TOOLS PROGRAMMING

Lindsay Golder 1301186 CMP405

Contents

Summary

The “World of Flim Flam Creator” (WOFF) tool is a simple tool with a multitude of features for helping the user create. Some of these attributes include creation of new objects, focussing on selected objects, manipulating selected objects, moving the camera by clicking and dragging the mouse, object attribute editing and implementing a wireframe mode. These were all included to help the user in their activities. This document will: instruct on how to use the application; describe the features implemented to improve the tool’s usability and with world editing; and will discuss the accomplishments, failures, and improvements to be made in the future.

Controls

To utilise the application, multiple buttons and mouse movements must be learned. Table 1 illustrates the keys needed to interact with parts of the application.

Table 1: Keyboard Input

Key	Action
WASD	Move Camera forward, right, back, left
YX	Move camera down/up
QE	Camera Rotation
I	Zoom in on selected object

First, to create a new object the button highlighted on the toolbar below must be clicked. This creates an object in a separate place from the previous object and is also made selectable.

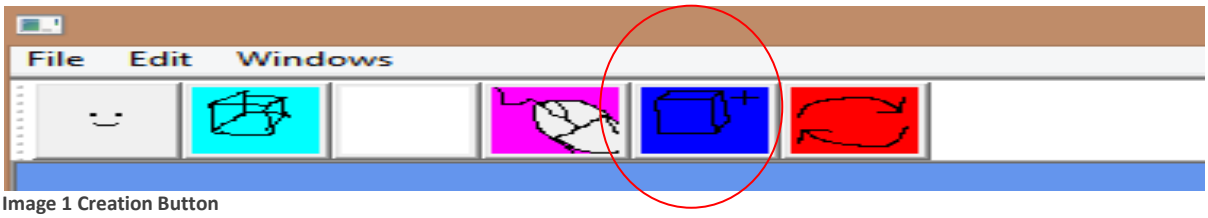


Image 1 Creation Button

Similarly, if the button with the wired cube is clicked, it will change all the objects created into wireframe mode, and the cube button next to it will return it to normal.
Furthermore, to move the camera up and down, simply press the “X” and “Y” keys. To move the camera forward and backwards, and left to right with the mouse, you must click on the highlighted button below. This button displays a picture of a mouse. You can now click and drag around the scene to move the camera, though, it is recommended to try this last.

Image 2: Mouse Camera Button

Next, to focus on certain objects, you must first click on the “Edit” menu and then choose “Select”. This brings up a list of selectable objects. Please select one of these objects and then press “Okay”. Now, if you press the “I” key, the camera will focus on the selected object.
Now, to manipulate the selected object’s attributes, you must select the “Windows” menu and go down to “Tools”. This will open a new dialogue (exhibited in Image 3).

Here is where you can enter values into the fields to change the selected object’s translation, rotation and scale. Once these details are entered, press the “Set” button to set these values. Next you must press the refresh button, highlighted below in image 4, to move and change the objects as defined.

