

**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. Game-Based Learning Logo 5](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240372)

[Figure 2. First Call of Duty vs. Newest Call of Duty: Growth of Technology in Visuals 5](#_Toc6240373)

[Figure 3. Versatility in Implementation of Virtual Reality 6](#_Toc6240374)

[Figure 4. Illustration of the different elements game--based learning incorporates 6](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240375)

[Figure 5. Flow diagram displaying the design approach 7](#_Toc6240376)

[Figure 6. Project Timeline 9](#_Toc6240377)

[Figure 7. Unity Logo 10](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240378)

[Figure 8. Unreal Engine Logo 10](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240379)

[Figure 9. Scene used during engine configuration 10](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240380)

[Figure 10. Picture supporting the proposed alien spaceship setting 11](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240381)

[Figure 11. Gameplay flow diagram 11](#_Toc6240382)

[Figure 12. Examples of some assets found in the game 12](#_Toc6240383)

[Figure 13. SteamVR Objects 13](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240384)

[Figure 14. Code for the body collider object script 13](#_Toc6240385)

[Figure 15. Picture of the in-game hand models 14](#_Toc6240386)

[Figure 16. Picture in each lens vs. Perceived image 14](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240387)

[Figure 17. Picture of a single floor piece 15](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240388)

[Figure 18. Overview of the entire room 16](#_Toc6240389)

[Figure 19. LCD Screen object 16](file:///F:\Senior%20Design%20Report%2004.9.2019.docx#_Toc6240390)

[Figure 20. Cover screen, question screen, and correct screen. 17](#_Toc6240391)

[Figure 21. View of Multiple-Choice Question Buttons, and full screen object view 18](#_Toc6240392)

[Figure 22. Picture of some of the keys found in the game 19](#_Toc6240393)

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 1. 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?  

Figure 2. First Call of Duty vs. Newest Call of Duty: Growth of Technology in Visuals

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. "I really saw a difference in my jump shot and free throws," says 20-year-old wing Kelly Oubre, who grew up playing "Call of Duty" and is used to wearing a headset. "I could see my mechanics, what I needed to do right." Oubre's true shooting percentage is up this year, from 50.7 to 53.4.[2] This technology is literally changing people’s careers for the better, so applying it to education for engineers is most definitely worth it.

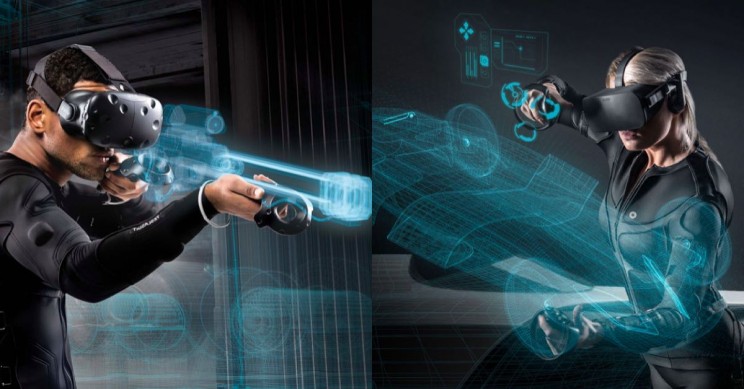


Figure 3. Versatility in Implementation of Virtual Reality

It is obvious that a combination of education, gaming, and virtual reality can provide a powerful learning tool for students if done correctly. While school subjects being a main part of a game does not sound like the most enticing idea, using the concepts of game-based learning correctly and productively could prove that to be false. Combining different gaming elements like action, shooting, puzzles, and the creative implementation of visual and sound effects with educational topics can create an experience where the player is having as much fun as a noneducational game, while learning at the same time. Games provide a sense of competition that could drive students to try harder to learn subjects so they can succeed in the game, in turn, leading to success in the classroom. This is a learning tool that needs to be taken seriously and given a chance in schools or at home today.

Figure 4. Illustration of the different elements game--based learning incorporates

