ADDING UNITY3D AN AUTHORING LAYER FOR NON-PROGRAMMERS

PIOTR MARSZAŁ

FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de Fin de Grado en el Grado de Ingeniería Informática

June 2016

Directores:

Baltasar Fernández Manjón
Iván Martínez Ortiz

Autorización de Difusión

PIOTR MARSZAŁ

20 June 2016

El/la abajo firmante, matriculado/a en el Grado 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 Grado: “Adding Unity3D an authoring layer for non-programmers”, realizado durante el curso académico 2015-2016 bajo la dirección de Baltasar Fernández Manjón y 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.

Resumen en castellano

[Enter abstract here, no longer than 350 words. Be sure to retain the Section Break below.]

Palabras clave

Summary

This thesis explores how to add o Unity 3D™, a widely used game industry tool, a layer to allow the creation of games for non-programmers. The job was done based on the experience, authoring metaphor, and functionality of eAdventure platform converted as an extension of the Unity 3D™ editor. The key requirement was to maintain the compatibility with projects already created with the eAdventure's editor in order to facilitate the transition between tools for the eAdventure community, this way the tool do not only allows to create new adventures, but also edit products made with older versions of the tool. This work complements the work done by another student that focus on the creation of a runtime environment also based on Unity 3D™ to allow the execution of eAdventure.

These two extensions will become uAdventure, a continuation of eAdventure vision resolving its biggest technical problems due to its dependency on the Java platform. Thanks to the support for multiplatform build provided by the Unity 3D™ (used as a kind of middleware), this new version of the eAdventure will open new opportunities for developers of educational games for mobile devices.

Keywords

Unity 3D, eAdventure, Serious games, Authoring tool

Index

Contenido

Chapter 1 - Introduction

Introduction and motivation

Computer games are quite popular, just speaking about AAA computer games productions (top commercial games) are projects with huge budgets (Destiny - $ 140 million, Star Wars: The Old Republic - $ 200 million, The Witcher 3 - $ 81 million)[1] [2], long production cycle, and a large team responsible for the development of product. Among the commercial game, there is a place for independent productions, created by smaller teams (or single person). Both cases relate to the commercial productions, whose main task is to provide pleasure from the gameplay or story.

Apart from that types of games, significantly smaller parts in video game industry is taken by serious games (also called educational games), usually created at the request of educational units and their main goal is to provide knowledge or improving the students skills as an additional learning tool that has the advantage of engage the students.

A complementary mechanism to foster the students engagement is *gamification*, that is, the usage of mechanisms common for games to enhance the participation in the task which, in other circumstances, can be tedious. The technique is based on the pleasure that comes from overcoming the next achievable challenges, competition and cooperation. Usage of this type of mechanisms is often the only way to encourage the audience to deepen their knowledge in a particular area. Another, equally important, advantage of serious games is the possibility of using them as a simulation game. It is particularly important in the case of medical units or military, where laboratory or battlefield tests are much more expensive (and more risky) than creation of proper application.

However, a key problem associated with all genres of games is their high creation cost. Apart from the necessity to purchase the appropriate tools, it is necessary to pay for specialists who will be able to create game content. While the content of educational games is usually supplied by the originator (who is also a domain expert in the specific field), most of the available tools on the market require technical skills (even programming).

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

[2] http://www.gamespot.com/articles/this-is-how-much-the-witcher-3-cost-to-make/1100-6430409/

Paying professionals who will design or implement mechanisms of the game may exceed the budget available for the project, what, in particular for non-commercial product (which, in most cases, serious games are), can be an insurmountable barrier.

The one of possible solution are tools aiming to facilitate the creation of games. By simplifying the whole process persons who have no previous contact with programming are able to make a game. Typically, the possibilities of such tools are relatively simple and limited, (for instance: to one genre or limited pool of interaction), however thanks to that simplicity users can focus on creation of educational content, bypassing most of the technical problems.

One example of this kind of tools is eAdventure, which is free platform developed by the e-UCM e-learning research group at the Universidad Complutense de Madrid. It allows to create 2D adventure games. The reason why this genre was chosen is associated with relatively low cost of creation this type of game, while maintaining high educational value.

Motivation of this work

Despite numerous advantages, eAdventure is has one huge disadvantage: it does not support building games for mobile platforms, which is a huge limitation at the current commonness of this type of devices. On the other side of the barricade is free (with some restrictions) game engine called Unity 3D™, which allows to build for over 20 platforms. It is currently the most popular game engine, which can significantly affect the influx of new users. Moreover, Unity 3D™ engine editor is extensible.

Together with the project supervisors we decided to create uAdventure, which moves most of the functionality (skipping some aspects, like Assesment and Adaptation profiles) and the rules of operation of the eAdventure engine to the form of Unity engine extension.

uAdventure consists of two aspects: the editor, allowing user to create new and modify existing games and emulator, allowing user to run games in a format appropriate for eAdventure inside Unity engine. The present work describes the authoring tool, data model, and issues connected with import/export of already created games.

The second piece of the project, the emulator, as under the responsibility of Iván José Pérez Colado, master student at the Universidad Complutense de Madrid. Iván also create the base version of the editor for effects and conversations, then this work expanded and embedded

them inside actual authoring platform. The cooperation was mutual, Ivan in his part is using data model and importing component made by me.

The final product, which uAdventure going to be, will require closely integration of our two parts. Like the standalone eAdventure, uAdventure will be continuously developing and improving. We hope uAdventure will become a worthy successor of eAdventure, that preserves the community gathered around the engine and allow to gain a new audience.

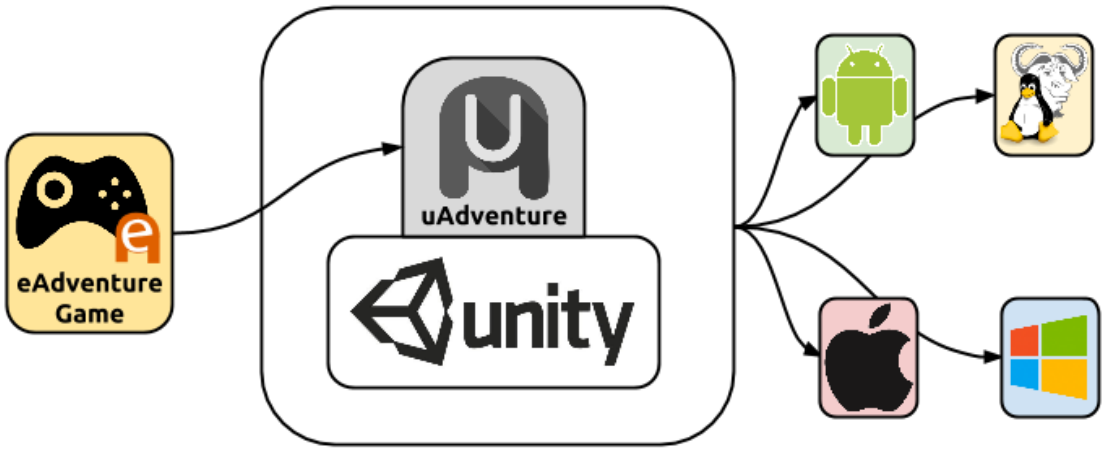

Figure 1-1 Concept of *uAdventure*, author: Ivan Perez-Colado

Structure of this document

The rest of this document is structured as follows:

- Chapter 2 - provides a brief introduction of the relevant authoring and creation tool for both general video games and serious games.
- Chapter 3 - describes the game model and the actual implementation of the authoring layer created on top of Unity 3D™.
- Chapter 4 - provides some conclusions and describes some future lines of work.
- Chapter 5 - Provides the required conclusions and future work in Spanish.

Chapter 2 - State of the art

eAdventure

eAdventure is a "platform aimed to facilitate the integration of educational games and game-like simulations in educational processes in general and Virtual Learning Environments (VLE) in particular"[3], developed by the e-UCM e-learning research group at Universidad Complutense de Madrid.

The main ideas behind eAdventure are:

- Facilitation of game creation for not game developers.
- Time and cost reduction needed to creation specified type of game.
- Incorporation of education-specific features in game development tools.
- Provide a graphical editor that allows to create a serious game without writing a single line of code.

eAdventure is used to creating educational adventure point &click games, built in 2D environment, from first-view perspective, where the actual player is not represented in the game, and a third-view perspective where the player is represented using an avatar. eAdventure has been developed for more than 10 years. During that time, the community of users steadily increased, in particular, more than 60000 of users downloaded eAdventure platform.

eAdventure is written in Java language. Whole game is described in .xml files, which are not platform-dependent. Nowadays, biggest disadvantage (connected with used technology) is a lack of support to building to different target platform, especially for mobile devices.

The large penetration of smartphones and tablets gives a new possibilities for usage of serious games not only in education, but also in medicine or science. The evolution of eAdventure to support multiple platforms should not only keep the current users but also attract new ones.

[3] e-adventure.e-ucm.es

Unity 3D™

Unity 3D™, or just Unity, is the most popular game engine in world, adapted to both three and two dimensional products. The versatility of Unity is its distinctive feature.

Unity has two types of license: free and professional, which is paid. The difference between them relate mainly to the availability of optional, more advanced functionality. Companies whose income is greater than $ 100,000 per year are required to purchase the paid version.

Unity is a multiplatform engine - in both license versions, it is possible to build a game on the more than 20 platforms, including:

- Desktop: Windows, Linux and MacOS X
- Mobile devices with iOS, Android, Windows Phone 8
- Windows Store Apps
- WebGL (for browser based games)

Multitude of options available with relatively low costs of transferring the game to other target platform is undoubtedly a plus of this engine.

Community around the Unity3D is one of the largest in the world. Thanks to the activity of users on the official and unofficial forum in most cases it is possible to find solution to the problem which you currently have to face. This fact makes such complicated tool as game engine affordable not only for professionals but also for laymen.

Another advantage of the engine is its official store Unity Asset Store. There are different types of resources, such as 3D models, animations, sounds and scripts. For relatively small money anyone can buy useful products, even find free assets.

Unquestionable advantages of the engine makes Unity currently the most popular game engine in the world. According to data from the manufacturer webpage share of the global market is about 45 percent. 47 percent of game developers are using the Unity, and for 29 percent it is the main working tool. The number of registered users is more than four million including both professionals and independent developers. Results of their work reached about 600 million players.

Other tools

Unity is not the only one available commercial game engine. There are products of varying complexity, with different target audience and purpose. Each of them has advantages and disadvantages, which may be decisive during the choice of tools to create the game. Each of them has different restrictions, requirements, offers various possibilities. The rest of this section provides a brief discussion of the more relevant tools for this work.

Gamemaker

Figure 2-1 Gamemaker interface [4]

One of the most popular environment for creating and designing two-dimensional games. Uses drag and drop interface, allowing to make a game without writing a single line of code. Can be used for both prototyping and creation of appropriate games. Provided scripting language GML (Game Maker Language) may expand the possibilities available through the drag and drop interface. It allows user to build games for iOS, Android, Windows Phone, Tizen, UWP, HTML5, Windows, Mac, Linux, Xbox One, PS 4, PS Vita and PS 3.Free version of Game Maker is available, but it allows to export only Window desktop games. Exportation to other platforms is possible after buying the Professional version and the appropriate exporter.

[4] http://i2.wp.com/www.ogrejungle.com/wp-content/uploads/2013/05/GameMaker-Studio_Native-Physics-support-from-Box2D.png

Advantages:

- drag and drop interface - allows user to create games without writing single line of code
- GML - scripting language that expand the capabilities of the engine
- exporting games to multiple platforms
- low system requirements

Disadvantages:

- exporters for games to platform other than the Windows desktop are paid

Construct 2

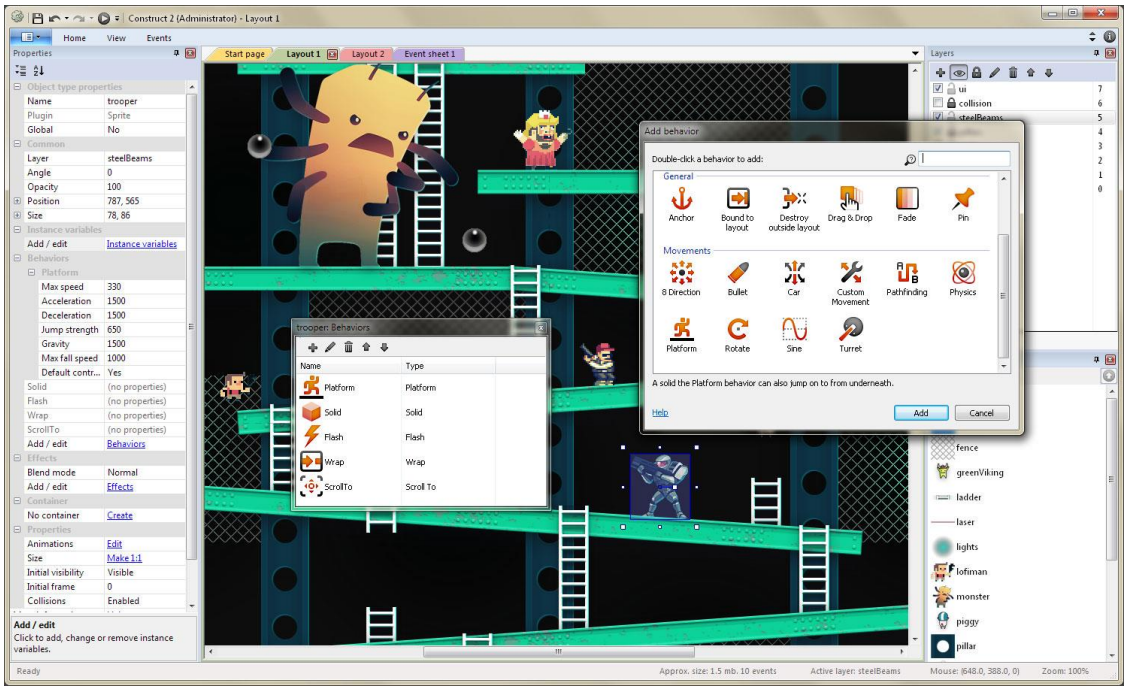

Figure 2-2 Construct 2 interface[5]

Another engine designed to create 2D games using drag and drop interface. It produces HTML5 based game, which can be run in desktop and (some of) mobile devices browsers. It allows

[5] https://static1.scirra.net/images/fresh/c2/gallery/fullsize/jpg/behaviors-panels-01.jpg

to build standalone games for PC, Mac and Linux. The free version of the engine allows only to build desktop browser games - support for other available types of builds is included in Personal/Business license.

Advantages:

- easy to use
- extendible (Javascript SDK)

Disadvantages:

- performance of game is not so well
- standalone games run in wrapper

Scratch

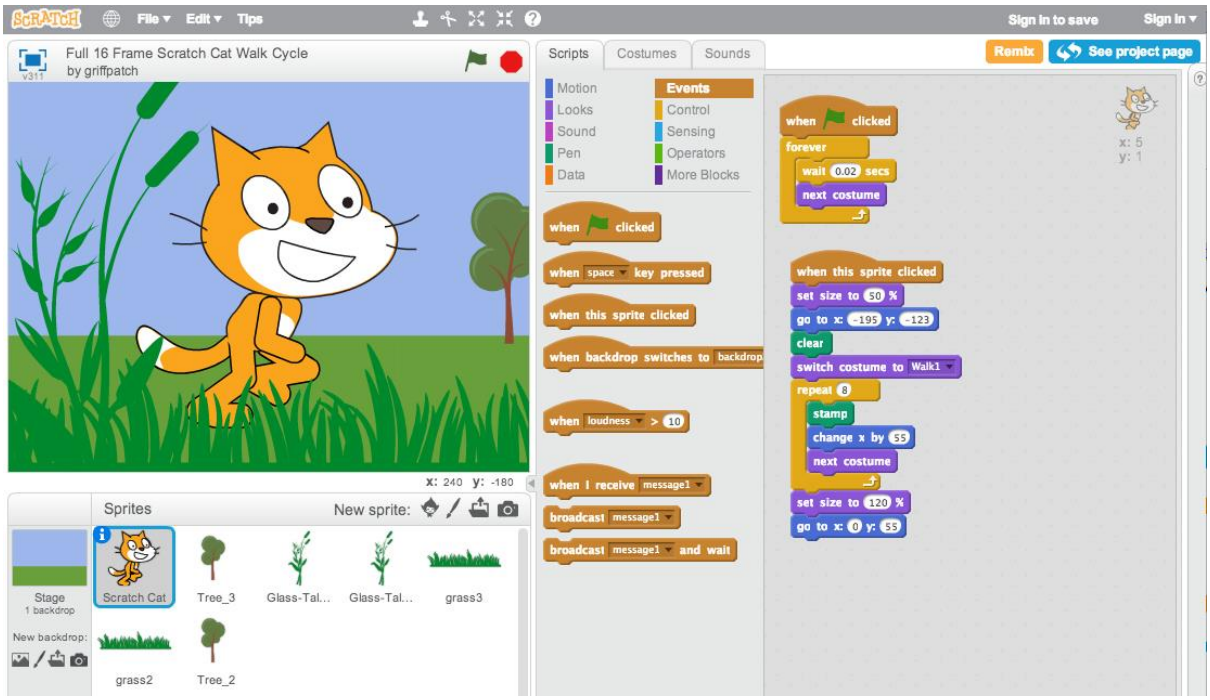

Figure 2-3 Scratch interface[6]

[6] http://news.mit.edu/sites/mit.edu.newsoffice/files/images/2013/20130514110054-1_0_0.jpg

Scratch is a free visual programming language developed by Massachusetts Institute of Technology created mainly for teaching children basics of programming. It is also used by students or teachers to easily create simple games and animations.

Advantages:

- easy to use
- free

Disadvantages:

- very limited
- only simple games can be created

Libgdx

Desktop / Android / Blackberry / iOS / HTML5 Java game development framework. In comparison to the previously described tools, this is a framework, not an engine. It has no editor, whole game is created from the source code. Provides a unified API that works across all supported platforms. Apart from the wrapper on a low level operations, it consist of different components - mathematical, physic, etc.

Advantages:

- completely free
- multiplatform
- unified API

Disadvantages:

- programming is necessary to create a game

Unreal Engine

One of the most complex and polished game engine. It is a very powerful tool, used widely in large commercial productions. It allows user to create games in two- and three-dimension and build them to iOS, Android, HTML5, Windows, Mac, Linux, Xbox One, PS 4 PS 3. It is free, but

Epic Games charges a 5% royalty based on gross revenue. It requires from user high technical knowledge. C++ programming language is used. There is a Blueprint visual scripting system, often used during prototyping. The source code of engine is open - thanks to that, professional studies can freely modify all aspects of the tool, from editor UI to rendering or AI system.