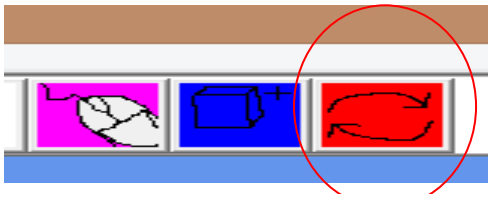


Image 4: Refresh Update Changes Button

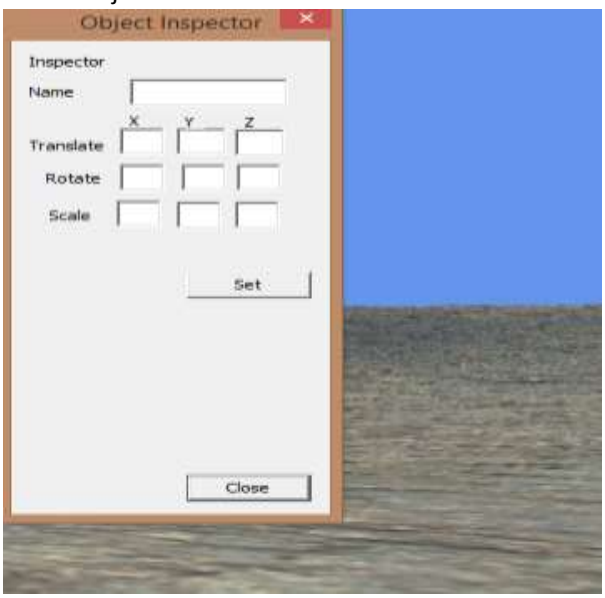


Image 3: Inspector Dialog & Set Button

Features

Usability

The first few steps of creating a tool that is appropriate for a user is to have well-designed user interaction. Any help for the user to use the tool is usually well-appreciated. Some of the usability in the original framework were enhanced to try and complete this goal. This includes having an optional mouse-controlled camera, object focussing and wireframe mode.

Camera Enhancement

Since the user would spend a lot of time in the tool environment, it was vital that the camera movement was improved.

More Camera Movement

Since the framework only moved within the x and z axis, it was extended by adding the ability to travel upwards and downwards. This was necessary because of the latter object manipulation function that would allow the objects to be moved higher than the camera in its' previous state.
This was completed by making a new vector that would represent the camera's up vector, by changing its' Y component to 1 as shown below.

```
m_camUp.x = 0.0f;
m_camUp.y = 1.0f;
m_camUp.z = 0.0f;
```

Using a method similar to how the camera moves right and left, new Booleans were created to check when the camera should move up and down. These would be checked as true if the X or Y buttons were pressed. The up vector would then be combined with the speed of the camera to update the position of the camera. This resulted in the camera being able to move up and down at the press of a button.

Mouse-controlled Camera

A great number of tools created involve moving the tool environment with the mouse. Game engine giant Unity 3D (2005), stated by the company to be the most used external game engine by developers (Unity3D, 2016), uses the technique of moving around the scene by pressing the middle button of the mouse to drag the scene up and down. The user can also zoom into the scene using the mouse wheel.
Similarly, Unreal Development Kit 3 (2009) has camera movement by using the mouse, though it involves dragging the mouse forward to move forward etc., almost like pushing into the scene using the mouse. This method is a lot closer to what was implented in the WOFF tool.
Based on these methods of moving around the scene, a similar function was set up in the tool. The scene can be moved around by clicking and dragging using the left mouse button. The environment is moved around in a diagonal motion.

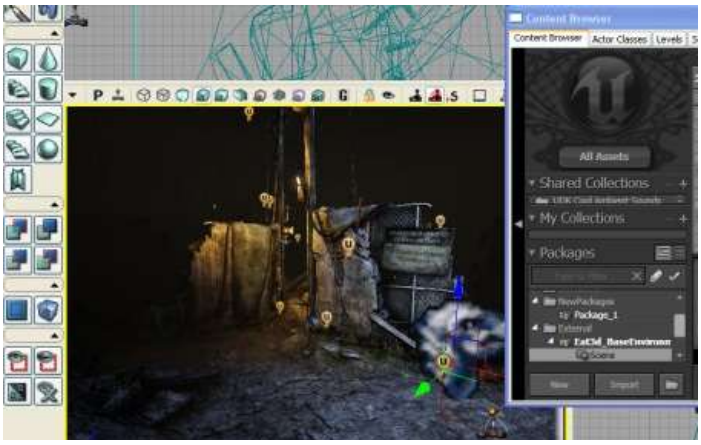


Image 5: UDK3- Free Unreal Engine Editor (Banzai, 2015)

This is achieved by first creating multiple Booleans that are used to tell which actions the mouse is doing. Next, there is testing the detection of mouse clicks, then- as seen in the code below- saving the position of the mouse at this point.

```
clickPoint.x = m_toolInputCommands.oldMouseX;
clickPoint.y = m_toolInputCommands.oldMouseY;
```

Thereafter, the application tests if there is any dragging being detected, and if so, save the new positions and compare them to the old ones and using the Capture function provided.

```
if (DragDetect(msg->hwnd, clickPoint))
{
	m_toolInputCommands.dragging = true;
	SetCapture(msg->hwnd);
```

```
}
else m_toolInputCommands.dragging = false;
```

By comparing, the application can tell which direction the camera should move in. When the mouse is no longer being dragged or clicked, the camera would no longer move. Lastly, a button was created that would activate this camera mode.
This method worked well and a basic mouse-controlled camera had been created.

