ENVISAGE

ENhance **VI**rtual learning **S**paces using **A**pplied **G**aming in **E**ducation

H2020-ICT-24-2016

D4.1 - Architecture and Interface Design

Dissemination level:	Public (PU)
Contractual date of delivery:	Month 5, Feb 28, 2017
Actual date of delivery:	Month 8, May 29, 2017
Workpackage:	WP4 - Virtual labs authoring tool
Task:	T4.1 - Architecture and interface design
Type:	Report
Approval Status:	Pre-final
Version:	Final
Number of pages:	53
Filename:	D4.1_DesignAndInterfaceArchitecture_Final.docx

Abstract

Relying on the functional requirements gathered in T1.2, we define the architecture of the "Virtual labs authoring tool" that will be capable of integrating the functionalities developed in WP2 and WP3 into a web interface component able to communicate with a game engine remotely in order to produce a lab. As regards the integration with WP2 and WP3, the adopted architecture should make provision for effectively injecting the necessary metrics in the virtual lab project to assess the user behavior during the experience of the virtual lab. The selection of the basic technologies will also be made. Another responsibility of this task is to analyze the functional requirements provided by WP1 in order to deliver an accurate interface design of the authoring tool suitable for the virtual labs. Mockups will be used to simulate the workflow between the supported functionalities and also act as a running exercise between the interface designers and the author-users in defining the look and feel of the authoring tool.

Co-funded by the European Union

Acknowledgment

This work is part of project ENVISAGE that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731900.

Copyright

History

Version	Date	Reason	Revised by
v0.1 (alpha)	23/3/2017	Initial draft with Table of Contents	Dimitrios Ververidis
v0.2	03/5/2017	First version	Dimitrios Ververidis
V0.3	05/5/2017	Incorporating visual analytics	Dimitrios Ververidis, Christoffer Holmgård
V0.4	12/05/2017	Incorporating tracking	Dimitrios Ververidis, Fabian Hadiji
V0.5 (beta)	19/05/2017	Beta version for internal review	Line Ebdrup Thomsen
V0.6	25/05/2017	Pre-final version	Dimitrios Ververidis
V0.7 (final)	29/05/2017	Final version	Spiros Nikolopoulo

Author list

Organization	Name	Contact Information
CERTH	Dimitrios Ververidis	ververid@iti.gr
CERTH	Stathis Nikolaidis	stathis.nikolaidis@iti.gr
CERTH	Spiros Nikolpoulos	nikolopo@iti.gr
CERTH	Ioannis Kompatsiaris	ikom@iti.gr
UoM	Christoffer Holmgård	holmgard@itu.dk
Goedle.io	Fabian Hadiji	fabian@goedle.io

Executive Summary

In this deliverable, we present the architecture of the virtual labs (games) authoring tool which is a web based application to create web based or desktop based virtual labs. The virtual labs authoring tool is a WordPress plugin that can be used by educators to create custom Unity3D games without writing any code. The plugin allows the educator to create game projects, scenes and assets by defining only their category and uploading the 3D models. The categorization of the entities allows the entity to inherit the necessary behavior from its category and thus hiding unnecessary details from the educators. The WordPress plugin visualizes game analytics that are already embedded into the games as object behaviours. This visualization allows the educators to improve the games accordingly in order to adapt to learners' needs.

Envisage approach is connecting two dominating technologies in two separate fields, namely WordPress for web based content management systems, and Unity3D for game authoring, in order to achieve a high quality solution for building virtual labs and games in general.

Abbreviations and Acronyms

CMS	Content Management System
API	Application Program Interface
GUI	Graphic User Interface
WebGL	Web Graphics Language
SDK	Software Development Toolkit
MAT	Unity3D Material file format
OBJ	Wavefront Object file format
MTL	Wavefront Material file format

Table of Contents

1. State of the art review for educational games authoring

In this deliverable, an architecture of the virtual labs authoring platform is defined that:

1. delivers high appealing and educational games for kids;
2. it is based on existing tools for game authoring so that we will not re-invent the wheel;
3. it will not become soon obsolete;
4. it meets the "Living lab" approach of D1.1 and D1.2 where the game analytics are used by teachers to improve the game;
5. it is user-friendly for the educator;
6. it is generalizable for many use cases.

These factors will be analyzed in the following and a decision for the architecture design and interface will be derived.

1.1 High Appealing educational games for kids

There are several learning games for kids available online, but after a while they are abandoned. The main reason is the lack of several features that most of the educational games online do not have, such as enjoyment, passionate involvement, structure, motivation, ego gratification, adrenaline, creativity, social interaction and emotion [Prensky '01].

Educational games can be both two dimensional and three dimensional. Two dimensional games have been extensively used in the past as a learning experience with several examples such as those found in Go-Labs[1], Photodentro[2], and Fun4thebrain[3]. These games involve puzzles, crosswords and multiple-choice questions by clicking or drag-n-drop actions. Three dimensional games offer a more rich experience where the student can be immersed into the educational task. Such examples are Second Life, Minecraft and Quest Atlantis[4]. Second Life has obtained a negative opinion from parents since it is highly addictive and it can expose kids to danger. Minecraft is often used in educational contexts but the minimalistic graphics do not offer a realistic representation of many problems that should be visible in real graphics. Such a case is Quest Atlantis, as shown in Figure 1.1, as it is an educational gaming environment for upper elementary and middle school kids, in lessons from science to social issues but it is abandoned in the last five years since it is costly to update and maintain.

In Envisage, the target is to build virtual labs, and therefore we will focus on 3D games as they offer a great level of immersion for virtual experiments and simulation. However, 3D games cost much more than 2D games because they are difficult to develop. These two

[1] http://go-lab-project.eu/
[2] http://photodentro.edu.gr/aggregator/
[3] www.fun4thebrain.com/
[4] http://atlantisremixed.org

aforementioned conclusions will lead us in the final architecture in the end of this section.

Figure 1.1: Atlantis Remixed was a 3D game specifically designed for education.

1.2 Existing game authoring tools

There are two categories of authoring tools for making educational games, namely those for 2D and those for 3D. As regards 2D games, there are several authoring tools targeting educational purposes, such as Manga High[5], which is one of the most advanced ones. Others

[5] https://www.mangahigh.com

can be found in Classroom Aid[6]. However, authoring tools specifically for making 3D educational games do not exist, as it is not commercially viable to make a game editor only for educational games. Therefore, learning game designers use mainstream authoring tools such as Unity3D to develop such games, as shown in Figure 1.2.

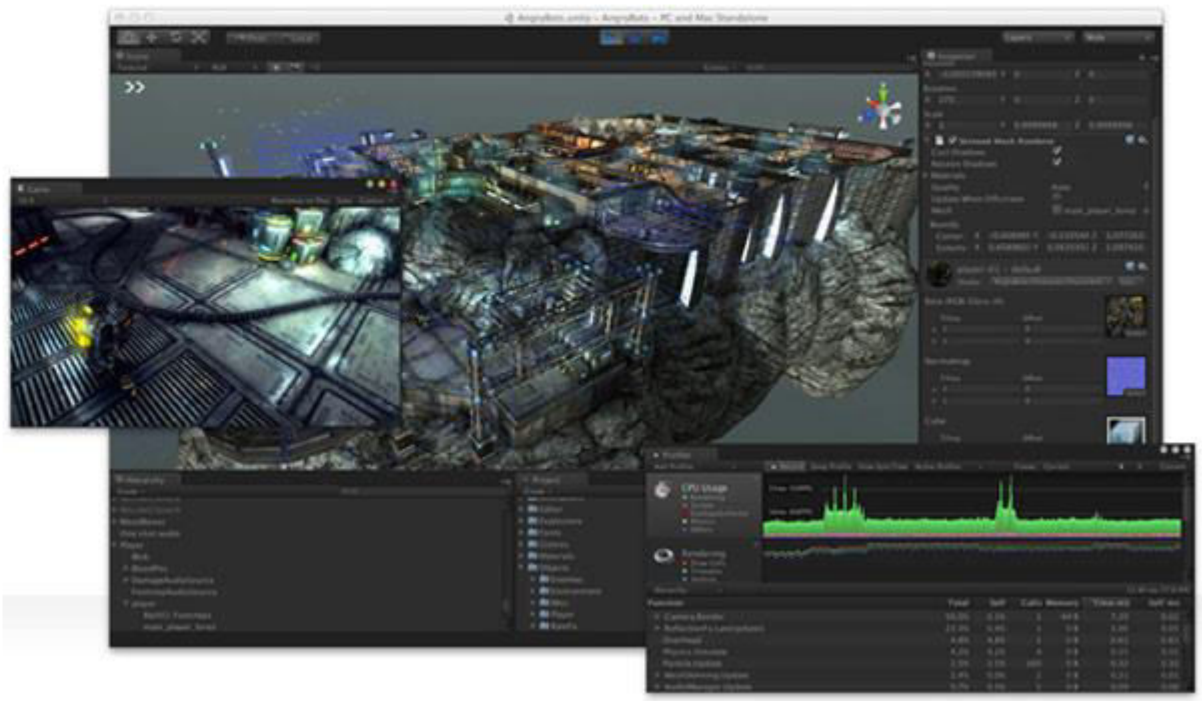


Figure 1.2: Unity3D environment for making 3D games.

Mainstream authoring tools for 3D games are desktop applications born at '90s and matured through several decades as games became popular across all ages. In Table 1.1, we have summarized the major authoring tools. The most popular desktop-game development tools are the Unity3D and Unreal Engine. Their major advantages is the export of games in binary formats for desktops, mobiles, consoles, and javascript for the web. As regards the javascript web format, the game is exported into javascript format either with a transformation tool called emscripten (Unity3D, Unreal Engine and Godot) or by incorporating a javascript framework (Copperlicht). Emscripten is a transpiler that transforms binary into machine-generated javascript code.

Unity3D and Unreal have plug-ins for incorporating multiple interfaces, such as Kinect, Oculus, LEAP motion and other types of sensors. Unity3D is open source but has a license that requires a fee when the income of the resulted game exceeds 100.000$. Something similar is valid for Unreal Engine. Another option is the Torgue3D gamemaker, being a completely open software with MIT license and quite mature. However, it has a very small community and it does not export the games into mobiles and consoles. Open source solutions such as Blender, Godot, and Copperlicht are immature, lightweight or lacking

[6] http://classroom-aid.com/play-and-learn/game-building/

export formats. The rest of the development tools such as CryEngine, GameMaker, Copperlicht, and ShiVa are closed source with a proprietary license.

Unity3D is the most widespread solution due to the community that has been built around it, its three decades in the market, due to its mixed policy of open source and proprietary license, and its market named as Asset store that allows external developers to sell or share custom plug-ins.

Table 1.1: Game authoring tools.

Authoring tool	Complier/ framework	GUI	License	Export game	Features	Url
Unity3D	Unity	Unity	Open but Proprietary if earnings > $100k	Desktop, Mobile, Web, Consoles	Leap, Kinect, Oculus ++	unity3d.com
Unreal Engine	Unreal	Unreal	Open but Proprietary if earnings >$12k	Desktop, Mobile, Web, Consoles	Leap, Kinect, Oculus ++	unrealengine.com
Torque3D	GFX	Torque3D	MIT	Desktop, Web	LEAP, Oculus, RazerHydra	garagegames.com
Blender	Blender	Blender	GPL	Desktop	Incomplete	blender.org
Blend4Web	Blender	Blender	Proprietary	Web	Plug-in for blender	Blend4web.com
Godot	Godot	Godot	MIT	Desktop, Mobile, Web	Lightweight	godotengine.org
Copperlicht	Copperlicht	Copperlicht	GPL	Web	No plug-ins for third party software	ambiera.com/copperlicht

Unity3D is selected for our architecture because it combines realistic graphics, wide community, commercial plan for viability, and many expansions to several interfaces. However, it can only be used by programmers because it is very complicated and requires programming and graphics language knowledge. However, not all teachers/educators are programmers; in fact, only a small part of them possesses the required skills. Therefore this problem should be addressed in our architecture.

1.3 Trendy Technologies

The technology that has become trendy the last 5 years is the WebGL, which allows 3D graphics to be natively rendered in Web browsers under the HTML5 standard. WebGL today replaces Java applets and Flash languages in gaming. WebGL has several pros and cons. The pros are that it is easily accessible because it is accessed from web browser, always updated, it is transparent as regards security issues (whereas Java and Flash games were not), and it does not affect the operating system since it is not installed anywhere. The cons is that its quality is inferior to desktop games, there are not many authoring tools, libraries or frameworks for WebGL games, and there is no dedicated physics engine as all authoring tools use a transpilled version of Bullet library for C++ named as canon.js or ammo.js.

Some web-based game authoring tools, libraries or frameworks are shown in Table 1.2. The oldest one with the biggest community is Three.js, which is a framework that provides higher level commands for 3D graphics. It is well documented but lacks a graphic tool. Microsoft's' Babylon.js is the second most popular framework. It also has a GUI with extensive capabilities which, however, targets for programmers. SuperpowersHTML5 is a new game development tool built on top of Three.js framework that has developed a GUI targeting programmers also. The GUI also supports real-time collaboration between programmers that allow for making a game faster. Other open technologies are XeoEngine and Turbulenz but they do not have a GUI. Commercial game development tools with advanced GUIs also targeting programmers are PlayCanvas, and Goo. From all these choices, Three.js seems to be the most suitable choice, as it is maintained by an open community and not a company as Babylon. SuperpowersHTML5 can be also considered for making a simplified GUI version on top of Three.js.

Table 1.2: WebGL tools-libraries-frameworks for making 3D games

Gamemaker	**Complier/ framework**	**GUI**	**License**	**Export game**	**Url**
Threejs	Three.js	Three.js Editor	MIT	Web	https://threejs.org/
Babylon	Babylon	Babylon Editor	ASL 2	Web	**https://www.babylonjs.com/**
SuperpowersHtml5	Three.js	SuperpowersHTML5	ISC (GPL like)	Web, Desktop	http://superpowers-html5.com/index.en.html
XeoEngine	XeoEngine (SceneJS)	No	MIT	Web	https://github.com/xeolabs/xeogl

Turbulenz	Turbulenz	No	MIT	Web	http://biz.turbulenz.com/
PlayCanvas	PlayCanvas	PlayCanvas	Proprietary	Web, iOS	https://playcanvas.com/
Goo	Goo	Goo Create	Proprietary	Web	https://learn.goocreate.com/

From our research we realized that existing WebGL frameworks are not yet mature for providing a high quality game, instead, it is better to rely on desktop game authoring tools to provide a transpilled game of a binary into WebGL. However, WebGL frameworks are suitable for making 3D level editors for remotely modifying desktop game projects because they are frameworks that allow a) to make a level editor customized for educators and b) to develop one solution for all teachers' computers because web-browsers exist to any desktop operating system, namely Mac, Windows or Linux.

