

Producing a Level Editor For 3D Environments

CS3D661

Alex Sampson

15015483

Carl Jones

Statement of Originality

CS3D661 Individual Project

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerized sources have been fully referenced, and I fully realize the consequences of plagiarizing any of these sources.

Student Name (Printed): Alex Sampson

Student Signature: A. Sampson

Registered Course of Study: BSc Computer Games Development

Date of Signing: 27/4/20

# Abstract

This project’s focus is the development of a tool that is used to create 3D levels for games by exploring what elements make for a good level editor, what features are designed in specific ways to better their use case, and the implementation of said features to fit the overall goal of the project.

The outcome of the project is to have a tool that can be used by anyone to create 3D environments with the toolset provided within the project.

The project is created using Unity Game Engine due to its expansive toolset as well as personal prior knowledge to the software.

Contents

[Statement of Originality 2](#_Toc38909057)

[Abstract 3](#_Toc38909058)

[Project Summary 5](#_Toc38909059)

[Project Aim 5](#_Toc38909060)

[Project Objectives 5](#_Toc38909061)

[Pre-Existing Level Editors 6](#_Toc38909062)

[Professional Tools 6](#_Toc38909063)

[In-Engine Tools 6](#_Toc38909064)

[Methods of Production 8](#_Toc38909065)

[OpenGL 8](#_Toc38909066)

[Unity Game Engine 8](#_Toc38909067)

[Loading 3D Models 8](#_Toc38909068)

[Selecting Objects 9](#_Toc38909069)

[Project Planning 10](#_Toc38909070)

[Project Development 11](#_Toc38909071)

[User Camera 11](#_Toc38909072)

[Object Interaction 11](#_Toc38909073)

[Object Creation 11](#_Toc38909074)

[Coordinate Snapping 12](#_Toc38909075)

[Mouse Interaction 13](#_Toc38909076)

[Texture Loading 14](#_Toc38909077)

[Exporting Files 17](#_Toc38909078)

[Input 19](#_Toc38909079)

[Keyboard and Mouse Inputs 19](#_Toc38909080)

[User Interface 20](#_Toc38909081)

[Translation Setting Menu 20](#_Toc38909082)

[Example Scene One 21](#_Toc38909083)

[Example Scene Two 22](#_Toc38909084)

[Example Export File 22](#_Toc38909085)

[Conclusion 23](#_Toc38909086)

[Unity Engine 23](#_Toc38909087)

[Transforming 23](#_Toc38909088)

[User Interface 23](#_Toc38909089)

[Texturing 23](#_Toc38909090)

[Import/Export 24](#_Toc38909091)

[References 25](#_Toc38909092)

[Project Repository 25](#_Toc38909093)

# Project Summary

## Project Aim

The aim of this project is to create a functioning 3D level editor tool that includes all of the required features able to produce and export a 3D level to a file that is then usable to load the scene.

## Project Objectives

The first objective for the project is to create an environment where 3D models/files can be loaded into. This 3D model will be imported into a local file to ensure the project has no external dependencies, so that it can still function correctly even if the external files are moved or deleted. The Level Editor will have a UI element to display all the available models which will load them from the local files, with a preview version on display. To save on performance, the local version could be either a low-resolution version of the model or a single image file, both options being generated when the model is imported.

The next objective is to be able to access the model file previously imported and position it into the scene. Objects that are placed into the scene must then be able to be selected to enable transformations to them individually. The selection process could be through a UI element which lists every active object in the scene for the user to select from, or directly with the mouse cursor using rays cast in a 3D space from the camera, through the cursors position relative to the world.

The objects, when selected must be allowed to be interacted with, including having the object be able to be translated within the environment by having features to change the objects scale, position and rotation. A secondary objective with the interaction may be to allow the objects colours and textures to be changed, as well as to add any dynamic features such as a constantly moving object, for example a train passing by the scene continuously.

The final primary objective is to export the completed environment to a single file that can be used to load the environment as a standalone file, as opposed to it being confined to the editor.

A secondary objective for the project file management is to implement a save/load feature so that the in-progress levels could be saved inside the editor to be loaded later. This feature isn’t required for the functionality of the level editor, however the ability to save and load the project is a feature that will improve the overall use of the program.