## 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

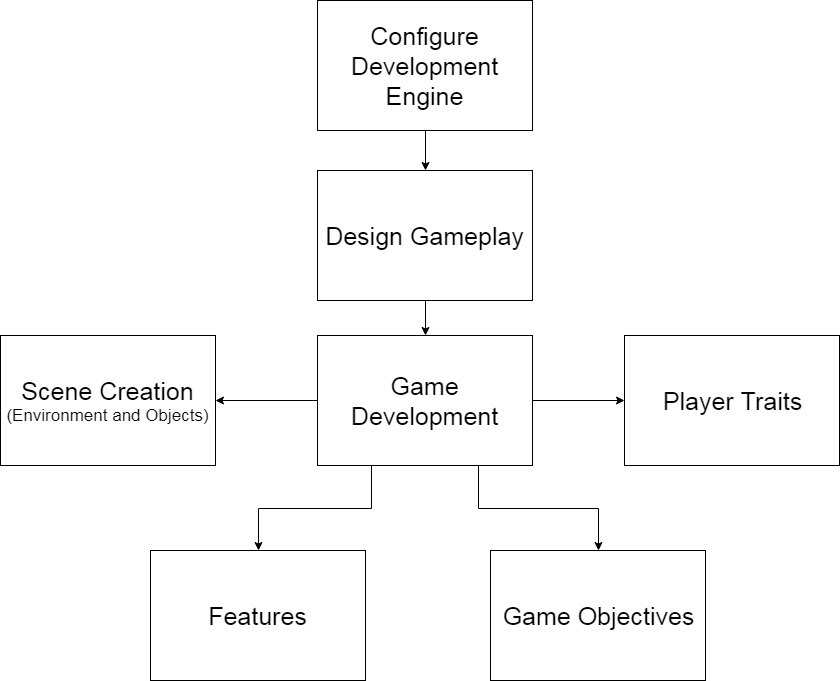


Figure 5. Flow diagram displaying the design approach

1. Implementation Plan

\*This was a solo project designed by Tyler Gantz

## 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: Scene Development
  + Subtask 1. SteamVR Player Prefab
  + Subtask 2. In-Game Environment
  + Subtask 3. Multiple Choice Question Screens
  + Subtask 4. Keys
  + Subtask 5. Circuit Components
  + Subtask 6. Energy Shields
* Task 4. Development: Player Traits
  + Subtask 1. Health
  + Subtask 2. Ammo
* Task 5. Development: Features
  + Subtask 1. Heads-Up Display (HUD)
  + Subtask 2. Multiple Choice Question System
  + Subtask 3. Draw Mode
  + Subtask 4. Gun system
  + Subtask 5. Enemies
  + Subtask 6. Audio and Visual effects
* Task 6. Development: Game Objectives
  + Subtask 1. Puzzle 1
  + Subtask 2. Enemy encounter 1
  + Subtask 3. Circuit Design 1
  + Subtask 4. Puzzle 2
  + Subtask 5. Circuit Design 2/Enemy Encounter 2

## 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 22** | Task 3 continued Subtask 3 | Designed and developed puzzles, enemies, |

Figure 6. Project Timeline

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 7. Unity Logo

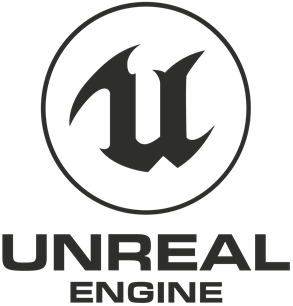
For Unreal Engine, it was widely considered the more powerful engine. This is the engine that some of the largest development companies use because it can achieve higher heights than Unity can visually. One would think that it was an obvious choice to go with the more powerful engine, however, one must 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 8. 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 9. Scene used during engine configuration

## 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 10. Picture supporting the proposed alien spaceship setting

The diagram below provides a detailed layout of how the game will flow.

A close up of text on a white background

Description automatically generated

Figure 11. Gameplay flow diagram

**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.

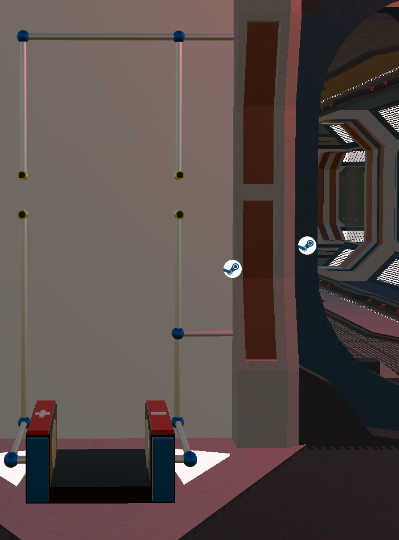


Figure 12. Examples of some assets found in the game

## Implementation of Task 3. Scene Development

**Subtask 1. SteamVR Player Prefab**

The first object that needs to be mentioned in the scene is the SteamVR player prefab. This prefab includes object, object components, and scripts every developer needs to get started on a VR game. The screen shot on the right displays the most important objects found in the prefab.

The body collider object contains the components that essentially act as the mid-section of the player in the game. While there is no visible body object, there is a capsule collider component attached to this object that is oriented as if there was a body on the player. This is a necessary component to ensure the player interacts with the other objects in the scene correctly. For example, if this component did not exist, an attacking enemy would have no way of contacting the player. This object also contains a small but important script that establishes the relationship between the height of the capsule collider and the “head” of the player. It is what makes the perception of the player from eye sight to the ground feel realistic in-game.

Figure 13. SteamVR Objects

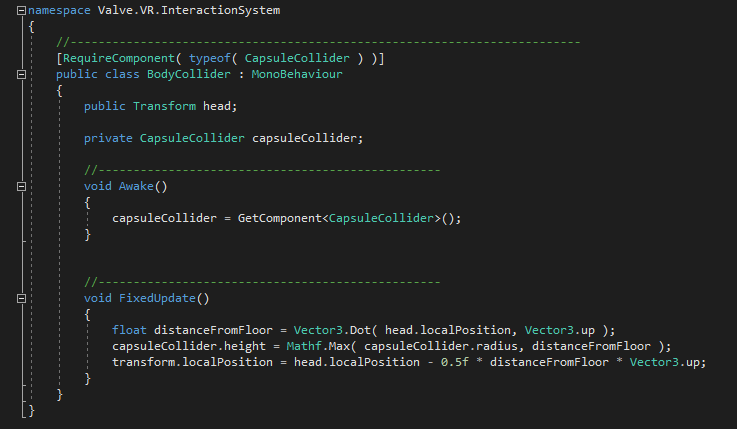


Figure 14. Code for the body collider object script

Next up on the list are the hands of the player. The left and right hands each contain colliders just like the body object, along with some of the most vital scripts for object interaction in-game. By default, the colliders attached to the hands are just spherical colliders placed near the visible hands the player can see in-game. However, 2 modifications were made to the colliders. The first was that the sphere colliders needed to be move and resized so they fit the hand models in-game better. It caused problems when working with interactable objects because the player had to reach too far into the objects to interact with them. The second modification was adding a small capsule collider to more realistically fit the hand models. As seen in the accompanying picture of the hand models, the pointer finger sticks out while the rest of the fingers create a fist. For some of the interactions where precisely contacting objects was unavoidable, it was essential that a collider was added to the pointer finger to add a precision collider to avoid unwanted collisions with the larger-sized sphere collider. Specific examples of scenarios like this will be pointed out later in the report.

