

**Senior Design Project**

**The Design of a Virtual Reality Game for STEM Education**

ENGE476 Senior Design Project I

Department of Engineering and Aviation Sciences

University of Maryland, Eastern Shore

Tyler Gantz

Project Advisor, Dr. Lei Zhang

Submitted

December 7th, 2018

List of Contents

[List of Figures 3](#_Toc453190988)

[Abstract 4](#_Toc453190989)

[1. Introduction 5](#_Toc453190990)

[1.1 Backgound/Motivation 6](#_Toc453190991)

[1.2 Objective 6](#_Toc453190992)

[1.3 Design Requirements 6](#_Toc453190993)

[1.4 Design Constraints 6](#_Toc453190994)

[1.5 Design Methods 6](#_Toc453190995)

2[. Implementation Plan 7](#_Toc453191000)

[2.2 Timeline/Milestones/Delivery Plan 7](#_Toc453191005)

[3. Implementation 8](#_Toc453191006)

[3.1 Implementation of Task 1. 8](#_Toc453191007)

[3.2 Implementation of Task 2. 9](#_Toc453191008)

[3.3 Implementation of Task3. 11](#_Toc453191008)

[4. Conclusion. 15](#_Toc453191009)

[Acknowledgement 16](#_Toc453191010)

[Appendix 17](#_Toc453191011)

[REFERENCES 19](#_Toc453191018)

List of Figures

Figure 1. Design Method Flow Diagram………………………………………………………………………………… 5

[Table 1. Project Timeline and Delivery Plan 6](#_Toc453191241)

[Picture 1. Unity Logo 7](#_Toc453191241)

[Picture 2. Unreal Engine Logo 7](#_Toc453191241)

[Picture 3. Sample Scene 8](#_Toc453191241)

[Picture 4. Outer Space Picture to Support Setting 8](#_Toc453191241)

[Diagram 2. Gameplay Flow Diagram 8](#_Toc453191241)

[Picture 5. Self-Designed Circuit Object 10](#_Toc453191241)

[Picture 6. Futuristic Weapon Pack Example 10](#_Toc453191241)

[Picture 7. Force Field Effects Example 10](#_Toc453191241)

[Picture 8. DC Circuit Challenge Overview/Unity Software included 12](#_Toc453191241)

[Picture 9. Space Rotation Script Public Variable 12](#_Toc453191241)

[Picture 10. Energy Shield in use for Objective 14](#_Toc453191241)

[Picture 11. Generators for In-Game Objective 14](#_Toc453191241)

[Picture 12. Space Rotation Script Public Variable 15](#_Toc453191241)

Abstract

By the end of the project, summarize the project into short text and put here.

1. Introduction

## Backgound/Motivation

From the beginning of any engineering program, students are required to take difficult courses that involve new math and physics concepts. From experience, this can be overwhelming, as a lot of these concepts are foreign both mathematically and mentally. Visualizing topics such as circuit configurations or truss problems in statics are not always intuitive to new students.

As it stands, the standard curriculum for engineering students at UMES consists of just lectures with a handful of labs included in a four-year degree program. While this proves to be successful for most students, more learning tools are always welcome. Now, what if those learning tools broke away from the monotony of the typical lecture, where students sit quietly and takes notes, with the only interaction being asking questions to a professor. What if they provided an alternative to a lab, which typically only has students flipping a switch, measuring a value, then flip another switch, measure that value. In other words, what if there was a learning tool available to students that is both fun and interactive, but still educational.

This is where game-based learning comes in. Game-based learning, or GBL, uses competitive exercises, either pitting students against each other or getting them to challenge themselves in order to motivate them to learn better. Incorporating GBL into a curriculum has shown to motivate students to learn and, in turn, learn more effectively. Games provide an element of fantasy to the players, something that is not present in the typical lecture or lab. This element is what provides a fun and memorable experience for the students playing, which is what makes this approach to education so effective. [1]

Figure . Game Based Learning Logo

Technology in gaming has come a long way since its beginning back in the 80’s. Visually, games have evolved from the 8-bit, pixelated graphics to the stunning 4K graphics found in games today. The way the players control and play their games has also evolved from using simple game 8 button game pads to full motion controls. However, there is one innovation that stands out among the rest, and that’s virtual reality (VR). Especially in recent years, VR has become a prominent part of gaming with the release of popular hardware, like the HTC Vive and the Oculus Rift. Gaming platforms, such as Playstation, Steam, and various mobile applications all offer VR support with compatible hardware. Having this technology at our disposal is great, but is it really that vital to incorporate it rather than just developing a simple video game?