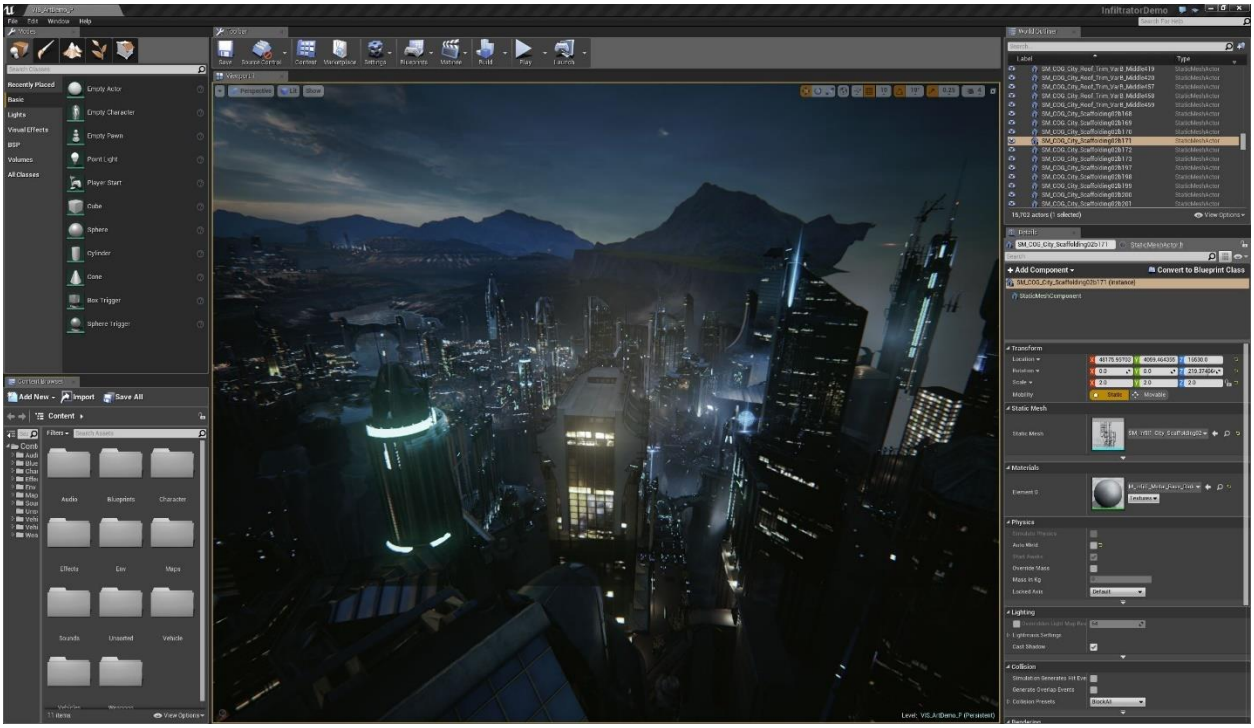

Figure 2-4 Unreal Engine 4 interface[7]

Advantages:

- open-source code
- Blueprint visual scripting system
- one of the most powerful game engines, commonly used in AAA productions

Disadvantages:

- charges royalty
- high degree of complexity
- high system requirements

[7] https://docs.unrealengine.com/latest/images/Engine/UI/LevelEditor/UE4Interface_5.jpg

Based on this review and the previous experience of the e-UCM team, we choose Unity 3D™ as the base platform to create an authoring tool that mimics the current functionality of the eAdventure editor. Next chapter describes the architecture, main features and the journey to create the authoring component of the new uAdventure platform.

Chapter 3 - Implementation

Data model

The main feature behind creating extension is the ability to import projects created in standalone eAdventure. To minimize compatibility issues, I decided to create a data model corresponding to the data model used in the original eAdventure.

The first adapted the idea is to use same interfaces as was used in the Java version of engine: Described, Detailed, Documented, HasDescriptionSound, HasSound, HasTargetId, Named, Positioned, Titled. Each of the models implements the appropriate type of interface.

The class hierarchy is similar to the hierarchy in eAdventure. Thus: each of the Effect types inherits from AbstractEffect. Class Element is the base for classes representing Atrezzo, Barrier, Item, NPC, Player, ActiveArea, etc. Selected fragments of class diagram are presented below:

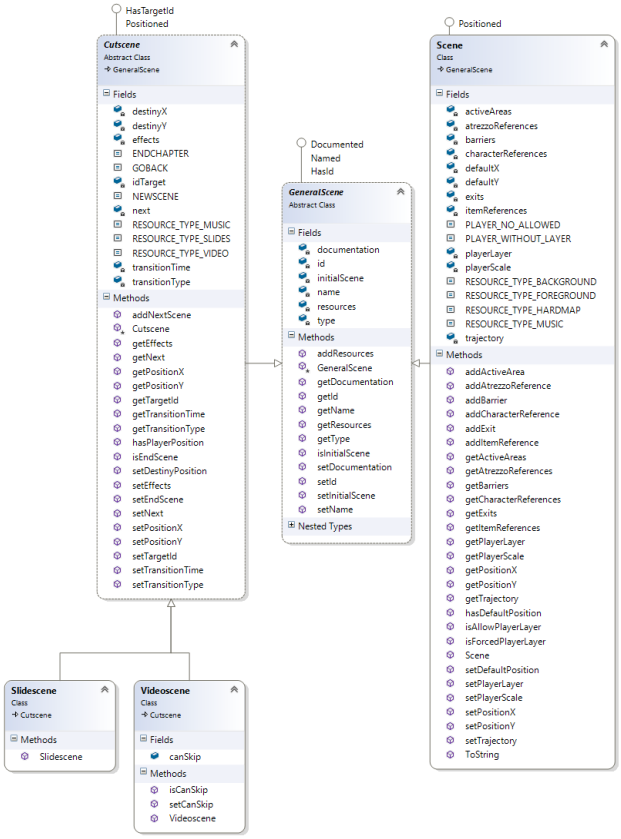

Figure 3-1 Scenes class diagram

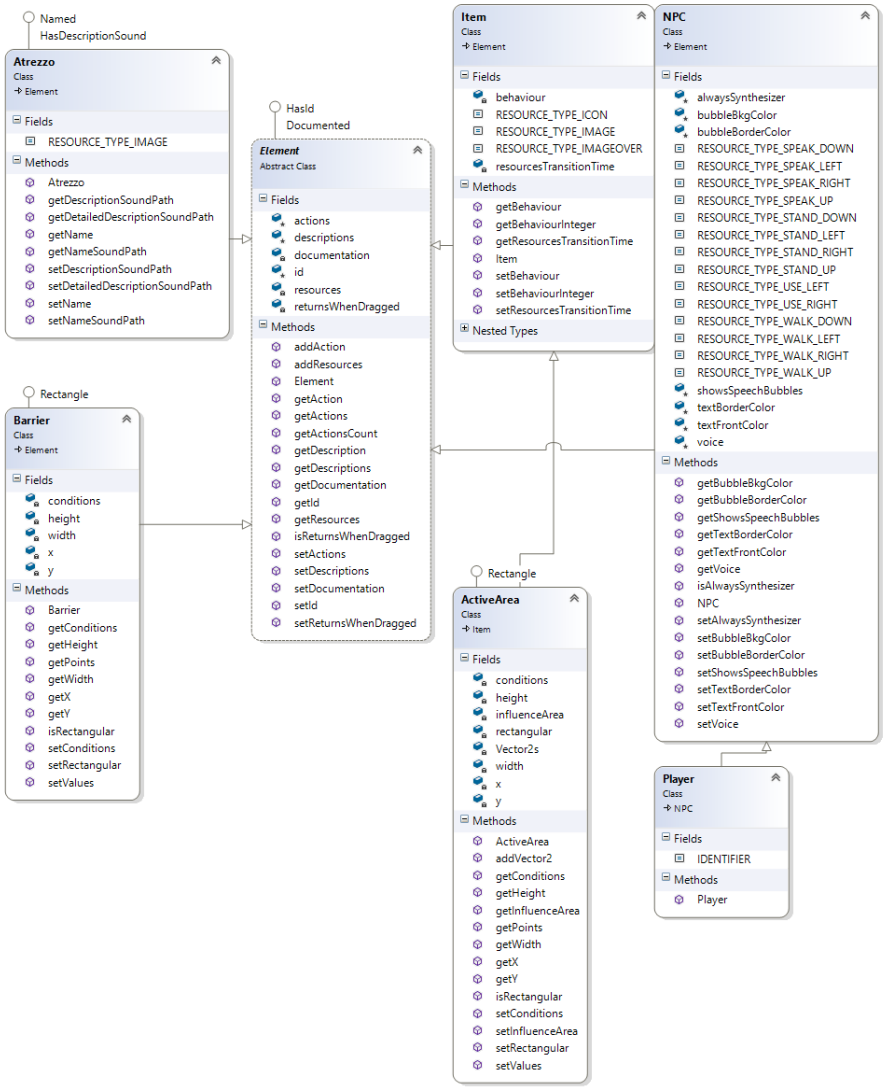

Figure 3-2 Elements class diagram

Parser/Importer

The usage of data model which is almost the same as data model representation of the eAdventure makes much easier the problem of importing projects created in the standalone version of the eAdventure. Imported is unpacked content of the .ead file, i.e. .eap project file and folder with the same name containing assets, .xml files and .dtd files which describe the whole adventure.

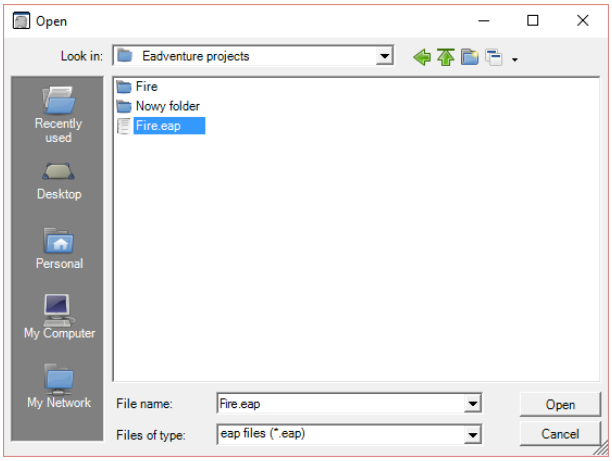

Figure 3-3 Open project file selector

Importing project into a Unity is synonymous with parsing .xml files, filling the appropriate data models and copy adventure assets to the appropriate folder Assets / Resources. The last operation enables to use the functionality of Unity associated with loading and casting to the specific assets engine format.

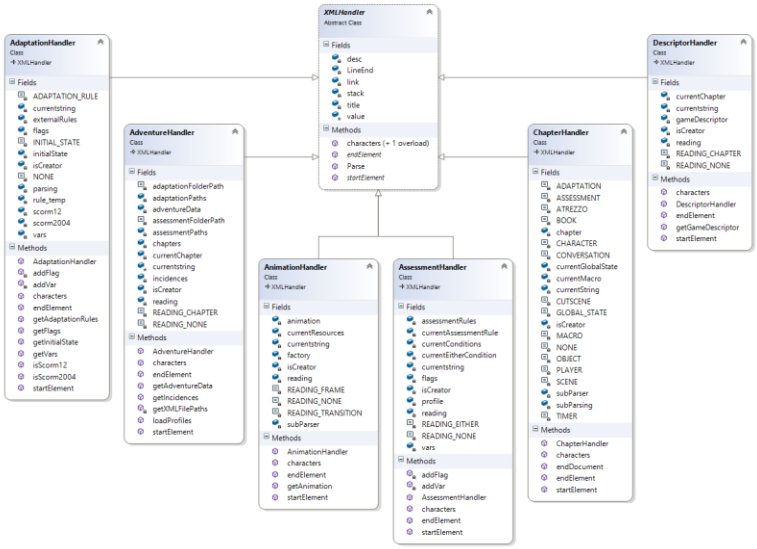

Figure 3-4 Import handlers class diagram

Parsing is based on handlers for the most external aspects of adventure. Used handlers: AnimationHandler, ChapterHandler, DescriptorHandler. For parsing concrete aspects of the game I used SubParsers from the level of handlers. All subparsers inherit from SubParsers class and are

responsible for parsing eg. books (BookSubParser), effects (EffectSubParser) or NPC (CharacterSubParser).

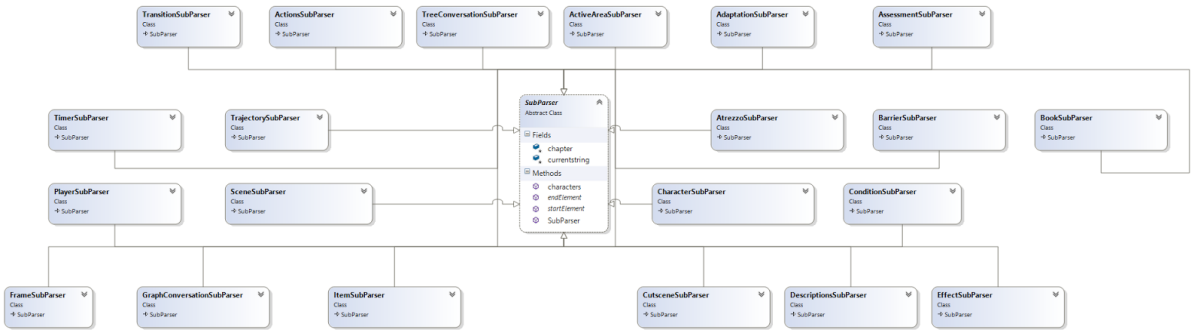

Figure 3-5 Subparsers class diagram

I made two approach to subparsers: in first, parsing files of the project was carried out in a similar way to parsing by the original eAdventure engine. However, since C # and Java use a different parsing approach, files operations and methods to parse .xml files, I had to write some kind of functions wrapper using C # language to the methods used in the original engine source code (in which SAX is used). However, this approach was not satisfying in terms of time needed for operation.

Second approach is based in pure C# XML parsing style, based on System.Xml XML structure predicting and storing it into objects of appropriate type (i.e. in ItemSubParser I try to select every resource, description, action and effect nodes from root item node, and for each of them iterate through all collection and set parsed objects or attributes values).

It turns out that second approach is more efficient - importing same game takes several time less in comparison to first Java-wrapper style solution (in most cases - about 5 times faster).

Exporter

Exporting game task is splitted, as in the case of the importer, into smaller sub-tasks. I have created a superior class, Writer, which is responsible for dividing export tasks to minor tasks (for example, concerning export of single scene, animation, conversation, etc.) and delegating them to the helper classes. Exports, except to creating appropriate .xml and .dtd files describing the game in a manner analogous to the standalone version of eAdventure (each chapter corresponds to a

separate .xml file, etc.), is also equivalent to saving changes in the project and copying related assets, maintaining proper file structure and creating .eap reference file. Target directory of exportation is Games folder in the Unity project root directory.

New game

The new game creation window is similar to the window in the standalone version of eAdventure. User can choose the type (perspective) of the game (first or third person). After that, the user is prompted to select a directory for new game project. I decided to use standard C # save file dialog from the System.Windows.Forms namespace. Unity does not have native support for it - I had to use System.Windows.Forms.dll file as Unity Engine plugin. Every native-platform code plugins have to be put inside the "Plugins" folder inside "Assets" directory.

Figure 3-6 Welcome window

User interface (windows system)

Potentially the largest part of recipient of created extension are existing users of eAdventure, so I decided that the user interface should look almost the same as the original version - current user should not feel lost and should had the feeling of communing with almost the same product.

I divided interface into two main parts: the part of the welcome screen (where user can create new adventure, import .ead file) and appropriate editor window.

Figure 3-7 New FPS game window

The main problem during creating a user interface of Unity editor extension is necessity of using the old interface tools (the new one, introduced in version 4.6, is based on canvas). For this reason, the creation of the interface is significantly more time consuming. The old system does not support most of the necessary controls, like combobox, dialog boxes, menu items, so I had to implement it by myself using a limited pool of available controllers.

Classes corresponding to the aforementioned windows: WelcomeWindow and EditorWindowBase inherit from EditorWindow and act as containers for specific windows that are embedded in certain places inside these containers. Each separately view corresponds to a separate class, each of them inherit from LayoutWindow, which inherits from BaseWindow. All the classes associated with the concrete view are responsible for the distribution of its interface elements and event handling.