A picture containing person, clothing, man, wall

Description automatically generated

Figure 15. Picture of the in-game hand models

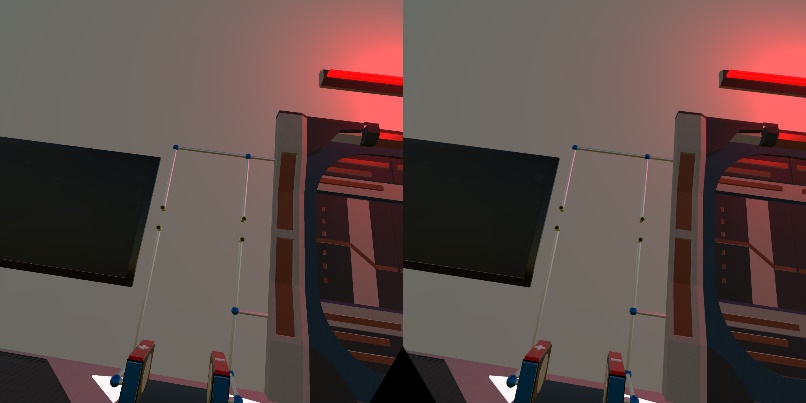
The VRCamera object is what provides the view from the HTC Vive headset. The trait that sets it apart from a typical 3-D game camera is that it has controlled values for stereo separation and stereo convergence. Stereo separation in relation to VR is the distance between each eye. Each eye perceives the world slightly differently due to them being separated from each other, however the brain perceives the view as one single image. This setting is what creates immersive feeling in VR, where it feels like your in a new world, rather than looking at a screen inide a heaset. It automatically sets the field of view so that the perception is similar to the real world. This perception of seeing one single picture rather than two separate pictures is the result of stereoconvergence. The values for each of these are fine tuned to create the most realistic view possible for the player and to help avoid eye-strain and motion sickness.

Figure 16. Picture in each lens vs. Perceived image

The [SteamVR] object contains simple but useful scripts. The first script initializes SteamVR on start-up. That ensures SteamVR will open so the player does not have to worry about initializing it themselves and helps avoid communication errors with the equipment. The other script initializes action sets created for the controllers in the game. This is what provides the game the specific input instructions mapped out by the SteamVR plugin for each of the controllers. It is vital in ensuring the game runs correctly with the hardware

**Subtask 2. In-Game Environment**

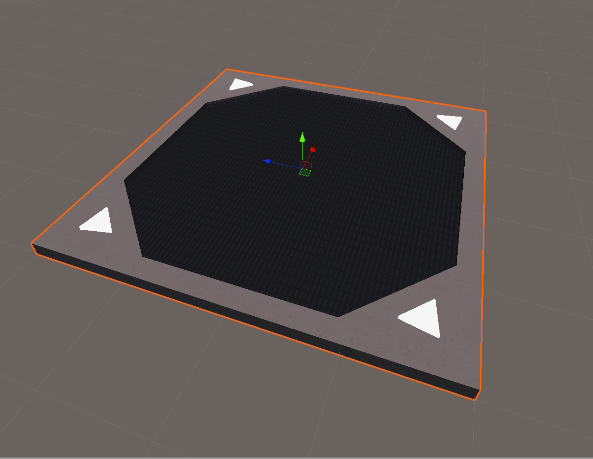
After the idea to create an action escape room game came to mind, it was time to create and build the room itself. A few key points that needed to be adhered to was a lot of space was needed so all features and objectives could fit, it needed to take on a futuristic setting, and it needed to include structures that could be used in fights against enemies. The Sci-Fi Styled Modular Pack (SFSMP) was the go-to asset pack for building the room as it seemed to something to fulfill every need. This was made possible with detailed floor pieces with built in lights and a lattice cover that scaled to larger sizes nicely. The 1.5-times scaled objects were placed in a 5x5 grid. The room ended up being 225 sq. units in size, which is perceived to be quite large when the player enters the VR world. It was nice start to creating the room.

Figure 17. Picture of a single floor piece

Next, surrounding vertical objects were placed, like walls, doors, and windows. The walls are made up of several thin, but tall and wide cubes made directly in the Unity editor. Doors were placed on the front and far end of the room, one designated to be the door the player wa theoretically entered from, while the door on the far end is the one the player needs to open to escape. In each of the 4 corners and in the center of the left and right walls, windows were placed so the player can see into outer space. The corner windows were made out of 3 corner glass panels found in the SFSMP stack on top of each other with some overlap for a decorative effect. The windows in the side walls were created by rotating the big windows found in the same asset pack 90 degrees in the z direction and then elongating them to match the height of the walls. Once again, each of the objects scaled nicely, which created good visuals to the primitive version of the room.

As previously mentioned, structures needed to be placed in the room that could provide cover during the action sequences in the game. The solution to this was placing 4 pillars, with one in each of the 4 quadrants of the room. The columns are aesthetically pleasing and also block sightlines, which can provide the player safety in enemy fights.