In the Merriam-Webster Dictionary, virtual reality is defined as an artificial environment which is experienced through sensory stimuli (such as sights and sounds) provided by a computer and in which one’s actions partially determine what happens in the environment. From this definition, one can tell that VR offers something that no 3-D game on a computer screen can offer, and that is true immersion. It is because of these immersive capabilities that virtual reality has proven to be an effective method in teaching and providing experience for users in situations that would not normally be easily accessible or reproduced. For example, VR is being used in sports to improve certain aspects of players’ games, such as situational awareness or correction in body mechanics. A specific example comes from a study that had shown that football players were able to shave off a second of their decision making in accordance to a defensive coverage by simulating these situations and testing them on how to handle it. Another example is of Kelly Oubre Jr., a player for the Washington Wizards in the NBA, who used VR to take a closer look at his shooting form. As a result, his true shooting percentage (a stat that factors in all types of shots in a basketball game) increased 3 percentage points. This technology is literally changing people’s careers for the better, so applying it to education for engineers is most definitely worth it.



Figure . Photos displaying the versatility of Virtual Reality and its applications

It is obvious that a combination of education, gaming, and virtual reality can provide a powerful learning tool for students.

## Objective

The objective of this project is to develop an effective game-based, virtual reality learning tool for electrical engineering students.

## Design Requirements

1. Educate and entertain through the immersive features of VR gaming.
2. Completable within 20 minutes
3. Cover 3 subjects in the Electrical Engineering field: DC Circuits, AC Circuits, and Digital Circuits.

## Design Constraints

1. The game will be developed for the HTC Vive.

2. The game must be playable within the minimum PC hardware and space requirements provided by HTC:

* Computer:
* GPU: Nvidia GeForce GTX 970, AMD Radeon R9 290 equivalent or better  
  CPU: Intel i5-4590, AMD FX 8350 equivalent or better  
  RAM: 4 GB or more  
  Video Output: HDMI 1.4, DisplayPort 1.2 or newer  
  USB Port: 1x USB 2.0 or better port  
  Operating System: Windows 7 SP1, Windows 8.1 or later, Windows 10

**Play area:** 2m x 1.5m (6ft. 6in. x 5 ft.)

## Design Method

*Figure 1. Flow diagram depicting the approach to developing the game*

1. Implementation Plan

## Tasks

* Task 1. Game Development Engine Configuration
  + Subtask 1. Unity vs. Unreal Engine
  + Subtask 2. Configure for development
* Task 2. Design Gameplay
  + Subtask 1. Scripting the game
  + Subtask 2. Create, or source required components to build the game
* Task 3. Development
  + Subtask 1. Scene Development
  + Subtask 2. Develop key features
  + Subtask 3. Develop

## Timeline/Milestones/Delivery Plan

|  |  |  |
| --- | --- | --- |
| **Time** | **Task** | **Comments** |
| **Semester 1**  **Week 4**  **To**  **Week 8** | Task 1 | Required weeks of deciding on a game engine and configuring Unity after choosing. Provided a smooth transition into developing the game. |
| **Week 4**  **To**  **Week 8** | Task 2 | Designed the gameplay, including setting and objectives |
| **Week 8 to Week 15** | Task 3  Subtask 1 and 2 | Began development on the game. Started by designing key features, such as the multiple-choice question system, shooting system, and in-game circuit components. |
| **Week 15 to**  **Week 27** | Task 3 continued Subtask 3 | Designed and developed puzzles, enemies, |

1. Implementation

## Implementation of Task 1. Game Development Engine Configuration

**Subtask 1. Unity vs. Unreal Engine**

The first task of the project required choosing a game engine to work with. The two options at hand were Unity, and Unreal. Research was done to find what obstacles and details would be presented with either one. Both engines possess a learning curve, but navigating through that curve is different for each engine. From a programming standpoint, Unity games are programmed using C# scripts, while Unreal uses C++. The key difference is that Unity has been available to the public since 2011, while Unreal has only been free and easily attainable for anyone outside of a professional setting since 2015. It is because of this fact that Unity help and information is readily available in books or online from a quick Google search.

Figure . Unity Logo

For Unreal Engine, it was widely considered the more powerful engine. This is the engine that some of the largest development companies use because it can achieve higher heights than Unity can visually. One would think that it was an obvious choice to go with the more powerful engine, however, one must also consider the documentation available for Unreal. The level at which the game runs from a functionality stand point is more important than any level of visual quality. Also, it is well documented that Unreal Engine powered games require more computing power, which directly conflicts with the minimum PC requirements constraint placed on the project. Considering all the information gathered, Unity was chosen to develop the game.

