De Montfort University

Annihilation Intelligence (A.I.)

A 3D 1st Person Horror Mobile Phone Game

Final Report

By Simon Chiu

Computer Games Programming

P14133969

Abstract

Annihilation Intelligence is a 3D 1st Person Horror mobile phone game, developed in Android Studio for the Android OS (operating system). This has been developed as a foray into creating and developing an entire game, as the Computer Games Programming course does not venture far from development, and focuses on the coding and programming of a game rather than the game as a whole.

The overall aim of the project started with exploring the term *Horror Vacui* as found in the literature review for the 'Interim Deadline'. The idea that anything under the horror genre utilised the fear of unknown space in order to engage the player was intriguing, as well as the development of a game under a genre not touched upon in the course. The development of the game also explored 3D development for a mobile phone platform, a continuously growing market that many people have access to, and the limits both physically, and in computational power, of a mobile phone that developers must work with.

Annihilation Intelligence was developed using Android Studio, utilising j-PCT-AE for 3D, coded in the Java programming language, as well as XML for the activity layouts. 3D objects were created in Autodesk Maya 2016, along with simple textures, and music and sound effects were created in FL Studio, Bfxr, or recorded.

Table of Contents

1 Introduction

1.1 Background and justification

There are a few reasons for the choice of project. Firstly it was surrounding a system of great interest: Mobile phone games. In today's society, mobile phones are, and continue to be, a large part of everyday life. It has long evolved from a simple communication device, to a utility for work, a platform for playing games, and even for scheduling a person's day to day life. Besides being a large part of life, the mobile phone game market has also been constantly expanding, as it is a gaming platform where the console is easily accessible to people (as most people own a mobile phone) and the games are generally quick to play. People can play simple games to pass the time while travelling, or while waiting.

This is an area of games creation which is extremely interesting. Under the Computer Games Programming course, students develop some experience with mobile games, namely from the IMAT 2608 Mobile Games module, where students developed simple 2D mobile phone games in Android Studio. However, this was limited to 2D games, and this project presents a chance to create a 3D mobile phone game, and explore difficulties which may arise from 3D mobile phone game development. With the increase in the power of mobile phones, 3D games have become increasingly popular for mobile phones, with large video games companies who originally make games for home-console systems also joining in mobile phone development.

Another reason for this project is the general interest in games creation, notably independent game development (Indie games). Indie games have become more popular over the years, which do not rely on large graphics or visuals, but instead focused more on story or game play to entice players. There are many cases of successful indie games, for example the '*Five Nights at Freddy's'* series by Scott Cawthorn, which has been discussed in the previous literature review in the 'Interim Deadline'. But another example is '*Stardew Valley'* by Eric Barone. Developed over four years, all parts of the game, from programming and design, to art and composition, were all done by Eric. This is an inspiring story for indie games developers, showing Eric's passion for this game, and his hard work spent on the game.

As well as the creation of a game, the genre – Horror – is also a point of interest. In recent years, Horror games have been a popular choice for indie developers, as using jump scares or the use of atmosphere in the game is a simple, yet effective method of engaging the player. During the literature review, the term *Horror Vacui* was explored, and the fear of something which might exist, yet may not necessarily be seen. This was very interesting, and the idea of trying to create a Horror game while avoiding the cliché uses of jump scares or sudden movements was intriguing.

2 Summary of Project

2.1 Project Plan

The 'Interim Deadline' showed promise in the use of jPCT-AE for 3D development, as well as showed the detection of touch-screen inputs to control the character. For the final version, a project structure had to be formed, similarly to how games were structured in the IMAT 2608 Mobile Games module from second year. This separates the project into Activities, Classes and Views. Activities contain the different 'screens' of the game, such as the start-up screen, the options screen, etc. Classes contain the Java classes used to define objects, for example the player, certain objects in the world, etc. Lastly the Views are how the game is displayed. The game will use a surface view to render and draw objects.

2.1.1 Repository/ Source Control

Before continuing from the 'Interim Deadline', an online repository was set up to hold the project, allowing better management of the project, as well as recording changes to the project, and time stamps of when the project was updated. Appendix 5.1 shows the list of commits to GitHub, an online repository service to upload projects in progress, as well as allow easy access for other people to a project. Each of these commits represent a change to the project, such as new activities and classes, fixes to bugs and errors, as well as changes to assets, documentation, or even testing in a new branch.

2.1.2 Activities

As a foundation to build upon, activities are the 'pages' or 'screens' of an Android app, used to create a flow of how a user interacts with an app. Users are able to switch between activities, and each activity holds different parts of the app. This also acts as a shell for the app, showing how the app will be structured, and how one activity is able to send data to subsequent activities.

2.1.2.1 Activity Life-cycle

Fig.1 shows the planned activity life-cycle, and how a user can move between activities. Starting up the game enters the Splash Activity. This splash screen allows us to start loading parts of the game, hiding this from the user with a loading screen. From here the Menu Activity is accessed. This is the main hub from which other activities can be accessed. The Splash Activity will not be accessible anymore. Trying to return to the previous activity (the Splash Activity) from the Menu Activity will instead close the game, showing a toast message to confirm if the user wants to quit the game.

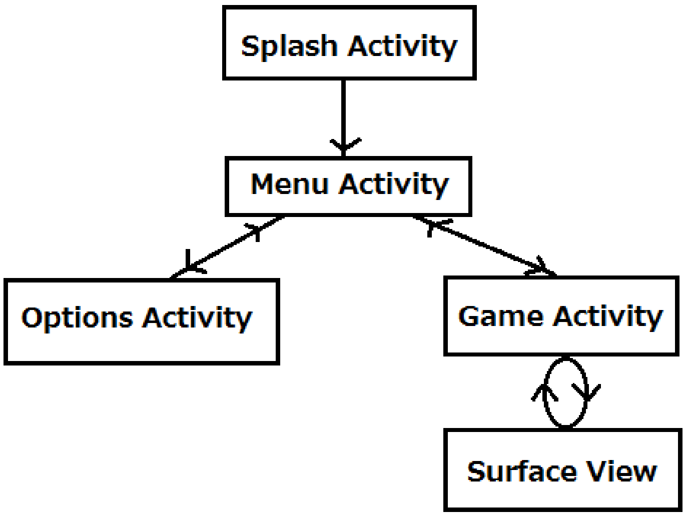


Fig.1 – Planned activity flow. The Surface View runs alongside the Game Activity

The Menu Activity can then branch to either the Game Activity or the Options Activity. In the Options Activity, the user is able to set certain settings, such as the volume of the music and sound effects, as well as an option to reset these to default settings. Unless the user confirms these settings, leaving this activity and returning to the Menu Activity will return the settings to the last used setting. The Game Activity is linked to a Surface View, which is how the 3D objects are rendered to the screen, as well as updated over time. The interaction between the Game Activity and Surface View is important, for example when the player touches the screen, the Game Activity takes this input, and then passes it to the Surface View to process, and when the Surface View requires access to the Game Activity, such as playing a sound effect, it can run a function to access the activity. When the game ends, the Surface View will return this to the Game Activity, and it will return to the Menu Activity.

2.1.2.2 Flow of data through Activities

Certain data will be reused in different activities, such as the volume of the music, which must be consistent throughout the activities, and must use the volume set in the Options Activity. When moving from one activity to another, certain data can be set and passed by changing the Intent, which is essentially the new activity created in the previous activity, and set data before being ran. By doing this, data can also be recovered from a child activity back to its parent.

For example, the options data will be loaded from the Splash Activity, and passed to the Menu Activity. When accessing the Options Activity, the data is again passed, and changing any of the options and leaving the Options Activity will return the new data to the Menu Activity, which will update its local version of the data, as well as the text file where it was originally loaded from.

Java does not allow the use of Pointers in ways that C++ does. A pointer cannot be created for an integer for example, and then passed through the activities. Because of this, each activity has a local version of any passed data, and starting or ending an activity requires setting the current activity with any new data. There are pointers in Java to an extent, but this is more for accessing the data in an Activity Layout.

In Fig.2 the activity layout for the Options Activity is displayed. The various check boxes, radio boxes and seek bars can be accessed with Java pointers, searching for the ID of a widget. Using this, the widgets can be accessed and changed from the Activity code. Widgets such as buttons are given an 'onClick' variable, which becomes a function in the Activity code.

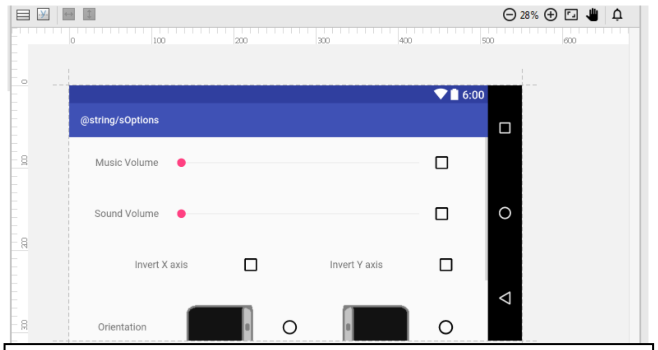


Fig.2 – The Activity Layout for the Options Activity

2.1.2.3 Why data flow is important

The setup of data to be used throughout the app is extremely important. Loading times can be saved by not loading the same data over again in different parts of the app, but instead by loading all the data in the Splash Activity, acting as a loading screen to then allow the rest of the app to run more smoothly. Players do not want to be wasting time reloading data, especially on a device with limited computational power. Similar to early console games, efficiency is key to developing a fast-running game.

2.1.2.4 Loading/ Saving Data

Certain data, such as the volume of music or the orientation of the screen should be saved, so if a user has changed the settings, they can keep those settings when they run the app again. To do this, certain data is saved to a text file, and this can be loaded from and saved to. In the Splash Activity, this data is loaded by reading the text file and parsing each line in the text file. If the text file or directory does not exist (such as when running the app for the first time, or if the data is somehow deleted) a default text file will be created with the default settings.

Similarly when saving data which has changed from the options, data is saved as individual strings, with backup data if it cannot be written properly to the text file. Having default data to fall back on is imperative to avoid errors, or potentially crashes.

2.2 Development

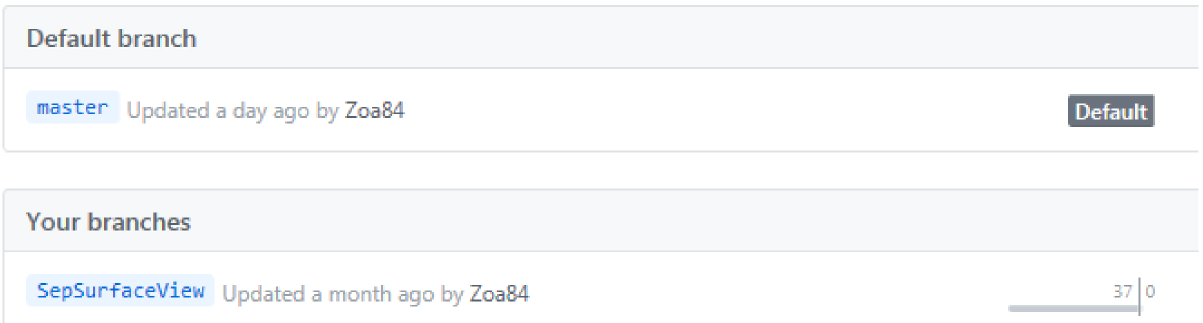
2.2.1 Singleton Classes

As well as the activities as a foundation of the project, the flow of data between these activities had to be set up. The use of intents was explained in the Project Plan; however, this is limited to certain data types, such as integers or strings. Problems arose with the need to pass music and sound, which is loaded from the Splash Activity, to the rest of the project. To fix this problem, a singleton class was created: a class where only one instance of it exists. This means any data set to this instance can be accessed by other activities and each activity will not have a local version, but instead share a global version of the class. The Media class was created, which is first used to load music and sound files from the Splash Activity, and is then used in subsequent activities to access music and sound effects. As well as allowing access to music and sound, this class also acts as the media player, and plays music and sound effects from the class.

To use this class, an activity can run a function on an instance of the class, passing parameters such as the volume, and which sound to play. However, on using this, the problem of manually using numbers to access the assets arose. This makes it difficult if assets are changed, or if other sound effects are required. Similar to the Media class as a singleton, a Defines class was created, used not only to define terms for music and sound effects, but as a singleton, it was also used for defining certain terms for options, and even for mathematical terms, such as converting from degrees to radians.

2.2.2 Separating the Game Activity and Game Surface View

The Game Activity from the 'Interim Deadline' originally had the surface view integrated into the activity. Although this made it easy to use variables between the two, this also made the code very messy and not very maintainable. This should be separated into an activity and a surface view class, however, this could potential break the code, and jPCT-AE may not actually work when doing this. To avoid changing the main code, while also being able to test a potentially better layout, a branch on GitHub was created, to test separating the two apart (Fig.3).


Default branch

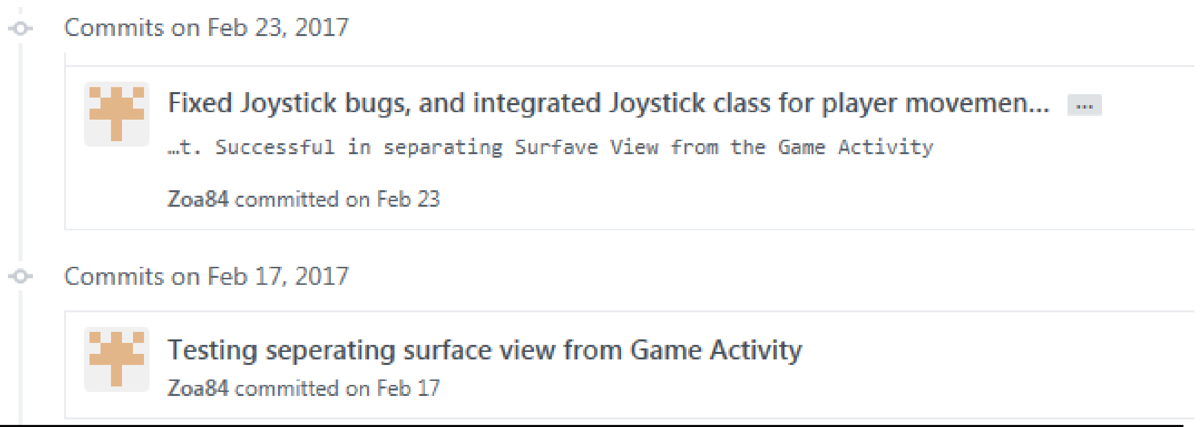
master Updated a day ago by Zoa84 Default

Your branches

SepSurfaceView Updated a month ago by Zoa84 37 | 0

Fig.3 – The 'master' branch (Top), the default branch for development, and the 'SepSurfaceView' branch (Bottom) for testing changes before merging with the master branch

With a branch on GitHub, different things can be tested (in this case, separating the surface view) and be pushed to the repository, without worry of replacing the original code; essentially having a separate repository. Once done with the branch, it can then be either merged, saving the changes done with the branch, or it can be deleted, if for example any changes made did not work, or if it was only for testing. The use of branches in a project, especially a group project, is very important, as it allows members of a group to work on individual parts of the code, without worry of affecting others or getting affected by others' changes. Although not as vital in a solo project, this is still a valuable tool in development. In the case of separating the activity from the surface view, there were no problems, and the branch was successfully merged to the master branch, updating the repository with the changes (Fig.4).


