6-14-2018

Virtual Reality Sherlock: A Crime Scene Reconstructor

Ellen Tseng
Santa Clara University, etseng@scu.edu

Ken Wakaba
Santa Clara University, kwakaba@scu.edu

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 13, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Ellen Tseng
Ken Wakaba

ENTITLED

Virtual Reality Sherlock: A Crime Scene Reconstructor

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

[illegible]
Thesis Advisor

[illegible]
Department Chair

Virtual Reality Sherlock: A Crime Scene Reconstructor

by

Ellen Tseng
Ken Wakaba

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 14, 2018

Virtual Reality Sherlock: A Crime Scene Reconstructor

Ellen Tseng
Ken Wakaba

Department of Computer Engineering
Santa Clara University
June 14, 2018

ABSTRACT

When an investigation team arrives to the scene, they only have a limited amount of time to gather as much evidence as they can. Evidence can include, but is not limited to: fingerprints, pictures/videos, blood samples, or any other biological evidence. Due to the limited amount of time, a few risks arise; they may not have collected enough evidence, the evidence itself may not have captured the full scope of the scene, and the possibility that the evidence itself may have been damaged or destroyed. Our aim is to develop a low-cost, customizable VR crime scene reconstructor. This software allows CSI as well as the court to revisit a crime scene by inputting only the necessary components of the crime in question based on previously collected data and witness accounts. Rather than using expensive cameras to capture an overly-realistic scene, a solution that is not computationally expensive is required because of not only the amount time it takes to render the setting, but also the requirement of high-end hardware to process the data. We propose a reconstructor that allows the user to construct the scene piece by piece, which lets the user understand the details individually rather than as a whole picture.We believe our VR simulator will be helpful not only in training CSI investigators but also in the courtroom by allowing juries to concentrate on the most pertinent details of a scene.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

A crime scene is defined as an area where a crime has been committed and physical evidence pertaining to the crime can be collected [1]. Crimes include, but are not limited to, homicide, car accidents, theft, and assault. When an investigation team arrives to the scene, they only have a limited amount of time to gather as much evidence as they can. The type of evidence gathered from a crime scene depends on the type of crime; a murder may have blood samples to gather, whereas in a burglary, fingerprints may be left behind. Collected evidence entails pictures and videos taken at the scene of the crime, as well as any biological evidence obtained. Though the team may be thorough in their collection of evidence, some evidence may be neglected; the pictures themselves may not capture the full scope of the scene, and the possibility exists of biological evidence being destroyed or damaged due to contamination or negligence in the care of the evidence. Therefore, juries only have a limited view of a scene when the pictures are shown to them, making it ever more difficult for them to come to a verdict.

1.2 Related Work

3D laser scanners are already on the market and survey the environment, saving the information collected from points of references into memory. With multiple scanners combined, a larger recreation of a scene can be stitched together. Unfortunately, scanners typically cost thousands of dollars, though smaller, cheaper ones are becoming available. However, the range in scanning area is not as great and so it can still cost a lot of money to scan a large area. Although helpful, the current technology can also be difficult to use. If the investigation team chooses to pursue the 3D modeling route, they still must either hire an expert to scan the scene, or attempt to figure out the technology themselves [2]. The software, although powerful, can come with a large learning curve, which could be expensive both in time and money to train others to use. Additionally, there is very little interaction with the results from these scans. Much like Google Earth, users are able to view the scene in 3D but they cannot interact with any of the objects, such as picking up certain objects.

1.3 Objective

Our aim is to develop a low-cost, customizable VR crime scene reconstructor. This software allows CSI (Crime Scene investigation) teams as well as the court to revisit a crime scene by inputting only the necessary components of the crime in question based on previously collected data and witness accounts. Rather than using expensive cameras to capture an overly-realistic scene, a solution that is not computationally expensive is required because of not only the amount time it takes to render the setting, but also the requirement of high-end hardware to process the data. We propose a reconstructor that allows the user to construct the scene piece by piece, which lets the user understand the details individually rather than as a whole picture. Though simplistic, the scene contains less noise and has less materials to process. With the added benefit of more efficient training, it can also remove any sort of bias people, especially the jury, may have. From there, we can assign actions to human and car models to perform a playback of the crime based on the information provided. Our solution provides a full visual of the crime view in different perspectives.We believe our VR simulator will be helpful not only in training CSI investigators but also in the courtroom by allowing juries to concentrate on the most pertinent details of a scene.

Chapter 2

Requirements

We have established three types of requirements to make sure that our system works and meets our criteria: functional requirements, which describe how our system will work; non-functional requirements, which describe how a system should behave; and design constraints, which provide limits on how our system operates. Each requirement has a level of priority; Critical indicates that our system must include that requirement, and Suggested indicates that we would like to include the requirement into our system but it is not mandatory.

2.1 Functional

The system will allow:

- users to edit the scene and the components that go along with it (critical)
- users to build the scene piece by piece (critical)
- scenes to be saved for future editing (recommended)
- users to input new data collected from the field such as photos (suggested)
- the viewing of different scenes (suggested)
- for multiple users, each with a VR headset, to view a scene at the same time (suggested)
- the ability to power save in case of power failure (suggested)
- allow for the connection of multiple Head-Mounted Displays (HMD) (suggested)
- allow users to scale objects within the scene (suggested)

2.2 Non-Functional

The system will:

- operate smoothly to prevent users from experiencing motion sickness from the frame per second delay (critical)
- allow scenes to be displayed in a clear manner (critical)

- be low cost (critical)
- have an intuitive user interface (recommended)

2.3 Design Constraints

- The system must be able to run on the Oculus Rift

Chapter 3

Use Cases

Within this chapter, we will list the use cases of Sherlock. Each use case will describe how an actor, in this situation a user and the audience, will interact with our software and the actions/events that may take place [3].

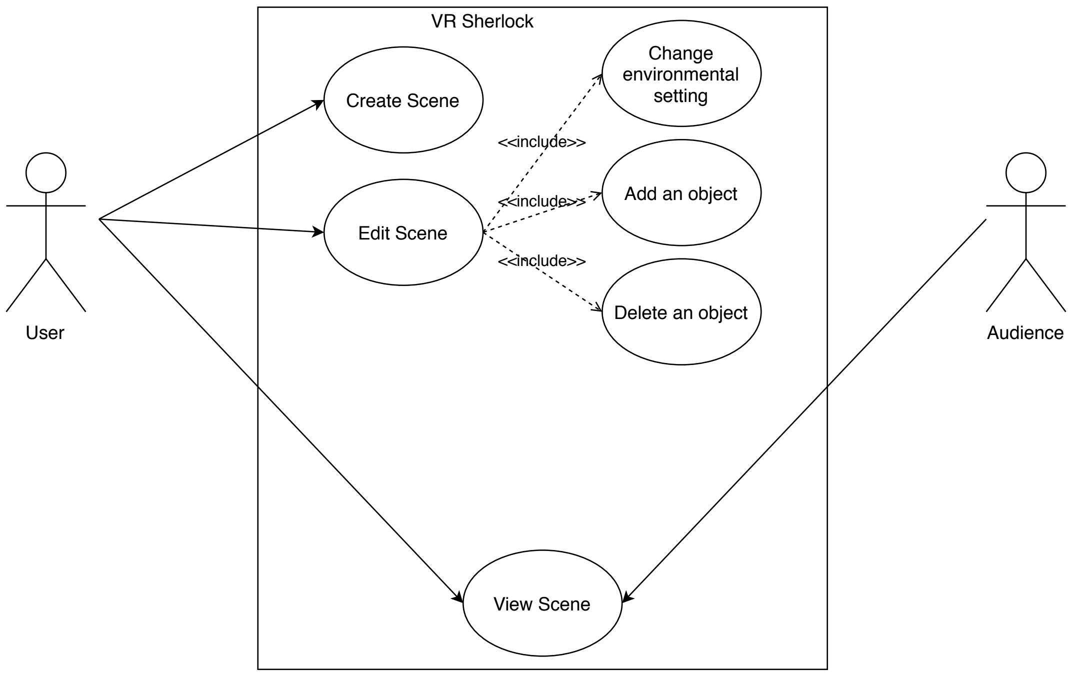


Figure 3.1: Use case diagram for VR Sherlock: A Crime Scene Reconstructor

3.1 View Scene

Goal: The user will be able to view a crime scene that has been previously created

Actor: User, Audience

Pre-Conditions: System must be on and user must have selected the option to start and select the option to view

Post-Conditions: User in a crime scene environment

Steps:

1. User turns on Crime Scene Simulator
2. User selects option to view crime scene
3. User selects which scene to view
4. Audience views from other HMDs or on a screen

Exceptions: No crime scenes have been previously created and saved

3.2 Edit Scene

Goal: The user will be able to enter the system and edit a previously created crime scene

Actor: User

Pre-Conditions: System must be on and a crime scene must have already been created

Post-Conditions: User has successfully edited certain components of the crime scene

Steps:

1. User turns on Crime Scene Simulator
2. User selects option to edit crime scene
3. User selects the scene in which they want to edit
4. User selects from multiple options what they want to add/remove/edit in the active scene
   (a) User has the option to change the environmental setting of the crime scene
   (b) User can open up the inventory menu and add more objects in
   (c) User can select an object already in scene and delete it
   (d) User can select an object already in scene and edit it if possible

Exceptions: No crime scenes have been previously created and saved

3.3 Create Scene

Goal: The user will be able to enter the system and create a crime scene

Actor: User

Pre-Conditions: System must be on and crime scene components must have been previously created

Post-Conditions: User has successfully created a crime scene

Steps:

1. User turns on Crime Scene Simulator
2. User selects option to create crime scene
3. User walks through steps to select different aspects such as time of day and environment object

Exceptions: No objects have been created for the user to user

Chapter 4

Activity Diagram

Figure 2 shows the activity diagram for the user. When the user starts up the VR program, he or she has the options to create a scene, edit a scene, or view a scene if available.