The room obviously needed some decorative elements as well. One of the more impressive looking objects in the SFSMP was the projector. The base object of it looks great, but the projection particle system added a futuristic flare to it. A giant projector was placed in the center of the ceiling, which will act as the system in which the character receives items from allies. The lights included in the asset pack were also great for decoration. There are multiple styles included in the pack. Lights were applied to the left and right side of each pillar, along with on top of the doors.

An overview of the completed room is shown below.



Figure 18. Overview of the entire room

**Subtask 3. Multiple Choice Question Screens**

A repeating objective the player will face in the game are multiple choice questions. In order to create a functional multiple choice question machine, 2 problems needed to be solved: finding a way to successfully display the question and answer choices, and how to allow the player to input an answer.

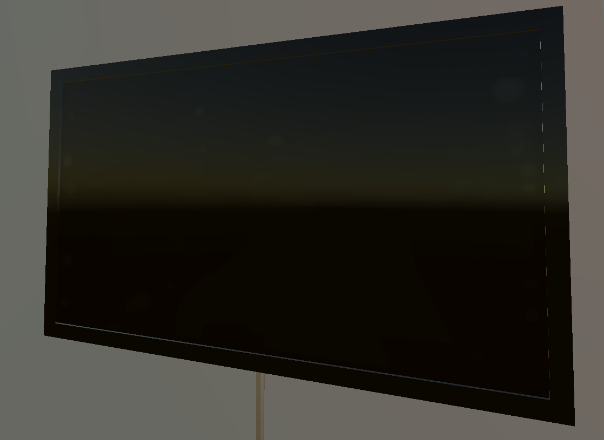
There are multiple ways of displaying text and images in Unity. However, choosing a clean, aesthetitcally pleasing way to present the questions was the priority. This is where the LCD Widescreen asset came into play. This asset provides an object with an attached shader that is essentially an in-game LCD flat screen TV. Images or videos can be attached to any object as a material in Unity. However, what sets this object apart is the aforementioned shader. The shader on the object creates visible “pixels”, just like one would see if he were to look up close to an a real LCD screen. To make each screen look more like a TV, some 3-D cubes were created and elongated and set with a glossy material to act as the trim around the screen.

Figure 19. LCD Screen object

The next step was to create and place all of the on-screen objects found on the multiple choice screens. Unity offers multiple user-interface (UI) objects, a few of which were perfect for this task. The first step was to attach an object called a canvas to the LCD screen object. A canvas is essentially an invisible 2-D base to any UI-based object. The canvas was resized so it fit the shape of the LCD screen. Now that the canvas was placed, the multiple choice images and text could be set up. Each question has 3 screens to transition through, the cover screen that shows the question number, the main question screen, and the correct/incorrect screen. An empty parent object was created for each screen. Then, a UI image object was place on each, and resized to fit the canvas. Each UI image was recolored to black. After each screen was established, the text objects could be placed. One of the key goals of creating these screens was to allow scripts to control what exactly is displayed on each screen. So, at the time of creation, the text objects just included some generic text describing what it will be used for. The cover screen just displays the question number. The main question screen displays the question text, the question timer, and the answer choices. The answer choices use UI button objects for the visual effects, but the script will just change the text inside the button. The correct/incorrect screen just displays whether the player got the question right or wrong. The programming of this feature will be explained later in the report.

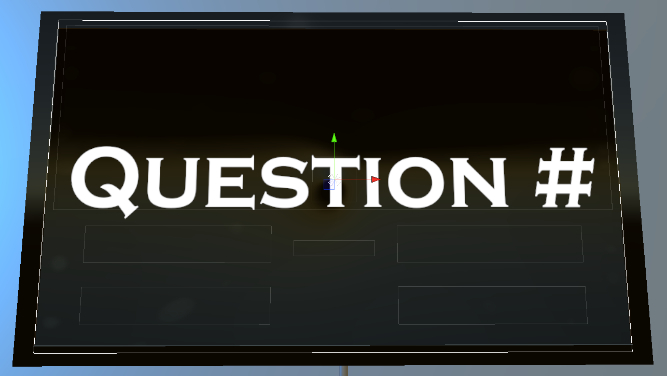
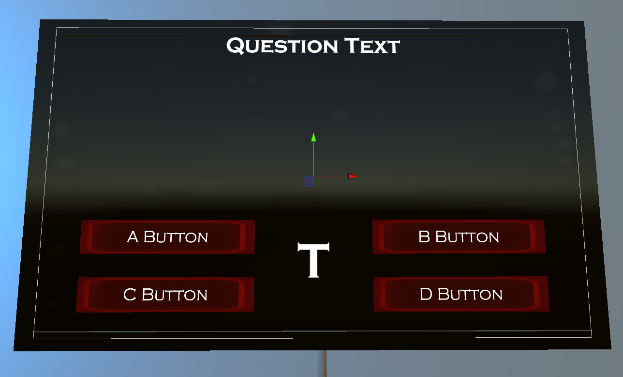
 



Figure 20. Cover screen, question screen, and correct screen.