Commits on Feb 23, 2017

Fixed Joystick bugs, and integrated Joystick class for player movemen...

…t. Successful in separating Surfave View from the Game Activity

Zoa84 committed on Feb 23

Commits on Feb 17, 2017

Testing seperating surface view from Game Activity

Zoa84 committed on Feb 17

Fig.4 – GitHub commits showing when development in a separate branch started, and when it was merged to the master branch

2.2.3 Transferring a scene from Maya and positioning objects

For creating a scene, Autodesk Maya was used to create objects and position and rotate them to create the desired scene. The translations and rotations can then be taken to the project, and placed in the set position and rotation. Appendix 5.2 shows how a scene created in Maya can then be drawn through the game. This also includes the objects translations and rotations, which can be applied to an object in the code. A problem with this was how the object loaded into the game. Fig.5 shows the world axis in Maya, with the arrows pointing in a positive direction. This is the axis the transformations are based on in Maya; however, objects loaded in jPCT came out rotated in X by 180 degrees. Any transformations would be relative to the upside-down object, following the world axis in Fig.6.

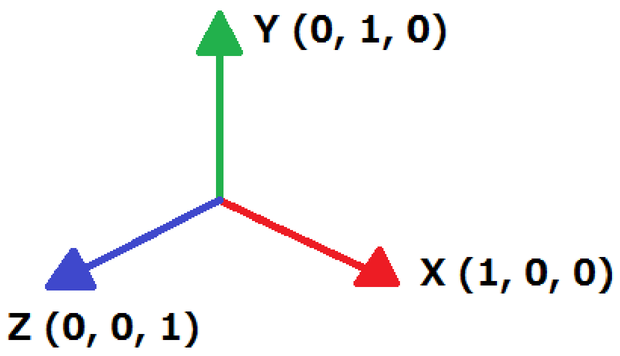


Fig.5 – The world axis in Maya. The arrows point positively

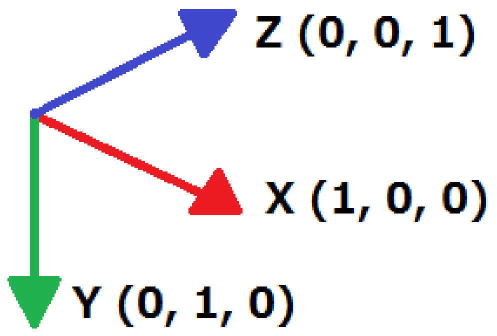


Fig.6 – The world axis in jPCT. Important to note that objects are loaded 180 degrees in X, making this axis the same as Maya's but rotated

To rectify this, a class called TransformFix was created to rectify any transformations to jPCT transformations. Whenever an object is loaded, the ObjectLoadFix() function will be called, rotating the object and the objects mesh 180 degrees in X, and clearing the rotation, effectively setting the object in the correct rotation, without actually changing the objects rotation. This, however, kept the jPCT world axis from Fig.6, meaning the Y and Z transformations were reversed. Using the TransformFix class, further functions were added, which can be called when translating or rotating an object, to transfer the transformations from Maya to jPCT axis (Appendix 5.3).

This rectified all the problems from loading the desired objects in jPCT, but also rectified any future objects added to a scene, notably when development shifted focus from the game engine to the game itself during the later stages.

2.2.4 Joysticks and Touch-sensing

To emulate the joysticks you would have on real controllers, virtual joysticks were created, which can be used to move the player, and for the player to look around. The Joystick class works by loading textures in 2D, and draws them over the 3D game as a HUD (Heads up Display) similar to the buttons from the 'Interim Deadline'. These are made up of two parts: the joystick background, which does not move, and the joystick itself, which will move around the joystick background depending on the player's touch. Next, some functions were created within the Joystick class, which would limit how far the joystick detection goes, and would detect the distance from a finger to the centre of the joystick, and use this distance to move or rotate the character. The hard part of this was sensing more than one finger on the screen, as both joysticks need to be usable simultaneously.

When sensing touches in Android Studio, the touch event returns various numbers to symbolise the different types of actions, such as detecting when a finger touches the screen, if it is moving around the screen or if a finger has been released. This does not include if a finger is being pressed down on the same position, causing the joystick to not recognise a stationary finger. This would cause the player to stop moving or turning, even though an input is being passed to the joysticks. To fix this, an integer with different states was used, so it would only turn off when a finger was released. This worked for both joysticks; however, during testing a player would still be unable to use more than one finger on the screen. This is because in Android Studio, subsequent fingers are not tracked by the conventional actions explained above, but instead by an action pointer, which gives an index to the subsequent fingers. These subsequent actions also do not have a move action, meaning they could not detect movement from subsequent fingers, and would not update the position of the joystick in the same style as the first finger.

1st Finger	Subsequent Fingers
ACTION_DOWN	ACTION_POINTER_DOWN
ACTION_UP	ACTION_POINTER_UP
ACTION_MOVE	No Move for pointers

Fig.7 – Inputs from the touch screen. Only the 1st finger has a move input

Fig.7 shows the inputs used from the touch screen. With the difficulty of using two joysticks, there were alternatives to consider, such as only using a single joystick to move and rotate the character (similar to the early '*Resident Evil*' games), or by changing one of the joysticks to a Directional-Pad, which would only detect moving forward, backward, left and right. However, after searching around the topic, several topics with roughly the same problem were found, each with different workarounds, and a mixture of answers was used to create the answer for this project. Albeit a bit messy, the fix works properly, and both joysticks can be used in the desired fashion.

Another problem which came up with using the joysticks was switching the 'first' finger. If two fingers are on the screen, and the first finger which touched the screen is lifted, the second finger becomes the 'first' finger, and will use the first set of inputs. Because of these various problems, the 'touchEvent' function in the GameSurfaceView class is a bit messy, but it has fixed most of the issues surrounding two inputs.

2.2.5 Text and the Text Buffer

In the ‘Interim Deadline’, text was drawn as a texture, loaded and displayed alongside the HUD. Although this would be fine for a game that only uses a few different words, this project will need to use various texts at various points. Using jPCT-AE, text can be drawn as multiple textures, where a font or an image of an alphabet is loaded, and separated into multiple images of single characters. When drawing text, the string parameter passed is read character by character, and drawn to the screen using the position of the previous character. By then getting the width of the line of text, the text can be positioned relative to the position passed, aligning the text.

With a basic text class, the Text Buffer class was created, used as a messaging system from the game to the player. This is mainly used when the player is interacting with objects in the game. This works by sending text (from the Surface View) to the Text Buffer class, which then orders the text and draws them to the screen. After a set amount of time, the text is removed from the buffer, and the remaining text is also reordered. This is important for the player, as it allows them to grasp what is happening in terms of the gameplay. Without it, the player would be unable to complete the puzzles. When testing this class however, there were multiple glitches and bugs, such as when the four slots of the messaging system were used, text being deleted too early, and some text staying instead of being deleted. This was mostly due to the method of deletion, which used a countdown timer where upon completion, the text would be deleted. If, during this time, the text is replaced by other text, this countdown would instead delete the new text.

Looking at this problem programmatically, the Text Buffer class is similar to a queue, a FIFO (First-In First-Out) data structure where data (the messages) is added to the back of the queue, and is deleted from the front of the queue. The difference here is the Text Buffer also has a function to delete all the messages, used when travelling to another floor, as old messages should not be kept from the previous floor. As the countdown timers cannot be stopped or paused, they interfere with messages on the next floor, effectively breaking the queue system. A few different fixes were tested to rectify this, but as this was done relatively late in the timeframe, a basic message system was implemented instead, where messages drawn are kept indefinitely, and are only deleted when moving to another floor, or when a new message is added. This removes the glitches, but also simplifies the messages to a less impressive state.

2.2.6 Asset Management

Asset management in Android Studio can be difficult, as certain assets should be separated, yet some must be kept together. In Android Studio, all images and textures are separated from the other assets, and fall under the category of 'Drawable'. Assets under this category are recognised by Android Studio as images for the game, and uses specific functions relating to 'Drawable' which allows us to read and use these images.

For this project, there are a few different types of images used. There are background images for the menu or options, 2D textures for the HUD or inventory, and 3D textures for the objects; however, you cannot create directories within 'Drawable' to sort these. There are some workarounds, including not using the 'Drawable' directory and using different directories; however this also changes how to load the images, as Android Studio uses its own functions to reach the 'Drawable' directory. Instead, the images are named slightly differently to differentiate them.

Objects are loaded with an object name (such as 'chair' or 'table'), and these same names are used for the images, allowing the same name to be used when loading the textures. This simplifies creating new objects in the game, as a texture name does not need to be specified. Other images are used as 2D images, and are given the tag 'img' before their name, which will stop any conflicting objects trying to load the wrong image as a texture. While images are separated to the 'Drawable' directory, other assets are sorted into different directories from the 'assets' directory. This includes music, sound and objects.

2.2.7 Memory Management

As the platform for this project is a mobile phone, there are many limitations to consider. As well as the physical limitations of a mobile phone – namely the absence of peripherals and controls – there are also the computational limitations. A phone has far less computational power than a computer, meaning memory management, and management of assets, is far more important. At early stages of the project, the size of objects or textures was not a large problem, but as the project progressed, there were signs of the app running slower than usual, and eventually there were crashes pertaining to lack of memory.

With Android Studio, you can run apps and monitor how they are running, including information on CPU and memory usage, as well as debug errors encountered during testing. By using these monitors, the origin of memory errors could be located. From Fig.8, it was clear that memory usage was too high as soon as the app started. Research into memory management for phone games revealed that the main cause of memory issues were from the assets, specifically the size of textures and objects. On checking the textures, there were a few large textures which could be smaller, as well as file types which used up more memory. The same image as a .bmp (Bitmap) file used up more memory than a .png (Portable Network Graphic), so they were all normalised to .png, without any loss to quality or transparency.

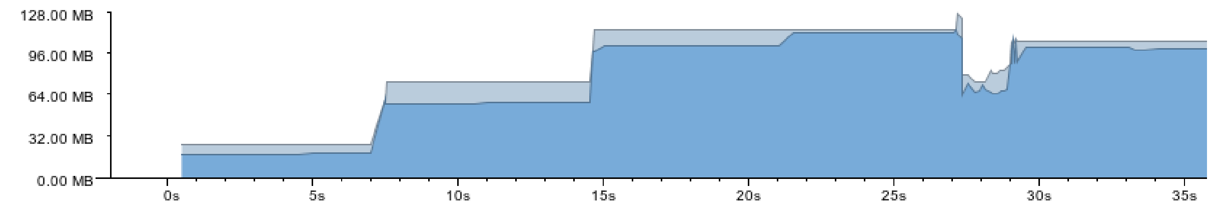


Fig.8 – The memory usage of the app. The spike at 7.5 seconds was from entering the Menu Activity, and the spike at 15 seconds shows entering the Options Activity. Upon reaching 128MB, the app will crash

Large textures were also made smaller where possible. This is a problem often encountered in games creation between programmers and artists. An artist will want to put in as much detail as they can for a texture; however, a programmer will want to keep everything as simple as possible to make it run faster. The textures must also be high enough quality for the player to see, especially if they contain text. As an example, the background image for the Menu Activity was a 1080x1080 pixel .bmp file, which used 3.33MB. This was reduced to a 270x270 pixel .png file, only using 60.6KB, approximately 1/50 of the original image size.

Among other image changes, these all made a big difference, as seen in Fig.9; the memory usage has dropped significantly. The memory usage rises on certain events, such as starting the game, but this is to be expected as objects are loaded in.

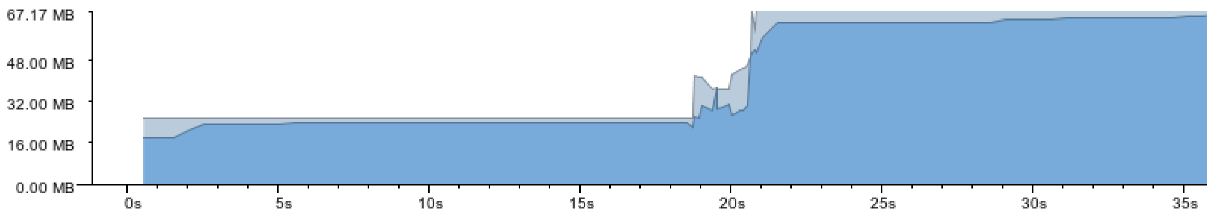


Fig.9 – The memory usage of the app after texture changes. The spike at 20 seconds was from entering the Game Activity. Note the scale difference from Fig.8, approximately half of the scale

2.2.8 Instruction Activity

During a supervisor meeting, where parts of the project were shown, it was pointed out that instructions should be added for the player. Although not originally planned, this is an essential part of any game, as it should not be assumed the player knows how to play or what they should do. To add this, the original activity layout was changed to accommodate an Instruction Activity, which can be accessed from the Menu Activity (Fig.10). This is a simple screenshot of the game, which explains what each button does, and how to use the inventory.