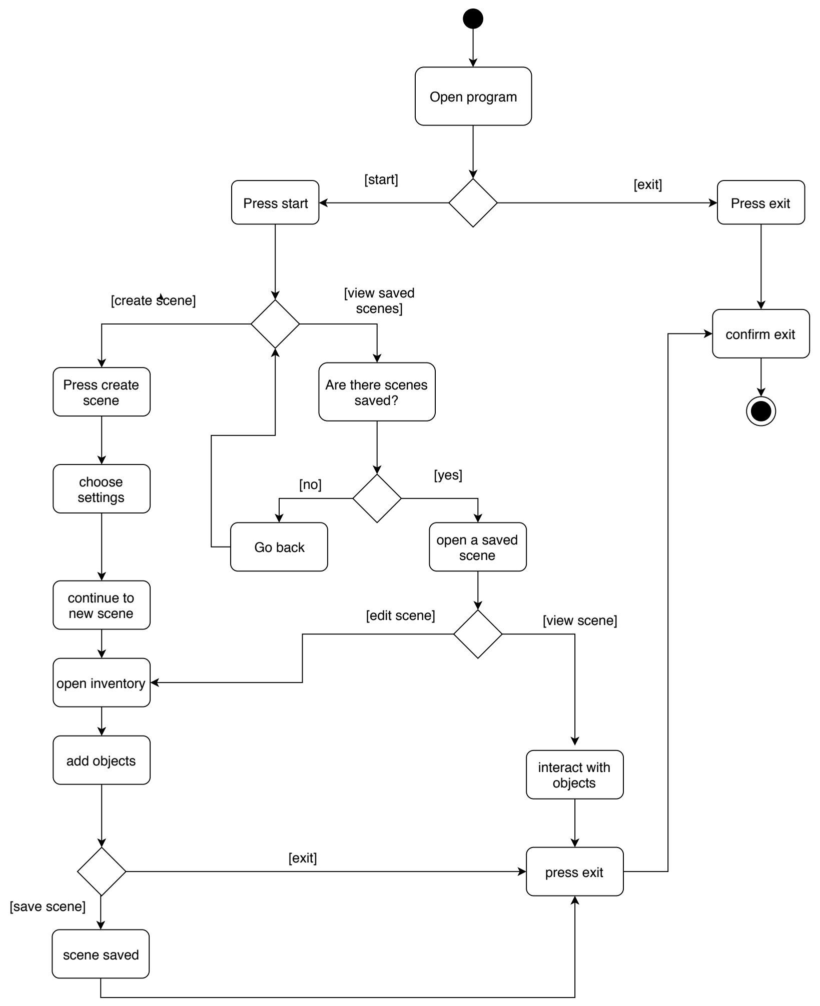


Figure 4.1: Activity diagram for VR Sherlock: A Crime Scene Reconstructor

Chapter 5

Technologies Used

Hardware

- Oculus Rift Consumer Version (CV): The Oculus Rift CV comes with an HMD that provides 3D-like imaging and audio, Constellation, its position tracking sensor, and motion controllers to operate the system [4].

Software

- Unity: Unity is a game engine that supports 3D graphics and scripting with C# which will be useful for providing models with actions. Unity also provides texturing and lighting tools that will enhance the simulated environment [5].
- Autodesk Maya: Maya is a 3D modeling software that was used to model many of the objects in Sherlock. What set it apart was that it includes a human rig which was useful in our development [6].
- Blender: Blender is a free 3D modeling software used to model many of the objects in addition to Autodesk Maya [7].
- Adobe Photoshop: Photoshop is an image editing software that was used for the texturing and coloring of the objects modeled in Autodesk Maya and Blender [8].

Chapter 6

Architectural Diagram

Figure 6.1 highlights the architectural design we chose. We deemed it best to use an Event Based architecture due to the way our system will function [9]. The system will wait and listen for user input such as head movement of the VR headset or input from the controllers. Once they system has received the input, it will then respond with an action within the system such as processing an image/text or object manipulation.

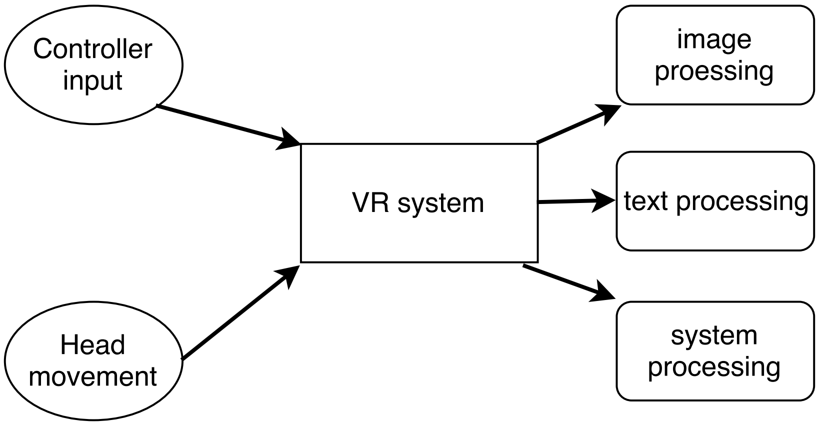


Figure 6.1: Event Based Architecture for Sherlock

Chapter 7

Design Rationale

7.1 Technology

We chose to use the Oculus Rift as opposed to the HTC Vive [10] for various reasons. The Rift offers prolonged use due to its level of comfort. With the padding provided around the face, adjustable straps, and a lighter, smaller profile, it offers a higher level of comfort when compared to the Vive. The Rift is also more intuitive to set up and requires less time. With the Vive, the sensors must be mounted at a specific height and are not allowed to move because it will mess up the calibration; however, the Rift sensors can simply be placed on a table and plugged into the system it is running on, making it more user friendly. In addition, the Rift controllers provide a hand-like presence with its gesture controls versus the feel of gripping a long object like the HTC wand. With the added benefit of the SCU Imaginarium housing many units of Oculus Rifts, acquiring a VR headset was fairly simple.

The Unity Game Engine is a top choice when it comes to VR development on the Oculus Rift. The game engine is focused on simplifying the development of a new product or game with features such as the Unity Editor. Virtually everything within the editor is visual, meaning you have the ability to select, build, and modify your product with minimal effort. It also allows users to test their projects within the editor so they may preview the look the target audience will receive. Unity also allows the benefit of scripting in either JavaScript, C#, and Boo. We chose to build our design in C# because it allows for more flexibility and faster iterations. Once again because the SCU Imaginarium is dedicated to VR development, the systems within the lab all have Unity licences for students to use, so there is no need to buy the licences.

Our basic product will allow users to select objects to create their own crime scenes. To enable us to create these objects, we chose to use a few 3D modeling softwares as well as image editors. Both Maya and Blender are 3D modeling softwares but each offer their own benefits. When you are creating 3D objects, you first create the objects texturing then you apply it onto the object model. These textures can be downloaded off the web or can be created from scratch. We utilize Photoshop for texturing because it allows us to both edit any texturing we may have found from the web and create our own. With the textures created, we then apply them onto the object model. Maya has a built in human rig, which is essentially a full body skeleton that allows full control over any of its joints and movements. With this added benefit, applying the textures onto the rig is made easy with Mayas user interface. We also use Blender for 3D modeling because it is more portable and less clustered, allowing intuitive use.

7.2 User Interface

The starting appearance of the software is based off of the holodeck from Star Trek [11]. Our idea is to have the user enter a blank virtual world that slowly becomes a ”solid” environment. Rather than having static images produced by 3D laser scanners as it is with some crime scene reconstructors, we have the user add objects included in the software (e.g. tables, chairs, knives, shattered glass, etc.)that we believe would be relevant to the crime scenes. We want our software to be interactive and for the user to get all the details of an object. In a static scene, only one side of every object is seen. With our design, we are giving the user the flexibility to interact with the scene with activities such as picking up a piece of evidence and rotating it around.

We want the software to be intuitive and user-friendly. Upon starting, the user is shown a screen in the center of their field of vision. Because the user has to look at a button before selecting it, we designed the menu screens in a way that would only require limited head movements.

When a user creates a new scene or edits a scene, he or she is shown a screen that allows them to select the environment setting of the scene. The five options provided (Day, Night, Cloudy, Raining, Snowing) determine the lighting for the scene and provides the effect of being in a solid place rather than a virtual world. When it comes to adding objects from the inventory provided, we have the user adjust the size and appearance so that it can fit the real crime scene as closely as possible.

Chapter 8

Description of System Implementation

This section is dedicated to describing the functionality delivered in the final iteration of this product compared to the requirements as well as how the product was developed.

8.1 Functional requirements

The system will allow:

- users to build the scene piece by piece

  Our system provides an inventory menu for users to add objects into an environment. Users can grab an object and place it wherever they wish as well as manipulate its rotation.

- users to edit the scene and the components that go along with it

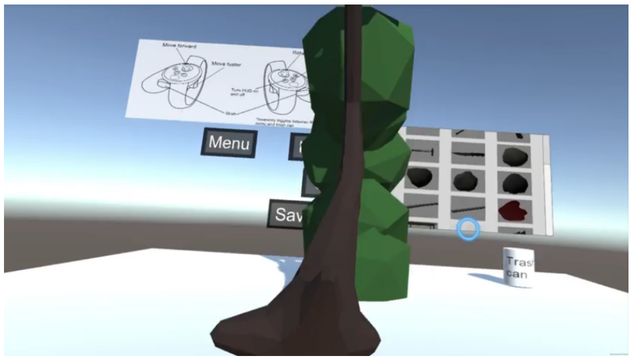


Figure 8.1: Inventory menu with object spawned

- scenes to be saved for future editing

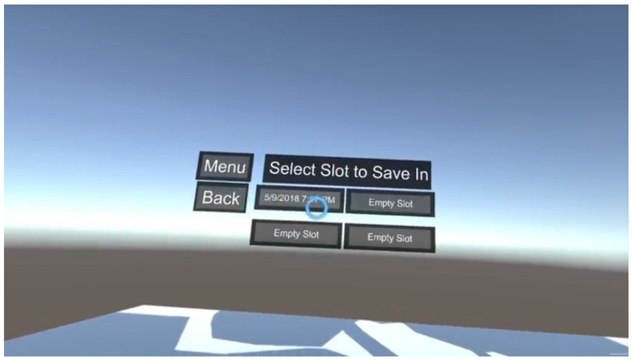


Figure 8.2: Save menu with date and time confirmation

- the viewing of different scenes

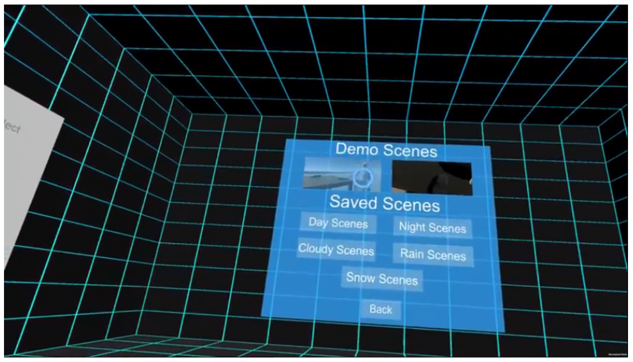


Figure 8.3: Scenes available for viewing

8.2 Non-functional requirements

The system will:

- operate smoothly to prevent users from experiencing motion sickness
- allow scenes to be displayed in a clear manner

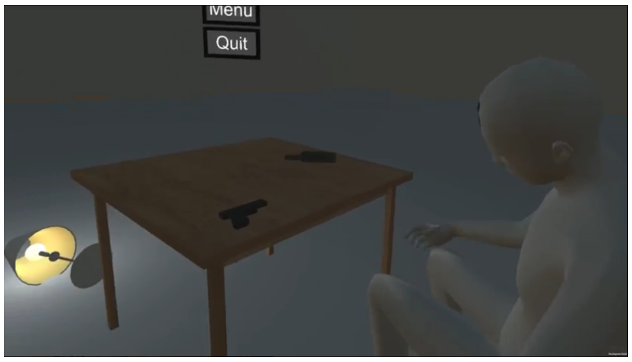


Figure 8.4: A saved scene in sherlock

- be low cost

  To use VR Sherlock, the user only needs and oculus rift and a computer.
- have an intuitive user interface

8.3 Implementation

With our goal and requirements outlined, we needed to begin building our product. This process involved enrolling in the Virtual Reality Bootcamp courses. With the aid of this course, we created a user backlog listing our objectives.

