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

H2020-ICT-24-2016

D4.2 - First Version of the "Virtual Labs" authoring tool

Dissemination level:	Public (PU)
Contractual date of delivery:	Month 7, 30th April, 2017
Actual date of delivery:	Month 10, 28st July , 2017
Workpackage:	WP4 - Virtual labs authoring tool
Task:	T4.2 – Implementation of the virtual labs authoring tool with generalizable templates
Type:	Demonstrator
Approval Status:	Final for submission
Version:	0.9
Number of pages:	62
Filename:	D4.2_FirstVersionVirtualLabsAuthoringTool.docx

Abstract: The "Virtual labs authoring tool" is a plugin for WordPress that allows educators to design experiments with an easy to manipulate graphic user interface. The educators are able with drag-n-drop functionalities to design the experiment in 3D space and allow the learner to play and learn. The authoring tool has an interface in a web browser (web page) that it is able to generate virtual labs in a certain game engine, i.e. Unity3D. These games can be compiled either for Web or for desktop use. They are available through a link or they can be downloaded to be installed in a another server or desktop. The aforementioned procedure is achieved through game project templates that are split into pieces of code to re-design a new game. These templates have the necessary metrics measurement mechanisms embedded that monitor a learners' behavior and communicate to the game and visual analytics components as defined in WP2 and WP3.

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	29/5/2017	Initial draft with Table of Contents	Dimitrios Ververidis, Stathis Nikolaidis
v0.2	5/7/2017	Categories and fields for each game Asset3D	Dimitrios Ververidis
v0.3	11/7/2017	New screenshots	D.V.
v.0.4	14/7/2017	Section 1 and Section 2 almost completed	D.V., A. Papazoglou, S. Nikolaidis
v.0.5	19/7/2017	Sections 1, 2, and 3 almost completed	D.V.
v.0.8	25/7/2017	Sections 1,2,3, and executive summary completed. GIO and UoM are inserting content in 4,5, and 6	D.V, Christoffer Holmgard, Fabian Hadiji, Marc Mueller
v.0.9	28/7/2017	All sections are completed and reviewed from internal reviewers. Reviewers changes were acknowledged.	DV, CH, FH, MM

Author list

Organization	Name	Contact Information
CERTH	Dimitrios Ververidis (DV)	ververid@iti.gr
CERTH	Stathis Nikolaidis (SN)	stathis.nikolaidis@iti.gr
CERTH	Anastasios Papazoglou - Chalikias (AP)	tpapazoglou@iti.gr
UoM	Christoffer Holmgard (CH)	holmgard@gmail.com
GIO	Marc Mueller (MM)	marc@goedle.io
GIO	Fabian Hadiji (FH)	fabian@goedle.io

Executive Summary

In this deliverable we describe the developments towards a user-friendly platform that can generate virtual labs through high quality game engines and web interfaces. In the previous deliverable (D4.1 Architecture design) we presented our idea which is briefly to employ WordPress in order to develop a new editor for Unity3D game engine. This editor hides all the programming details from educators and allows them to build educational game projects through their web browser. The game projects are compiled by Unity3D game engine and the game output is a high quality product for WebGL, desktop, consoles, or mobile devices. The whole procedure is based on game project templates that incorporate high level organization to allow object behavior inheritance. In D4.1, we have provided the feasibility study of the attempt and some mockups, whereas here we provide quality assurance and augmentation through the implementation of several features, visualizations and structural improvements.

The structure of the developed virtual labs authoring tool is 3 dimensional as it allows first the generation of multiple instances of game projects, e.g. Energy Lab A, Energy Lab B, etc.; second, the generation of multiple versions per game project, e.g. Energy Lab A v1, Energy Lab A v2, etc.; and third, the extension of game project types, e.g. Energy Lab, Chemistry Lab, etc. Although, we focus only on Energy Labs in this deliverable, we provide evidence that a second type of labs is plausible and it will be delivered in the next deliverable.

Significant help towards the development of the second dimension, i.e. the versions of each lab, is provided through the use of the developed shallow analytics that are injected seamlessly and a priori in the games during their generation. Their feedback is augmented with several metrics plus the output of machine learning algorithms and visualized in the virtual labs authoring tool to allow educators to obtain an automatic tool for advice.

All code repositories are public available under github organization ENVISAGE-H2020:

https://github.com/Envisage-H2020

The main repository is the ‘Virtual-Labs-Authoring-tool’ which is a plugin for WordPress.

Abbreviations and Acronyms

CMS	Content Management System
API	Application Program Interface
GUI	Graphic User Interface
JSON	JavaScript Object Notation
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. Introduction to the architecture and the interfaces

The basics of the architecture that was defined in D4.1 will be outlined in order to provide an insight about the implementation of the described prototype. The ENVISAGE approach is to connect two dominating technologies in two separate fields, namely WordPress for web based content management systems, and Unity3D for game authoring. Here, WordPress is used to generate Unity3D game projects (Virtual labs game project) with an user-friendly interface for totally novice users, and then compile it with the Unity3D game engine in WebGL format.

The overall structure of the architecture is shown in Figure 1.1. The educator uses a web browser and enters the web page of the virtual labs authoring tool which is hosted in a WordPress site. In technical terms, the virtual labs authoring tool is a WordPress plugin that can be used by everybody to be installed in their website and transform the website into a game creation tool. This most prominent feature of this plugin is to allow creating game projects, scenes, and assets by defining their behavioral category, uploading the 3D models (for Assets only), and defining some field values. This categorization of the entities allows the entity to inherit a behavior from its category (taxonomy in WordPress language) and thus hiding unnecessary details from the game author. For already deployed games, visualization of game analytics inside the plugin allow the game authors to view the players' behavior and so as to re-design the game. Upon the game is re-designed, it can be compiled with the plugin which utilizes the Unity3D game engine for exporting into WebGL language. The game can be played from the web server or it can be downloaded to be placed in another server.

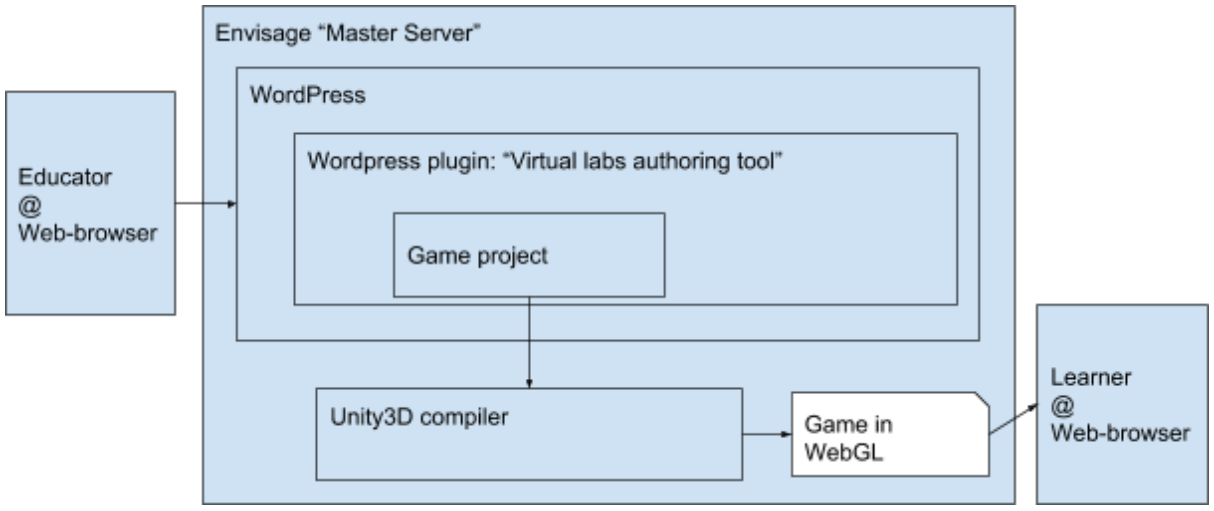

Figure 1.1: Virtual Labs game authoring tool is a WordPress plugin at a server containing Unity3D.

The overall architecture of the plugin is shown in greater detail in Figure 1.2. The backbone of the system are 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, and c) the “Deep Analytics Server” that processes the game analytics of the “Shallow Analytics Server” and ports the results to the “Master Server” as feedback for educators.

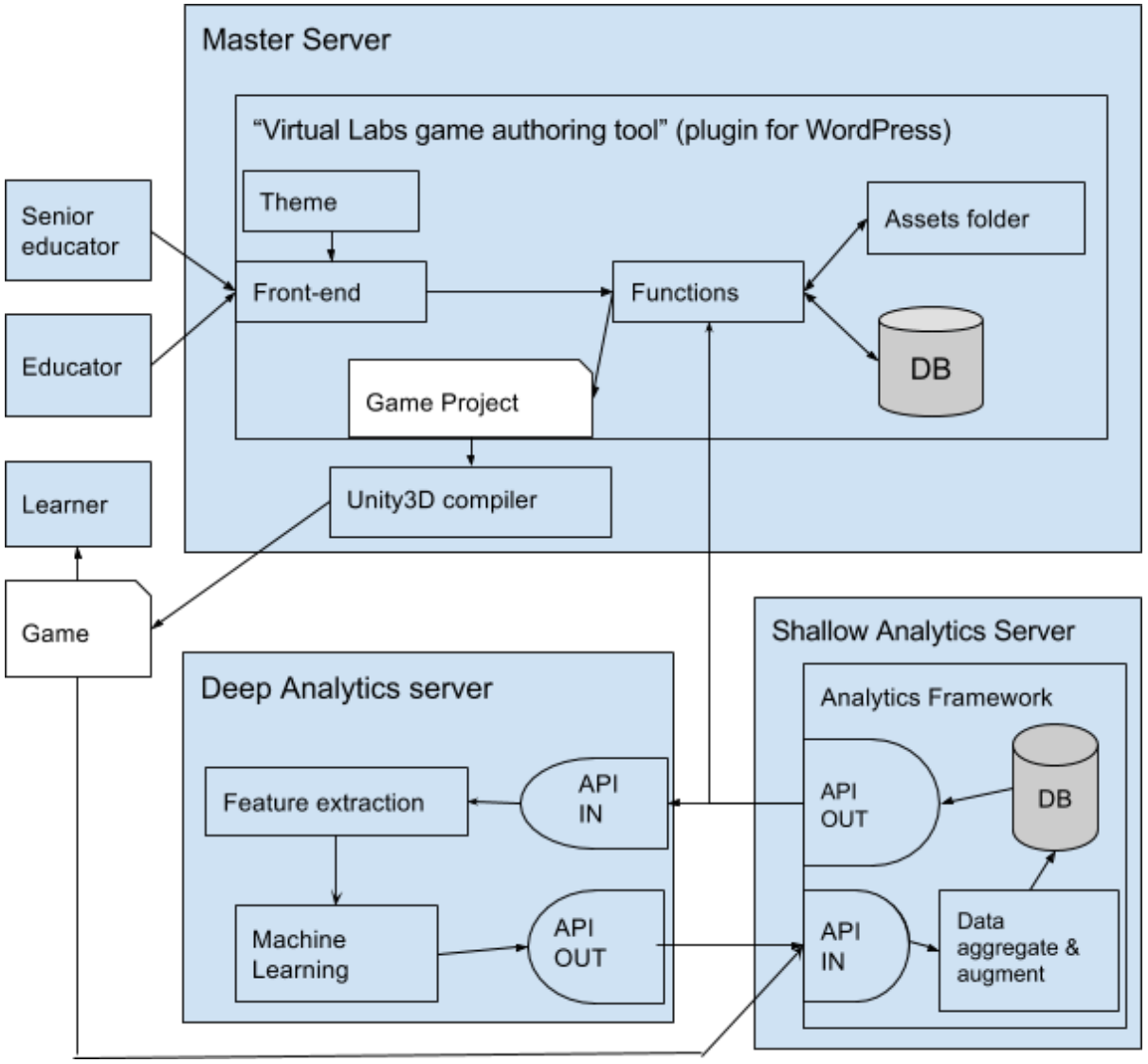

Figure 1.2: Second diagram of the architecture that displays logical components in greater detail.

The Wordpress plugin has the following main functionalities:

1) Provide a **front-end** web interface that allows the educators to login, create or edit a game project. This front-end is shaped through developed **page templates** that provide the required functionalities. 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. More details about the front-end and the back-end can be found in Section 2.
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).
4) Compile the Unity3D game compiler, and provide the game as a link for sharing.

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 the 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 5).
3) Store analytics data in a database schema.
4) Provide an API for sending analytics data to the Deep Analytics Server or to the Master Server (API OUT).

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

1) Provide an API for receiving analytics data from the 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 learner 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 "Virtual labs authoring tool" and creates a new Game Project. The project has automatically some scenes inside, such as the "Main menu", "Login", "Help", "Credits", "Options", "SceneSelector", and one empty "Educational Scene". The "SceneSelector" scene is where the user can select among several "Educational Scenes". The senior educator then uploads the necessary assets of the scene, e.g. for WindEnergyLab these are terrains, wind turbines, buildings, various scenery decorations. For each Asset, the educator defines their category and their parameters. Then, a simple educator edits the first Educational Scene or creates new Educational scenes for editing. A scene can be edited by placing certain assets to certain positions, or modify some parameters of the Assets, e.g. average Wind speed.

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 assets 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.

1.1 Amendment to original architecture

As regards the internal organization of the architecture, we have made a parallax of the original architecture that allows the game authoring to be more flexible. In D4.1, we had proposed an architecture in WordPress that almost meets the architecture of Unity, which is outlined in Figure 1.3. Namely, each “Game Project” is a CPT (custom post type) that has the taxonomy “GameProjectType” which defines the kind of “Game Project”, e.g. Wind Energy, Chemistry etc. Each Game Project consists of Scenes. Scenes are defined also as a CPT with two taxonomies, namely a) ParentGame, which defines to which Game Project this scene belongs to, and b) SceneType that defines the kind of the scene, e.g. “Main Menu”, “Help”, “Educational Simulation Scene”, etc. Assets3D were defined also as CPT that had two taxonomies, namely a) ParentScene, i.e. the scene that the Asset3D belongs to, and b) BehaviorType, that defines the behavior of the Asset3D, e.g. Terrain, Energy Producer, Energy Consumer, etc. The user can drag and drop an Asset3D to a scene, in order to be instantiated, and each scene is saved in a json file. However, this structure caused duplicates in Assets3D, because the Assets belonged to Scenes, and in every Scene, the user had to upload the same Asset3D again in order to be available for the Scene. Therefore we have made a change that it is shown in Figure 1.4. Now each Asset3D belongs straight to a Game Project. This allows all the Assets3D to be available to all the Scenes of the Game Project, and therefore, to avoid duplicates. Now, the Asset3D has also two taxonomies but the first differs, namely the first is a) the Game Project that the Asset3D belongs to, and the second is the same, namely b) the BehavioralType that defines the type of the Asset3D.

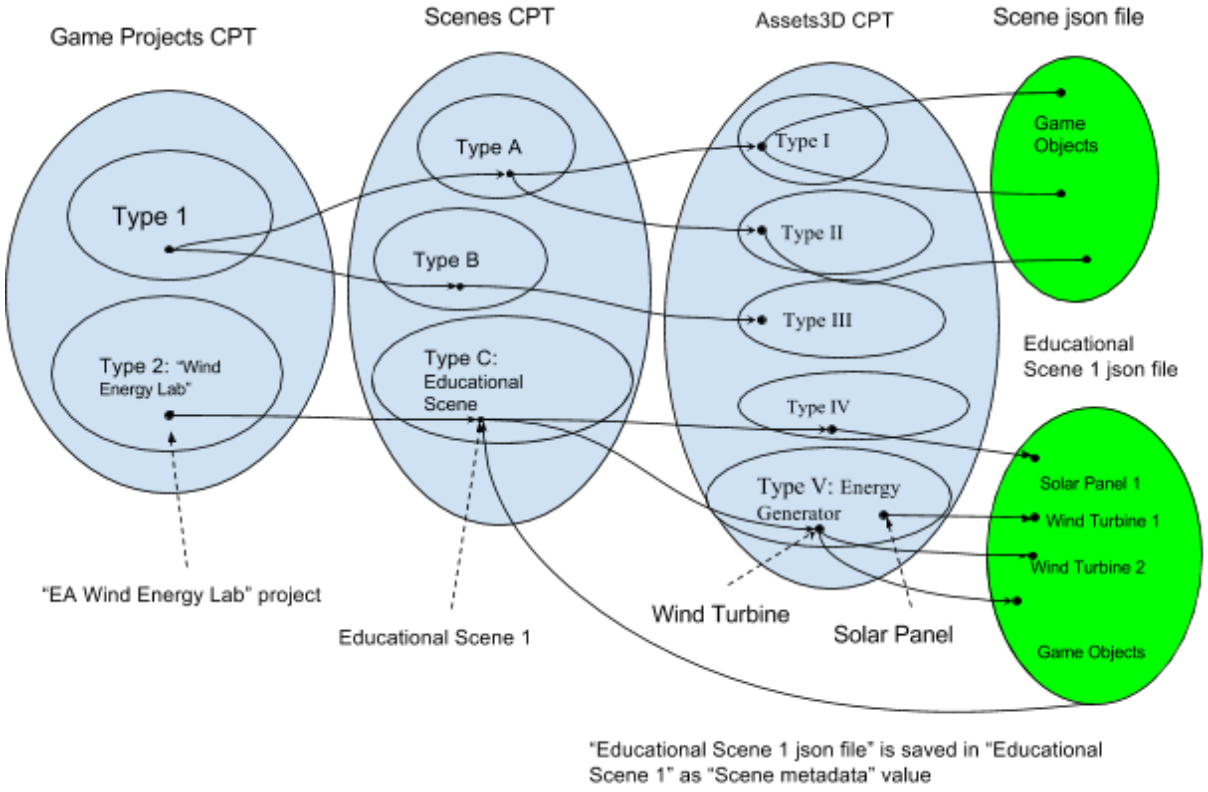

Figure 1.3: Original architecture of the wordpress plugin. Each Asset3D belonged to a Scene.