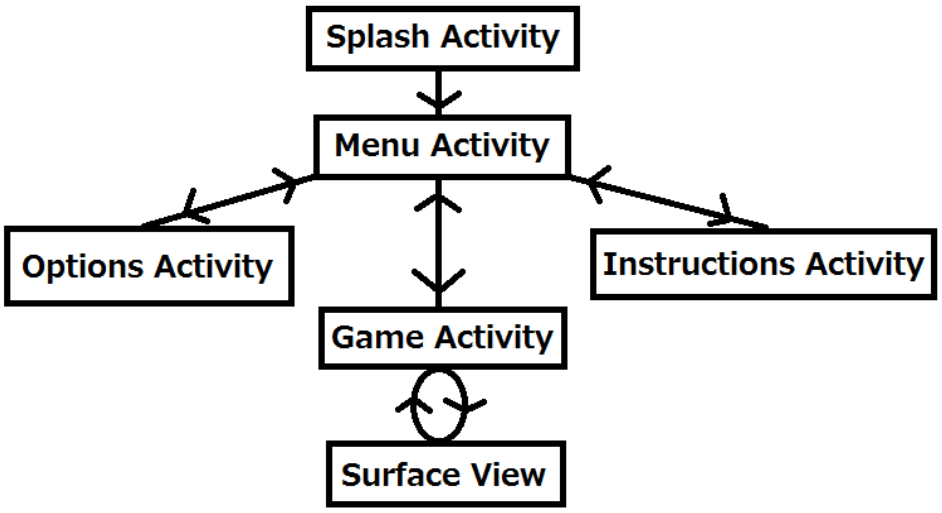


Fig.10 – The new activity flow. The newly added Instructions Activity is accessible via the Menu Activity

Although a relatively minor change in the diagram, as this new activity does not concern the Splash, Options or Game Activity, having to integrate a new activity in the middle of development took a noticeable amount of time, especially compared to how easy it was to create all the activities at once near the start of this project.

2.2.9 Changes to original project spec

During development of this app, the limited time became a concern for the project. The original plan from the 'Interim Deadline' included creating several floors as part of the game, including narration and multiple puzzles. As the deadline came closer, it became apparent that the project would not be able to have all of the features originally planned in the specification. To ensure a suitable amount of time was left for testing, playtesting, and for the final report, the game was shortened to allow more time on other parts. This included shortening the overall game to five floors (Ground floor to fourth floor) and thus limiting the puzzles originally planned. The story was also kept brief, to allow more time for the gameplay and mechanics.

Although this change affected parts of the project, the fundamentals were kept to show the potential of the game. The overall game engine was still implemented, showing creation and displaying of a 3D environment, as well as moving and interacting with objects in the room. Items can be collected to an inventory, then used where necessary to progress the game, and a method of winning or losing is also available. Although unfortunate, this allowed the report to be more in depth, giving more time to produce a well written report as opposed to rushing the report for the sake of the software.

2.2.10 Changing floors

The game has a total of five floors (a Ground floor and floors 1 to 4) which the player can move between. Each floor is a class with objects to load, each inheriting from a base class called Floor. Each class that inherited from Floor would have functions required in all scene classes, such as an object loader and garbage collector. There are also hidden floors, used when the game is completed. To move between these floors, a finite-state machine is used, allowing the player to move between floors properly, and not move to certain floors they should not reach. From Fig.11, after the Game Activity begins, the player can move between floors using stairs, which can go one floor higher or lower, and by using the elevator, which is limited to floors 2, 3 and 4. Floors 4 and 1 contain traps which can lead to a game over, and the Ground floor has the win condition for the player: escaping the building.

With the use of a finite-state machine for the different floors of the game, it becomes noticably easier to debug any bugs and glitches, as the diagram clearly shows which floors are accessible from any floor, as well as which floors are not accessible by each floor. If, for example, the player could move from the 2nd floor to the Ground floor, by checking the state machine there is no direct link from the 2nd floor to the Ground floor, and the problem can then be traced within the 2nd floor.

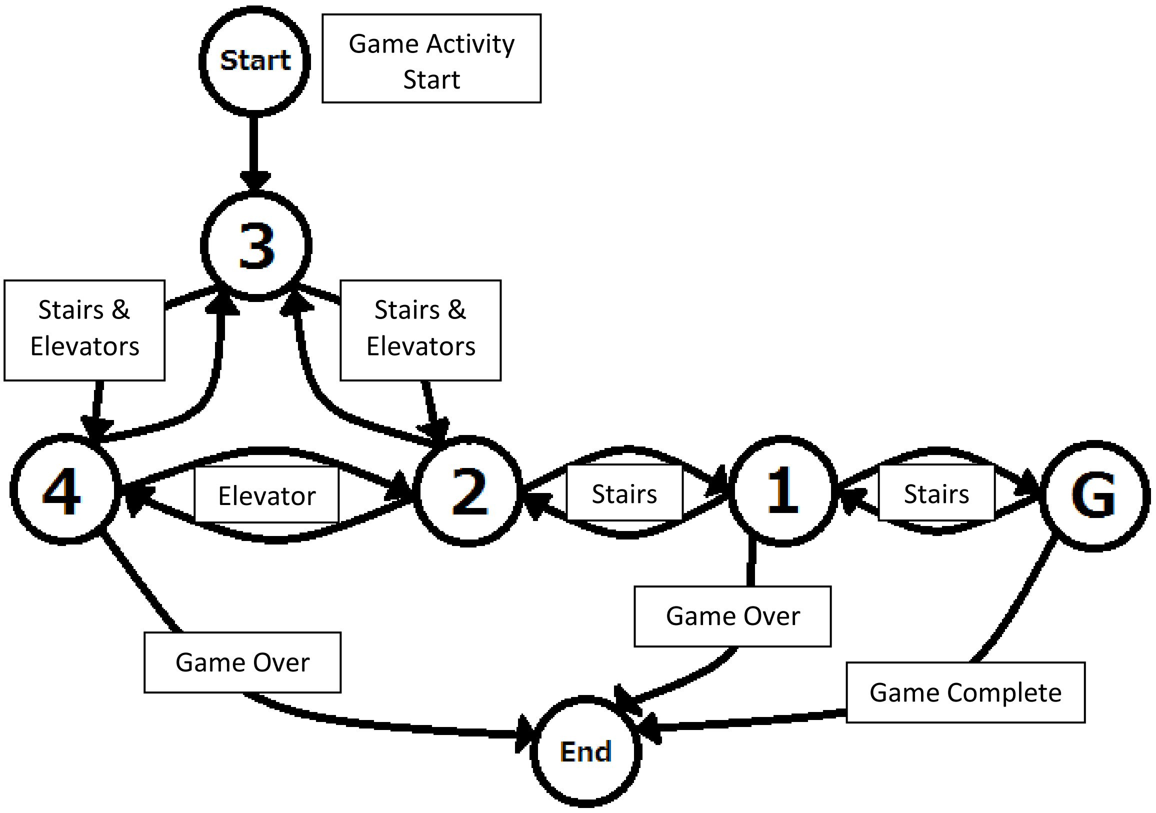
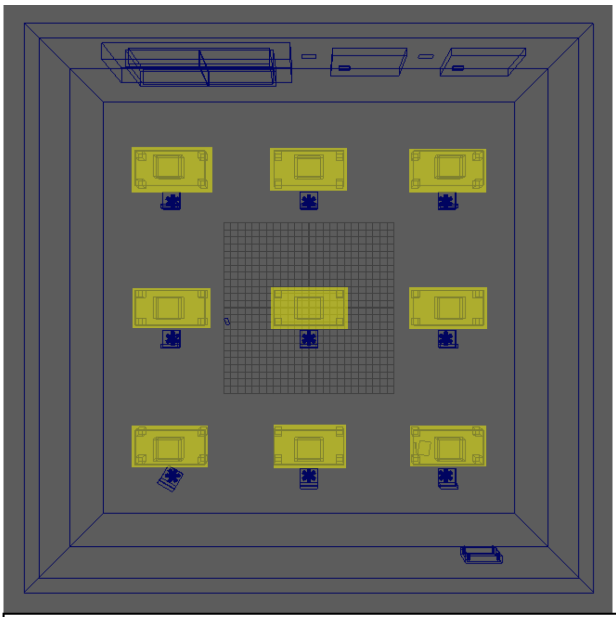


Fig.11 – A Finite-State Machine diagram of travelling between floors. The elevator is only usable on floors 2, 3 and 4, and stairs are usable between floors

2.2.11 Collisions

For collision detection, the game uses a Collision Map class, which can be passed collision areas the player cannot enter. When moving, this class is called, and it checks if the player will be colliding with anything in the next frame. If this is true, the player will move back to where they were last frame, effectively not moving. On the walls, where a collision is in either X or Y, these can be checked separately, therefore the player can 'slide' along the walls, whereas objects in a room, which have both their X and Y checked at the same time, cannot be 'slid' around. As the player cannot jump or otherwise move in the Y axis, the collisions can be checked as if they are bounding boxes, checking if a position is colliding an object in one axis, and then the other. In Fig.12, the floor it is showing in 3D is converted to a 2D collision map, which the player must move around.



```
//Add collisions to the collision map
Collisions[0] = new CollisionMap(-20, 20, 5, 3);
Collisions[1] = new CollisionMap(-20, 0, 5, 3);
Collisions[2] = new CollisionMap(-20, -20, 5, 3);
Collisions[3] = new CollisionMap(0, 20, 5, 3);
Collisions[4] = new CollisionMap(0, 0, 5, 3);
Collisions[5] = new CollisionMap(0, -20, 5, 3);
Collisions[6] = new CollisionMap(20, 20, 5, 3);
Collisions[7] = new CollisionMap(20, 0, 5, 3);
Collisions[8] = new CollisionMap(20, -20, 5, 3);
```

Fig.12 – The floor in 3D (Left) shows how a top-down view can change a 3D scene to a 2D map, with the yellow boxes representing the collision maps from the code (Right)

The collision maps works by passing a position (in X and Y, with the centre as the origin) and a size in X and Y. With simple AABB (Axis-Aligned Bounding Box) collisions, although they are not very high level, and physics would be difficult to implement on top, since this project is developed for a mobile phone platform, the level of collisions can be simplified. Higher level 3D object collisions would have to consider all three vectors, and the rotation of each object colliding. Especially if this is then checked for every object, the computational power required would go up significantly.

3 Testing

A key component of this project was the testing of software, including during development and after getting feedback from testers. Testing was a key part of development, especially between tasks, and when integrating to the online repository. Much of this type of testing was documented in the Development, specifying problems found during implementation, how they were resolved, and any further problems in testing post implementation and fixing these before continuing with the next task.

3.1 Test Cases

Test cases were prepared to test the software at version 1.0 as described in the GitHub commits (Appendix 5.1). This is a version where most of the desired mechanics for the game have been implemented, and testing here would allow enough time to fix any bugs or errors, along with adequate time for cleaning the code and making sure the code is well commented. This version will be improved upon based on these tests before more in-depth playtesting takes place. These test cases can be accessed under the 'Appendix' under the section 'Test Cases', with any relevant screenshots or diagrams accessible from the 'Appendix' under 'Test Diagrams'.

Many of the test cases were completed without any complications, although there were a few exceptions, some of which were glitches on the original plan, and some were personal preferences. An example of a personal preference change was the Seek bar widget test in Test No. 7. Although the volume was properly adjusted to the desired input, this widget was still usable while the check box was set to false. This meant the volume could be adjusted, even though it was not being used. Based on personal preference, this was changed to become disabled while the checkbox is false (Test Diagram 16).

A glitch which needed fixing was the sound effect being played twice when changing the orientation using the images (Test No. 4). Pressing the radio button successfully played the sound effect and changed the orientation, as would pressing the image, except the sound effect would play twice. On inspecting the code, this is caused by the function to change the orientation being called twice by pressing the image. When pressing the image, a function to change the radio group is called. This sets the correct orientation; however, this called another function to check the radio group twice due to the number of radio buttons in the radio group. Instead, a function to change a radio button is called – setChecked() instead of check() – which only affects one radio button, and the programming will handle any changes to the radio group (Test Diagram 17). This still allowed the sound effect to play when pressing the radio button, and fixed the sound effect playing multiple times from the image presses.

Another glitch was the joysticks in-game, specifically when a finger would leave the area the joystick would track (Test No. 11 and Test No. 12). Leaving the tracking area would keep the joystick in the last tracked position, causing the character to move or turn in the same direction. Comparing this to a conventional physical controller, moving too far from the joystick would rebound the stick to its original position. To emulate this, the code was changed slightly, with the joystick being reset to the starting position when the finger moves out of the joystick range.

The last problem found from the test cases was the volume of the sound effects (Test No. 17). Although they were properly adjusted by the volume set in the Options Activity, each individual sound effect has its own volume, with some sound effects playing too loud, and some sound effects playing too quietly. To fix this, Audacity was used, an open source audio-editing software, which can be used to edit the sound effects. Editing the sound files is relatively easy, dragging the amplitude up or down to increase or decrease the volume (Test Diagram 18). However, by equalising the sound effects, they were now all too quiet compared to the music. To fix this, either the sound effects can all be amplified to match the music, or the music can be de-amplified to match the sound effects. As increasing the amplitude of sounds or music can lead to a loss of quality, the music was de-amplified, fitting the volume of it to the volume of the sound effects.

3.2 Playtesting

As well as individual tests taking place to make sure the different parts of the game are working correctly, an overall playtest, playing through the game multiple times, was done to test the combination of each of these components as a whole. This is extremely important, as individual working parts do not necessarily work together, and errors and bugs can occur. Playtesting can also be done by other people, finding parts of the game to improve upon which a creator or designer may easily miss.

During the playtesting of this project, individuals played through the game, giving their thoughts and comments on the game, as well as areas that could be improved. General notes were made from people's opinions, and detailed here as a general overview of feedback from players.

The biggest concerns were in the difficulty of the game, or the navigation of the game. For example the note on the fourth floor was quite hidden in the corner, causing players to easily miss this and reach the first game over state. As the game is not made to be challenging, the position of the note was changed. Instead of in the corner, it is now placed next to the door the player comes out of, making it easier to find. In terms of navigation, players would get lost while moving between floors. Although the floor they went to would appear in the messaging system for a short time, they could easily miss it, or quickly forget which floor they are on. To rectify this, a sign was added to each floor with the floor number, similar to how real offices may have a sign to show the floor number (Test Diagram 19).

3.3 Changes from tests

With the testing complete, the project was revised to fix any errors, bugs and glitches found, and to update the commenting on the code to increase maintainability. This is important in case the project is continued or re-worked, especially if it is re-worked by a different programmer. The project also underwent some graphical changes, making it a more aesthetically pleasing deliverable project for the final deadline. This includes changes to the HUD, the UI, the menu graphics and the textures in-game (both 2D and 3D objects).