Figure . Unreal Engine Logo

**Subtask 2. Configure Unity for Development**

Once Unity was chosen, it needed to be configured for development. In order to configure the engine, tutorials were completed on scene building. The tutorials provided a work shop for working with object creation and script writing. Once the tutorials were completed, a pre-built scene was found online, which could be used to test the SteamVR Plugin. The SteamVR plugin is an asset pack found on the Unity Asset Store that provides a multitude of scripts and prefabs that are necessary for the basic functions of the HTC Vive. Scripts for object interaction, like picking up and throwing, and the teleportation feature are examples of the most useful assets in the pack. Applying these scripts to the prebuilt scene allowed for coding-focused experimentation more than taking the time to have to design a functioning scene. This is where an understanding of how the HTC Vive detects inputs in code was developed, which was vital for developing the game.

Figure . Sample environment used to configure engine

## Implementation of Task 2. Design Gameplay

**Subtask 1. Scripting the Game**

 Task 2 was all about designing the game from a creativity standpoint. Throughout the early configuration stages of the project, settings and suitable objectives within a certain setting were brainstormed. After a few weeks of weighing out options, the setting was chosen to be an alien space in outer space. The overarching objective was to escape the alien spaceship. The player is going to be put to the test through multiple choice questions (with an electrical engineering focus), physical objectives, and puzzles. The game is supposed to function like an escape room, which is an activity that has become popular over the recent years where groups enter a room full of puzzles, brainteasers, and scavenger hunts. The group has to solve these within a time limit to successfully complete the challenge. This is the concept that the game will take on. It provides a fun, interactive environment that requires the player to think through each step.

*Figure 5. Outer space, alien spaceship to support setting.*

Figure . Outer space, alien spaceship to support setting idea

A close up of text on a white background

Description automatically generated

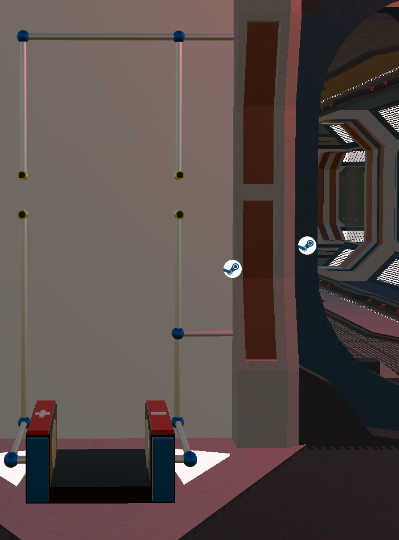
*Figure 6. Gameplay Flow Diagram, laying out what the player should expect*

**Subtask 2. Create or Source the Required Components to Build the Game**

In order to create this setting with its objectives, in-game objects needed to either be made or sourced. Unity allows for objects of basic 3-D shapes to be created and placed throughout the game in any kind of orientation desired. From there, you can add components, such as colliders, rigid bodies, scripts, etc. to the object, providing the user with a lot of customization. For this project, the structural design of each of the rooms, all circuit components, and over 90% of the scripts will be self-made using Unity’s built object creation or 3rd party software.

However, in order to make the highest quality game objects, skills in fields like animating and 3-D modeling are necessary. Unity presents a way for programmers to get by without possessing high level skills in these disciplines by providing the Unity Asset Store (UAS). The UAS consists of thousands of objects, materials, sounds, animations, etc. that have been made by other users who may have more experience working with these assets. Unity developers can then purchase these assets to then use in their own games. It breaks down a wall that would undoubtedly hold back programmers from developing great games. Here is a list of some of the assets purchased for this project, with a short description of how they are implemented:

* Vast Outer Space Skybox – The skybox, or shell, to the environment that provides the visual sensation of being in outer space
* Sci-fi Styled Modular Pack – A collection of assets themed around a futuristic setting. The majority of the objects used for building the setting were found in this pack, like the battery objects, the floor pieces, windows and doors.
* Force Field Effects – A collection of objects that take on a force field effect that will be used to block off objects from the player.
* Futuristic Weapon Pack – A set of high detailed weapons to be used in the game as the primary weapons for players in objectives that involve shooting.
* CRT LCD Shader – An asset that provides a screen in which you can attach a video player to. Mostly provides a level detail to the game because at close glance, the player can see lines of pixels as if he/she were looking at a real LCD screen.