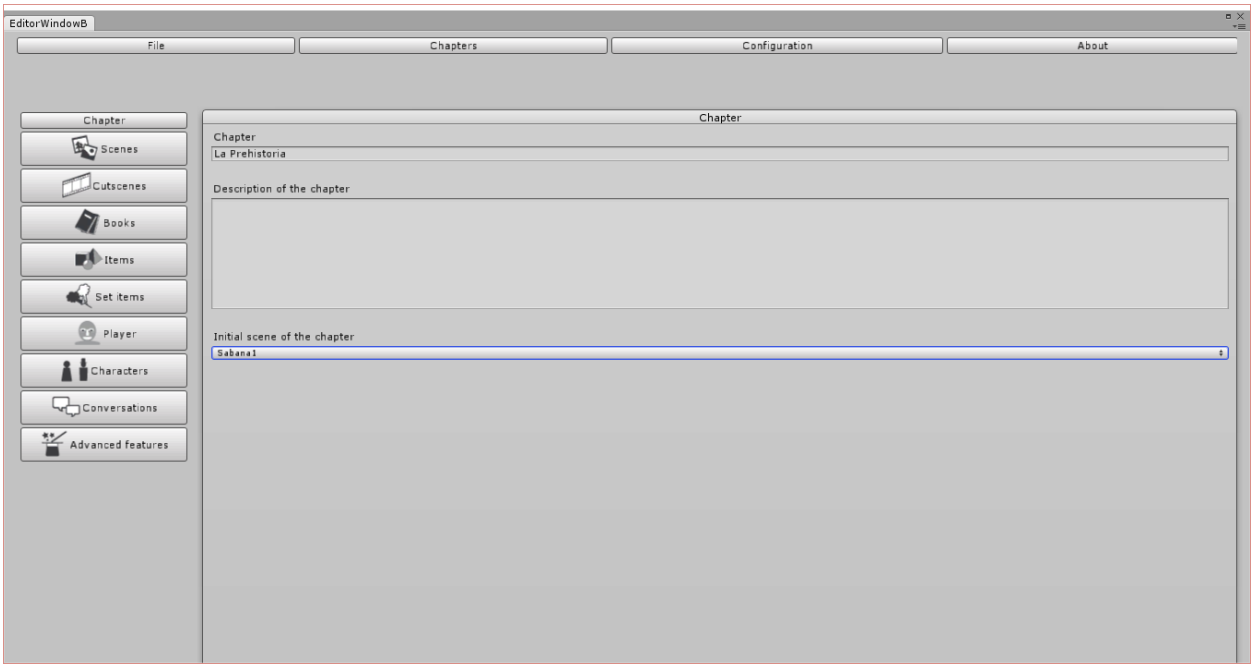

Figure 3-8 Overview of editor window

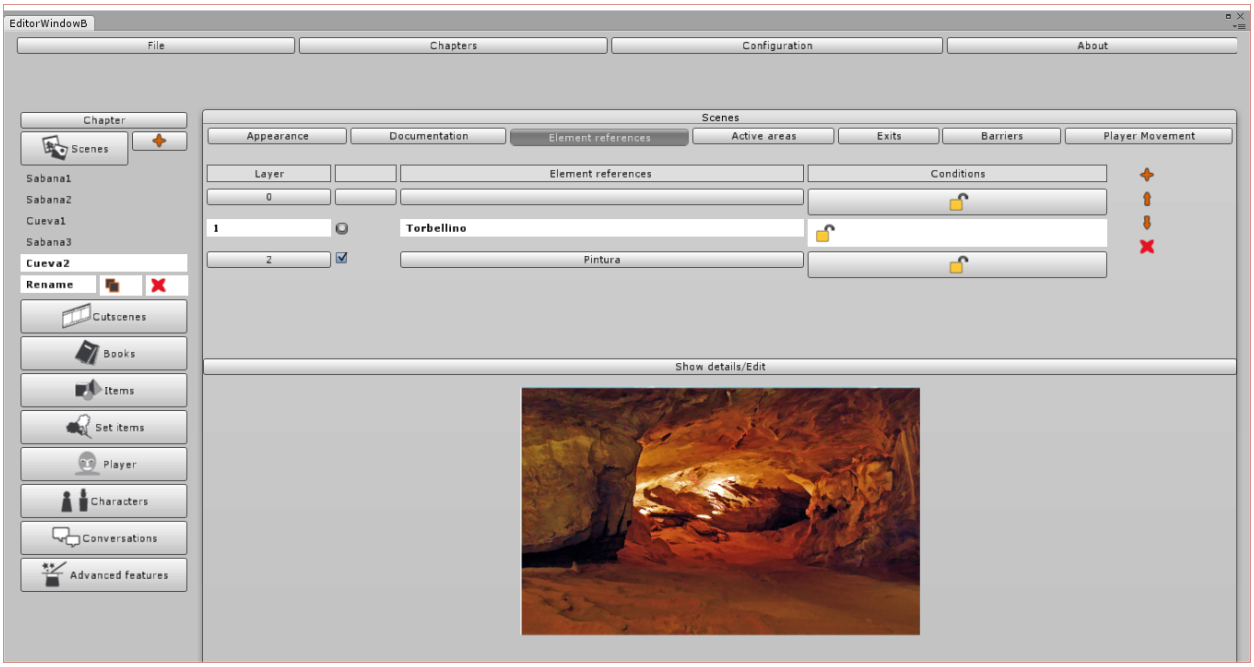

Figure 3-9 Overview of editor window

User interface (dialogs, prompts)

On top of windows system described in the previous section I created a dialogs/popups system. Each type of popup inherits from EditorWindow class. For communication between the

dialogs and windows creating them I created DialogReceiverInterface interface - each window which is instantiating a dialog implements the mentioned interface, whereas each dialogue has a reference to the object that initiated the creation (through interface). Reference is set in the initialization method of dialogue. The interface has two methods: OnDialogOk and OnDialogCanceled, called by dialogs objects when user press the appropriate buttons. Methods take parameters passed to the window classes which initiated the creation of popup (usually it is a string message object). The system was designed in accordance with the object-oriented programming principles - the code is reusable, base classes are not changed, only, where it is necessary, extended. I decided to separate following base types of dialogs:

- BaseAreaEditablePopup - used to edit objects properties relative to the background. It allows to change position (and, where it is possible, dimensions and/or scale) exits, active areas, barriers, element references, player movement of scene and arrows of books.

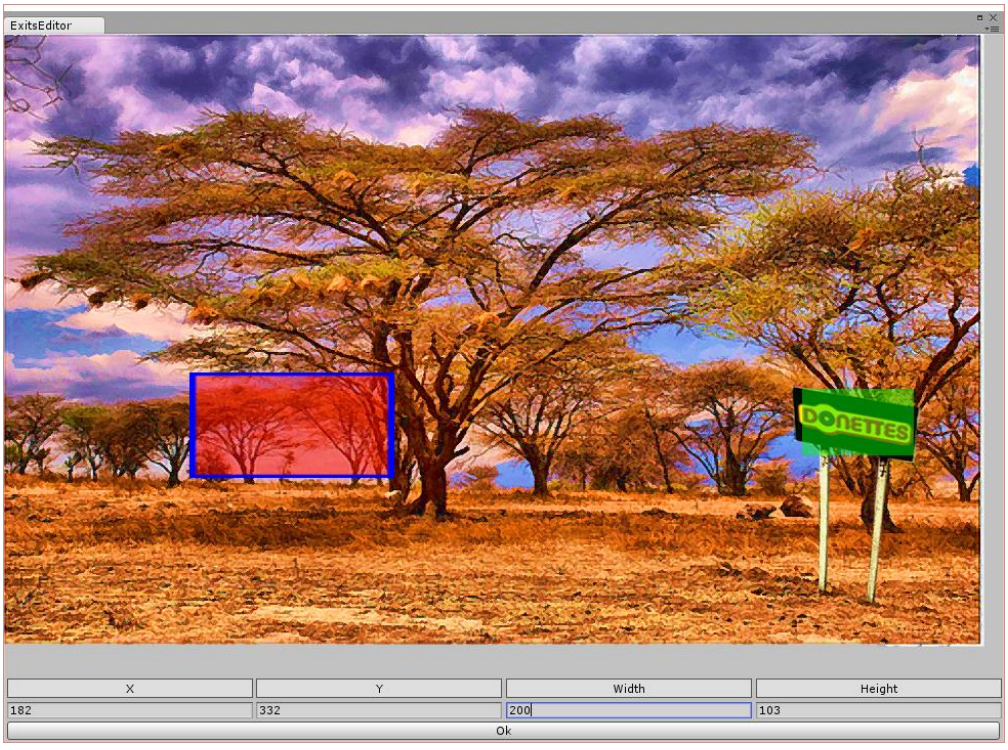

Figure 3-10 Example of *BaseAreaEditablePopup*

- BaseChooseObjectPopup - used to select one of the elements of the collection, ie. scene selection associated with exit and item/atrezzo/NPC reference selection during adding scene elements

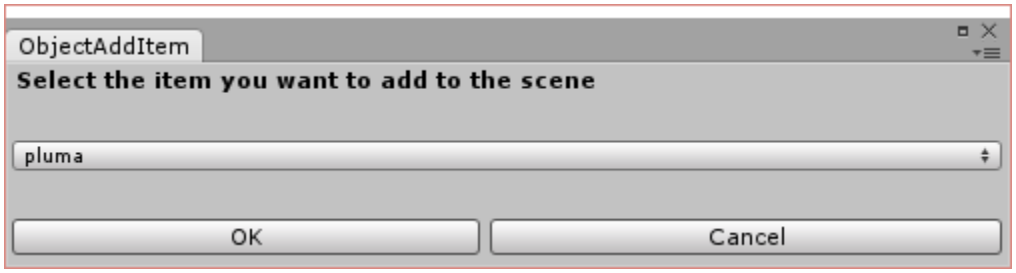

Figure 3-11 Example of *BaseChooseObjectPopup*

- BaseCreatorPopup - special type of popup used in windows designated to creation (and edition) of content: cutscenes and variables / flags editor

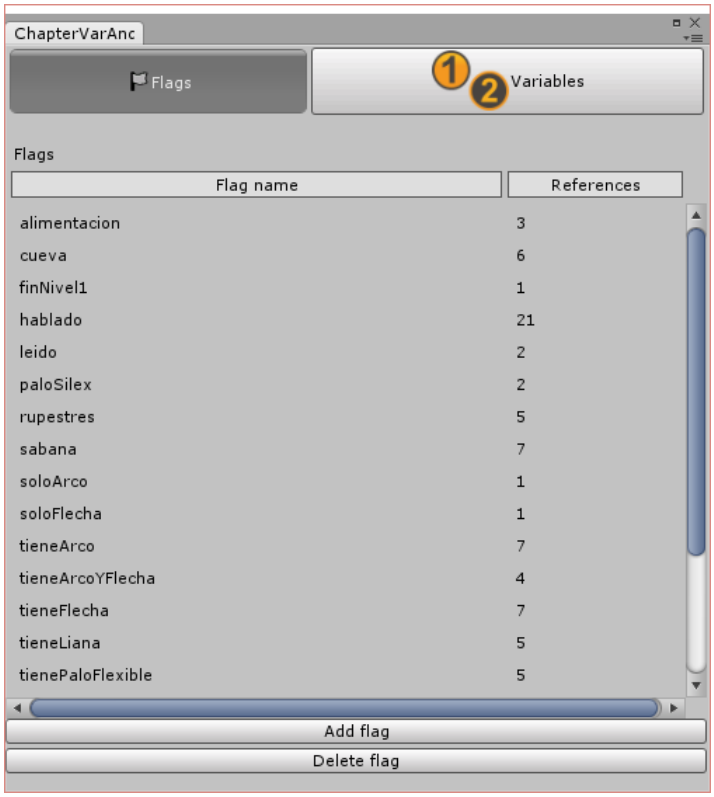

Figure 3-12 Example of *BaseCreatorPopup*

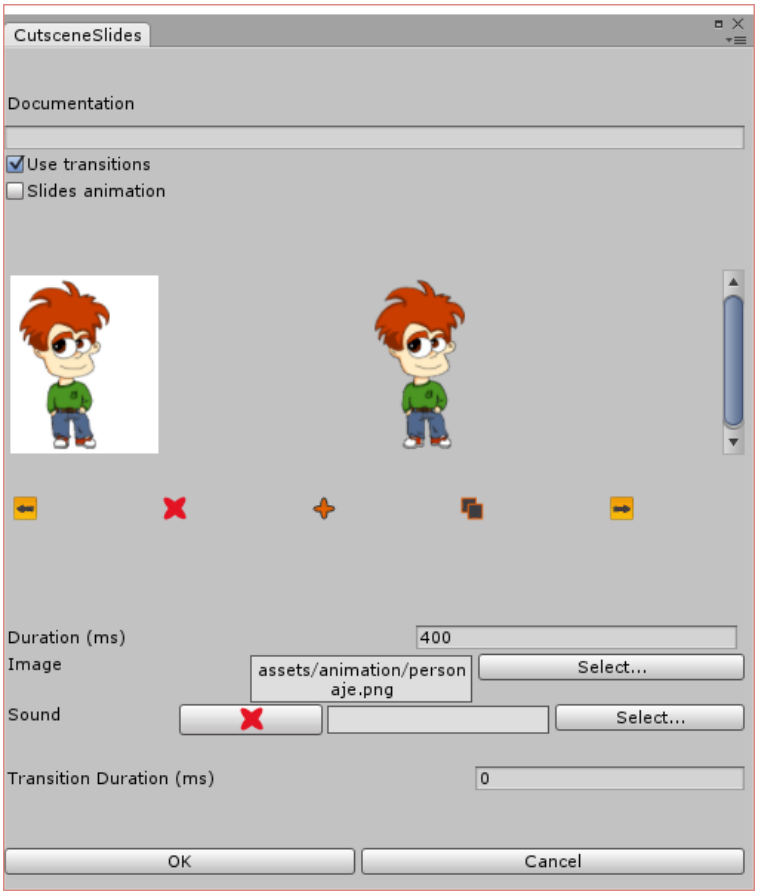

Figure 3-13 Example of *BaseCreatorPopup*

- BaseFileOpenDialog - the most commonly used type of the popup. User indicates asset file of specified type which wants to use in the game. For file selection, I decided to use standard C# file open dialog from the System.Windows.Forms namespace. After choosing the appropriate file are copied to the current project directory (for animation - .eaa file and all of its frame)

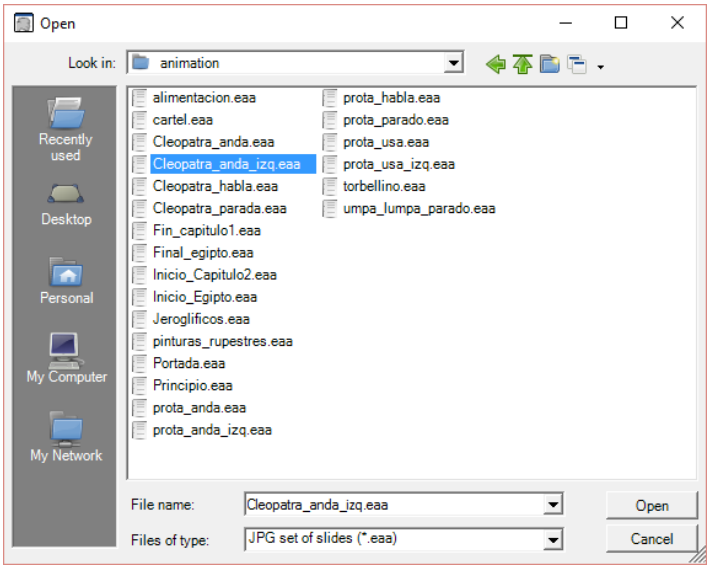

Figure 3-14 Example of *BaseFileOpenDialog*

- BaseInputPopup - used to entering name of selected elements of the game: chapter, cutscene, etc. If it is necessary, typed data is validating (eg. it is not allowed to have two flags of the same name) - if the name is not correct, “Ok” button is not active and user cannot accept operation.

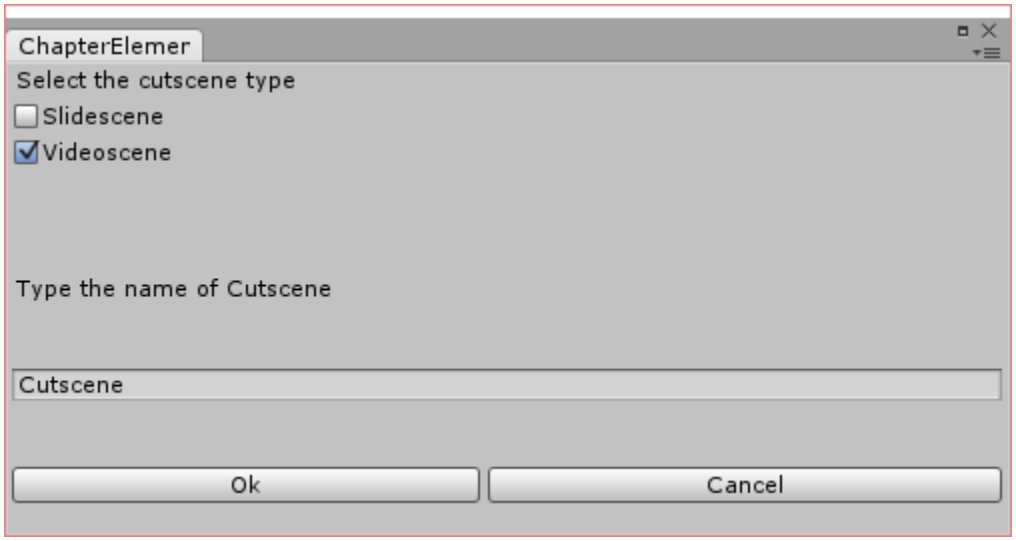

Figure 3-15 Example of *BaseInputPopup*

- ConfirmationDialog - dialog box displaying the confirmation question and enabling the approval (or cancellation) the execution of desired action. Used in the case of confirmation of critical changes, eg. deleting whole chapter.

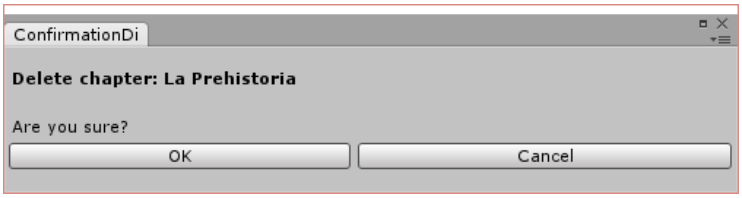

Figure 3-16 Example of *ConfirmationDialog*

Of course, for most of mentioned dialog concrete (inherited) version of them are used. Most of the will be discussed later in this paper, during the describe of editor functionality.

General editor options

The proper editor window consists of three parts - the top menu (1), the left menu (2) and the windows container associated with particular aspect of the chapter (3).

Figure 3-17 Editor overview – top menu (1), left menu (2), windows container (3)

The top menu (1), displays option lists after user click on the button, is responsible for the management of adventure. It provides options of saving and exporting the game, chapter selection, and adding or removing a chapter, editing flags and variables.

The left menu (2) is linked directly to the selected chapter and the third part of the editor window - window container.

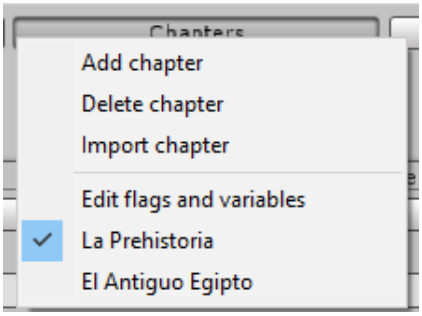

Figure 3-18 Chapter upper menu

Selecting one of the options from the left menu is, in most cases (except chapter, player advanced features), associated with the displaying list of objects of a given type occurring in the current chapter under the clicked button. Picking one of the items creates window version associated with the selected object in the central container. Clicking on a category button displays a list of objects (identical to the information submitted in the left pane, below the category button).

From the left menu, user can also add new specific elements of the chapter - after selecting the category next to its name an extra button "+" appears. Clicking on it calls input popup for entering a name. In the case of creating a new cutscene, there is also necessary to choose type of scene - videoscene or cutscene. It is also possible to remove or duplicate selected item (duplicated element has name changed in terms of game consistency - id will not be repeated).

The main part of the editor view, which is windows container, is managed by the EditorWindowBase class. It handles events associated with the left menu and, depending on the executed action, changes the displayed window type. In most aspects of the chapter, after clicking on the category, list of chapter items of specified type is displayed; clicking on a specific element of the list element causes proper editor view for a particular aspect of the chapter.

Common components of different chapter elements

Editor views for various chapter items have some common parts. In order not to repeat, I decided to describe them before going on with each editor view. In the case of substantial differences, changes will be indicated.

Documentation

“Documentation" view for each chapter aspect is almost the same. It consists of fields of TextArea type to describe a specific object and, optionally, the name of the documentation.