# Pre-Existing Level Editors

## Professional Tools

A picture containing text

Description automatically generatedSource SDK

The Source SDK is a collection of tools used for authoring content for the Source engine. Valve Hammer Editor, included in the Source SDK, is a 3D level creation tool for use with the Source and GoldSource Engines for titles such as Half-Life, Counter Strike and Team Fortress (Valve Corporation, 2019). Hammer is a professional tool used by the official developers as well as publicly available through their Steam platform.

Figure - Hammer Editor

GtkRadiant

Radiant Level Design Tools is a similar tool to Hammer in that it is used to create 3D environments for games powered by the id Tech engines such as Quake, Wolfenstein and Doom (GtkRadiant Team, 2016). GtkRadiant currently exists as an open-source level editor but is also the base of the level editor currently used by Call of Duty developers Treyarch and Infinity Ward as their engine is a modified variant of the id engine.

The tool contains the basic functions of a level editor such as placing objects, lights and entities down to create environments, however it also includes a more advanced toolset for creating set pieces such as triggers, which when supplied with proper key-value pairs, can create a dynamic environment with moving doors, hazards and other interactable objects.

In summary, the two tools referenced above are very refined in their design for a very specific use case, and contain a wide variety of features which allows the users to create intricate, professional levels however this comes at the cost of having a rather high learning curve, as using these features in conjunction with each other can be a burden without the proper experience.

## In-Engine Tools

Halo Forge

Halo’s Forge is an in-engine game mode where players can create and translate objects within the Halo Universe to create variants of pre-existing levels and game-modes. These tools are a lot more rudimentary than the professional tools previously mentioned as they are designed with a focus on user experience and ease of use. Halo’s forge is a mode in which the player can add and remove props from existing levels and place their own down in any configuration within the limitations of the map, as well as the prop count limit that exists for performance and design reasons. The player can switch between a grounded player perspective and a free moving monitor, which allows for instant creation and testing without having to export to an external tool.

Using the editors listed above, both professional and in-engine, helps outline which features are key to this project and how to design the level editor for the specified purpose. To keep the development focused and within schedule, the project will lean more towards the in-engine simplicity to ensure the planned features can be developed fully, with any excess time being used for the more in-depth features more typically found in the professional tools.

# Methods of Production

As the brief outlined above does not require the use of a specific language or technology to be used, the most suitable method must be determined for this project.

## OpenGL

OpenGL is an API (Application Programming Interface) that could be used in this project to handle the graphics and rendering of the level editor’s 3D environments. OpenGL can be integrated into programs written in many different languages, with C and C++ generally being used as OpenGL itself being written in C (OpenGL, 2019).

If the project were to be written in C++ from scratch, OpenGL would be a great tool to use for the rendering, however, this may take a considerable amount of time to fully utilise, leaving less time to develop the key features.

## Unity Game Engine

Unity Game Engine is a multi-platform engine used for a wide variety of productions. Unity contains a vast array of tools that can be used to create games, animations, applications and more. Unity has a wide variety of libraries and components available from the base editor that can be used during the production of this project to easily implement features that otherwise would take longer independently due to the nature of their complexity.

## Loading 3D Models

The first stage of producing the editor is to determine a suitable method of loading the 3D models from a file to be used in the editor. One of the biggest problems to overcome with this aspect is that 3D objects can come in many different file formats such as .OBJ, .FBX and .3DS. All of these file formats will need to either be supported or be stated that they aren’t available in the project as they will all require code that will handle them properly to allow them to be imported into the editor.

The most basic method for handling file formats would be to only include a parser for a single file type, such as a Wavefront OBJ file, and require the user to convert the model outside of the editor which will save time on development for the project however will limit the user functionality as well as the user experience.

Using an existing library such as Assimp, the Open Asset Import Library, would allow any file format supported by the library to be imported into the editor with relative ease, as the library parses the formats and converts them in a uniform manner so that all file formats would appear the same once in the code to be used by the program (Assimp, 2018). The use of Assimp would improve the functionality of the project but has the potential to add an overhead on the time it takes to produce the importer as it would include learning a new library I have not previously used. This could also cause issues with the code if not implemented correctly due to inexperience.