First we compiled a list of items that would be in a crime scene, such as human bodies in various poses, knives, guns, blunt objects, and blood. The items were modeled and textured in both Blender and Maya. They were then exported into Unity where we adjusted the textures and size of each model. To make our project work for the Oculus Rift, we imported Oculus's Virtual Reality utilities and development kits into Unity. With these libraries, we were able to set up an environment in which the user can move around with only the thumbsticks and use the right controller to point and select.

Next came actually building the simulation so that the objects imported could be used. The environments created were built in scene files that contained the data menus needed and the coded that allowed the objects to be generated. With a menu selection script, the user is able to switch from scene to scene through the menu on the head up display. While the user can build his or her own crime scene, we included two demo scenes as well for training purposes.

Chapter 9

Test Plan

9.1 Unit Testing

Unit testing is the testing of individual components of our project to make sure that they could work independently of each other [12]. When building Sherlock, we used a bottom-up method, testing as we went. We would build separate scene files, one for player control and grabbing of primitive objects, one for defining the size of the models we imported, and several scenes for the menus and buttons.

9.2 Integration Testing

We would then integrate the scenes together, first with having the player grab the imported models to see if they were the proper size and had the right bounding box for grabbing. We would then script the menu scene to lead to this grabbing scene. Once all the user interface menus and controls were tested in their respective scenes and we ensured that the communication between all components was smooth, we would then have Unity built the project and create an executable file [13].

9.3 Acceptance Testing

For acceptance testing [14], while we were in the Virtual Reality Bootcamp class, we would have our peers and professor test our executable file. We would also bring in people from outside the VR lab, especially those unfamiliar with the device to test our project. Through their verbal and usage feedback, we would make adjustments to Sherlock so that it would be more user friendly. This feedback is what inspired our tutorial scene.

Chapter 10

Societal Issues

10.1 Ethical

Due to the fact that we were the ones defining the specifications of our system, we had to consider the ethical implications that may come about during the development of our project.

10.1.1 Group Ethics

We examined the Association for Computing Machinerys Code of Ethics and the IEEE Code of Ethics during the production of our project [15]. The code of ethics describe a set of principles in which all engineers should follow when developing a product for the public. These rules, such as "Be honest and trustworthy" and "Give proper credit for intellectual property" helped our group avoid any ethical issues that may arise. By following the set of principles provided, we were able to work alongside each other with utmost trust and acknowledging and honoring each others achievements. The Software Code of Ethics provides insight on how ethics can be applied to software developers [16]. Using these set of ethics, we ensured that Sherlock was developed with the publics' interest in mind and ensuring that it met professional standards.

10.1.2 Project Ethics

The biggest ethical concerns faced when developing Sherlock was the issue of our system being used by younger age groups. One of our future goals is to place Sherlock in the Oculus Asset Store for free to allow the public to use. The age group of VR users ranges from children to adult professionals with the majority in the younger generations. Because our project is a crime scene reconstructor, it presents the possibility of scenes being created that may not be suitable for young kids. It also raises the concern that our project may create a sense of violence in children by presenting objects such as weapons during the VR experience. We address this issue by making reducing the number of objects that may be deemed as graphic and violent. Although our project is intended to be used by adults who work in the crime field, we were sure to take into account all the possible users that may come across our project. Another ethical issue that was considered was the concern that people could "reverse engineer" these animations, meaning there's a possibility that one can plan how to commit a crime based on the outcome they want. Although there is no sure way of preventing this from happening, we made sure that the functionality of Sherlock aids in solving crimes, not facilitating and planning them.

10.2 Social

Sherlock was ultimately designed to aid not just the individuals in a crime scene unit, but each and every individual in a community. With Sherlock giving CSI teams a better understanding of forensic investigation and more practice with forensic analysis, the rate at which crimes will be solved should increase. With more cases being closed, crime rates will drop as incarcerations go up, leading to a safer society to live in.

10.3 Political

We hope for the political impact of our project upon the society to be a positive one. We would like for our software to be used in court as evidence as well as to help with testimonies. There have been numerous occasions in which the evidence collected by CSI teams has led to wrongful convictions and imprisonment. This is a result of poor training of the forensic teams and problems with the method in which they analyze the scenes and evidence themselves [17]. With the help of our software, police departments can properly train their forensic teams how to better interpret evidence, making it more concrete when used in court. This could also have the potential to reform forensic techniques.

10.4 Economic

There was no concern of money during the course of the project development. Most of our development was done in the Unity Game Engine which can be downloaded for free from the Unity website. In addition, thanks to being granted access to the Santa Clara University Imaginarium, all the resources needed to develop our software were provided to us; the Unity Game Engine was downloaded on all computers along with Autodesk Maya, Blender, and Microsoft Visual Studio. Each computer also had its own Oculus Rift configure to it.

If we were to develop our software on our own, the main cost of concern would be the Oculus Rift itself for testing and ensuring our software runs in a VR environment, as well as the Autodesk Maya software which is a paid subscription, much like Adobe Photoshop.

10.5 Health and Safety

There are quite a few health and safety concerns associated with our Senior Design project. When booting up the headset, you are always presented with a safety warning about using the VR system. While wearing the headset, your vision of the outside world is completely taken away, making you blind to the environment around you. This leads to potential accidents such as tripping and falling over objects or hitting individual around you when using the system. When setting up the play area, always ensure that you have given yourself enough room to move around and that all of your surroundings have been cleared away.

Another health concern that is presented by not only our software, but VR systems in general, is the risk of eye damage and motion sickness. As with staring at electronic screen such as tablets and phones, users have the potential to develop myopia or what is commonly referred to as nearsightedness. Along with the risk of myopia comes eyes stain, headaches, and for some users, nausea; “In real life, our eyes naturally converge and focus on a point in space, and our brain is so used to this that it’s coupled the two responses together. Virtual reality separates those, confusing the brain” [18]. This risk can be minimized by making sure the user does not use the headset for too long. If at any point the experience is getting uncomfortable, remove the headset and stop looking at it.

10.6 Manufacturability

The production of our project was very cheap thanks to the resources that were provided to us by the SCU Imaginarium. Sherlock only requires an Oculus Rift headset and a computer/laptop with a strong enough graphics card that supports VR applications.

10.7 Sustainability

Our project will continue to be sustainable so long as we continue to modify and adapt the environment and objects based on user experience and needs. There always be a need for forensic teams meaning that there will always be a need for tools that will train said teams. As forensic analysis methods change, our software will change with it.

10.8 Environmental Impact

Our system does have an environmental impact because all our software is centralized within the headset and computer itself. With the ability to view a scene that has been either previously created or imported in, users have no need to visit the real life location in person. This removes the need of using transportation vehicles that could harm the environment as a means of travel. In the aspects of visualization, instructors no longer have to print out visuals of what standard scenes may look like or of what harmful objects might be. Instead, they can simply create a scene and either present the scene to users or have the users immerse themselves in the scene. This removes paper products out of the situation.

10.9 Usability

We developed our software with usability and accessibility in mind. Our intended audience/users for our software are CSI teams and team leaders who want to learn better methods of forensic analysis, as well as instruct others on those methods. Our user interface is simple and quite intuitive for users. We ensured that along each step of the way our users know how the system works by providing visual instructions as well as a tutorial to walk through our software step-by-step. The User Experience is also important. Our goal is for first time users to come into the VR environment and already have a sense of what to do and how to use the system.

10.10 Lifelong Learning

The biggest take away along our journey with Sherlock was a glimpse at how game and project development truly is in the industry. We began with a simple idea and constructed a time-line in which our idea would turn into a working product. We utilized the Scrum methodology, a growing agile project management methodology in the software industry, as a way to plan and organize our goals and tasks. Using two week sprints, we were able to communicate and coordinate efficiently to accomplish our goals, much as how professionals do in a full time position. Along with learning and incorporating this methodology, we have learned to adapt and tackle any challenges that may come along unexpectedly in development. Along with learning Unity development from scratch, our group was learning and utilizing C# for the first time. These two technologies were the largest contributors to the development of our software whenever we encountered errors or bugs in our software, it was our responsibility to read up and research the technologies so that we know how to address the issue and prevent it in the future. The tech industry is always incorporating new technologies as they gain popularity, causing others to become outdated. The ability to learn quickly and adapt in the event of change is crucial in the field we will become a part of in the future.

10.11 Compassion

Compassion is the awareness of another's suffering and the desire to alleviate the suffering. The goal for Sherlock is to release the suffering and stress police departments and CSI teams have when a case is presented to them. As stated earlier within the paper, one of the problems CSI teams encounter is the lack of evidence acquired or the tampering/destruction of said evidence. With Sherlock, teams can be better equipped with knowledge when entering a scene, being more efficient with their time and analysis. By doing so, both the team and the department are better equipped to tackle the case at hand so that it can be closed at a swifter rate. Our functionality of being able to create a scene makes it easier for instructors to communicate and provide a visual aid to their students to gain a better understanding of forensic analysis methods.

Chapter 11

Conclusion

Our group was able to create a virtual reality software that will allow people to interact with a crime scene in a way that they cannot in real life.

