.
- ***SpeakChar*** – inherited from abstract *Speak*, represents a character speaking.
- ***SpeakPlayer*** – inherited from abstract *Speak*, represents a player speaking.

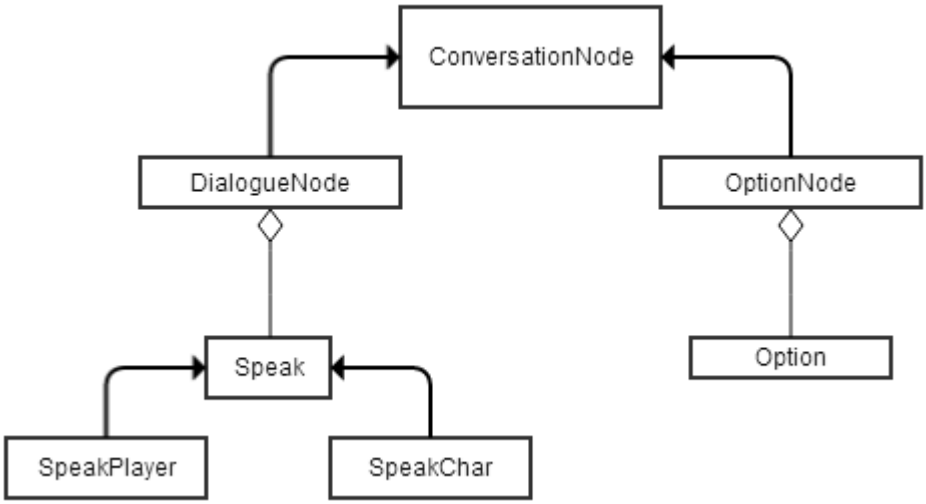

Figure 29 Conversation element UML diagram

Macro elements

Macro element is defined as a sequence of various events. Events can range from setting a global flag to moving to another scene. Every macro element is inherited from a base abstract class *MacroItem* (see Figure 30).

- ***ActivateFlag*** – activates a global flag.
- ***DeactivateFlag*** – deactivates a global flag.
- ***SpeakPlayerMacro*** – represents the player speaking.
- ***TriggerSceneMacro*** – navigates to another game scene.

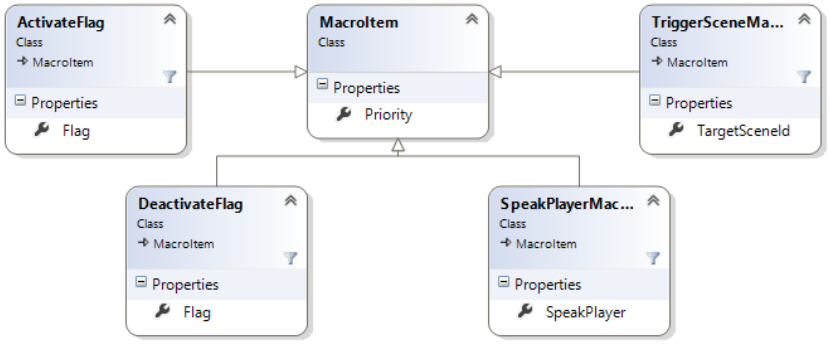

Figure 30 UML diagram of macro item elements

Scene elements

Scene elements are one of the essential building blocks. It acts as a container for all characters, objects and other elements placed on it. Figure 31 explains the scene element structure through a UML diagram.

- ***ActiveArea*** – presents a part of the scene which the user can interact with.
- ***Exit*** – providing navigation away from the current scene. For example from one scene to another.
- ***SceneCharacter*** – defining a character placed on a scene.
- ***SceneObject*** – defining an object place on a scene.

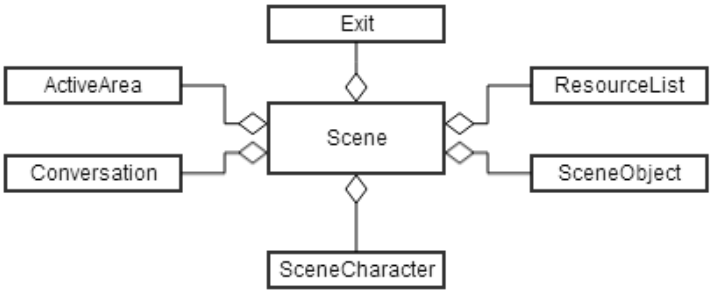

Figure 31 Scene element UML diagram

4. The template library: Unity code snippets

While generating Unity scripts a lot of functionality will repeat itself. To achieve reusability, the idea is to create a library of script templates containing small code snippets which can be slightly modified to implement functionality.

Background

In snippet Code 1, a template for displaying a scene background is shown. The resource image gets loaded into a texture, which is the base for the sprite object. To make it a part of a Unity scene, a game object gets created with the previously created sprite attached to it. Parameters like *id, uri, width* and *height* get set depending on the properties of background image in question.

- **id** – background image identification
- **uri** – path to background image file
- **height** – image height in pixels
- **width** – image width in pixels

```
1. Texture2D txtBg{id} = Resources.Load("{uri}") as Texture2D;
2. Sprite spriteBg{id} = Sprite.Create(
3.     txtBg{id},
4.     new Rect(0,0,{width},{height}), new Vector2(0,0));
5. GameObject bg{id} = new GameObject ("Background{id}");
6. bg{id}.AddComponent ("SpriteRenderer");
7. bg{id}.GetComponent<SpriteRenderer> ().sprite = spriteBg{id};
```

Code 1 Script template for displaying a scene background image

Background Music

Background music or sounds can be dynamically played during the game. Either way, every sound has to be declared. Most common type of sound is the looping background music. Code 2 shows how the *AudioSource* object gets created from the raw audio file (*mp3, wav, ogg*), and afterwards attached to the game object representing the scene background.