The UI used the default textures for buttons, seek bars, check boxes and radio buttons. The textures for these are mostly fine, except for the button texture, which is very dull and basic. Similar to the textures for the background, they can be created and set within the XML, which will automatically stretch them to fit as a button (Test Diagram 20). These give the game a more finished feel. Focusing on the HUD, other phone games with virtual buttons and joysticks were reviewed, and many followed the theme of grey, semi-transparent buttons, using symbols or objects to symbolise different functions, such as a cog wheel to symbolise options or settings, and a hand to symbolise using or picking up an item.

With the textures in-game, many of them were simplified to quickly get the game to a playable state, and these were improved. This includes adding more detail to some textures, and also changing the size of textures to make them smaller, especially if the associated object is fairly small, such as the screwdriver texture, which was reduced from 128x128 pixels to 64x64 pixels. Some objects, such as the doors and elevator, were also changed slightly, to scale with the character and other objects in the room. With these changes, the final version of the project, dubbed version 2.0, was now created. Appendix 5.7 shows screenshots of the final developed version. Lastly, an app icon was created. This is used to define the app alongside other apps on a phone screen (Fig.13).

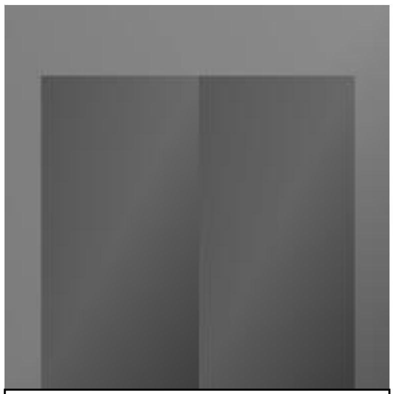


Fig.13 – The app icon for the game

4 Conclusion

4.1 Product Evaluation

As a whole, the finished product showed the potential of 3D game development using Android Studio, and gave me insight into the challenges concerning 3D game development for mobile phones. The game itself has deviated from the original idea presented in the 'Interim Deadline', especially in terms of the story and length of game. On reflection, the proposed idea had a strong focus on the story to convey to players, whereas a focus on gameplay mechanics would have been more suitable for this project.

I was satisfied with the development of the activity flow, and of the base engine and the classes created for the engine. These classes can be reused in other projects, or rebuilt as classes in other languages for different projects, for example the Media class, which can play loaded music and sound effects. Using the game developed from the IMAT 2608 Mobile Games module, the method of playing music in that was fairly crude, and this class could be reused to replace and improve that game.

Given more time, there are various changes that could be made to improve the game. In terms of the mechanics, the biggest improvement would be to change the collisions. jPCT-AE does have functions and features to implement collisions between objects. This is done by having an invisible mesh to test for collisions, using primitive shapes such as cubes or spheres. They work independently from the objects loaded, but can be moved with these objects for collisions. Due to the nature of the game, collisions were kept simplified to focus on other parts, using simple AABB collisions, which would also use less computational power. The type of game also limited implementing physics, which were not necessary as part of the game, but could be developed for the engine for future products.

Other improvements include the aesthetics, such as the level of detail of objects and textures, and improvements to the music and sound effects. The aesthetics are what draws players in, which makes it an important part of development, but before that, the integration of the story and the length of the game should also be improved. From the 'Interim Deadline', the original plan for the game had to be changed due to the limited time. The integration of the story with the game was also heavily dampened because of this. With more time, these areas could all be improved upon, making a more complete game as opposed to a demo.

4.2 Evaluation of approach

The approach to the project in terms of the development had varying success. For project management, the use of source control with GitHub proved useful, even as a solo project. As well as a method of overseeing the project development (especially with the commits to the project), I was also able to make use of the repository's functions, specifically the use of branches for separating the surface view from the Game Activity. Although I was mostly sure that the two could be separated, I was unsure if I would lose access to certain functions or features; so having a branch to separate and test this before merging to the master branch was extremely useful.

Following the software development life cycle, all of the tasks for this project were first planned before implementation, such as how a class will be structured, what kind of functions it would need, then it would be implemented and tested. Only when the tests went as planned, and any known bugs were fixed, would the implementation be pushed on GitHub to the repository. From experience in working in groups, if there are any known bugs, especially those that would break the project, then it should not be committed to the repository. Even from various talks over the course, I heard of problems where programmers would push broken code at the end of the day, causing problems for others the next day. With this in mind, I made sure there were no problems with the code, and in some cases, I would leave code until the next day to double check it was fine before committing.

Reflecting on the method of development of the game, the main concern was the order I developed the project, with the early stages focusing on the design of the overall software, as opposed to development on the gameplay and game engine. This method of development put too much focus on the design and appeal of the project. Over the course of the Computer Games Programming course, I have participated in Game Jams: 48 hour sessions where small groups are formed to develop a game following a theme. These gave me an insight into how a company or group of people can develop a game from the early stages. They focused on the development of the game mechanics rather than the development of a menu or flow of activities. Taking this view to my project, too much time may have been spent on the design aspects, such as the flow of data and activities, and the creation of assets such as textures, objects and music.

If attempted again, the key point of improvement would be the priority, focusing on the game engine and mechanics. With my focus on the development of a whole game as opposed to the gameplay, I was mostly worried about potential problems with the design of the application if I had completed the game first, for example when adding the Instruction Activity near the end of development, which would be easier at the start of development.

In terms of the development time, comparing with the project plan outline from the project contract, more time than planned was required for the development of the project. Comparing the project plan from the 'Interim Deadline' with a Gantt chart of the actual time taken (Appendix 5.4), there are a few key differences.

The research and literature review turned out to require much more time than anticipated, especially the research, as this was constantly ongoing with the prototype development, and even for the development of the final version, much to do with learning jPCT-AE while creating the game. With the literature review, a lot of time was spent on reading articles and papers relating to the review, specifically areas that interested me, such as the Horror genre and the studies taken place surrounding the Horror genre.

Much time was also spent on the prototype development, although the Gantt chart makes this difficult to perceive. The story and resource creation actually used a lot less time than anticipated, most likely due to creating the story in a concentrated period, and the resources were kept minimal for the 'Interim Deadline', but were also created during development of the final project file. There were drawbacks to leaving asset creation this late, as it interfered with potential time spent on project development. On the other hand, this allowed me to know exactly what assets were needed at a point in time, and I could tailor it to the current requirements. For example, during the final development, it became clear that keeping memory efficiency was important, and part of this included designing 3D models with a low polygon count to improve performance.

A presentation for the 'Interim Deadline' was also no necessary. Instead, regular meetings with my supervisor allowed me to show what had been done both in terms of the write up, but also of the actual software completed, with examples of the functionality displayed using my phone as a platform. Development for the final deadline portion of the project plan was mostly similar. The 'Deliverable Project File' took an extra week. However, there was a week or two where development was halted for testing and writing the final report. Due to the nature of Gantt charts, any split in development could not be shown.

Both these Gantt charts are also inaccurate in terms of gauging time lost during holidays, personal matters or time spent on other coursework throughout the year. Although this is hard to measure for the project plan, the actual project time could show this. A daily report during project development would give a much greater insight into any problems arising during development; however, this approach can be long-winded, and a focus on keeping to schedule can hinder development on key areas that require more time spent.

4.3 Evaluation of tools

In terms of the choice of tools to develop this game, there were a number of positives and negatives in each of the tools, as well as using the tools together.

Coming from the IMAT 2608 Mobile Games module, I have already had experience with Android Studio and Java, albeit for developing a 2D game. During development of 'Annihilation Intelligence', the architecture of Java had both good and bad points. Especially as the course has a heavy centric on C++, there were different features I could not use, such as the use of pointers. Java always passes types by value, as opposed to C++ which can pass pointers and references. Although this led to problems, such as being unable to pass a media player between activities, there were workarounds, which were sometimes easier than a C++ method of implementation, for example the media player became a singleton class, which was easier to use.

Another big difference was the garbage collection. For this project, there was a problem with the destruction of objects and data, specifically when the game ended, and the data and objects from the surface view had to be deleted. Although garbage collection should be automatic in Java, there were errors loading the Game Activity a second time in the app, as some variables were not cleared and incorrect data was kept from before, causing the game to crash. With the excess memory usage, there were also problems with running out of memory, as data was loaded to new memory locations as opposed to the previously used locations. To fix this, I manually cleaned up the data, setting variables to null before the activity was finalised. In C++, although I would have more control in destroying objects, I would also be responsible for the destruction of all objects, meaning I would have to pay more attention to all classes for proper garbage collection.

Another evaluating point of Android Studio is the use of activities, and the flow between these activities. They provide a method of separating an application, making it easier to manage from a development standpoint, as well as to debug, as an error can be traced to a single activity. If, for example, the game was developed using Game Maker Studio, the use of rooms can simulate activities, yet there would not be the interaction between activities as we have in Android Studio, such as accessing functions or data between the activities and the surface view. This was extremely useful as functions would not need to be rewritten in different activities, and data can be more easily accessed, as Game Maker Studio cannot access data in rooms.

jPCT-AE was the main tool for 3D in this project. There were a number of pros, such as simplifying certain parts like loading objects and textures, while still giving me control in the use of the loaded objects. Comparing with development tools such as Unreal Engine, I could manually control the objects and meshes with code, whereas objects in Unreal can be simply placed onto a scene, and have attributes changed. Albeit easier to manage, as well as having an editor to view the scene, another key difference is the size of the engine. As this project is for a mobile phone, Unreal may have a larger number of options, but they were not necessary for this project, and instead would lead to problems with memory.

But there were difficulties in using jPCT-AE, such as its limited user support. As it is not as vast as Unreal Engine or Unity, it was difficult to find answers to any problems I had. Even though there is documentation for the functions of jPCT-AE, troubleshooting for more widely used engines would have been much easier, and answers could have been found within the editor for Unreal and Unity. An example of a problem I had was with the world axis that jPCT-AE used. As explained in the development, jPCT uses its own world axis, different to the axis used in Autodesk Maya, causing objects to be positioned and rotated differently. Although this was eventually fixed, the problem would have been easily solved using an editor, where we can see how transformations affect objects, or if the user support and troubleshooting base was larger for jPCT-AE.

With the simplification of certain parts pertaining to the use of 3D, another part of this is the use of shaders and lighting. Although jPCT-AE does have lighting, this is extremely limited, and cannot be used in the style used in the IMAT 3111 Shader Programming module. In this module, we have learned about the use of shaders as a method of calculating lighting, texturing, and even some advanced techniques to improve the aesthetics in a 3D environment. These methods however, cannot be implemented with jPCT-AE, as we do not have in-depth access to the shaders. On the other hand, considering the platform and limited computational power, it may be best that shaders cannot be edited. Although the effects of shaders are extremely powerful, we simply do not have enough power to effectively use it on a mobile phone platform. The option to use shaders however, would still be preferable. For example, Unreal Engine allows us to use textures and materials, and have more access to lighting than jPCT-AE, and this can be used in mobile phone game development, albeit sparingly due to the platform.

Having to learn to use jPCT-AE in a short time to produce a game was also a good experience, and is often the case once a person graduates and finds a job with a company. Being able to adapt and learn software in a short amount of time is an invaluable skill. This has also allowed me to evaluate between jPCT-AE and other methods of 3D development, and I personally preferred the features we had access to in other systems. Given a chance to work on this project again, I would consider using alternative software, using what I have learnt here to produce a better project, although I do see how jPCT-AE is definitely good for smaller games.

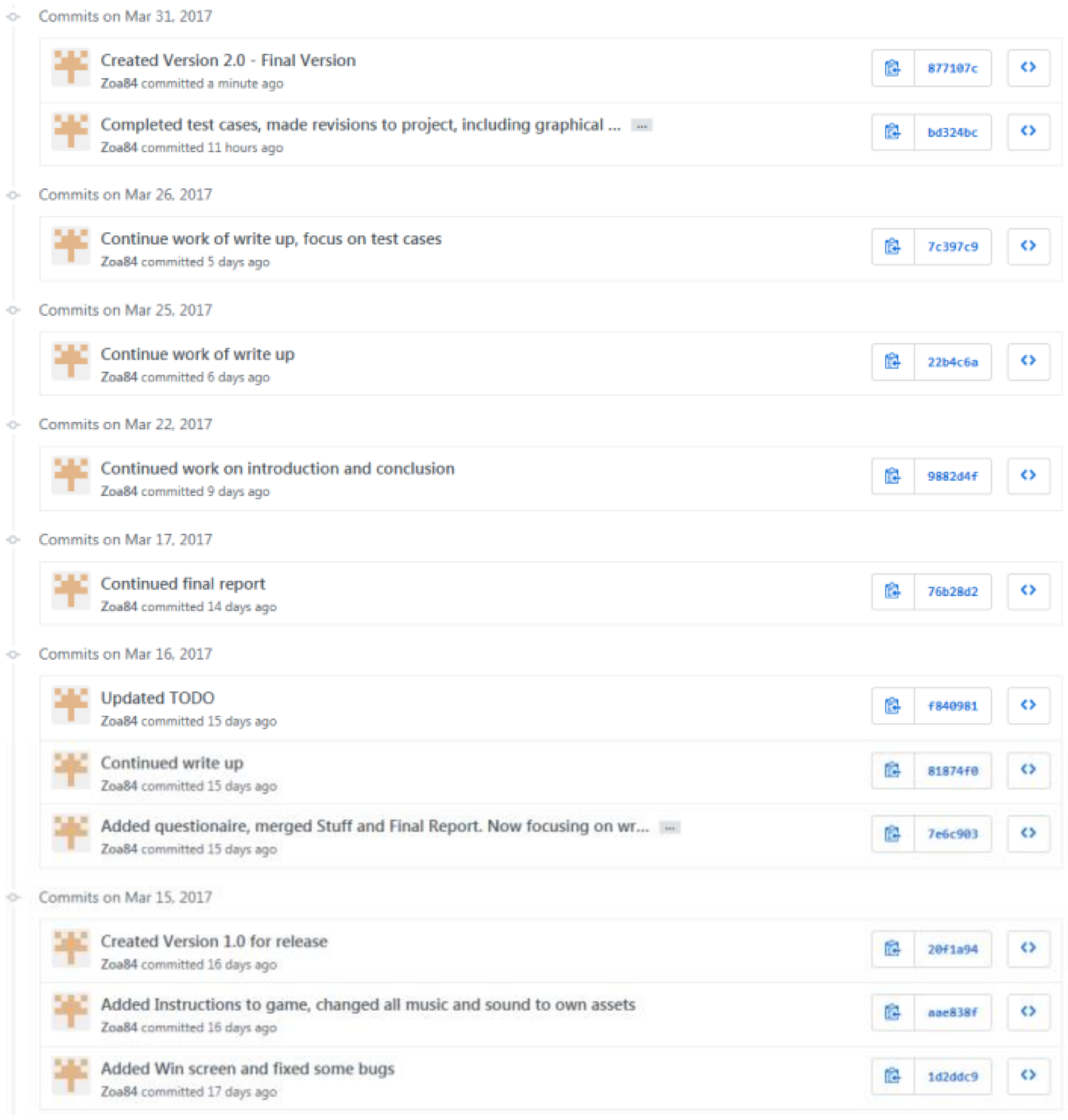
4.4 Final comments

Overall, it has been a good learning experience of having to complete a game under a time limit, and has given me time to reflect on how games companies can have the problem of completing a game under a time limit, and being forced to potentially under-deliver given a time limit. In fact the whole course over these past few years has changed my perception of games and games companies, and has made me appreciate the hard work which goes towards a video game.