There were many obstacles that were faced along the way of developing our software but we were able to overcome them and learn along the way. One of the biggest obstacles encountered was our work environment. Thank to the generosity of Max Simms and the Santa Clara University Imaginarium, we were able to take advantage of working on our system in the VR lab. However, this presented the issue of having to work locally, meaning that we always had to travel and be in the VR lab itself to continue development. We would have to work around the class schedules since there were courses still being taught in the lab, which meant that most of the time we would work late in the night. Along with the fact that we has to work locally, all our work and data was saved locally onto the computers or on our external hard drives. This meant that we had to work on the same computers each time and if they were occupied, had to return at a later time. This also made integration and version control tedious because we always ran into the case where one version would work on one computer but fail to work on another.

Even with all of these obstacles we were able to produce a concrete solution for crime scene analysis and investigation. Our solution of allowing users to interact with the objects themselves rather than simply looking at a static image gives them a better understanding of what is occurring in the scene itself. By creating objects of our own for the users to utilize in the scene creation, we give them the freedom of focusing on specific details and instances. In a real crime scene, there are a lot of details thrown at investigators all at once and it is virtually impossible for them to capture or comprehend all that is going on. Our implementation allows users to focus on those details they missed, giving them a better understanding of the scene. This also adds the benefit of eliminating the need for expensive cameras to capture an overly-realistic instance of the crime scene.

As stated in Chapter 8, there were quite a few features in which we were able to complete and include into our system but for the future there is still more work to be done. The saving feature of the software is functional but loading still contains a few bugs that must be addressed. We will continue to implement and support additional objects as development moves forward. An idea we also want to incorporate is an Object Indicator feature which, when users hover over a certain object, will give a brief description of the object and its nature. Along with that, we would also like to include animations for ”What Could Have Happened” when users are viewing a scene to give them visual cues and a walk through of the scene. Once we are satisfied with our software and we have addressed all major issues, we would like to release for public use on the Oculus Store.

Overall, developing the software was an informative experience. Having no prior VR development experience, every step along the way was a learning experience. We were able to take advantage of the Unity Game Engine and its powerful yet intuitive UI to create our game environments and scenes. We also began learning C# which is the choice language when developing in Unity. Of course learning a language is an ongoing process but we were able to gain a great deal of knowledge while working with it. Even with the obstacles and constraints, we experienced first hand what game development, as well as project development, look like and we were able to work efficiently as a team to produce software in which we are proud of and we look forward to continuing development on our project.

Chapter 12

References

[1] USLegal, 'Crime Scene Law and Legal Definition'. [online] Available at: https://definitions.uslegal.com/c/crime-scene. [Accessed: 31- May- 2018].

[2] NCAVF, 'Forensic Crime Scene Reconstruction, Virtual Reality'. [online] Available at: http://www.ncavf.com/what-we-do/crime-scene-reconstruction. [Accessed: 28- Sep- 2017].

[3] Jacobson Ivar, Christerson Magnus, Jonsson Patrik, vergaard Gunnar, "Object-Oriented Software Engineering" - A Use Case Driven Approach, Addison-Wesley, 1992. [Online] Available at: https://en.wikipedia.org/wiki/Use_case#cite_ref-1 [Accessed: 11- June- 2018]

[4] Oculus VR, "The Oculus Rift, Oculus Touch, and VR Games at E3", June 11, 2015. [Online] Available at: https://www.oculus.com/blog/the-oculus-rift-oculus-touch-and-vr-games-at-e3/ [Accessed: 11- June- 2018]

[5] Unity, "Game Engines - How do they work?". [Online] Available at: https://unity3d.com/what-is-a-game-engine [Accessed: 11- June- 2018]

[6] Autodesk, "Maya features". [Online] Available at: https://www.autodesk.com/products/maya/overview [Accessed: 11- June- 2018]

[7] Blender, "About". [Online] Available at: https://www.blender.org/about/ [Accessed: 11- June- 2018]

[8] Adobe, "Adobe Photoshop CC". [Online] Available at: https://www.adobe.com/products/photoshop.html [Accessed: 11- June- 2018]

[9] Techtarget Network, "Event-driven architecture (EDA)". [Online] Avaliable at: https://searchmicroservices.techtarget.com/definition/ driven-architecture-EDA [Accessed: 11- June- 2018]

[10] Vive, "VIVE VR SYSTEM". [Online] Available at: https://www.vive.com/us/product/vive-virtual-reality-system/ [AccesseD: 11- June- 2018]

[11] Star Trek, "Holodeck". [Online] Available at: http://www.startrek.com/database_article/holodeck [Accessed: 11- June- 2018]

[12] D. Huizinga and A. Kolawa, Automated defect prevention: best practices in software management. Hoboken, NJ: Wiley-Interscience, 2007, pp. 75

[13] M. A. Ould and C. Unwin, Testing in software development. Cambridge: Published by Cambridge University Press on behalf of the British Computer Society, 1994.

[14] R. Black, Managing the testing process: practical tools and techniques for managing software and hardware testing. Indianapolis, MN: Wiley, 2009.

[15] ACM, 'ACM Code of Ethics and Professional Conduct'. [Online] Available at: https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct. [Accessed: 2- June- 2017].

[16] IEEE, 'Software Engineering Code of Ethics'. [Online] Available at: https://www.computer.org/web/education/code-of-ethics. [Accessed: 2- June- 2017].

[17] The Washington Post, 'Forensic science not as reliable as you may think'. [Online] Available at: http://www.pulitzer.org/files/finalist
[Accessed: 26- May- 2018].

[18] CNN, 'The very real health dangers of virtual reality'.[Online] Available at: https://www.cnn.com/2017/12/13/health/virtual-reality-vr-dangers-safety/index.html [Accessed: 26- May- 2018].

Appendices

Appendix A

User Manual

Below are the instructions on how to use Sherlock. It has been broken up into 3 main components: Instructions on how to run the Tutorial, how to Create a Scene, and how to Load a Scene

When Sherlock is initially loaded, the user will spawn in a cube like structure where they will be presented with the initial Main Menu in their front view.

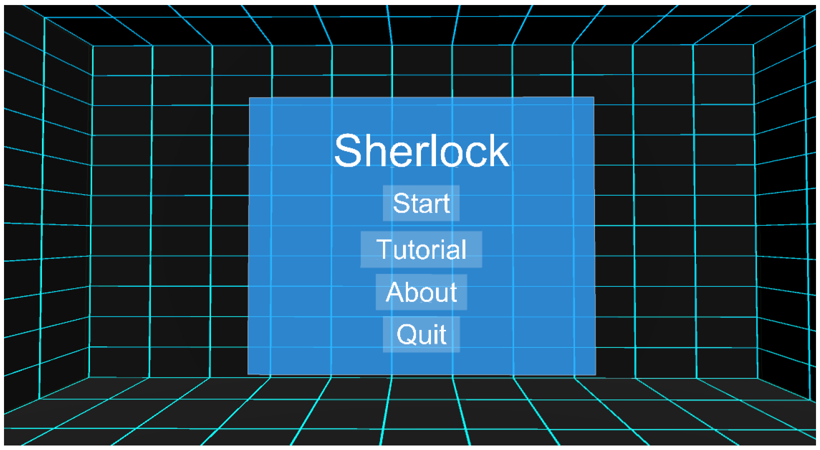


Figure A1: Start Menu

By turning their head and with it the Oculus Rift headset, they will see a panel which will display basic instructions and functions for the touch controllers; this includes how to grab an object, how to select objects, and how to move using the joy-cons. The Main menu will display 4 separate options in which the user can select.

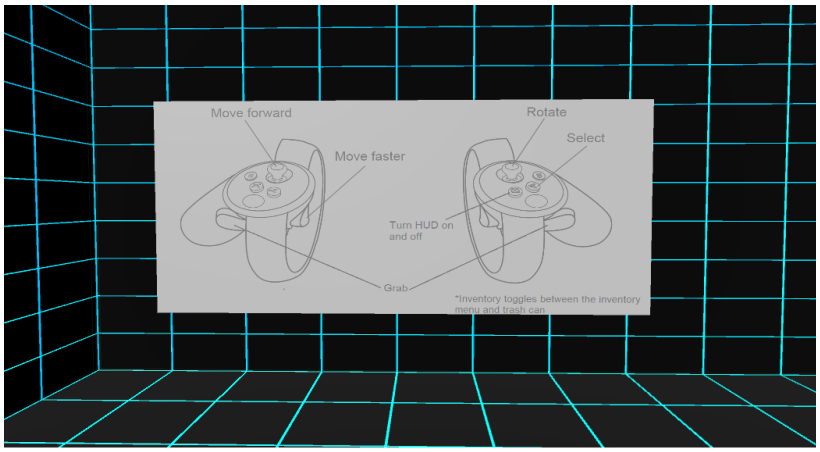
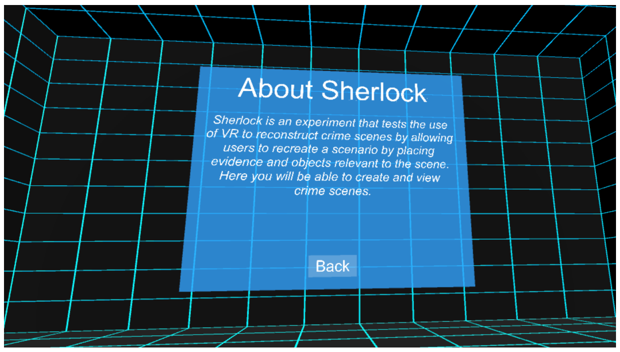


Figure A2: Controls Menu

Option 1 is to click the **Start** button, which will prompt another menu to appear displaying the options to **Create a Scene** or **Load a Scene**. This will be explained in more detail in subsections B.2 and B.3.

Option 2 will allow the user to select to run the **Tutorial**. The tutorial is meant for users who are not familiar with Sherlock and would like some more insight and practice with it. The Tutorial will be explained in detail in subsection B.2.

Option 3 will give the user a brief description of what Sherlock is and our main mission and intent was for creating it.



About Sherlock

Sherlock is an experiment that tests the use of VR to reconstruct crime scenes by allowing users to recreate a scenario by placing evidence and objects relevant to the scene. Here you will be able to create and view crime scenes.