1.4 "Living Lab" approach

Envisage is based on the "Living Lab" approach, namely expose the game to the students, gather analytics, modify the game, re-expose the game and repeat this cycle until a mature solution is reached. In order to allow such an approach, a web based game should be developed that is easily updated without any need for re-installation. Therefore WebGL should be selected as a technology for making games.

The "Living Lab" approach requires the gathering of data (shallow analytics) and the process of the data with machine learning methods (deep analytics). Shallow analytics require the injection of metrics functions that track users' actions in the game and submit the data to a dedicated server. The metrics functions should be inside the WebGL javascript code. However, desktop game authoring tools, that we are going to employ for providing high quality games, produce a transpiled version of WebGL javascript that it is machine generated and not understood by humans. Therefore, the metrics functions should be installed before the transpile, namely when building the project in C++ or Javascript (Unity3D and Unreal support coding in both languages). Further details about the injection of metrics functions can be found in Section 4.

As regards deep analytics, i.e. the statistical process and the visualization of metrics data should be presented to teachers in order to improve the games, it should be programmed a) in a web based language as regards the visualization and b) a server based language as regards the statistical process of the data. Web based visualizations are suitable because they can be accessed easily by teachers from a web-browser. A suitable web-based visualizations technology is HTML-Javascript, as it will be further discussed in Section 5. Statistical process of the data should be done in Server side since it involves complicated numerical functions and machine learning. Suitable technology for machine learning in server side is Python language because it has components that allow numerical functions (numpy), machine learning functions (scipy) and even deep-learning functions (TensorFlow).

1.5 User friendly for the teacher

The game authoring tool should be user-friendly to the teacher. Desktop-based game authoring tools (Unity3D, Unreal Engine), as it was previously discussed, are difficult to master for teachers because they mainly target programmers. Envisage therefore, needs to develop a game authoring tool that:

1. is web based in order to be easily accessible and portable to many operating systems
2. hides unnecessary technical details from teachers, but simultaneously being functional,
3. stay within the budget of Envisage project
4. find track into an existing community hoping to attract the general interest of game developers.

In order to fulfill the aforementioned requirements, we need to rely to an existing web-based Content Management System (CMS) that provides the main web functionalities such as security, user management, content management, uploading functionalities, etc. To address this need, two options exist, either to use a general purpose CMS where WordPress is dominating with 60% of popularity among all websites globally, or use a educational focused CMS (learning CMS, LCMS) where Moodle is dominating. WordPress has a greater variety of plugins, a wider community of users and developers, and it is born in 2010. Moodle started in 2001, it has a limited community of developers, but it is very popular among teachers and students. We have decided to select WordPress for building our game authoring tool, as a plugin in WordPress terminology, because a) WordPress also has 3rd party plugins that allow it to become a LCMS (e.g. DashLearing for WordPress plugin), b) WordPress can find track also to game authors but not only for teachers; and c) Moodle technology looks obsolete as regards modern web pages.

1.6 Generic authoring tool

The scope of Envisage is to provide a game authoring tool that is generalizable for many use cases, which will help towards its commercialization. Therefore, Envisage will be based on game templates where each template allows to build several games for a specific subject e.g. chemistry, physics, or renewable energy. The templates allow to hide technical details from the teachers because the teacher has only to select a category for an item, so as the item to automatically inherit all the behaviors of its category. An example could be, if for a 3D object the selected category is “Terrain”, then the 3D object becomes a fixed ground where items stay on top of it. In order to develop the template, an original game is studied, optimized, and the Unity3D code is split into pieces. Unity3D employs the YAML (Yet Another Markup Language) format, which is a kind of XML. Each piece of YAML (snippet) is modified in order to have variables instead of fixed values at certain positions. Therefore, each game object is actually an instance of the YAML snippet with different variable values.

2. Requirements from the use case scenario

The requirements of the use cases scenarios as described in D1.1 and D1.2, are reviewed here from a technical aspect that will determine the architecture of the virtual labs game authoring tool. Envisage, will develop a virtual labs game authoring tool has the ability to cover several lab cases through respective game templates, and allow the educators to generate an arbitrary number of game instances at multiple setups focusing on customly defined goals. The game templates allow the abstraction and generalization of the developed authoring tool. However, for the needs of defining the architectural design, we focus on a certain game template that will be used for generating games related to renewable sources of energy, which is currently named as "Wind Energy Simulation" lab. The virtual labs game authoring tool, produces games that have a certain structure in their "Main Menu" which is described next.

2.1 Shared high-level structure across generated games

All generated games should follow a certain formalism in the Main Menu so that the generalization is possible through the developed authoring tool. The following formalism is popular in the majority of the modern games and it will be adopted in our architecture.

All games should have an entry scene, named as "Main Menu", which is the central point of the game where the learner can select what to do next by the means of buttons. A mockup of the "Main Menu" scene can be seen in Figure 2.1.

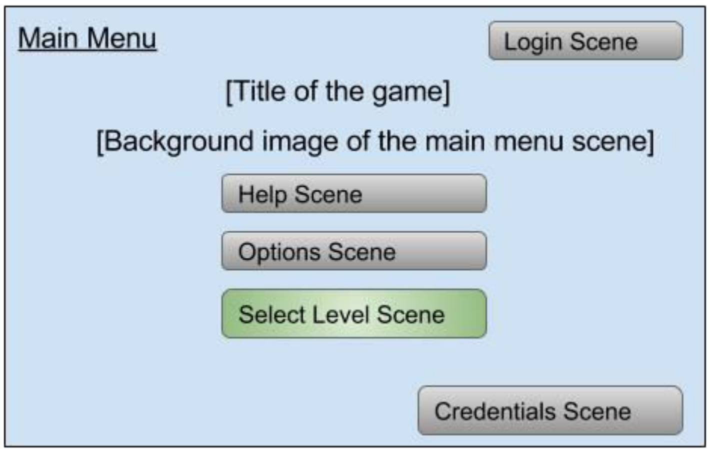


Figure 2.1: Main menu scene organization.

The Main Menu should have "Help", "Options", "Login", "Credits", and "Select Level" scenes for the following reasons.

1) **Help** scene: The educators should be able to put the learning material in this scene in order to guide the learners. The time spent by the learner in this scene should be tracked.
2) **Options** scene: In this scene learners could be able to change game generic

parameters such as detail level so that the game can run smoothly on low-end devices.

3) **Login** scene will allow to provide learner input such as name, surname, school etc in order to improve the game analytics
4) **Credits** view: The organization that developed the game should be acknowledged.
5) **Select Level** scene will allow the learner to select an Educational Scene for playing. This is described next.

The games should have an arbitrary number of "Educational Scenes" focusing on each educational goal as defined by the educators, and according to D1.1, the time spent in each Educational Scene should be tracked. Figure 2.2 depicts how "Select Level" scene will be implemented. Each Educational Scene is represented by a tile with a featured image and the name of the scene. The scenes are presented in an ascending order from left to right, and a slider will exist to allow to see and select scenes that escape from the screen limits. The educator will have the option to lock successive scenes, so that the learner should pass from the previous scene in order to unlock the next one.

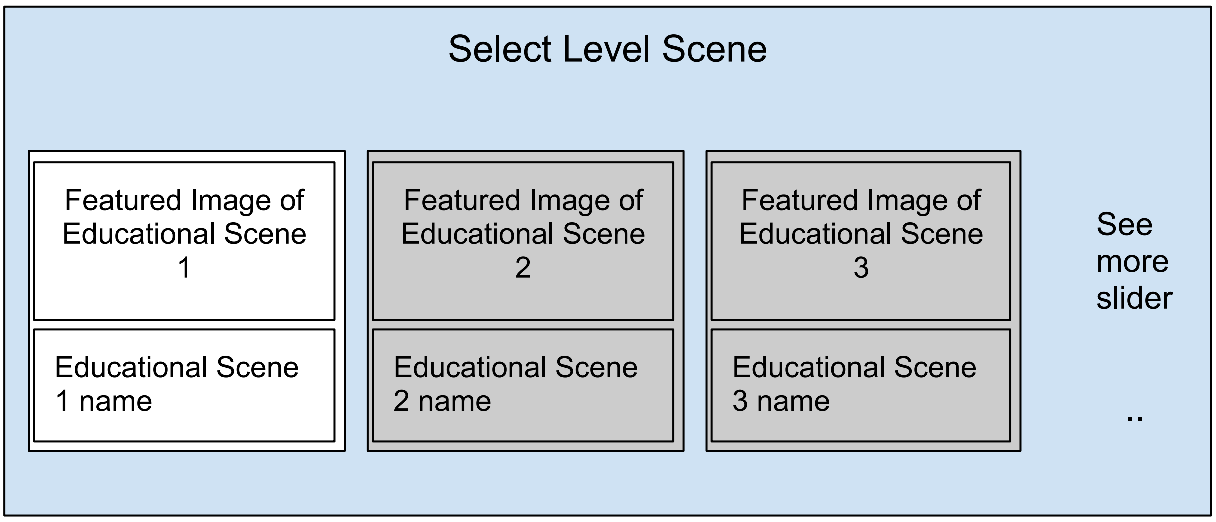

Select Level Scene
Featured Image of Educational Scene 1
Educational Scene 1 name
Featured Image of Educational Scene 2
Educational Scene 2 name
Featured Image of Educational Scene 3
Educational Scene 3 name
See more slider
..

Figure 2.2: Select level scene allows the learner to play a certain scene.

As regards the virtual labs game authoring tool, the overall prerequisite is that all games could be created and edited from the web browser, and should be compiled to be played also from the web browser without the need of downloading and installation. Next, the content of an Educational Scene is described.

2.2 Content requirements and technical specifications inferred from the "Wind Energy Simulation" lab

A certain lab, namely the simulation of production-consumption of electrical power in an urban area was selected as a prototype in order to examine the game requirements; to develop a Unity3D game that can be extended; and for allowing us to estimate the abstraction that should be made in order to develop the authoring tool.

The target of the simulation is to adapt the energy production to the energy consumption, where both vary over time. An image of a prototype virtual lab is shown in Figure 2.3.

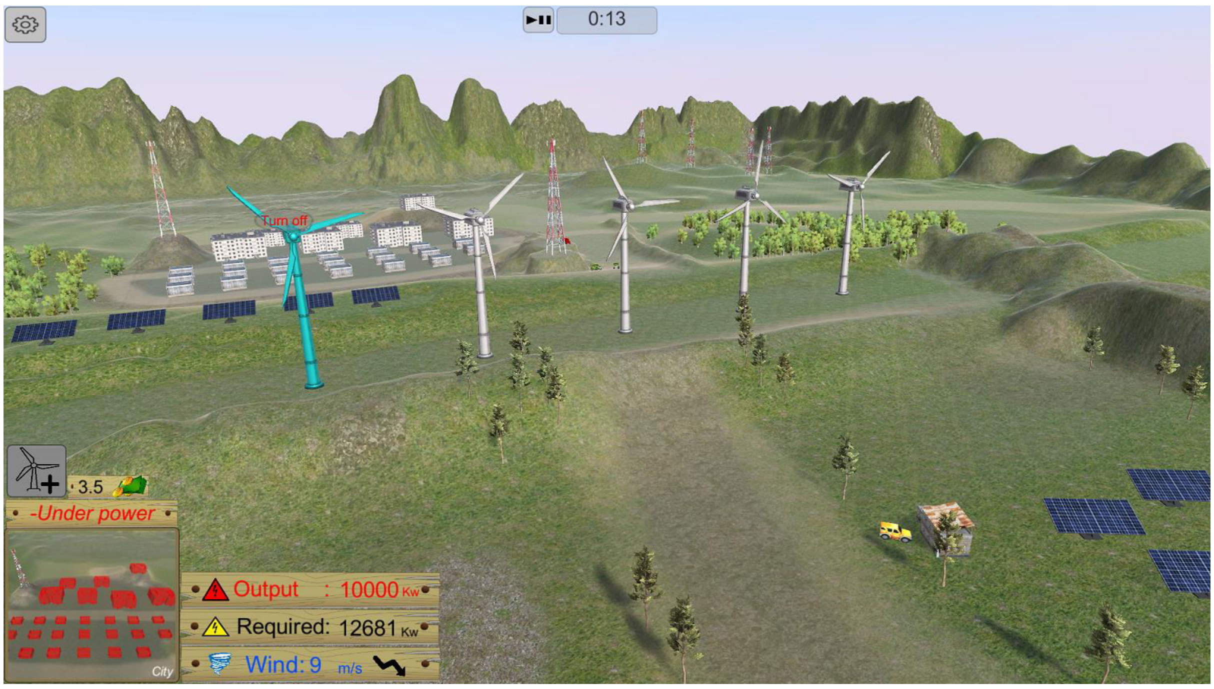


Figure 2.3: Prototype of virtual Lab for learning about renewable sources of energy.

The learner actions in this Educational Scene are as follows

a) the learner can change the number of wind turbines by pressing the plus button in left side of the game. This action should be tracked in order to investigate when the user starts interacting with the game and when each turbine is placed.
b) the learner can turn-off a turbine by clicking on it when the power generation greater than the consumption. This action should be tracked for allowing to investigate the learner inference capability.
c) the learner can change the air speed variability (upper-lower, limit) and the city consumption (upper-lower limit) through the gear icon on the left. This action should be tracked in order to see that the learner runs the simulation in various conditions.
d) the learner can repair a turbine if the turbine outputs smoke by clicking on it.

Visualizations

e) the terrain, the wind-turbines, and the city are 3D models
f) the user can orbit around the scene to see the turbines from all sides
g) energy production-consumption and wind speed are provided in the lower-left panel
h) the city is depicted in an overlay map in the left corner that displays the buildings in red when the power from wind-turbines is not enough, green if it is in balance, or blue if it is overpowered.