To conclude, the project has developed a good base engine for the development of 3D mobile phone games. The functions and methods created can be used to create a game fairly easily, such as the simple method of adding an object to the scene using only a few lines of code. In fact the development of the game design –the multiple floors, the objects and interactions- were done towards the very end of the project period under a short time.

5 Appendix

5.1 GitHub Commits



Commits on Mar 31, 2017

Created Version 2.0 - Final Version
Zoa84 committed a minute ago — 877107c

Completed test cases, made revisions to project, including graphical ...
Zoa84 committed 11 hours ago — bd324bc

Commits on Mar 26, 2017

Continue work of write up, focus on test cases
Zoa84 committed 5 days ago — 7c397c9

Commits on Mar 25, 2017

Continue work of write up
Zoa84 committed 6 days ago — 22b4c6a

Commits on Mar 22, 2017

Continued work on introduction and conclusion
Zoa84 committed 9 days ago — 9882d4f

Commits on Mar 17, 2017

Continued final report
Zoa84 committed 14 days ago — 76b28d2

Commits on Mar 16, 2017

Updated TODO
Zoa84 committed 15 days ago — f840981

Continued write up
Zoa84 committed 15 days ago — 81874f0

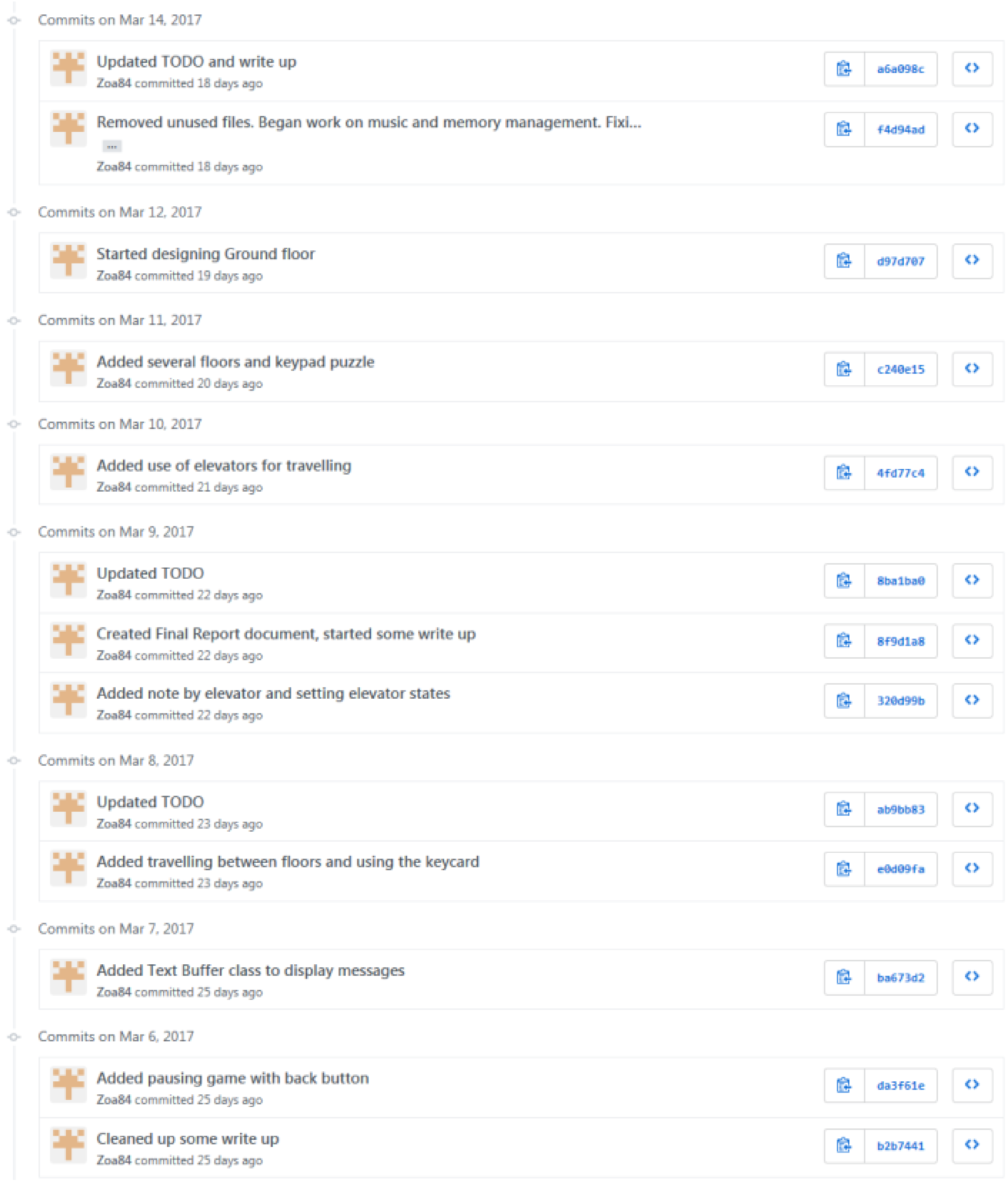
Added questionaire, merged Stuff and Final Report. Now focusing on wr...
Zoa84 committed 15 days ago — 7e6c903

Commits on Mar 15, 2017

Created Version 1.0 for release
Zoa84 committed 16 days ago — 20f1a94

Added Instructions to game, changed all music and sound to own assets
Zoa84 committed 16 days ago — aae838f

Added Win screen and fixed some bugs
Zoa84 committed 17 days ago — 1d2ddc9



Commits on Mar 14, 2017

Updated TODO and write up
Zoa84 committed 18 days ago
a6a098c

Removed unused files. Began work on music and memory management. Fixi…
…
Zoa84 committed 18 days ago
f4d94ad

Commits on Mar 12, 2017

Started designing Ground floor
Zoa84 committed 19 days ago
d97d707

Commits on Mar 11, 2017

Added several floors and keypad puzzle
Zoa84 committed 20 days ago
c240e15

Commits on Mar 10, 2017

Added use of elevators for travelling
Zoa84 committed 21 days ago
4fd77c4

Commits on Mar 9, 2017

Updated TODO
Zoa84 committed 22 days ago
8ba1ba0

Created Final Report document, started some write up
Zoa84 committed 22 days ago
8f9d1a8

Added note by elevator and setting elevator states
Zoa84 committed 22 days ago
320d99b

Commits on Mar 8, 2017

Updated TODO
Zoa84 committed 23 days ago
ab9bb83

Added travelling between floors and using the keycard
Zoa84 committed 23 days ago
e0d09fa

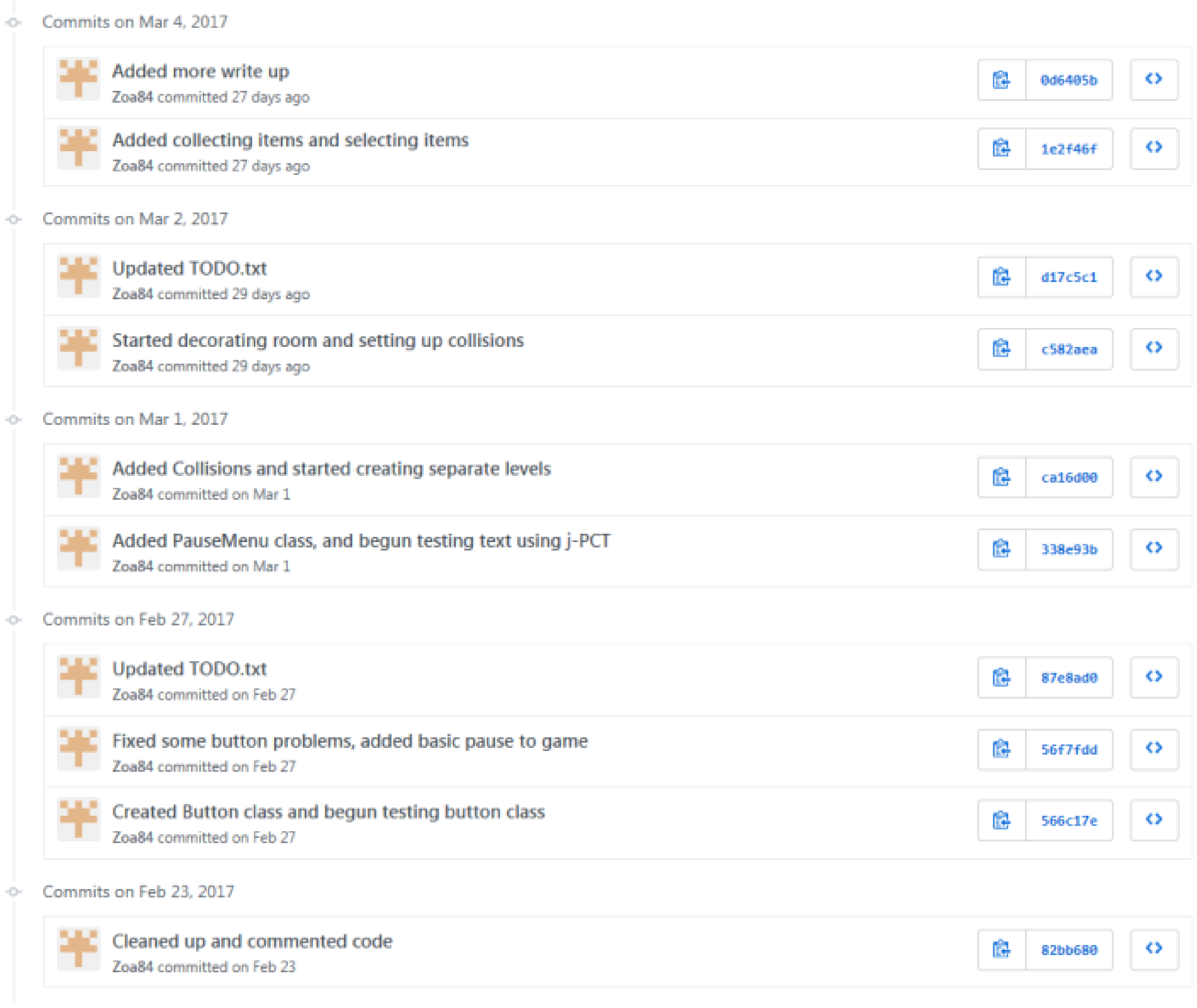
Commits on Mar 7, 2017

Added Text Buffer class to display messages
Zoa84 committed 25 days ago
ba673d2

Commits on Mar 6, 2017

Added pausing game with back button
Zoa84 committed 25 days ago
da3f61e

Cleaned up some write up
Zoa84 committed 25 days ago
b2b7441



Commits on Mar 4, 2017

Added more write up
Zoa84 committed 27 days ago — 0d6405b

Added collecting items and selecting items
Zoa84 committed 27 days ago — 1e2f46f

Commits on Mar 2, 2017

Updated TODO.txt
Zoa84 committed 29 days ago — d17c5c1

Started decorating room and setting up collisions
Zoa84 committed 29 days ago — c582aea

Commits on Mar 1, 2017

Added Collisions and started creating separate levels
Zoa84 committed on Mar 1 — ca16d00

Added PauseMenu class, and begun testing text using j-PCT
Zoa84 committed on Mar 1 — 338e93b

Commits on Feb 27, 2017

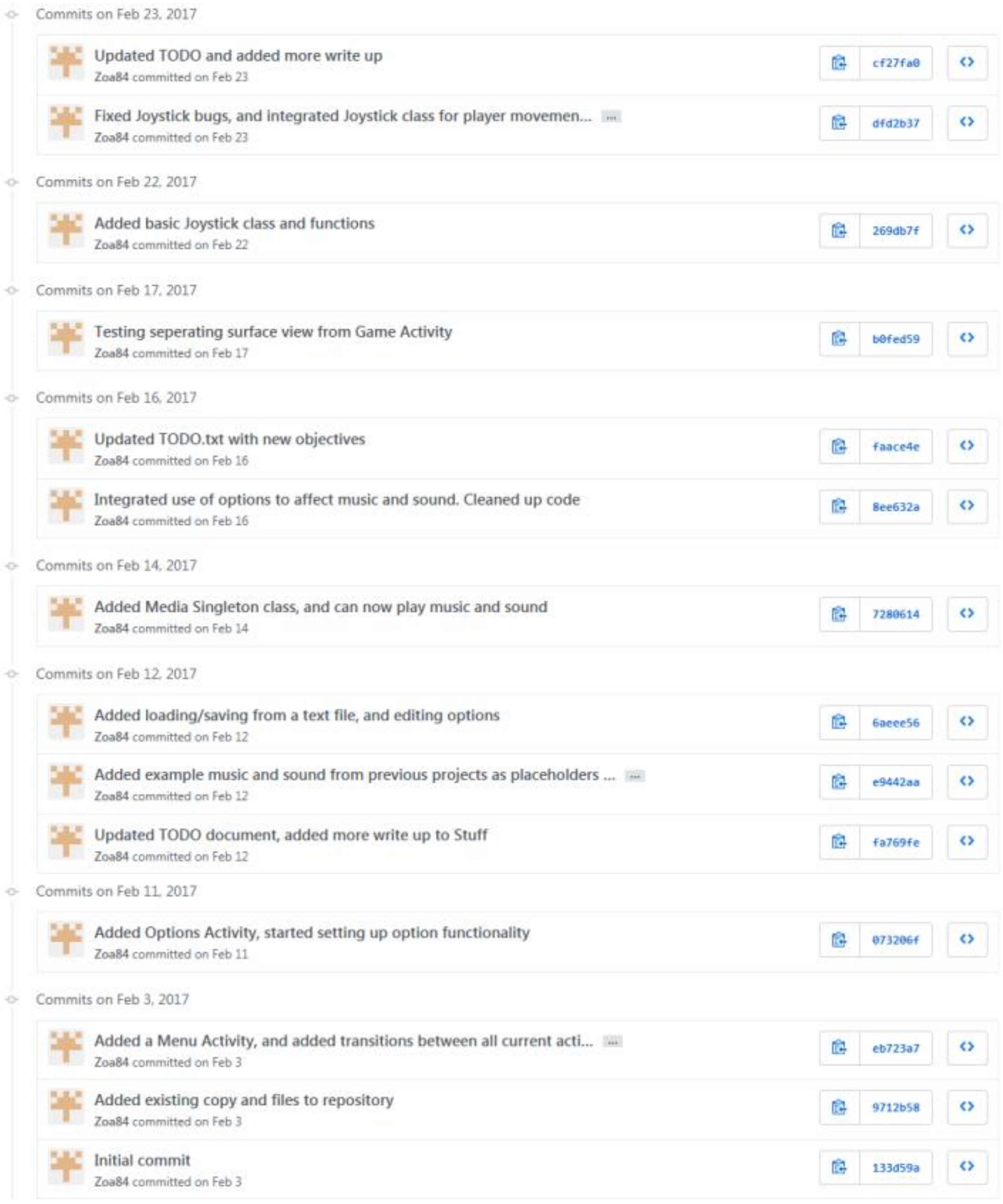
Updated TODO.txt
Zoa84 committed on Feb 27 — 87e8ad0