Now that all of the display objects were created, solution for taking input from the players was needed. The original idea for this was making the question screen a touch screen. However, the screen would’ve had to be much smaller for the player to get close enough to touching it, while also being able to read it without having to move back from it. We found an asset pack in the store called “Simple Animated 3-D Buttons” that provided just what we needed. Just like the title of the pack states, it provided buttons in 2 different shapes and sizes. However, at first, the buttons only recognized mouse clicks since they were designated for non-VR, 3-D games. After looking into the objects built-in code, all that was needed was changing the *OnClick()* function to the *OnTriggerEnter()* function. Now, when the player comes in contact with the buttons collider, it will run everything inside the function, rather than having to click on it. After getting the buttons to function correctly, they were placed on a table (provided in the SFSMP). One large square button was placed that will be used for starting the question, and then 4 smaller, round

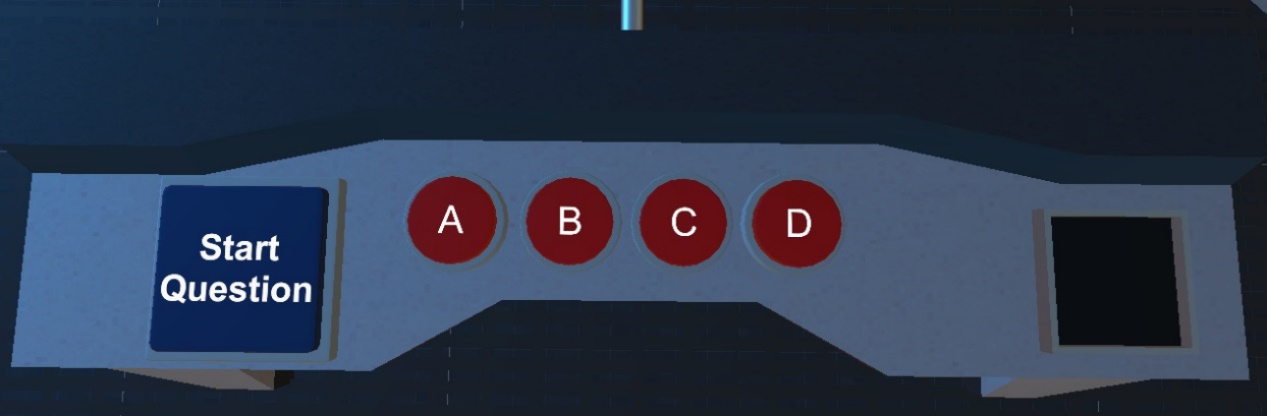
buttons were placed, one for each answer choice. Like stated earlier, the programming portion of this feature will be descibed later.



Figure 21. View of Multiple-Choice Question Buttons, and full screen object view

**Subtask 4. Keys**



Figure 22. Picture of some of the keys found in the game

In order to create a better sense of progression in the game, we decided to establish a set of items players need to retreive from completing objectives. After going through some of the asset packs already in the game, we found these artifacts in the SFSMP, and they were a great solution for the issue. A few objects had to be built into the scene in order to correctly implement this key system. The main function of the keys is to activate a designated multiple choice question. So, we had to create a way for players to know which key goes with which question, and a slot in which the players can place the key to activate the question.

There were multiple ways that we could have gone with displaying the correct keys for each question. For example, the original idea was just to place a picture of the correct key on the screen of each multiple choice question. However, since the game has a futuristic setting, we decided to create a more technologically advanced object to achieve this. Next to each multiple choice question screen, the player will find what appears to be a hologram version of a key. The key found inside that hologram is the key designated for that question. The hologram key has no collider attached to it, so it is not player-interactable. However, it does have a rigid body and script attached to it so some visual effects could be added to it. The script creates a constant rotational velocity and vertical sinusoidal movement pattern on the hologram, providing a “floaty” visual.

Figure 23. Hologram found next to Question 1

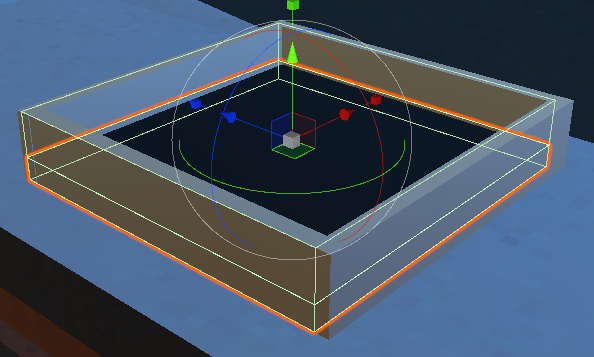
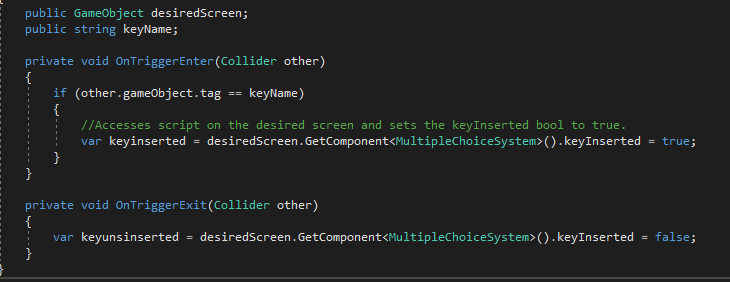
Next, we had to add a key slot to the multiple choice screen set up. These were built using 3-D cubes in the Unity editor. 4 rectangular blocks were oriented into a square shape large enough so that the base of each key will fit. Then, a cube was placed at the base of the slot to provide a flat surface for the key to sit on. This base cube was the most important piece because it is the object the script will be attached to to detect if the correct key is inserted. Before the script was written, an extra box collider was attached to the base cube. This collider was moved above the base, and resized to about the shape of the inside of the key slot. It was then set as a trigger, so it will not actually collide with anything, however, it will track if something is inside of it. Now that the trigger was set, a script could be written, checking for an object with the correct tag to enter the trigger collider. If all conditions are met, a bool value in the multiple choice script is set to true. That script will be discussed in the multiple choice feature section later. Screenshots of the key slot with highlighted colliders and the code for the key slot are shown below.