- **uri** – path to the file containing background music

```
1. AudioSource audioSource = bg.AddComponent<AudioSource>();
2. audioSource.clip = Resources.Load("{uri}") as AudioClip;
3. bg.AddComponent ("BgMusic");
```

Code 2 Script template for playing background music

Conversation sequence

A conversation consists from a series of connected nodes. The nodes can represent a simple text being said by the player's character or by a NPC. Other type of nodes are option nodes, where the player has multiple choices available. Depending on what the user choses, the conversation will go in that direction. User goes through the conversation by clicking the left mouse button. Code 3 shows a snippet that reacts to the mouse button being pressed. Current conversation node displayed is disabled, and the next one is activated.

- **thisSpeak** – active conversation node
- **nextSpeak** – conversation node to activate on mouse button down

```
1. if ({thisSpeak})
2. {
3.       if(Input.GetMouseButtonDown(0))
4.       {
5.             {nextSpeak} = true;
6.             {thisSpeak} = false;
7.             return;
8.       }
9. }
```

Code 3 Script template for transition between conversation nodes

Description display on mouse over

When the mouse pointer is above an object, snippet Code 4 displays a text with a description of the object positioned relative to the mouse pointer. Text is only shown, while the condition *overObject* is true.

- **description** – text description of the object
- **height** – text box height in pixels

- **overObject** – Boolean property which says if mouse pointer is above the object
- **screenHeight** – height of the game screen in pixels
- **width** – text box width in pixels

```
1. if ({overObject})
2. GUI.TextArea(new Rect(Input.mousePosition.x, {screenHeight} -
   Input.mousePosition.y, {width}, {height}), "{description}");
```

Code 4 Script template which displays a tooltip on mouse over

Next slide scene

Code 5 snippet shows how slide scenes change. Every slide scene consists of one or more slides, images really, and each has a container with corresponding slides loaded. When a slide scene gets activated, first slide (index number 0) in the container gets shown. As the player navigates from one slide to another, first one gets removed from the container and destroyed as a game object, while the second one becomes first in the container and gets activated (shown on the screen).

- **id** – identifies the slide scene

```
1. GameObject slideToDestroy = slides{id} [0];
2. slides{id}.RemoveAt (0);
3. Destroy (slideToDestroy);
4. return slides{id}.Count;
```

Code 5 Script template for transition between slide scenes

Visibility

Code 6 shows an example of a code snippet which implements the visibility of a game object. Modifying the *condition* value, and applying it to the corresponding game object, reusability is successfully achieved.

- **condition** – logical expression which determines the object visibility

```
1. if({condition})
2. this.renderer.enabled = true;
3. else
4. this.renderer.enabled = false;
```

Code 6 Script template for toggling object visibility

5. Transformation rules: Generating Unity scripts from the model

The main conversion results are Unity scripts in C#. They enable user to recreate scenes in Unity, and provide the functionality for the game. There are three types of generated scripts:

- Scene generating script
- State script
- Behavior scripts

Scene generating script

Only one instance of this script type gets generated. Its main responsibility is scene recreation inside of Unity. As shown in Code 7 each method that creates a scene, has an attribute *MenuItem*, which enables it to be called and executed from Unity editor (Figure 32). Other than scene creation it enables behavior scripts to call slide scenes when needed.

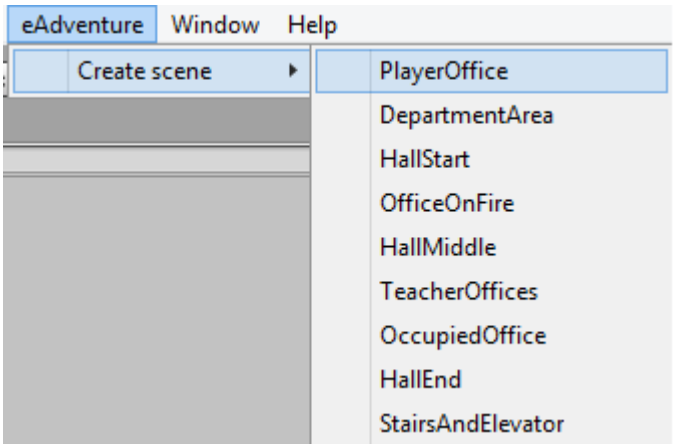

Figure 32 Generated eAdventure menu in Unity

```
1. public class Script : MonoBehaviour
2. {
3.       static GameObject backgroundX1;
4.       static List<GameObject> slidesY1;
5.
6.       [MenuItem ("eAdventure/Create scene/X1")]
7.       private static void CreateSceneX1();
```

```
8. public static void StartSlideSceneY1();
9. public static int NextSlideSceneY1();
10.      }
```

Code 7 Scene generating script example

State script

Like the scene generating script, only one instance of this file type gets generated (see an example Code 8). It contains global state variables, most commonly Boolean type. Since it doesn't have multiple instances, all variables have static behavior. This script (class) provides global state information to all other scripts. If, for example, a behavior type script wants to know the state of variable *PhoneAnswered* it asks the state script, which acts like a central state storage (This example will be described in detail in CHAPTER V).

```
1. public class State
2. {
3.       public static bool PhoneAnswered;
4.       public static bool CheckedOfficeOnFire;
5.       public static bool TriesToWalkByOfficeOnFire;
6.       public static bool AlarmActivated;
7.       public static bool TriesToMoveOnWithoutActivatingAlarm;
8. }
```

Code 8 Example of a generated state script from The Fire Protocol game

Behavior scripts

Behavior scripts, are really classes and they are most important and most complex of all generated items. Each class is inherited from the base class *MonoBehaviour*, and is attached to a game object placed on the scene. Every game object has a corresponding behavior script attached to it. One script can be attached to multiple game objects where each has its own instance of the script, but one game object can only have one script attached to it as visualized on Figure 33.