Game rules and rewarding

i) if the energy need and production are in equilibrium the learner earns some virtual coins. If the user repairs a turbine, then loses some virtual coins. Every 5 seconds, the status of the game should be reported which is a vector consisting of

1) wind speed
2) energy production
3) energy consumption
4) turbines places
5) turbines set off
6) money owned

In a second type of educational scenes, the configuration of wind speed limits and energy consumption limits will be forbidden to change. The wind energy profile and the consumption profile will be set a priori from the educator based on the needs of an actual area.

In a third type of educational scenes, the configuration of wind speed limits and energy consumption limits will be adapted automatically based on the virtual coins earned (Dynamic Difficulty Adjustment). If the coins are too much then the limits will be moved to a position to increase the difficulty to earn coins, and vice versa.

2.3 What the game authoring tool should be able to change.

Based on the prototype, we found certain entities that should be isolated and customly defined by the game authoring tool. These are outlined in Table 2.1. Namely, energy can be generated from "Windmills", "Solar panels", and "Hydro-electric power plants" whereas it is consumed from "City buildings". The landscape is formed by "Land" and "Water" objects.

Table 2.1: Varying entities in a "Energy - Simulation" game.

Asset3D type	Description	Properties
Wind mill	A 3D object that can - generate energy - have animation of propellers - can be damaged and repaired	- 3D model - Energy production at various air speed conditions - damage probability - repair cost
Solar panel	A 3D object that can - generate energy - can be damaged and repaired - have animation to point at sun direction	- 3D model - Energy production at various sun-temperature conditions - damage probability - repair cost
Hydro-electric power plant	A 3D object that can - generate energy - can be damaged and repaired	- 3D model - Energy production at water flow conditions - damage probability - repair cost

City building	A 3D object that - consumes energy	- 3D model - Energy consumption with respect to time
Land	A 3D object that - it serves as the landscape	- 3D model - wind flow deterioration depending on the landscape shape
Water	A 3D object that - represents water	- 3D model - water flow coefficient with respect to time

The game authoring tool should allow the educator the following actions during the process of game design :

1. to customize the profiles of the wind flow, water flow, cloud-ness and temperature so that a certain scene can obtain the necessary profiles to simulate a year of energy production;
2. enable to the educator to add multiple scenes within one game, where each scene represents a certain place on Earth;
3. the game should have a priori 5 fixed scenes such as “Main Menu”, “Credits”, “Login”, “Help” and “Options” where the only thing that can be changed by the educator is the text language, the position of the buttons, and some featured images;
4. to place instances of the aforementioned game objects such as “Wind mill”, “Solar panel”, “Land” etc. to certain places so that a custom scene is defined. This should be achieved with a drag-n-drop GUI with a 3D gizmos (graphic help elements);

Other properties of the game authoring tool are as follows.

5. the generated games should by definition have game tracking code as defined in Section 2.1;
6. The energy generators should have a on/off switch by their definition
7. the game authoring tool should also allow the visualization of game analytics that will help educators to design a better game;
8. the game authoring tool should allow for the remote compiling of the game and to provide to the educators the web or file link of the game in order to disseminate it.

2.4 Hardware specifications

The games should be able to run in a personal computer with a web browser such as “Chrome”, “Firefox” etc or they should be able to be downloaded for a local installation as binaries for a better experience. As games target in elementary and secondary school children, the hardware recommendations should be for middle-end devices. However, the games should provide the option to deteriorate the detail level in order for the game to run smoothly on low-end personal computers. A graphics card (GPU) is recommended. In detail the software-hardware minimum recommendations are:

a) PC Operation System: Windows 7, Linux, or Mac
b) CPU: i5 processor of first generation or any equivalent
c) RAM: 4 GB
d) Web browser: Chrome, Firefox, Edge

Optimum specifications are:

e) CPU: i5 processor of 3rd generation or any equivalent
f) RAM: 8 GB

3. Architecture and interfaces design

3.1 Introduction to the implementation pipeline

We seek an architecture that will serve as the skeleton where all the necessary features will be added. We have a strong emphasis on developing a low complexity, easily modifiable, but linear evolving implementation that will not be limited in the future to its design and will allow to cover whatever feature will be needed a posteriori. A simplistic overview of the architecture is shown in Figure 3.1.

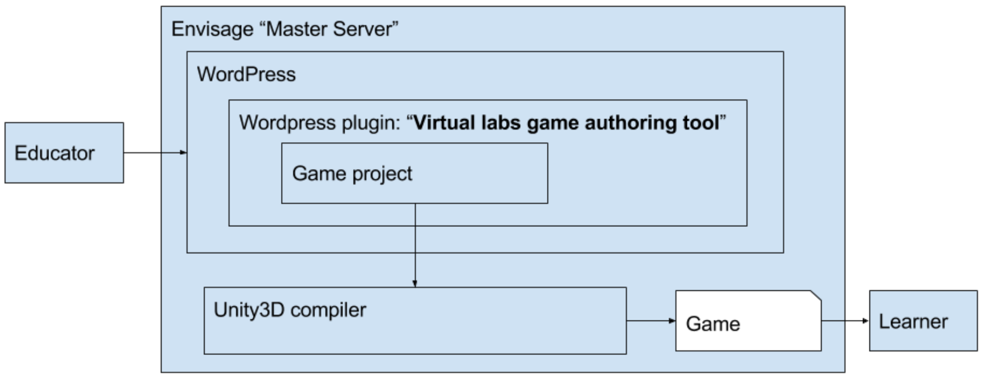


Figure 3.1: virtual Labs game authoring tool is a WordPress plugin.

Envisage "Master Server" is a standard personal computer with the following software:

a) Windows or Ubuntu operation system,
b) an http server such as Apache server,
c) a MySQL database server,
d) WordPress CMS, in order to generate automatically a Web portal and a MySQL database schema that will allow for user and multimedia management (register, upload, edit, view etc.),
e) Unity3D that offers the compiler for compiling game projects from Unity3D project format (YAML) to WebGL format or a binary format (Mac, Windows, Android etc.).

The aforementioned software, apart from Windows, is provided free. Unity3D is provided free or under a commercial license as discussed in Section 1.

The main product of Envisage is the WordPress plugin named as "**virtual labs game authoring tool**". The plugin will be able to generate game projects, based on templates of code that can be compiled from Unity3D compiler. Secondarily, game analytics technologies are going to be incorporated in order to achieve a feedback mechanism for the educators. More details are described next.

3.2 Architecture design

The overall architecture of the plugin is shown in greater detail in Figure 3.2. The backbone of the system is three servers, namely:

a) the “Master Server” that contains the functionalities for authoring games, i.e. the “Virtual labs game authoring tool”.
b) the “Shallow Analytics Server” that collects-stores-aggregates-augments raw game analytics
c) the “Deep Analytics Server” that process the game analytics of the “Shallow Analytics Server” and ports the results to the “Master Server” as feedback for educators.

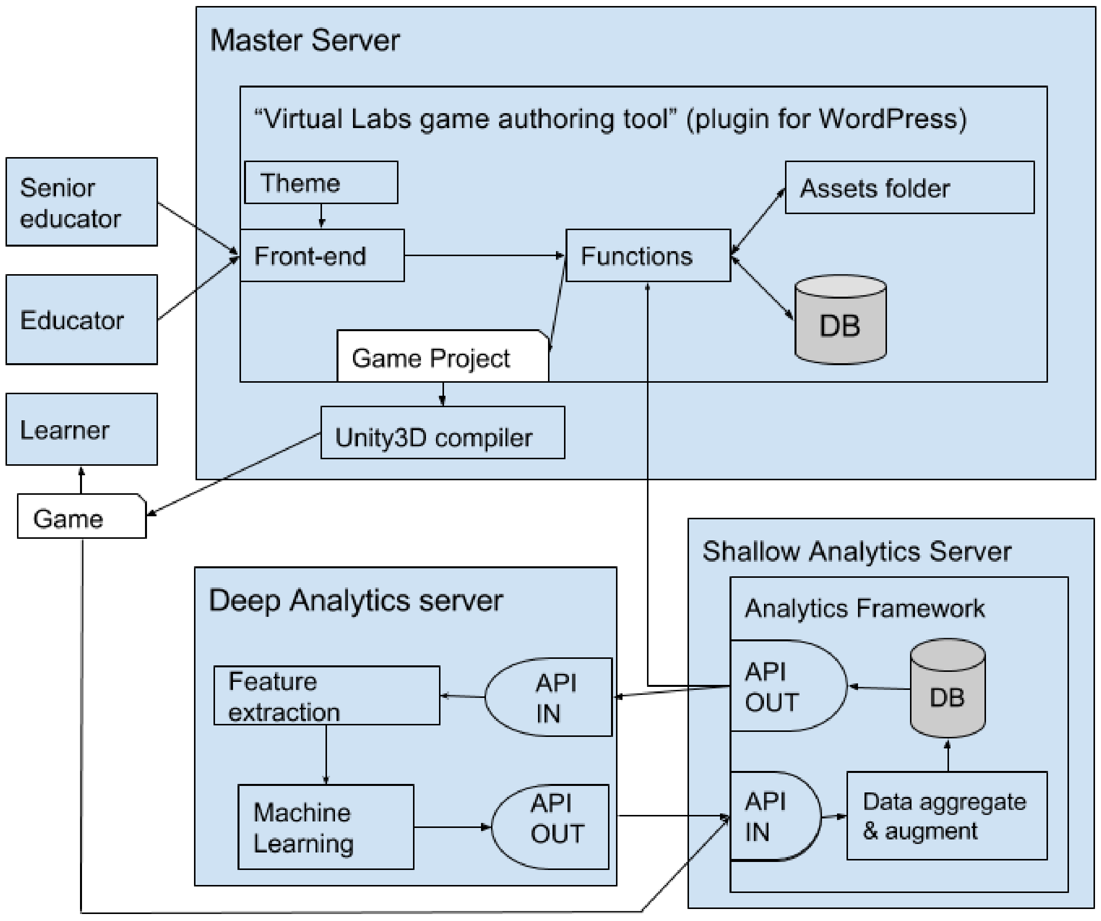


Figure 3.2: Second diagram of the architecture that displays logical components in greater detail.

In the Master Server, the Developed Plugin is installed under a WordPress CMS framework. The Developed Plugin has the following main functionalities:

1) provide a **front-end** web interface that allows the educators to login, create or edit a game. This front-end is shaped through a developed **theme** that provides the required usability level. There is also a back-end, which however is used only by programmers for testing and debugging purposes because it does not have the

required level of usability;
2) manage the logical entities of the games such as Games, Scenes, and Assets 3D. These entities are stored in the MySQL database schema or in an assets folder if there are big 3D files;
3) assembly the Unity3D game project (YAML) that can be compiled from Unity3D game compiler;
4) provide the game to the learners either as a link if it is compiled for web or as a zipped file if it is compiled for desktop use.

The "**Shallow Analytics Server**" provides the following functionalities:

1) provide an API to receive analytics data from games (API IN). The compiled games will already have metrics functionalities embedded from the game template and they will be automatically connected with the API IN through HTTP protocol. Also the API IN allows for processed Deep Learning Analytics to be received and stored in the Shallow Analytics Server;
2) aggregate and augment raw data (see Section 4);
3) store analytics data in a database schema;
4) provide an API for sending analytics data to Deep Analytics Server or to Master server (API OUT).

The "**Deep Analytics Server**" provides the following functionalities:

1) provide an API for receiving analytics data from Shallow Analytics Server (API IN);
2) estimate features on analytics data that have meaningful information;
3) process features with machine learning techniques in order to extract a meaningful pattern;
4) provide an API for sending processed user analytics data to the Shallow Analytics Server in order to be stored. These analytics will be used by the educator, through proper visualizations, to modify the game.

A typical scenario is as follows: a senior educator enters the front-end of the Master Server and creates a new game project with some empty scenes and uploads the necessary assets of the scene. Then edits the scenes, e.g. by placing certain assets to certain positions. The game project is then compiled to a game and it is disseminated to the learners. After some time, a simple educator enters the front-end, examines the games analytics and accordingly makes the necessary changes in order to improve the game. The difference between the senior educator and the simple educator is that the simple educator does not have to upload any scene asset as all assets will be already uploaded. The game is then compiled again and a new version is disseminated. This cycle goes on until the game achieves the educational target.

3.3 Definition of game entities

In this section, the entities of a game in the virtual labs game authoring tool will be described. In order to define entities in WordPress, we have to understand both WordPress and Unity3D mechanisms. WordPress is based on the following principles that are depicted graphically in Figure 3.3, namely:

1. The main structure that stores information in WordPress is a Custom Post Type, or briefly CPT.
2. Each CPT has customly defined fields and fields' respective values that are called metadata fields and values, respectively.
3. The main structure that stores categorical information in WordPress is called taxonomy.
4. Each taxonomy is a set of customly defined values that are called taxonomy terms.
5. Each CPT can have customly defined taxonomies and taxonomies respective terms.
6. Each taxonomy term can have also customly defined fields and fields' values that are called taxonomy metadata and values, respectively.

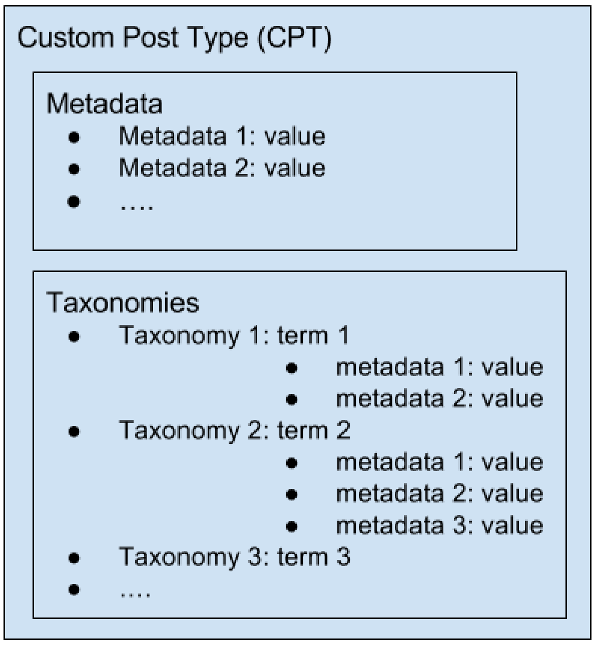


Figure 3.3: WordPress main structure for saving data.