Figure 24. Multiple Choice Key Slot and Associated Code

**Subtask 5. Circuit Components**

One of the most important topics in electrical engineering education is circuit design. Understanding how to correctly build circuits under certain conditions is essentially the practical application of everything learned in school. It is because of this that we knew we had to put circuit design objectives in the game. In order create these objects, we obviously needed to build the circuit components. Since this game was designed mostly as reinforcement to early electrical engineering education, the objectives were mostly centered around basic DC circuits. Therefore, the objects needed were batteries, resistors, and wiring.

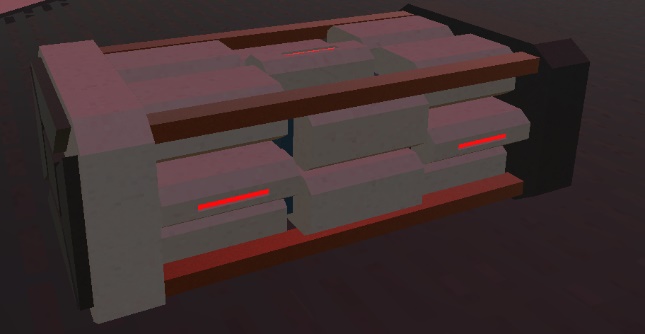
Thanks to the SFSMP, finding a good battery model was rather simple. The pack contains 3 different batteries: small, medium, and large. However, the default size of all of the models were quite large. We ended up choosing the medium battery, but scaled it down to x = 0.1, and y = z = 0.15. Now, we had a solid battery model that made sense in terms of size and fit the scene stylistically.

Figure 25. Battery used for circuit design objectives

Next, we had to make resistors. The resistor models were made completely from scratch using objects and materials in the Unity editor. The resistors can be broken down into 3 parts, the main resistor capsule, the wires, and the colored bands (all of which were placed in a parent object). The main piece of the resistor was created using a capsule object in the Unity 3-D editor. It was resized to x = z = 0.12, and y = 0.2, leaving to be much larger than a realistic resistor, but it would have been impractical and hard to work with if it were that small. Next, the wires on each end were created using a cylinder object, and resized to x = z = 0.03, and y = 0.06. Since they had to protrude down on both ends of the resistor capsule, they were moved to position x = 0.15 and -0.15, and both at y = 0.93. The final part of the resistors were the colored bands. The resistors are 4 band resistors, so the bands represent the first digit, the second digit, the multiplier, and the tolerance of the resistor. They were created by altering 3-D sphere scale values into numbers that would essentially create flat disk-shaped objects. Those scale values were x = 0.02, y = z = 0.125. Then, each band was placed in realistic positions on the resistor capsule. All of which were placed at y = 1, but then spread out along the x-axis at -0.109, -0.078, - 0.046, and 0.096. Now that all pieces of the resistor were placed, materials could be assigned to each. A glossy, tan colored material was placed on the capsule to mock a real resistors color. A silver metallic material was attached to the wires on each end. For the bands, a glowing, emissive material was made for each possible band color. These materials could be placed as needed on each band depending the desired resistance.

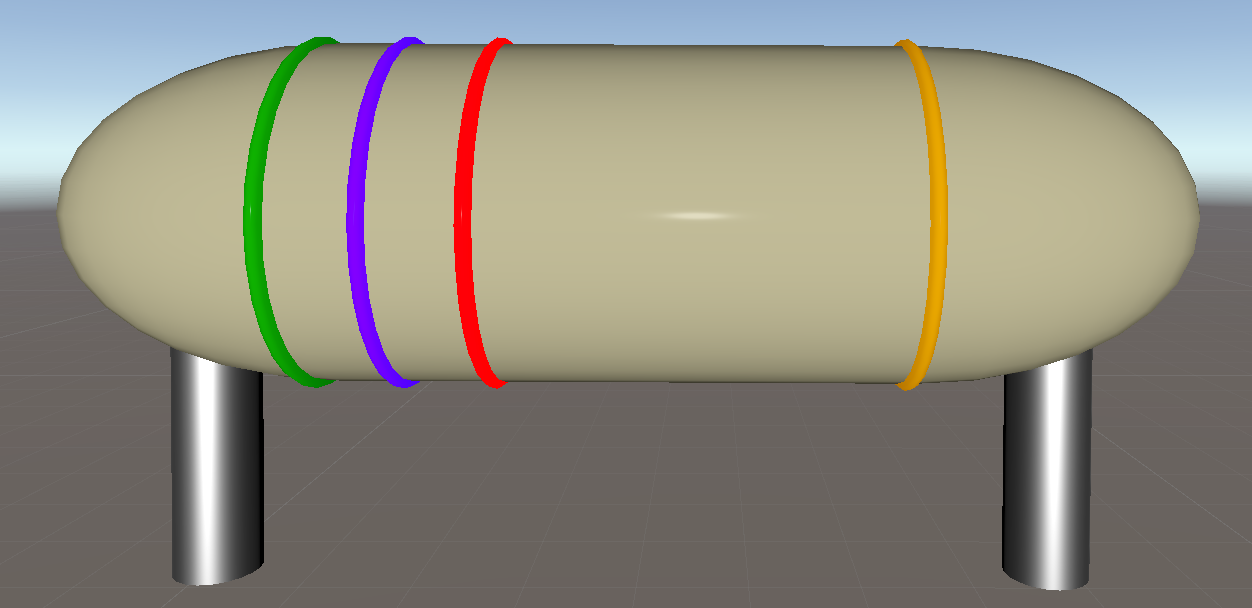
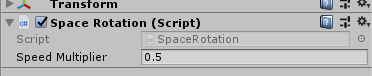


Figure 26. A 5700 Ohm Resistor

**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. 

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.