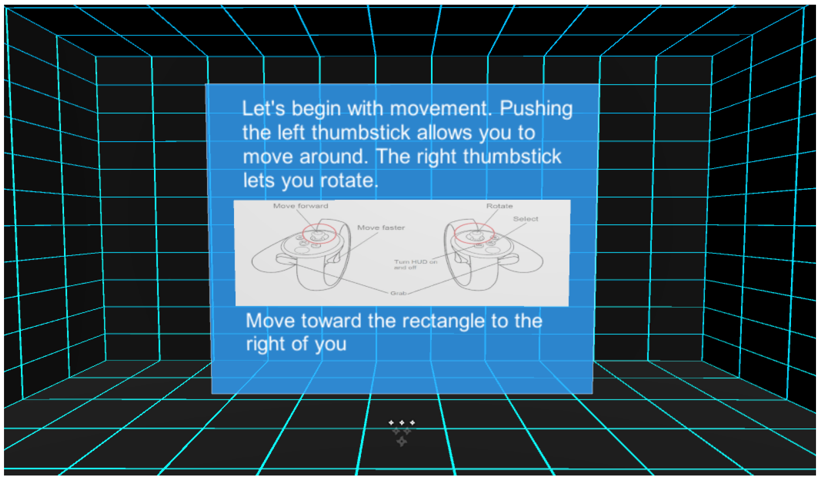
Figure A3: Description of Sherlock

The last option allows the user to select to quit the application when they are ready to do so.

A.1 Tutorial

Once you select the Tutorials Option, you will be prompted with the initial tutorial page where you will be reminded once more how to move around in the environment using the touch controllers. To proceed to the next part of the Tutorial, you will need to move over to the light blue box located in the right corner behind your avatar.

Once you proceed to the square, you will spawn in the next section of the tutorial where you will learn how to not only move faster in your given environment, but also learn how to grab an object as seen in Figure 11. As you spawn in, a table will fall from above and land right in front of your avatar. Walk forward and pick up the table with either one of your hands, and walk over to the cube over to the right and drop the table on top of the cube.


Let's begin with movement. Pushing the left thumbstick allows you to move around. The right thumbstick lets you rotate.

Move toward the rectangle to the right of you

Figure A4: Introduction to Tutorial

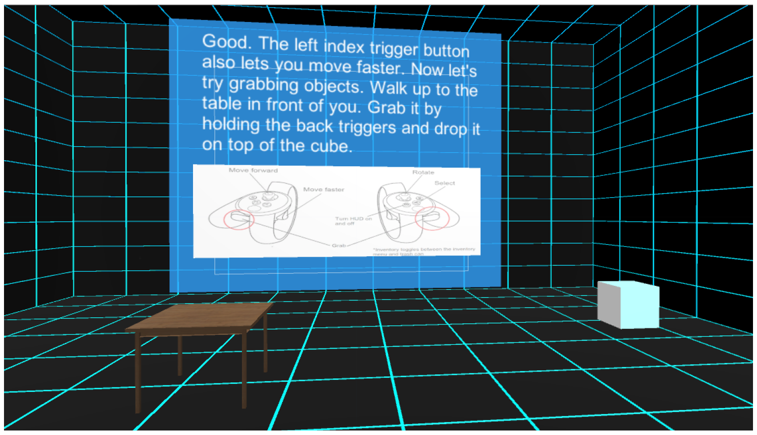
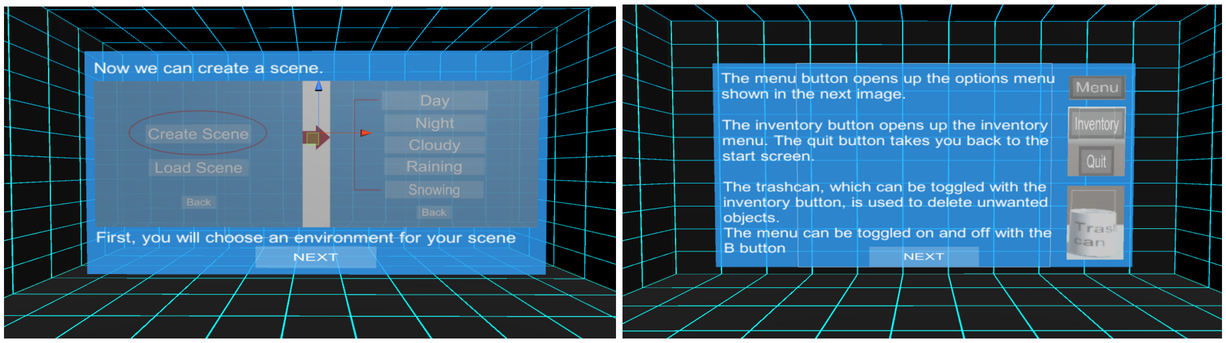

Good. The left index trigger button also lets you move faster. Now let's try grabbing objects. Walk up to the table in front of you. Grab it by holding the back triggers and drop it on top of the cube.

Figure A5: Learning How to Grab Objects

Once you drop the table on the cube, you will spawn once more in a different portion of the tutorial where you will learn how to create a scene, as well as what each of the buttons do once you are in the scene, as seen in Figure 12 and Figure 13.


Now we can create a scene.

First, you will choose an environment for your scene

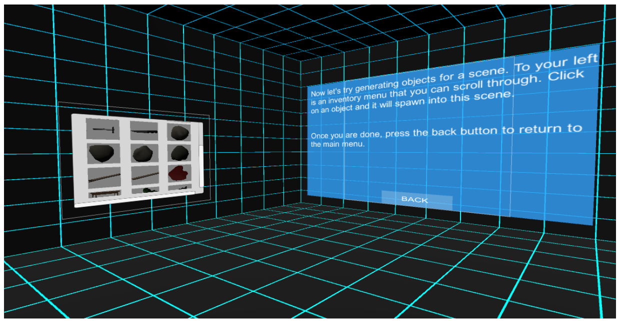
The menu button opens up the options menu shown in the next image.

The inventory button opens up the inventory menu. The quit button takes you back to the start screen.

The trashcan, which can be toggled with the inventory button, is used to delete unwanted objects.
The menu can be toggled on and off with the B button

Figure A6: Learning How to Select a Scene and Navigate Scene Buttons

Once you select **Next**, you will spawn into the last portion of the Tutorial where you will have the chance to learn how to spawn items and use the inventory menu. When you spawn in, the **Inventory Menu** will be located to the left of you avatar. To spawn an object, simply hover the cursor over the desired object and press the **A** button on your right touch controller. The object will then spawn in the scene. From there you will have the ability to pick up the object and move it around, similar to how you grabbed the table earlier in the tutorial. Once you are satisfied with the Tutorial, simply press the **Back** button and you will spawn back to the Start menu.


Now let's try generating objects for a scene. To your left is an inventory menu that you can scroll through. Click on an object and it will spawn into this scene.

Once you are done, press the back button to return to the main menu.

Figure A7: Learning How to Spawn Objects and Move Them Around

A.2 Create a Scene

To **Create** a Scene select the **Start** when you first enter Sherlock. You will then be prompted with two options, **Create Scene** or **Load Scene**. Select **Create Scene**.

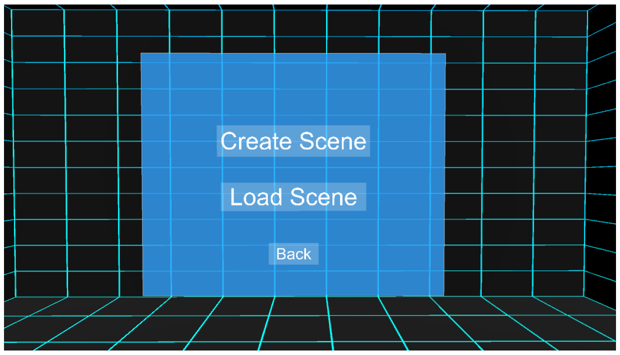


Figure A8: Select to Create a Scene or Load a Scene

The next menu will display five separate environmental options: **Day**, **Night**, **Cloudy**, **Raining**, and **Snowing** [1]. Once the user has selected an environmental scene, they will be spawned into an empty sandbox environment with nothing around them.

Within you selected environment, you will spawn with a few item in view. There will be an Menu button floating above the scene as well as the user guide for the touch controllers (note that as you swivel your head to the left/right, the menu button and user guide follow your gaze). At any point in the scene if you wish to make the menu disappear, simply press the **B** button located on you right touch controller. To the right of your avatar located at the top right corner of the sandbox will be the trash can. When spawning objects if you wish to remove them, you simply need to pick them up and drop them on top of the trash can. The objects will then disappear from the scene.

[1]The difference is only apparent in the visual of each scene, not in the functionality. All objects will function exactly the same in a Day scene as opposed to a Snowing scene. The purpose of the distinction is so each scene can have a more realistic feel for the user once they are inside.

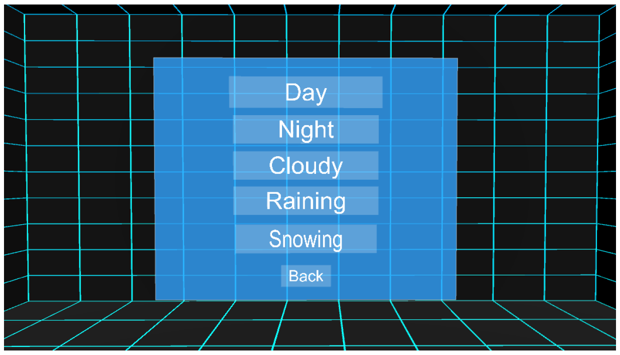


Figure A9: Select the Type of Environment For Your Scene

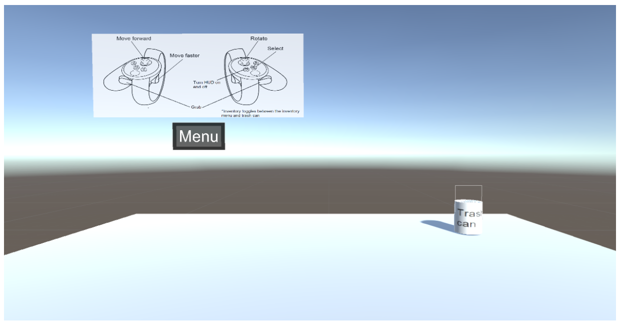


Figure A10: Day Scene Sandbox Upon Spawning In

When you are ready to begin spawning objects and placing them where you wish, press the **Menu** button to spawn more options. Once selected, an **Inventory** button, **Quit** button, **Save** button, and **Load** button will appear as seen on Figure 17.