## Selecting Objects

When the objects are present in the scene, the editor must have a method of selecting the objects to be transformed. An ideal way to select objects would be to use the mouse and click on the objects in the scene. To achieve this, the program can cast a ray from the position of camera, through the mouse in world position and then select an object if the ray collides with the object. Adding ray casting to the program may increase the development time depending on how it is implemented.

An easier method of selection is to have a UI element that contains a dropdown menu with all of the available objects in the scene to be selected. This method would be a very simple way of implementing selections, however it would hamper the user experience.

# Project Planning

The planning of the project is broken down into two sections, the primary and secondary objectives, with each containing a set of objectives needed. The primary objectives outline the key features needed to produce a simple level editor that hits all the objectives for the project, however, will be lacking in features to create anything more than a basic scene, with very little support for a good user experience. Each section of the primary objectives shouldn’t ideally take more than a week to produce, from conceptual work to implementation, with the exception of the final objective, exporting the finished level, as that will require extra time for research and design.

The Secondary objectives are designed to improve the overall quality of the Level Editor, as well as the user experience of the product by implementing features that are designed to ease the workflow within the tool.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Figure - Estimated Time Taken Per Objective

Figure - Time Management Gantt Chart

# Project Development

The first stage of project development was to set up the initial environment for the project, as referenced in Figure 3. As the decision was made previously to develop the tools using Unity, the process of initialising the project was simplified, allowing for more development time within the project.

## User Camera

The first feature to implement into the project is the camera so that the user is able to move around the environment. The camera requires movement along the X and Z axis relative to its local rotation. It is vital that the local rotation is considered to allow for forward/backward and left/right movement. The third axis of movement, along the y axis, won’t take into account any local rotation and instead will simply move the camera up and down according to world coordinates.

The camera will also need to be able to rotate to allow the user to look around the 3D environment from different angles. The rotation can be handled in different ways, based on the function and position required. The simplest method of rotation is to parse a mouse or joysticks input to rotate. For accuracy and convenience, preset positions will be included to align with each axis to allow for more accuracy when translating objects.

## Object Interaction

A screenshot of a computer

Description automatically generatedObject interaction can be regarded as the main functionality of the project, including object selection and manipulation. When starting the development of the object interaction, a simple test object was created to ensure features, when implemented, function correctly. The object interaction uses the mouse cursor for most of the selection and uses Unity’s own input functions to handle all of the mouse button pressing as well as mouse dragging.

Figure - A simple test object

The user object contains a state for the transformations which will be controllable by the user. The user object will simply use a switch statement to check the state to determine the function to perform.

## Object Creation

The next area to focus development on is to add new objects to the scene, as the current testing environment only contains a rudimentary button to spawn a specific object, in particular the cube as seen in Figure 4. The creation system will require features for the user to be able to set which object will be added, as well as allowing parameters to be set, such as scale or texturing.

For the initial version of the implementation, the UI functionality will be disconnected from the main scene, with functions in place to spawn the object a set position in front of the camera. This method will be the easiest to implement, however with the simple implementation comes a lack of functionality and style. Future design tweaks to this system will be to implement drag and drop functionality to the UI so that the objects can be selected and dragged into place with a single action.

## Coordinate Snapping

To ensure the maps objects are placed in an orderly manner and able to fit together, objects positions, rotations and scale are rounded to the nearest integer. This allows objects to be easily positioned correctly without objects clipping into each other or the user having any misalignment issues.

To implement coordinate snapping into the existing translation methods a simple function was created to return a rounded vector when called, with a parameter of how precise the rounding is. This function is used between the calculations of translations, and before the application, to provide a simple method for the grid-like system.

A screenshot of a cell phone

Description automatically generated

Figure - Vector rounding

A screenshot of a cell phone

Description automatically generated

Figure - Applying rounded position

## Mouse Interaction

The main way the user will interact with objects is by using the mouse to drag objects into place, scale them to the desired size and apply rotation.

A picture containing screenshot, sitting, screen, television

Description automatically generated

Figure - On Mouse Down

When the mouse is first pressed down, three variables are set. The mzCoord variable is set to the selected object’s z coordinate, converted to screen coordinates. mOffset is set to a vector from the selected objects position to the mouse’s current world position. Finally, the currentScale is simply the selected objects current scale.