In order to define game entities in WordPress we should also understand Unity3D mechanism, which is shown Graphically in Figure 3.4, and described in the following:

1. A Game Project is a set of Scenes, Assets, and Game Project Settings.
2. A Scene is a set of Game Objects and some Scene Settings.
3. A Game Object is an instance of an Asset 3D (a 3D model). A Game Object has Components with Properties that define its behavior.
4. Several Assets 3D are available to be instantiated as Game Objects in a Scene. However, an Asset 3D may not be instantiated to any Scene, but still exist as a resource in the Game Project for future use.

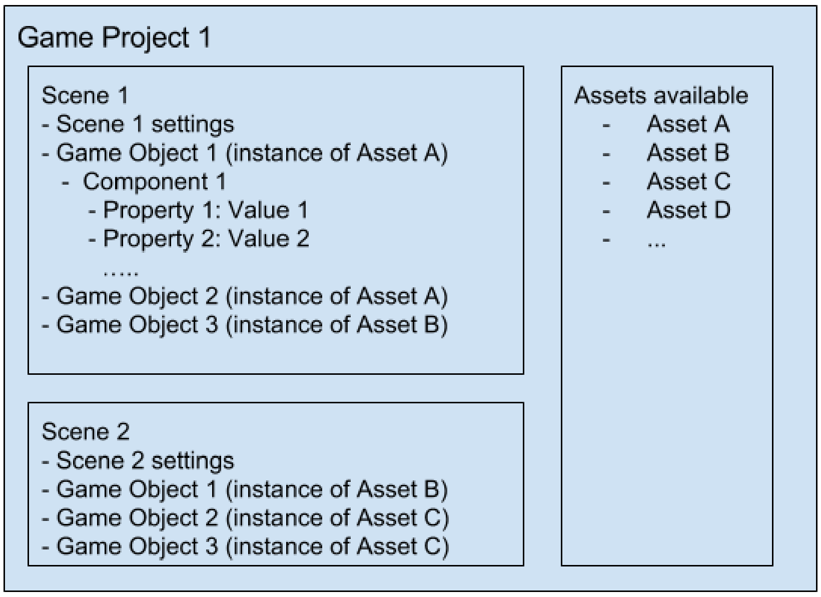

Game Project 1
Scene 1
- Scene 1 settings
- Game Object 1 (instance of Asset A)
- Component 1
- Property 1: Value 1
- Property 2: Value 2
.....
- Game Object 2 (instance of Asset A)
- Game Object 3 (instance of Asset B)
Scene 2
- Scene 2 settings
- Game Object 1 (instance of Asset B)
- Game Object 2 (instance of Asset C)
- Game Object 3 (instance of Asset C)
Assets available
- Asset A
- Asset B
- Asset C
- Asset D
- ...

Figure 3.4: Structure of a typical Unity3D game.

In order to match Unity3D entities to WordPress entities, we define 3 CPTs, namely CPT Game Project, CPT Scene, and CPT Asset3D. All these CPTs have custom metadata and custom taxonomies. Metadata are used for storing field information such as the color of a 3d model, whereas taxonomies are used for defining the behavior of the CPT inside the game, e.g. "Terrain". In Table 3.1, we present the matching between Unity3D entities and WordPress entities.

Table 3.1: Matching WordPress entities with Unity3D entities

#	Unity3D Entities	WordPress Entities
1	Game Project Example: "EA Wind Energy Lab" project	CPT Game with taxonomy "Game Type" Example: - "EA Wind Energy Lab" is an instance of CPT Game. - "EA Wind Energy Lab" should have for "Game Type" taxonomy the term "Wind Energy Lab" selected.
2	Scene Example: "Educational Scene 1"	CPT Scene with taxonomy "Scene Type" Examples: - "Educational Scene 1" is an instance of CPT Scene - "Educational Scene 1" should have for "Scene Type" taxonomy the term "Educational scene" selected.
3	Asset	CPT Asset3D

	Example: “Wind Turbine”	with taxonomy “Asset3D Type” Example: - “Wind Turbine” is an instance of CPT Asset3D - “Wind Turbine” should have for “Asset3D Type” the term “Energy Generator” selected.
4	Game Object Example: “Wind Turbine 15”	An instance of CPT Asset3D Example : - “Wind Turbine 15” is an instance of CPT Asset3D “Wind Turbine”
5	Game Properties	Are defined from “Game Type” taxonomy term metadata values, e.g. 3D vs 2D game property
6	Scene Properties	Are defined from “Scene Type” taxonomy term metadata values, e.g. fog, occlusion settings, rendering settings
7	Game Object properties	Are defined from “Game Type” taxonomy term metadata values, e.g. hasGravity= false

The logical connection of the entities is shown in Figure 3.5. The Games Projects are divided by “Game Project Types”, the Scenes by “Scene Types”, and the Assets3D by “Assets Types”. The scene’s objects 3D setup (positions, rotations, scalings) are saved in a json file of our own protocol so that we can store in the MySQL database the scene. So, ...

1. each game project that belongs to a certain Game Project Type has scenes of certain Scene Types,
2. each scene that belongs to a certain Scene Type has Assets3D of certain Asset3D Types,
3. each Game Object in a Scene is an instance of an Asset3D, and
4. each Scene CPT has a metadata field where the scene 3D setup of the Game Objects is saved in a simple json format.

For example, the structure for the “EA Wind Energy Lab” project is as follows:

1. “EA Wind Energy Lab” is an instance of Games Project CPT with taxonomy “Wind Energy Lab”
2. “Educational Scene 1” is an instance of Scenes CPT with taxonomy “Educational Scene”
3. “Wind Turbine” and “Solar Panel” are instances of Assets3D CPT with taxonomy “Energy Generator”
4. “Solar Panel 1” is an instance of the Asset3D instance “Solar Panel”
5. “Wind Turbine 1” and “Wind Turbine 2” are instances of Asset3D instance “Wind Turbine”

6. The “Educational Scene 1 json file” is a text file that it is saved in “Educational Scene 1” metadata field named as “Scene json”.

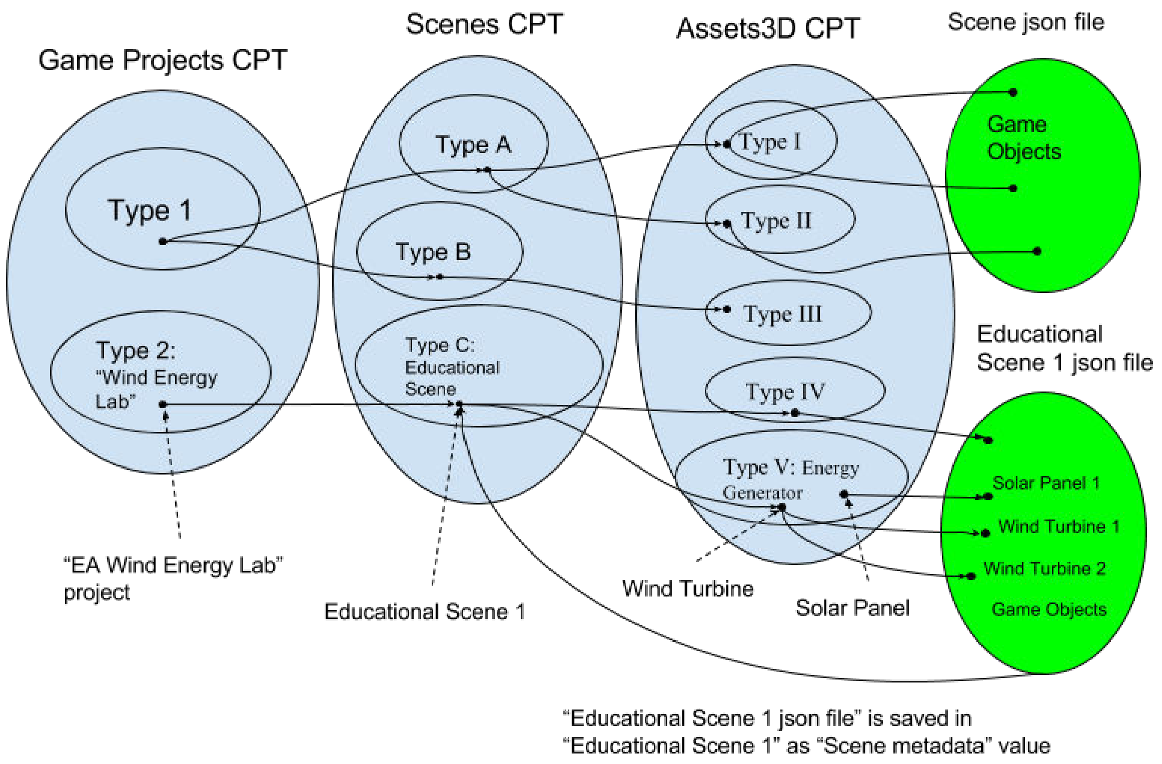


Figure 3.5: Adopted logical connection between Game Projects, Scenes and Asset3D CPTs.

The currently available “Type” values are explained in Table 3.2. A Game Project has the taxonomy **“Game Project Type”** which defines what kind of game will be developed. For the selected case, the game project type takes the value *“Wind-Energy-Lab”*. The metadata fields of the game project are title, description, longitude and latitude.

A Scene is a CPT that stores the game objects, namely their id, position, rotation, and scale. Each game object is an instance of an Asset3D. The taxonomy of the Scene is named as **“Scene Type”** and defines what type of scene will be developed. For the selected game, the scene type value is *“Isometric”* and refers to the kind of games that are viewed from above the ground, which are also called 3rd person view or bird’s eye view games. Each taxonomy value also has many fields where the template information is stored, i.e. pieces of Unity3D code that will be described in Section 3.5.

An Asset3D is a CPT that has metadata fields such as title, description, obj (3D file), mtl (material file), jpgs (texture files), and others depending on the taxonomy values of the Asset3D. The taxonomy values terms also define the behavior of the asset in the game as they contain YAML template code.

Table 3.2: Taxonomies and metadata fields for each CPT.

CPT	Taxonomies	Metadata fields
Game Project	**"Game Project Type"**: a taxonomy that defines what kind of game will be made. For the case of the game examined it can take the value "*Wind-Energy-Lab*"	**"Title":** The title of the game **"Description"**: A description of the game (optional). **"Latitude"**: location latitude (optional) **"Longitude":** location longitude (optional)
Scene	**"Scene Type"**: a taxonomy that defines the kind of the scene that will be made. For the case of the game examined, it can take the value *"Isometric".*	**"Title"**: The title of the scene **"Description"**: A description of the scene (optional) **"Scene json":** The scene is saved in a custom json format for storing ids, position, rotation, and scale of each object in the scene (see Section 3.5). **"Longitude"**: location longitude **"Latitude"**: location latitude **"Wind profile":** a function defining the wind direction, and intensity over a year **"Cloud cover profile":** a function defining the presence of clouds in the area over a year **"Temperature profile":** a function defining the temperature level over a year.
Asset3D	**"Asset3D Type":** A taxonomy that defines 1. what fields this asset will have 2. which template code to use in order to construct an instance of this asset when inserting it into the scene Asset3D type values for an "Isometric" scene of the "Wind-Energy-Simulation" game are: *a) Land terrain* *b) Water*	<u>Standard fields</u> **"Title":** the title of the asset **"Description":** the description of the asset <u>Varying fields depending on Asset3D type taxonomy</u> a) Land terrain 1) **Obj file**: the mesh of the asset 2) **Mtl file**: the material definitio 3) **Jpg file**: mesh texture (multiple jpgs allowed) b) Water 1) **obj file**: Mesh of water c) Consumer of energy

	c) *Consumer of Energy* (e.g. a town or village) *d) Generator of Energy* (wind turbine, solar panel, hydro-electric power plant) *e) Point-of-Interest (POI) Image-text* *f) POI Video*	1) **Obj file**: mesh 2) **Mtl file**: material 3) **Jpg texture file** 4) **Consumption profile over a year**: a function that defines how much energy this item consumes over time. d) Generator of energy 1) **Obj** file 2) **Mtl** file 3) **jpg** texture file 4) **Energy production with respect to air speed** 5) **Energy production with respect to solar radiation** 6) **Energy production with respect to water flow**

3.4 Front-end interface

The front-end interface is GUIs that allow a user-friendly management of Game Projects, Scenes, and Assets3D, targeting towards novice users with limited, or no, knowledge of game authoring processes. The mockups and the first implementations for the GUI will be discussed in the following.

The overall organization of the GUI is presented in Figure 3.6. Briefly, the user enters Screen 1 after successful login, which is the *Game Project Manager* that can be used to create or edit an existing game. Next, the user enters Screen 2 which is the *Scene Manager*, which can be used to create or edit an existing scene or compile the full game. If the user selects to edit a scene, then Screen 3 appears, which is the *Scene Editor* screen. If the user selects to create a new scene, then Screen 4 appears, which is the *Scene Creator*. If the user is in Screen 3, and wants to upload a new game asset, then Screen 5 appears which is the Assets3D Manager. These screens are described in greater detail in the following.