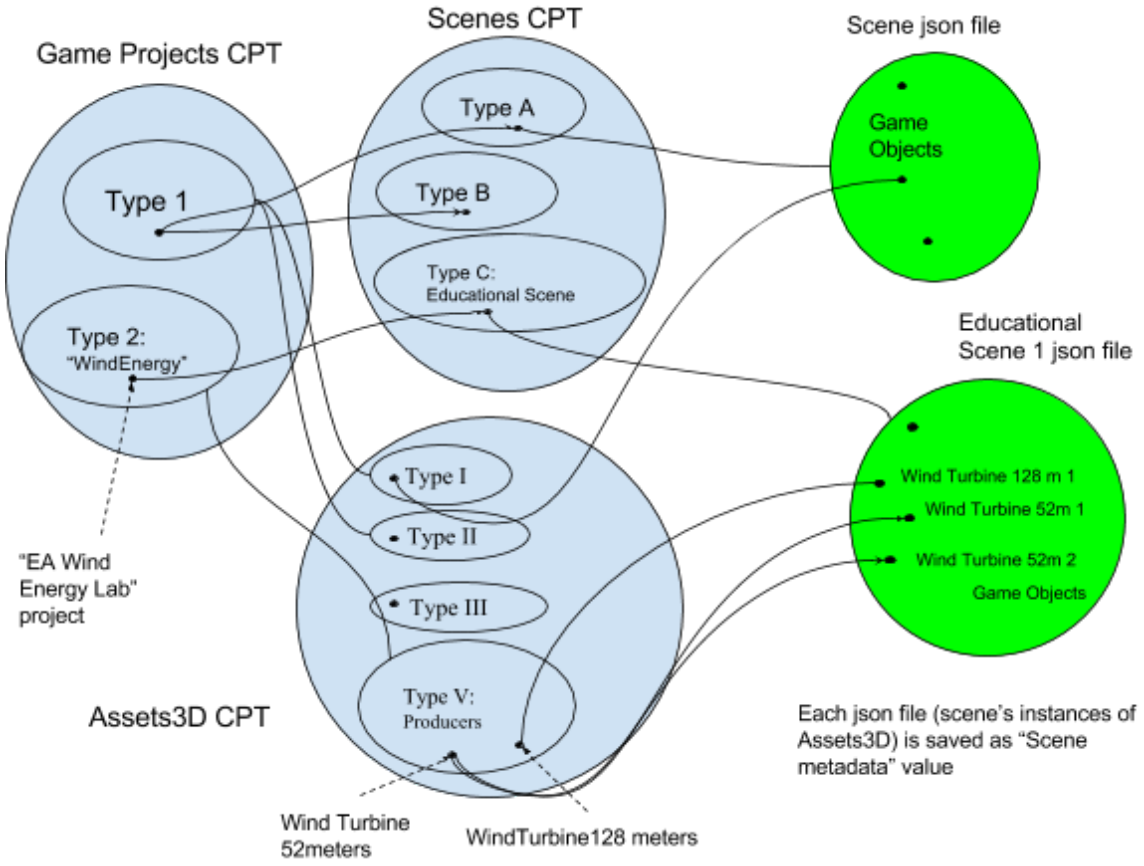

Figure 1.4: New architecture. Now, each Asset3D belongs to a Game Project.

2. Virtual labs authoring tool

The virtual labs authoring tool consists of three components, namely a) the front-end that shows simple visualizations to the educators in order to author games, b) the back-end that shows advanced visualizations to administrators of the website and how to define new Game Project, Scenes, and Assets3D taxonomies, and c) the assembler-compiler that combines all Game Scenes, Assets and Settings into a Unity3D project and compiles it into a game. These components will be outlined next.

2.1 Front-end interface

The front-end interface is a user-friendly Graphical User Interface (GUI) that allows the management of Game Projects, Scenes, and 3D Assets, effectively targeting novice users, that have limited or no knowledge of game authoring processes. The overall organization of the GUI is presented in Figure 2.1.

Firstly, after successfully logging in to the website, the user enters the *Game Project Manager* screen that offers functionality to create a new game, and edit or delete an existing game. Next, the *Scenes Manager* screen is shown, in which an existing scene can be edited or deleted, and a new scene can also be created. There is a link to the *3D Asset Manager* screen and if desired, the user can compile the full game from this screen.

If the user selects to edit a scene, then the *Scene Editor* screen appears, with contents that depend on the type of scene. For 2D scenes a form is shown that the user can submit to change its data. For 3D scenes a fully functional 3D editor is show so that the user can spatially manipulate and arrange 3D assets in a plane. All screens are described in greater detail in the subsequent sections that follow.

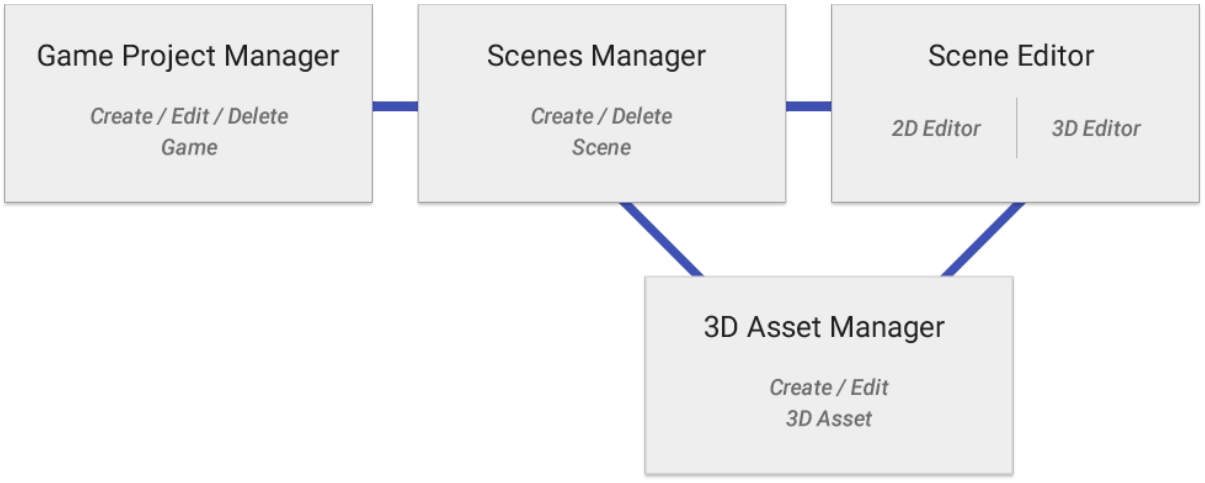

Figure 2.1: Overall organization of the front-end GUI.

2.1.1 Game Project Manager

An implementation of the Game Project Manager is shown in Figure 2.1. In this screen, the educator can create a new game project and delete or edit an existing one. To access a preexisting project, the educator must click on one of the list entries at the left Projects section. There is also a delete button that creates a warning popup window to make sure that the educator really wants to delete this project. A new game project can be created by entering the title of the game project and by selecting its type. When clicking on the CREATE button, a new game is successfully created and the educator is transported to the Scenes Manager screen.

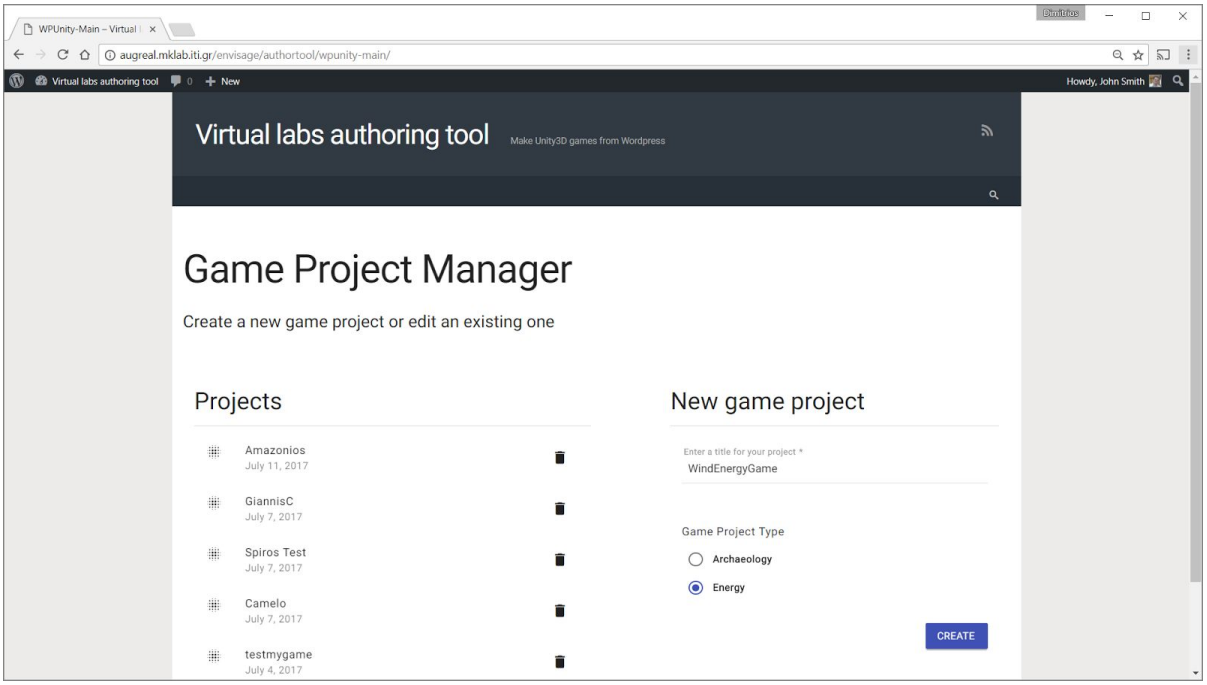

Figure 2.1: Implementation of the Game Project Manager

2.1.2 Scenes Manager

As shown in Figure 2.2, in this interface each scene of the game is represented as a card. Each card has a thumbnail of the scene that also serves as a link to the scene editor, a scene title, a description and two buttons for the edit and delete scene functionalities. There are some scenes that the game project manager creates by default. These scenes are required and cannot be deleted so the delete functionality is disabled, although all scenes that are created by the educator are deletable.

In this screen there is also the 'Compile Game' button that when clicked initiates the compilation process of the whole game. A new screen appears that allows the educator to compile the game in various formats such as WebGL, Windows, Mac, or Linux. Upon successful compilation a link is provided to download the game in zip format. For WebGL games, a second link also appears to play the game in the web browser.

By clicking on the 'Add New Scene' button, a new section expands that enables the creation of a new scene by filling in the necessary information. This information includes a title, a

description and an image that serves as a scene thumbnail.

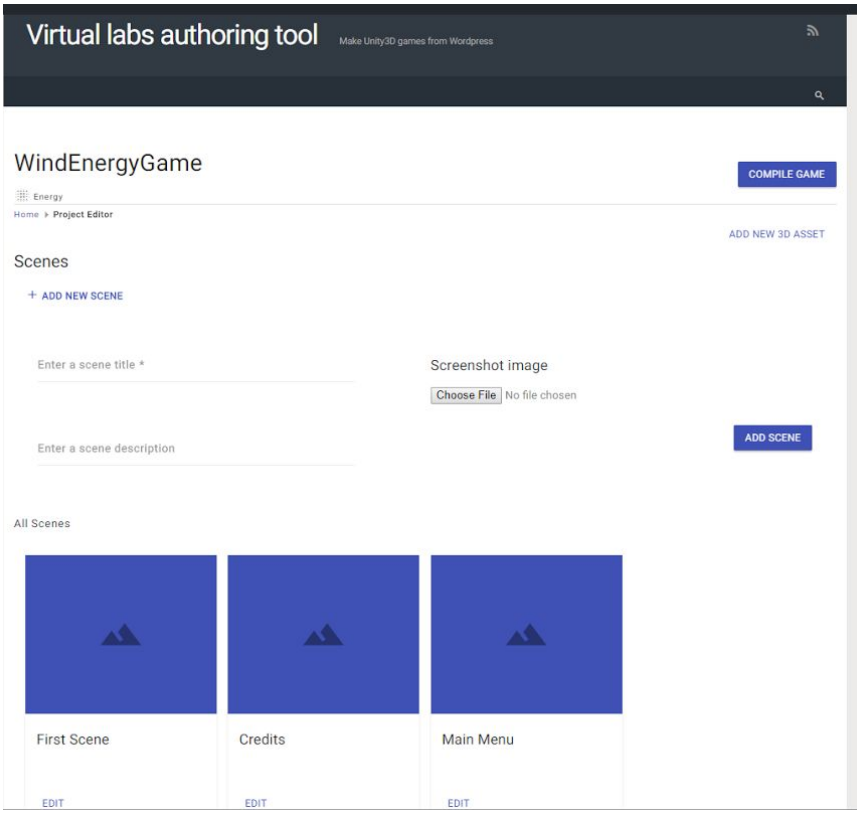

Figure 2.2: Implementation of the scenes manager.

2.1.3 Scene Editor

When the scene that is being edited is two-dimensional, the 2D scene editor launches that is in fact a form with fields that vary according to the scene. A 2D scene editor implementation screenshot can be found in Figure 2.3.

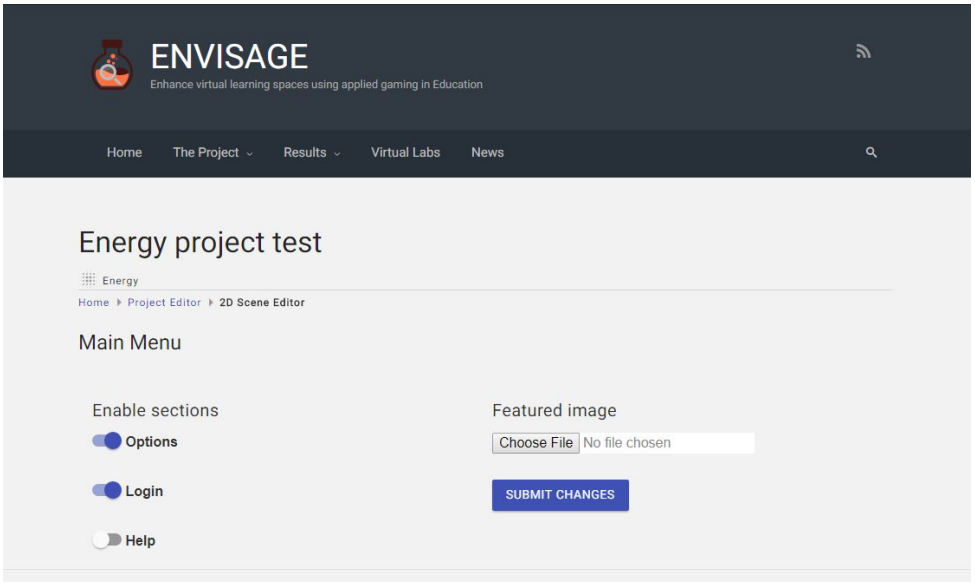

Figure 2.3: Scene editor screen for 2D scenes.

If the scene that is being edited is three-dimensional then the 3D scene editor launches.

We have developed a web-based 3D Editor to enable modifications of a scene through a web-browser. In order to achieve this fast, we used the three.js[1] library that allows to develop 3D graphic elements using HTML5 and WebGL through high level commands. Three.js allows saving a scene in the JSON format where we have standardized our own custom structure that serves the need of converting the scene setup to the Unity3D scene format. A screenshot of the 3D editor can be seen in Figure 2.4.

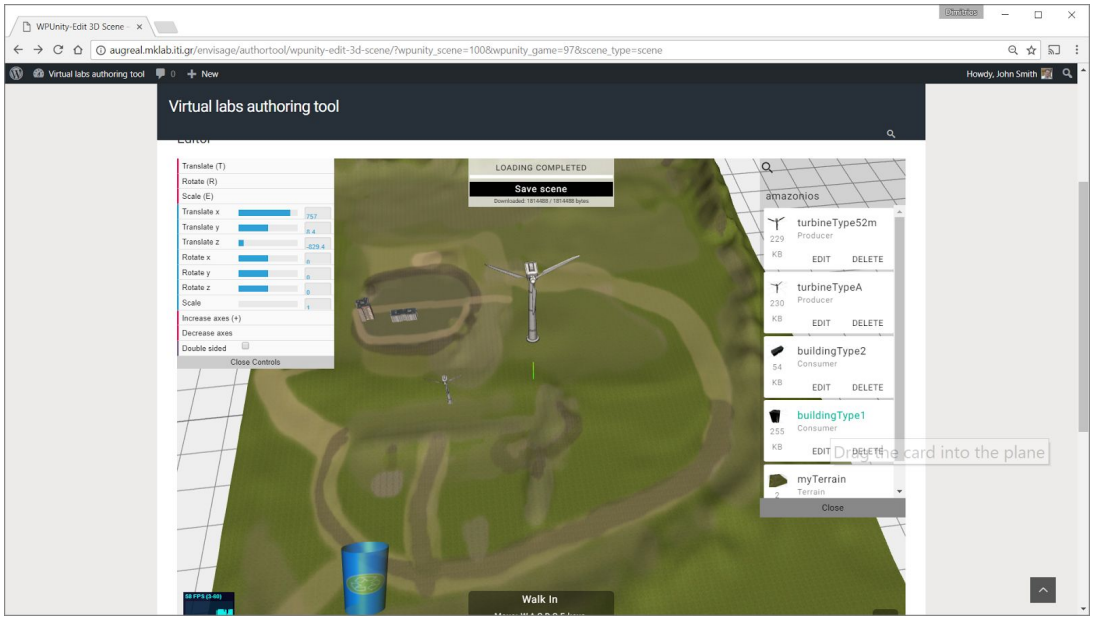

Figure 2.4: A scene editor for the web using Three.js.

The 3D Editor screen consists of three basic parts, namely the 3D view of the Scene where

[1] http://threejs.org

the user can manipulate 3D objects on a plane, the left panel that features controls and editable parameters of each selected object, and a right panel where all available 3D assets are listed and can be dropped inside the 3D plane, edited or deleted. There is also a search function for the scenarios where one Game Project has numerous 3D assets.

The main functionality of the 3D editor is to allow an educator to drag and drop 3D assets on the 3D plane. This action adds an instance of the 3D asset to the scene. Multiple instances of the same asset can exist in the scene, i.e. multiple wind turbines. The educator can edit the rotation, position, and scale of an instance either through GUI controls (gizmos) or by entering numerical values for a more accurate result in the left panel. 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 editing functionalities, can convert a three.js scene into a JSON file. A three.js scene is comprised of 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 inside a JSON file following a protocol. We have defined our own protocol as for the time being there is no standard format for saving three.js 3D scenes. If the educator re-opens the 3D editor for a particular scene, then the JSON file is loaded and the three.js scene is recreated exactly like it was saved the last time it was edited. We decided to use the JSON format instead of the Unity3D YAML scene format because it is more compatible with web technologies such as three.js. Only when the game is compiled, the JSON scene is converted into a Unity3D YAML scene.

In a recent version of the plugin, game analytics are embodied in a tab next to 3D editor as shown in Figure 2.4b. The analytics contain information regarding the entities such as the game, the scenes, and the assets. The parameters for each entity regard the frequency of use, the duration of use, game score statistics etc. that are described in great detail in Sections 4,5 and 6.

