COS 301 Capstone Project 2017

Vulknut Software Engineering

Test Reports

Compiled By

Peter Boxall - u14056136
Claude Greeff - u13153740
Marin Peroski - u13242475
Johan du Plooy - u12070794
Bernhard Schuld - u10297902

3D VR Presentations

GitHub Repository: Valknut Software Engineering

Contents

Test Reports

This project posed a few challenges with regards to a suitable testing environment. The best methodology that we found in order to properly test everything we were working on was to use the standard white box and black box approach, while maintaining Unity as the software that will house all of the testing. Usability testing is also another method that will be employed in the project as development progresses. Unit testing will be incorporated where possible.

Internal: Testing With Team Members

The following describes the various methods employed in order to thoroughly test the virtual presentation software.

White Box Testing

When testing occurred with members the black box and white box methods were employed at this stage. Team members would first white box test their own work in the Unity engine in order to determine if everything was running as expected. Considering that the majority of the project's code has some form of visual impact, the team member testing their own work will first have to set an outcome that they would expect once their code has been implemented into the project. They would then run the Unity engine and test whether their code behaves the way that they expected or intended. The code produced by each team member was uploaded to their GitHub branch only if it was working in their environment.

Black Box Testing

Once the team member's code is uploaded, they would inform the rest of the team and at least one other member would perform a pull request from their branch. That member will be told what the expected outcome is, thus they would test their code without knowing the inner workings of it. That individual will then run the Unity engine on their local machine in order to test the code thoroughly. After all checks have been passed (does it work as intended, is the functionality indeed correct and does the program behave as expected) they would approve the pull request to the dev branch.

Usability Testing

Usability testing will involve gathering members with various backgrounds with regards to their computer knowledge. This will allow for a thorough test in that both advanced and simple features are adequately covered. The testing pool of users will range from roughly five to ten users. They will be asked to perform a set amount of tasks and the time taken for each individual to complete their task will be recorded. They will also be allowed to ask questions if they get stuck. These questions will also get recorded in order to help improve the software's usability (with the individual's consent). After they have completed their task we will ask them for their feedback and how they felt about the software experience overall. All of these results will be taken into consideration.

Introduction

Before the October Recess we went to EPI-USE to demo our progress up to that point and to conduct usability tests with some of their employees. The feedback was verbally open-ended so that we could better determine where our software system could be improved.

The general consensus was that the overall quality of our software system was satisfactory, but they did mention that we could improve in the following areas:

1. Add a help function to easily check on the controls
2. Pause videos in the scene using a hotkey.

3. Change the distance between objects the user when interacting with them
4. Have a more intuitive contextual menu
5. Add text to the scene

Implementation

All of the above features were implemented during the October recess in the following ways:

1. A help function was added and is always visible during the scene. The hotkey for this function is the universal F1 hotkey.
2. Videos may now be paused using the "V" hotkey.
3. The Distance between objects and the user while interacting with said objects can now be increased or decreased via the mouse scroll wheel.
4. The contextual menu was changed to look more appealing and also contains numbers next to each item in the menu to be more intuitive.
5. Text can now be added to the scene via images. You may add text to an image using your preferred image editor and then import the image into the scene, provided that the image is in .jpg format.

Unit Testing

Unit testing will be employed where possible, although this method of testing will be the least recurring one due to the nature of the project. It will be employed in code where checks need to be, for example. where object checks need to occur (if the object is NULL or not); if the object has certain properties such as an audio component attached to it; if images, audio, video, textures and materials are indeed detected and do fill up their corresponding data structures.

Functionality Tested

The list below specifies each feature that was tested up until the current development point.

1. Object manipulation: this entails all interaction that can occur with an object such as picking it up and moving to around. The issues encountered here dealt with the way the object handled in the users hands. At first it rotated freely and it collided with other objects. These abnormalities were taken care of.
2. Object scaling: this code was implemented without any problem. The objects scaled correctly.
3. Object rotation: this code was implemented without any problem. The objects rotated correctly. The addition of axis resetting was added to make it easier to return the object back to its original form.
4. Object creation: the objects had to be spawned from a central point on the map. The first problem encountered was that the objects would spawn into a stationary point and remain there. Code was added to ensure that they were moved to the user's hands in their current position.
5. Object deletion: this code worked correctly on implementation
6. Image importing: this code was first implemented in a primitive form by being assigned to an object. This was then changed for versatile application by using the Unity ray cast feature (line of sight) in order to apply the image to an object.

7. Audio importing: this code was first implemented in a primitive form by being assigned to an object. This was then changed for versatile application by using the Unity ray cast feature (line of sight) in order to apply the audio to an object. This caused a problem at first as multiple audio components could be created and assigned to an object, thus a check was made to ensure that only one would be created.

8. Video importing: this code was first implemented in a primitive form by being assigned to an object. This was then changed for versatile application by using the Unity ray cast feature (line of sight) in order to apply the video and audio to an object. The same checks were made to the audio here as stated above. The video caused a few issues as only the .ogv format could be used. Conversions had to be made first and then assigned as a texture to the object.

9. 360 video exporting: a Unity asset was taken to perform the recording for us however, it currently runs at a low framerate when recording takes place. The exported product is of high quality and framerate. Tinkering with the default script may need to occur in order to optimise the recorder.

10. Skybox importing: the skyboxes needed to be placed into the resources folder first so that they could be loaded into the scene live. Once that was resolved they were working properly.

11. Contextual menus: Contextual menus were added after the initial demo and have been tested.