Object Focussing

Another important attribute included in some tools including Unity (2005) is the focus being shifted to the object that the user has selected. This involves either the object being highlighted and/or the camera moving to near the object's position to have it in clear view. This is a useful addition because the user can get a close inspection of the object they are editing.
Currently, the tool has a select method where all of the objects in the scene are added to a list and can be selected using the dialog. Once one of these objects have been selected, the user can press the "I" button to them zoom in on this object. This was done by accessing the selected object's position which was found by considering the display list created from the scene graph. Since this was just moving the camera, it was not necessary to use the scene graph directly.

```
m_camPosition.x = m_displayList[m_InputCommands.selected - 1].m_position.x;
m_camPosition.y = m_displayList[m_InputCommands.selected - 1].m_position.y + 0.2;
```

The selected object's ID, minus one because of the IDs starting at number 1, was used as the number of the element of the array to access. The camera's position was altered depending on the current object's position, but differed slightly from the objects to allow the camera to see the object in plain view. This worked successfully, showing the appropriate object close-up on the screen.

Wireframe Mode

After looking at tools such as Maya (1998), it was established that wireframe modes in tools can be extremely useful. For example, it can help with editing edges or vertices on an object, allow them to see the accuracy of the topology, and can even help the user be made aware of the amount of polygons being used on a mesh. Although this tool would not be going into as much detail as Maya, it is still a useful feature to have for future development. Image 6 shows how wireframe mode has been activated on a sphere in Maya.

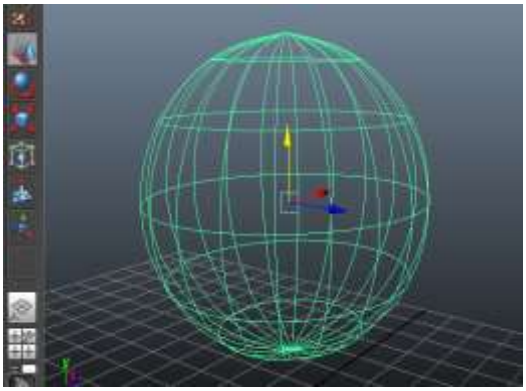


Image 6: View Modes (Kumar, 2012)

To obtain something similar, the tool was extended to allow the objects to appear in wireframe mode at the press of a button. A button was created in the main MFC file and checks were made to query whether it had been pressed yet or not. This would return a positive Boolean named "wireframe". In the Game file, where the objects are rendered, a check was made to see if wireframe mode would return true. If it did, all the objects in the scene graph would appear in wireframe mode. This was done when calling the Draw function.

```
m_displayList[i].m_model->Draw(context, *m_states, local, m_view, m_projection,
m_InputCommands.wireframe);
```

As shown in the code, the last entry to the function controlled whether wireframe mode was activated, and the external Boolean "wireframe" was inserted here.
This resulted in the objects being triumphantly shown in wireframe mode, and would be ready for any future development that would make this more useful to the user.

World

A few enhancements were added to the editor that would help the user with interacting with the area. This includes object manipulation, creation of new objects and basic object attribute editing. As established throughout this document, the WOFF creator tool involves the use of databases to edit the world. These improvements all involved manipulation of a database and they will be discussed in the following sections.