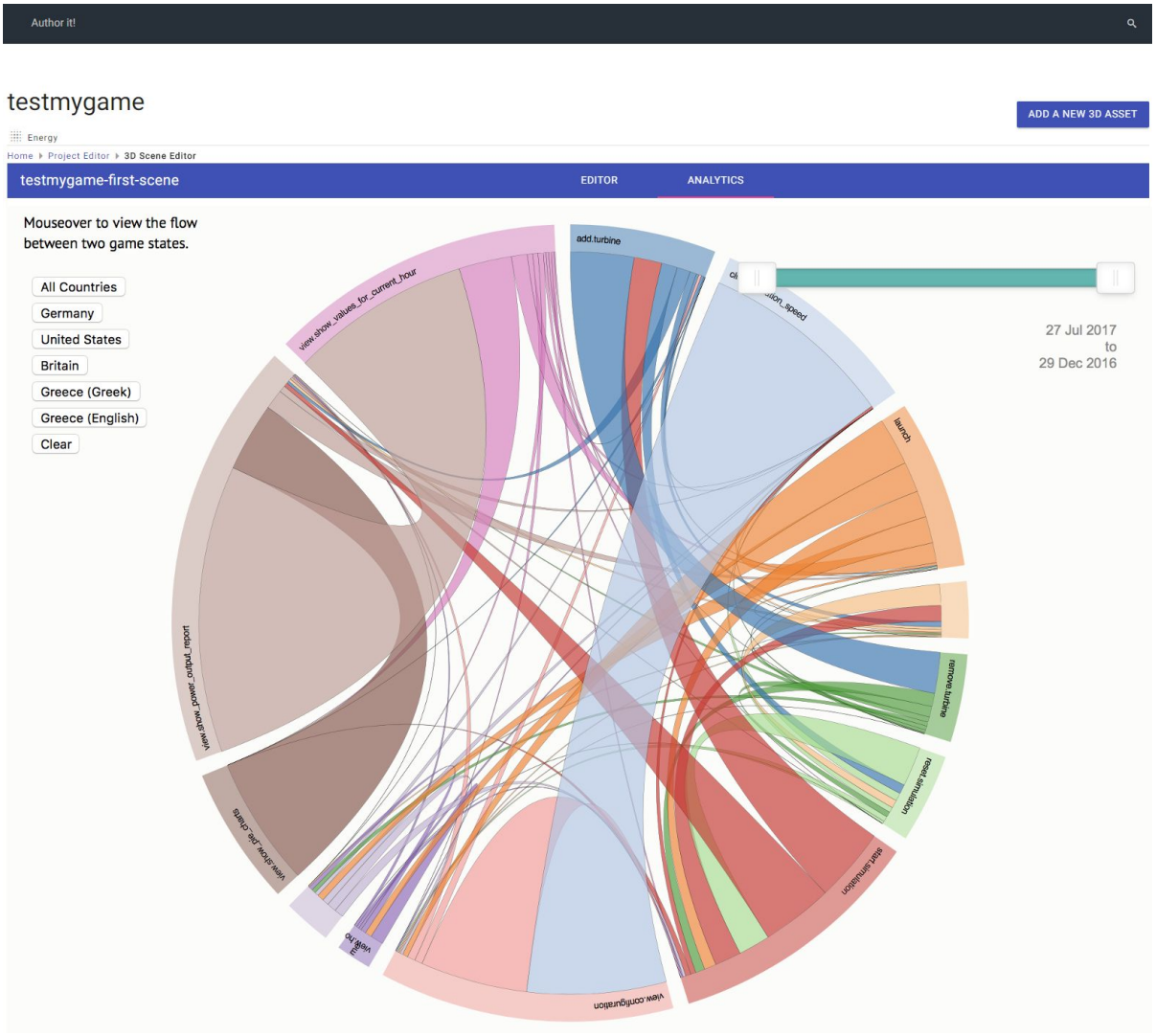

Figure 2.4b: Analytics are displayed in a tab next to scene 3D editor.

2.1.5 3D Asset Manager

The educator can upload a new 3D asset using the “Add new Asset” button, which is only available for senior educators, and is accessible from the Scenes Manager or the 3D editor screens. The 3D Asset Manager screen is shown in Figure 2.5. Each 3D Asset has various fields such as files for 3D representation (obj, mtl, jpg texture) and various fields for its parameters (e.g. power consumption mean and variance). The category of the asset affects the kind of fields of the asset. Further information about the categories of the assets will be described in Section 3 for the certain use case.

Analytically, the steps to create a new 3D asset are a) select the category of the asset based on the type of the current game project, b) write the title and an optional description, c) upload the 3D representation files that include an mtl (material) file, an obj (mesh) file, and a jpg texture file, and d) set the asset fields based on its category. The 3D model is rendered in a panel and the user can save a snapshot of the 3D model to be used as an icon during the

3D Editor.

A 3D asset can be edited, when clicking on the 'Edit' button from the 3D Editor (Scene Editor) screen. All information of a 3D asset can edited except from its category.

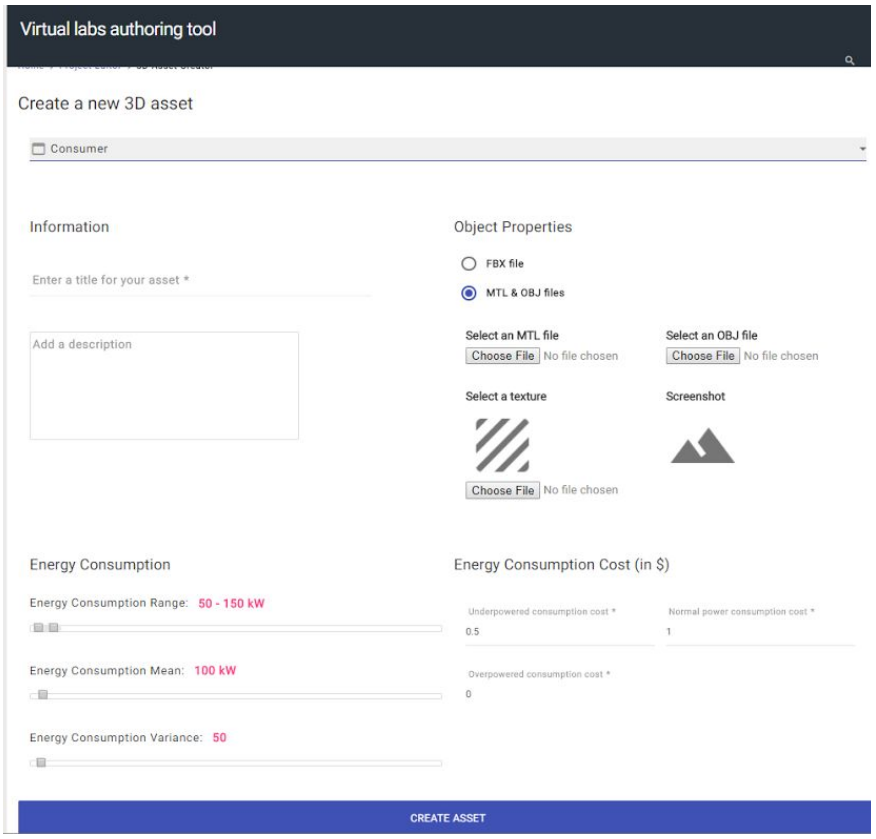

Figure 2.5. The 3D Asset Manager for the energy "Consumer" asset of the "Energy" Game Type.

2.2 Wordpress Back-end

This section is addressed to administrators of WordPress sites, and describes how to augment the capabilities of the virtual labs authoring tool for supporting more game types, scene types, or asset types. It is assumed that basic knowledge of WordPress' back-end is already available. When entering the back-end, our plugin causes three items to appear in the left-hand navigation bar as shown in Figure 2.6, namely "Game Projects", "Scenes" & "Assets 3D". These items include all the functionalities that are related to the plugin and they will be described analytically in the following lines.

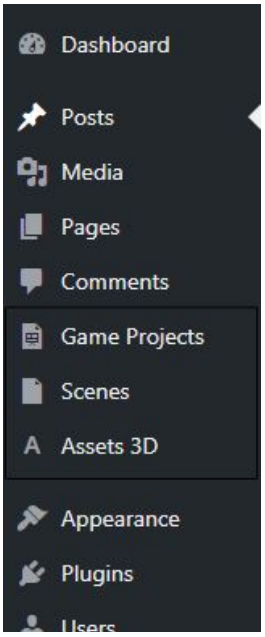

Figure 2.6. Back-end navigation bar with the 3 new items, namely Game Projects, Scenes, and Assets3D.

2.2.1 Game Projects

The "Game Projects" item refers to the Game Projects entity of Unity, and it has 4 sub-items as shown in Figure 2.7, namely "All Game Projects", "Add New", "Game Categories", and "GameTypes". The administrator can view and edit all Game Projects of all authors presented in a list. Via the "All Game Projects" screen, the educator can select the Game Project he wishes to edit, delete, or view. A bulk edit feature allows educators to change certain fields for a group of Game Projects. "Add New" creates a new Game project.

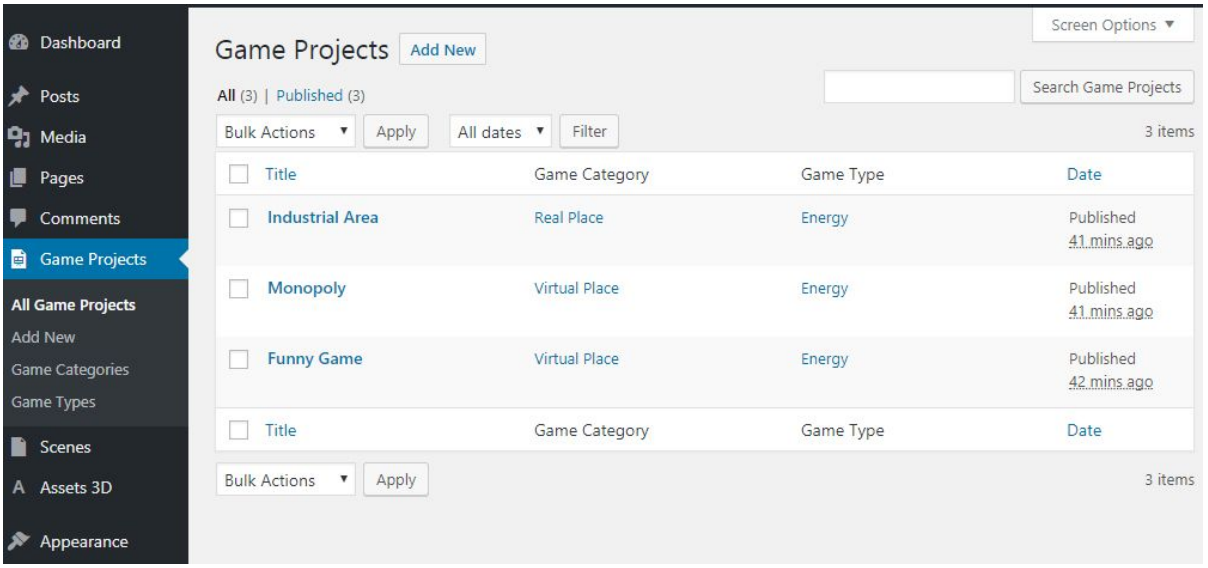

Figure 2.7. "All Game Projects" screen at back-end.

"Game Categories" can be used for the categorization of games into customly defined

categories for searching purposes only. It is the standard taxonomy entity for all post types of Wordpress.

"Game Types" is our defined taxonomy which is used for the functional categorization of Game Projects into different types of Game Projects. The terms for this taxonomy are our own terms "Energy" or "Chemistry", as shown in Figure 2.8. Each term has its own metadata fields. Each term has namely the 16 fields for the Project Settings files (Figure 2.9).

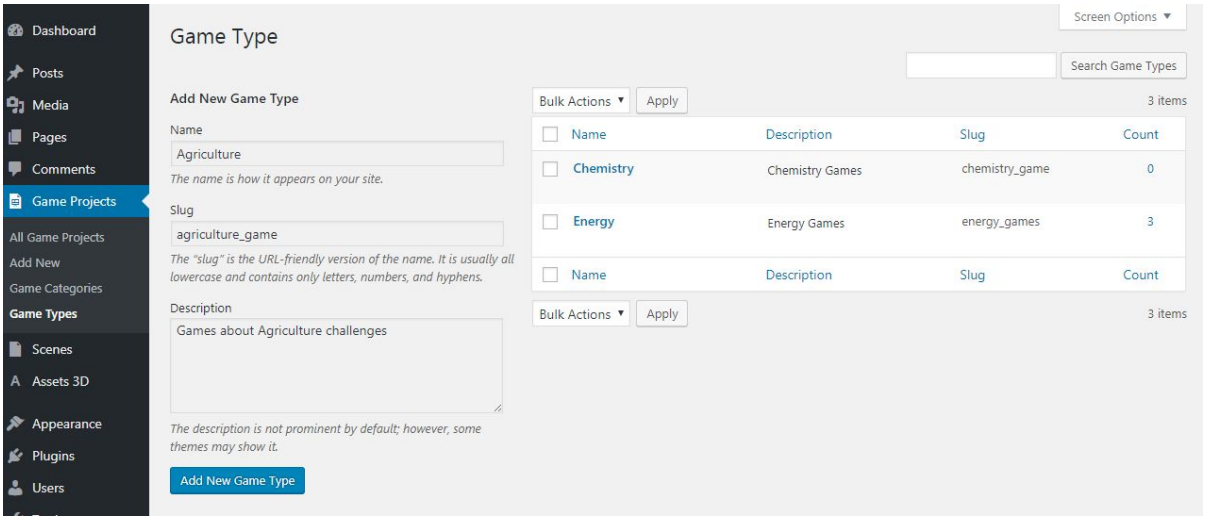

Figure 2.8: Game Types allows the Virtual Labs authoring tool to create various types of Games.

Analytically, in Figure 2.8, on the left side, a tab allows to create new "Game Type" terms ("Add New Game Type") by filling in the Game Type name (e.g. "Agriculture"), its slug (an abbreviation of the type name), a brief description about the new type of game (e.g. "Games about agricultural challenges"), and pressing the "Add New Game Type" button.

By clicking on the GameType term, e.g. "Energy", Figure 2.9 appears that enlists the 16 metadata fields for this term. These metadata fields contain the 16 texts that correspond to Unity Project Settings files in ProjectSettings folder. These text fields work as patterns and they are copied to every "Energy" game that it is generated. They can be edited by clicking the Edit button. Concisely, the 16 Project Settings are the following:

Audio Manager (refer to AudioManager.asset unity files), *Cluster Input Manager* (ClusterInputManager.asset), *Dynamics Manager* (DynamicsManager.asset), *Editor Build Settings* (EditorBuildSettings.asset), Editor Settings (EditorSettings.asset), *Graphics Settings* (GraphicsSettings.asset), *Input Manager* (InputManager.asset), *Nav Mesh Areas* (NavMeshAreas.asset), *Network Manager* (NetworkManager.asset), *Physics2D Settings* (Physics2DSettings.asset), *Project Settings* (ProjectSettings.asset), *Project Version* (ProjectVersion.asset), *Quality Settings* (QualitySettings.asset), *Tag Manager* (TagManager.asset), *Time Manager* (TimeManager.asset), *Unity Connect Settings* (UnityConnectSettings.asset).

Particularly, the EditorBuildSettings, that contains the scenes to be compiled, is the only

field edited automatically in order to include any new scenes created by the educator.

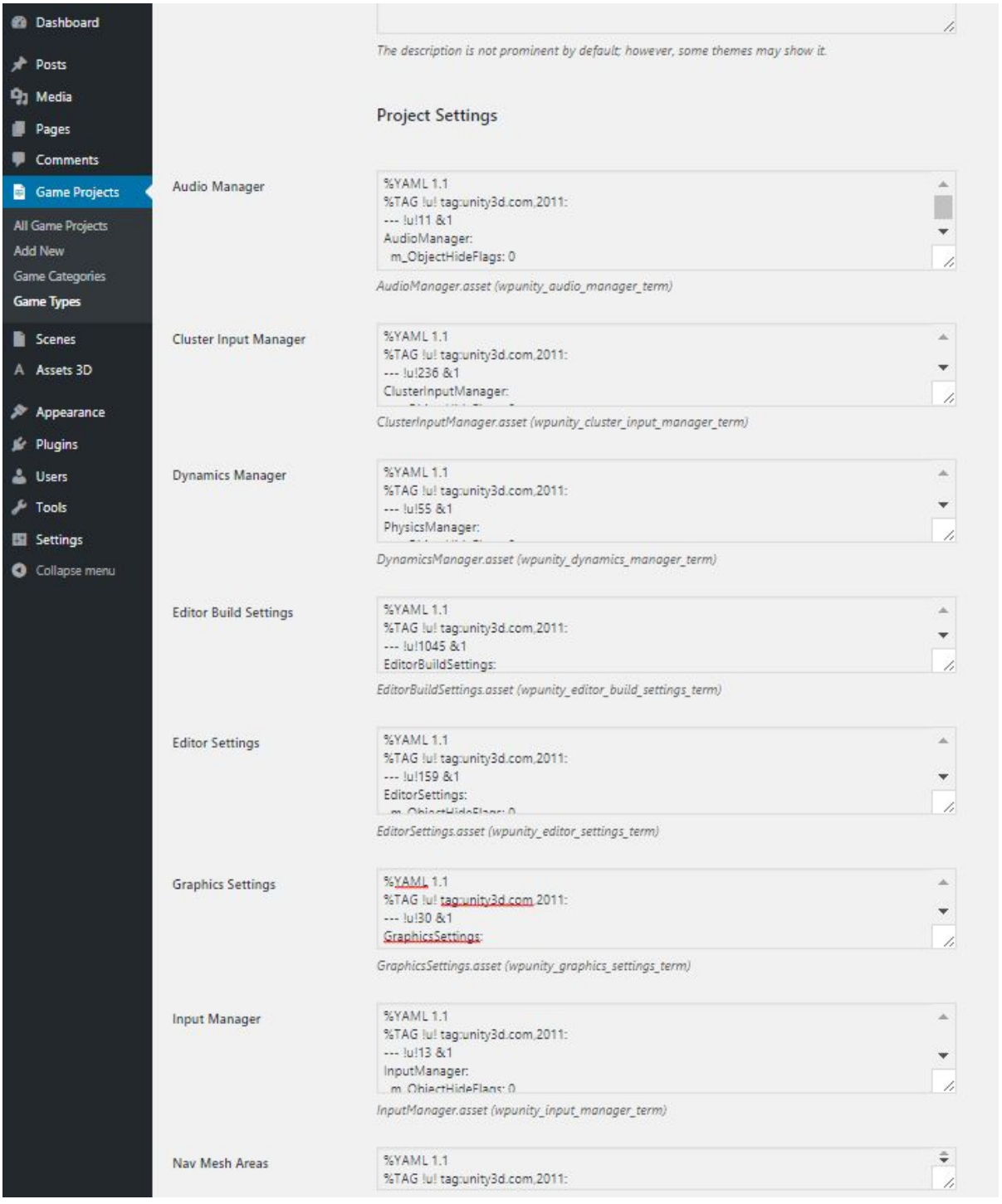

Figure 2.9: “Energy” Game Type Screen with default Project Settings

2.2.2 Scenes

Scenes refer to any scene generated for any game. The “All Scenes” button leads to the list of scenes as the example shown in Figure 2.10. Each row in the list corresponds to a Scene. A row consists of the following columns, a) Title (the title of the Scene), b) Scene Game (the game that this Scene belongs to), c) Scene Type (the functional category of the Scene), and d) Date of last change. The administrators can select the Scene or Scenes they wish to edit, delete, or view. Multiple Scenes can be selected for deletion and for editing.