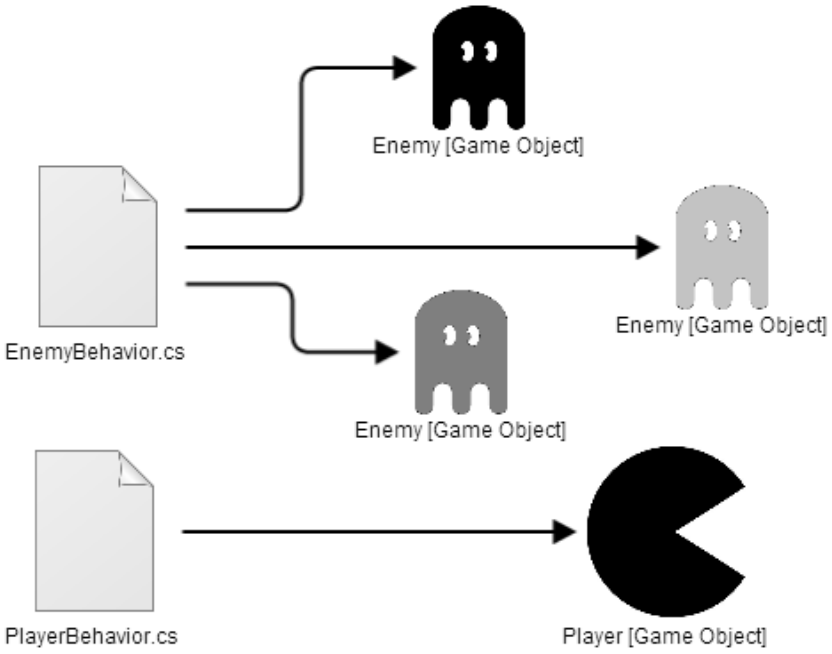

Figure 33 Behavior script to game object relationship

Behavior classes can implement predefined methods that Unity calls during runtime. Most important ones are *Start*, which gets called when the game object connected to the script is created and *Update* which gets called for every new frame drawn (about 60 times a second). There are other special methods that get called when an event is triggered, like on mouse button, or on keyboard press.

6. Conclusions of this Chapter

The basic concept of the approach proved to have a great potential. Translating the eAdventure games to Unity platform is possible. The advantages over manually recreating the game in Unity are huge. While the time invested in fully developing the proposed tool is certainly larger than the time it takes to recreate a game in Unity, once created, the tool will work for any eAdventure game.

The presented applied approach proved to be very time consuming, especially refactoring wise. Unity offers a powerful and an easy to learn framework which included everything needed to replicate eAdventure games. Therefore writing Unity scripts did not pose almost any difficulties, as the designing of the object model which was modeled after the eAdventure game model. The part that proved most difficult was translating the game logic. Since the tool development process was test driven, constant refactoring showed that other development methods could have been a better solution. Even with the difficulties mentioned, the approach is reasonably successful. Given enough time for refactoring and further development, the tool would cover all features which eAdventure offers and would to deliver domain experts knowledge and ideas to game developers.

CHAPTER V. Use cases

In this chapter we are going to demonstrate how the converter works on a few examples. Games created in eAdventure tool will be recreated in Unity maintaining their functionality and design.

1. The Fire Protocol game

To demonstrate the functionality of the developed converter, “The fire protocol game” created in eAdventure will be used as an example (Figure 34). The example doesn’t show all eAdventure capabilities, it demonstrates just the basics. As the title says, game is designed to teach the player how to act and what to do in a case of fire in a building. Game starts with a scene of player’s office, while he receives a call about a fire in the building. Player must take actions to successfully save himself and alert other colleagues in danger. There are different ways of ending the game, depending on the choices user makes.

Storyline

1. Answer the telephone and get information about the fire
2. Check offices and locate the fire
3. Activate fire alarm
4. Alert nearby colleagues
5. Exit the building using safest available way

Figure 34 The Fire Protocol game running in eAdventure

Features used

This game was the first one used while developing the converter tool. Therefore every feature had to be implemented from scratch and was designed so it would satisfy requirements for this game in particular, with an idea to expand functionality later on.

- Background music
- Conditions
- Conversation
- Information tooltip
- Scene transition
- Slide scene
- Use action
- Warning information

Conversion process

Considering game's low complexity and size, the number of generated script files is relatively low. Table 3 shows a list of output files. For better and more general description of generated script types see "Transformation rules: Generating Unity scripts from the model".

Table 3 Unity script files generated as a result of The Fire Protocol game conversion

File name	Type	Attached to
DepartmentAreaBackground0.cs	Behavior script	Asset
HallEndBackground0.cs	Behavior script	Asset
HallMiddleBackground0.cs	Behavior script	Asset
HallStartBackground0.cs	Behavior script	Asset
OccupiedOfficeBackground0.cs	Behavior script	Asset
OfficeOnFireBackground0.cs	Behavior script	Asset
PhoneBehaviour.cs	Behavior script	Object
PhoneRingingBehaviour.cs	Behavior script	Object
PlayerOfficeBackground0.cs	Behavior script	Asset
PlayerOfficeBackground1.cs	Behavior script	Asset
Script.cs	Scene generating script	Not attached
StairsAndElevatorBackground0.cs	Behavior script	Asset
State.cs	Global state container	Not attached
TeacherOfficesBackground0.cs	Behavior script	Asset

Scene sequence

Figure 35 shows how game scenes interchange during gameplay. Game starts with a slide scene which introduces the player with the situation. *Player office* is the first scene from which the player cannot advance without answering the ringing telephone. All of the scenes are shown as ordinary rectangles, while slide scenes are shown as rectangles with rounded corners. Game has a linear story flow and there are three possible ways to finish it, with only one of them being the good way.