A screenshot of a cell phone

Description automatically generated

Figure - On Mouse Drag

While the mouse is being dragged, the state determines which translation is applied. The simple state switch expression means that only one form of translation can be applied at any one time as well as making it very easy to structure the code as well as modify if needed. The OnMouseDrag function is called every time the mouse position updates, resulting in an instant, accurate update to the object.

A screenshot of a cell phone

Description automatically generated

Figure - Position Drag

The implementation of the position dragging feature was rather simple, the current mouse position is added to initial offset as this determines how far the mouse has moved within the drag. Before the implementation of the vector rounding, this was the only application to the transform. As the rounding was added, this is calculated before being applied. As the mouse world position is calculated and used as opposed to the screen position, the translation takes into consideration the camera’s position allowing the object to be dragged along every axis relative to the camera.

A close up of a screen

Description automatically generated

Figure - Rotation Drag

The rotation drag function works in a similar fashion to the position dragging. The rotX and rotY variables are calculated by getting the distance from the mouse to the objects position and dividing it by the rotationSpeedMod variable. The object is then rotated using these two floats, in world space. Finally, the rounded rotation is calculated using the Vector3 round function, then applying the vector to the object.

A screenshot of a cell phone

Description automatically generated

Figure - Scale Drag

The scale dragging, similar to the other methods, works by calculating the difference in position of the mouse and object and applying a modifier. As scale cannot go into negative values, the value is clamped. The scale is added to the current scale of the objects and then the total is rounded. Finally, if the value is equal to zero, it is set to a vector3 one before applying the scale to the object.

## Texture Loading

Applying a custom texture is a key feature that needed to be implemented to be able to provide details to the environments created as opposed to simply using base colours or development textures. This system must be able to allow the user to choose a file from their drive and apply it to an object. As the environments need to be exported and imported, the texture file must be able to be saved locally to be imported. To ensure the correct texture is loaded, a randomly generated key will be given to a texture when initially loaded to match the local file to If a texture file is missing from the local files, a missing texture placeholder will be loaded in its place.

A screen shot of a computer

Description automatically generated

Figure - Texture Selection Button

The texture application process is made up of four functions to achieve the required target. The first is to provide the user with a file explorer to load an image.

A picture containing food

Description automatically generated

Figure - File Explorer Code

This single line uses the built in Unity EditorUtility library to open the file explorer, with parameters set to only use pngs. The function returns the file’s path, as opposed to the file itself. The path is stored in a variable to be used at a later stage.

Screen of a cell phone

Description automatically generated

Figure - Load Given Image

The GetImage function first checks if the path is null or not. If the path is not null, the file is loaded using Unity’s WWW class, and set in the tex variable.

A screenshot of a cell phone

Description automatically generated

Figure - Setting Texture to Object

The code checks if a valid selected object exists, and if so, the mesh’s material is set to a new material with the texture provided.

A picture containing drawing

Description automatically generated

Figure - Saving Texture File Locally

To ensure the texture can be used locally, the file is saved into the application data to be loaded in future imports. In its current iteration, a number is assigned to the file based on the order it was imported, however a much more universal implementation is to create a random key unique to each texture.

A screenshot of a cell phone

Description automatically generated

Figure - Checking for Local Files

To ensure that a local texture file is not saved multiple times, a function needs to be implemented to check the file’s path, and if the file’s path matches the application’s local assets path, then the file does not need to be saved again. To achieve this, the string of the image’s path is compared to the local files, and if the statement is true, the texCode variable is set to the local file’s name. The texCode is taken from the files path, and splitting it using the character ‘/’, and taking the last split in the array. This returns the filename including .png, so the same method is used again, using ‘.’ to split the string, and only storing the first section which will be the code used for the file.

## Exporting Files

The exported map file needs to contain information about every custom object on the map, such as the objects mesh, transform information and texture file. The file will be readable in a text editor, as well as by the importer to recreate the map. As the files can be read by the user in plain text, the values can also be edited in a text editor to then be read by the importer although this will mean that more checks will have to be done by the importer that the file is in a valid format.

A picture containing black, table, holding, food

Description automatically generated

Figure - Export Map Function

The export process contains three functions: setting the name of the map file, creating the file the map will be stored in, and exporting each map object to the file. The function takes in one parameter currently, the name of the map file provided by the user.

A screen shot of a smart phone

Description automatically generated

Figure - Setting Map Name

The first function simply takes in the name parameter and stores it into a local variable.