Object Creation

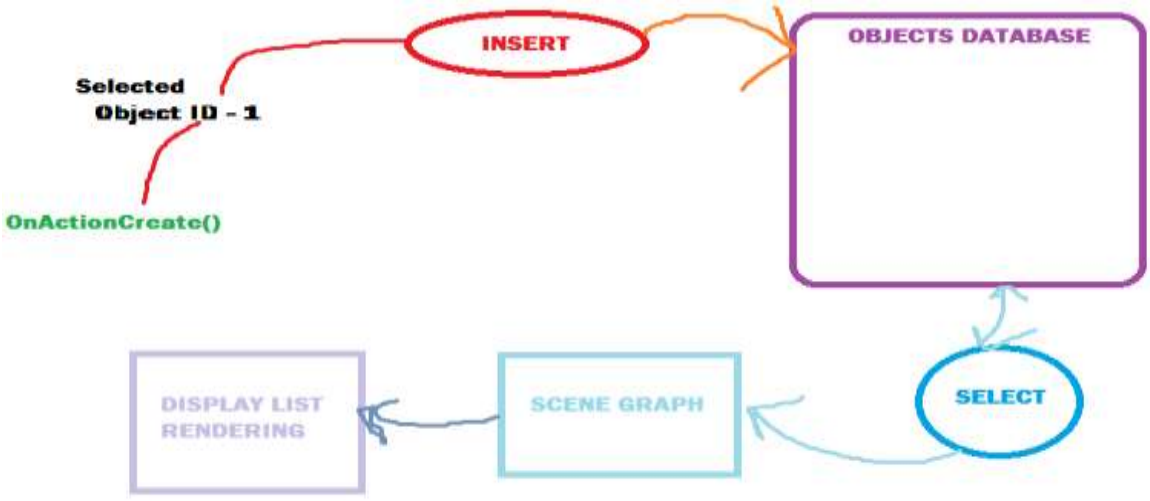
Image 7: Maya's Creation Toolbar

A very important feature of many tools is the option to create new objects within the scene. This method is prevalent in 3D modelling software such as Maya (1998) where an object can be created on the click of a button, as is evident in Image 7, where each shape shown in the toolbar can be created in the Maya tool. This is a very easy-to-follow and quick way of making primitives, and the images being used instead of words make this part of Maya accessible to multiple languages. The tool would aim to have a similar method with the same simple design.

First, a button was created that resembled Maya's cube creating button. Then, a new function was set up that would communicate with the "Objects" database using the "Insert" command. Inserted into the database was a new entry that contained most of the same elements as the other objects already in the scene. However, the differences incorporated the object's ID and position. The ID was made to be the total number of objects incremented by one. This was a way of counting to the latest ID and increasing it to give the new object a unique ID.

The position was changed in a similar fashion to prevent the creation of multiple objects in the one place. The new object's ID was used as the "X" and "Z" positions to give them somewhat of a distribution across the board. These changes were submitted to the database.

To establish this new object in the scene, the scene graph had to be updated. This was done by first creating a new function named "UpdateSceneGraph" where data was retrieved from the database using the "Select" command, specifying the latest row in the database by using "WHERE ID = (SELECT MAX(ID) from Objects". The new object was created using the data taken from the database. Now this object could be pushed onto the scene graph, and the display list could be updated to render said object.



This flow chart shows roughly how the function communicates with the database, and other functions follow a similar path. This resulted in a new object being created which looked the same as the other objects in the scene, like that of Maya.

Object Manipulation

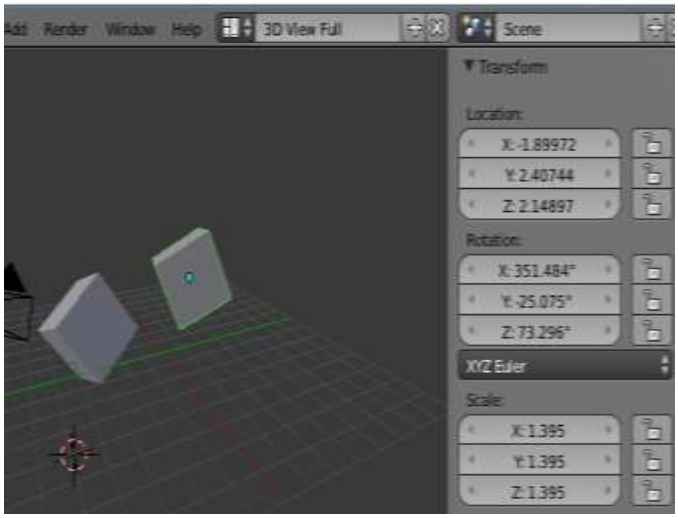


Image 8: Transform Panel (CodeManX, 2013)