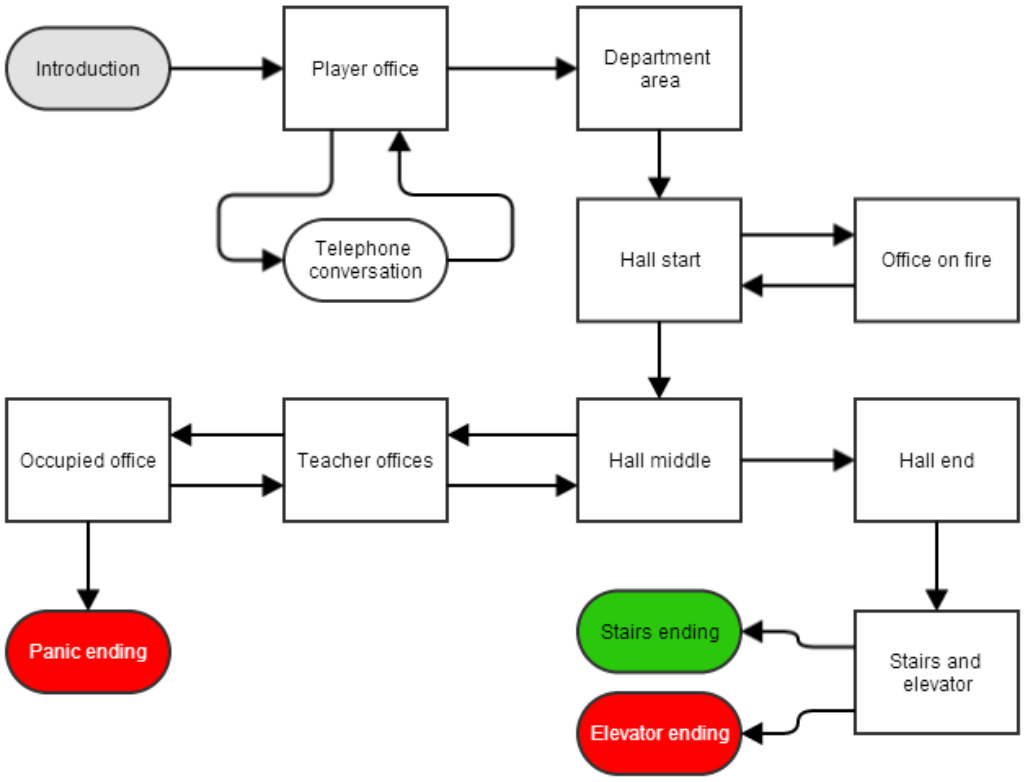

Figure 35 Fire protocol scene flow diagram

Game limitations

This game uses only basic features of eAdventure software, so it was a logical first step while testing the approach. Even so, there were some difficulties and limitations. Original game includes one video, since the free version of Unity doesn't support any kind of video playback, it was not possible to implement this feature without using the professional version which is not free. The video in question is a negligible part of the game and the game is perfectly playable without it. Other difficulties involved triggering scenes and conversations. Those difficulties were successfully handled by changing transformation rules and modifying the internal model, but they could prove harder to solve when dealing with more complex eAdventure games requiring more iterations and work to solve them.

The Fire Protocol game in Unity

```
1. private  void Update()
2. {
3.       if(!State.PhoneAnswered)
4.       {
5.             this.renderer.enabled = true;
6.       }
7.       else
8.       {
9.             this.renderer.enabled = false;
10.            }
11.      }
```

Code 9 Generated visibility script for telephone ringing object

Code 9 shows a snippet of a generated behavior class attached to *phoneRingning* object which represents the phone in ringing state. As the phone has a different appearance while ringing it's necessary to show the game object only when applicable. *PhoneAnswered* is a global variable that knows if the phone has been answered, therefore also if it's still ringing. Update method gets called constantly while the game object to which the class is attached exists. The result of the working script is showing the game object if the phone is not answered and hiding it if it is.