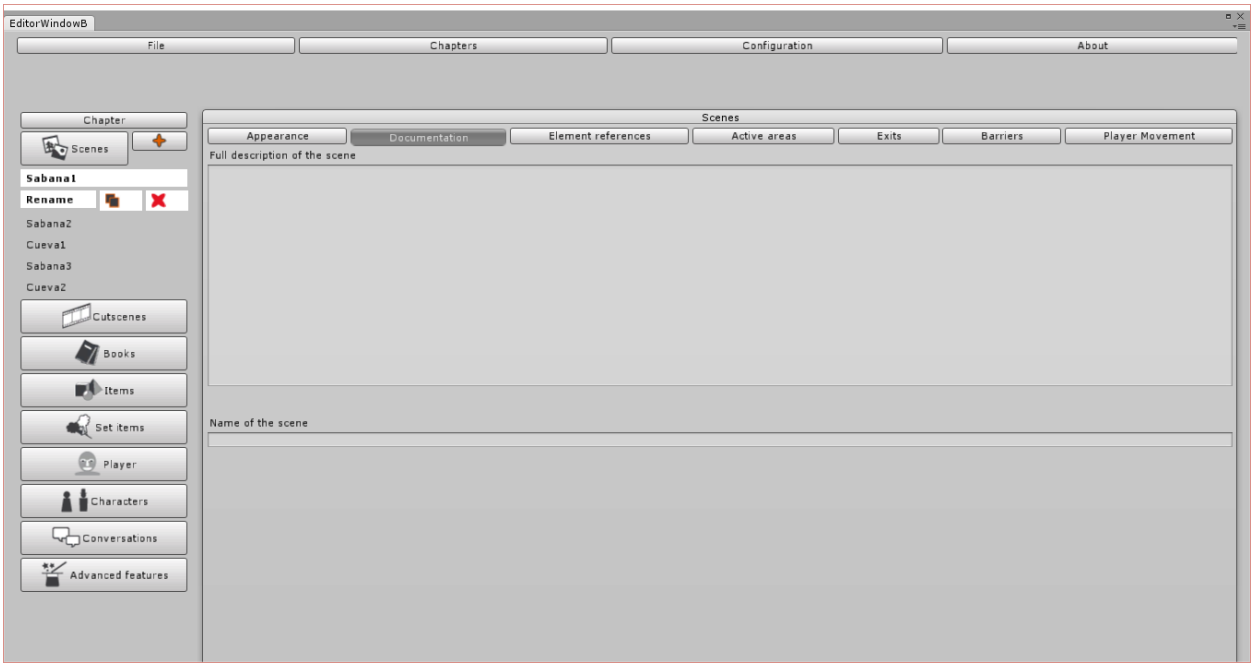

Figure 3-19 Documentation view

Tables

Table is commonly used view component. Unity does not natively support tables, so I had to build my own version of it. Base version is consist of GUI Buttons, Labels and Boxes, arranged in the specific order and the specific position.

Table has two parts: the main table, which is responsible for the displaying elements and handles the choice of single element, and the right panel, made of buttons for adding, removing an element, duplicating them or moving up or down. All buttons are not available in all cases.

Handling buttons events is always the same - only events called after the action (eg. removing scene exit corresponds to different object than removing area appearance) are swapped.

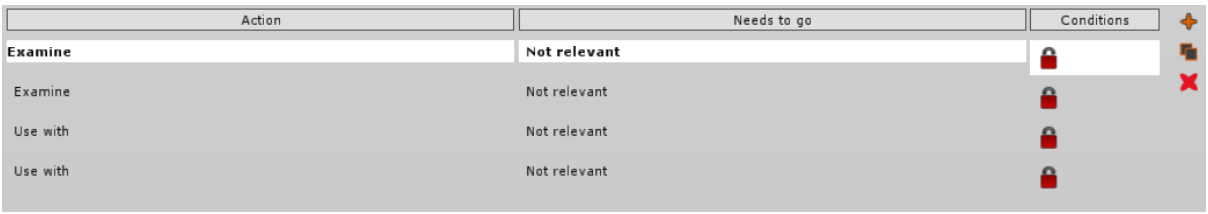

Figure 3-20 Table component

Asset chooser

Commonly used is also a group of GUI components associated with assets selecting. It consists (usually) of the description of the destination files, the reset button (which removes reference to selected file), box displaying the relative file path and, the most crucial, the button calling window inherited from the BaseFileOpenDialog, allowing to select a specific file type from hard drive.

Figure 3-21 Asset chooser component

Conditions editor

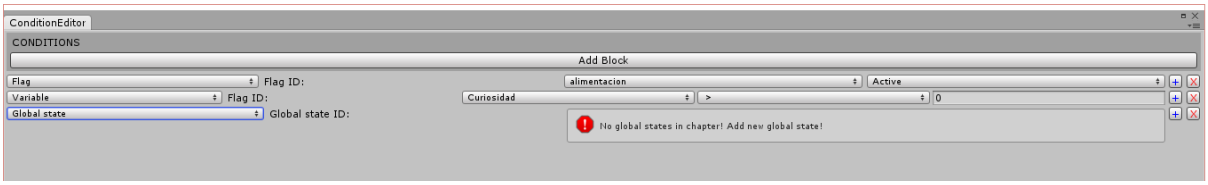

Figure 3-22 General conditions editor overview

Conditions editor is a component used standalone (eg. for editing conditions of Active areas) and as part of effect editor. For both cases, according to principles of object-oriented programming, the base code is the same, the only differences are connected with code responsible for the form of presentation (window of standalone version is part of the effect editor window).

During initialization of editor, parameter of type ConditionController is given - for the editor does not matter what type of object it is associated with it.

Condition system is based on the usage of Flags, Variables and Global states. Each condition block applies to one of them. User can choose type from EditorPopup - if in game there is no properly flag, variable, global state, instead of fields for editing corresponding comment is shown.

Possible modifications of each condition:

- Flag - selection of Flag from EditorPopup and setting Active/Inactive value
- Variable - selection of Variable from EditorPopup and setting numeric value
- Global state - selection of Global state from EditorPopup and setting is satisfied/is not satisfied value

The editor itself was written in accordance to factory design pattern. Editors for each condition type implement ConditionEditor interface. When factory instance is created with usage of LINQ expressions the application domain is searched through (System.AppDomain.CurrentDomain.GetAssemblies) and types of conditions are selected (SelectMany (p => s.GetTypes ()). Where (p => typeof (ConditionEditor). IsAssignableFrom (p))). This affects the extensibility of solution - for new type of condition the only thing to do is creation of a new class implementing the ConditionEditor interface.

Effects editor

Effects editor is used to edit various types of effects, which are sets of actions associated with specific situations in a game. During initialization of editor, parameter of type EffectController is given - for the editor does not matter what type of object it is associated with it and what kind of effect it concerns.

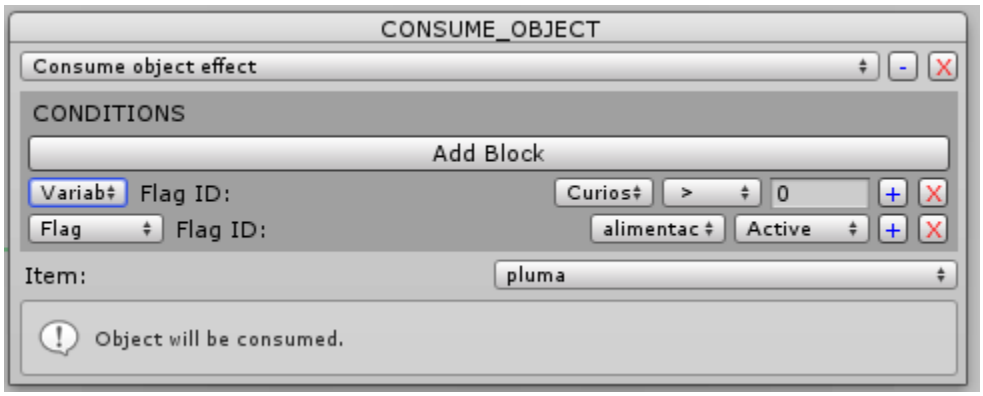

Figure 3-23 Single effect block in *Effects editor*

Effect editor system is based on the usage of different types of effects. List of effects editors corresponds to the list of available effects in eAdventure engine (each point given effect and possibilities of modification has been discussed):

- ActivateEffectEditor - changes state of Flag to active; modification through the selection of target Flag from EditorPopup
- CancelActionEffectEditor - prevents default set effects to be executed
- ConsumeObjectEffectEditor - removes Item from inventory; modification through the selection of target Item from EditorPopup
- DeactivateEffectEditor - changes state of Flag to inactive; modification through the selection of target Flag from EditorPopup
- DecrementVarEffectEditor - decrements selected Variable by specified value; modification through the selection of target Variable from EditorPopup and typing numeric value
- GenerateObjectEffectEditor - adds Item to inventory and removes it from scene; modification through the selection of target Item from EditorPopup
- HighlightItemEffectEditor - highlights Item; modification through the selection of target Item from EditorPopup and type of highlight (No highlight, Red/Green/Blue highlight, Borders highlight)
- IncrementVarEffectEditor - increments selected Variable by specified value; modification through the selection of target Variable from EditorPopup and typing numeric value

- MacroReferenceEffectEditor - launches selected Macro; modification through the selection of Macro from EditorPopup
- MoveNPCEffectEditor - moves selected Character to specified position; modification through the selection of target Character from EditorPopup and target x and y coordinates
- MoveObjectEffectEditor - moves selected Item to specified position with interpolated scale and speed; modification through the selection of target Item from EditorPopup, target position x and y coordinate, translate and scale speed and size-scale
- MovePlayerEffectEditor - moves player to specified position; modification through typing target x and y coordinates
- PlayAnimationEffectEditor - plays sound; modification through the selection of target sound file
- PlaySoundEffectEditor - plays animation; modification through the selection / creation / edition of animation
- SetValueEffectEditor - sets selected Variable specified value; modification through the selection of target Variable from EditorPopup and typing numeric value
- ShowTextEffectEditor - shows text at specified position; modification through typing of text will be shown at selected position
- SpeakCharEffectEditor - selected Character will say specified sentence; modification through the selection of target Character from EditorPopup and typing text which will be said
- SpeakPlayerEffectEditor - player will say sentence; modification through typing text which will be said and typing target x and y coordinate where text will appear
- TriggerBookEffectEditor - opens selected Book; modification through the selection of target Book from EditorPopup
- TriggerConversationEffectEditor - starts selected Conversation; modification through the selection of target Conversation from EditorPopup
- TriggerCutsceneEffectEditor - plays selected Cutscene; modification through the selection of target Cutscene from EditorPopup
- TriggerLastSceneEffectEditor - goes back to previous scene;
- TriggerSceneEffectEditor - changes to selected Scene; modification through the selection of target Scene from EditorPopup target x and y coordinates where player will be spawned

- WaitTimeEffectEditor - stops game for specified time

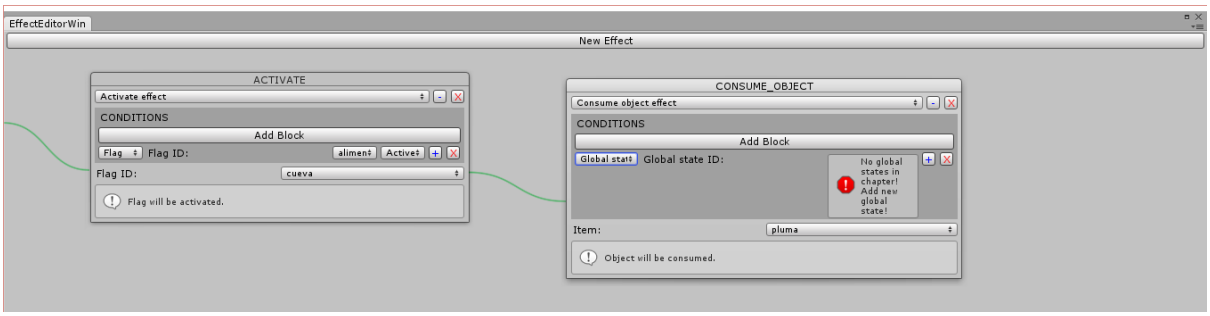

Figure 3-24 Connections between nodes in *Effects editor*

In the editor window each block effect is represented by a separate window (which can be collapsed) connected to each other by Bezier curves.

Like the Condition editor, Effects editor was written in accordance to factory design pattern. Editors for each effect type implement EffectEditor interface. When factory instance is created with usage of LINQ expressions the application domain is searched through (System.AppDomain.CurrentDomain.GetAssemblies) and types of effects are selected (SelectMany (p => s.GetTypes ()). Where (p => typeof (EffectEditor). IsAssignableFrom (p))). This affects the extensibility of solution - for new type of condition the only thing to do is creation of a new class implementing the EffectEditor interface.

For each effects it is possible to add conditions blocks. Design of architecture allows to use whole code associated with Condition editor - the only change is part responsible for graphical presentation in the editor.

Ability of adding the appropriate types of effects in the editor is associated with the presence of elements of that type in the game - for example, option to add ActivateEffect will not appear if user did not add anyflag, TriggerCutsceneEffect will not be available unless user declare at least one cutscenes, etc.

Language system

Language system is based on the usage of language files from eAdventure standalone version. Just like the original, 10 languages are available. After changing the language, whole texts changes in real time. The language files are in .xml format; each label is a separate element of the structure ("entry"), has assigned attribute ("key"), which identified it, and the appropriate text

content. Translations are stored in the C# Dictionary collection. Language change is equal to loading values into Dictionary from appropriate language file.

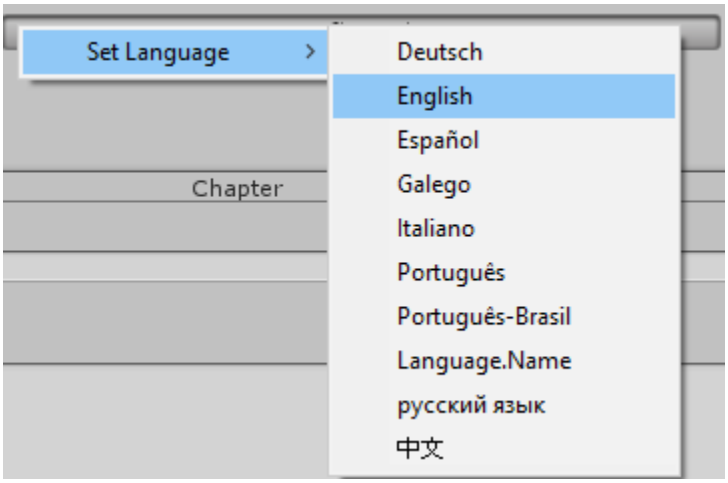

Figure 3-25 Language system menu

Chapter

Chapter window is one of the most basic one. From this window user can change the name of the chapter, choose the initial scene of chapter and set chapter description. Description is optional and it does not appear anywhere in game.

Scenes

Scene window supports the highest amount of subwindows. Two of them (barrier and player movement) appear only when game type is third person.

Appearance

Scene appearance window allows user to make basic configuration of the appearance of the scene. The user can determine scene background, foreground mask and background music, looped during scene. It is worth to mention about optional foreground mask - black and white image corresponding to the background image, which can be used to indicate with part of background scene will be rendered behind or in front of the object. In the bottom half of the window a preview of actual scene background is displayed.

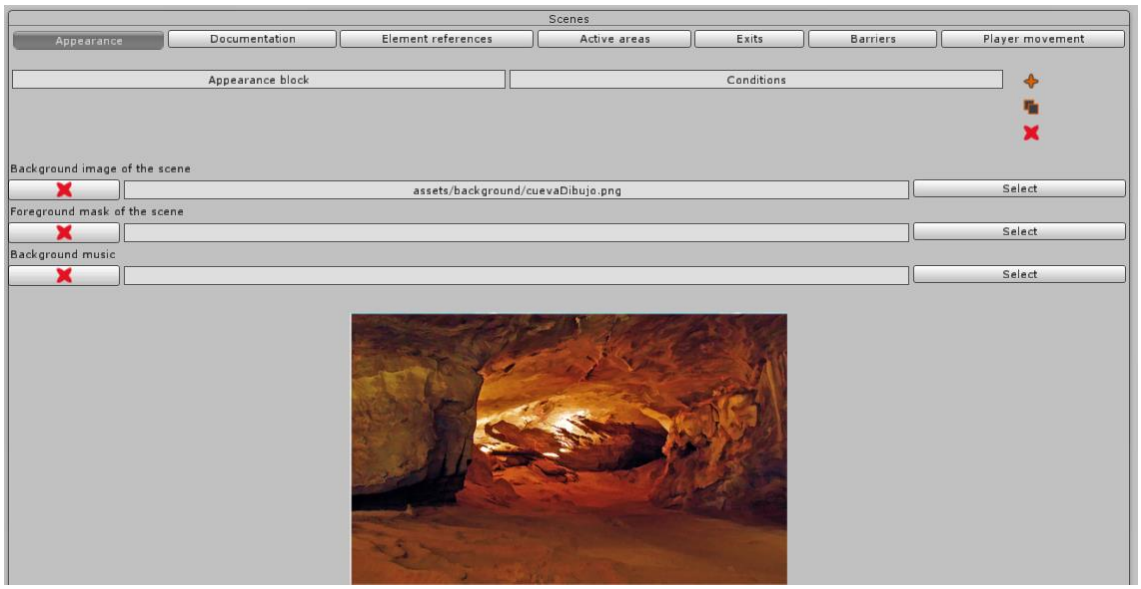

Figure 3-26 Scene appearance view

Figure 3-27 Scene background and corresponded foreground mask

Element references

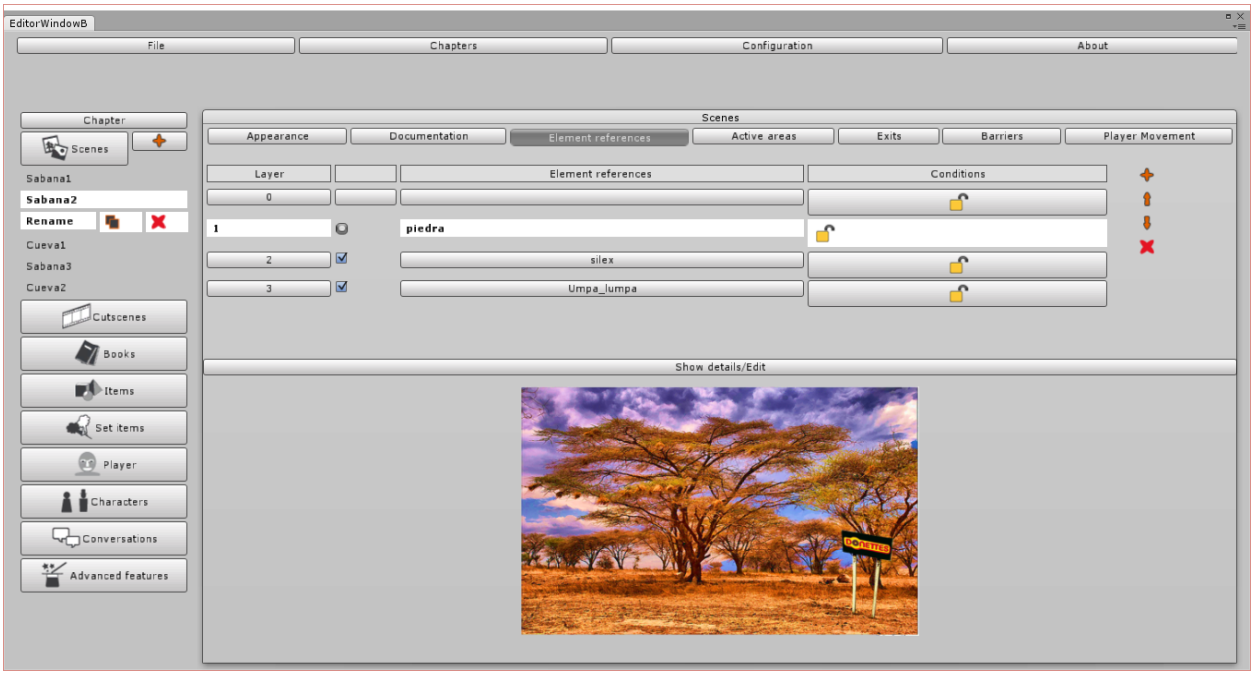

Figure 3-28 Scene element references view

Element references view is used to add and manage references to items, attrezzos or NPCs in scene. Clicking on the Add button in the table's right panel displays options as Unity GenericMenu. After that, windows of type BaseChooseObjectPopup appears to choose one of the chapter elements of a given type. From the table, user can edit the conditions for each reference.