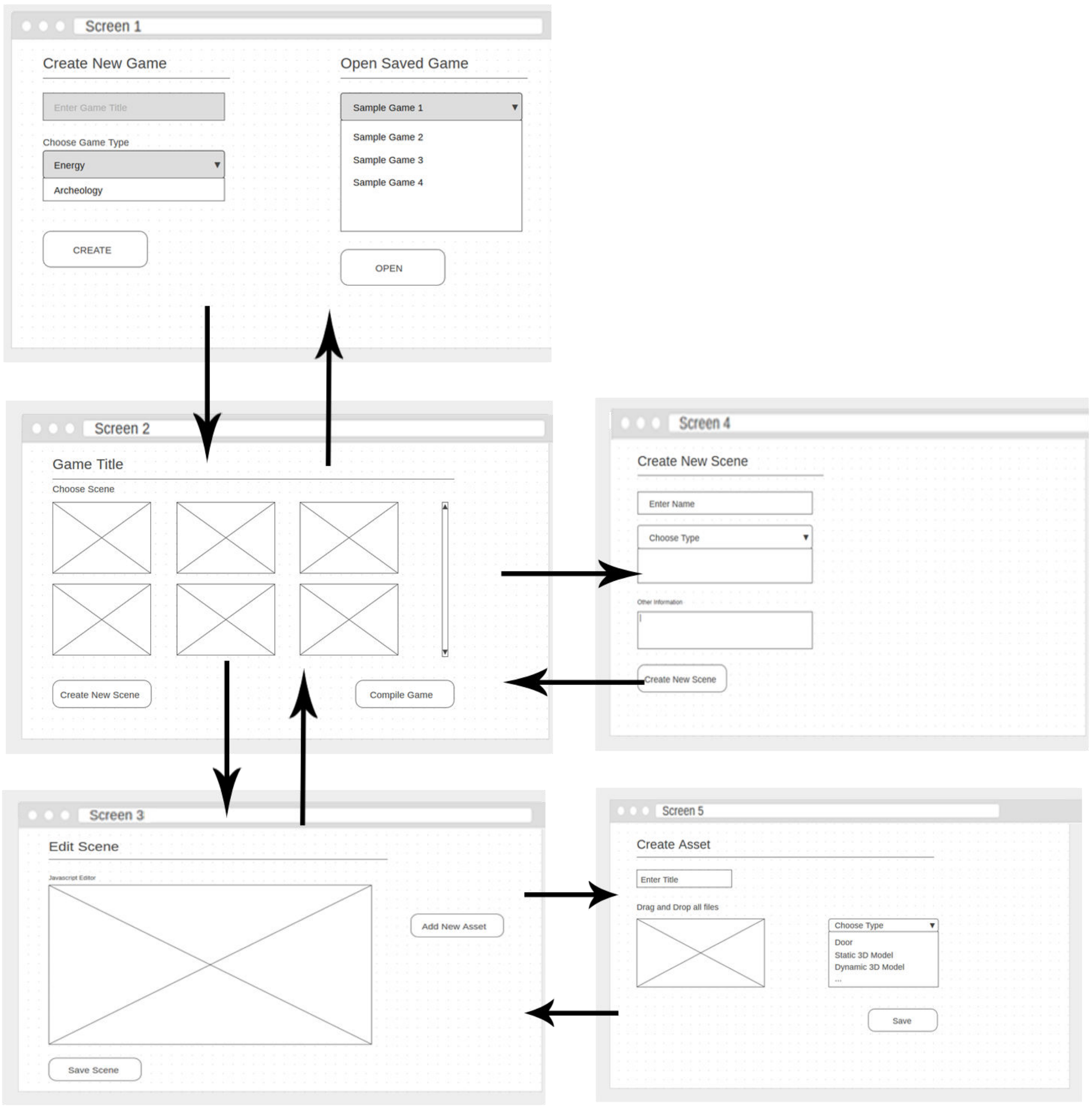
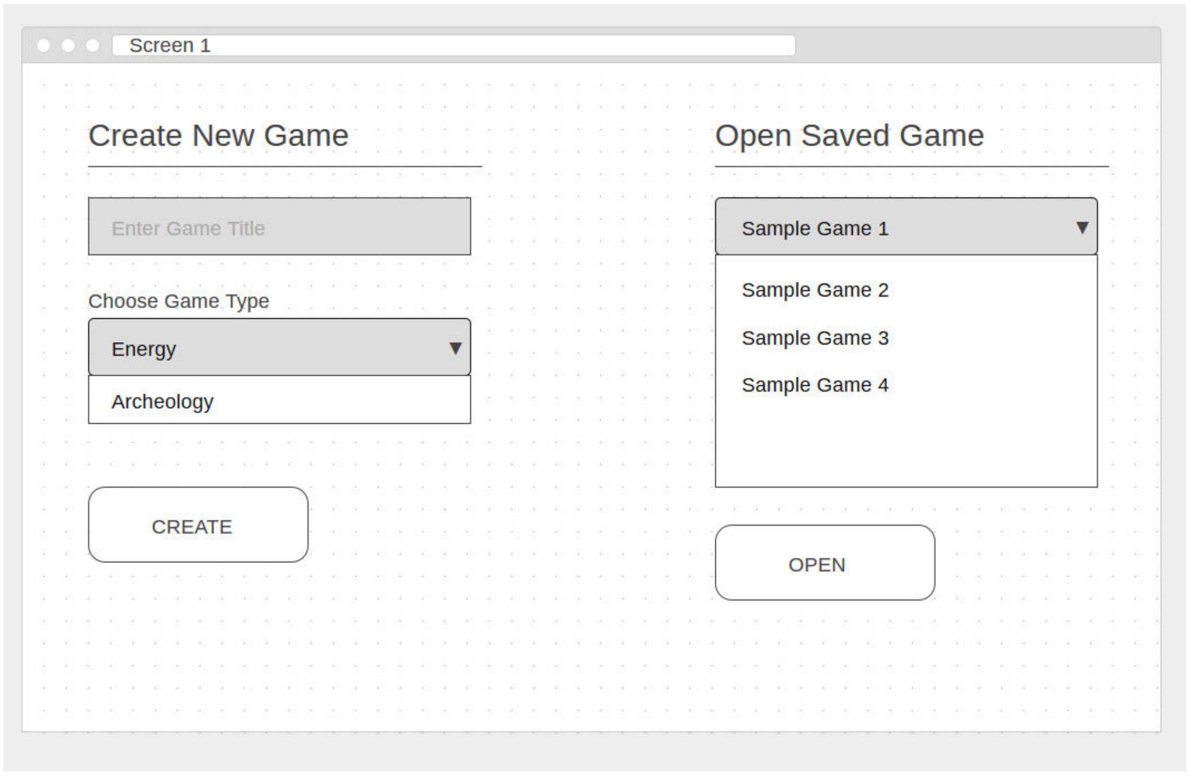
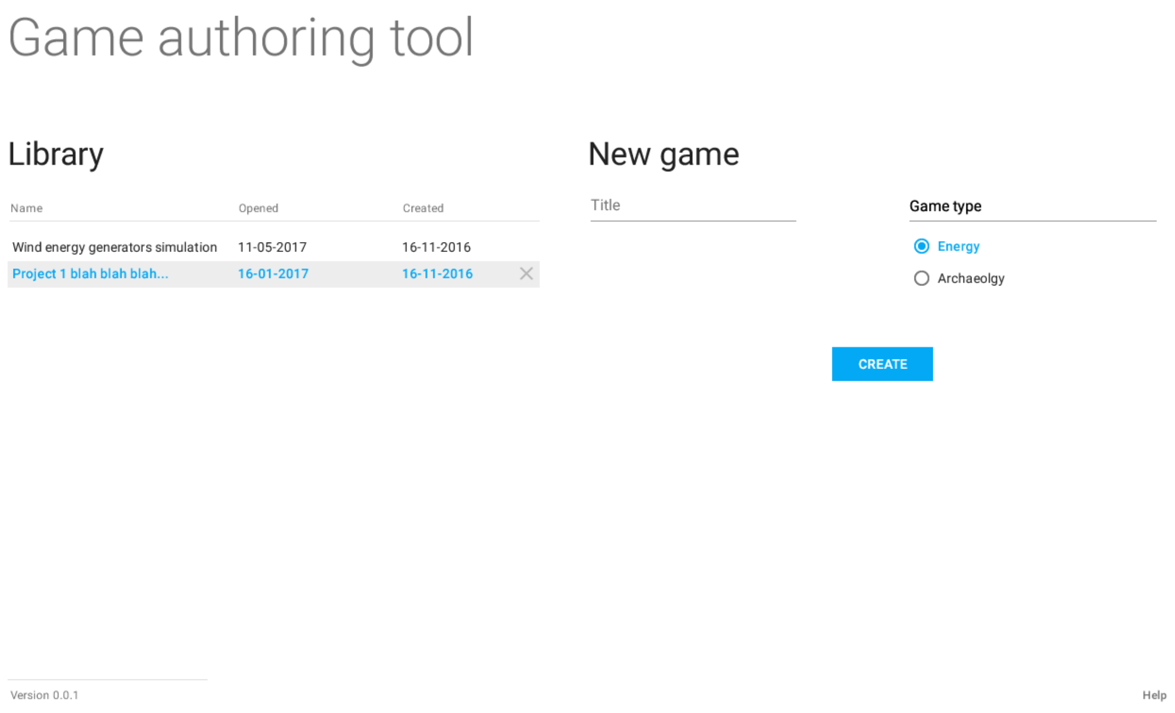


Figure 3.6: Mockups and overall organization of the front-end GUI.

3.4.1 Game manager

A mockup and a first implementation of the Game Project Manager are shown in Figure 3.7. In this screen, the educator can create a new game project or edit an existing one. A game project can be created by entering the title of the game project and the type of the game project.



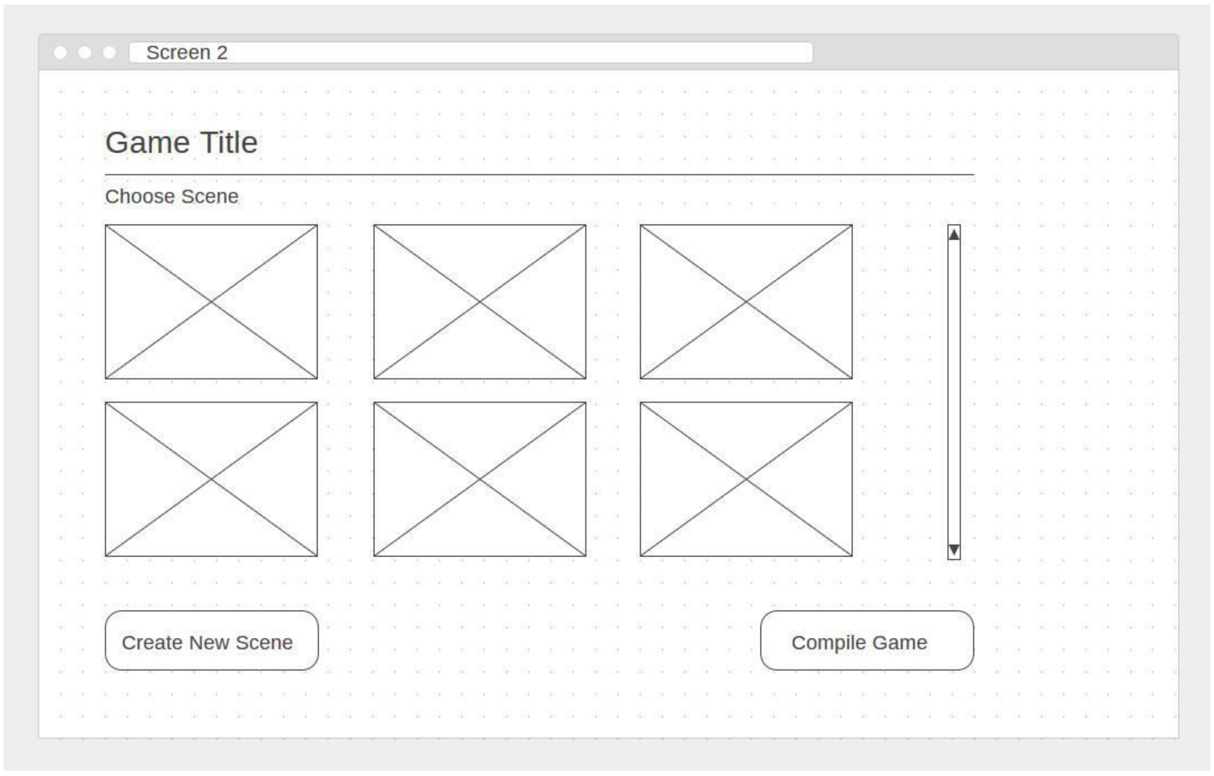


Name	Opened	Created
Wind energy generators simulation	11-05-2017	16-11-2016
Project 1 blah blah blah...	16-01-2017	16-11-2016

Figure 3.7: Mockup (top) and first design implementation (bottom) for creating a new game and or editing an existing one.

3.4.2 Scenes Management

After the creation of the game, the educator is led to Scene Management screen (Screen 2) as shown in Figure 3.8 (mockup and first implementation). In this interface, each scene of the game is represented with a thumbnail, and if clicked the user can modify the respective scene. A button, named "New Scene" allows to create a new scene, which will be further described in Section 3.3.4.



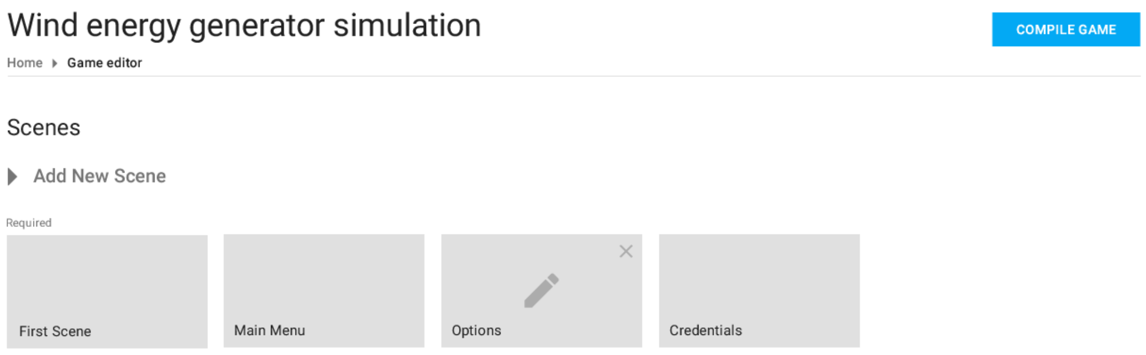


Figure 3.8: Mockup (top) and first design implementation (bottom) for the scenes management of a game.

3.4.3 Scene Editor

A mockup and a first implementation of the Scene Editor is shown in Figure 3.9. The educator can upload a new Asset3D using the “Add new Asset” button, which is an option available only for senior educators, and use the scene javascript 3D editor to manage the Assets3D in the scene as it will be described in Section 3.5.