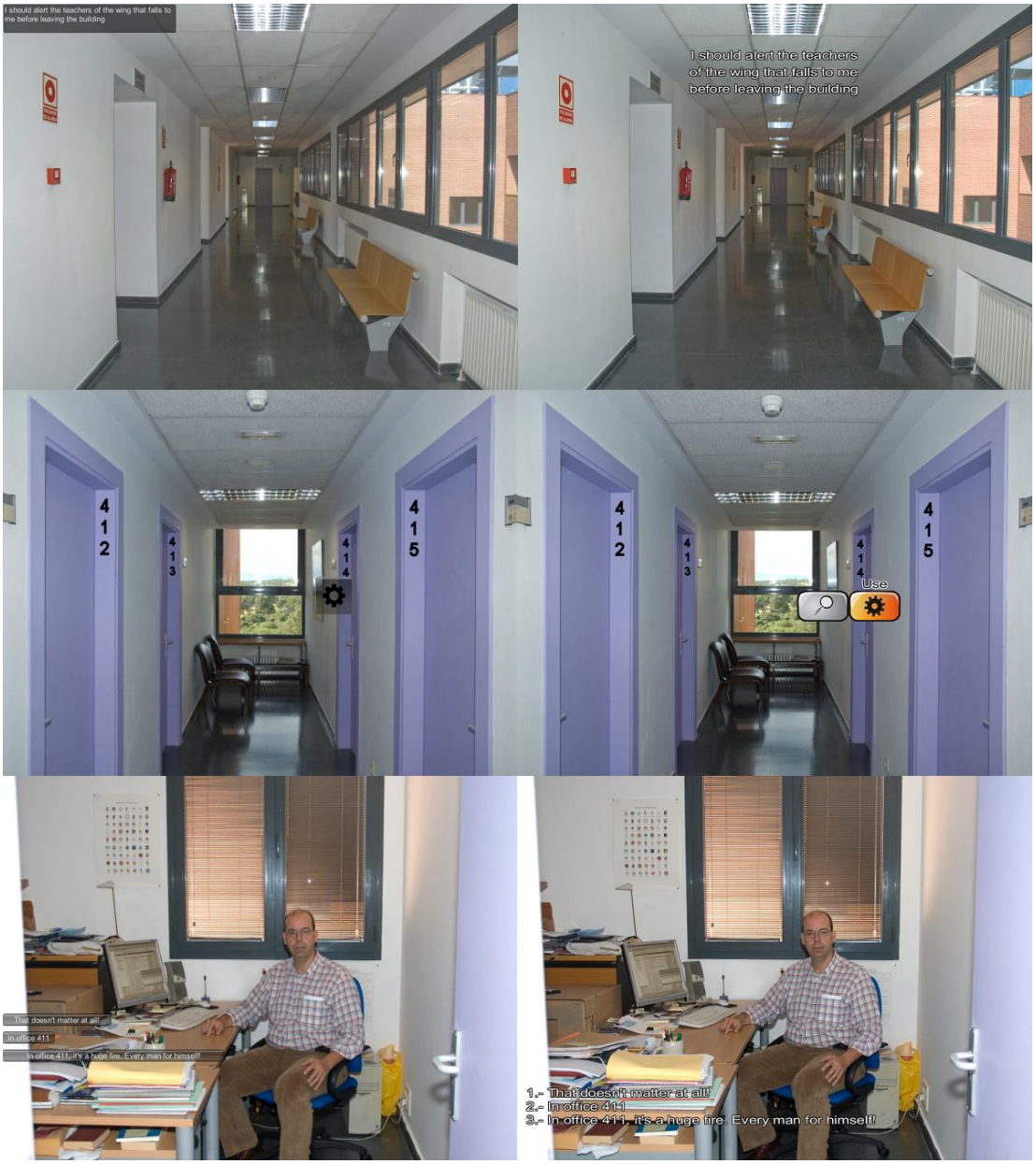

Figure 36 Fire Protocol game comparison, Unity and eAdventure

2. Radioactive Elements

The previous game already existed so it was used to confirm that the approach and the tool worked but to analyze the usefulness of the tool for the creation of a new game, and also due to the time constraints limitations of this thesis, the choice was to create one from scratch. The game uses and expands features implemented in The Fire Protocol game.

Radioactive Elements (see Figure 37) is a game that teaches the player about radioactivity and radioactive elements. The game consists of several quizzes that players need to pass. Before taking the quiz, it's necessary to take a look at the slide scenes which contain the subject material. The game also includes an interactive experiment in which players learn about penetrative characteristics of different types of radioactivity.

Storyline

1. Read materials about radioactivity
2. Take a look at the radioactivity experiment
3. Take the radioactivity quiz
4. Read materials about uranium
5. Take the uranium quiz
6. Read materials about plutonium
7. Take the plutonium quiz
8. Finish the game by exiting the classroom

Figure 37 Radioactive Elements game running in eAdventure

Features used

This game was built with having limitations from the previous game in mind. The goal was to reuse and expand existing features. Although every feature already existed, some refactoring was necessary to successfully translate the game. Features used in this game are:

- Condition
- Conversation
- Information tooltip
- Scene transition
- Slide scene
- Use action
- Warning information

Scene sequence

Figure 38 explains how game scenes interchange during game play. Game starts with the *Classroom* scene which is also the central one. There is only one possible ending, the successful one.

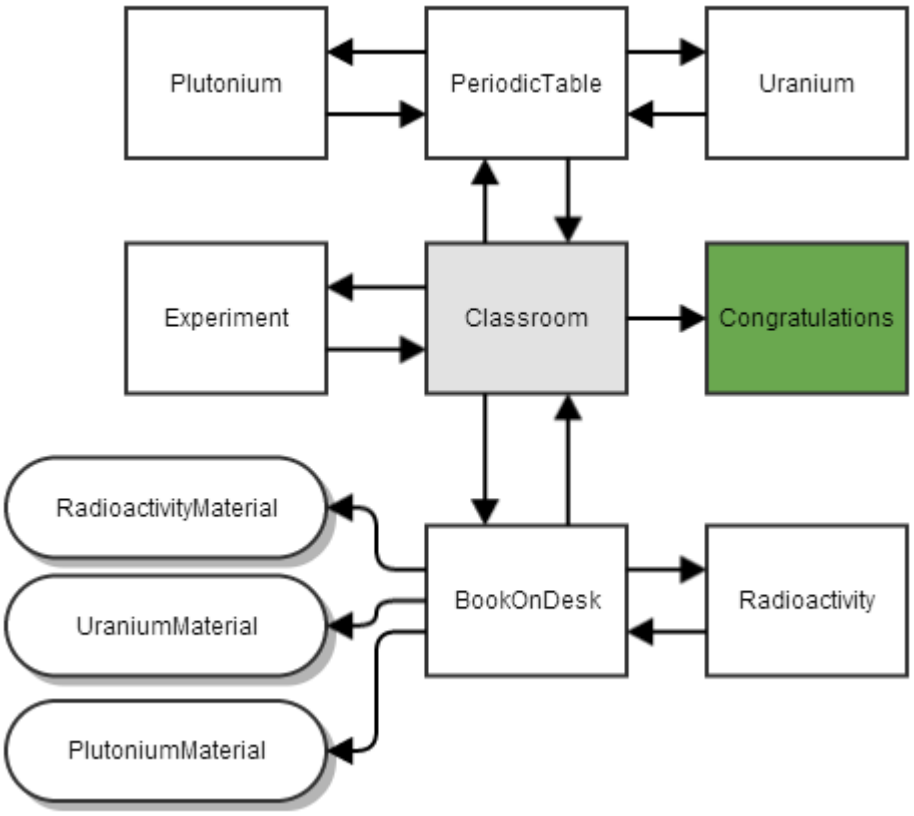

Figure 38 Radioactive elements game, scene flow diagram

Conversion process

As explained in a previous chapter (Transformation rules: Generating Unity scripts from the model), the result of the translation are scripts of different type. All scripts generated as a result of the conversion are presented on Table 4.