The second part of the view is invoked editor of the objects in the scene properties (instance of window type BaseAreaEditablePopup). It is necessary to select the table element whose properties user want to modify - after that edit button become visible. In created window background is rendered with all of the referenced elements added into the scene. In addition, around the selected object blue frame is drawn (scaled texture with transparency).

Positions of the selected object can be modified in two ways: by dragging object with pressed mouse button or by setting value in fields below the image; also scale of image can be changed - the base size of object reference is the size of image associated with it, expressed in pixels. Changing the position or scale will change Unity Rects associated with the displayed object and selection frame. Unity does not provide a method for checking if cursor is over a specific component, so when you move objects with the mouse, check through a comparison of coordinates of the cursor and the square position containing object is necessary.

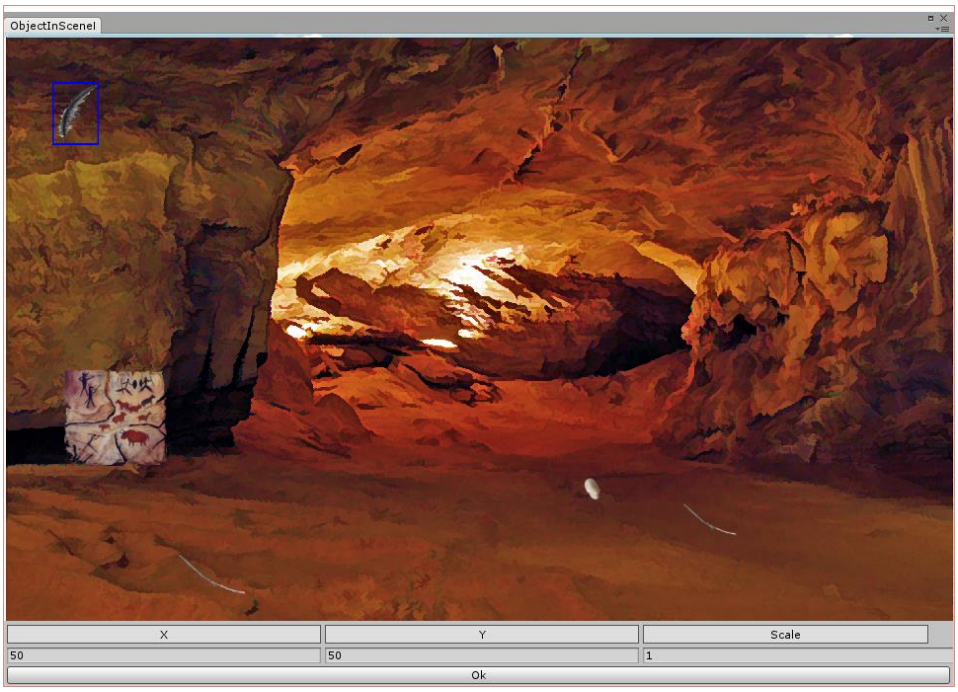

Figure 3-29 Objects in scene references editor

Active areas

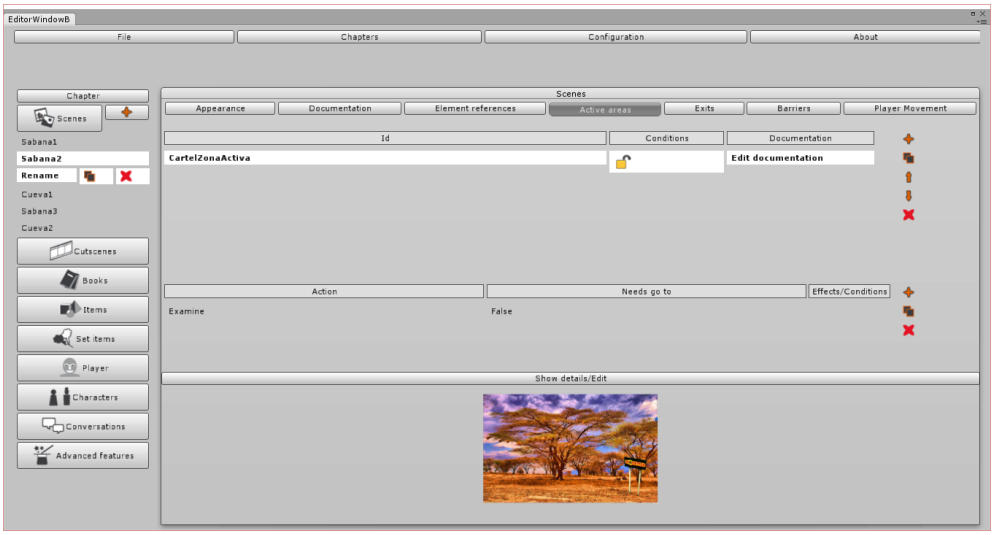

Figure 3-30 Scene active areas view

Active areas view is used to add and manage interactive rectangles, which are part of the scene. Set of actions is linked with each active area. Action editor (each active area is associated with an individual set of actions) is based on the previously described table.

Like in the element references, click on the add button in the table's right panel displays options as Unity GenericMenu. Basic (not referred to other objects) types of actions are available: Use, Examine, Grab. For each of the actions user can edit Conditions, Effects and Non-effects.

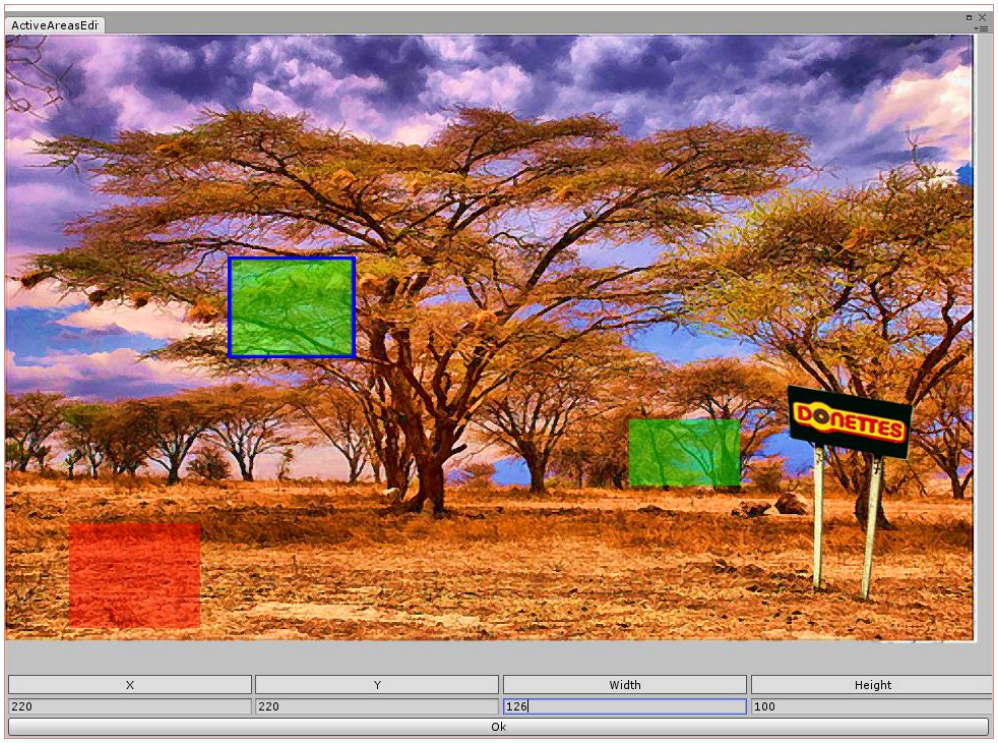

Figure 3-31 Active areas editor

The final aspect of the view is the invoked editor of active areas properties. Scene background is rendered with all exits (red textures with transparency) and the active area (blue textures with transparency) representation. Around the selected area blue frame is rendered . The principle is analogous to the editor locations of the element references - handled by dragging an item or by setting values in fields below image. Instead of scale, dimensions of area are editable.

Exits

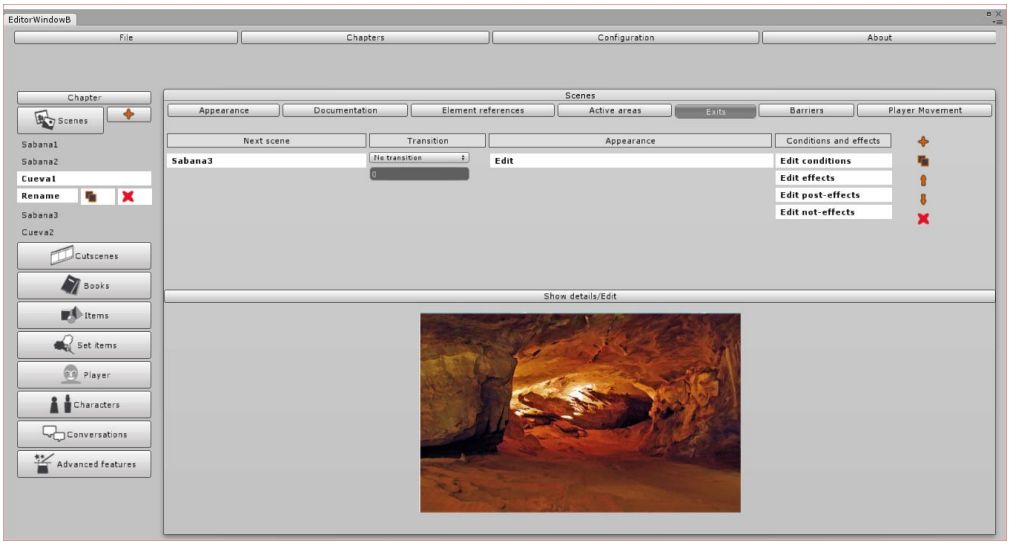

Figure 3-32 Scene exits view

Exits view is used to add and manage exits from the scene, i.e. areas after being interacted (with fulfillment conditions) causes game progress to the next stage. Click on the add button in the table's right pane displays window of type BaseChooseObjectPopup to choose one of chapter scene to whom game should progress. From the table, user can modify the time and type of transition between scenes (no transition, top to bottom, left to right, right to left, fade in), visual and audio aspects of exit, including cursor image when hovering over an exit area and text which will be displayed above it. In addition, from the table it is possible to modify the conditions, effects, post-effects and not-effects.

The final aspect of window is exit properties editor. It is analogous to the active area editor in terms of usage and components (all exits and active areas are rendered) - only difference is that the changes made by the user refer to exit.

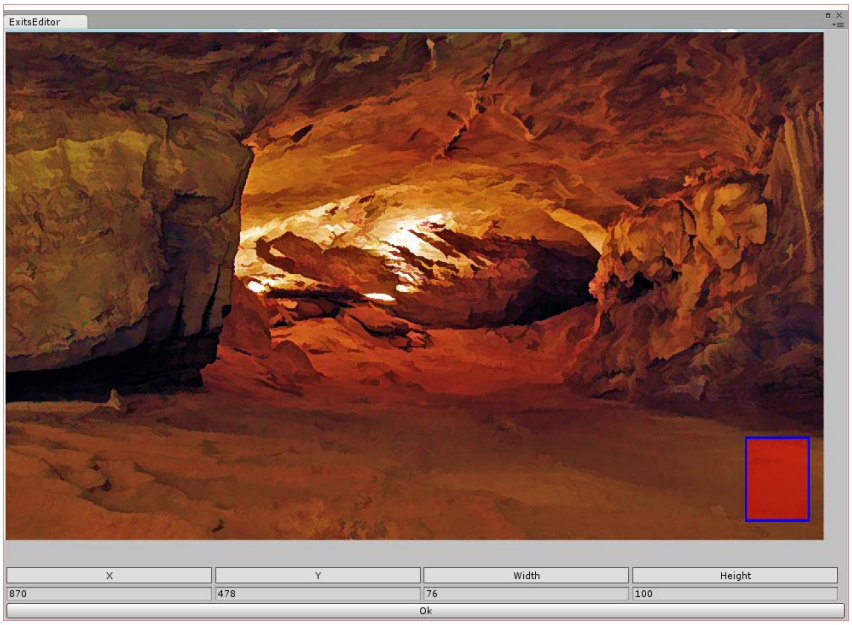

Figure 3-33 Exits editor

Barriers

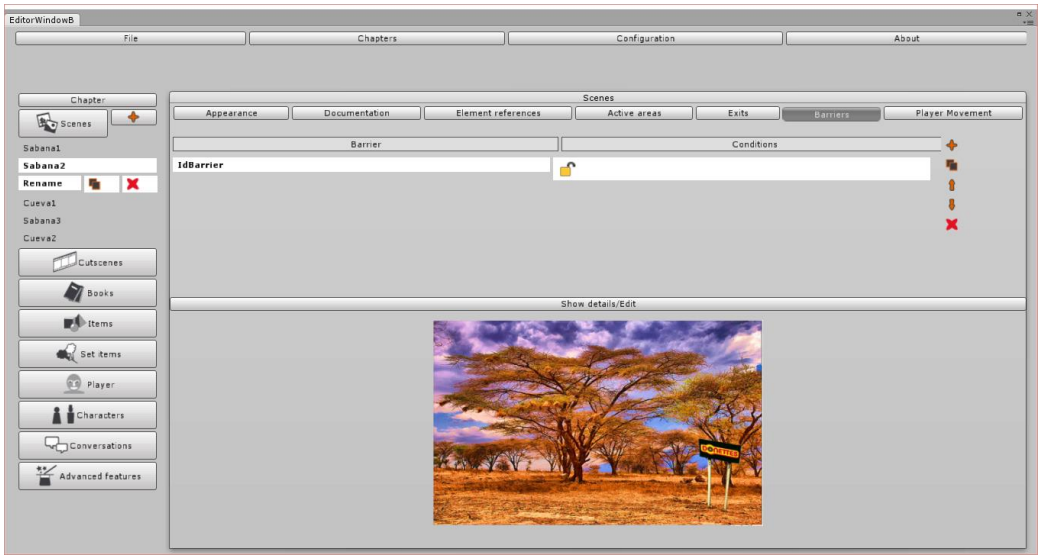

Figure 3-34 Scene barriers view

Barrier view is available only for third person games. It is also the least complicated - it is used to add and manage barriers - rectangles preventing main character to go through occupied area. To each of barrier a set of conditions is related. If conditions are satisfied, main character will be blocked by barrier; if conditions are not satisfied - barrier becomes not active.

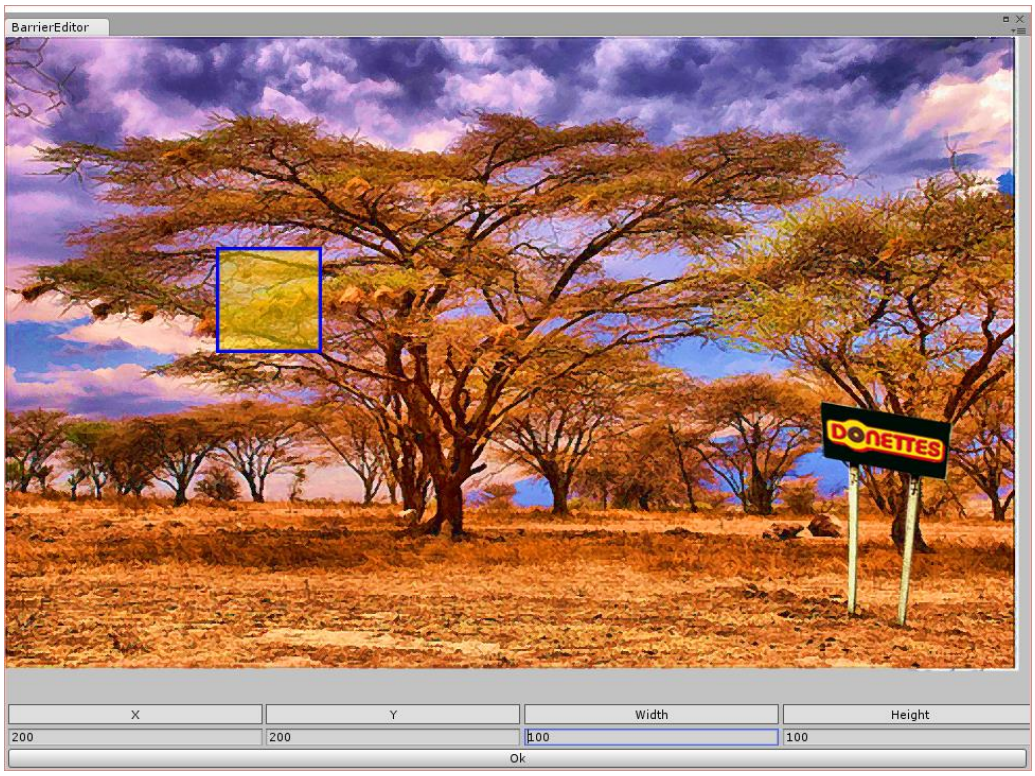

Figure 3-35 Barrier editor

Barrier properties editor is, like the previous ones, instance of type windows BaseAreaEditablePopup, but in this case, except the background only representations of barriers are rendered. Editing position and dimension of barrier is similar to the previous views.