*Picture 7. Example of Force Field Effects*

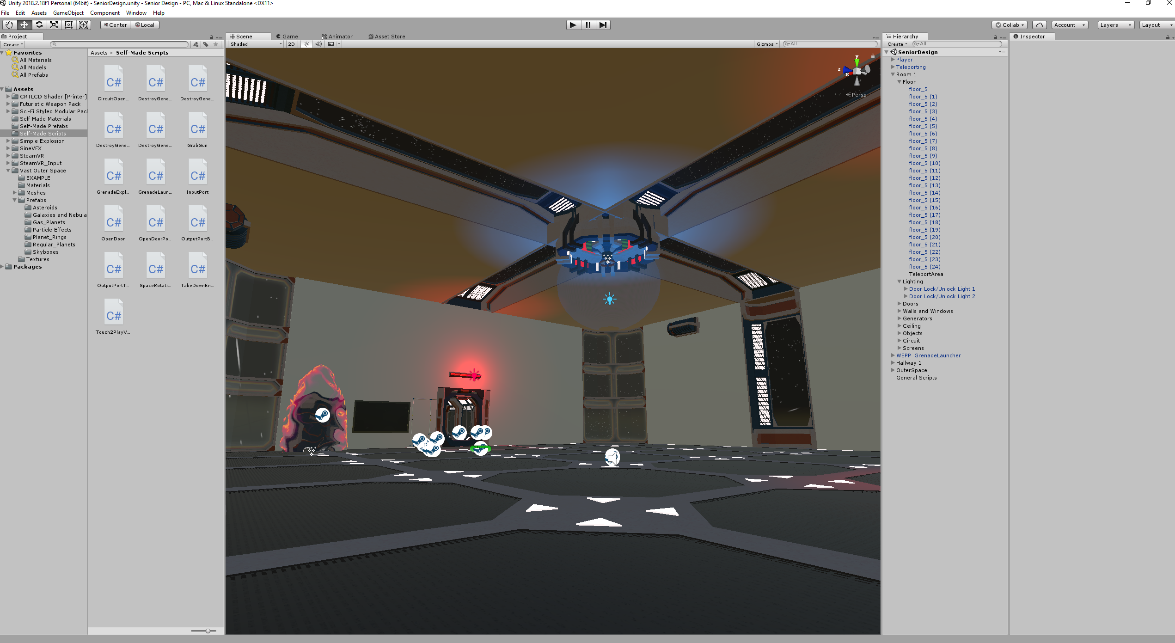
*Picture 6. Grenade Launcher from Futuristic Weapon Pack*

*Picture 5. Self-Designed circuit object*

## Implementation of Task 3. Development

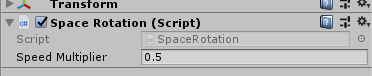
**Subtask 1. Develop the Scene**

The first step in the actual development process of the game was to build the room itself. The room shaped in a square, consisting of 225 sq. units. From the perspective of the player inside the VR glasses, the room will appear quite large. As stated earlier, structural pieces from the Sci-Fi Style Modular pack were used as flooring, windows, and doors. There were no favorable assets in the pack for large wall pieces, so they are self-made using large rectangular shaped cube objects (self-made shaders and materials will be applied to give them a better look). Using the transformation options (position, rotation, and scale) that Unity provides, the objects were situated to create a room you would expect to see in a spaceship. Objects, such as generators and batteries, were also included in the scene. They will be used for in-game objectives, like circuit design.



*Picture 8. Picture of the DC Circuit Challenge early in development, including the menus from the Unity engine software.*

**Subtask 2. Develop Key Features**

Once the room was created, it was time to develop some of the more important features that will be used throughout the entire game. The first feature developed was a script to make the skybox rotate. This was done to create a sensation of movement throughout space. This was done by writing a script using the “time.deltatime” built-in variable that denotes the amount of time it took to complete the last frame as the game runs. Using this time value, and multiplying it by a desired speed value (the variable used was “speedMultiplier), then applying it to the rotation transform of the skybox created the rotation. “speedMultiplier” was created as a public variable so tweaks to the speed could be done in the Unity inspector. 

Picture 9. Speed Multiplier in the Unity Inspector

With the game requiring recreational objectives, a feature was needed to ensure that was possible. A shooting system was implemented to satisfy this requirement. One of the most popular gaming genres are first person shooters, so many players could find enjoyment in this. The specific gun that was targeted for this was the grenade launcher. That required a script for the launching of a projectile, along with a script for exploding rounds.