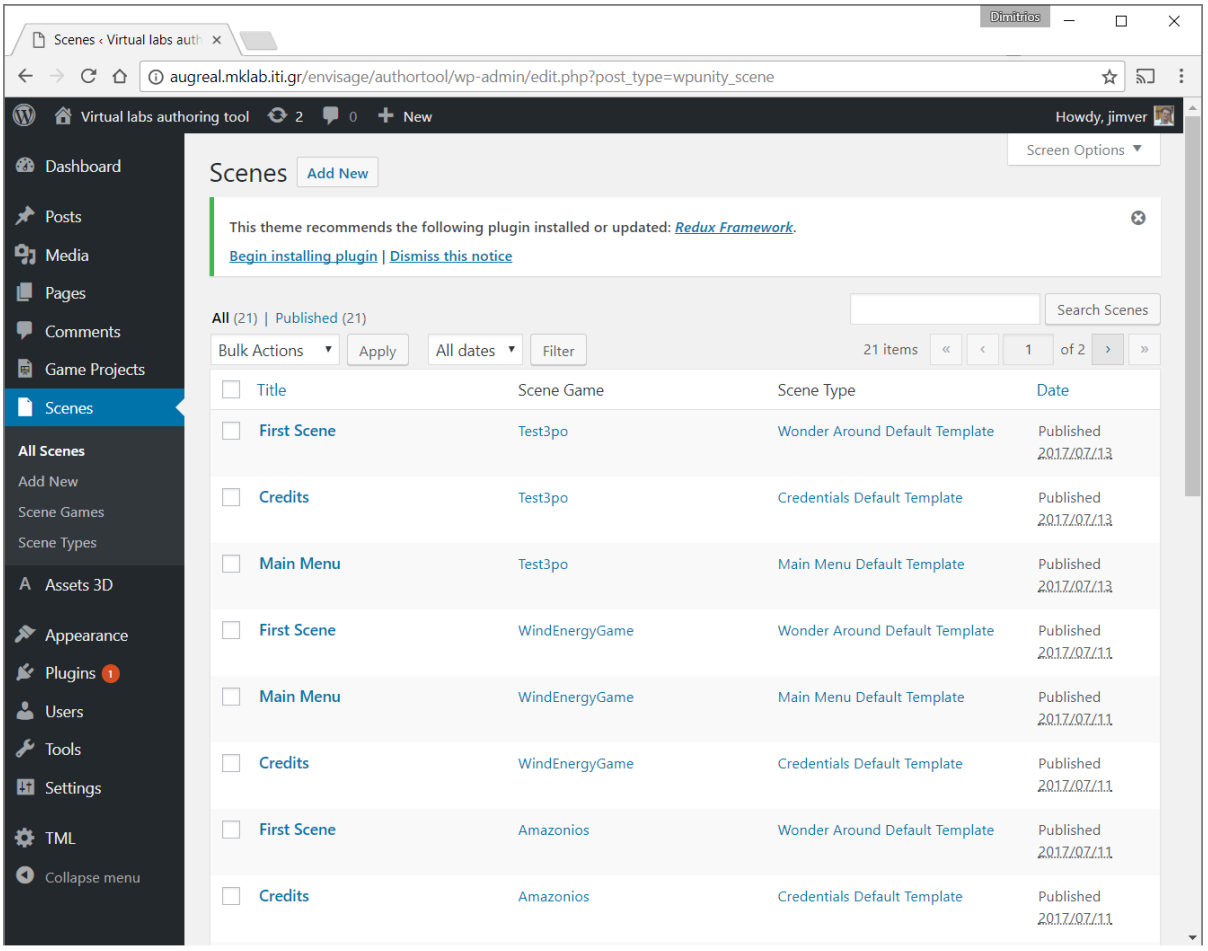

Figure 2.10: Editing Scenes from the back-end.

Every time a new Game Project is created, the following scenes are automatically generated.

Main Menu – A 2D Scene that navigates the learner through his experience.

Credits – A 2D Scene that provides information about the organization that developed the game and should be acknowledged.

First Scene – An initial 3D Scene of the Game which can be edited by the educator as the first scene of the whole game.

The following scenes are created automatically during the compiling process:

Scene selector: A 2D scene that allows the learner to select an educational scene among several scenes.

Help: A 2D scene that provides instructions to the learner.

Login: A 2D scene that provides input fields to allow the learner to login.

Settings: a 2D scene that provides the operating system game settings (screen resolution, graphics quality).

Their parameters can be found inside the interface of the Main menu.

In the left sidebar, the "Scene Games" subitem contains all the "Game Projects" as categories for the Scenes. Each Scene can belong only to one "Game Project". There is no need for modifying this taxonomy as it is automatically created when creating a game.

In the left sidebar, "Scene Types" is our custom taxonomy that refers to the types of the of scenes. The terms for this taxonomy are namely "Main Menu Default Template", "Credits Default template", and "Educational Scene default template". Each term has several meta-datafields that correspond to YAML code patterns to be used for generating the game project.

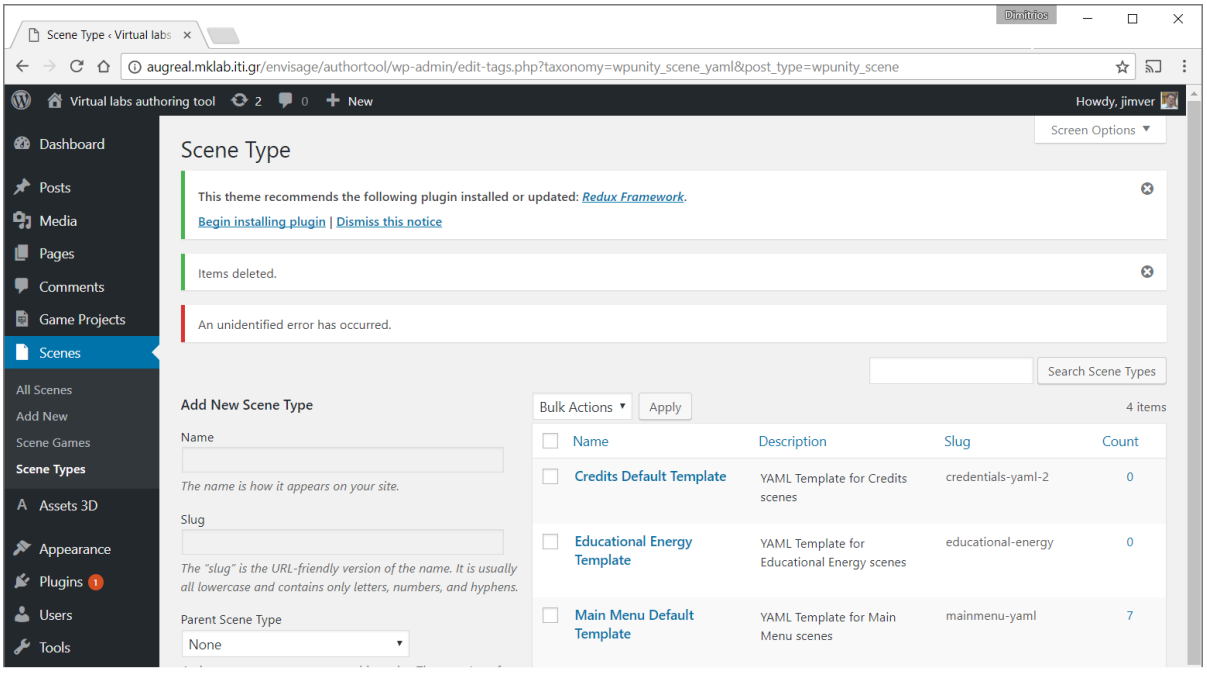

Figure 2.11: Scene Types is the functional categorization of Scenes.

2.2.3 Assets 3D

By pressing the "All Assets 3D" button, the list of all Assets of all games is shown as the example presented in Figure 2.12. The educator can select the Asset or Assets he wishes to edit, delete, or view. Various search and filtering options allow educators to find the Assets that they want to edit or delete. The list consists of four columns, namely the title of the

Asset, the Game Project that it belongs to, and the data of the last change.

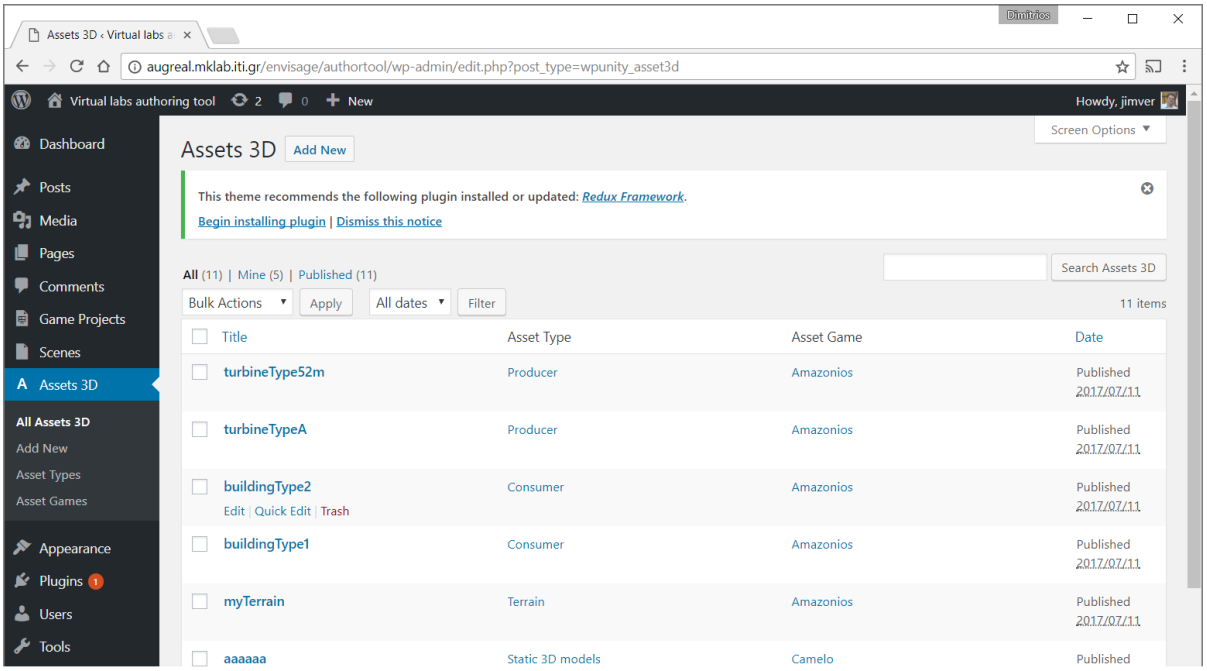

Figure 2.12: List of All Assets in all games.

The column **"**Asset Type" is a taxonomy whose terms are shown in Figure 2.13. It is a taxonomy that allows the categorization of Assets into categories of certain behavior, e.g. for the "Energy" lab we have "Consumers" or "Producers" to indicate that the asset is consuming or producing energy. An asset can belong only to one "Asset Type". The taxonomy terms (e.g. "Consumer") have metadata fields that are filled with YAML code that it is used as a pattern for the Assets that belong to this Asset Type when compiling the game.

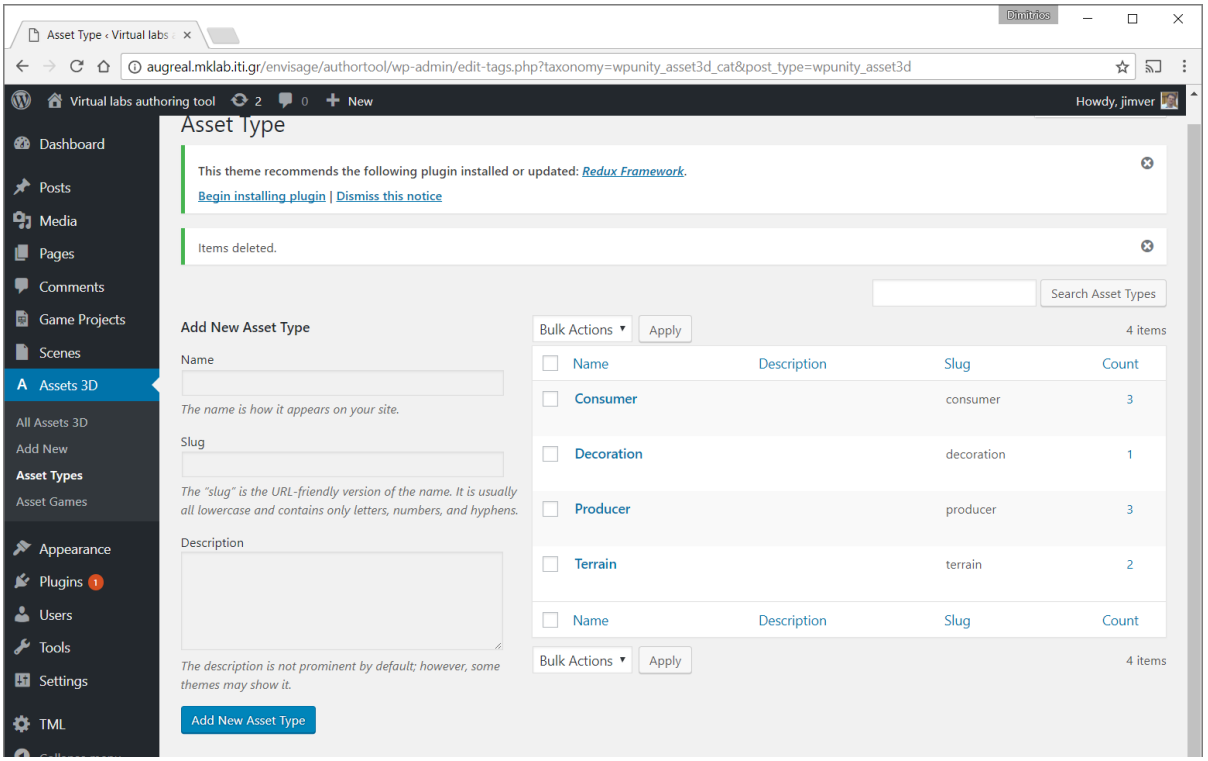

Figure 2.13. Asset Types terms for the “Energy” Game Types.

A new “Asset Type” term can be created from “Add new Asset Type” from Asset Types as in Figure 2.13 by providing the following fields:

Name – The actual name of the term.

Slug – Is the name abbreviation (lowercase, letters, numbers, hyphens, no spaces).

Description – A description of the term to allow educators to understand what this asset type term is about.

When clicking on type term, e.g. “Terrain”, the metadata fields for this term appear where the YAML pattern can be edited as in Figure 2.14. The editing of the YAML patterns requires full knowledge of Unity3D and the functionality of the template.

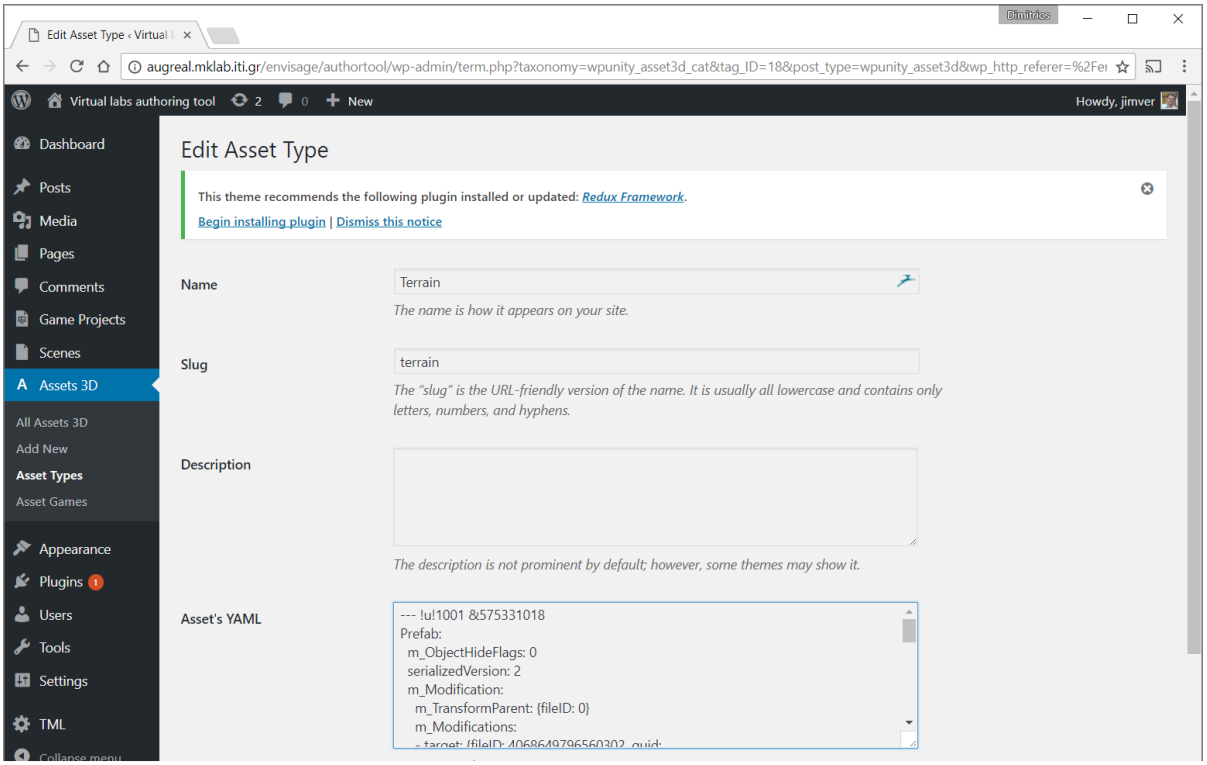

Figure 2.14: "Terrain" term metadata field is the place where YAML patterns can be inserted.

2.3 Assembling and compiling

In order to compile a game, it should first be transformed from our WordPress format into Unity3D project format, namely to make the necessary files that are connected with each other in a meaningful Unity format.

A Unity3D game project consists of 2 folders namely **ProjectSettings Folder** and **Assets Folder**. ProjectSettings Folder contains 16 YAML (yet another markup language) files that store project settings. Their extension is .asset. These 16 YAML files are saved in WordPress as taxonomy term metadata of the custom post type “Game Project”.