Player movement

Like the barrier view, player movement view is available only for third person games. As the name implies, it is used to define the character movement in the scene. Character can move in two ways: first, more basic - navigation is limited only to moving left or right, and user in this case defines only the initial position (and scale) of the protagonist.

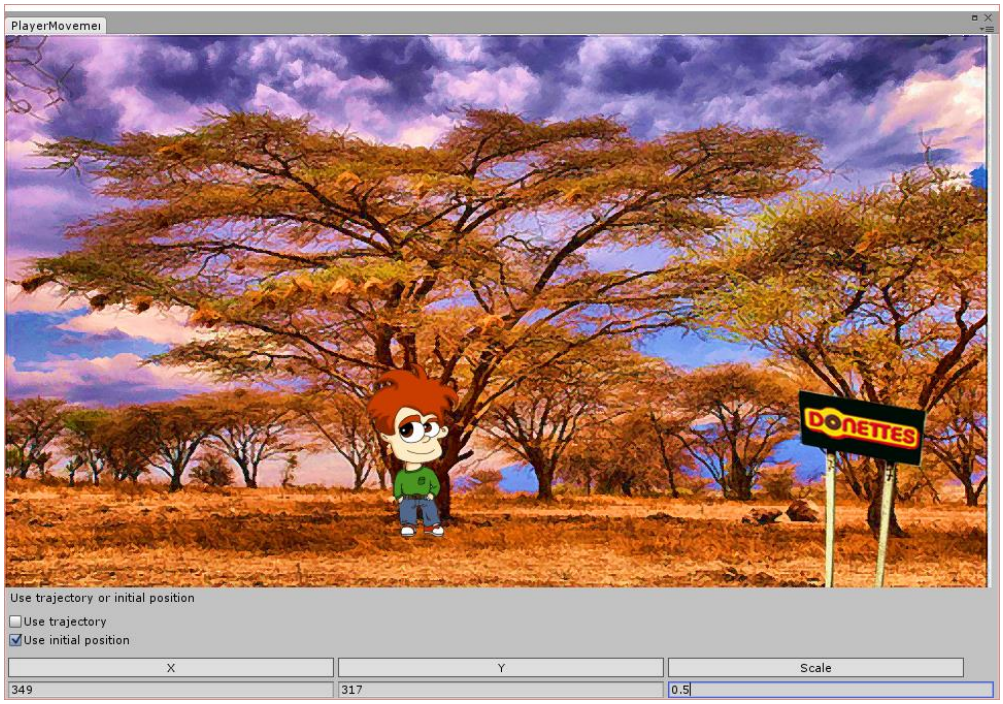

Figure 3-36 Player movement editor (initial position)

A second, more complex approach, is based on so-called trajectories. Trajectories can be defined and edit by 4 tools.

The first is the "Add node" tool. When is selected, after clicking on empty space new trajectory is created. After Clicking on space occupied by node, user is able to change its position by mouse drag. During dragging, rectangle representing node position is constantly recalculated. When mouse is above node, user can scale current node by clicking key “+” or “-” on keyboard.

The second tool is "Add new side". When tool is selected, user is able to create connections between nodes (is necessary to select two nodes to be connected). The lines representing sides are drawn as one pixel white texture stretched and rotated (by using method GUIUtility.RotateAroundPivot) with calculated angle: Mathf.Rad2Deg * Mathf.Atan (d.y / d.x), where d is a vector between the start and end position.

The third tool is "Set initial node". When it is selected, user can choose starting node by clicking on it. Node selected as initial has red dot rendered in the middle of player texture.

The last, fourth tool is "Delete node". When it is selected, user can remove available trajectory nodes. After removing the node also associated sides between nodes are being remove.

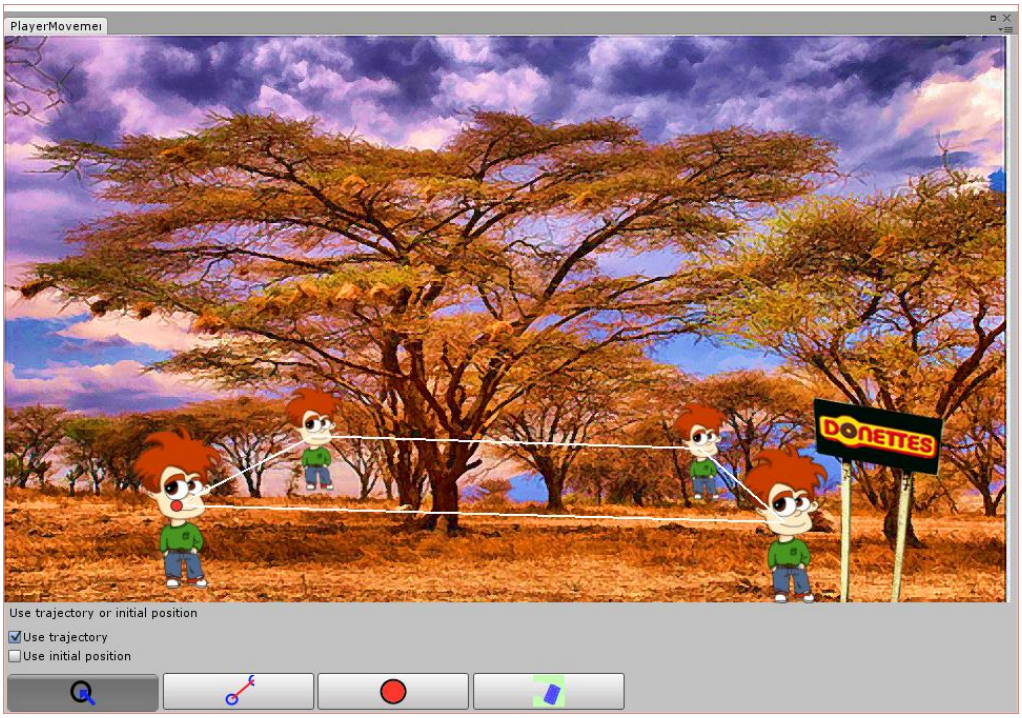

Figure 3-37 Player movement editor (trajectories)

Cutscenes

Cutscenes window contains three subwindows. One of them is the documentation window, so only the other two will be discussed in this chapter.

Appearance

Cutscene appearance view have two different forms, depending on the type of cutscene. For videoscene, configuration is limited to selection video file meant to be played and determine if user can skip the video with a mouse click.

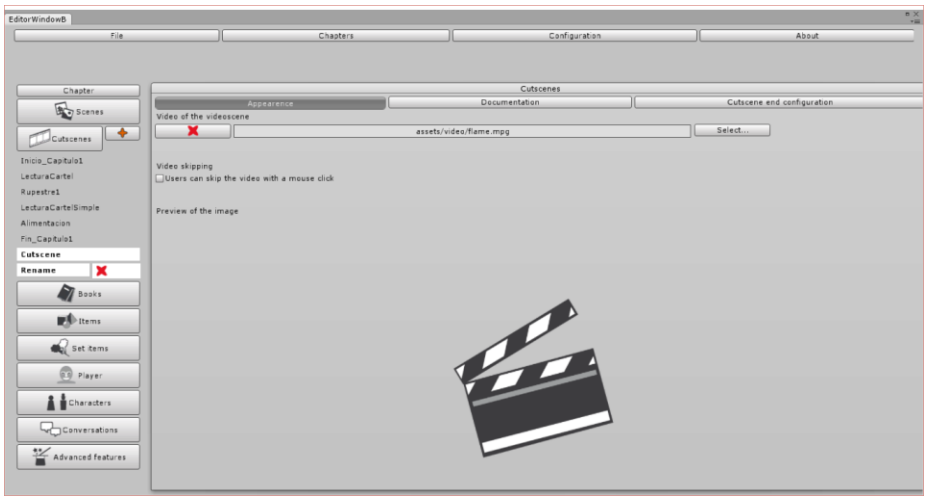

Figure 3-38 Cutscene appearance view (video)

For slidescenes it is possible select a .eaa file representing animations, edit them and create a new one. In addition, user can set background music for a cutscene. Create/edit cutscene option merits special attention. If cutscene is not selected, instance of BaseInputPopupDialog will appear. After typing valid name, cutscenes creator will appear.

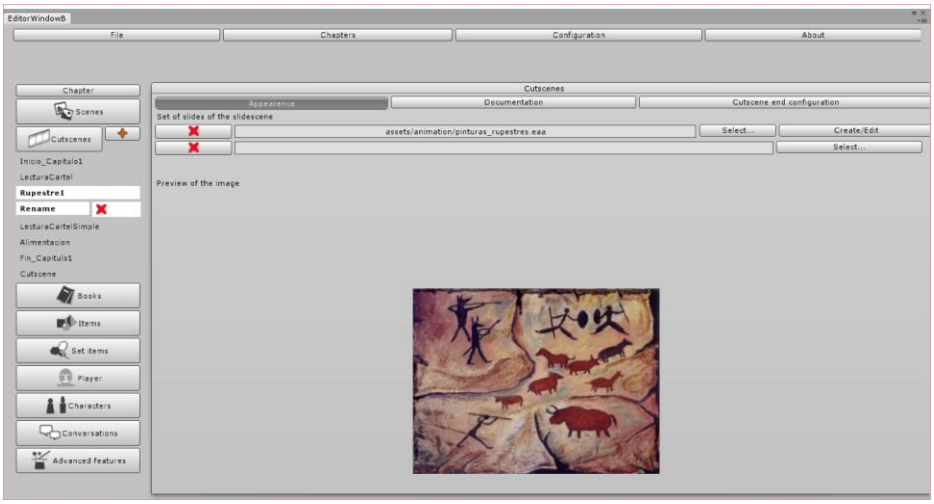

Figure 3-39 Cutscene appearance view (slidescene)

Slidescene cutscene, represented by .eaa file (which is de facto almost equal to the .xml file). It consists of a set of successive frames and transitions between them. User can write documentation to animation, choose if slides transition and animation should be used. In addition, it is possible to add, delete, duplicate frames, and move them in forward/backward. For each frame

is possible to change duration, image that will be displayed and sounds effects. Image and sound files are picked by the instance of BaseFileOpenDialog. Moreover user can choose the duration of transition to next frame. All changes are saved to the .eaa file in the animation directory (/asset/animation), together with referenced files.

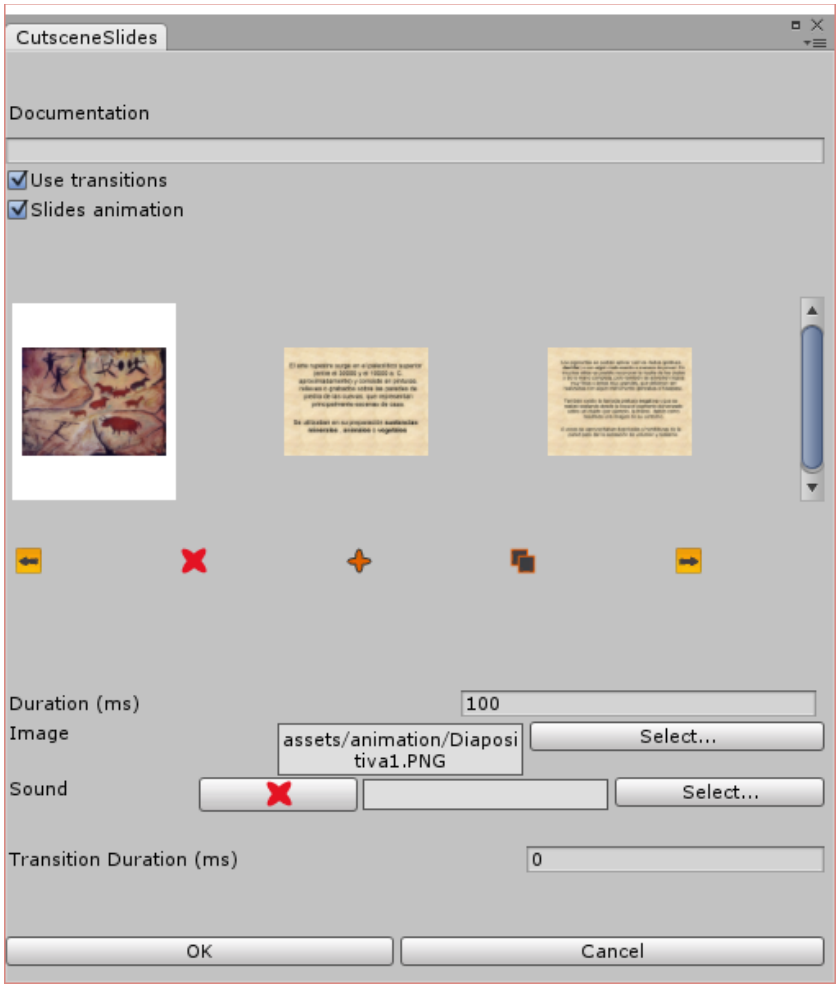

Figure 3-40 Slidescene editor

Cutscenes and configuration

Cutscenes end configuration view allows user to select behavior after cutscene reaches end. Possible options allows to return to previous scene, end chapter or go to a new scene. For the last option user additional configuration is needed - user must specify target scene (either scene or cutscene), edit effects, select transition type and time.

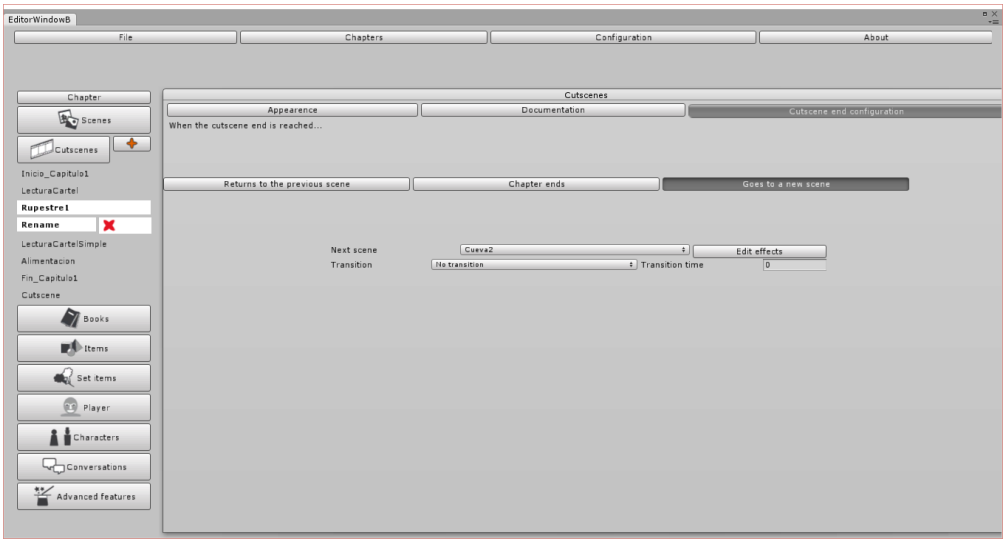

Figure 3-41 Cutscene end configuration view

Books

Book windows contains three subwindows. One of them is the documentation window, so only the other two will be discussed in this chapter.

Appearance

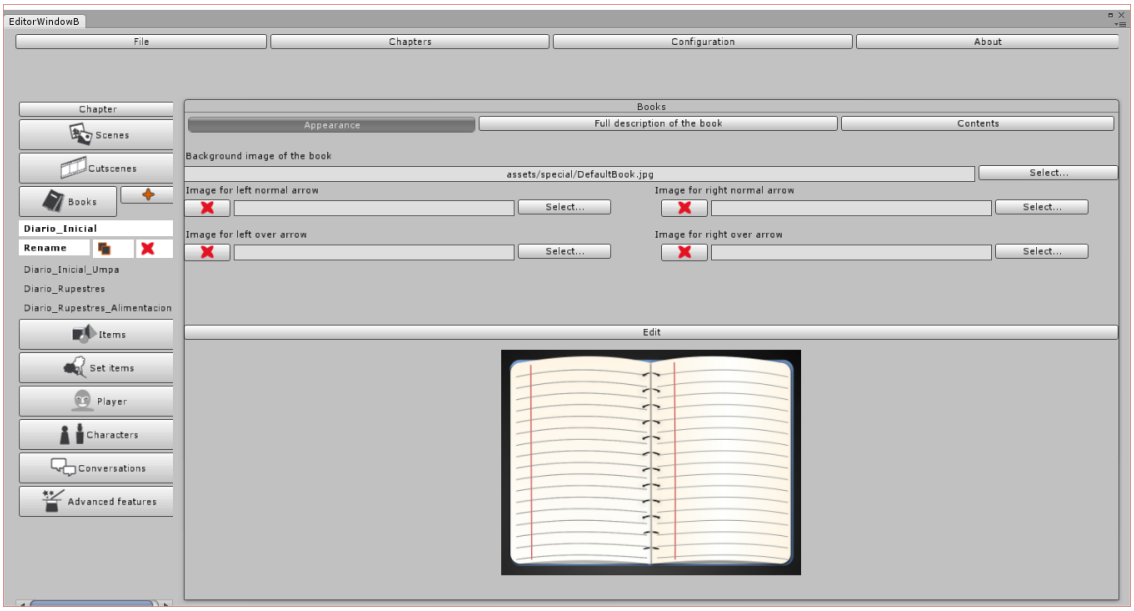

Figure 3-42 Book appearance view

Appearance view allows user to make basic configuration of the visual aspect of the book - background and left / right arrows (normal and mouse-over) images, which can be selected by the instance of BaseFileOpenDialog. In addition, user can configure the position of arrows relative to background with possibility to set default value (20 pixels from down-left or down-right corner of book image) from the window, which is an instance of BaseAreaEditablePopup. Just as previously, position editing is supported with dragging an item or by setting values in fields below image.