A screen shot of a computer

Description automatically generated

Figure - Creating Map File

The first line set’s the map’s file path, which combines the map folder directory with the mapName string as well as the .txt file extension. If the file doesn’t already exist, an empty file is written to the mapPath. As the whole file is overwritten when saved, nothing further needs to happen if the file already exists.

A screenshot of a cell phone

Description automatically generated

Figure - Exporting Map Objects

When saving the map, the first stage is to clear the map file by writing an empty string over the file. An array is then created with every map object, with each element being passed into the AddObject function, to add each map to the map file.

A screenshot of a cell phone

Description automatically generated

Figure - Adding Each Object

For each object, a list of information is stored, including the transform data, mesh data and texture file. They are each taken from various components of the object passed into the function and stored in individual temporary variables. The final contents that are passed into the text file is created by combining every previous string created. Said contents are appended into the text file.

# Input

An intuitive input system is an integral part of the project that needs to be thoroughly planned out to ensure the projects user experience is satisfactory.

## Keyboard and Mouse Inputs

A widespread adoption of WASD for movement in most software tools and games means that anyone with familiarities in similar projects will be able to grasp the implementation very easily. The WASD movement will be applied to the camera’s movement controls, with W and S for forwards and backwards respectively, and A and D for left and right respectively. Two additional buttons, Q and E will be assigned to control the cameras Y axis, moving it down and up along the axis to allow for easier navigation around the environment.

To control the rotation of the camera, a very clear and obvious option is the mouse to look around the scene in a 3D view. By parsing the mouse’s movements in its X and Y axis, the rotation of the camera can be set. This is an effective measure of handling the camera rotation, however the mouse will be the main input for most of the features, which means the mouse cannot be enabled for rotation at all times. The most effective way of handling this is to add a simple modifier key that enables the rotation when held. This is achieved by simply placing all of the logic for the rotation inside an if statement for when the modifier key is down. For both ease of use and logic reasons, this key is currently set to be the left CTRL key which is easily accessible in relation to the other movement-based keys.

The transformation states of the user are the next features to consider on the key mapping. The user can toggle between three states, being translation, rotation, and scale. When considering how to approach the switching of states, there are three options when it comes to key mapping. The first one being the use of a single key that switches the state to the next one in line as if the states were contents in an array being cycled through. This method is good for conserving keys for future features however the user experience of a system like this will be rather tedious when constant switching between the states is required. A slight improvement to this system would be to add a second key for going backwards which allows every state to be accessible within a single button press, assuming the three-state system is implemented. While this method is a lot easier to use than the single key method, it still does not achieve the best possible level of intuition as the keys are not directly mapped to access a state and instead change function based on the current state. This can mean that the users target state, and in turn target key, will change based on context which may be confusing or frustrating to users. From both personal opinion and based on pre-existing software implementations, the most user-friendly, straightforward implementation would be to bind each state to a different key so that the user knows that when the particular state key is pressed, the transformation state is guaranteed to be in that state. The obvious downside to this implementation is that it takes up more keys that could be used for other features, however this is very unlikely to cause any future problems in development. For this implementation the keys R, T and Y have been used for rotation, translation, and scale, respectively. These keys have been assigned to the state switching as it will be likely that the user will require easy access to these keys, with these features being in constant use, and to have the keys located next to each other to ensure the key mapping overall is cohesive and rationally designed.

## User Interface

Alongside the keyboard and mouse inputs, an effective and clear user interface is an important toolset to improve on the workflow and ease of use as well as enabling many features that couldn’t easily be mapped to keys.

One of the features that the UI will allow for is an object inspector panel. This panel will be able to display information about an object such as current position and rotation of the object, as well as allowing the user to input specific parameters via text boxes.

The User Interface will contain a menu, similar to those found in most production tools, that includes project management functions such as the save/load feature, file and importing settings as well as exporting the final product.

The following figure is an example toolbar from Visual Studio which contains the familiar features and tabs such as File, Edit and Help. Keeping in line with the industry standard bar will improve user familiarity and allow them to identify feature locations easily.