When the user presses the “Compile” button, as the example shown in Figure 2.16, a dialogue pops up asking the educator to provide the output format, where Web is the default one. Upon pressing proceed, all the information in Wordpress is assembled to a Unity3D project, and then a compile command is executed by the Unity3D engine. There are several steps for assembling and compiling the project that are explained below.

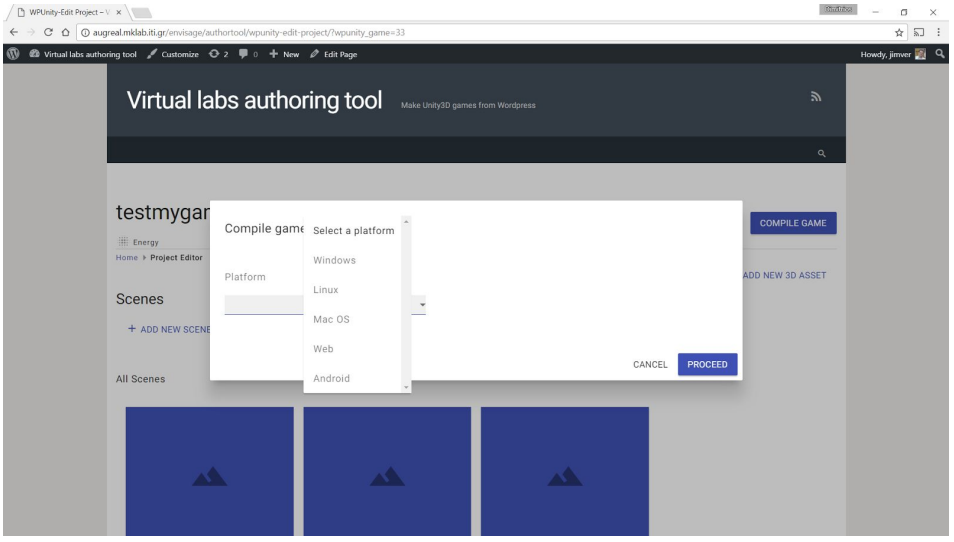

Figure 2.16: Compile button and following interface.

Guidelines for assembling and compiling a game project

1. The following folders (cases sensitive) are generated in “Uploads” with the name of the Game Project plus Unity string, e.g.
    a. “Uploads/myGameProjectUnity”
    b. “Uploads/myGameProjectUnity/ProjectSettings”
    c. “Uploads/myGameProjectUnity/Assets”
    d. “Uploads/myGameProjectUnity/Assets/Editor”
    e. “Uploads/myGameProjectUnity/Assets/scenes”
    f. “Uploads/myGameProjectUnity/Assets/models”
    g. “Uploads/myGameProjectUnity/Assets/StandardAssets”
    h. “Uploads/myGameProjectUnity/builds”

2. **Uploads/myGameProjectUnity/ProjectSettings**: The 16 taxonomy terms meta-data of the WordPress taxonomy GameType where "myGame" belongs to are saved as files in the "ProjectSettings". Changes happen to the following text metadata before saving into a file.

   EditorBuildSettings.asset: EditorBuildSettings.asset contains which scenes will be compiled in the build of the game for Windows, Linux or Mac outputs, e.g.

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!1045 &1
EditorBuildSettings:
 m_ObjectHideFlags: 0
 serializedVersion: 2
 m_Scenes:
 - enabled: 1
        path: Assets/scenes/MainMenu.unity
 - enabled: 1
        path: Assets/scenes/Help.unity
 - enabled: 1
        path: Assets/scenes/Login.unity
 - enabled: 1
        path: Assets/scenes/Options.unity
 - enabled: 1
        path: Assets/scenes/SceneSelector.unity
 - enabled: 1
        path: Assets/scenes/___[EducationalScene_1]___.unity
 - enabled: 1
        path: Assets/scenes/___[EducationalScene_2]___.unity
...
 - enabled: 1
        path: Assets/scenes/RewardScene.unity
```

   as parameters are the names ___[EducationalScene_1]___ and ___[EducationalScene_2]___ and any similar that are replaced with the real names of the Educational Scenes.

   Also the **TagManager.asset,** the file that contains the available tags for the objects that identify their functionality id[2], in the game should be

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!78 &1
TagManager:
 serializedVersion: 2
 tags:
 - consumer
 - producer
 - terrain
```

[2] Where custom tags consumer, producer, and terrain are used in the WindEnergyGame

```
layers:
- Default
- TransparentFX
- Ignore Raycast
-
- Water
- UI
- (26 lines)
m_SortingLayers:
- name: Default
  uniqueID: 0
  locked: 0
```

3. **Uploads/myGameProjectUnity/Assets/Editor:** Here a script in C#, named as "WebGLBuilder.cs" is used to import objects into the Unity3D project such as make materials, meta[3] files, and add scenes into the procedure for compiling to WebGL.

WebGLBuilder.cs

```
using UnityEditor;
class WebGLBuilder {
        static void build() {


      AssetDatabase.ImportAsset("Assets/models/building1/building1.obj",
                                                ImportAssetOptions.Default);

      AssetDatabase.ImportAsset("Assets/models/building1/building2.obj",
                                                ImportAssetOptions.Default);


        string[] scenes = {"Assets/scenes/S_MainMenu.unity",
                           "Assets/scenes/S_Login.unity",
                           "Assets/scenes/S_Help.unity",
                           "Assets/scenes/S_1.unity",
                           ....
                           "Assets/scenes/S_Reward.unity",
                           "Assets/scenes/S_Credits.unity",
                           "Assets/scenes/S_Settings.unity",
                           "Assets/scenes/S_SceneSelector.unity"};

        string pathToDeploy = "build";
        BuildPipeline.BuildPlayer(scenes, pathToDeploy, BuildTarget.WebGL, BuildOptions.None);
        }
}
```

4. **Uploads/myGameProjectUnity/Assets/models:** Here all the Assets3D of WordPress that

[3] If a meta object already exists it will not overwrite a previous one.

should be copied. For each Asset3D, there are several files depending on the category of the Asset3D. These files are explained in Appendix I.

5. **Uploads/myGameProjectUnity/Assets/scenes:** Here all the .unity files of each scene that should be created and filled with content. Such as:
    i. MainMenu.unity:
    ii. Login.unity:
    iii. Options.unity:
    iv. Help.unity:
    v. SceneSelector.unity:
    vi. Credits.unity
    vii. s1.unity
    viii. s2.unity
    ix. …
    x. Reward.unity

    The file .unity.meta is automatically created during the compile process and there is no need to generate this file with another mechanism. The .unity files contain thousands of generated coded lines and it is out of the scope to described them here. A description was provided in deliverable D4.1.

6. **Uploads/myGameProjectUnity/Assets/StandardAssets:** This folder is a prefixed folder with fonts, analytics scripts, game scripts, materials and textures, that are the same across all games of the certain Game Type, e.g. WindEnergy_Auxiliary_Assets. In WordPress, this folder is already stored in

    wordpressunity3deditor/StandardAssets/Energy

    and it should be copied as it is.

7. **Uploads/myGameProjectUnity/builds:** This folder contains the compiled game.

8. Compiling: The game project is compiled for WebGL format from command line. This is allowed through the following commands which depend on the operating system that runs on the server. This could be can be either Windows or Linux (Ubuntu).

In Windows server to WebGL html: The following command exploits WebGLBuilder.cs that was explained in step 3 of the assemblier. All the following content should be placed in a .bat file.

```
set mypath=%cd%
@echo %mypath%
"C:\Program Files\Unity\Editor\Unity.exe" -quit -batchmode -logFile stdout.txt -projectPath %mypath% -executeMethod WebGLBuilder.build
```

In Linux server to WebGL html: The following command exploits WebGLBuilder.cs that was

explained in step 3 of the assemblier. All the following content should be placed in a .sh file.

```
#/bin/bash

projectPath=`pwd`

xvfb-run --auto-servernum --server-args='-screen 0 1024x768x24:32'
/opt/Unity/Editor/Unity -batchmode -nographics -logfile stdout.log
-force-opengl -quit -projectPath ${projectPath} -executeMethod
WebGLBuilder.build
```

output: The output is generated in \builds folder as an index.html and a resource folder.

stdout.txt: contains the log file of the compiling.

Information for compiling to desktop version (Windows, Mac or Linux) can be found in Appendix II.

The monitoring of compiling consists of periodical checks of stdout.txt log file and of monitoring the process in the server's RAM. The latter is achieved with the following commands.

For Windows server -The following php command generates a report in CSV format for which Unity processes are running in the server.

```
$processUnityCSV = exec('TASKLIST /FI "imagename eq Unity.exe" /v /fo
CSV');
```

For Linux server: The same is achieved in Linux with the following command:

```
$processUnityCSV = exec('pgrep Unity');
```

3. Alignments with the use cases

In this section, we discuss the output of the Virtual Labs authoring tool, i.e. the generated game, with respect to each use case. In order for the authoring tool to generate games of a certain type, a template game should be optimized and split into parts that server as YAML patterns for Game Types, Scene Types, and Asset Types. Since the Wind Energy lab was the only available game template during the writing of this deliverable we focus only on the respective use. The Chemistry lab template will be discussed in the following deliverable.

3.1 How a generated game looks like

The games contain by default certain 2D scenes, which allow the basic functionalities of games with GUI elements. These 2D scenes are outlined in the following:

Main Menu - It is the central point of the game where the learner can select what to do next as shown in Figure 3.1. The title, e.g. “Renewable Energy VR Lab” and the image below the title are editable from the virtual labs authoring tool. All the other GUI elements are fixed. An option is provided to hide Login, Settings and Help if the educator does not wish to have these buttons available.

Figure 3.1: Main menu scene

Login - Button loads a Scene where the input fields for the learner name, the surname, and the school can be found. An example is shown in Figure 3.2. The information is encrypted before transmission into a unique identifier that can not be inversed, i.e. there it is not possible to extract a learner’s name from its encrypted identifier. This scene is not editable from the authoring tool.

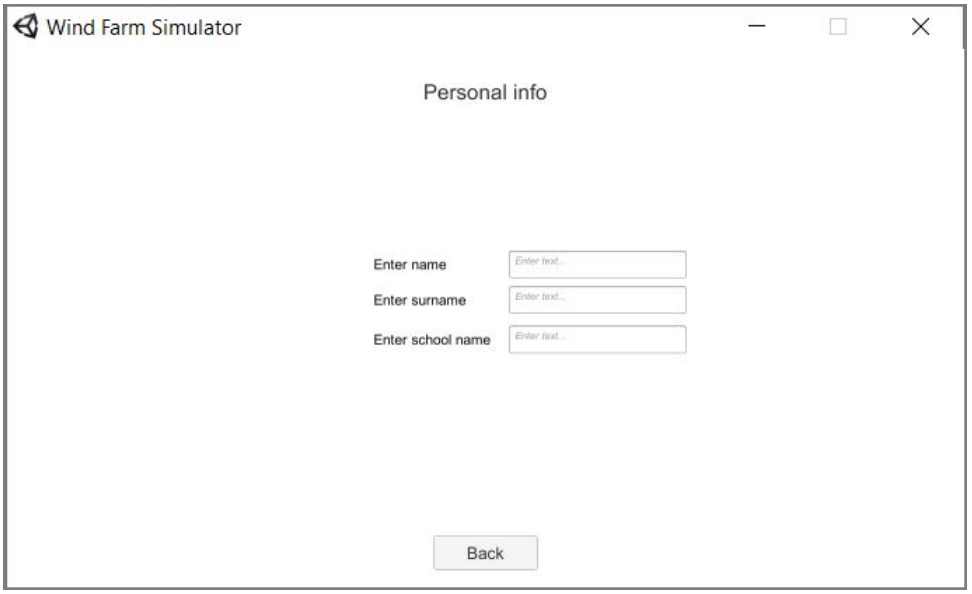

Figure 3.2: Login scene

Settings - Button loads a scene that provides controls for changing screen size and details level as shown in Figure 3.3. This screen is useful in low-end devices that should have a low resolution and details level in order for the game to be played smoothly. This scene is not editable from the authoring tool.

Figure 3.3: Settings scene.

Credits - Button loads a scene where the authors of the game are acknowledged. An example is shown in Figure 3.4. The editable elements by the authoring tool are the image and the description of the authors.

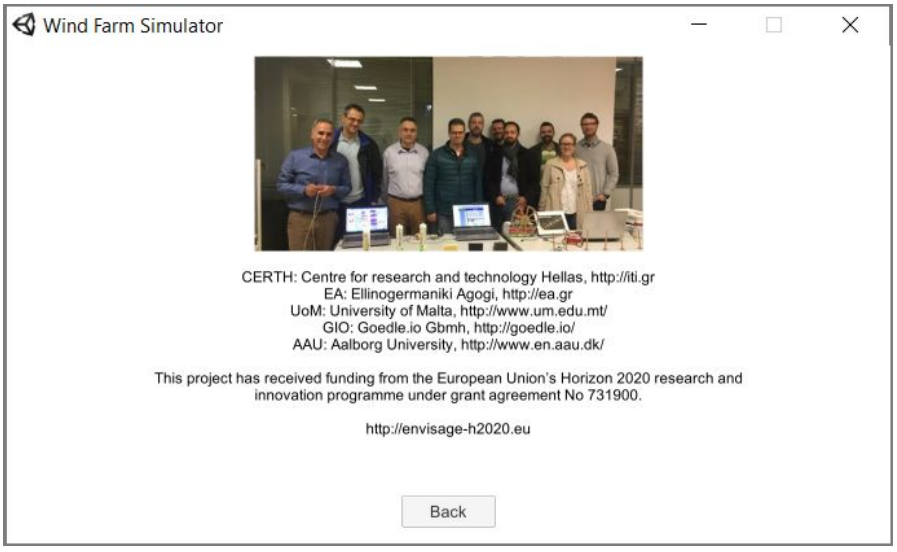

Figure 3.4: Credits scene

Help - Button loads a scene where the learning material is provided in the form of an image and a text as shown in Figure 3.5. Both image and text are editable from the authoring tool.

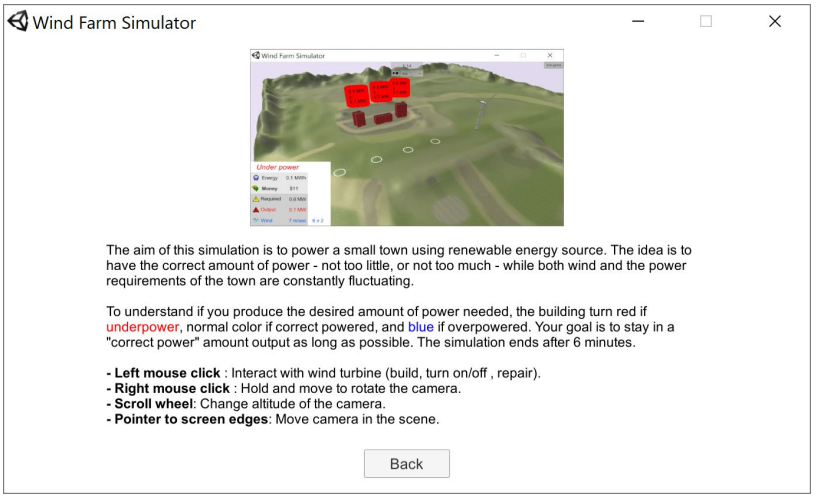

Figure 3.5: Help scene provides the learning material and instructions for the generated games.

Play - Button loads a scene named as "Scene Selector" that allows the user to select an Educational Scene to play among several choices. An example is shown in Figure 3.6. This scene is automatically generated from the authoring tool based on how many Educational Scenes are available. The title, description, and image of the scene are those fed as input during the creation of each scene. The title "Select a Scene" is editable. Next, the Educational Scenes are described.

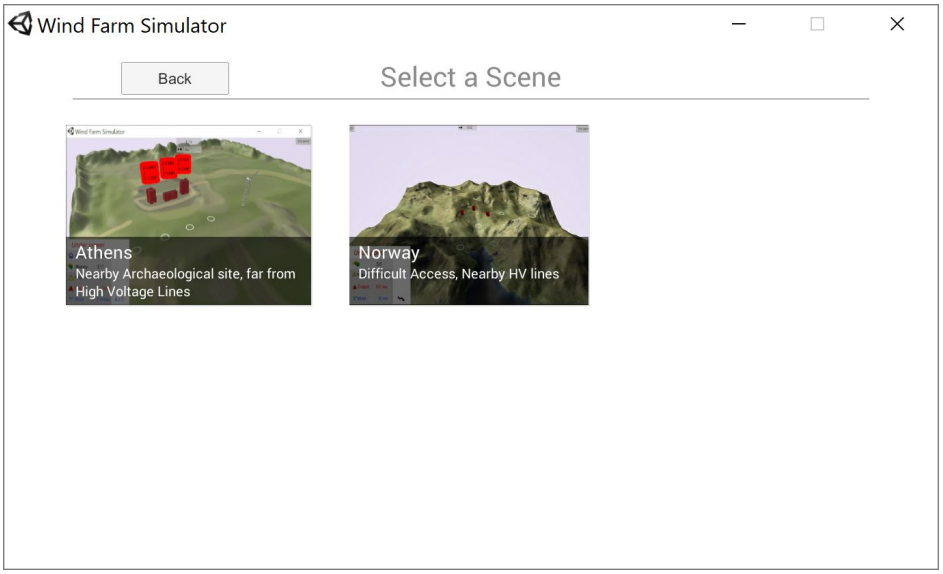

Figure 3.6: Scene Selector scene allows to select a “level”.

Educational Scenes - These are the energy production-consumption simulation scenes that contain the main interaction for achieving the learning objective. An example is shown in Figure 3.7. Here the energy consumers are the buildings which are colored as red indicating that they are underpowered. Above each building, a billboard shows the mean and the variance of the consumption of each building. The circles in the terrain indicate candidate positions for inserting a wind turbine. On hovering above each cycle, information of the candidate turbine is shown. On clicking a cycle, a turbine is built over it. Depending on the size of the turbine rotor the nearby candidate positions are destroyed automatically in a range of 1.5 times of the rotor size. When hovering on each turbine a billboard over the turbine shows the characteristics of the turbine as well as the current output for the current wind speed. On bottom-left, the current state of the game is shown. This state consists of the following metrics: the total energy produced so far, the money earned, the required power, the generated power, and the wind (current, mean, variance). The simulation lasts 6 minutes in real time that corresponds to 24 hours. A turbine can be damaged after some time, and the learner can click on it to repair it with some cost. The editable simulation parameters by the virtual labs authoring tool will be described in Section 3.2. The interactions allowed by the learner will be described in Section 3.3. Upon finishing the simulation the Reward scene is shown which is described next.

Figure 3.7: The simulation of energy production-consumption constitute the educational scenes.