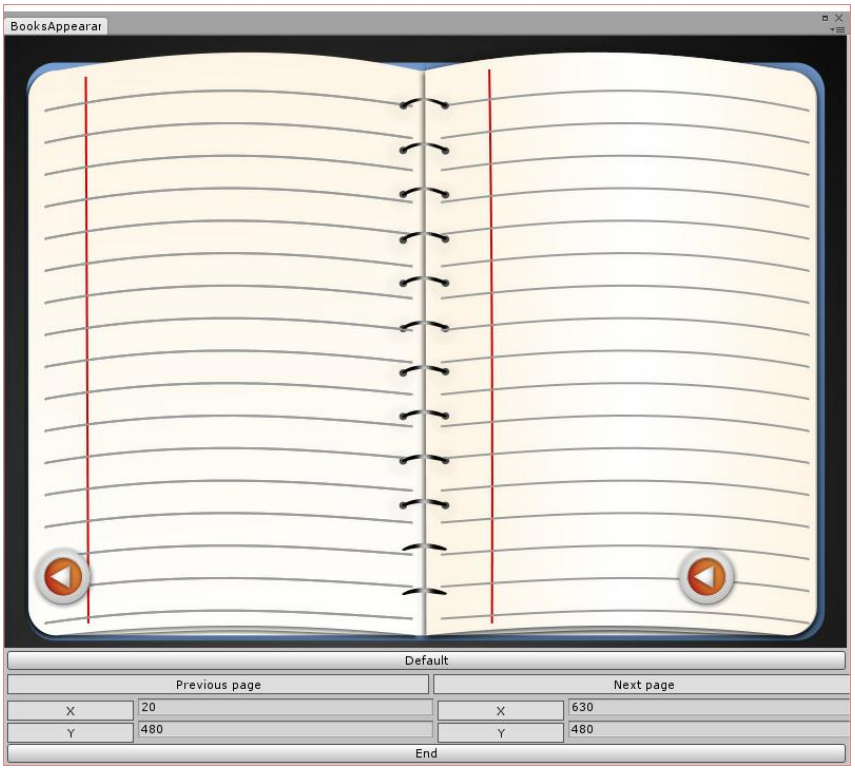

Figure 3-43 Book editor

Content

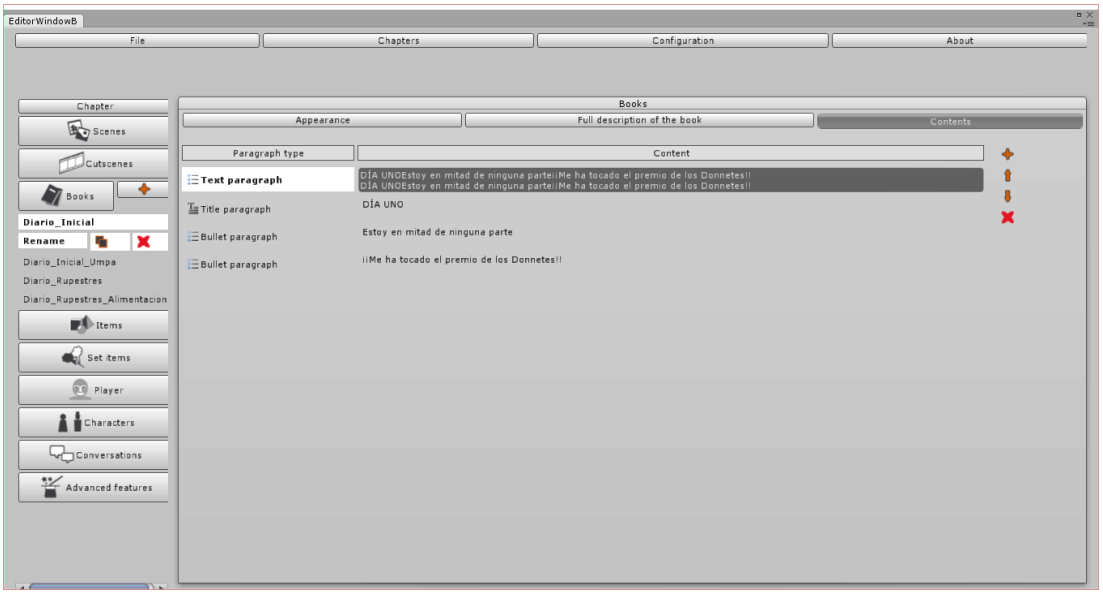

Figure 3-44 Book content view

Content view provides the functionality of books content creation. Book may consist of the following paragraphs: title paragraph, text paragraph, bullet paragraph and image paragraph. Content management is achieved by a table component. Adding more paragraphs is invoked by clicking on the add button in the table's right panel (which displays available types as Unity GenericMenu). For text paragraphs edition is based on the text in TextArea modification. For images, the user can select the target file by instance of BaseFileOpenDialog.

Items

Appearance

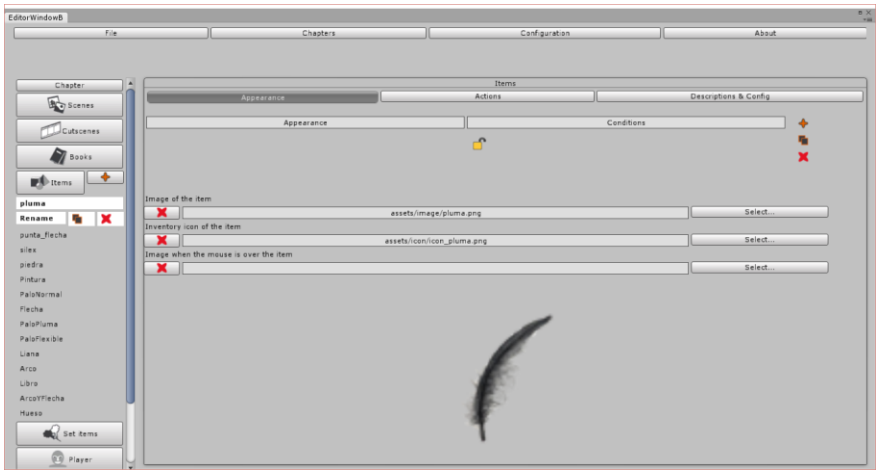

Figure 3-45 Items appearance view

Item appearance allows user to make basic configuration of the visual aspect of the item - item image, image which will be visible in inventory and image presented when mouse is over the item. All of assets can be selected by instance of BaseFileOpenDialog.

Actions

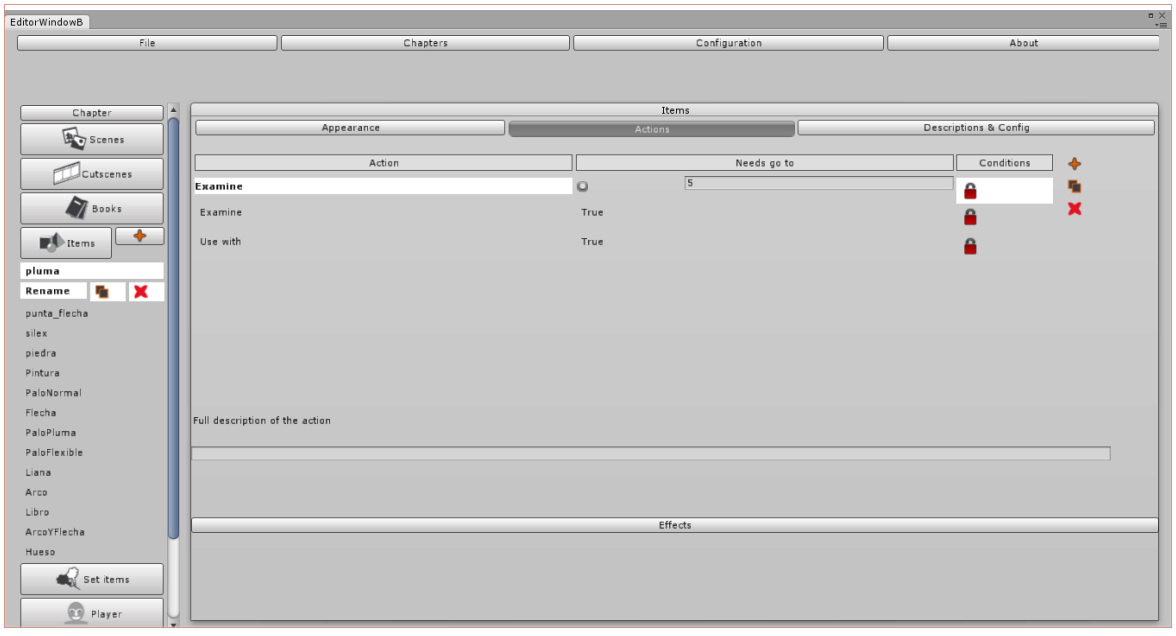

Figure 3-46 Items action view

Action window is probably the most significant view for item configuration - it allows to specify actions that can be taken as an interaction with the object. The actions can be divided into two categories: simple (ie. not linked to another part of the game) and complex (ie. referred to other element of the chapter). The first group includes action: "Use," "Examine", "Grab", while the second: "Use with ...", "Give it ..." "Drag it ...". In the second group, it is necessary to select the part of the action - target element which action will affect. Target, in the case of complex action, may be selected from the list EditorPopup appearing near to the action name. With every action set of conditions and effects are related - effects associated with the action will be applied if all conditions of interaction with the object of are met.

Description and configuration

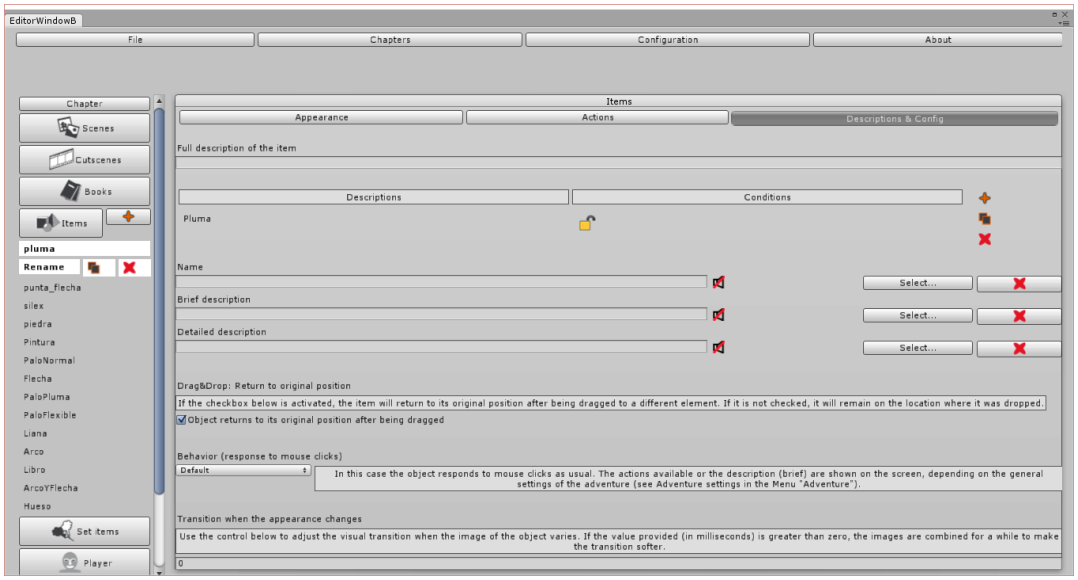

Figure 3-47 Items description and configuration view

Last item subwindow is Description and config view. It allows to add blocks of description (text and sound). Use of the block is dependent on the fulfillment of conditions assigned to it. Configuration section is for setting such options as return to original position after drag and drop (user can choose, if item should return to its original position after being dragged or should remain on the location where was dropped), response to mouse clicks and transitions times between changes of appearance.

Set items

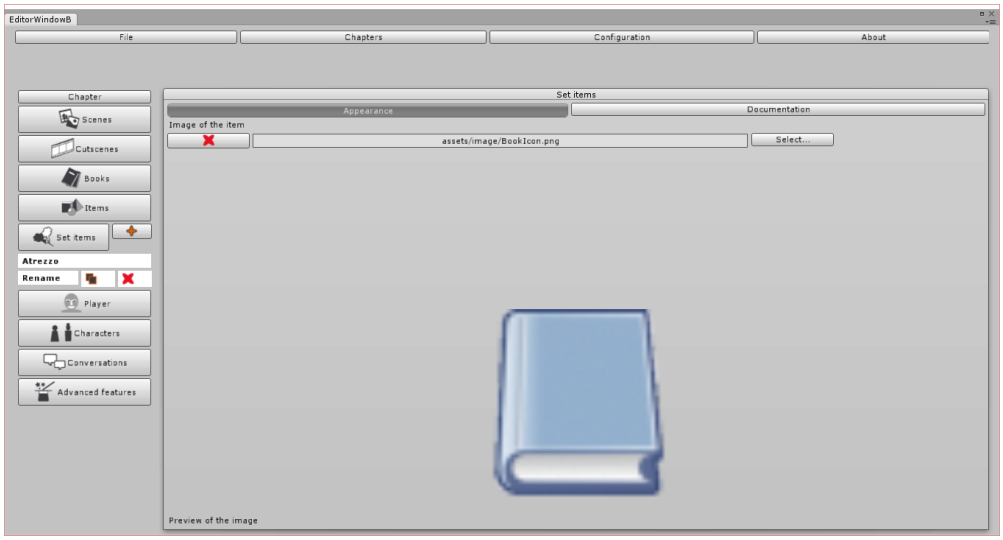

Figure 3-48 Set item view

Set item windows consists of two types of windows: documentation and appearance subwindow. From appearance view atrezzo appearance can be modified - it is only option of atrezzo modification.

Player

Player window consists of two windows, while for the third person game third window becomes available - appearance editor.

Appearance

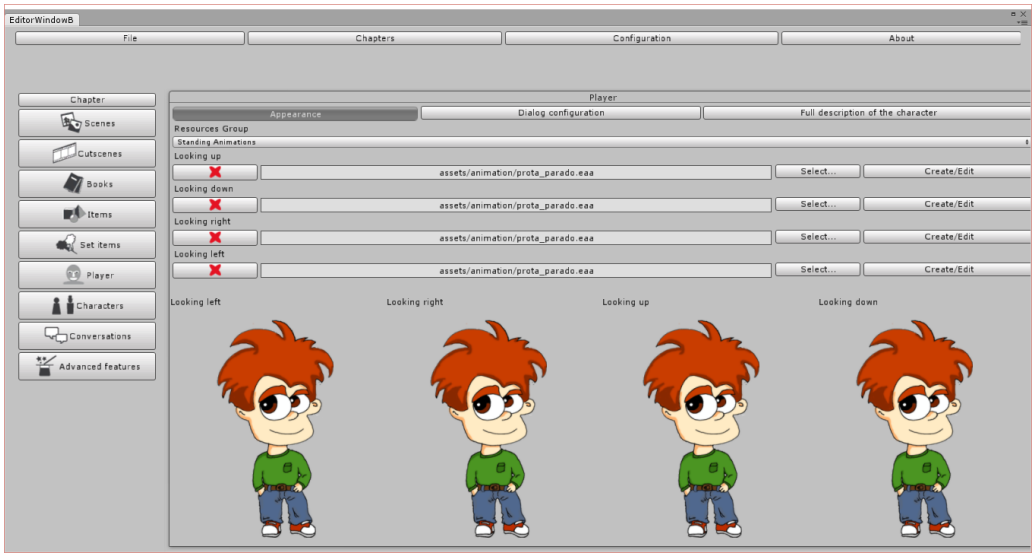

Figure 3-49 Player appearance view

Appearance window is available only for third person game. It allows user to edit a set of protagonist animations for four different modes/states of the player (modes are changed from EditorGUILayout.Popup):

- Standing animations - modifiable set of animations: look up, look down, look right, look left
- Talking animations - modifiable set of animations: speak up, speak down, speak right, speak left
- Using animations - modifiable set of animations: use object to the left, use object to the right
- Walking animations - modifiable set of animations: walk up, walk down, walk right, walk left

Used animation type depends on the position of the character relative to the "direction" of interaction. For individual animation, as in the case of slide scenes, user can select a .eaa file representing animations (selected by the instance of BaseFileOpenDialog), edit it or create a new one. If cutscene is not selected, instance of BaseInputPopupDialog will appear. After putting valid name, creator of cutscenes will become visible.

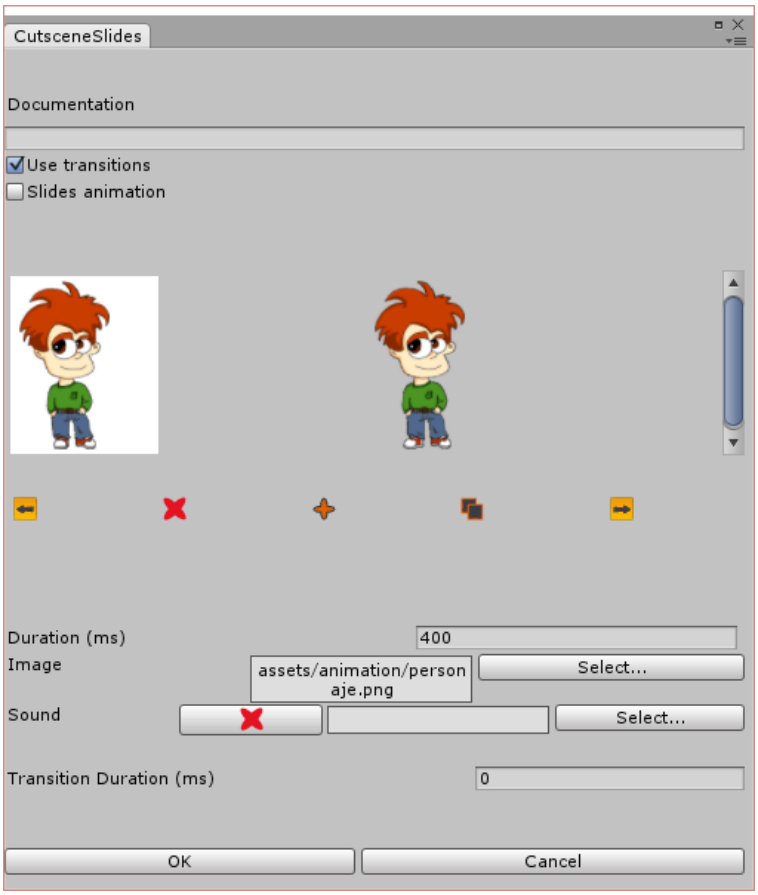

Figure 3-50 Character cutscene editor

Animation editor has been designed to be used to edit slidescenes, animations and player characters (NPCs animation) - one of parameters passed during the initialization of the window is relative animation file path, so it can used in many places, without duplication fragments of code. Therefore, the principle of action and implementation details coincide with the descriptions in the chapter of cutscene editor.