Fixed some button problems, added basic pause to game
Zoa84 committed on Feb 27 — 56f7fdd

Created Button class and begun testing button class
Zoa84 committed on Feb 27 — 566c17e

Commits on Feb 23, 2017

Cleaned up and commented code
Zoa84 committed on Feb 23 — 82bb680



Commits on Feb 23, 2017

Updated TODO and added more write up
Zoa84 committed on Feb 23 — cf27fa0

Fixed Joystick bugs, and integrated Joystick class for player movemen... …
Zoa84 committed on Feb 23 — dfd2b37

Commits on Feb 22, 2017

Added basic Joystick class and functions
Zoa84 committed on Feb 22 — 269db7f

Commits on Feb 17, 2017

Testing seperating surface view from Game Activity
Zoa84 committed on Feb 17 — b0fed59

Commits on Feb 16, 2017

Updated TODO.txt with new objectives
Zoa84 committed on Feb 16 — faace4e

Integrated use of options to affect music and sound. Cleaned up code
Zoa84 committed on Feb 16 — 8ee632a

Commits on Feb 14, 2017

Added Media Singleton class, and can now play music and sound
Zoa84 committed on Feb 14 — 7280614

Commits on Feb 12, 2017

Added loading/saving from a text file, and editing options
Zoa84 committed on Feb 12 — 6aeee56

Added example music and sound from previous projects as placeholders ... …
Zoa84 committed on Feb 12 — e9442aa

Updated TODO document, added more write up to Stuff
Zoa84 committed on Feb 12 — fa769fe

Commits on Feb 11, 2017

Added Options Activity, started setting up option functionality
Zoa84 committed on Feb 11 — 073206f

Commits on Feb 3, 2017

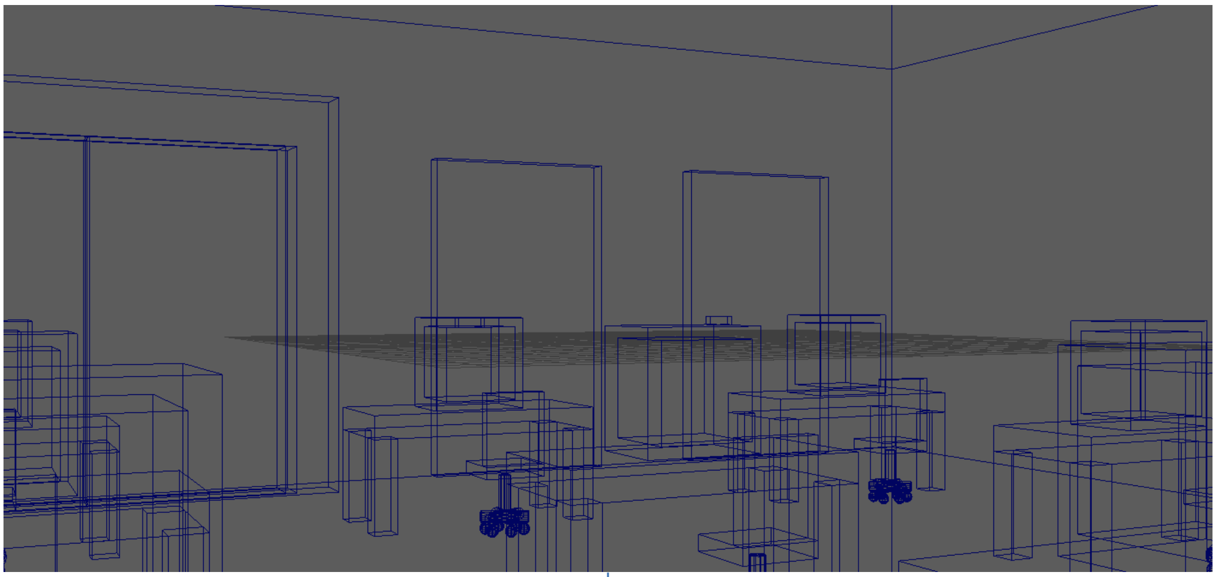
Added a Menu Activity, and added transitions between all current acti... …
Zoa84 committed on Feb 3 — eb723a7

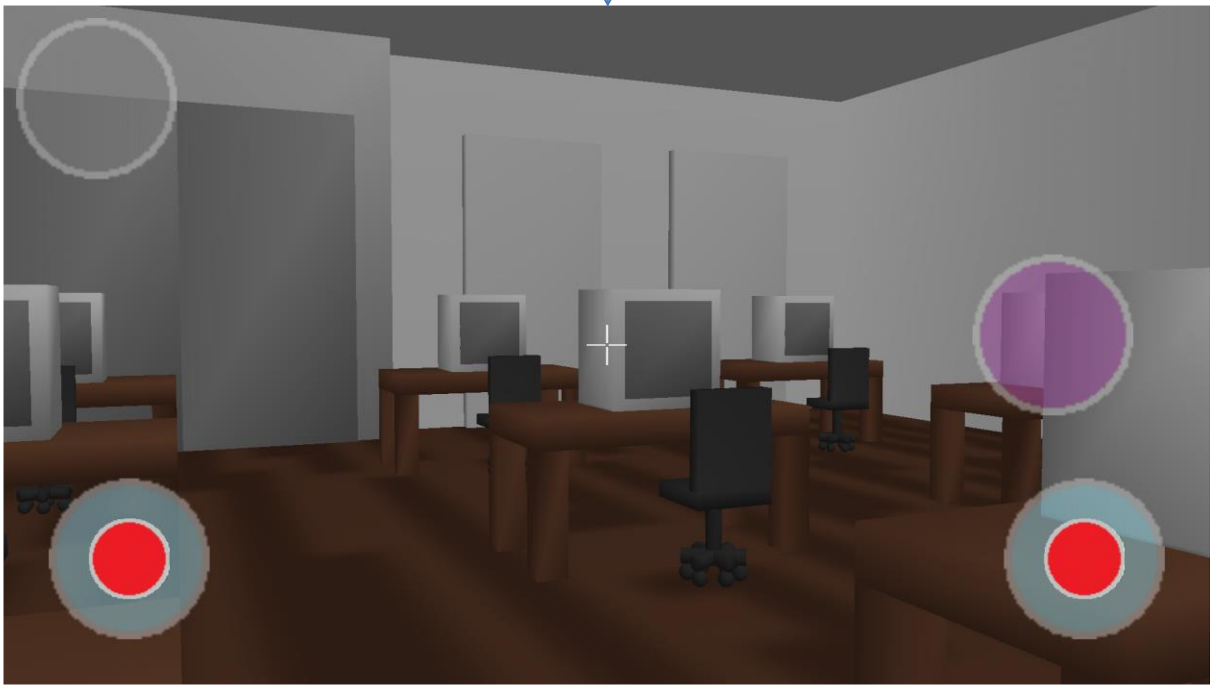
Added existing copy and files to repository
Zoa84 committed on Feb 3 — 9712b58

Initial commit
Zoa84 committed on Feb 3 — 133d59a

5.2 Scene creation in Maya to game

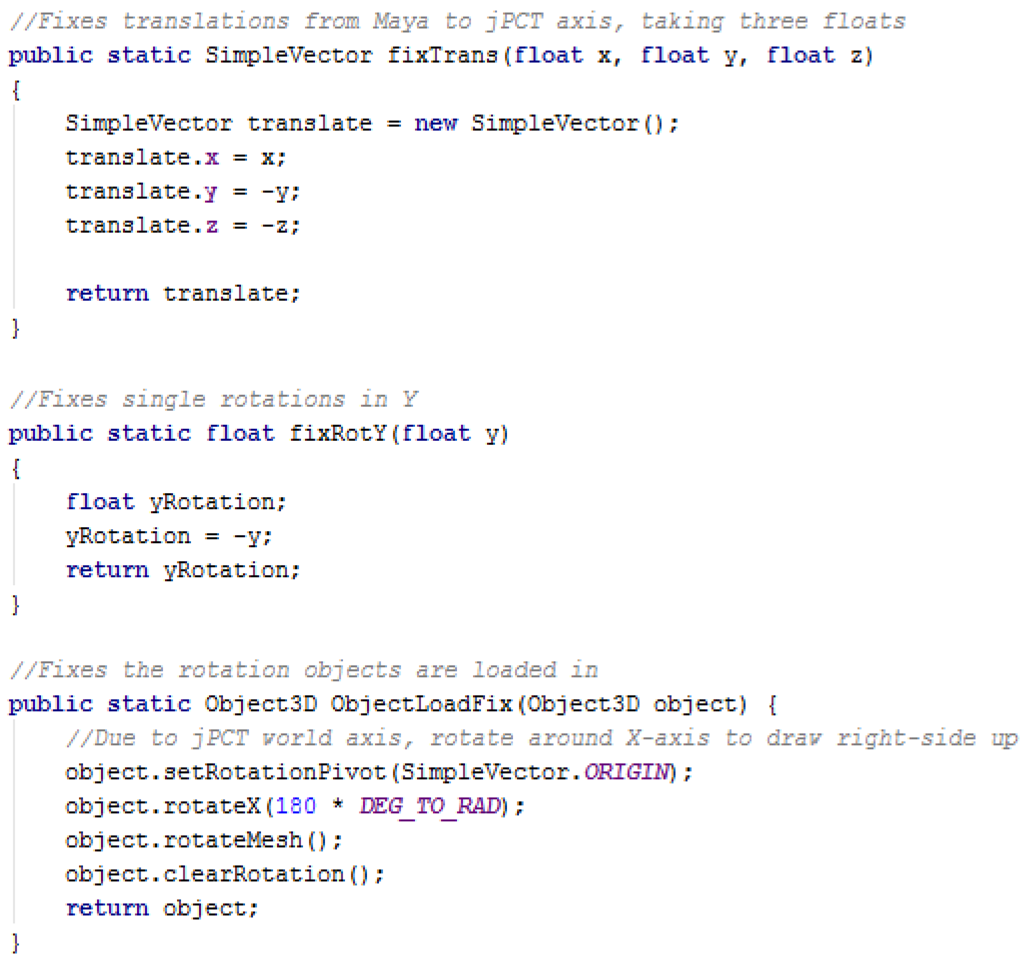
The screenshots show how a scene created in Maya using several objects, can be used and rendered with textures using jPCT-AE and Android Studio.





5.3 Functions to fix object transformations

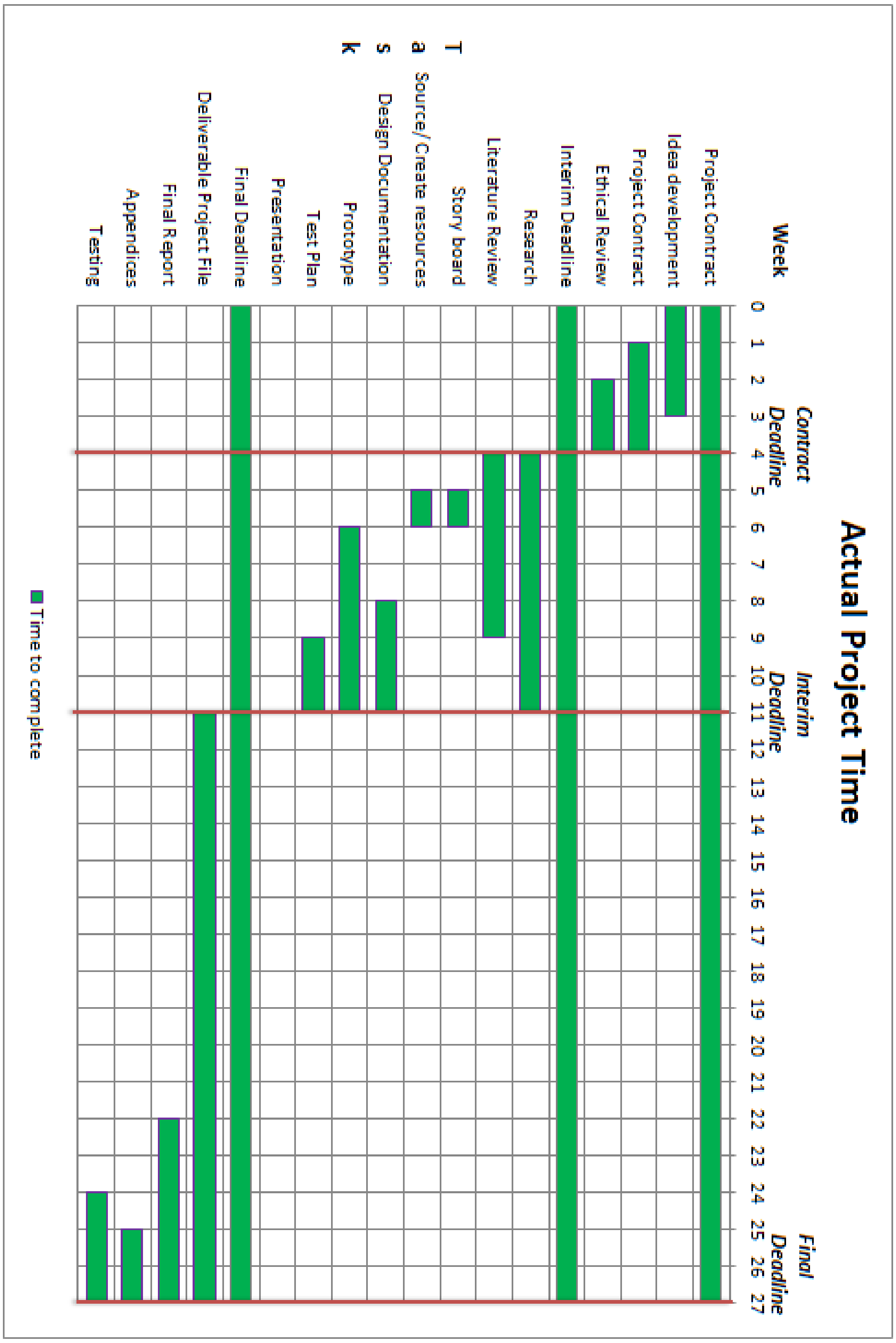
Functions used to fix the transformations. Objects loaded would use the ObjectLoadFix() function to rotate their meshes, and the fixTrans() and fixRotY() functions can be used to position the objects using Maya coordinates.



```
//Fixes translations from Maya to jPCT axis, taking three floats
public static SimpleVector fixTrans(float x, float y, float z)
{
    SimpleVector translate = new SimpleVector();
    translate.x = x;
    translate.y = -y;
    translate.z = -z;

    return translate;
}

//Fixes single rotations in Y
public static float fixRotY(float y)
{
    float yRotation;
    yRotation = -y;
    return yRotation;
}

//Fixes the rotation objects are loaded in
public static Object3D ObjectLoadFix(Object3D object) {
    //Due to jPCT world axis, rotate around X-axis to draw right-side up
    object.setRotationPivot(SimpleVector.ORIGIN);
    object.rotateX(180 * DEG_TO_RAD);
    object.rotateMesh();
    object.clearRotation();
    return object;
}
```