An extremely useful feature found in various tools and game engines is the ability to edit the object's position, rotation and scale. The image depicted in Image 8 shows the transformation window found in free-to-use 3D modelling software Blender, and the cubes being transformed to the entries. The user can alter the object's settings by typing the number in the given fields, or by dragging the field left and right to increase or decrease the number. This tool would focus on the former rather than the latter. It was deemed necessary for this tool to include an aspect like this as it would be useful for the objects to be movable. First, a modeless dialog was created that resembled a very familiar translation tool, with three edit control boxes (X,Y and Z) for each action. One problem arose, yet, as the edit boxes would return a string instead of the float needed. To tackle this problem, the dialog box text was taken and converted using the "swscan_f" function into a float. This new float variable could then be used to change the object's settings.

A button was created named "Set" which would be used for establish the values in the fields as the object's new settings. New functions were created that would edit and update the scene graph with the new positions etc. being accepted as parameters. Furthermore, the display list was rebuilt and updated to move the object visibly into the correct place.

Again, the selected object's ID was used to access the scene graph's details and could then alter the positions, scales and rotations of the selected object. Once the dialog was closed, the button could be pressed which would update the object's position in real-time. This resulted in the object being altered in the user-dictated way.

Attribute Editing

Another useful inclusion in a tool is the ability to change the assets used for the aesthetics of the object and giving it identity, also seen as attribute editing. This is very prominent in Unity 3D which has an inspector, which is depicted in Image 9, that can be used to edit many parts of the object including its place in the world, plus its' model, scripts, materials, name, and many more. An object can be made to look extremely different despite it having a very similar set-up as the other objects.

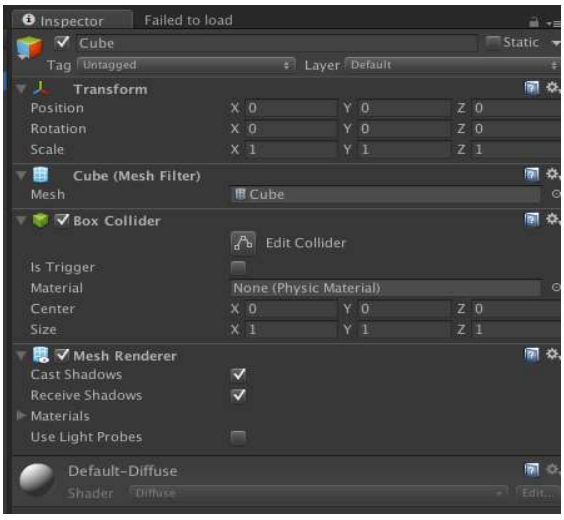


Image 9: Screenshot of Inspector(Sojitra, 2016)

The tool makes a start at giving the user the opportunity to edit the object's attributes by allowing them to change the texture. The dialog that controlled object manipulation was edited to become more like an inspector, and was given its own class named "ToolsDialog" which handled the modeless dialog. On top of the placement editing, there were fields created for the texture and mesh file destinations. Here the player could enter the file destination into each and the application would load the appropriate texture, if entered correctly and the file valid.

The creation of this method was done in a very similar way to object manipulation, though, involved a

different way of conversion. This was odd because the edit control used “CStrings” but the database required a standard string. To attain this, the “CString” was first converted to an LPCSTR before being defined as a standard string. This string could then be used in the database for texture and model directories, like how the positions were entered in the database.
There was also a problem if there was no entry into the fields, so to circumvent this causing issues, default directories were made for the texture and model file names in this circumstance.