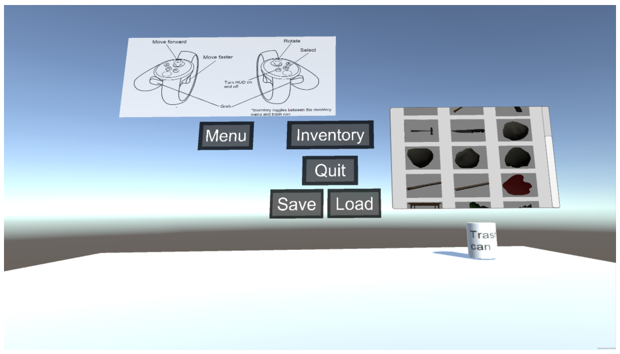


Figure A11: Inventory Menu Options

The **Inventory** button simply allows you to make the inventory menu appear or disappear (Remember, you can make the entire menu disappear by pressing the **B** button on your right touch controller). To spawn objects into the scene, simply hover the cursor over the desired object and press the **A** button. The object will then spawn in front of you. From there, you will have the ability to pick up and move the object to your desired location within the scene.

When you are ready to save your scene, press the **Save** button. Once selected, new options will appear as seen in Figure 18. You will be prompted to select one of 4 save slots to save your scene. Once you know which slot you wish to save in, hover over the slot button and press **A** (Note that when you press the button, the text on the button will change to show the current date and time, giving you visual confirmation that the save has occurred).

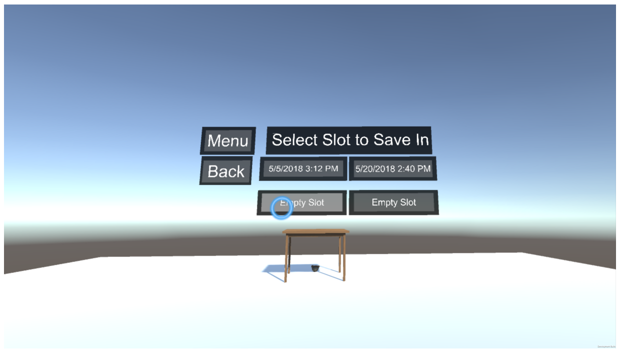


Figure A12: Save Options

At any point when you are working on you scene you can return to the **Save** option to save your scene [2]. Once you are satisfied with your scene, select the **Quit** button to return to the Start Menu.

[2]The system does not support Auto Save so please ensure to save periodically.

A.3 Load a Scene

To **Load** a Scene select the **Start** when you first enter Sherlock. You will then be prompted with two options, **Create a Scene** or **Load a Scene**. Select **Load Scene**. Once selected, you will be prompted with a menu seen in Figure 19.

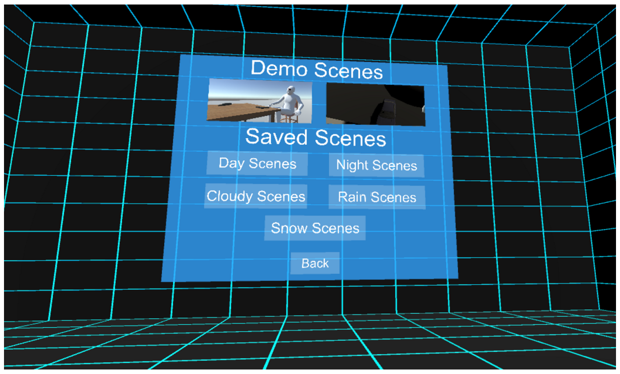


Figure A13: Load Scene Selection

You are given two types of scenes in which you can load, **Demo Scenes** and **Saved Scenes**. The **Demo Scenes** are scenes provided to you by the developers to demo the system [3]. The **Saved Scenes** are the scenes created and saved by the actual users themselves. The scenes are organized by the five types of environment they were saved in.

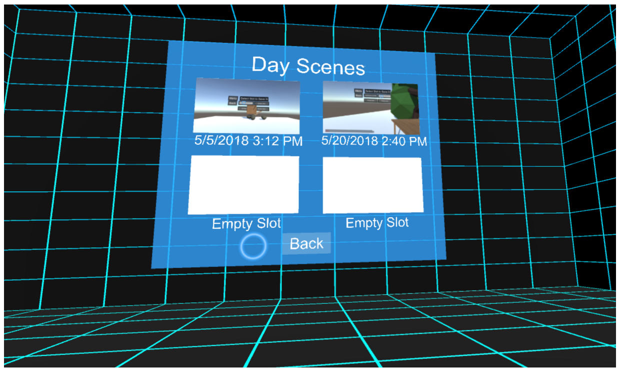


Figure A14: Load Environmental Scene Selection

Selecting one of the folders will display a menu much like Figure 20 where you will be presented with the previously saved scenes if users have worked in that environment before. There will be four slots in which scenes have been saved, each having the date and time in which the scene is saved, along with a screen shot of when the save button was pressed. To load a scene, simply hover the cursor over one of the images and press the **A** button on your right touch controller [4]. Selecting the **Back** button will bring back the options to select **Demo Scenes** or **Saved Scenes**.

[3]When you load into a Demo Scene, you will only be able to interact with a few of the objects of the scene. This is different than **Create Scene** where you have full control in the movement and placement of objects

[4]The Load feature has yet to be fully supported with the current version of the system

Appendix B

Source Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using UnityEngine.EventSystems;

public class NewMainMenu : MonoBehaviour
{
    int count = 0;

    public void BtnTest()
    {
        Debug.Log(EventSystem.current.currentSelectedGameObject.name);
    }

    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void QuitGame()
    {
        Debug.Log("Quit " + ++count );
        Application.Quit();
    }


    public void LoadAddOnClick(int level)
    {
        SceneManager.LoadScene(level);
    }

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DeleteObject : MonoBehaviour {

        void Update ()
        {
                if(OVRInput.Get(OVRInput.Button.Three))
                {
                        Debug.Log("works");
                        Destroy (this);
                }

        }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System.Globalization;
using System.Text;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Spawn : MonoBehaviour {
    private bool spawned = false;
    private float decay;

    public List<GameObject> allObjects = new List<GameObject>();
    public static List<string> objectlist = new List<string>();


    [SerializeField]
    public static List<UserData> objectData = new List<UserData>();

    public Object spawn;
    private static Canvas inGameCanvas;
    private static int childCount;

    private static int objectCount;

    Scene scene;
    string _DayFileLocation, _NightFileLocation, _CloudyFileLocation, _RainFileLocation, _S
    private static string path1, path2, path3, path4;
    UserData myData;
    string _data;

    List<string> readList = new List<string>();
    string readTime1, readTime2, readTime3, readTime4;
```

```
Vector3 VPosition;

public void LoadAddOnClick(int level)
{
    SceneManager.LoadScene(level);
    switch (level)
    {
        case 3:
            activeScene = "dayScene";
            break;
    }

    Debug.Log("Active Scene " + activeScene);

    //System.DateTime localDate = System.DateTime.Now;
    //Debug.Log(localDate.ToString("g"));

    _DayFileLocation = Application.dataPath + "/Saves/DaySaves";
    _NightFileLocation = Application.dataPath + "/Saves/NightSaves";
    _CloudyFileLocation = Application.dataPath + "/Saves/CloudySaves";
    _RainFileLocation = Application.dataPath + "/Saves/RainSaves";
    _SnowFileLocation = Application.dataPath + "/Saves/SnowSaves";

    _FileName1 = "SaveData1.txt";
    _FileName2 = "SaveData2.txt";
    _FileName3 = "SaveData3.txt";
    _FileName4 = "SaveData4.txt";

    switch (activeScene)
    {
        case "dayScene":
            path1 = Path.Combine(_DayFileLocation, _FileName1);
            path2 = Path.Combine(_DayFileLocation, _FileName2);
            path3 = Path.Combine(_DayFileLocation, _FileName3);
            path4 = Path.Combine(_DayFileLocation, _FileName4);
            break;
        case "nightScene":
            path1 = Path.Combine(_NightFileLocation, _FileName1);
            path2 = Path.Combine(_NightFileLocation, _FileName2);
            path3 = Path.Combine(_NightFileLocation, _FileName3);
            path4 = Path.Combine(_NightFileLocation, _FileName4);
            break;
        case "cloudy":
            path1 = Path.Combine(_CloudyFileLocation, _FileName1);
            path2 = Path.Combine(_CloudyFileLocation, _FileName2);
            path3 = Path.Combine(_CloudyFileLocation, _FileName3);
            path4 = Path.Combine(_CloudyFileLocation, _FileName4);
            break;
        case "raining":
            path1 = Path.Combine(_RainFileLocation, _FileName1);
            path2 = Path.Combine(_RainFileLocation, _FileName2);
            path3 = Path.Combine(_RainFileLocation, _FileName3);
            path4 = Path.Combine(_RainFileLocation, _FileName4);
            break;
            case "snowing":
                path1 = Path.Combine(_SnowFileLocation, _FileName1);
                path2 = Path.Combine(_SnowFileLocation, _FileName2);
                path3 = Path.Combine(_SnowFileLocation, _FileName3);
                path4 = Path.Combine(_SnowFileLocation, _FileName4);
                break;
        }

        Debug.Log(path1);
        Debug.Log(path2);
        Debug.Log(path3);
        Debug.Log(path4);
        generateFile(path1, path2, path3, path4);


        // we need soemthing to store the information into
        myData = new UserData();

        GameObject mainObject = GameObject.Find("InGameCanvas");

        if (mainObject != null)
        {
            setCanvas(mainObject);
        }


        Load(1);
        initCount();
        Load(1);

    }

    private void Awake()
    {

        // Where we want to save and load to and from


        scene = SceneManager.GetActiveScene();
        activeScene = scene.name;

        System.DateTime localDate = System.DateTime.Now;
        Debug.Log(localDate.ToString("g"));

        _DayFileLocation = Application.dataPath + "/Saves/DaySaves";
        _NightFileLocation = Application.dataPath + "/Saves/NightSaves";
        _CloudyFileLocation = Application.dataPath + "/Saves/CloudySaves";
        _RainFileLocation = Application.dataPath + "/Saves/RainSaves";
        _SnowFileLocation = Application.dataPath + "/Saves/SnowSaves";

        _FileName1 = "SaveData1.txt";
        _FileName2 = "SaveData2.txt";
        _FileName3 = "SaveData3.txt";
        _FileName4 = "SaveData4.txt";
```