Table 4 Unity script files generated as a result of Radioactive Elements game conversion

File name	Type	Attached to
alphaBehaviour.cs	Behavior script	Asset
bookBehaviour.cs	Behavior script	Object
BookOnTheDeskBackground0.cs	Behavior script	Asset
ClassroomBackground0.cs	Behavior script	Asset
CongratulationsBackground0.cs	Behavior script	Asset
detector0Behaviour.cs	Behavior script	Object
detector1Behaviour.cs	Behavior script	Object
detector2Behaviour.cs	Behavior script	Object
detector3Behaviour.cs	Behavior script	Object
detectorActive0Behaviour.cs	Behavior script	Object
detectorActive1Behaviour.cs	Behavior script	Object
detectorActive2Behaviour.cs	Behavior script	Object
electronBehaviour.cs	Behavior script	Object
ExperimentBackground0.cs	Behavior script	Asset
gammaBehaviour.cs	Behavior script	Object
PeriodicTableBackground0.cs	Behavior script	Asset
PlutoniumBackground0.cs	Behavior script	Asset
RadioactivityBackground0.cs	Behavior script	Asset
Script.cs	Scene generating script	Not attached
State.cs	Global state container	Not attached
UraniumBackground0.cs	Behavior script	Asset

Radioactive Elements game in Unity

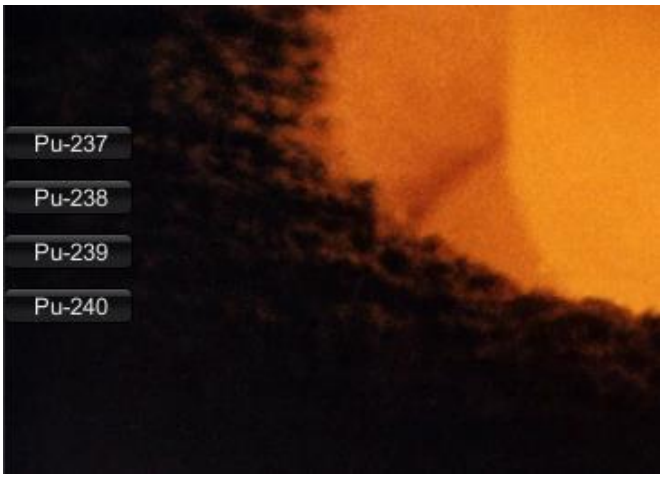

Figure 39 Radioactive Elements game running in Unity3D

Figure 39 shows four options from which the player can choose one to answer the question. Code 10 contains an excerpt from a generated script which is responsible for creating the options. Every option triggers the next node in sequence and turns the current option node off. If you look closely, it is easy to spot that three out of four options lead to the same next node, and therefore the wrong one, since there is only one correct answer. Every node is identified with *SpeakAB123* where last five characters represent a hexadecimal id.

```
1. if (GUI.Button (new Rect (0, {y-axis}, {w}, {h}), "Pu-237"))
2. {
3.       Speak55786 = true;
4.       Option0d50d = false;}
5. if (GUI.Button (new Rect (0, {y-axis}, {w}, {h}), "Pu-238"))
6. {
7.       Speak55786 = true;
8.       Option0d50d = false;}
9. if (GUI.Button (new Rect (0, {y-axis}, {w}, {h}), "Pu-239"))
10.      {
11.            Speakec53f = true;
12.            Option0d50d = false;}
13.      if (GUI.Button (new Rect (0, {y-axis}, {w}, {h}), "Pu-
   240"))
14.      {
15.            Speak55786 = true;
16.            Option0d50d = false;}
```

Code 10 Excerpt from a generated script, result of Radioactive Element game translation

During gameplay in Unity, game behaves the same way as in eAdventure. Figure 40 shows a comparison between gameplay on the two mentioned platforms. The only visible differences are the GUI components like mouse pointers, buttons and text areas. They can also be modified to look more like the original, but since it would only benefit from an aesthetical point of view, they were left at the default Unity settings.