Dialog configuration

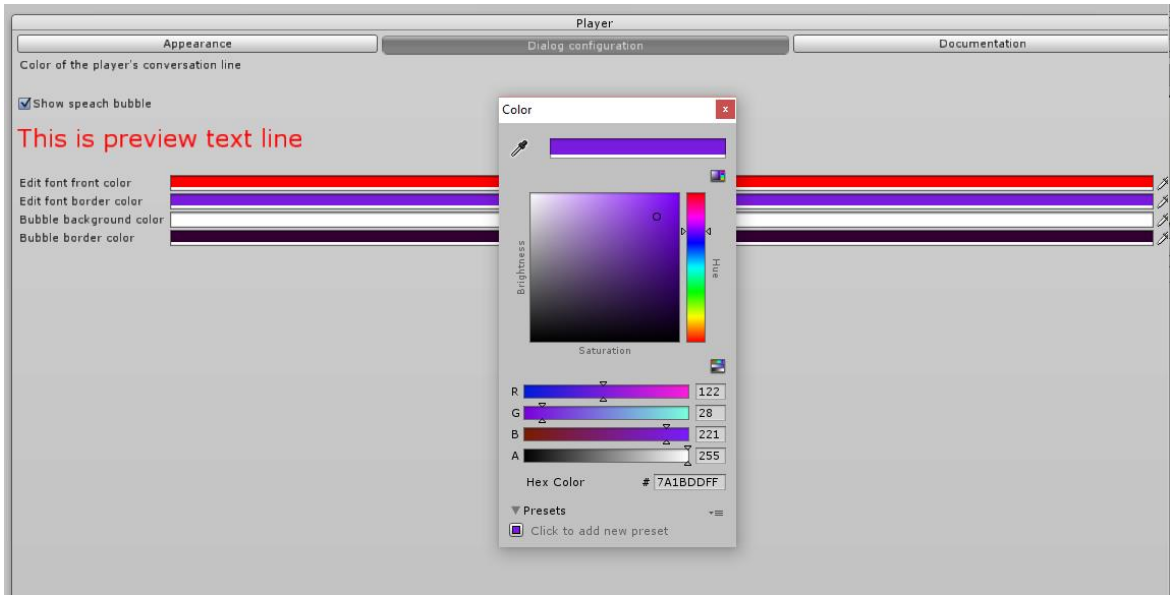

Figure 3-51 Player dialog configuration view

Dialog configuration view allows to configure visual aspect of sentences spoken by player character. Separately configurable is the appearance of font and background in which the text will be displayed. In both cases, customizable is appropriate color and border color.

For font user can change font color and color of its border; when it comes to background - bubble color and border of the bubble. Color selection is handled by Unity engine native controls of type EditorGUILayout.ColorField.

Characters

NPC configuration is similar to the configuration of player for a third-person game. The main difference is the additional subwindows for editing actions and more advanced documentation window. Both views, Appearance and Dialog configuration are almost identical (the only difference is target object). For Appearance view customizable is the same set of animations (standing, talking, walking and using animations); Dialog configuration view allows user to change the same visual aspects of sentences spoken by a character (font color and border, background bubble color and border).

Documentation

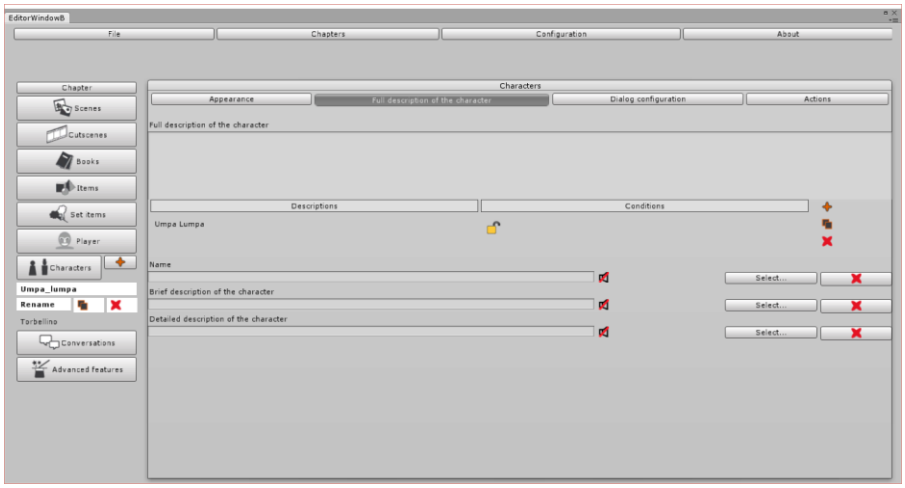

Figure 3-52 Character documentation view

Documentation view of NPC, in comparison to other documentation views, has additional property of block description. Block can be sound and text and concerns Name, Brief description and Detailed description.

Actions

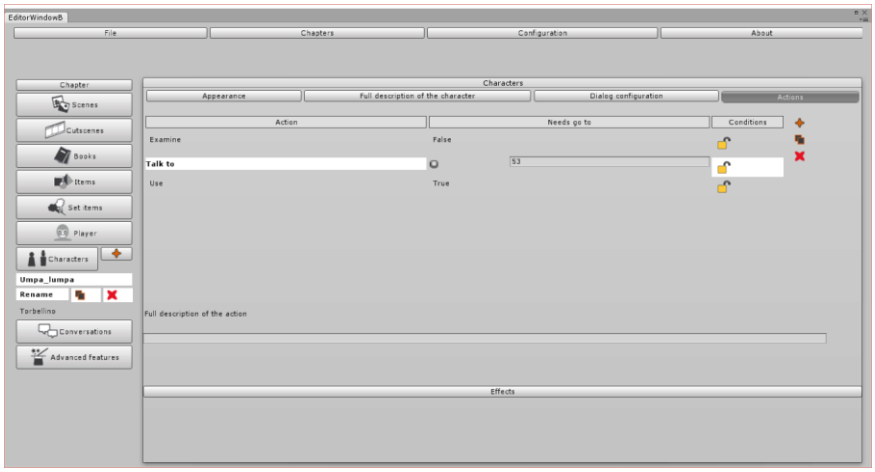

Figure 3-53 Character action view

Action view of NPC is almost the same as for Item. It allows to specify actions that can be taken as an interaction with the character. The only difference is connected with available set of actions - "Use," "Examine", "Talk to ...", "Drag it ...". Only last type of action, "Drag it ...", belongs to the complex type of action group - user should choose target from the list EditorPopup appearing

near to the action name. Rules concerning conditions and effects are the same as in the Action for items.

Conversations

Conversation editor is used to create/edit conversation. Conversation is one of possible way of interacting with the characters. Conversation is represented as a set of nodes connected together (in editor - Bezier curves are used for visualization). There are two types of nodes: Dialogue and Option node. Visually, each node is represented as a separate draggable subwindow inside Conversation editor window.

Figure 3-54 Dialog conversation node

Dialog node is a set of dialogue lines that will be read by the NPC or player character (in the order of declaring). User, besides adding/deleting dialogue lines for each of them, is able to set conditions and choose speaker. Each node can be linked to a set of effects. For each node, user can create (and set) child nodes.

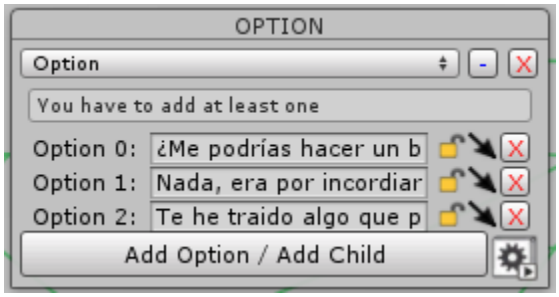

Figure 3-55 Dialog option node

Option node indicates points of a conversation in which player is obligated to make a choice. For each node user can add dialogue options that are associated with following child nodes. For each option, there is a possibility to assign the conditions, choose the linked node. Additionally, from the node level, user can edit effects and add new dialogue options.

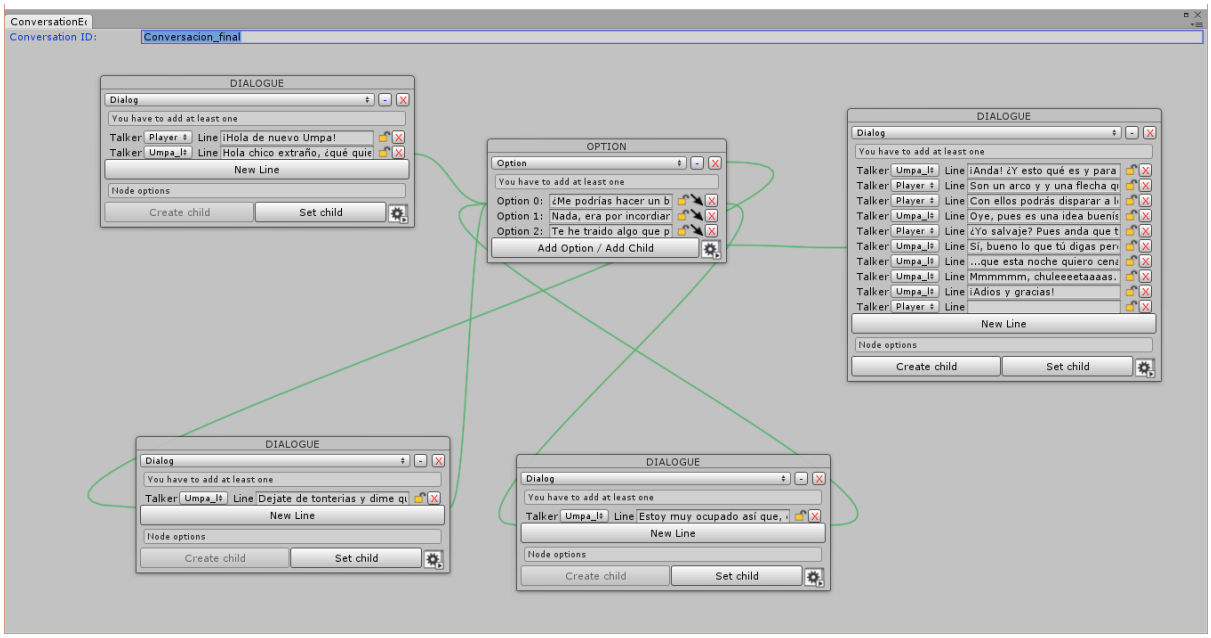

Figure 3-56 Conversation editor overview

Conversation editor was written in accordance to factory design pattern. Editors for each node type implement ConversationNodeEditor interface. This affects the extensibility of solution and accordance with object-oriented programming principles.

Advanced features

Advanced features windows contains three subwindows: List of timers, Global states, Macros.

List of timers

List of timer view allows user to configure timers, which are a mechanism for triggering blocks of effects after desired amount of time (or periodically). From the table component it is possible to add/delete/duplicate timer and set its basic parameters - time and visibility in game.

Each timer is linked to the set of conditions and effects. When initial conditions are met timer starts counting. After counts end, defined set of effects is called. Timer has more configuration aspects. User can define conditions which will abort counting (End conditions). In addition, modifiable is the visual aspect of timer in the game (name, count-down and visibility after being stopped). Additionally, timer can be set to start multiple times or run in the loop.

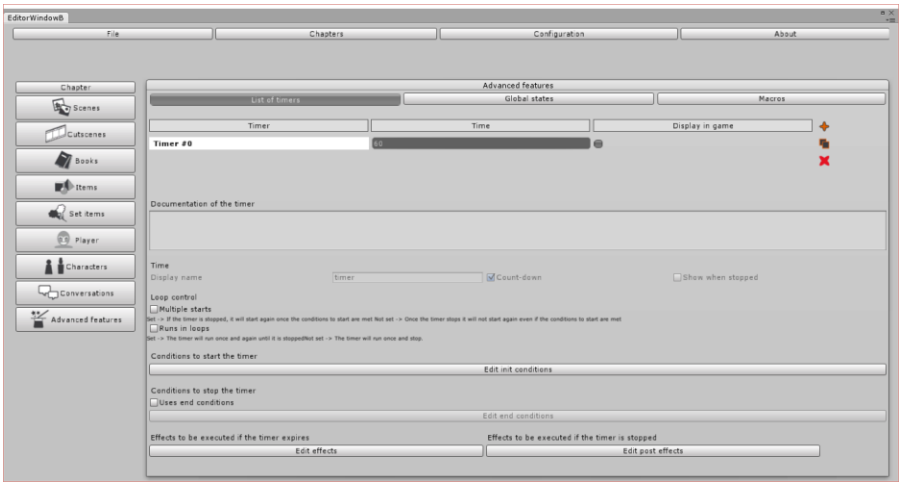

Figure 3-57 List of timers view

Global states

Global states view allows to add/delete/duplicate global state, which is set of conditions forming one entity. Their existence allows user to use them in different parts of the adventure (without logic duplication). Apart from documentation, modifiable is set of conditions.

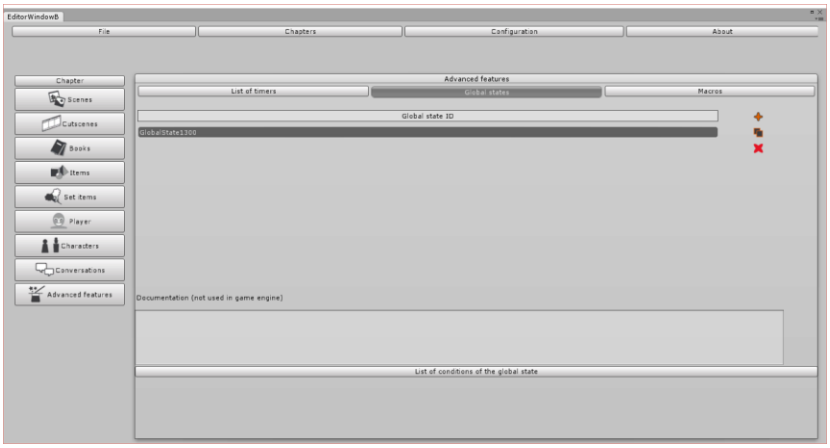

Figure 3-58 Global state view

Macros

Macro view is analogous to the Global states view, however, concerns effects instead of conditions. Macro is a set of effect formed into one entity, which allows to trigger the same block of effects without duplicating them. Apart from documentation, modifiable is set of effects.

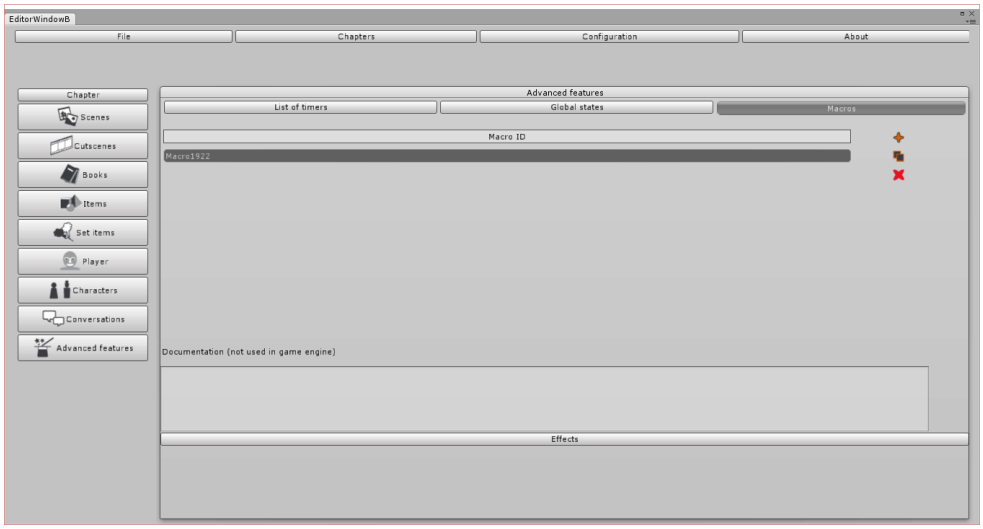

Figure 3-59 Macros view

Chapter 4 - Conclusions and Future work

Conclusion

The present report describes the authoring tool created on top of the Unity 3D™ in order to replace the eAdventure editor and serving as the first piece of the future uAdventure. The Unity 3D™ engine extension provides almost all of the functionalities available in a standalone version of the eAdventure editor, particularly the authoring of the assessment and adaptation profiles inside the authoring tool, because it will require a complete separate project. Note the that current version of the editor is able to open existing eAdventure game descriptions and analogous to standalone eAdventure way, make new adventures and modify existing ones, created both by eAdventure and uAdventure, which satisfies the main objectives of the project.

My work, together with emulator created by Iván José Pérez Colado, moves most of aspects of eAdventure to Unity. As a result, certain advantages of eAdventure tool have been extended by the possibilities offered by Unity (such as multiplatform building). I believe product, which is uAdventure, is a worthy continuation of the more than ten-year eAdventure history.

Future work

Further work on Unity editor extension could focus on creation of an alternative user interface targeted for advanced users of Unity engine. It could made possible to work directly on Unity engine objects, to which imported game logic (and content) are translated into (like Prefabs). Another aspect could be the support for using the native Unity engine components, such as animation editor. Working directly on game objects without using a proper editor, should reduce time needed to create games by professional Unity users, and consequently - reduce cost of the product.

Thanks to Unity ability to build the same game for multiple platforms, it makes sense to add a support for new games aspects available in mobile devices. Technologies such as QR code scanner, Bluetooth, NFC and GPS can be used as a basis component for construction of Location-based games. uAdventure extension in this case could be done by adding new types of actions associated with the right technology. With this type of solution, uAdventure would become a tool adapted to use for yet-another type of applications (probably the most accessible one), which certainly could have affected its popularity.

Like the standalone version of eAdventure, uAdventure at the moment does not support creating multilanguage games - the only solution to move game to another language is to change all the text content manually. This task may have be significantly simplified by additional tool, introducing an interface for the definition of the text in the selected group of languages. Adding language system is equal to increasing potential target audience with relatively small cost.

Chapter 5 - Conclusiones y trabajo futuro

Bibliography

del Blanco, Á., Torrente, J., Marchiori, E. J., Martínez-Ortiz, I., Moreno-Ger, P., & Fernández-Manjón, B. (2012). A framework for simplifying educator tasks related to the integration of games in the learning flow. *Educational Technology and Society*, *15*(4), 305–318. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84873851960&partnerID=tZOtx3y1

Moreno-Ger, P., Torrente, J., Bustamante, J., Fernández-Galaz, C., Fernández-Manjón, B., & Comas-Rengifo, M. D. (2010). Application of a low-cost web-based simulation to improve students' practical skills in medical education. *International Journal of Medical Informatics*, *79*(6), 459–67. http://doi.org/10.1016/j.ijmedinf.2010.01.017

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

Torrente, J., Del Blanco, Á., Marchiori, E. J., Moreno-Ger, P., & Fernández-Manjón, B. (2010). e-Adventure: Introducing Educational Games in the Learning Process. In *IEEE Education Engineering (EDUCON) 2010 Conference* (pp. 1121–1126). Madrid, Spain: IEEE.

eAdventure engine webpage, http://e-adventure.e-ucm.es/

Unity3d documentation, http://docs.unity3d.com/Manual/index.html