```
switch (activeScene)
{
    case "dayScene":
        path1 = Path.Combine(_DayFileLocation, _FileName1);
        path2 = Path.Combine(_DayFileLocation, _FileName2);
        path3 = Path.Combine(_DayFileLocation, _FileName3);
        path4 = Path.Combine(_DayFileLocation, _FileName4);
        break;
    case "nightScene":
        path1 = Path.Combine(_NightFileLocation, _FileName1);
        path2 = Path.Combine(_NightFileLocation, _FileName2);
        path3 = Path.Combine(_NightFileLocation, _FileName3);
        path4 = Path.Combine(_NightFileLocation, _FileName4);
        break;
    case "cloudy":
        path1 = Path.Combine(_CloudyFileLocation, _FileName1);
        path2 = Path.Combine(_CloudyFileLocation, _FileName2);
        path3 = Path.Combine(_CloudyFileLocation, _FileName3);
        path4 = Path.Combine(_CloudyFileLocation, _FileName4);
        break;
    case "raining":
        path1 = Path.Combine(_RainFileLocation, _FileName1);
        path2 = Path.Combine(_RainFileLocation, _FileName2);
        path3 = Path.Combine(_RainFileLocation, _FileName3);
        path4 = Path.Combine(_RainFileLocation, _FileName4);
        break;
    case "snowing":
        path1 = Path.Combine(_SnowFileLocation, _FileName1);
        path2 = Path.Combine(_SnowFileLocation, _FileName2);
        path3 = Path.Combine(_SnowFileLocation, _FileName3);
        path4 = Path.Combine(_SnowFileLocation, _FileName4);
        break;
}


generateFile(path1, path2, path3, path4);
Debug.Log(path1);
Debug.Log(path2);
Debug.Log(path3);
Debug.Log(path4);

// we need soemthing to store the information into
myData = new UserData();

GameObject mainObject = GameObject.Find("InGameCanvas");
Debug.Log("Canvas: " + mainObject);
if (mainObject != null)
{
    setCanvas(mainObject);
}


Load(1);
```

```
    initCount();
}

void Start()
{

    /*
    // Where we want to save and load to and from


    scene = SceneManager.GetActiveScene();
    activeScene = scene.name;

    System.DateTime localDate = System.DateTime.Now;
    Debug.Log(localDate.ToString("g"));

    _DayFileLocation = Application.dataPath + "/Saves/DaySaves";
    _NightFileLocation = Application.dataPath + "/Saves/NightSaves";
    _CloudyFileLocation = Application.dataPath + "/Saves/CloudySaves";
    _RainFileLocation = Application.dataPath + "/Saves/RainSaves";
    _SnowFileLocation = Application.dataPath + "/Saves/SnowSaves";

    _FileName1 = "SaveData1.txt";
    _FileName2 = "SaveData2.txt";
    _FileName3 = "SaveData3.txt";
    _FileName4 = "SaveData4.txt";

    switch (activeScene)
    {
        case "dayScene":
            path1 = Path.Combine(_DayFileLocation, _FileName1);
            path2 = Path.Combine(_DayFileLocation, _FileName2);
            path3 = Path.Combine(_DayFileLocation, _FileName3);
            path4 = Path.Combine(_DayFileLocation, _FileName4);
            break;
        case "nightScene":
            path1 = Path.Combine(_NightFileLocation, _FileName1);
            path2 = Path.Combine(_NightFileLocation, _FileName2);
            path3 = Path.Combine(_NightFileLocation, _FileName3);
            path4 = Path.Combine(_NightFileLocation, _FileName4);
            break;
        case "cloudy":
            path1 = Path.Combine(_CloudyFileLocation, _FileName1);
            path2 = Path.Combine(_CloudyFileLocation, _FileName2);
            path3 = Path.Combine(_CloudyFileLocation, _FileName3);
            path4 = Path.Combine(_CloudyFileLocation, _FileName4);
            break;
        case "raining":
            path1 = Path.Combine(_RainFileLocation, _FileName1);
            path2 = Path.Combine(_RainFileLocation, _FileName2);
            path3 = Path.Combine(_RainFileLocation, _FileName3);
            path4 = Path.Combine(_RainFileLocation, _FileName4);
            break;
        case "snowing":
            path1 = Path.Combine(_SnowFileLocation, _FileName1);
            path2 = Path.Combine(_SnowFileLocation, _FileName2);
            path3 = Path.Combine(_SnowFileLocation, _FileName3);
            path4 = Path.Combine(_SnowFileLocation, _FileName4);
            break;
    }


    generateFile(path1, path2, path3, path4);
    Debug.Log(path1);
    Debug.Log(path2);
    Debug.Log(path3);
    Debug.Log(path4);

    // we need soemthing to store the information into
    myData = new UserData();

    GameObject mainObject = GameObject.Find("InGameCanvas");
    Debug.Log("Canvas: " + mainObject);
    if (mainObject != null)
    {
        setCanvas(mainObject);
    }


    //Load(1);
    initCount();

     */
}

public void readFile(string path, int currentFile)
{
    string line = "";

    using (StreamReader readtext = File.OpenText(path))
    {
        switch (currentFile)
        {
            case 1:
                readTime1 = readtext.ReadLine();
                break;
            case 2:
                readTime2 = readtext.ReadLine();
                break;
            case 3:
                readTime3 = readtext.ReadLine();
                break;
            case 4:
                readTime4 = readtext.ReadLine();
                break;

        }
```

```
            fileScene = readtext.ReadLine();
            Debug.Log("File Scene: " + fileScene);
            while ((line = readtext.ReadLine()) != null)
            {
                readList.Add(line);
            }
            readtext.Close();
        }
    }

    public void setButtons()
    {
        if (File.Exists(path1))
        {
            readFile(path1, 1);
            Debug.Log("Time 1: " + readTime1);
            if (!(string.IsNullOrEmpty(readTime1)))
            {
                GameObject.Find("Slot1_button").GetComponentInChildren<Text>().text = readT
                GameObject.Find("Slot1_button").GetComponentInChildren<Text>().fontSize = 
            }
            else
            {
                GameObject.Find("Slot1_button").GetComponentInChildren<Text>().text = "Emp
                GameObject.Find("Slot1_button").GetComponentInChildren<Text>().fontSize = 
            }
        }

        if (File.Exists(path2))
        {
            readFile(path2, 2);
            Debug.Log("Time 2: " + readTime2);
            if (!(string.IsNullOrEmpty(readTime2)))
            {
                GameObject.Find("Slot2_button").GetComponentInChildren<Text>().text = readT
                GameObject.Find("Slot2_button").GetComponentInChildren<Text>().fontSize = 
            }
            else
            {
                GameObject.Find("Slot2_button").GetComponentInChildren<Text>().text = "Emp
                GameObject.Find("Slot2_button").GetComponentInChildren<Text>().fontSize = 
            }
        }




        if (File.Exists(path3))
        {
            readFile(path3, 3);
            Debug.Log("Time 3: " + readTime3);
            if (!(string.IsNullOrEmpty(readTime3)))
            {
                GameObject.Find("Slot3_button").GetComponentInChildren<Text>().text = readT
```

```
                    GameObject.Find("Slot3_button").GetComponentInChildren<Text>().fontSize = 
                }
                else
                {
                    GameObject.Find("Slot3_button").GetComponentInChildren<Text>().text = "Emp
                    GameObject.Find("Slot3_button").GetComponentInChildren<Text>().fontSize = 
                }
            }


            if (File.Exists(path4))
            {
                readFile(path4, 4);
                Debug.Log("Time 4: " + readTime4);
                if (!(string.IsNullOrEmpty(readTime4)))
                {
                    GameObject.Find("Slot4_button").GetComponentInChildren<Text>().text = readT
                    GameObject.Find("Slot4_button").GetComponentInChildren<Text>().fontSize = 
                }
                else
                {
                    GameObject.Find("Slot4_button").GetComponentInChildren<Text>().text = "Emp
                    GameObject.Find("Slot4_button").GetComponentInChildren<Text>().fontSize = 
                }
            }
        }


        public void setCanvas(GameObject mainObject)
        {
            inGameCanvas = mainObject.GetComponent<Canvas>();
        }

        public static Canvas getCanvas()
        {
            return inGameCanvas;
        }

        public void setChildCount(int ccount)
        {
            childCount = ccount;
        }

        public void initCount()
        {
            objectCount = 0;
        }

        public void incrementCount()
        {
            objectCount++;
        }

        public int getCount()
```

```
{
    return objectCount;
}

public void adduserData(string objectName, int count)
{
    UserData user = new UserData();
    user._iUser.name = objectName;
    user._iUser.id = count;
    objectData.Add(user);
}

public void spawnObject(string objectName)
    {

    //Instantiate (spawn, transform.position+(transform.forward*2), transform.rotation
    GameObject newObject = (GameObject)Instantiate(spawn, transform.position + (transfo
    newObject.transform.parent = getCanvas().transform;


    Debug.Log(newObject);
    Debug.Log(objectData.Count);
    incrementCount();
    setChildCount(getCanvas().transform.childCount);
    adduserData(objectName, getCount());

}

public void screenShot(string c_scene, int slot)
{
    string path = "";
    switch (c_scene)
    {
        case "dayScene":
            path = _DayFileLocation;
            break;
        case "nightScene":
            path = _NightFileLocation;
            break;
        case "cloudy":
            path = _CloudyFileLocation;
            break;
        case "raining":
            path = _RainFileLocation;
            break;
        case "snowing":
            path = _SnowFileLocation;
            break;
    }
    string shot = c_scene + slot + ".png";
    ScreenCapture.CaptureScreenshot(Path.Combine(path, shot));
}

public void Save(int slot)
```

```
{
    List<string> newList = new List<string>();
    List<int> IDList = new List<int>();
    List<string> nameList = new List<string>();
    string line = "";
    string path = "";
    string time = "";
    string currentScene = "";
    string saveTime = System.DateTime.Now.ToString("g");

    //System.DateTime localDate = System.DateTime.Now.ToString("g");
    //Debug.Log(localDate.ToString("g"));


    screenShot(activeScene, slot);
    switch (slot)
    {
        case 1:
            path = path1;
            GameObject.Find("Slot1_button").GetComponentInChildren<Text>().text = saveT
            GameObject.Find("Slot1_button").GetComponentInChildren<Text>().fontSize = 

            break;
        case 2:
            path = path2;
            GameObject.Find("Slot2_button").GetComponentInChildren<Text>().text = saveT
            GameObject.Find("Slot2_button").GetComponentInChildren<Text>().fontSize = 
            break;
        case 3:
            path = path3;
            GameObject.Find("Slot3_button").GetComponentInChildren<Text>().text = saveT
            GameObject.Find("Slot3_button").GetComponentInChildren<Text>().fontSize = 
            break;
        case 4:
            path = path4;
            GameObject.Find("Slot4_button").GetComponentInChildren<Text>().text = saveT
            GameObject.Find("Slot4_button").GetComponentInChildren<Text>().fontSize = 
            break;
    }



    using (StreamReader readtext = File.OpenText(path))
    {
        time = readtext.ReadLine();
        currentScene = readtext.ReadLine();
        while ((line = readtext.ReadLine()) != null)
        {
            newList.Add(line);
        }
        readtext.Close();