5.4 Actual Project Time Chart

Gantt chart showing the actual time spent on the project. More time was required for the research and the literature review.



5.5 Test Cases

Screenshots or diagrams used in any of these tests can be accessed from the ‘Appendix’ under the ‘Test Diagrams’ section.

Test No.	Test Case	Inputs	Expected Output	Observed Output	Screenshots/ Diagrams	Comments
1	Data loading	Options.txt text file, stored in the phone’s memory	Data should be loaded from a text file in the Splash Activity, and can be observed from the Options Activity	The data was loaded correctly, with the various widgets in the correct position	Test Diagrams 1A Test Diagrams 1B	This was tested twice using different data
2	Data loading without data	N/A	When loading non-existent data, the game should create a text file with default settings	A text file with default settings was created under the ‘Annihilation Intelligence’ directory	Test Diagrams 2A Test Diagrams 2B	This was tested with and without the directory
3	Button widgets	Finger presses	Pressing any button widget should do the action written on the button, including moving between activities or ending the game	All of the buttons tested successfully executed the desired action, such as moving between activities, saving data or ending the game	Test Diagrams 3	There are various buttons in the game, with a few different ones tested. Ending the game also worked
4	Radio Button widgets	Finger presses	As a radio group, choosing one option should deselect the other option, as well as update the orientation in real time	The radio group works successfully, switching between the two options and updating the orientation	Test Diagrams 4	The options can also be chosen by tapping the images; however, this caused the sound effect to play twice
5	Music changes	Switching activities	When switching between certain activities, different music should play	The music from the last activity is properly stopped, and music for the next activity is played	N/A	If the activity uses the same music as the last activity, the music will continue without pausing
6	Check box widgets	On/Off	Tapping a check box widget will switch it between on and off, with any suitable changes taking place	The check boxes switch correctly, and music and sound also stop if turned off in this activity, and in other activities if saved	N/A	The check boxes for inverting the camera was tested alongside the movement

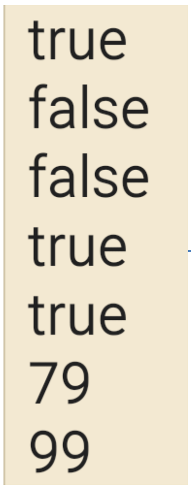
7	Seek bar widgets	Integers (0-99)	Using the seek bar slider, setting a value between 0 and 99 will correspond to that volume for music or sound	As well as the correct volume being used from the input, the music changes in real time with the volume choice, and a volume of 0 successfully creates no sound	N/A	Regardless of the check box option for the music or sound, the seek bars can still be adjusted
8	Saving settings data	Finger presses, different settings data	Pressing any of the buttons will leave the Options Activity, and successfully save the correct values to the text file	The three buttons successfully leave the app, and also saves the set data to the text file. The toast message confirms that data is saved	N/A	There are three options from the Options Activity, however, each saves data over the text file
9	Phone disruptions	Taking phone calls, force closing app	With any kind of phone disruption, the game should be able to handle it without crashing or corrupting itself	The app can handle taking a phone call, and successfully returns to the app when finished. Force closing the app also causes no problems	Test Diagrams 5A Test Diagrams 5B	It is difficult to judge how well it handles force closures in certain situations, especially if in the middle of saving data
10	Using phone 'Back' button	'Back' button	Pressing the 'Back' button should successfully leave certain activities, pause the game, and exit the game depending on the current activity	The Options Activity returns to the Menu Activity and the Menu Activity successfully leaves the game. When in-game, the pause menu appears, and can be closed by pressing the 'Back' again	Test Diagrams 6A Test Diagrams 6B Test Diagrams 9	When pressing 'Back' from the Menu Activity, a toast message appears to confirm the players choice (Test Diagrams 6B)
11	Character Movement	Left Virtual joystick	Using the left joystick should move the character, moving forward, backwards, and strafing	The character moves properly like the 'Interim Deadline'. The distance from the centre of the joystick determines the speed, and going outside the bounds locks the joystick at the last position	Test Diagrams 7	Character movement was tested in the 'Interim Deadline'; meaning this test focused on the joysticks
12	Camera Rotation	Right Virtual joystick	Using the right joystick should rotate the character, locking when looking straight up or down	The rotation works correctly, and locks when looking straight up and down. Going outside the bounds of the joystick locks it to the last position	Test Diagrams 8	Camera rotation was tested in the 'Interim Deadline'; meaning this test focused on the joysticks, and locking the camera up and down

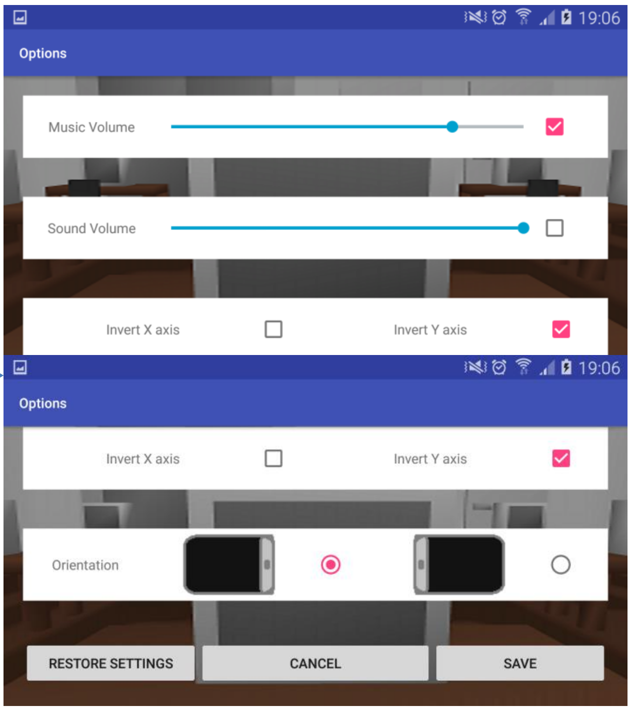
13	Inverted Camera Rotation	Right Virtual joystick	With the settings for inverting the camera in the Options Activity, the character should rotate the other way	The inverted rotation works properly while still retaining the camera lock and the bounds of the joystick shown in the 'Camera Rotation' test	N/A	The inverted controls work separately (only X or Y inversion) and together
14	Pausing and Resuming	Pause Button 'Back' Button	Pressing the Pause button or the 'Back' button should open the Pause menu. We should also be able to close this menu to resume	The Pause menu successfully opens when pressing either button. Pressing 'Back' again closes the menu, as does pressing the Resume button	Test Diagrams 9	N/A
15	Pause Menu Interactions	Virtual joysticks Pause Button Interact Button Inventory	With the Pause menu open, we should not be able to use the character controls as the game is paused	The various controls are properly disabled while the Pause menu is open. Sound effects also are not activated while paused	N/A	N/A
16	Returning to Menu	'Return to Menu' button	With the Pause menu open, we should be able to exit the Game Activity, and re-enter the Menu Activity	Pressing the 'Return to Menu' button from the Pause menu successfully returns to the Menu Activity	Test Diagrams 10	The toast message from leaving the Game Activity confirms we have left the game without completing it
17	Correct Sound Effects	All buttons Inventory Interactions	Depending on the action or circumstance, the correct sound effect should play	The correct sound effects are played in the right situation, with the volume set in the Options Activity	N/A	Some sound effects were too loud while some too quiet. These should be equalised
18	Interactions	Interact Button	Pressing the button alone should do nothing, but pressing when facing a collectable item or an interact-able object should pick up or use the item	As expected, the Interact button has no function alone, and facing an interact-able object (with the text 'Interact' showing) successfully picks up or uses the item	Test Diagrams 11	The sound effects for interactions only play if an interaction takes place. Items collected also show in the inventory properly

19	Inventory Use	Inventory	With collected items, tapping the items in the inventory should select/ de-select them	The correct item is selected and de-selected when pressed. Pressing a different item to the selected one switches the selected item	Test Diagrams 12	N/A
20	Interactions with Items	Inventory Interact Button	With certain items, interacting with objects should yield different results as part of the game	Interacting with this door with and without a key card gives different results as expected	Test Diagrams 13	Not every item has a different interaction with different objects, only those that are relevant
21	Collisions with walls	Left Virtual joystick	When moving towards a wall, the character should stop against the wall, and not go through it	The character successfully stops before going through the wall, and instead slides along the wall	N/A	The character also does not go through the walls in a corner
22	Collisions with objects	Left Virtual joystick	When moving towards an object, the character should stop against the object, and not go through it	The character successfully stops before going through an object. However, they do not slide against the object like with the walls	N/A	N/A
23	Game Overs	Interact Button	Interacting with certain objects should cause a Game Over as part of the game	Game Overs successfully happen at the correct situation, switching to a Game Over screen, and shortly returning to the Menu Activity	Test Diagrams 14	The music also stops, and the sound effect plays for the Game Over state
24	Completing the game	Interact Button	Interacting with the front door of the building should complete the game	The game ends correctly, entering a Win screen, and shortly returning to the Menu Activity	Test Diagrams 15	The music also stops, and the sound effect plays for the Win state, and returning to the menu shows a toast message confirming the game was completed

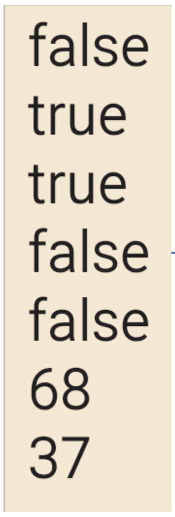
5.6 Test Diagrams

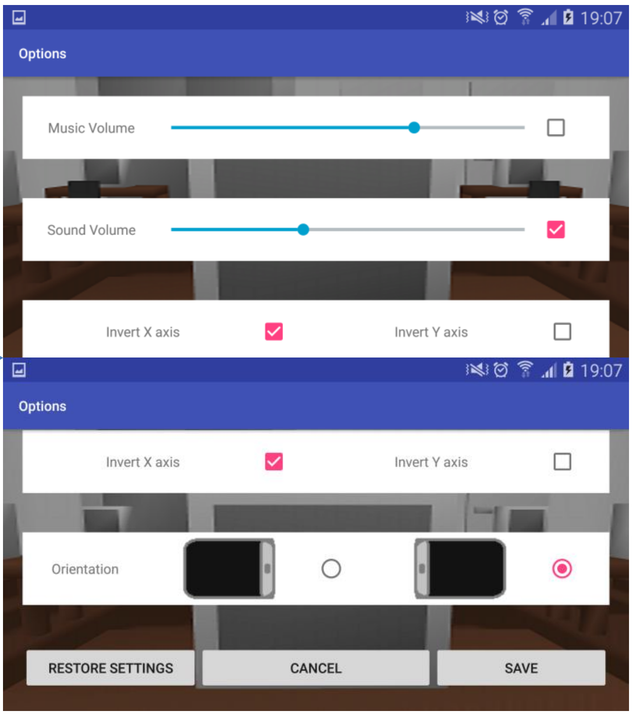
1A.


true
false
false
true
true
79
99


19:06
Options
Music Volume ✓
Sound Volume ☐
Invert X axis ☐ Invert Y axis ✓
Orientation ● ○

1B.