A screenshot of a social media post

Description automatically generated

Figure - Visual Studio's Toolbar Menu

## A close up of a sign Description automatically generatedTranslation Setting Menu

To give more control to the user on manipulating the objects the translation setting menu was added so that the values could be passed in by value, overriding any previous translations. The menu consists of 3 segments, one for each form of translation, with a title and 3 input boxes for the X,Y and Z axes. When a value is updated in the text boxes, the values are parsed and passed into the transform of the currently selected object.

# Example Scene One

The following figures contain some example scenes created with the first iteration of the transformed tools during development. As demonstrated the objects can be moved and rotated on all three axes as well as scaled. At this stage in development, the scaling is linearly across the axis, however the planned implementation of axis locking and precise value settings.

A screenshot of a computer

Description automatically generated

Figure - An example cube utilising all three transformation functions

A screenshot of a computer

Description automatically generated

Figure - Three cubes manipulated into rudimentary steps

A screenshot of a computer

Description automatically generated

Figure - Multiple objects positioned in 3D space

A screenshot of a computer screen

Description automatically generated

Figure - Alternative angle of Figure 7

# Example Scene Two

A picture containing circuit, table

Description automatically generated

Figure - Example Environment

The above figure demonstrates a basic example scene created within the tools which contains the three different rudimentary meshes transformed into various sizes, with different textures being loaded and applied from file.

## Example Export File

A close up of a sign

Description automatically generated

Figure - Example Text File

The above figure is the result of a map created in the tool, with eight line representing eight objects, each with their various attributes previously detailed.

# Conclusion

The aim of the project was to produce a 3D Level Editor tool that allowed the user to create 3D environments by transforming simple meshes around a scene, applying textures and exporting them to a file, which can be later imported back into the tool, including importing the file on another system.

## Unity Engine

Using Unity Engine allowed me to easily implement features using Unity’s own libraries that otherwise would have taken a lot of research, time and planning to properly include. During development, access to Unity’s own libraries meant easier implementation of features such as: file management, mouse and keyboard input, rendering scenes, 3D space coordinates, UI management and object structure. (Unity, 2020)

## Transforming

Overall, the transforming tools were implemented with an adequate method to produce required results. All three forms of translation: rotation, position and scale were implemented with relative success with the mouse dragging feature. The dragging used a linear scaling with minimal inaccuracies, if any, when applying the translations.

## User Interface

The creation of the user interface was the weakest element of the project, with only minimal time and planning being allocated to its creation. This has resulted in a very barebones design with only the key features being accessible to the user, with very little in regard to improving on an effective user experience. The input boxes for manually editing the transform values did function, however they were not without faults as an issue occurred where the values would persist and be applied to the next item that is selected.

A useful feature that wasn’t able to be implemented in time is an indication of the current selected object.

## Texturing

The texture application works as intended by allowing users to open the file browser and selecting a png file. The local saving works perfectly, with each file being assigned a random reference code which is used in the importing and exporting stages.

To improve the feature, more file formats could be supported to allow users to use a variety of image formats before converting them to a single format locally.

Another feature that would have been implemented with further development time is the ability to change the properties of the texture such as: tiling, scaling, transparency.

## Import/Export

The import/export system works as it was designed, with every map object being saved to the file, with the various elements in an ordered format. The file is written in a timely manner, stored in a local address and is publicly available to the user to save, transfer and manipulate any data.

The importer parses all of the data correctly and accurately reproduces the given file.

# References

Assimp, 2018. *The Open Asset Importer Lib.* [Online]   
Available at: http://www.assimp.org/  
[Accessed 22 November 2019].

GtkRadiant Team, 2016. *Radiant Level Design Tools.* [Online]   
Available at: https://icculus.org/gtkradiant/  
[Accessed 18 November 2019].

OpenGL, 2019. *OpenGL API Documentation.* [Online]   
Available at: https://www.opengl.org/documentation/  
[Accessed 21 November 2019].

Unity, 2020. *Unity Documentation.* [Online]   
Available at: https://docs.unity3d.com/Manual/index.html  
[Accessed 1 January 2020].

Valve Corporation, 2019. *Valve Developer Community.* [Online]   
Available at: https://developer.valvesoftware.com/wiki/Valve\_Hammer\_Editor  
[Accessed 29 October 2019].

# Project Repository

The project files can be found [here](https://github.com/AlexSDevDump/3D-Level-Editor). Please note that the project is developed using Unity 2019.3 and will require access to said platform or newer.