Naming Objects

The framework had the option of selecting objects in the scene, and identified them using their ID’s. This was a handy method; however, it was difficult to know which object was selected due to them just being numbered. Naming an object can give it an identity and allow the user to call it something they will remember. Although a lot of the objects look the same in this tool, this would not be the case in the future. In the inspector, the user is given the option to rename the object which, after pressing the “refresh” button, is inserted into the selectable object list.
This was achieved by a similar method to the attribute editing. The next task was to update the selectable list, which was done by converting the object’s name string to a “wstring” by using the “assign” function equipped in “wstring”. This could then be added to the list. Then, the list had to return which object had been selected. This was done by getting the index number by using the list’s “GetCurVal” function. This index number was then used to access the scene graph and return the ID at this number. The current selection was then set to this ID.
This ended in the user being able to click the name of the object in the list and the appropriate object being selected despite the strings involved.

Critical Analysis

There have been many successes in the creation of the tool, but, there have also been times where there could have been vast improvements made to the methods used, or features that may have been easily added based on the knowledge gathered already. This section will discuss and highlight areas where the project was a worthwhile accomplishment, and places where it could be improved or extended upon in the future.
First, it is delightful to include the opportunity for the player to load in their own textures onto the models in the scene, and can see their changes being made almost instantly. Unfortunately, the user must type out the name of the destination of the texture to do this, which can cause errors if directories are misspelled or left blank. The blank issue was faced by having a default entry as the directory, but there are currently no fixes for misspellings. A possible fix to this would be to have drag and drop file name fields, like video editing tools like VideoPad Video Editor (2008), depicted in Image 10, where the player can simply drag in a file and the directory would be figured out by the application. Windows MFC provides the option of having fields accepting dragged files, so the next step would be to take these files and calculate the directory. This would save the player from tedious typing, and limit the possibility of errors arising. Overall though, the feature was implemented and the textures are rendered onto the model, which is the main purpose of this. The usability, in this case, could be improved in the future.

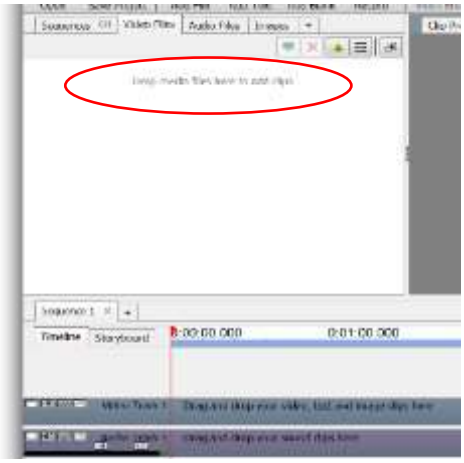


Image 10: VideoPad Video Editor(NCH Software,2015)

Furthermore, it could be seen as a victory that the objects could be turned almost transparent using wireframe mode as this could allow many opportunities in future development. However, as it stands all the objects in the scene are changed together. To optimise the wireframe mode, the wireframe could have been a checkbox in the object inspector window. This way, each individual object could have had its’ wireframe mode set to true instead of them all together. On the other hand, the attribute was displayed successfully anyway so the individual assignment of wireframe mode could be applied in the future easily.