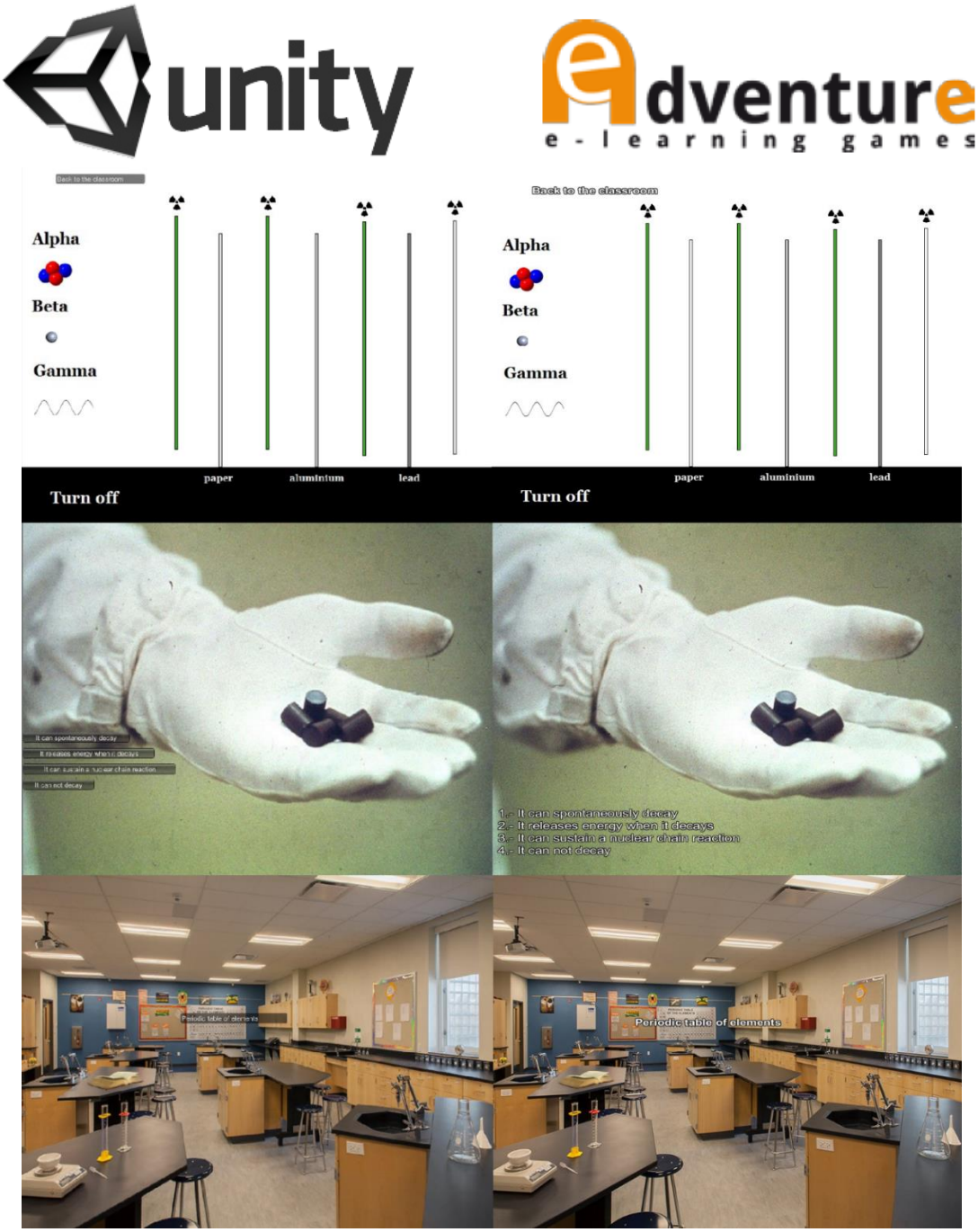

Figure 40 Radioactive Elements game comparison, Unity and eAdventure

3. Conclusions of this Chapter

The development process was test driven and based on the games in this chapter. This approach led to fast results, but required constant refactoring as more features of the game were implemented. After the first game (The Fire Protocol game) was finished, the idea was to handle more complex games. This is when the constant need for refactoring became a real problem. More complex games included a number of new features, but also required existing features to be adjusted which meant a lot of refactoring. Since the time available was not enough to overcome these difficulties, these more complex games were not used to further develop the tool.

The second game Radioactive Elements was built from scratch specially to test this approach. Taking the position of first the domain expert while developing the game in eAdventure and later on the game developer in Unity3D I didn't find any issues that would threaten the goal of this approach, which is improving the collaboration.

CHAPTER VI. Conclusions and future work

This chapter presents a discussion of this work, as well as the contributions of this thesis and proposes options for future work.

1. Summary

This thesis addresses difficulties in the development process of serious games. After exploring current approaches and different technologies available, a new approach is proposed. Proposed approach is based on connecting a high level game authoring tool made for non-programmers to a professional game development platform. The eAdventure platform developed by Complutense University of Madrid was chosen as it provided rather powerful features, while not requiring any programming skills from the user, and it was free for usage. For the professional game development platform, Unity3D, an all-in-one solution was a clear winner. Offering powerful features like its competitors at a considerably lower price and supporting almost all important platforms.

The applied approach focused on developing a tool which connects eAdventure and Unity3D and act as an intermediate between the two. Connection is achieved in one direction by translating an eAdventure game to a Unity project. This approach potentially improves collaboration between domain experts and game developers bypassing the requirement of possessing programming skills.

2. Discussion

Before choosing this approach, an alternative one was considered. Instead of developing a standalone application, the idea was to create a high level wizard-like application on top of Unity editor. Since that meant building an eAdventure like application from scratch the approach was discarded as it would not have fit in the time frame available.

Use cases which were used to demonstrate functionality of the developed tool have shown that this approach has strong benefits in facilitating the collaboration between domain experts and game developers. By translating eAdventure games to Unity3D, the main functionality is kept, but now there are no previous limitations in terms of graphics, features and possibilities which existed since eAdventure is a high level authoring tool, and not a professional game development platform. When the game is inside the Unity environment, game developers can introduce advanced graphics and features, and basically build upon the main concepts that domain experts have made inside eAdventure. The fact that experts don't have programming skills is successfully bypassed using this approach.

Developing the tool (Application of the approach) was rather time consuming, and with existing time limitation only partial features from eAdventure were covered.

3. Contributions

The main contributions of this work are:

- Discovering obstacles and difficulties in the development process of serious games.
- Critical analysis of various game development tools, engines and platforms suitable for developing serious games.
- An abstract approach in serious game development is proposed and explained. Approach focuses primarily on overcoming the difficulties that arise in heterogeneous teams, with the most important one being domain expert's lack of programming and game development skills.
- An applied approach in serious games development. Using eAdventure as a high level game authoring tool designed for non-programmers and Unity3D as a professional game development platform. Connecting the two, domain experts can deliver their ideas to developers more efficiently.
- Test of the applied approach with two use cases. Use cases prove that the approach has a big potential to facilitate collaboration between domain experts and game developers.

4. Future work

The tool that was developed as a result of the applied approach still has a lot of work to be done. Since the time was quite limited, only partial set of eAdventure features are supported. Source code of the designed tool is available online as a GitHub repository[42]. It is free for usage, distribution and further development.

Other than continuing the development in current direction, an interesting idea would be to try achieving the translation in other direction. Translating from the lower level game development platform to the high level game authoring tool can give domain experts feedback on how did game programmers develop their ideas and most importantly if they changed some key aspects of the game. This translation in the other direction is considerably harder to achieve, so it was not included in this approach.

Another interesting idea for future work would be to include the model-driven engineering (MDE) methodology. According to (Schmidt, 2006) MDE technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively. Translation of the eAdventure model to a class model (CHAPTER IV Creating the internal representation of the source model) was done manually, which was rather time consuming, and needed constant refactoring. Using MDE tools for model transformation could lower the amount of work in the development of the transformation tool and further improve the proposed approach.