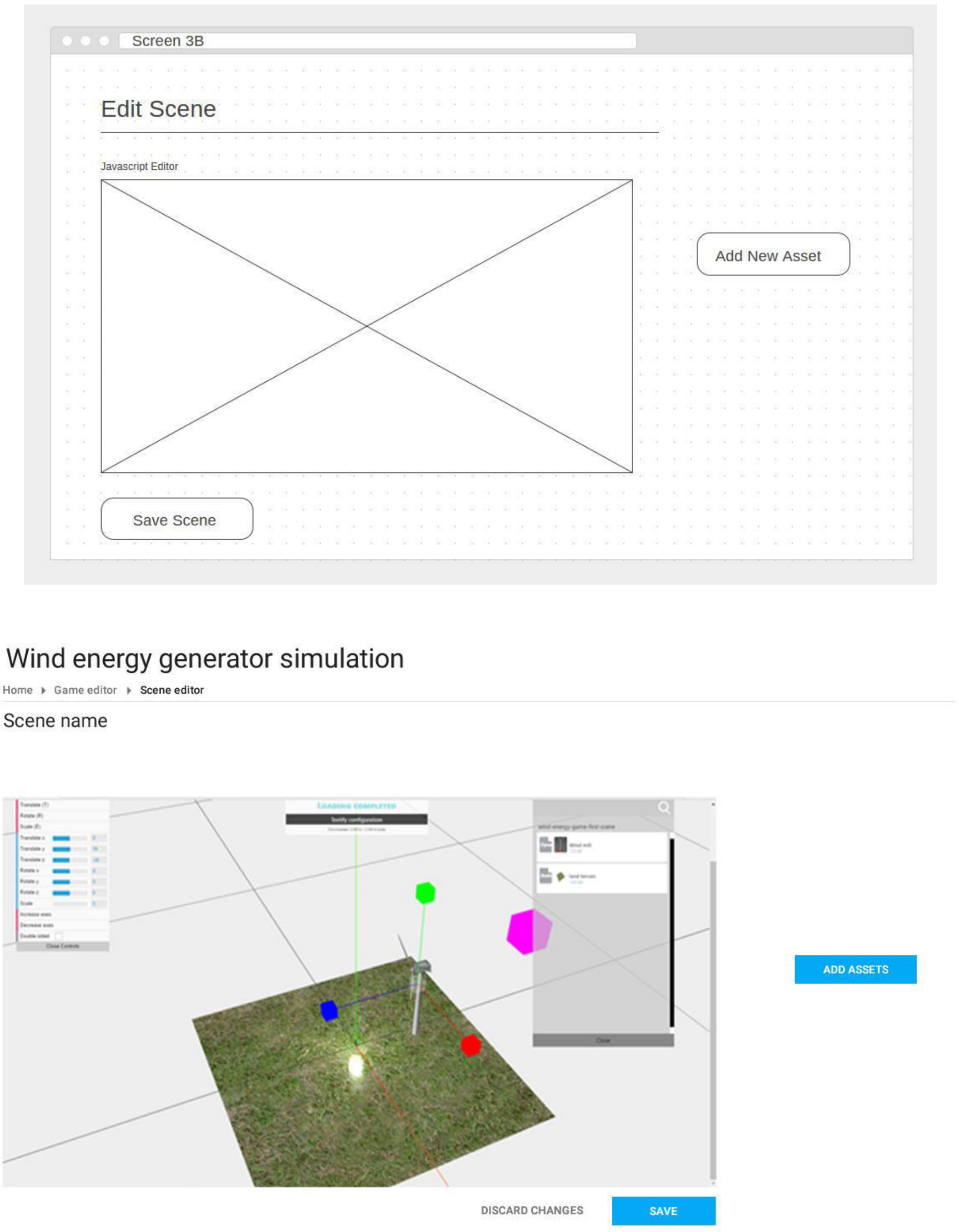
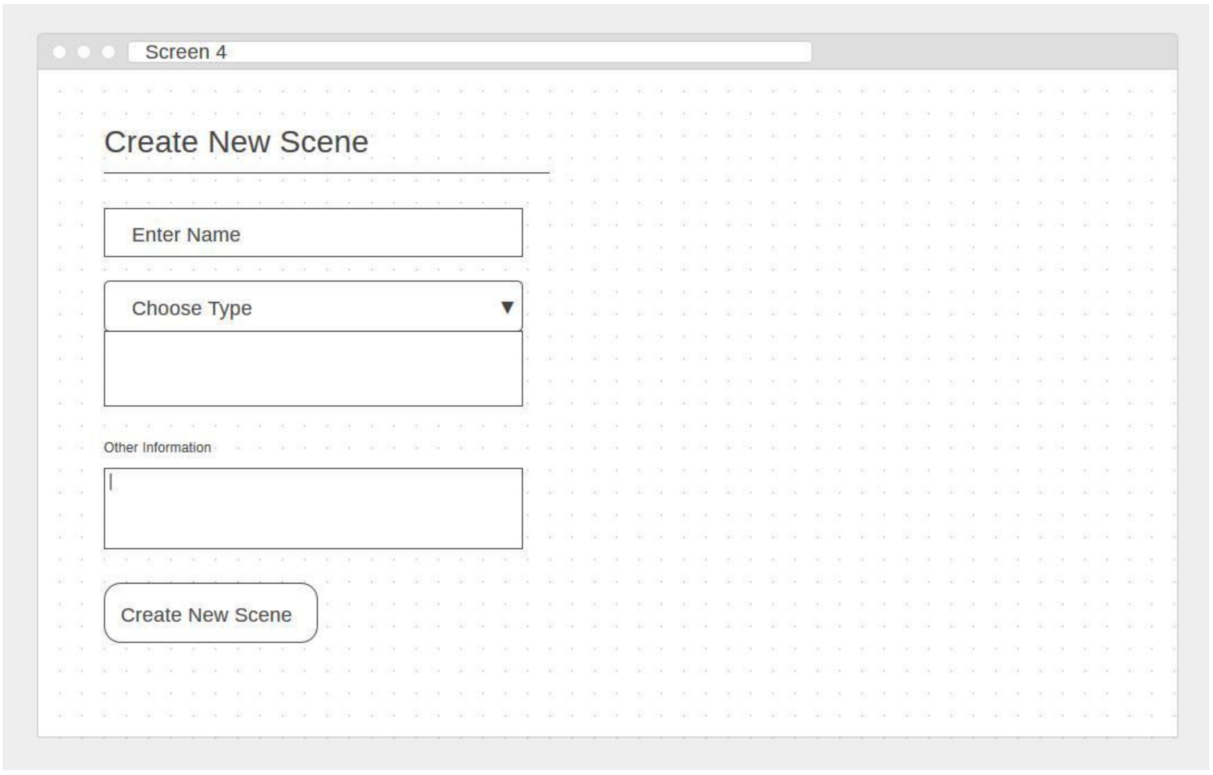


Figure 3.9: Mockup (top) and first design implementation (bottom) for the scene editor.

3.4.4 Scene Creator

The Scene Creator allows for a new scene to be created as shown in the mockup and first implementation of Figure 3.10. The scene can be created by entering a name, type, and a description of the scene.



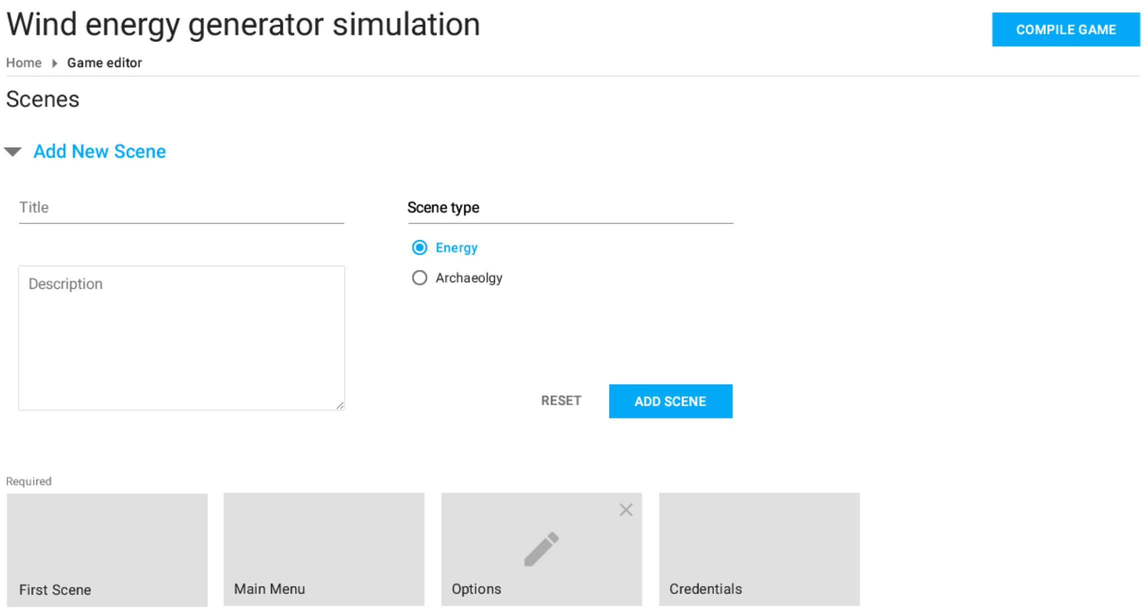


Figure 3.10: Mockup (top) and first design implementation (bottom) for the scene creator that allows to create a new scene.

3.4.5 Assets3D Manager

The Assets3D Manager screen is for creating new Assets3D and it is triggered when clicking “Add new Asset” in Scene editor. A mockup is shown in Figure 3.11. The educator provides a title, the Asset3D type, 3D files (obj, mtl, textures), and fills other fields depending on the Asset3D Type.

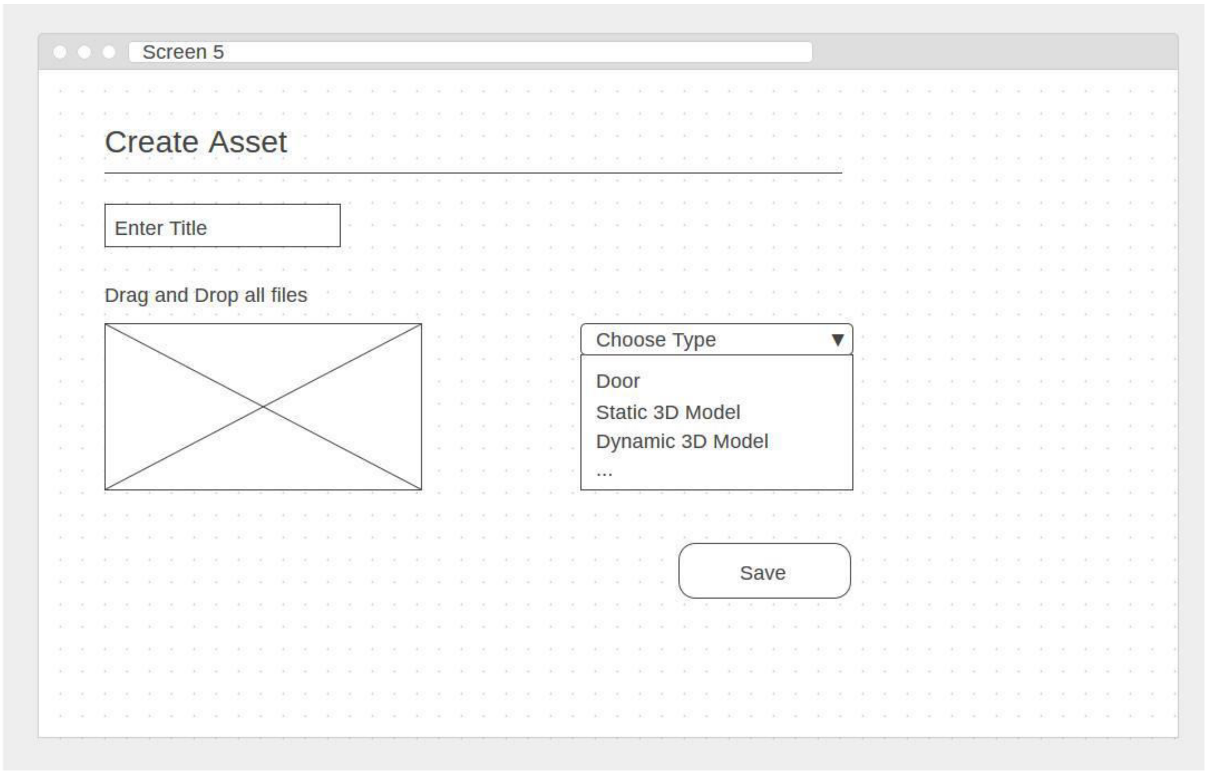


Figure 3.11: Mockup for creating an Asset3D.

3.5 Front-end Scene 3D editor

We have developed a web-based 3D editor to enable modifying a scene through a web-browser. In order to achieve this fast, we used the three.js[7] library that allows to develop 3D graphic elements using HTML5 and WebGL through high level commands. Three.js allows saving a scene in Json format where we have standardized our own format that serves the need of converting the scene setup to Unity3D scene format. Further details about the json format will be described in Section 3.6. A screenshot of the 3D editor can be seen in Figure 3.12. The Scene 3D editor consists of three parts, namely the 3D view of the Scene, the left toolbar of parameters, and the right toolbar of available Assets3D.

[7] http://threejs.org

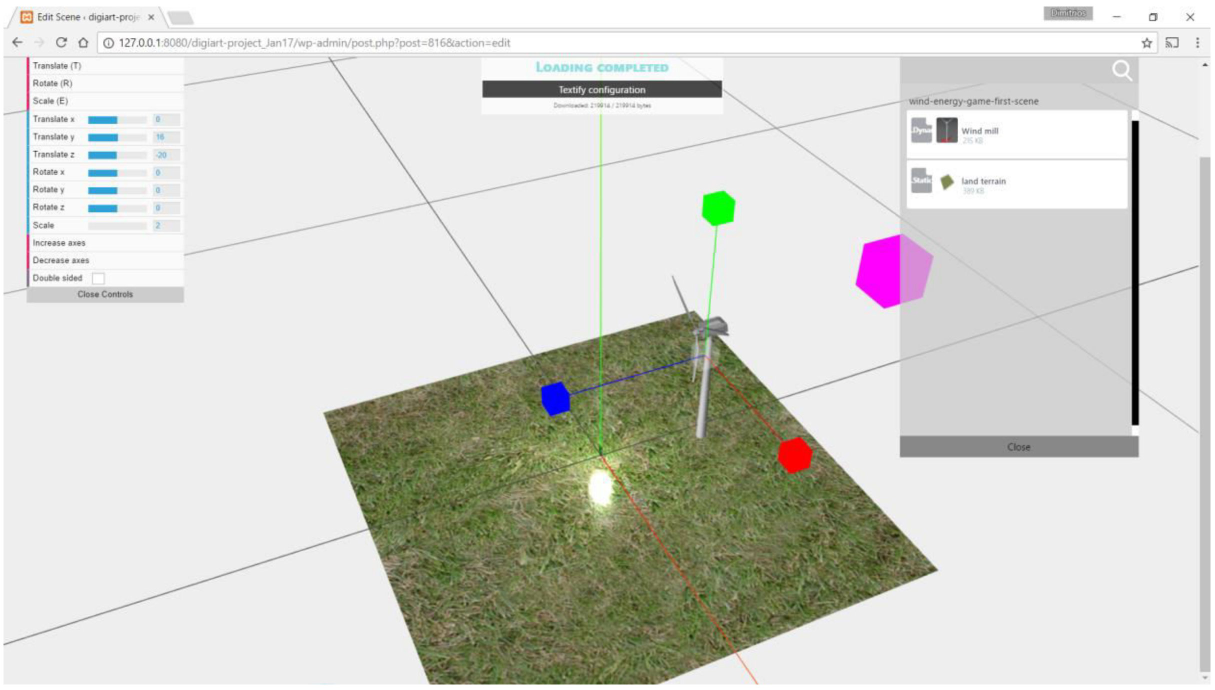


Figure 3.12: A scene editor for the web using Three.js.