Reward Scene - As shown in Figure 3.8, contains the final score and an overall evaluation for the simulation session. The learner can see his or her score as the money earned, the energy produced and the balance of among overpower, correct power, and under-power time.

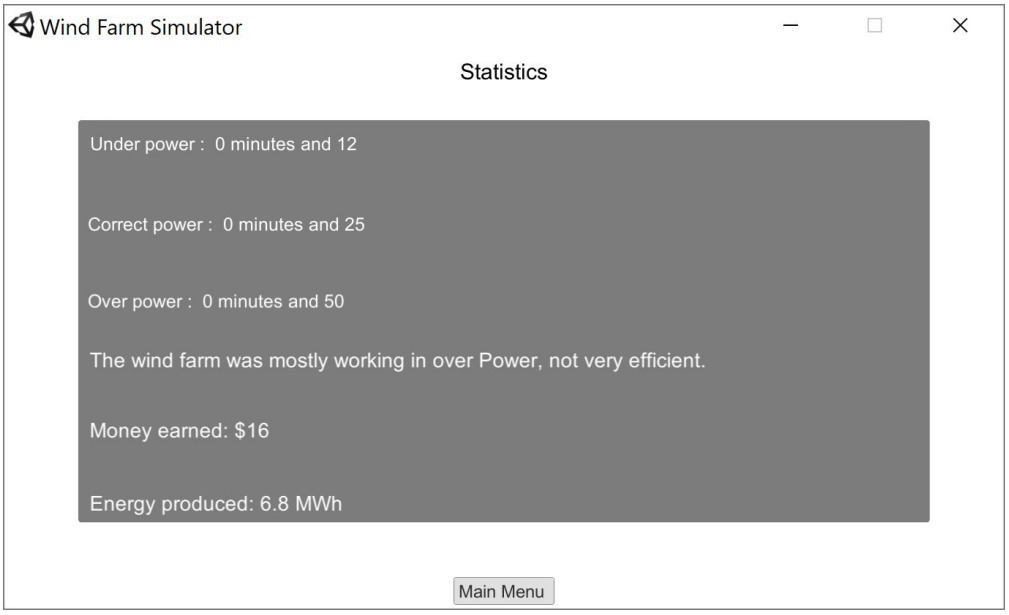

Figure 3.8: The reward scene displays the score of the learner.

3.2 What the educator can do with the Virtual labs authoring tool using the "Energy" lab template

The "Energy" template can be used to generate an arbitrary number of games, with an arbitrary number of Educational Scenes, and with an arbitrary number of game objects per Educational Scene. In the following lines, we will describe which parameters the educator can change by using the vlabs authoring tool. Briefly, the vlabs authoring tool copes with the following requirements.

- Allow the learner to select an Educational Scene to play among several Educational Scenes. Each scene has its own pros and cons that should be explained to learner.
- The energy consumption per scene should be modifiable.
- The wind speed per scene should be modifiable.
- Wind energy turbines should have the following modifiable parameters
    - Power Generation
    - Size
    - Cost to buy

The educator can do the following actions in the first prototype of the vlabs authoring tool

[Action A] Create multiple Educational Scenes: Each of these scenes is a certain area where the wind energy generators can be placed. This action is feasible by pressing the button "ADD NEW SCENE" as in Figure 3.9.

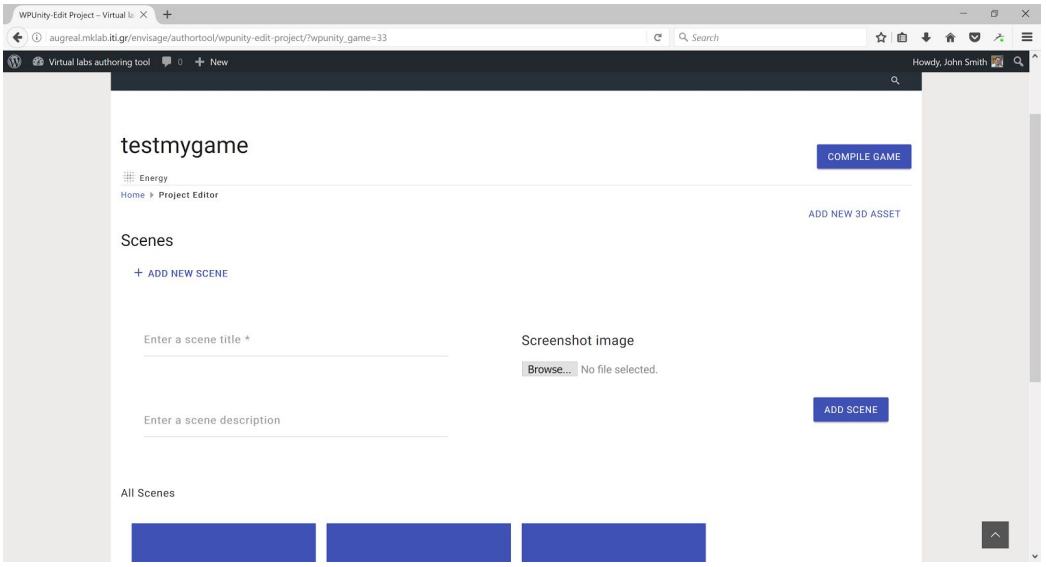

Figure 3.9: Making a new Educational Scene for "testmygame" game project.

In each scene, several game objects can be inserted by the educator. Otherwise, the area is totally empty.

[Action B] Insert a Terrain: A Terrain is a ground where turbines can be placed. This action

is feasible by drag-n-drop an Asset3D to the scene. Figure 3.10 shows a scene with a terrain. Only one Terrain can be placed in a scene.

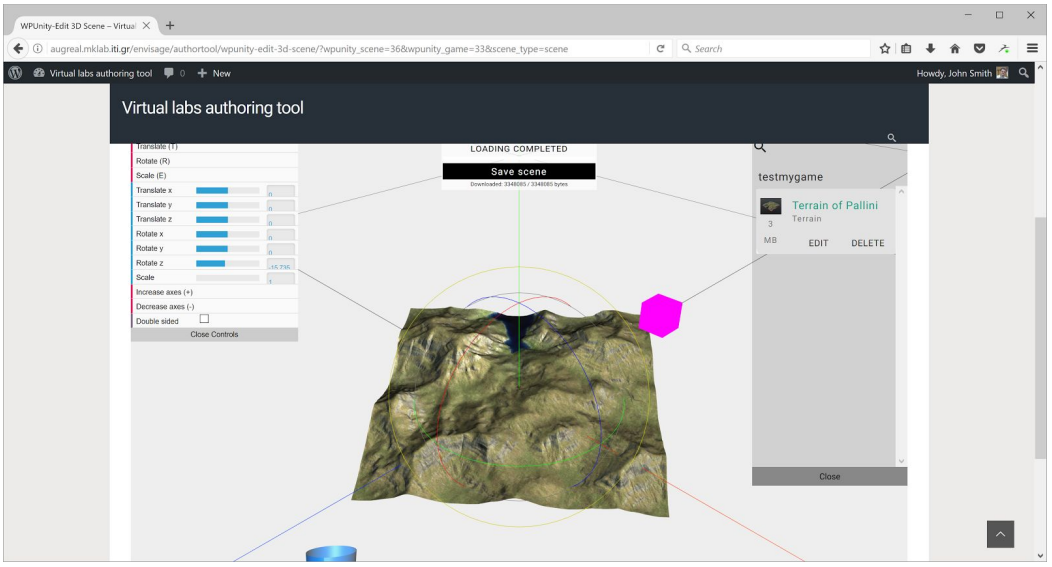

Figure 3.10: Inserting a terrain with drag-n-drop from right toolbar.

The Terrain has the following fields that should be defined by the educator by pressing the “Edit” or “Create new Asset”. The interface for creating a Terrain as shown in Figures 3.11a and 3.11b. First, in Figure 3.11a, the user should select the category of the Asset3D which in our case is Terrain. Then automatically several fields popup that are shown in Figure 3.11b.

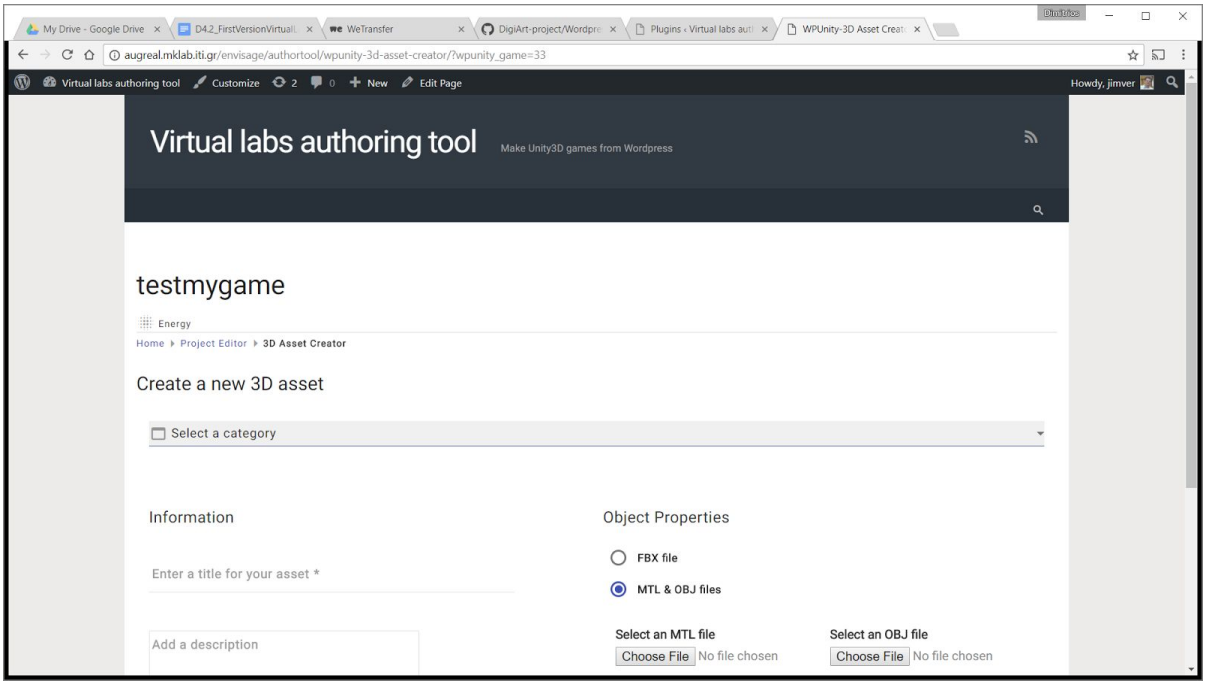

Figure 3.11a: Web page for creating a new Asset3D.

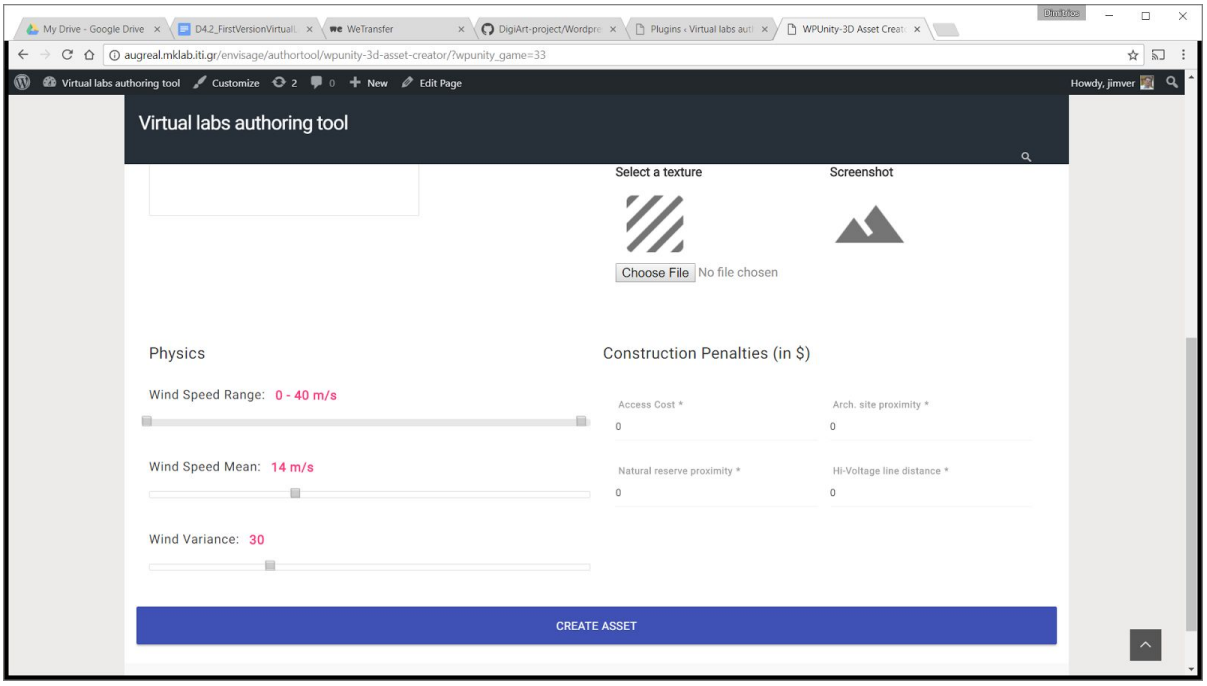

Figure 3.11b: Specific fields for the category “Terrain”.

The terrain has the following parameters that can be modified from the educator.

- Wind Speed
  - Mean : value in m/sec
  - Variance : value in $(m/sec)^2$
  - Range : Min - Max values in m/sec

Construction Penalties:

- Access cost penalty: value in $ (limits 0 to 5, default 0)
- Archaeological site proximity penalty: value in $ (limits 0 to 5, default 0)
- Natural reserve proximity penalty: value in $ (limits 0 to 5, default 0)
- Distance from High Voltage lines penalty: value in $ (limits 0 to 5, default 0)

[Action C] Insert a Decorator: A Decorator is a game object that can improve the immersiveness such as “Archaeological site”, “Power lines”, “Trees”, etc.. Their category, which should be selected when creating a new asset, is named as “Decorator”. Decorators can be dragged-n-dropped an arbitrary number of times in the scene as shown in Figure 3.12 for a tree. They do not have any fields.

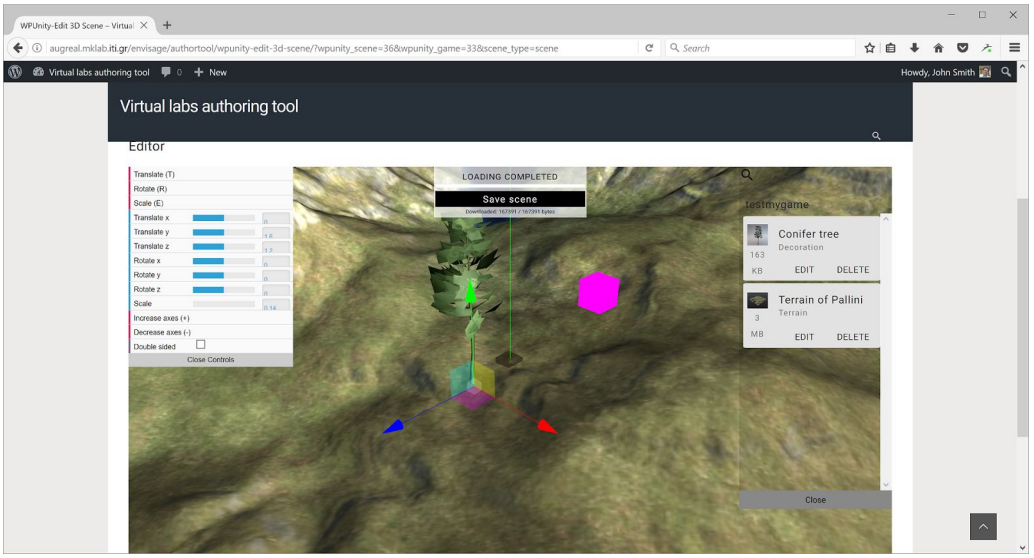

Figure 3.12: Example for inserting a tree, which is a decorator type of Assets3D.

[Action D] Insert a Consumer: A Consumer is a game object that consumes energy (e.g. a building). Several Consumers (block-of-flats, single houses, factories) will be available for drag-n-drop in the scene for multiple times. The total energy consumption is the sum of the consumption of all Consumers. A Consumer turns red if underpowered, blue if overpowered, and normal color if correctly powered. A screenshot is shown in Figure 3.13.

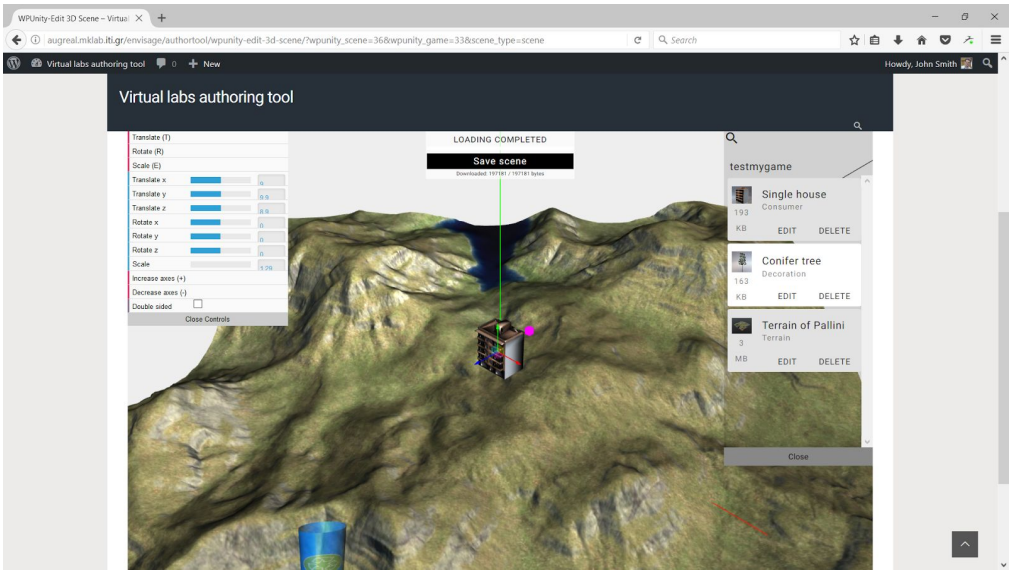

Figure 3.13: Inserting a building that consumes energy.