[42] https://github.com/mateanticevic/sgsp

List of Abbreviations

Abbreviation	Meaning
FPS	First person shooter
GML	Game maker language
GUI	Graphical user interface
HDR	High dynamic range
HTML	Hypertext markup language
IDE	Integrated development environment
JS	JavaScript
JSON	JavaScript object notation
LLVM	Low level virtual machine
MDE	Model-driven engineering
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Pictures Experts Group
RDBMS	Relational database management system
UML	Unified modeling language
WAV	Waveform audio file format
XML	Extensible markup language

References

Abt, C. C. (1970). *Serious games.* Viking Press.

Anderson, C. A., & Bushman, B. J. (2001). Effects of violent video games on aggressive behavior, aggressive cognition, aggressive affect, physiological arousal, and prosocial. *Psychological Science,*, 353–359. Retrieved from http://www.soc.iastate.edu/sapp/videogames1.pdf

Boyle, E., Hainey, T., & Connolly, T. M. (2011). The role of psychology in understanding the impact of computer games. *Entertainment Computing*, p. 6. Retrieved from http://www.sciencedirect.com/science/article/pii/S1875952110000200

Brustein, J. (2013, September 18). *Grand Theft Auto V Is the Most Expensive Game Ever—and It's Almost Obsolete*. Retrieved from BloombergBusinessweek: http://www.businessweek.com/articles/2013-09-18/grand-theft-auto-v-is-the-most-expensive-game-ever-and-it-s-almost-obsolete

Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012, September). A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, p. 26. Retrieved from http://www.sciencedirect.com/science/article/pii/S0360131512000619

Djaouti, D., Alvarez, J., Jessel, J.-P., & Rampnoux, O. (2011). Origins of Serious Games. *Serious Games and Edutainment Applications*, 25-43. Retrieved from http://www.ludoscience.com/files/ressources/origins_of_serious_games.pdf

Ferguson, C. J. (2007, December). The good, the bad and the ugly: a meta-analytic review of positive and negative effects of violent video games. *Psychiatric Quarterly*, pp. 309-316. Retrieved from http://link.springer.com/article/10.1007%2Fs11126-007-9056-9

Higgins, C. (2013, November 7). *SimCityEDU: Gaming in the Classroom.* Retrieved from mental_floss: http://mentalfloss.com/article/53544/simcityedu-gaming-classroom

Hypergrid Business. (2012, August 23). Retrieved from Serious games now $2 to $10 billion industry: http://www.hypergridbusiness.com/2012/08/serious-games-now-a-multi-billion-dollar-industry/

LeCompte, K., Moore, B., & Blevins, B. (2011). *The Impact of iCivics on Students' Core Civic Knowledge.* Waco: Mid-South Educational Research Association. Retrieved from http://www.academia.edu/2569136/LeCompte_Karon_N._Moore_Brandon_L._and_Blevins_Brooke._2011_._The_Impact_of_iCivics_on_Students_Core_Civic_Knowledge._Research_in_the_Schools_18_2_58-74

Marchiori, E. J. (2010). *WEEV: A Multidisciplinary Approach to Educational Game Development.*

Marchiori, E. J., Serrano, Á., del Blanco, Á., Martínez-Ortíz, I., & Fernández-Manjón, B. (2012). *Integrating domain experts in educational game authoring.* Madrid.

Marchiori, E. J., Torrente, J., del Blanco, Á., Moreno-Ger, P., Sancho, P., & Fernández-Manjón, B. (2011). *A narrative metaphor to facilitate educational game authoring.* Madrid.

Moreno Ger, P., & Torrente Vigil, J. (s.f.). *Protocolo de evacuación de la Facultad de Informática de la UCM.* Madrid.

Mozilla. (2014, March 12). *Mozilla and Epic Preview Unreal Engine 4 Running in Firefox*. Retrieved from The Mozilla Blog: https://blog.mozilla.org/blog/2014/03/12/mozilla-and-epic-preview-unreal-engine-4-running-in-firefox/

O'Brien, C. (2010, October 24). *Get ready for the decade of gamification.* Retrieved from Mercury News: http://www.mercurynews.com/ci_16401223

Proctor, M. D., Bauer, M., & Lucario, T. (2007). Helicopter Flight Training Through Serious Aviation Gaming. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 18.

Schmidt, D. C. (2006). Model-Driven Engineering. *Computer*, 25-31. Retrieved from https://www.dre.vanderbilt.edu/~schmidt/GEI.pdf

Szablewski, D. (2011). Impact Game Engine. *Game Developer Magazine*, 31-32. Retrieved from http://impactjs.com/files/GDM_May_2011.pdf

Taiga River. (n.d.). Retrieved from Center for Games & Impact: http://gamesandimpact.org/taiga_river/

Torrente, J., Borro-Escribano, B., Freire, M., del Blanco, Á., Marchiori, E. J., Martínez-Ortiz, I., . . . Fernández-Manjón, B. (2013). *Development of Game-Like Simulations for Procedural Knowledge in Healthcare Education.* Madrid.

Valve. (2011, June 23). *VPhysics*. Retrieved from Valve: https://developer.valvesoftware.com/wiki/VPhysics

Wexler, S., Aldrich, C., Johannigman, J., Oehlert, M., Quinn, C., & van Barneveld, A. (2007). *Immersive learning simulations.* The Learning Guild. Retrieved from http://www.elearningguild.com/research/archives/index.cfm?id=121&action=viewonly

Zyda, M. (2005). From visual simulation to virtual reality to games. *Computer*, 25-32.

Appendix A: Development platform

The chosen language to develop the application is C# in combination with .NET framework, while the chosen developing environment is Microsoft's Visual Studio[43].

C#

C# is a multi-paradigm, object oriented language, developed by Microsoft. Its latest release version is 5.0 and this one will be used in the application development. C# is a more advanced, powerful yet easier to use language version of C++. This programming language has been around since 2002, and currently as the main language of all Microsoft's platforms is widely used.

Visual Studio

When developing with C# and .NET framework it is inevitable to use Microsoft's powerful IDE Visual Studio, currently in version 12.0 (popularly named 2013). Most recent version comes with .NET framework 4.5.1 and is a requirement for developing newest Metro and Window Phone apps.

[43] http://www.visualstudio.com/