The main functionality of the 3D editor is to allow an educator to drag-n-drop Assets3D from the toolbar on the right in the 3D view of the Scene. This action adds an instance of the Asset3D to the scene. Multiple instances of the same Asset3D can exist in the scene, i.e. multiple wind turbines. The educator can now edit the rotation, the position, and the scale of an instance either through graphic elements (gizmos) or by entering numerical values for a more accurate result in the left toolbar. Other functionalities supported are typical 3D editing functionalities such as a) view the scene either in 3rd person view or 1st person view; b) orbit, pan, or zoom to an object for a better angle view; and c) select object with raycasting (click on 3D items).

The 3D editor, apart from the scene 3D editing functionalities, offers the functionality to convert a WebGL scene into a json file. A WebGL scene is objects in the browser's memory that are structured in a tree like format with parameters such as object name, translation, rotation and scale. We have developed a json converter function that stores these parameters in a json file with a protocol. We have defined our own as for the time being there is no standard format for WebGL 3D scenes. If the educator enters the WebGL for a second time, then the json file is read and the WebGL scene is recreated as it was saved the last time. We decided to use json format instead of Unity3D YAML scene format because it is more compatible with WebGL technologies such as Three.js. Only when the game is compiled, this json scene is converted into Unity3D YAML scene as it will be described in the following.

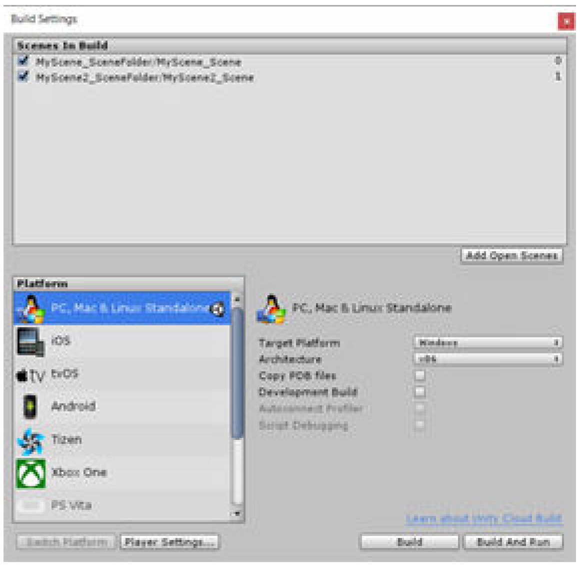
3.6 Conversion mechanism from WordPress to Unity3D game project

In order to compile a game, it should first be transformed from our WordPress format into Unity3D project format. This conversion is a tedious procedure. However, we have managed to check its feasibility. In order to explain the procedure, the structure of a Unity3D game project should be analyzed.

3.6.1 Unity game project

In this section, we will described how a Unity3D project is structured and what settings should be made in order for this project to be transformed into a WordPress template. The language used in Unity3D is YAML (Yet Another Markup Language) resembling the XML a lot. A Unity3D game project consists of two folders, namely **ProjectSettings Folder** and **Assets Folder**.

ProjectSettings Folder contains 16 files in YAML format that store the game settings. In our case, Edits should be done in two files, namely EditorBuildSettings.asset and EditorSettings.asset**.** EditorBuildSettings.asset contains which scenes will be compiled in the build of the game. These can be changed in the Unity3D Editor by going to menu -> File -> build Settings, as shown in Figure 3.13. This file is the list of the scenes belonging to the compiled game. The "WordPress Scene Manager" that we have developed modify this file accordingly in order to incorporate all necessary scenes into the compiled game.



Scenes In Build
- [✓] MyScene_SceneFolder/MyScene_Scene 0
- [✓] MyScene2_SceneFolder/MyScene2_Scene 1

PC, Mac & Linux Standalone

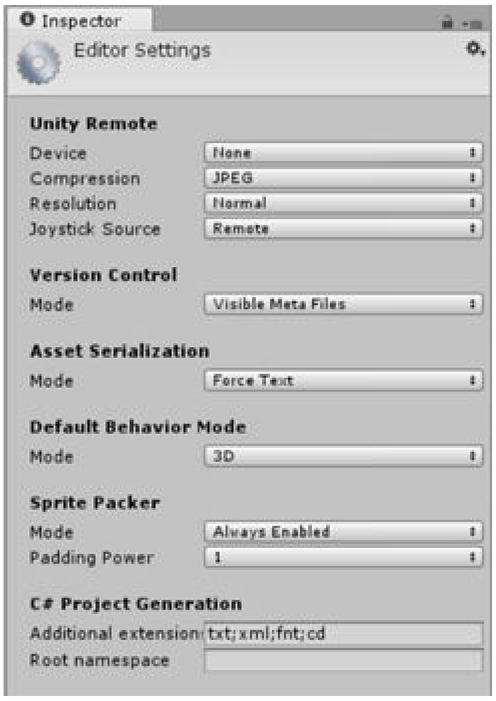
Target Platform	Windows
Architecture	x86
Copy PDB files	☐
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!1045 &1
EditorBuildSettings:
  m_ObjectHideFlags: 0
  serializedVersion: 2
  m_Scenes:
  - enabled: 1
    path:
Assets/MyScene_SceneFolder/MyScene_Scene.unity
  - enabled: 1
            path:
Assets/MyScene2_SceneFolder/MyScene2_Scene.un
ity
```

Figure 3.13: Modifying Unity3D Editor build settings graphically (left) and textually (right).

EditorSettings.asset file contains settings such as allowing the game meta files to be visible and all the files to be stored in text format instead of binary format. This is essential for extracting YAML patterns of code, which will be used in our authoring tool. These patterns are pieces of code where the value of the parameter is also parametrized by special characters that our WordPress plugin is able to understand and replace with certain values.

In Unity3D Editor, these settings can be changed in menu -> edit -> preferences -> editor and in the panel that will popup as the example shown in Figure 3.15 (left) by selecting “Version Control” mode as “Visible Meta Files” and “Asset Serialization” as “Force Text”. These settings produce the EditorSettings.asset file inside the ProjectSettings folder, which is shown on the right part of Figure 3.14.



Inspector
Editor Settings

Unity Remote
Device: None
Compression: JPEG
Resolution: Normal
Joystick Source: Remote

Version Control
Mode: Visible Meta Files

Asset Serialization
Mode: Force Text

Default Behavior Mode
Mode: 3D

Sprite Packer
Mode: Always Enabled
Padding Power: 1

C# Project Generation
Additional extension: txt;xml;fnt;cd
Root namespace:

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!159 &1
EditorSettings:
  m_ObjectHideFlags: 0
  serializedVersion: 3
  m_ExternalVersionControlSupport: Visible Meta
Files
  m_SerializationMode: 2
  m_DefaultBehaviorMode: 0
  m_SpritePackerMode: 2
  m_SpritePackerPaddingPower: 1
  m_ProjectGenerationIncludedExtensions:
txt;xml;fnt;cd
  m_ProjectGenerationRootNamespace:
  m_UserGeneratedProjectSuffix:
```

Figure 3.14: Modifying the EditorSettings graphically (left) and textually (right).

The **Assets Folder** is the most important folder and it contains all the necessary assets for constructing the game, namely Scenes, Scripts, Assets 3D, and Standard Assets (defaults by Unity). Its structure is as follows. Each asset, folder or file, is escorted by a meta file that has reference or settings information about the asset. In this manner the meta file defines the identity of the asset. For Envisage, we propose to use a tree form as shown below where all scenes are saved in one folder, all multimedia files in uploads folder, Standard unity assets in a separate folder, and all custom scripts in a homonymous folder.

- Scenes
    - Scene1.unity
    - Scene1.unity.meta
    - Scene2.unity
    - Scene2.unity.meta

```
        o ....
  ● Uploads
        ● Asset3D 1 (folder)
              o Asset3D 1.obj
              o Asset3D 1.obj.meta
              o Asset3D 1.mat
              o Asset3D 1.mat.meta
              o texture.jpg
              o texture.jpg.meta
        ● Asset3D 2 (Folder)
  ......
  .......
  ● Standard Assets (folder)
  ● Standard Assets.meta
  ● GameScripts (folder)
        o script1.cs
        o script2.js
```

As can be seen above, each file, or folder, is escorted by meta file. If a file, or folder, is moved to another directory then its meta file should be moved with it. In our case, the WordPress plugin is responsible for generating accurate meta files. The structure of a meta for a folder asset is as follows.

```
fileFormatVersion: 2
guid: 0425edc148da5504890f435baf6bbb6b
folderAsset: yes
timeCreated: 1481881843
licenseType: Free
DefaultImporter:
  userData:
  assetBundleName:
  assetBundleVariant:
```

where fileFormatVersion is 1 for Mac vs 2 for Windows. Guid (Graphic User Interface Identifier) is a string of 32 characters that identifies uniquely the folder asset when Unity constructs the "Project" panel, and timeCreated is the unix timestamp in secs from 1/1/1970 UTC.

Standard Assets Folder is a folder created automatically by Unity3D and stores common assets that are often used in games. This folder should be copied as it is in each game compilation.

Scene folder contains all the scene files and their meta in YAML format.

Uploads contains necessary 3D assets for a scene. The 3D assets are stored in folders where three items are necessary, namely the 3D mesh stored in an obj file, the material of the asset stored in a mat file and optionally the texture of the object stored in an jpg file.

jpg: it is the image representing the texture of a 3D object. It should always be rectangle with sizes at power of 2, i.e. 64,128,256,512 …., up to 8192. Although Unity3D supports also png format, we have been restricted to jpg for better transfer times via the internet. The first lines of the meta file of the texture is as follows (The file is too big showing many texture details that are not necessary to present here. The most important line is the guid string, which is the id of the texture file).

```
fileFormatVersion: 2
guid: 5c327651f9d4f8946a645bcdbfadc6c1
timeCreated: 1482232683
licenseType: Free
TextureImporter:
  fileIDToRecycleName: {}
  serializedVersion: 4
  mipmaps:
        mipMapMode: 0
        enableMipMap: 1
        sRGBTexture: 1
….
```

obj: It is a file in Wavefront file format, which is a text format for storing object meshes. Although Unity3D supports various formats, we decided to use obj due to its simplicity and universality. The most important lines of Obj meta file are as follows.

```
fileFormatVersion: 2
guid: 42aa2f61ccc79ff45914686b2622efd4
timeCreated: 1482234737
  materials:
        importMaterials: 1
        materialName: 2
        materialSearch: 2
```

mat: The material, i.e. color and texture, of the Obj mesh is saved in a mtl file. Unity3D transforms automatically mtl into mat. An example is shown in the following lines where m_Texture contains the id of the texture file and m_Colors contains the color of the texture.

```
--- !u!21 &2100000
Material:
  m_Name: floor-defaultMat
  m_Shader: {fileID: 46, guid: 0000000000000000f000000000000000, type: 0}
  m_ShaderKeywords: _EMISSION
  m_LightmapFlags: 1
  m_CustomRenderQueue: -1
  stringTagMap: {}
  m_SavedProperties:
        serializedVersion: 2
        m_TexEnvs:
        - first:
        name: _MainTex
        second:
        m_Texture: {fileID: 2800000, guid: 5c327651f9d4f8946a645bcdbfadc6c1, type: 3}
        m_Scale: {x: 1, y: 1}
        m_Offset: {x: 0, y: 0}
        m_Colors:
        - first:
        name: _Color
        second: {r: 0.38843808, g: 1, b: 0.33823532, a: 1}
```

js and **cs** are javascript and c-sharp scripts that define game rules, behaviors and others inside the game.

Scene.unity is the most important file type since it contains information about which game objects to construct and which assets to use for them. It also contains rendering information, light, fog, occlusion settings etc. In details, first there are the settings of the scene that do not depend on the game objects inside the scene, these are the Occlusion, Render, LightMap, and NavMesh settings that are common entities in 3D environments. Each item begins with "---- !u!yaml_class_id &fid" where yaml_class_id takes certain values that specify the class of the object, and fid is the identifier of the object, which is unique in the scene. This full class list can be found in Unity site8. An example of .unity file for the aforementioned components is as follows:

```
%YAML 1.1
%TAG                              !u!
tag:unity3d.com,2011:
--- !u!29 &1
OcclusionCullingSettings: ....
```

[8] https://docs.unity3d.com/Manual/ClassIDReference.html

```
--- !u!104 &2
RenderSettings: …..
--- !u!157 &3
LightmapSettings: …..
--- !u!196 &4
NavMeshSettings: ……
```

Next, is the light game object. One scene can have multiple lights. For open space scenes, where the only source of light is the sun (DirectionalLight), the light consists of a GameObject (wrapper container), the Light (type of light), and Transform (position and orientation) as follows.

```
--- !u!1 &5
GameObject:
  serializedVersion: 5
  m_Component:
  - component: {fileID: 7}
  - component: {fileID: 6}
  m_Name: Directional Light
  m_IsActive: 1
--- !u!108 &6
Light:
  m_GameObject: {fileID: 5}
  m_Type: 1
  m_Color: {r: 1, g: 0.95686275, b: 0.8392157, a: 1}
  m_Intensity: 1
  m_Range: 10
  m_SpotAngle: 30
  m_Shadows:
        m_Type: 2
        ….
        m_Bias: 0.05
        m_NormalBias: 0.4
        m_NearPlane: 0.2
        m_BounceIntensity: 1
--- !u!4 &7
Transform:
  m_GameObject: {fileID: 5}
  m_LocalRotation: {x: 0.40821788, y: -0.23456968, z: 0.10938163, w: 0.8754261}
  m_LocalPosition: {x: 0, y: 3, z: 0}
  m_LocalScale: {x: 1, y: 1, z: 1}
  m_LocalEulerAnglesHint: {x: 50, y: -30, z: 0}
```

The aforementioned game objects will be fixed in the scene. The educator, however, should be able to insert various new game objects when constructing each scene. These game objects should be customly pre-defined according to the following.