For the projectile shooting, the first step was to make a slight addition to the grenade launcher object in the Unity hierarchy. An empty child object called “Grenade Spawn” was placed under the grenade launcher parent object, and this empty object was positioned at the barrel of the gun by changing its position values in the inspector. An important detail was to make sure this empty object was making no contact with the collider attached to the barrel of the gun. Now that the object was set, the script was ready to be written. Firstly, a starting amount of ammo needed to be established. This was achieved by defining the float variable “grenadelauncherammo”, which was arbitrarily chosen to be 15 to begin with and will be tweaked later according to the desired balance of the game. An if statement was written to check if the ammo count is greater than zero in order to let the gun fire. In order to make the game as realistic as possible, the script had to specify where the player was holding the gun. By using the OnTriggerStay function and attaching the script the trigger of the gun itself, when the player picks up the gun and holds it at the handle, the script returns a bool value of “true” for the variable “fingerOnTrigger”. By using the OnTriggerExit function, the value will return to “false” if the gun leaves the players hand. Having that condition set, the script needed to be written to check for the “fingerOnTrigger” value and to see if the player was holding down the trigger on the Vive controller. This was achieved by nesting 2 if-statements inside the ammo if statement, checking to see if both “fingerOnTrigger == true” and if the Vive trigger was in the “GetStateDown” position. If the conditions were met for these statements, the function called “Fire” would run. This function consists of an instruction called “instantiate”. This essentially spawn an asset that is not already in the hierarchy. It has 3 arguments that need to be satisfied, those being the object to be spawned, the position in which it is spawn, and the at what rotation values it spawns with. The next command references the rigid body component of the grenade prefab that is being launched. A forward, constant velocity is applied to the grenade when it was spawned, essentially creating a projectile. Each time the function runs, it subtracts one from the ammo count. The same process was repeated for the pistol and assault rifle.

The script for exploding rounds also made use of the “instantiate” instruction. For this script, the instruction was placed inside the “OnCollisionEnter” function, which waits for the object it is attached to to collide with another object before executing what is inside the function. Within the function, an explosion is instantiated, and the grenade is destroyed. That means, if the script is attached to the grenade prefab, when the grenade is launched from the gun, and it contacts any object, an explosion will occur.

Just about every 2-D and 3-D game include a heads-up display, or HUD, that displays important information to the players, such as health, ammo, and score. A VR game needs the same, however, the HUD needs to be more thoughtfully implemented. In 2-D and 3-D games, the HUD is generally found all around the edges of the screen, preferably not interfering with the core gameplay, yet readily available for the player whenever necessary. In VR, the field of view does not really allow for extra objects in the site of the player. The solution for that was to attach a HUD to the left wrist of the player. The HUD has the appearance of a futuristic tablet that only appears when the player tilts the left wrist upwards, like checking a watch (in Unity, the if-statement checks if the left hand has a local rotation value between 200 and 250 in the z-direction.

A screenshot of a cell phone

Description automatically generated

**Subtask 3. Develop Gameplay**

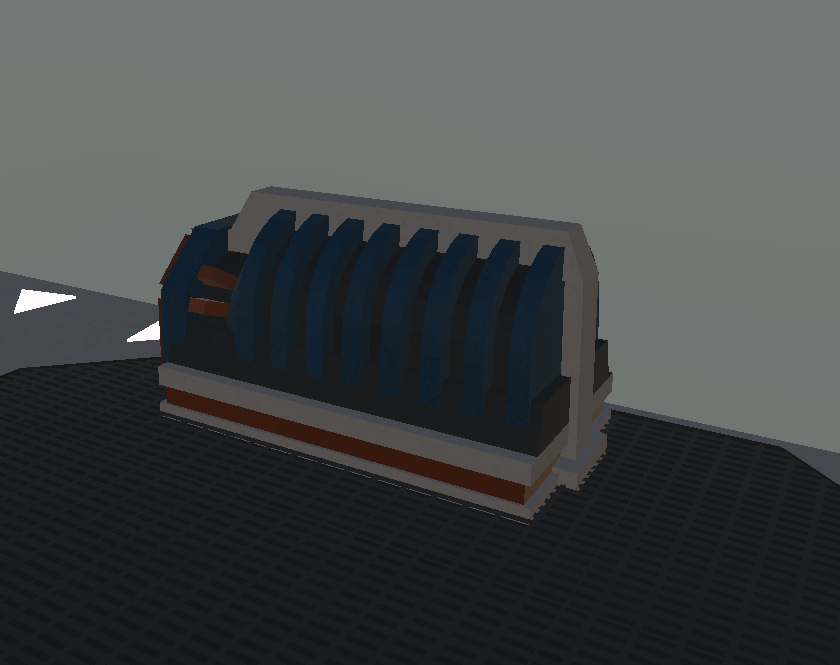
This section will essentially be a walkthrough of the game itself, and how each part of the game was implemented. It will be in the order that the player will face each objective as if they are playing the game.

To begin the game, the player will find himself trapped inside an alien ship of some kind. Shortly after waking up, an audio clip of an ally will begin checking on his well-being, followed by a clip of the head alien “boss”, informing the player why he is jailed, and what steps he can take to start his escape challenge. This was achieved by recording audio clips using voice changing software and inserting those clips into the game via an audio source object in the game that was set to play on start. A script was written to make those clips play one after another. Audio clips like these frequently appear in the game for both informative and aesthetic purposes.