Consumers have several fields that define their power consumption such as:

- Energy Consumption:
  - Paid money per kWh: 3 values in $ for overpower, for correct power, and for

underpower
  - Overpower: Min: -5, Max: 5, Default: 0.5
  - Correct power: Min: -5, Max: 5, Default: 1
  - Under power: Min: -5, Max: 5, Default: 0
- Range: A 2D vector: Min: 0 MW, Max: 2 MW, Defaults: [0 MW , 2MW]
- Mean : Min 0 MW, Max: 2 MW, Default 0.1 MW
- Variance : Min 0.001 MW^2 , Max : 1 MW^2 , Default: 0.15 MW^2

[Action E] Insert a Producer: A Producer is a game object that generates energy (e.g. a Wind Turbine or a Solar Panel). Producers can be dragged-n-dropped several times in the game by the educator. When the game starts they do not appear but a marker is shown on the ground to indicate to the learner that in this place where a Producer can be built (Candidate position). A screenshot of the producers in the vlabs authoring tool is shown in Figure 3.14.

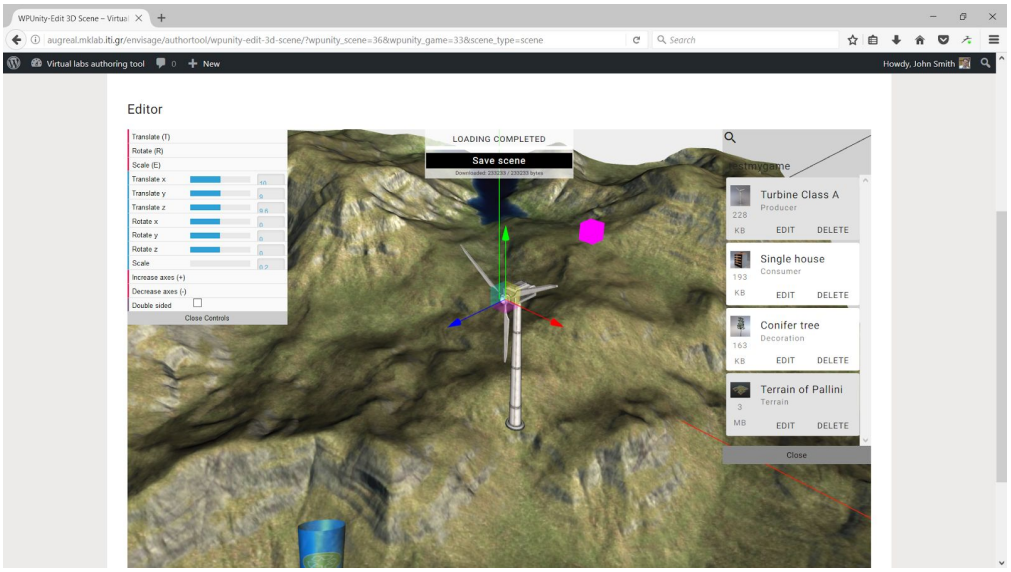

Figure 3.14: Screenshot for inserting producers in the vlabs authoring tool.

Each Producer has several fields such as

- Energy class - It is energy production curve with respect to the wind that it is written as pairs of values e.g. (0,0),(1,0)(2,0)(3,0)(4,0.5)(5,1) ... (25,6)(26,6)(27,6) where the first value in the pair denotes the air speed in m/sec and the second value denotes the power production in MW. These can be changed with the sliders shown in Figure 3.16.
- Size - The size of the turbine is fixed in meters (min : 3 m, max: 250m). Default 90m.
- Cost to buy - in $, Min $1, Max $10, Default $3.
- Damage coefficient probability - 0.001 to 0.02 which is the probability for the turbine to be damaged per second. Default 0.005.

- Damage repair cost - in $, Min $0.5, Max 5.

Figure 3.16: Power efficiency curve of the wind turbine. X axis is the Wind Speed, whereas y axis is the power production.

What the educator can not do in the first prototype of the vlabs authoring tool with respect to the requirements is

a) Define sub-areas inside the areas
b) Buy area extension
c) Reduce the efficiency of the Wind turbine (proximity to other turbines, parallel distribution)

These requirements will be investigated in the second phase prototype.

3.3 Learner actions allowed in the produced games

Here we provide a summary of the actions that learner can perform with respect to the learning objective.

[Action A] Select an area: to place the turbine among several choices where the pros and the cons are stated. An example is displayed in Figure 3.17.

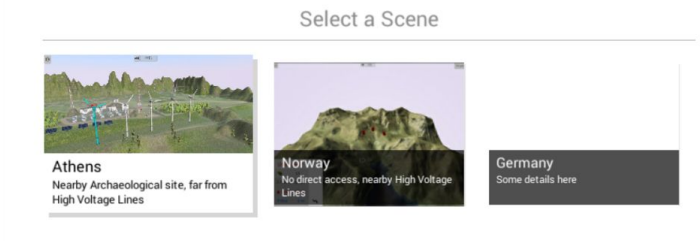

Figure 3.17: The learner can select an educational scene among several choices.

[Action B] Place turbines in candidate positions: A candidate position is shown with a marker as shown in Figure 3.18.

Figure 3.18: The marker denotes a candidate position for a turbine.

[Action C] Turn off a turbine - The learner can turn-off a turbine by clicking on it when the power generation greater than the consumption.

[Action D] Repair a turbine - The learner can repair a turbine if the turbine outputs smoke by clicking on it.

[Action E] Change simulation speed - The learner can change the simulation speed using the top-middle dropdown button.

Other navigation and visualization features are :

- The learner can orbit-zoom-pan around the scene to see the turbines from all sides.
- The learner can view money earned, current energy production-consumption, and current wind speed in the lower-left panel.

Game rules and rewarding

If the energy need and production are in equilibrium the learner earns some virtual coins. If the learner repairs a turbine, then he or she loses some virtual coins. The values are defined by the educator in the game generation process. Every 15 seconds, the status of the game is reported to the analytics server which are game state events consisting of

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

Section 4.1 explains the game state concept.

3.4 Hardware specifications

The generated games are web games which can be played through “Chrome”, “Firefox” and other web browsers. The hardware recommendations should be for middle-end devices. However, the games provide the option to deteriorate the detail level (from settings scene) in order for the game to run smoothly on low-end personal computers. 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: 1.5 GB free
d) Web browser: 64bit Chrome, Firefox, or Edge

Optimum specifications are:

e) CPU: i5 processor of 3rd generation or better
f) RAM: 2 GB free

4. Gathering data for shallow analytics

To track data for shallow analytics, GIO developed an SDK for Unity from scratch. To initialize the SDK and distinguish different labs, GIO provides privately an app key.

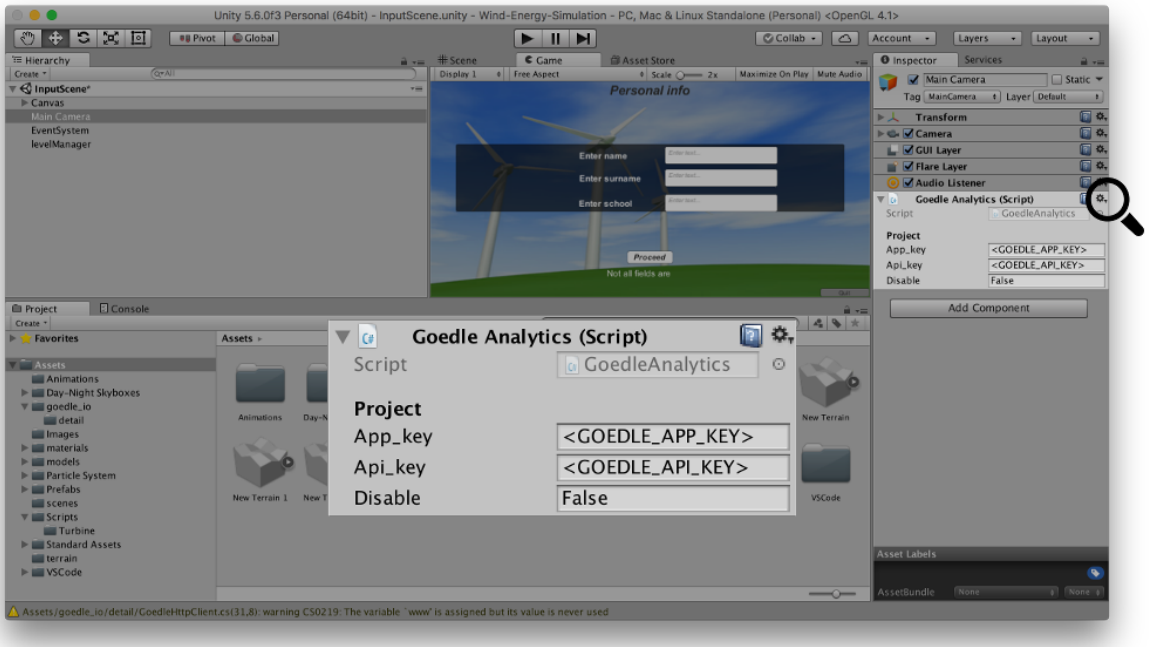

Figure 4.1 - Interface to initialize the Unity SDK

The SDK can be used with Unity for Windows, OSX, and for WebGL. Aside from the official Asset Store, the SDK is integrated as a plugin with an interface for tracking. In the GitHub repository of the ENVISAGE project, one can find the source code and a step by step manual[4] for integrating the SDK. The SDK provides, a method to track user actions and an additional method for tracking specific user traits. The tracking specification in deliverable D2.1, Section 4.1, was followed. For now, meta data transmitted with every launch of an Unity app are

- app version - current version of the virtual lab
- build number - current version of the GIO Unity SDK
- timezone - timezone offset of the user in milliseconds, e.g., 3600000 for UTC+1

Once a tracking point is defined, the tracking can be called with additional parameters. GIO created a new 'identify'-method which can be called with specific traits. As an example, the SDK can now collect a first name or a last name. For a grouping of learners into classes, GIO introduced an abstract group event. This event gets a special treatment in the GIO backend and adds a group id to the tracking. The following example shows how data is collected.

[4] https://github.com/Envisage-H2020/sdk_unity

- **event tracking**

```
track(state_update, 'wind_speed', 'warp');
```

- **group call**

```
track('group', 'GGS Turing');
```

- **identify**

```
identify('firstname', 'Kurt');
identify('lastname', 'Goedle');
```

4.1 Updated Tracking Concept

Together with other partners from the consortium, GIO created a tracking concept that respects pedagogical aspects, as well as ideas for special machine learning purposes. From this discussion, the notion of a game state was introduced. Here, state information is polled in an interval of 15 seconds and the data is then directly sent to the GIO backend. The game state contains current information about a learner which represents the progress over time. In the Wind Energy Lab for example, information about the power output or the wind speed value is included too. In the tracking method it is defined as '`state_update`'. Table 4.1 shows an extract of tracking points in the Wind Energy Lab. A full list is available online[5].

event	**event_id**	**event_value**	**description**
launch			Event when a learner starts the application
start.simulation			Event when a learner starts the simulation
resume.simulation			Event when a learner resumes the simulation after he paused the simulation via the play button
configure.simulation_speed		warp | fast | slow	Event when a learner adjusts the simulation speed (this is the duration of the simulation)
add.turbineDetailed	turbine_id	<turbine power>	Event when a learner inserts a turbine with id “turbine_id”
select.educationalScene	scene_name	<scene_title>	Event when a learner selects a certain educational scene among several, e.g. "Athens", "Koln", etc
select.scene	scene_name		Event when a learner selects a menu scene (Credits, Help, Settings, Login, Play)
press.uiButton	button_name		Event when a learner presses

[5] https://docs.google.com/spreadsheets/d/1zDWrM70NKz22iAV6nbWF30l2AVsyMGh8XKzYftTdnSw/edit#gid=0

			any UI buttons (e.g. Back, Main Menu, ...)
submit.score	<time_underpower> | <time_overpower> | <time_correctpower> | <energy_produced> | <money_earned> | <time_played>	<time_played_in seconds>	Events when a simulation finishes. The time played is added because we want to see if the users played until 24:00 or less.
identify	<first_name> | <last_name> |	<text_value>	Not changeable event, to submit the first/last name
group		<group_id>	Event when learners add themselves to a group, e.g a school class

Table 4.1 - Extract of tracking points in the 3d Wind Energy Lab

The development of the SDK and the definition of the tracking points is a continuous and ongoing process. The current status allows the project to have the same data tracking abilities within Unity as for the Google Tag Manager integration described in deliverable D2.1, Section 4.3.1.

4.2 Identifying Learners recurrently

One challenge during the development was the handling of user ids in order to track learners over more than one session. Once a learner starts the wind energy lab, the tracking begins but the learners are only recurrently identifiable if they enter any login information. Therefore, GIO implemented a new matching attribute. This helps to match tracking points which have happened before the login and after the login. For now, GIO only collects this matching information. The logic to merge these two tracking stages of learners is not yet implemented in the backend but will follow in the near future.

The process of identifying a learner recurrently has three steps

1. Creating a temporary id which only remains the same for one session
2. Creating a user id from the login information (based on first name, last name, school)
3. Sending a matching event (identify) to the backend with the temporary id, which we call anonymous id, and the user id

To create a temporary id for the matching which is unique for everyone, GIO makes use of

the GUID[6] creation function in C#. To identify a user recurrently, a user id is created based on the login information. Therefore, the first name, the last name and the school name is concatenated and hashed with MD5[7] and then transformed[8] to GUID format. This enables the identification of a learner independent of the session while respecting the privacy. The matching attribute and the user id based on login information are transmitted within the same tracking call with the identify method.

4.3 Non profit distribution of the SDK

Currently, there is no third party SDK in the Unity Asset Store that is able to make raw data accessible in a way we need it. Still, there is an Analytics[9] category in the Unity Asset Store where we planned to publish the ENVISAGE analytics SDK for more traction. However, one has to connect with Unity for the upload. GIO reached out to Unity but after a couple of mails back and forth with Unity, the official response was that they do not allow data/analytics SDKs in the Asset Store. Nevertheless, GIO is going to keep this possibilities in mind and might try to officially publish the SDK again in the future.

[6] https://msdn.microsoft.com/en-us/library/system.guid(v=vs.110).aspx
[7] https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5(v=vs.110).aspx
[8] This is done to have unified format of user ids.
[9] https://www.assetstore.unity3d.com/en/#!/search/page=1/sortby=popularity/query=category:132

5. Injecting analytics and its data into the authoring tool

There are different approaches that can be used to integrate the analytics component into the authoring tool. The most straightforward approach is that the authoring tool queries the GIO-servers for the data that is needed for a specific visualization. The response should be a JSON-object that contains all necessary information to draw the corresponding diagrams and charts.

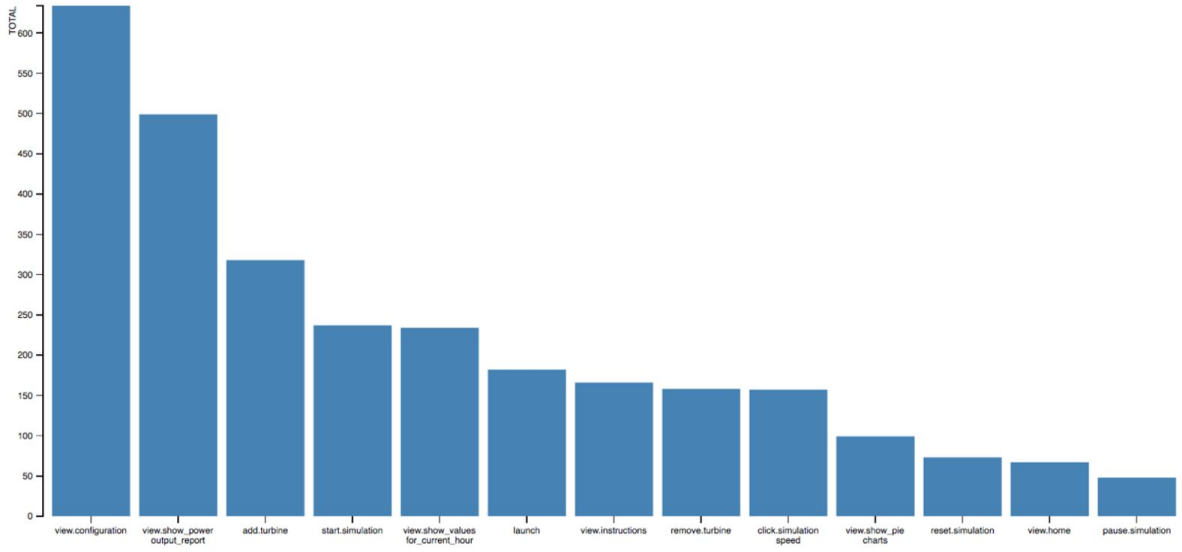

Figure 5.1: Each bar shows the absolute frequency of a particular event in a virtual lab. For example, the launch of a lab. To visualize this information, the frontend component requires at least the names of the different events and their occurrences.

This data can either be calculated directly on the GIO-servers as needed (see Section 5.1) or the GIO-servers return pre-calculated data that has been created previously in a batch fashion — potentially even created on other servers (see Section 5.2). And lastly, there is also the option that the authoring tool queries a different third-party service that returns the required data (see Section 5.3). In this case, the third-party will first query the GIO-API for the raw data, then do the calculations and return the data for the visualization to the authoring tool. We will also use the latter option to make the development of different analytics more flexible and less demanding on the deployment of the live infrastructure. For more details, please see Figure 4.3.

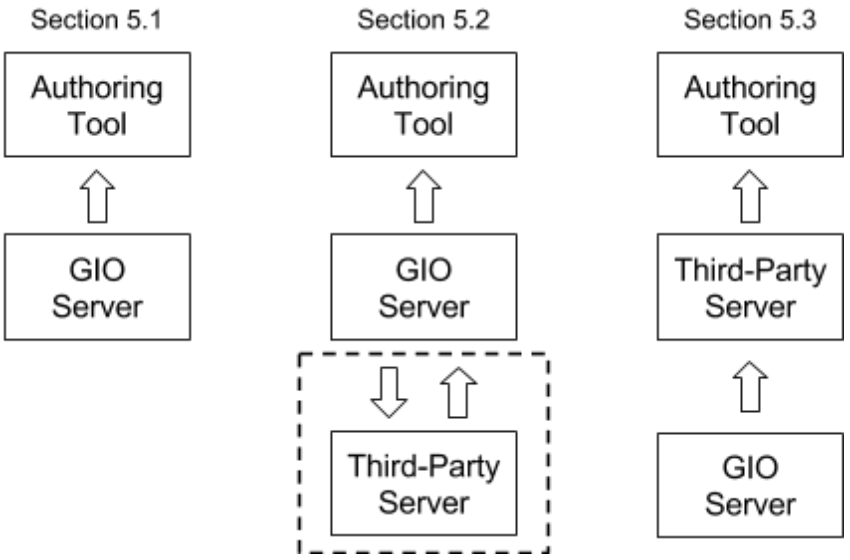

Figure 4.3: As described above, the data can be raw and analytics data can be accessed in different ways. For all three approaches, the next sections will give more details.