Another acquired goal was the ability to manipulate a selected object in multiple ways by entering details into fields and clicking a few buttons. The user could then define exactly where they wanted their objects to be, and how they would be there. Nonetheless, the interface was not very well-implemented because of the need to press one button to set the data, and another to move the objects. This was because the dialog was modeless but no data was being passed between the dialog and the main window. This could be fixed by either making it a modal dialogue, or manually feeding the data between the modes. Nevertheless, the main function was achieved even if the buttons did not react in the expected way, but could be improved in the future.
Similarly, there is a problem with the edit control boxes in the dialog as although the set the data appropriately, they are not kept. This means that any time the dialog is opened and the set button is pressed, some of the object's data will be reset if the box is blank. This could be corrected in the future by setting the defaults to the selected object's attributes. Currently, though, the feature works well even with its flaws.
Furthermore, the camera dragging could be deemed a success as it allows the user to travel across the scene without having to remember many key presses. However, with modeless dialogs being created on top of the main window, it sometimes throws off some of the dialog's methods. This is especially evident with the selection of objects. However, both ideas work well and making them cooperate could be the next step to take.
Also, it was excellent to see the creation of new objects in the tool, mimicking the likes of Maya, Unity, and Unreal Engine. Having the program communicate well with a database and seeing the results appear on screen was a major accomplishment. Despite this triumph, there could have been more aspects included in a very similar way with more time. From this starting point, it could have been simpler to include methods like deletion and copy/pasting. Deletion could have been included by sending the "delete" command to the database instead of the "insert". Copying could have been achieved by selecting the object's data from the database with the "select" command and then inserting a new entry- using "insert"- with the exact same data except from the ID. These could have been implemented without much fuss, but time limits cut them out. Since the creation method was so like these functions, however, it was considered an achievement since other methods could be created based on work already done.
For future development, as well as some of the ideas mentioned above, there are many paths that could be taken. One way could be implementing certain objects that are tagged as collectibles, editing their collectability in a similar way to attribute editing. These would be added into the game world and could alter the main player object in multiple ways, possibly their health amount. The design would be very like that of level maker tools like Mario Maker, as can be seen in image 10, where a toolbar is present and collectables, enemies and stat enhancements can be placed around the level. The largest problem would be needing to include collision meshes to tell when the player would be colliding with the collectable, which was difficult to include with a time constraint. This would be intended as the next step in development to have these features implemented.

Image 11: Super Mario Maker (Corriea, 2015)

Conclusions

To conclude, the tool framework was extended to make the experience more user-friendly and executed multiple goals. This incorporated usability improvements like allowing the player to navigate the scene in various new ways, allowing them to focus on certain selected objects, and letting the user see their objects in wireframe mode. World enhancements were completed by including the ability to create new objects at the click of a button, to name objects, to manipulate the objects' settings, and to edit the object's attributes. Almost all of these were based on existing tools that were tested out. The tool

successfully communicates with a database and from there, loads and edits textures and models. Plus, although there were some issues, these were discussed and possible solutions were proposed for future work. Overall, the resulting tool was a good extension of the framework and has promise for future development.

References

Banzai.2015.*UDK3- Free Unreal Engine Editor.*[online image].Available From: https://klubba.files.wordpress.com/2015/04/ngt_udk.jpg?w=700 Accessed 20 February 2017.

Blender.1995.[computer software].Windows PC. Blender Foundation.

CodeManX,2013.*Transform Panel*.[online image].Available From: https://i.stack.imgur.com/TirnA.png .Accessed 19 February 2017.

Corriea,AR.2015.Super Mario Maker.[online image].Available From: https://cdn1.lockerdome.com/uploads/691a40fb7b9c3c68021e28dac26cd3da9c2cb96ead41a3712e721b9fd213c6d8_large. Accessed 14 April 2017.

Kumar,C.2012.*View Modes.*[online image].Available From: https://cdn.tutsplus.com/cg/uploads/legacy/462_Maya_Hotkeys/images/14.jpg Accessed 19 February 2017.

Maya.1998.[computer software].Windows PC.Autodesk.

NCHSoftware.2015.V*ideoPadVideoEditor.*[online image].Available From: http://cache.filehippo.com/img/ex/4055__videopad_video_editor_1_24_4_15png.png Accessed 14 March 2017.

Sojitra,M.2016.*Screenshot of Inspector*[online image].Available From: http://www.theappguruz.com/app/uploads/2015/06/screenshot-of-inspector.png. Accessed 20 April 2017.

Unity3D.2005.[game engine].Windows PC. Unity Technologies.

UDK 3.2009.[game engine].Windows PC.Epic Games Inc.

VideoPad Video Editor.2008[computer software].Windows PC. NCH Software.