The first step of the game is to complete the first multiple choice question by inserting the key found on the table in the center of the room. The key system was a more creative approach to progression throughout the game, rather than just telling the player to go from objective to objective. This was programmed by checking the distance between the key and the key slot in an if-statement, with the argument being that the key needed to be less than 0.1 units away. When the player places the key in the slot, the light above the question will illuminate, which is the visual confirmation that the question was activated. When the if-statement is fulfilled, it instantiates a point light object on the light object above the question and changes the “question1activated” bool variable to true, which allows the player to begin the question. Once the player completes the question, he will be prompted to complete puzzle 1.

Puzzle 1 consists of a circuit powered by a large battery that is connected to 10 resistors in parallel with each other. In series with the parallel resistors is a transparent power shield, with what appears to be a key inside. An audio clip will play once question 1 is completed giving the player a description of the technology, from which the player must decide on a plan to take down the shield. The objective is to overload the shield, which the player should deduce that he must destroy the resistors with highest resistance, but only 5 resistors can be destroyed without causing an explosion on the ship. To program this, 10 public variables were created for each of the 10 resistances. In order to maximize the learning experience, the resistors needed to be placed at random each time the game is started so returning players cannot complete the puzzle by memorizing the pattern at which the resistors need to be destroyed. To achieve this, the resistors were placed into an array at the start of the script. Using the Knuth shuffle algorithm, the array is randomized and placed into a new array. From this new array, each resistor is assigned to a resistor port in numerical order (Resistor1 = randomresistors[0], Resistor2 = randomresistors[1],…). Each resistor is instantiated into their corresponding ports. Now that the resistors were established where they need to be, the puzzle needed to be programmed. To start, each resistor had a resistance value tagged to them that corresponds to the colored bands on them. The script converts the tag that is initially a string into a float variable. Each of the 10 resistances are added up in the “totalresistance” variable. While the circuit is in parallel, it is not really important for the specific puzzle to calculate the parallel resistance, so resistances were added as if they were in series. The player will be directed to shoot out the necessary resistors with the pistol found at the start of the game. Each time a resistor is destroyed from the circuit, it will subtract its value from the total resistance. As stated earlier, the player must only shoot out 5 resistors to avoid a large explosion. This was programmed by doing a child count on the parent object “Puzzle 1 Resistors”. If the child count equals 5, then the script will check if the total resistance at [insert correct resistance here]. (Stopped working here 04/02/2019).

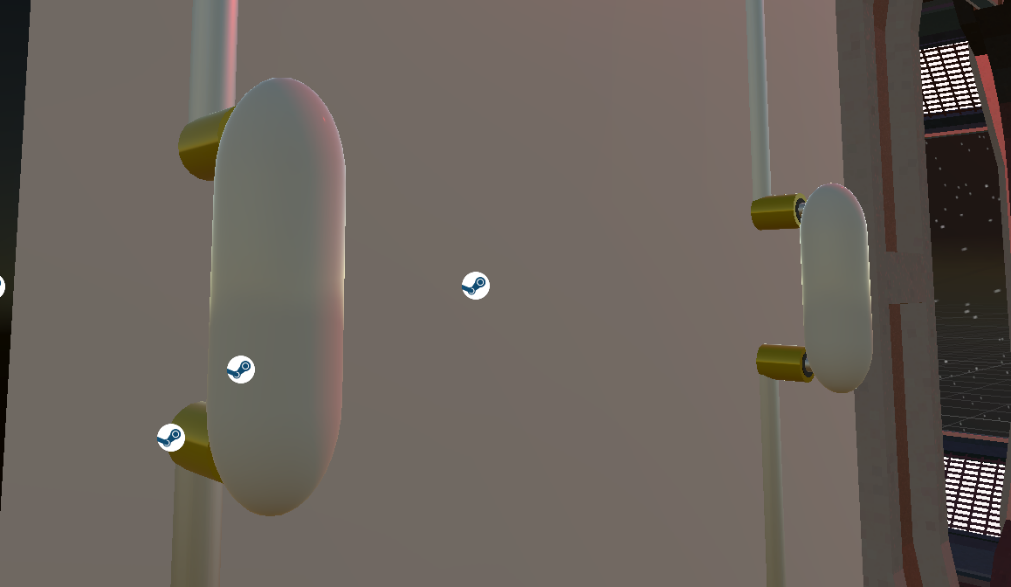
An objective in the game for the player is to bring down the energy shield keeping the player from reaching the circuit components to open the door by destroying the generators found throughout the room. Two scripts needed to be written to create this objective. Firstly, the generators needed to be destroyable. Using the OnCollisionEnter function again and checking if the tag of the object colliding with it is “Grenade”, made it so the generators could be destroyed using the grenade launchers. When the grenade collides with it, an explosion is triggered, and the object is destroyed. The second script was needed to check if all the generators were destroyed in order, which will then take down the energy shield. This was achieved by using the parent object called “Generators” as the public variable (this object contained all four generators in the room). The script takes the parent object, and repeatedly checks to see if the number of children objects inside is equal to zero. When all generators are destroyed, the child count equals zero, and the shield is destroyed, revealing the circuit components to the player.