**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.

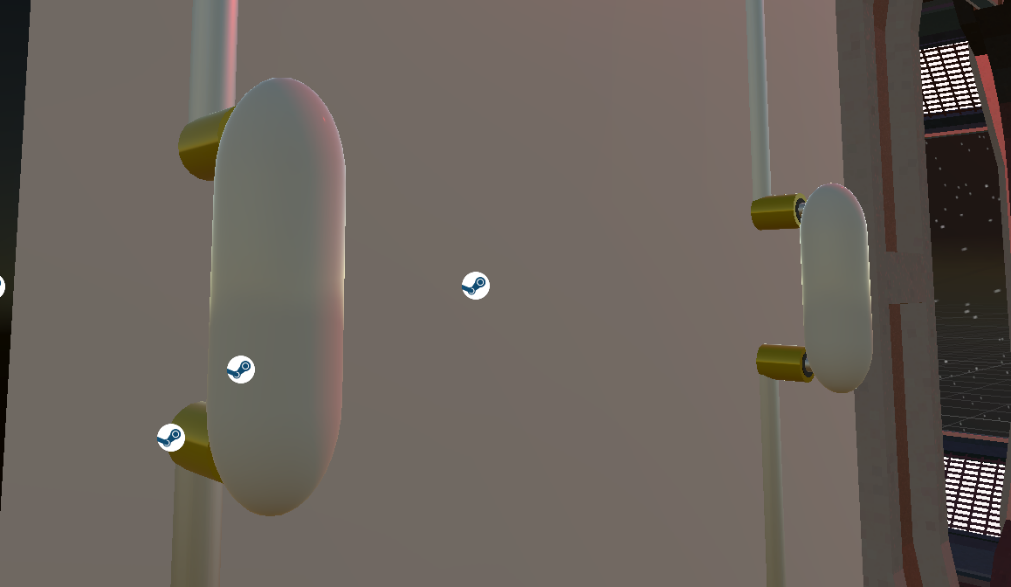
The first step of the game is to complete the first multiple choice question by inserting the key found on the table in the center of the room. The key system was a more creative approach to progression throughout the game, rather than just telling the player to go from objective to objective. This was 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.

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.



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.

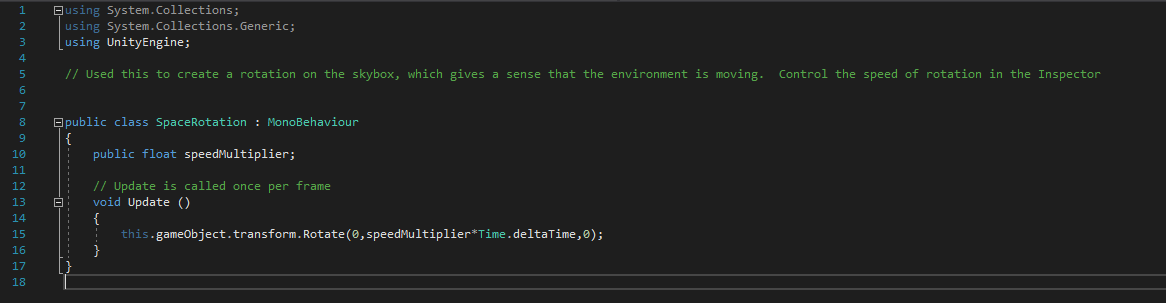
In terms of the technical information I have gained, I learned just about everything about C# programming that I know from this project. This is providing me with a great foundation in object-based programming that can be used in many programming professions. I’ve gained more from this project so far 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

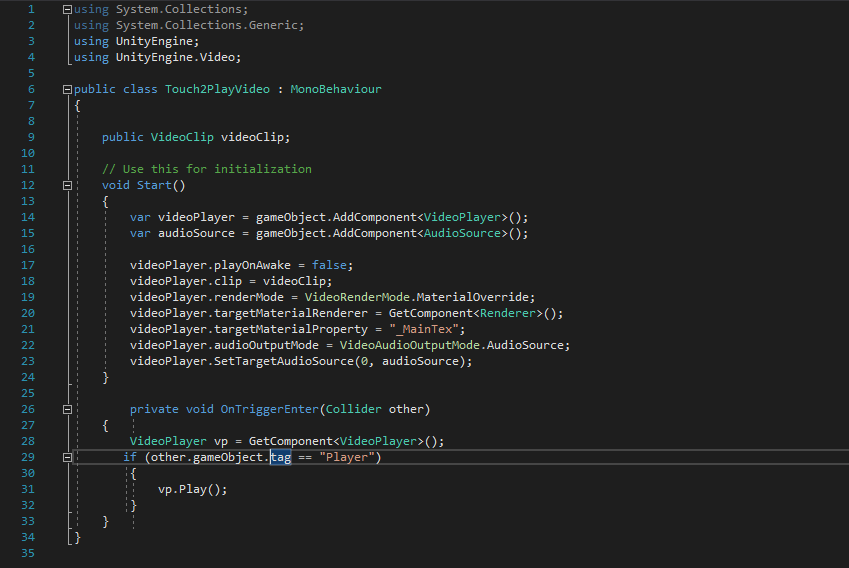
Joe McGinley - one of my best friends, and a brilliant programmer who’s worked with Unity and helped me along with the way.