5.1 Direct Online Data Access

Let us assume that we want to show a bar chart in the authoring tool as shown in the figure above and suggested in deliverable D2.3, Section 5.1. The endpoint will be:

```
curl
      -H 'content-type: application/json'
      -H 'X-goedle-app-key: <LAB_ID>'
      -H 'X-goedle-master-key: <MASTER_KEY>'
      https://api.goedle.io/apps/<LAB_ID>/bar_chart/events
```

The response in JSON of such an endpoint is as follows:

```
{
      'data': [
            { 'event': 'launch', 'count': 170},
            …
            { 'event': 'pause.simulation', 'count': 40}
      ]
}
```

However, without additional parameters, the response will most likely be too general as it contains data from all learners in the database across all schools and classes. Therefore, we add query parameters such as '`last_active_gte`' and '`last_active_lt`'. Both parameters allow us to reduce the number of learners to only those who have been active in a certain period of time. In many cases, we also want to focus on learners in a specific '`country`' or in a particular '`group`' (e.g., a class in a school). We will add such parameters as required so that the desired segment can be displayed. This endpoint

requires the authoring tool to allow the learners to pick these parameters accordingly. Additionally, the authoring tool needs a safe mechanism of storing the API master key within the tool which is referred to as '<MASTER_KEY>' in the example above.

After receiving a request, the learners database is queried for the learners matching the parameters and afterwards the data necessary for the bar chart will be calculated. For shallow analytics, the calculations will be according to deliverable D2.2 and D2.3. An example response is the JSON response depicted above.

5.2 Cached Data Access

This setting is more challenging because it requires that the data for the bar chart to exist already pre-calculated in the GIO database. Therefore, the parameters need to be limited in a certain way so that a manageable amount of data can be pre-calculated every a certain time interval, e.g., 24 hours. This is useful in different cases. In first place, some of the deep analytics will be computationally challenging so that an online calculation is not feasible. Pre-calculating the data ensures a smooth user experience. Additionally, the computation is not happening on the same server that return the data for visualization as different hardware requirements are present.

In this setting, the service calculating the data would either query the GIO-API for the raw data as described in deliverable D2.1, Section 7.1, or use the aggregated data access (as described in D2.1, Section 7.2). The aggregated data access looks very similar to the curl-request above. However, it returns user level data instead of aggregated data necessary for the bar char. For example, one could query the API for all learner that have been active on July 21, 2017 and later but before July 28, 2017:

```
curl
      -H 'content-type: application/json'
      -H 'X-goedle-app-key: <LAB_ID>'
     -H 'X-goedle-master-key: <MASTER_KEY>'
     https://api.goedle.io/apps/<LAB_ID>/users/?\
                                        last_active_gte=2017-07-21&
                                        last_active_lt=2017-07-28
```

After the raw data or learner data has been received and the calculations for the visualizations have been done as well, the data needs to be sent back to the GIO servers. For this purpose, another endpoint needs to be added. It is most reasonable to add a PUT-endpoint to the API that works the same way as the GET-endpoint above. E.g.,

```
curl
      -X PUT -d @data.json
      -H 'content-type: application/json'
      -H 'X-goedle-app-key: <LAB_ID>'
      -H 'X-goedle-master-key: <MASTER_KEY>'
      https://api.goedle.io/apps/<LAB_ID>/chord_diagram
```

The call assumes that 'data.json' contains all data necessary to show the visualization in

JSON, i.e., it should look similar to the JSON response above. Of course, this endpoint needs to be equipped with a functionality that handles parameters in a corresponding way to the GET-endpoint. Depending on the possible distributions of the parameters, the data has to be saved in the GIO database based on these parameters so that it can be retrieved accordingly.

5.3 Data Access via Third-Party

In some cases, it is easier to return the calculated data from other third-party servers. One particular example is the development phase where code is still being written and the features are not production ready yet. In this setting the authoring tool queries a temporary server that returns the information in JSON the same way as depicted above. In such a scenario the third-party server queries the GIO-API to obtain the raw data. It then calculates the necessary information to show in the visualization.

6. Visualization of Deep and Shallow Analytics

In order to analyze, model, and eventually visualize, learner behavior through shallow and deep analytics, we are implementing three separate technical solutions. The three solutions are interdependent and form a logical and operational stack, that can be deployed as one, and hence all three of them are described in this section. These relations are outlined in Figure 6.1.

The user-facing layer of the stack is the visualization service. This service displays observed and inferred information about the learners of a given virtual lab and informs the author of behaviors in the currently implemented version of the virtual lab as well as potential outcomes of changes to the virtual labs. The visualization service in turn receives its information from the shallow analytics service.

The shallow analytics service performs data aggregation, abstraction, and storage tasks, taking raw measurements gathered from the virtual labs and turning these into metrics that can be analyzed by the user. The shallow analytics service communicates with the deep analytics service that receives data, models the data, and transmits this back to the shallow analytics service which in turn stores the results and provides these to consumers i.e. the visualization service. Further information for each 3rd party library will be provided in Section 6.2.

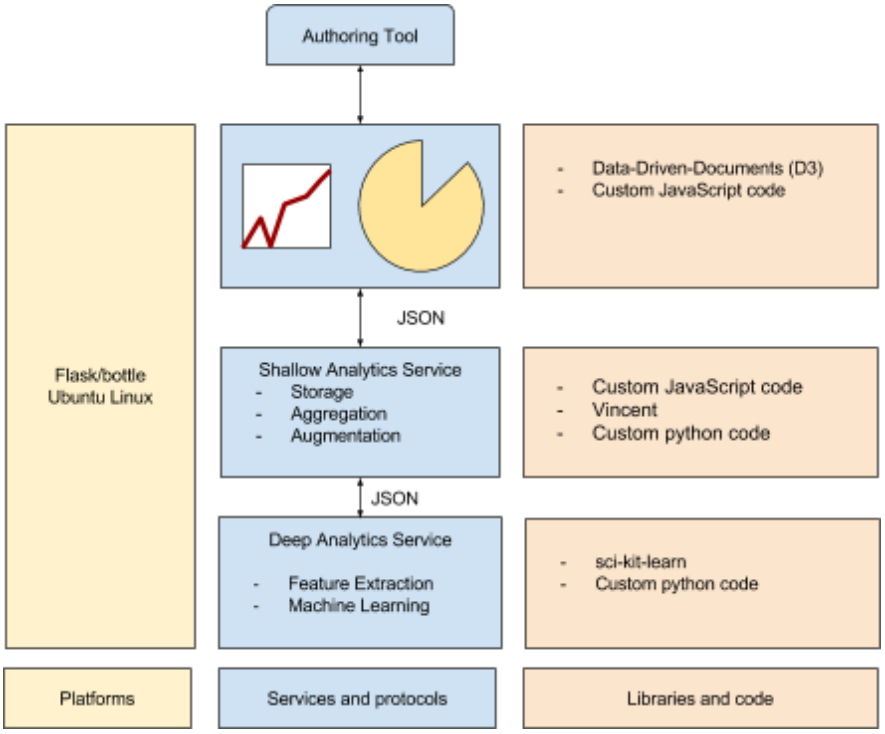

Figure 6.1: Overview of the visualization, shallow analytics, and deep analytics stack. Blue: services and their core functions; Yellow: necessary platforms; and Orange: used libraries and custom code.

6.1 Measured Data

The raw data points as they were described in Section 4, Table 4.1, are converted into a number of metrics by the shallow analytics service. Depending on the use case these may be converted on-line in the user's browser or off-line and stored with the data set. All events are grouped by user and session and turned into list data structures with one list per user per session. Individual sessions are demarcated using the "launch" event described in Table 4.1. If no further events are received from the same user for an extensive amount of time, the session is considered concluded.

With each list representing a series of events for each session for each learner, it now becomes possible to leverage the other event types to evaluate the learner's travel path through the application, as well as calculating the five key metrics of interest, defined in previous deliverables. Further, combinations of these features constructed from expert knowledge, as well as the raw event data, may be transferred to the deep analytics service for treatment.

6.2 Software libraries used

In Table 6.1, the software libraries used in the implementation and some installation instructions for assisting integration procedure are provided.

Software	**Use**	**Instructions**
Ubuntu Linux 16.04 LTS	Any operating system capable of providing internet access and executing Python code may in principle be used to host the visualization, shallow, and deep analytics services. For ENVISAGE, we are currently using Ubuntu 16.04 LTS.	Free to download from: https://www.ubuntu.com/download
Anaconda 4.4.0	A wrap of Python and supporting libraries. Anaconda is an open source data science platform that includes the most common data science and machine learning frameworks.	Free to download from: https://www.continuum.io/downloads

Flask/bottle	Flask/bottle are lightweight micro frameworks for generating and serving web pages using Python. Either may be more appropriate depending on the particular needs of an installation. For ENVISAGE we use Flask for development and bottle for production	Both Flask and bottle may be installed using the Python package manager pip, using either of the following commands:
		`pip install Flask`
		or
		`pip install bottle`

D3	Data-Driven-Documents (D3.js) is a JavaScript library for manipulating data and generating visualizations. D3 is	D3 is included with the content served by either the Flask or the bottle microframeworks to the

	responsible for generating the content rendered in the user's browser.	user's browser. As such it is included in the code found in the ENVISAGE repositories and no further installation is necessary.
Custom JavaScript	ENVISAGE uses custom JavaScript code for some data preparation and visualization in shallow analytics.	May be obtained from the ENVISAGE repositories.
Vincent	Vincent is a Python to Vega translator. It allows for generating D3 visualization specifications using a Python codebase. As such, it is the 'glue' between the backend data treatment and the D3 visualizations.	Vincent may be installed using the Python package manager pip, with the following command: `pip install vincent`
Vega	Vega is a visualization grammar that can be used to provide specifications for D3 visualizations. For ENVISAGE, Vega is the protocol used to communicate between the modeling and manipulation Python code and the D3 visualization code. Vega is structured in JSON, a standard format for exchanging data via the internet.	As Vega is used as a communication protocol between the Python code of the project and the D3 JavaScript visualizations, no installation is necessary.
scikit-learn	scikit-learn is a machine learning library for Python that contains many common algorithms, some of which are leveraged in ENVISAGE.	scikit-learn is installed as part of Anaconda (see above) but can be installed on its own via pip, using the following command: `pip install scikit-learn[alldeps]`
pandas	Pandas is a scientific data manipulation library for convenient preparation and treatment of data.	pandas is installed as part of Anaconda (see above) but can be installed on its own via pip, using the following command: `pip install pandas`
Custom Python code	ENVISAGE uses custom Python code to take care of data exchange, manipulation, aggregation, shaping, and presentation.	May be obtained from the ENVISAGE repositories.

Table 6.1. Platforms, frameworks, and libraries used for visualization, shallow, and deep analytics.

Running the visualization, shallow and deep analytics stack

For running the software stack described in this section in its current form, all the software in Table 6.1 must be installed on the server.

The visualization, shallow, and deep analytics stack must be checked out from the ENVISAGE repository.

If using Flask, the following command should be executed at the command prompt to start the service.

./envisage_analytics.py --port [PORT] --app-key [APP-KEY] --master-key [MASTER-KEY]

The analytics web pages may then be accessed from the server, using the port specified in the place of [PORT], e.g. port 80, port 443 or port 5000.

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.

Appendix I: Assets files

Here we provide information about each asset file and how it should be imported into the game project. Common parameters across assets are the fileFormatVersion, 1 for Mac vs 2 for Windows, guid (Global User Interface Identifier) which is a string of 32 characters that identifies uniquely the folder asset, and timeCreated is the unix timestamp in secs from 1/1/1970 UTC.

1. **obj and obj.meta**
    a. obj is a text file format for storing object meshes. Limitations is that the 'o' object tags should be replaced with 'g' group tags. Upper limit for the size is 256 MB as well for all files.
    b. obj.meta: contains the guid for the certain obj. It should be generated from our plugin according to the following pattern where fileFormatVersion, guid, and timeCreated should be replaced accordingly.

```
fileFormatVersion: 2
guid: 64d690459f3cb8a4ca08c28f4ac524bd
timeCreated: 1499687746
licenseType: Free
ModelImporter:
  serializedVersion: 19
  fileIDToRecycleName:
    100000: building1
    100002: //RootNode
    400000: building1
    400002: //RootNode
    2300000: building1
    3300000: building1
    4300000: building1
  materials:
    importMaterials: 1
    materialName: 0
    materialSearch: 1
  animations:
    legacyGenerateAnimations: 4
    bakeSimulation: 0
    resampleCurves: 1
    optimizeGameObjects: 0
    motionNodeName:
    rigImportErrors:
    rigImportWarnings:
    animationImportErrors:
    animationImportWarnings:
    animationRetargetingWarnings:
    animationDoRetargetingWarnings: 0
    animationCompression: 1
    animationRotationError: 0.5
    animationPositionError: 0.5
    animationScaleError: 0.5
    animationWrapMode: 0
    extraExposedTransformPaths: []
    clipAnimations: []
    isReadable: 1
  meshes:
    lODScreenPercentages: []
    globalScale: 1
    meshCompression: 0
    addColliders: 0
    importBlendShapes: 1
    swapUVChannels: 0
    generateSecondaryUV: 0
    useFileUnits: 1
    optimizeMeshForGPU: 1
    keepQuads: 0
    weldVertices: 1
    secondaryUVAngleDistortion: 8
    secondaryUVAreaDistortion: 15.000001
    secondaryUVHardAngle: 88
    secondaryUVPackMargin: 4
    useFileScale: 1
  tangentSpace:
    normalSmoothAngle: 60
    normalImportMode: 0
    tangentImportMode: 3
  importAnimation: 1
  copyAvatar: 0
  humanDescription:
    serializedVersion: 2
    human: []
    skeleton: []
    armTwist: 0.5
    foreArmTwist: 0.5
    upperLegTwist: 0.5
    legTwist: 0.5
    armStretch: 0.05
    legStretch: 0.05
    feetSpacing: 0
    rootMotionBoneName:
    rootMotionBoneRotation: {x: 0, y: 0, z: 0, w: 1}
    hasTranslationDoF: 0
    hasExtraRoot: 0
    skeletonHasParents: 1
  lastHumanDescriptionAvatarSource: {instanceID: 0}
  animationType: 0
  humanoidOversampling: 1
  additionalBone: 0
  userData:
```

	assetBundleName: assetBundleVariant:

2. **mtl**
    a. mtl contains the material description for the certain obj.

    The mtl is automatically imported and transformed into mat with the import script. mat files are described below.

3. **mat and mat.meta**
    a. mat contains the material description for Unity3D.
    b. mat.meta contains the guid for the mat file.

    Both mat and mat.meta are automatically generated with the import script.

4. **Textures jpg and jpg.meta**
    a. A texture jpg is an image for the texture of the obj. The size of jpg should be power of 2 and width should be equal to height.
    b. A texture .jpg.meta is a text file containing the guid for the texture jpg.

    Both texture jpg and texture jpg.meta are automatically generated with the import script

5. **Sprites jpg and jpg.meta**
    a. The sprite jpg is a plain jpg with no limitations in width and height
    b. The sprite .jpg.meta should follow a certain pattern that denotes that the image is sprite and its guid. The pattern is as follows where fileFormatVersion and guid should be replaced in each sprite.

fileFormatVersion: 2
guid: c09f5d3bd5a1ac34cba9de90fcb13da1
timeCreated: 1500039473
licenseType: Free
TextureImporter:
  fileIDToRecycleName: {}
  serializedVersion: 4
  mipmaps:
  mipMapMode: 0
  enableMipMap: 1
  sRGBTexture: 1
  linearTexture: 0
  fadeOut: 0
  borderMipMap: 0
  mipMapFadeDistanceStart: 1
  mipMapFadeDistanceEnd: 3
  bumpmap:
  convertToNormalMap: 0
  externalNormalMap: 0
  heightScale: 0.25
  normalMapFilter: 0
  isReadable: 0
	textureType: 8
	  textureShape: 1
	  maxTextureSizeSet: 0
	  compressionQualitySet: 0
	  textureFormatSet: 0
	  platformSettings:
	  - buildTarget: DefaultTexturePlatform
	  maxTextureSize: 2048
	  textureFormat: -1
	  textureCompression: 1
	  compressionQuality: 50
	  crunchedCompression: 0
	  allowsAlphaSplitting: 0
	  overridden: 0
	  - buildTarget: Standalone
	  maxTextureSize: 2048
	  textureFormat: -1
	  textureCompression: 1
	  compressionQuality: 50
	  crunchedCompression: 0
	  allowsAlphaSplitting: 0
	  overridden: 0

```
  grayScaleToAlpha: 0
  generateCubemap: 6
  cubemapConvolution: 0
  seamlessCubemap: 0
  textureFormat: 1
  maxTextureSize: 2048
  textureSettings:
    filterMode: -1
    aniso: 16
    mipBias: -1
    wrapMode: -1
  nPOTScale: 0
  lightmap: 0
  compressionQuality: 50
  spriteMode: 1
  spriteExtrude: 1
  spriteMeshType: 1
  alignment: 0
  spritePivot: {x: 0.5, y: 0.5}
  spriteBorder: {x: 0, y: 0, z: 0, w: 0}
  spritePixelsToUnits: 100
  alphaUsage: 1
  alphaIsTransparency: 0
  spriteTessellationDetail: -1
```

```
  - buildTarget: Android
    maxTextureSize: 2048
    textureFormat: -1
    textureCompression: 1
    compressionQuality: 50
    crunchedCompression: 0
    allowsAlphaSplitting: 0
    overridden: 0
  - buildTarget: WebGL
    maxTextureSize: 2048
    textureFormat: -1
    textureCompression: 1
    compressionQuality: 50
    crunchedCompression: 0
    allowsAlphaSplitting: 0
    overridden: 0
  spriteSheet:
    serializedVersion: 2
    sprites: []
    outline: []
  spritePackingTag:
  userData:
  assetBundleName:
  assetBundleVariant:
```

Appendix II: Compiling commands for desktop binaries.

Here we present information for compiling into desktop binaries.

In Windows server to Windows Binary: Place the following into a .bat file and execute it.

```
set mypath=%cd%

@echo %mypath%

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

In Linux server (Ubuntu 16) to Windows Binary: Place the following into a .sh file and execute it.

```
#/bin/bash

projectPath=`pwd`

xvfb-run --auto-servernum --server-args='-screen 0 1024x768x24:32' /opt/Unity/Editor/Unity -batchmode -nographics -logfile stdout.log -force-opengl -quit -projectPath ${projectPath} -buildWindowsPlayer "build/mygame.exe"
```

Defining other outputs

To Mac binary instead of Windows binary: Replace **-buildWindowsPlayer** with **-buildOSXUniversalPlayer,** and **.exe** with **.app**

To Linux binary instead of Windows binary: Replace **-buildWindowsPlayer** with **-buildLinuxUniversalPlayer** , and remove the **mygame.exe**