*Picture 11. Generators found in the room that will be destroyed*

*Picture 10. Energy shield concealing circuit components from the player*

The next objective was creating the circuit design portion of the room. The circuit was essentially a voltage divider, with the load being the door opener. The player is to insert the resistors into the ports and then the battery into the connector at the bottom of the circuit. Once the components are successfully placed, the door will open. The way this was scripted was by making both the input and output ports and the battery connector as triggers. When those triggers are satisfied, bool values are set as true. These bool values are then collected by another script, checking to see if they are true using if-statements. When the if-statement is satisfied, the animator for the door is cued, opening the door, allowing the player to escape the ship.



*Picture 12. Resistors inserted in the ports*

1. Conclusion.

At the conclusion of the first semester working on this project, I have learned a lot about the design process in game development. The main issue I have encountered is not having a well-defined design method in place. For the second semester, I will be sure to have a stronger approach for laying out objectives, so I can achieve the goals faster and more efficiently.

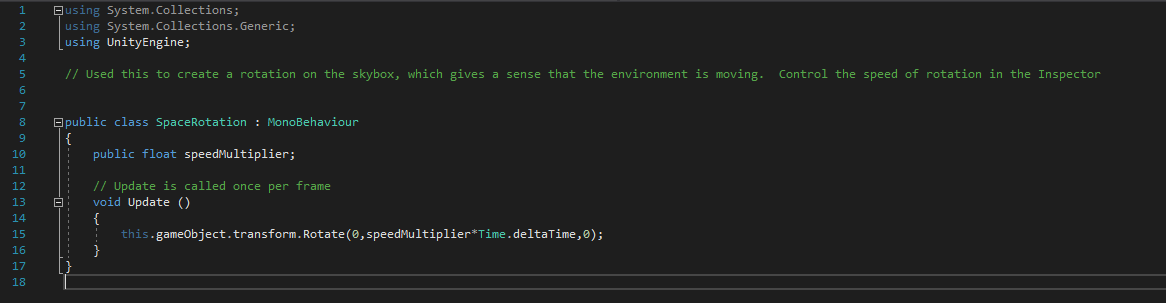
In terms of the technical information I have gained, I learned just about everything about C# programming that I know from this project. This is providing me with a great foundation in object-based programming that can be used in many programming professions. I’ve gained more from this project so far than most classes throughout my entire time in the Electric Engineering program here at UMES. Overall, this project has undoubtedly made me a better engineer in just one semester.

Acknowledgement

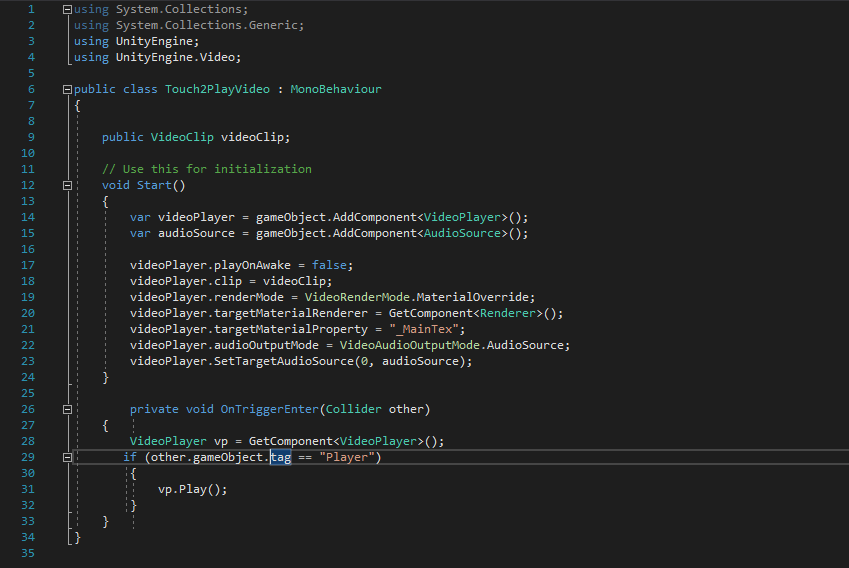
Joe McGinley - one of my best friends, and a brilliant programmer who’s worked with Unity and helped me along with the way.