Appendix

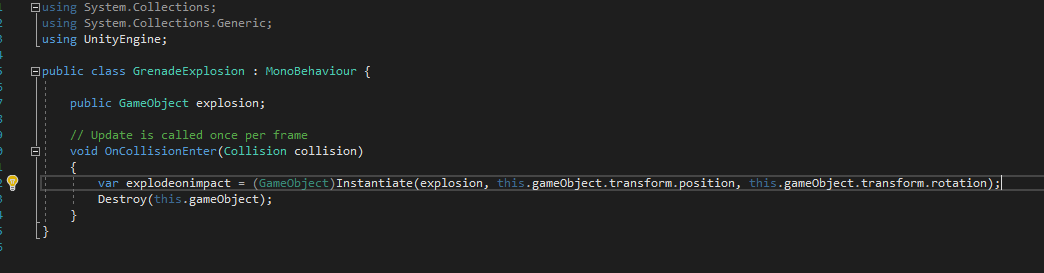
1. Source Code. Script for Rotating Skybox



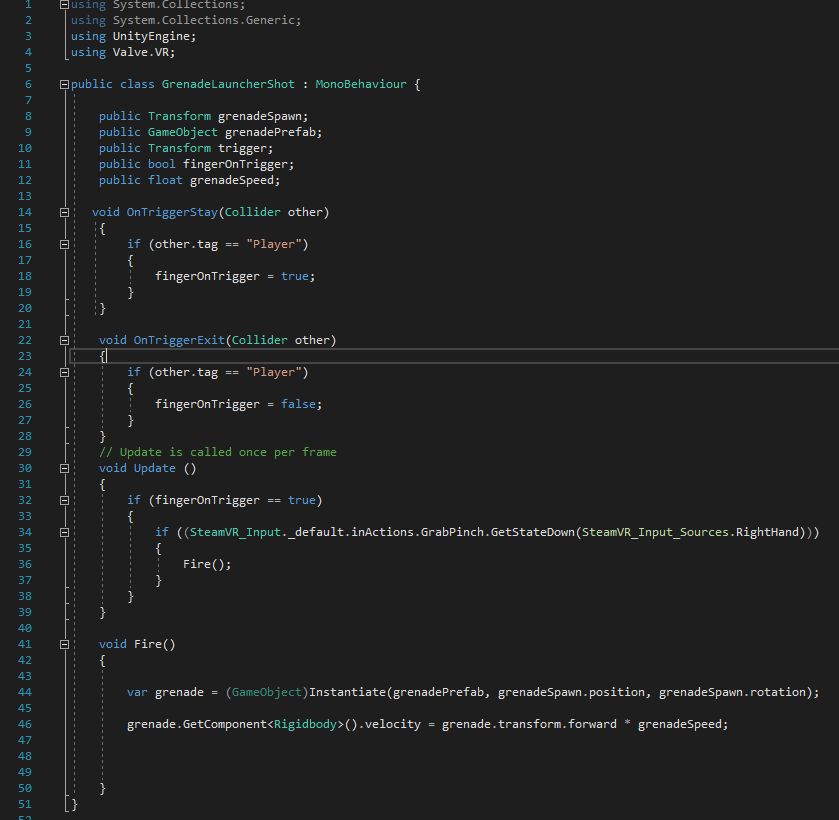
1. Script for Touch-Activated LCD Screen



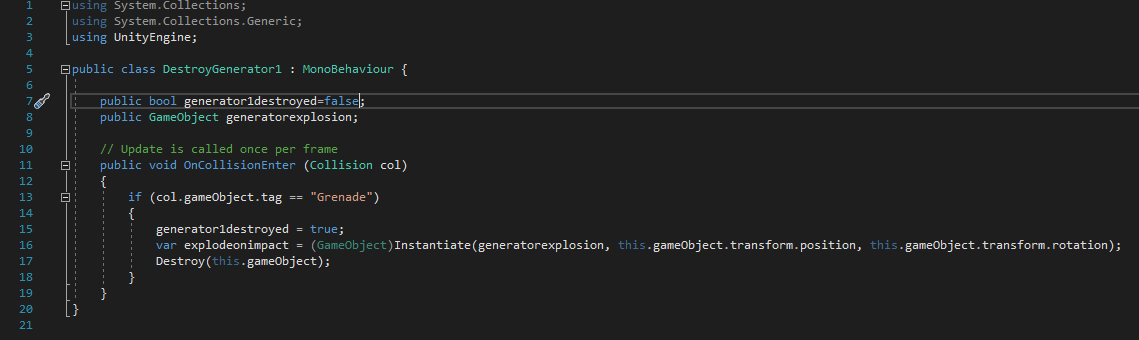
1. Script for Exploding Rounds



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

[2] Haberstroh, T. (2017, February 8). Game-Based Learning. Retrieved April 10, 2019, from http://www.espn.com/nba/story/\_/page/presents5/washington-wizards-virtual-reality