3.6.2 Predefined game object types

Each scene should contain a variety of game objects. These game objects should obtain a behaviour of a predefined class, named as taxonomy value. Thus, if the educator selects a taxonomy value for a new game object, automatically the game object inherits the properties of the taxonomy value class.

An example for the Asset3D Type “Land terrain”, is described in the following. If assuming that a json scene contains an instance of Asset3D of the type “Land Terrain”. Then myScene.json file should be converted into myScene.unity. The myScene.json file contains several game objects but if focusing only to our “Land Terrain” game object the conversion is as follows.

Our json game object has several fields, such as:

- id_instance=115231 (uniquely identifies the game object instance in the scene);
- id_asset3Dtype=15 (identifies the Asset3D type of the instance) ;
- id_mesh=155. (id of the mesh resource from WordPress to use for the 3D object);
- translation_XYZ (position vector);
- rotation_XYZ (rotation vector);
- scale (float value uniform for all dimensions).

The first thing is to take the the YAML code snippet from the Asset3D Type taxonomy meta for the Asset3D Type = 15 which is:

```
--- !u!1 &___[landterrain_fid]___ stripped
GameObject:
  m_PrefabParentObject: {fileID: 100000, guid: ___[landterrain_guid]___, type: 3}
  m_PrefabInternal: {fileID: ___[landterrain_prefab_fid]___}
--- !u!64 &___[landterrain_meshcol_fid]___
MeshCollider:
  m_ObjectHideFlags: 0
  m_PrefabParentObject: {fileID: 0}
  m_PrefabInternal: {fileID: 0}
  m_GameObject: {fileID: ___[landterrain_fid]___}
  m_Material: {fileID: 0}
  m_IsTrigger: 0
  m_Enabled: 1
  serializedVersion: 2
  m_Convex: 0
  m_InflateMesh: 0
  m_SkinWidth: 0.01
  m_Mesh: {fileID: 4300000, guid: ___[landterrain_guid]___, type: 3}
--- !u!1001 &___[landterrain_prefab_fid]___
Prefab:
  m_ObjectHideFlags: 0
  serializedVersion: 2
  m_Modification:
    m_TransformParent: {fileID: 0}
    m_Modifications:
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalPosition.x
      value: ___[landterrain_pos_x]___
      objectReference: {fileID: 0}
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalPosition.y
      value: ___[landterrain_pos_y]___
      objectReference: {fileID: 0}
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalPosition.z
      value: ___[landterrain_pos_z]___
      objectReference: {fileID: 0}
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalRotation.x
      value: ___[landterrain_rot_x]___
      objectReference: {fileID: 0}
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalRotation.y
      value: ___[landterrain_rot_y]___
      objectReference: {fileID: 0}
      ...
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalScale.x
      value: ___[landterrain_scale_x]___
      objectReference: {fileID: 0}
    - target: {fileID: 400000, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_LocalScale.y
      value: ___[landterrain_scale_y]___
      objectReference: {fileID: 0}
      ...
      objectReference: {fileID: 0}
    - target: {fileID: 100002, guid: ___[landterrain_guid]___, type: 3}
      propertyPath: m_Name
      value: ___[landterrain_name]___
      objectReference: {fileID: 0}
```

and replace all the WordPress keywords that begin from “___[“ and end with “]___” with information from json file. For example, the ___[landterrain_pos_y]___ takes the value of translation_XYZ(1).

Concluding, the Unity3D game project specifications are as follows:

1. Game projects should be saved in text format but not binary;
2. Each game object should be referenced with its name. Tags are avoided to lower complexity;
3. Each asset (obj, mtl, textures) should be saved in a folder (asset folder);
4. Each scene should be saved in a separate folders and all the assets of the scene should be stored within this folder in asset folder. Therefore a tree form structure is secured.
5. All game scripts should be saved in a root folder named as "Scripts";
6. the paths of the files are not saved anywhere, instead each file or folder is escorted with a .meta file, which stores various ids for the file or folder, that are used when constructing the scene by each scene file, e.g. by myScene.unity.

3.6.3 Compiling a Unity3D game project to a game

The game project should be compiled into a game binary or web format. This is allowed through the following command:

```
"C:\Program Files\Unity\Editor\Unity.exe" -quit -batchmode -logFile stdout.log -buildWindowsPlayer "builds\mygame.exe"
```

where

- "Unity.exe" is the compiler.

- "stdout.log" is the file, where all the messages during compilation are written. This file can be read sporadically with ajax to make a progress bar in the web client. It can also inform the curator that there is no problem in compilation. If any problem arises, it can be also shown.

- "buildWindowsPlayer" is a parameter denoting that the target platform is windows. It can be Mac, WebGL, Playstation, Xbox, Android etc.

- "Builds\mygame.exe" is the game binary. Together another folder is made with data named: "mygame_data", which is also necessary for the game to run. The folder and the exe have to be zipped into one file and a link should be sent to the user to download the zip file.

4. Analytics metrics injection for the game feedback mechanism

In this section, the implementation architecture of the shallow analytics is described briefly in order to ensure concordance with all the modules of the architecture. More details can be found in D2.1 where each shallow analytics module is thoroughly described.

Shallow Analytics architecture design is shown in Figure 4.1. It consists of two components, namely:

a) **Shallow analytics server:** The server will be hosted in Amazon Web Services that provide technologies for high traffic, storage, security and maintenance at low cost. The components of Shallow Analytics server are as follows:
    i) **API IN:** collects game analytics and deep learning analytics data through a server. Actually, it is code written in PHP or JAVA that allows metrics data to enter the server. It has the form of a url link.
    ii) **Data Aggregation:** refers to temporary storage of data in NoSQL in order to be buffered for batch process; enriched data with geo-location inferred from IP; enriched data with demographics inferred from location; enriched data with login information such as calculated sessions.
    iii) **Data Augmentation:** the augmentation of data with incoming data. It is first-level features estimation from the aggregated data in order to be used from Deep Analytics Server for visualizations or for second-level features estimation.
    iv) **DB:** the storage of data into a database. It stores the raw, the aggregated, and the augmented data in a database based on DynamoDB technology.
    v) **API OUT:** the construction of an API server for sending the data to the Deep Analytics Server for extracting meaningful information for the educators or to the game authoring tool for the visualization of game analytics. Actually, it is code written in PHP or JAVA that allows metrics data to exit the server securely in order to be used from the Deep Analytics server or the game authoring tool. It has the form of a url link.

b) **Metrics:** code for tracking user activities in the virtual Labs:
    i) a Unity3D plugin named as GIO SDK will be used to place metrics tracking functions in appropriate positions such as starting the scene, ending the scene etc. This will be described further later in this section.
    ii) the code triggers calls carrying metrics data to the API IN through http. The code will be written either as C++ or Javascript since Unity3D supports both languages.

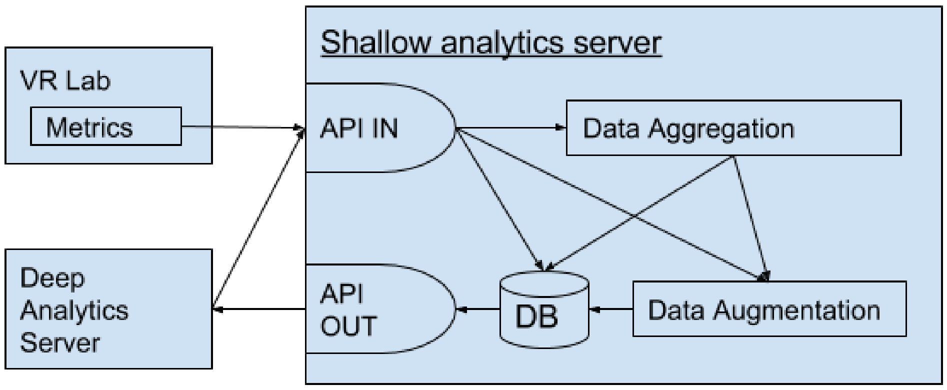


Figure 4.1: The architecture of the Shallow Analytics server.

As regards tracking code, the game should be able to track:

- How much time the learner spends in reading the instructions of the game with two tracking points such “InitiateStartScene”, “TerminateStartScene” that provide timestamps information
- Assess whether the student understands the value of the different parameters with Tracking points:
    - “OpenConfigurationPanel” and “CloseConfigurationPanel”
        - How many times when into the ConfigurationPanel
        - How much time it spent in each case
    - “StartSimulation”, “ReplaySimulation” and “EndSimulation”,
        - How many times when into the ConfigurationPanel
        - How much time it spent in each case
- Assess whether the student can optimize the generation-consumption trade-off with a pre-set configuration for the parameters found with tracking points:
    - “AddTurbine”, “TurnOffTurbine”, “RepairTurbine” -> Tracked on user-clicks
    - “UnderPower”, “OverPower”, CorrectPower” -> Tracked the evaluation output every 5 seconds.
    - “StartSimulation” and “EndSimulation”

5. Architecture for extracting deep analytics and visualizing shallow and deep analytics

In this section, we describe the architecture design of the components used to visualize shallow analytics and deep analytics to educators while using the virtual Lab game authoring tool. The architecture is described in Figure 5.1. Here we provide only an overview. Details on the deep analytics and visualizations can be found in D2.3.

Deep learning analytics values, e.g. the outcomes of particular models, are calculated using the scikit-learn machine learning library and a range of custom scripts leveraging this library. The machine learning deep analytics service is served using the Flask microserver framework for python. The Deep Analytics data are stored back on the Shallow Analytics server. The visualization code in the Master Server draws data produced from shallow and deep analytics operating. The visualizations, deemed useful for educators, are rendered using the D3.js visualization inside the Scene Editor of the WordPress plugin. Some calculations that produce values that are ephemeral and/or only locally relevant, such as filtering, splitting, scaling, other forms of normalization, and sorting of values are carried out on the client-side in the end user's browser, using JavaScript implemented as part of the D3.js visualizations.

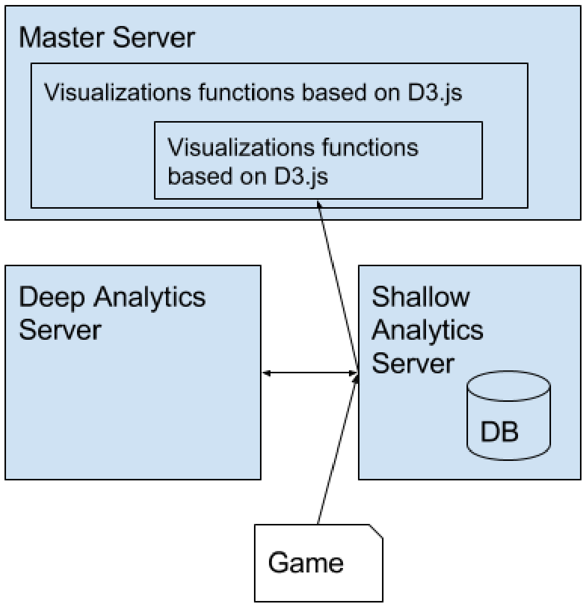

Master Server
Visualizations functions based on D3.js
Visualizations functions based on D3.js
Deep Analytics Server
Shallow Analytics Server
DB
Game

Figure 5.1: The Deep Analytics Server with respect to other servers.

Below, in Table 5.1, we provide an overview of the libraries involved for the Deep Analytics server and the visualizations in the Master server.

Table 5.1: Overview of the technologies for deep learning and visualization.

Library/software solution	Website/repository	License
D3.js	https://d3js.org/	BSD-3-Clause

Flask	http://flask.pocoo.org/	BSD-3-Clause
scikit-learn	http://scikit-learn.org/	BSD license (version unspecified)
python	https://www.python.org/	PSF License

6. Architecture summary

In the past, several attempts to make educational games were discontinued due to the lack of an appropriate architecture that would be self-sustained and commercially viable. We propose an architecture that reduces the interference of a programmer or a game designer in the pipeline as much as possible. We provide the educators with a tool to make their own games, improve them, and maintain them. This significantly lowers the cost, since the programming effort is very expensive for 3D games.

The virtual lab game authoring tool allows for a great number of extensions either in the existing Wind-Energy-Simulation lab or any other newly introduced lab because it is based on the notion of a game template. The existing template allows for several objects to be inserted into the system dynamically, allowing the games to be highly customized by educators. The template, thus, can be transformed from “Wind Energy Simulation” to a more general “Energy Simulation” template by inserting other energy assets such as hydro-electric generators, photovoltaic elements. New terrains can be inserted from google maps simulating real physical places. Furthermore, the overall architecture is not limited to certain one type of lab but it can be extended for other types of labs by developing a new template.

A high level of usability is achieved with the proposed game authoring tool for the educators. By hiding unnecessary complexities, it is a user-friendly tool for modifying Unity3D games remotely through a web-browser. Analytics are already injected into the game objects, allowing for seamless integration. Visualization of analytics near the scene editor allows for the educator to do the necessary modifications in order to further improve the game in a efficient manner.

The adopted architecture, based on WordPress and Unity3D commercial software, is a novel architecture that adds value upon already widespread tools to achieve a high probability of commercialization. This commercialization can be achieved through a) selling the access of the game authoring tool to educational organizations; b) selling the platform itself to companies that wish to start selling the service to educators; c) making new templates to support functionalities of other kinds of virtual labs; d) selling the authoring tool to WordPress sites; and e) sell the plugin in the Unity3D Asset Store.

7. References

D1.1, “Educational scenarios and stakeholder analysis,” Envisage project deliverable, Apr. 2017.

D1.2, “Data structure and functional requirements,” Envisage project deliverable, Feb. 2017.

D2.2, “User profiling and behavioral modeling based on shallow analytics,” Envisage project deliverable, May 2017.

D2.3, “Visualization strategies for course progress reports”, Envisage project deliverable, May 2017.

Prensky, Marc. "Fun, play and games: What makes games engaging." Digital game-based learning 5 (2001): 1-05.