Appendix

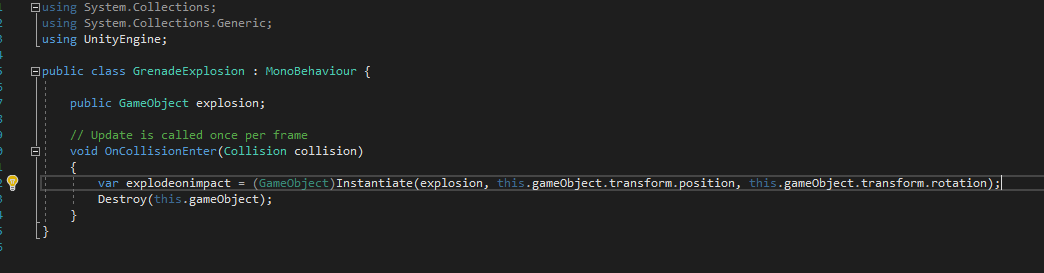
1. Source Code. Script for Rotating Skybox



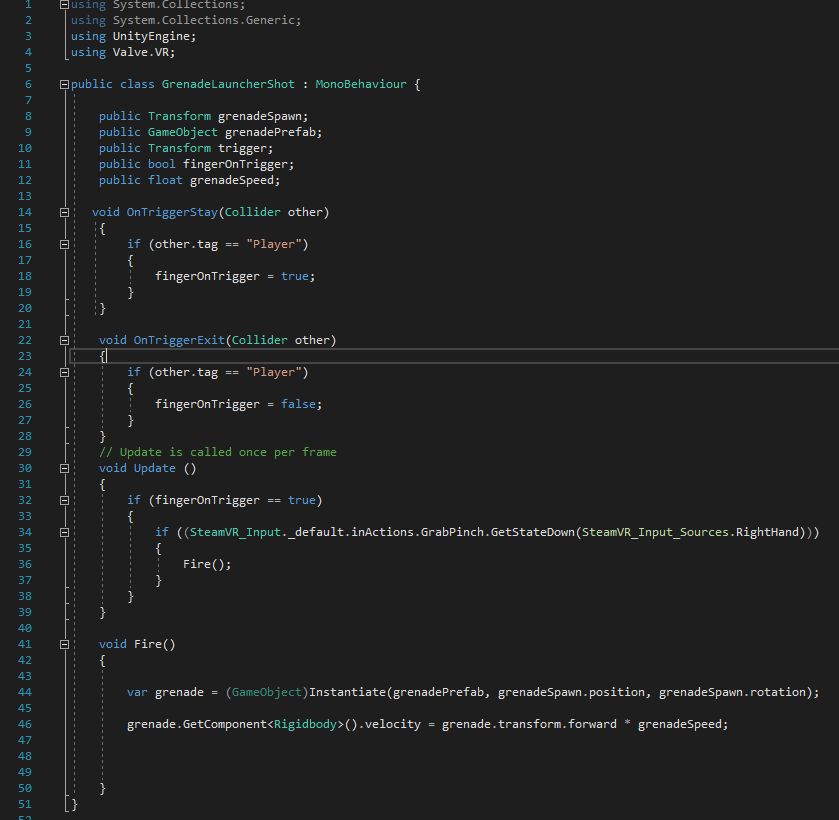
1. Script for Touch-Activated LCD Screen



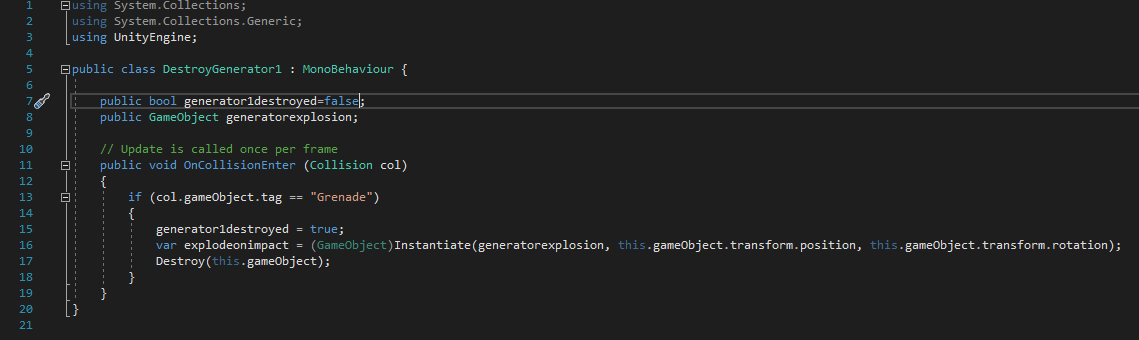
1. Script for Exploding Rounds



1. Script for Shooting Projectiles



1. Script for Destructrable Generators



REFERENCES

[1] Teed, R. (2018, May 07). Game-Based Learning. Retrieved December 12, 2018, from https://serc.carleton.edu/introgeo/games/index.htm