    }
```

```
        foreach (string newLine in newList)
        {
            string[] values = newLine.Split('|');
            string id = values[0];
            string name = values[1];


            string realId = id.Split(':')[1];
            string realName = name.Split(':')[1];
            IDList.Add(System.Int32.Parse(realId));
            nameList.Add(realName);
        }

        Debug.Log("Before save loop");
        Debug.Log(childCount);
        File.WriteAllText(path, string.Empty);
        //File.WriteAllText(path, string.Empty);

        for (int i = 0; i < childCount; i++)
        {

            Debug.Log("In game loop " + i);

            if( !(IDList.Contains(objectData[i]._iUser.id))   && !(nameList.Contains(object
            {
                objectData[i]._iUser.x = getCanvas().transform.GetChild(i).position.x;
                objectData[i]._iUser.y = getCanvas().transform.GetChild(i).position.y;
                objectData[i]._iUser.z = getCanvas().transform.GetChild(i).position.z;

                objectData[i]._iUser.xx = getCanvas().transform.GetChild(i).rotation.x;
                objectData[i]._iUser.yy = getCanvas().transform.GetChild(i).rotation.y;
                objectData[i]._iUser.zz = getCanvas().transform.GetChild(i).rotation.z;

                string xValues = "Position X: " + objectData[i]._iUser.x + " | " + "Rotatio
                string yValues = "Position Y: " + objectData[i]._iUser.y + " | " + "Rotatio
                string zValues = "Position Z: " + objectData[i]._iUser.z + " | " + "Rotatio

                string objectInfo = "ID: " + objectData[i]._iUser.id + " | " + "Name: " +
                Debug.Log(objectInfo);
                objectlist.Add(objectInfo);
            }

        }

        createFile(objectlist, path, saveTime);
    }

    public void generateFile(string firstPath, string secondPath, string thirdPath, string
    {
        Debug.Log("Entered Generate File");
        if (!File.Exists(firstPath))
        {
            Debug.Log("File 1 does not exist");
            StreamWriter writetext = File.CreateText(firstPath);
            if (File.Exists(firstPath))
            {
                Debug.Log("File 1 Created!");
            }
        }

        if (!File.Exists(secondPath))
        {
            Debug.Log("File 2 does not exist");
            StreamWriter writetext = File.CreateText(secondPath);
            if (File.Exists(secondPath))
            {
                Debug.Log("File 2 Created!");
            }
        }

        if (!File.Exists(thirdPath))
        {
            Debug.Log("File 3 does not exist");
            StreamWriter writetext = File.CreateText(thirdPath);
            if (File.Exists(thirdPath))
            {
                Debug.Log("File 3 Created!");
            }
        }

        if (!File.Exists(fourthPath))
        {
            Debug.Log("File 4 does not exist");
            StreamWriter writetext = File.CreateText(fourthPath);
            if (File.Exists(fourthPath))
            {
                Debug.Log("File 4 Created!");
            }
        }

    }

    public void createFile(List<string> list, string path, string time)
    {
        Debug.Log("Made it in Create file function");
        using (StreamWriter writetext = File.CreateText(path))
        {
            writetext.Flush();
            writetext.WriteLine(time);
            writetext.WriteLine(activeScene);
            foreach (string line in list)
            {
                writetext.WriteLine(line);
            }
            writetext.Close();

        }
        Debug.Log("File Written Successfully");
```

```
}

public void Load(int slot)
{

    Debug.Log("Made it in Load file function");
    List<string> newList = new List<string>();
    List<int> IDList = new List<int>();
    List<string> nameList = new List<string>();

    string line = "";
    string path = "";

    switch (slot)
    {
        case 1:
            path = path1;
            break;
        case 2:
            path = path2;
            break;
        case 3:
            path = path3;
            break;
        case 4:
            path = path4;
            break;
    }


    using (StreamReader readtext = File.OpenText(path))
    {
        Debug.Log("Time: " + readtext.ReadLine());
        Debug.Log("Current Scene: " + readtext.ReadLine());

        while ((line = readtext.ReadLine()) != null)
        {
            Debug.Log(line);
            newList.Add(line);
        }
        readtext.Close();

    }

    foreach (string newLine in newList)
    {
        string[] values = newLine.Split('|');
        string id = values[0];
        string name = values[1];
        string xPosition = values[2];
        string xRotation = values[3];
        string yPosition = values[4];
        string yRotation = values[5];
        string zPosition = values[6];
        string zRotation = values[7];


        string realId = id.Split(':')[1];
        string realName = name.Split(':')[1];
        string realXPosition = xPosition.Split(':')[1];
        string realXRotation = xRotation.Split(':')[1];
        string realYPosition = yPosition.Split(':')[1];
        string realYRotation = yRotation.Split(':')[1];
        string realZPosition = zPosition.Split(':')[1];
        string realZRotation = zRotation.Split(':')[1];

        IDList.Add(System.Int32.Parse(realId));
        nameList.Add(realName);

        Debug.Log("ID from Load: " + realId + " | " + realName);

            switch (realName.Trim())
            {
            case "Bat":
                Vector3 newBatPosition = new Vector3(float.Parse(realXPosition), float
                GameObject spawnBatObject = (GameObject)Instantiate(Resources.Load("Ba
                spawnBatObject.transform.parent = getCanvas().transform;
                break;
            case "Dead1":
                Vector3 newDead1Position = new Vector3(float.Parse(realXPosition), flo
                GameObject spawnDead1Object = (GameObject)Instantiate(Resources.Load("
                spawnDead1Object.transform.parent = getCanvas().transform;
                break;
            case "Dead2":
                Vector3 newDead2Position = new Vector3(float.Parse(realXPosition), flo
                GameObject spawnDead2Object = (GameObject)Instantiate(Resources.Load("
                spawnDead2Object.transform.parent = getCanvas().transform;
                break;
            case "Dead3":
                Vector3 newDead3Position = new Vector3(float.Parse(realXPosition), flo
                GameObject spawnDead3Object = (GameObject)Instantiate(Resources.Load("
                spawnDead3Object.transform.parent = getCanvas().transform;
                break;
            case "Dead4":
                Vector3 newDead4Position = new Vector3(float.Parse(realXPosition), flo
                GameObject spawnObject = (GameObject)Instantiate(Resources.Load("Dead8
                spawnObject.transform.parent = getCanvas().transform;
                break;
            case "Gun":
                Vector3 newPosition = new Vector3(float.Parse(realXPosition), float.Pa
                GameObject spawnDead4Object = (GameObject)Instantiate(Resources.Load("
                spawnDead4Object.transform.parent = getCanvas().transform;
                break;
            case "Hammer":
                Vector3 newHammerPosition = new Vector3(float.Parse(realXPosition), fl
                GameObject spawnHammerObject = (GameObject)Instantiate(Resources.Load(
                Debug.Log(getCanvas());
                spawnHammerObject.transform.parent = getCanvas().transform;
```

```
            break;
        case ”Knife”:
            Vector3 newKnifePosition = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnKnifeObject = (GameObject)Instantiate(Resources.Load(”
            spawnKnifeObject.transform.parent = getCanvas().transform;
            break;
        case ”Rock1”:
            Vector3 newRock1Position = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnRock1Object = (GameObject)Instantiate(Resources.Load(”
            spawnRock1Object.transform.parent = getCanvas().transform;
            break;
        case ”Rock2”:
            Vector3 newRock2Position = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnRock2Object = (GameObject)Instantiate(Resources.Load(”
            spawnRock2Object.transform.parent = getCanvas().transform;
            break;
        case ”Rock3”:
            Vector3 newRock3Position = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnRock3Object = (GameObject)Instantiate(Resources.Load(”
            spawnRock3Object.transform.parent = getCanvas().transform;
            break;
        case ”Rock4”:
            Vector3 newRock4Position = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnRock4Object = (GameObject)Instantiate(Resources.Load(”
            spawnRock4Object.transform.parent = getCanvas().transform;
            break;
        case ”woodBeam1”:
            Vector3 newWoodBeam1Position = new Vector3(float.Parse(realXPosition),
            GameObject spawnWoodBeam1Object = (GameObject)Instantiate(Resources.Lo
            spawnWoodBeam1Object.transform.parent = getCanvas().transform;
            break;
        case ”woodBeam2”:
            Vector3 newWoodBeam2Position = new Vector3(float.Parse(realXPosition),
            GameObject spawnWoodBeam2Object = (GameObject)Instantiate(Resources.Lo
            spawnWoodBeam2Object.transform.parent = getCanvas().transform;
            break;
        case ”Blood”:
            Vector3 newBloodPosition = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnBloodObject = (GameObject)Instantiate(Resources.Load(”
            spawnBloodObject.transform.parent = getCanvas().transform;
            break;
        case ”squareTable”:
            Vector3 newTablePosition = new Vector3(float.Parse(realXPosition), flo
            GameObject spawnTableObject = (GameObject)Instantiate(Resources.Load(”
            spawnTableObject.transform.parent = getCanvas().transform;
            break;
        case ”Tree”:
            Vector3 newTreePosition = new Vector3(float.Parse(realXPosition), floa
            GameObject spawnTreeObject = (GameObject)Instantiate(Resources.Load(”t
            spawnTreeObject.transform.parent = getCanvas().transform;
            break;
        case ”Evergreen”:
            Vector3 newEvergreenPosition = new Vector3(float.Parse(realXPosition),
            GameObject spawnEvergreenObject = (GameObject)Instantiate(Resources.Lo
                    spawnEvergreenObject.transform.parent = getCanvas().transform;
                    break;

                }

            }

            Debug.Log("File Loaded Successfully");
        }




}

// UserData is our custom class that holds our defined objects we want to store in XML forn
[System.Serializable]
public class UserData
{
    // We have to define a default instance of the structure
    public DemoData _iUser;
    // Default constructor doesn't really do anything at the moment
    public UserData() { }

    // Anything we want to store in the XML file, we define it here
    [System.Serializable]
    public struct DemoData
    {

        //name
        public string name;
        public int id;

        //Position
        public float x;
        public float y;
        public float z;

        //Rotation
        public float xx;
        public float yy;
        public float zz;
    }
}
```