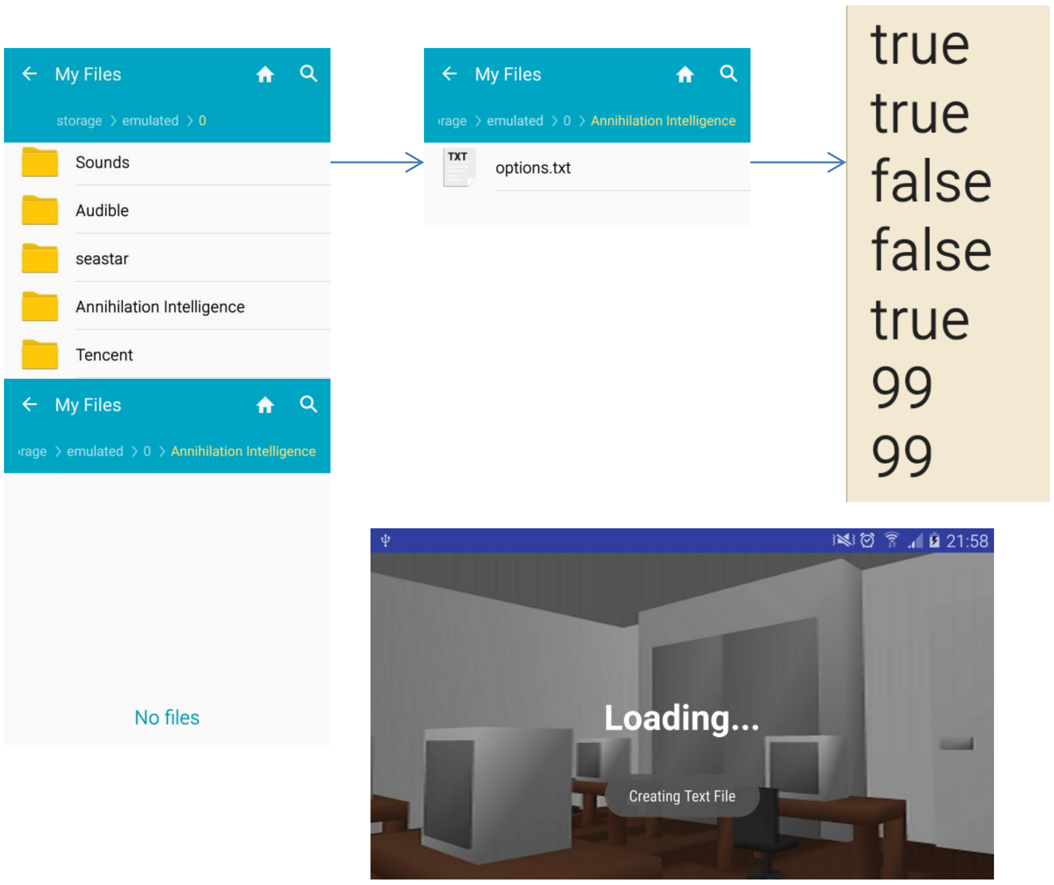

false
true
true
false
false
68
37


19:07
Options
Music Volume ☐
Sound Volume ✓
Invert X axis ✓ Invert Y axis ☐
Orientation ○ ●

Using the example data on the left, when the game was loaded, the Options Activity was accessed, which had the same data as shown.

Data from the options.txt file, in order of: Music checkbox, Sound checkbox, Invert X axis checkbox, Invert Y axis checkbox, Orientation (true = left-side down, false = right-side down), Music volume and Sound volume.

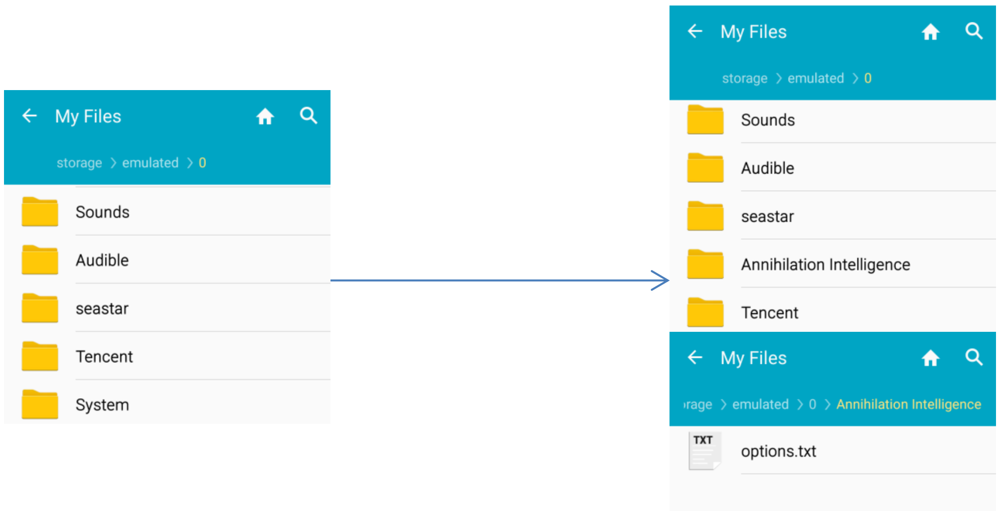
2A.



```
true
true
false
false
true
99
99
```

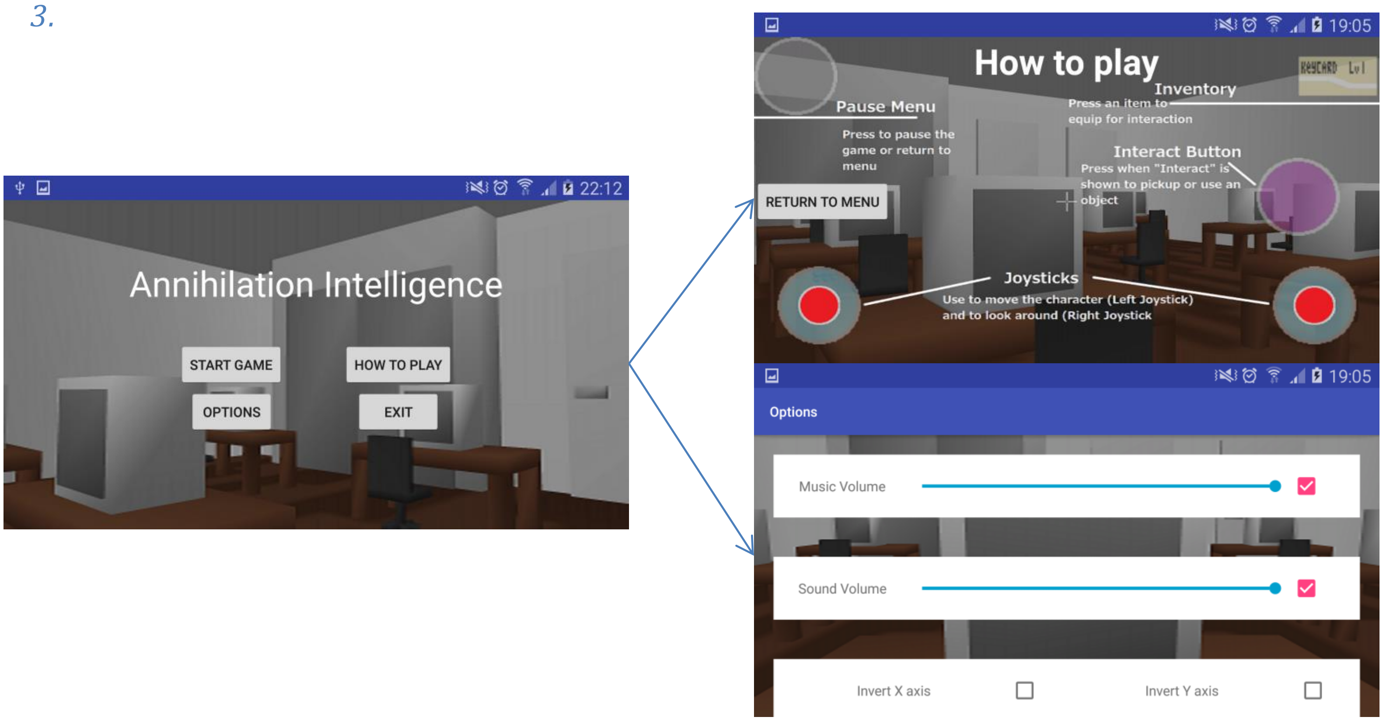
The screenshots on the left show the empty directory without the text file. When the game runs, if the directory is empty, an options.txt is created with default values. A toast message is also shown to alert the player of these changes.

2B.



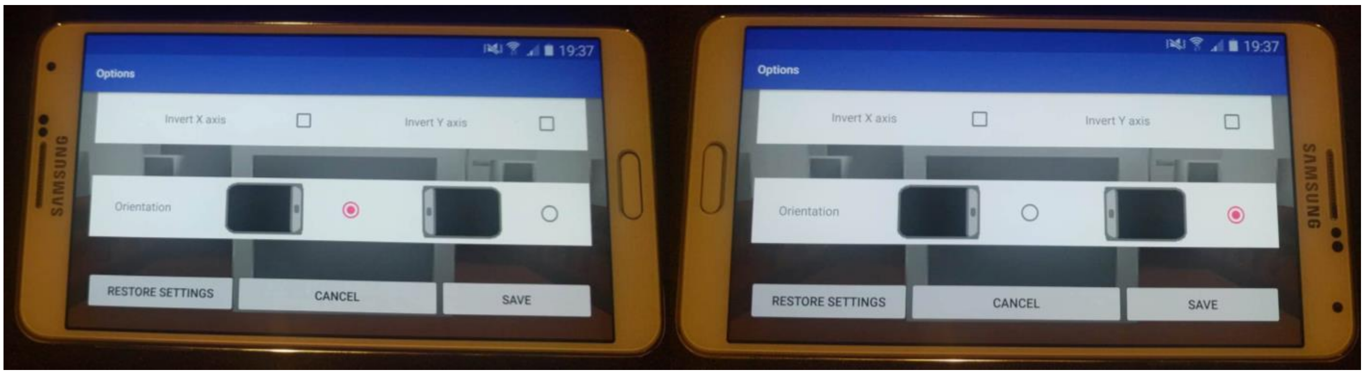
If the directory for 'Annihilation Intelligence' does not exist, the game creates a directory, as well as prepares default values.

3.


Annihilation Intelligence
START GAME
HOW TO PLAY
OPTIONS
EXIT
How to play
Pause Menu
Press to pause the game or return to menu
Inventory
Press an item to equip for interaction
Interact Button
Press when "Interact" is shown to pickup or use an object
RETURN TO MENU
Joysticks
Use to move the character (Left Joystick) and to look around (Right Joystick)
Options
Music Volume
Sound Volume
Invert X axis
Invert Y axis

Pressing any of the button widgets, the game moves to the next, correct activity. Above shows moving from the Menu Activity to the Instructions Activity and from the Menu Activity to the Options Activity.

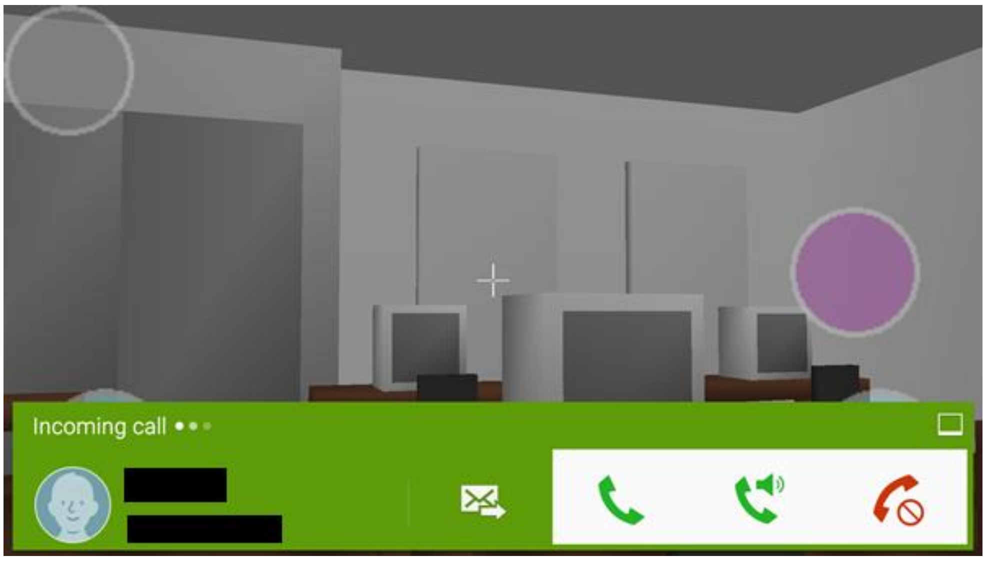
4.



The above picture shows how the left-side orientation (Left) and the right-side orientation (Right) compare. The radio group works in switching between the two options. They can be switched by pressing the radio button, or by pressing the image, with the orientation changing with the option chosen.

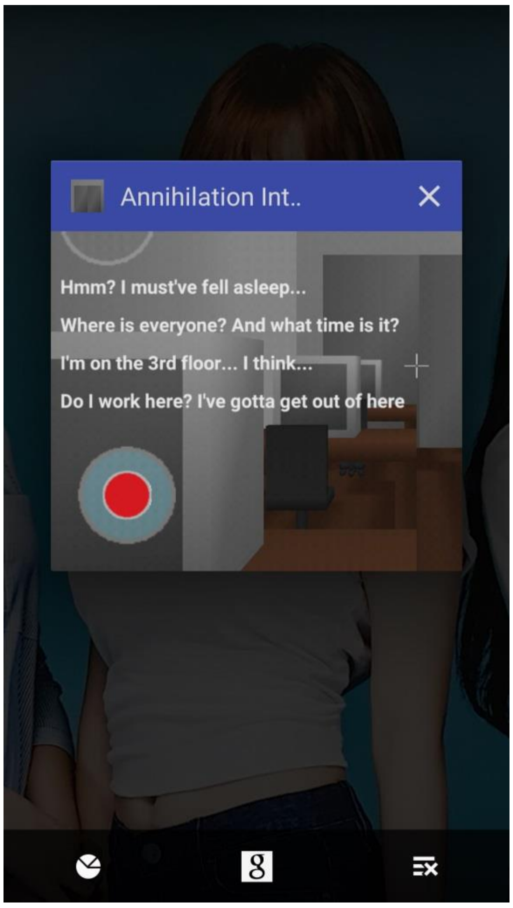
5A.

When receiving a phone call, the phone's interface layers on top of the game, allowing the player to answer or reject the call, then continue back to the game.



5B.

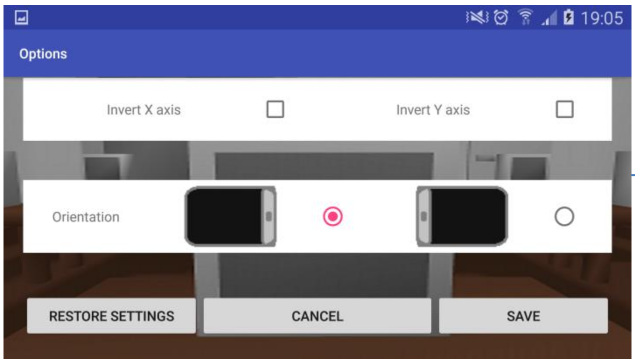
The application can be successfully paused, and force closed, without any problems.

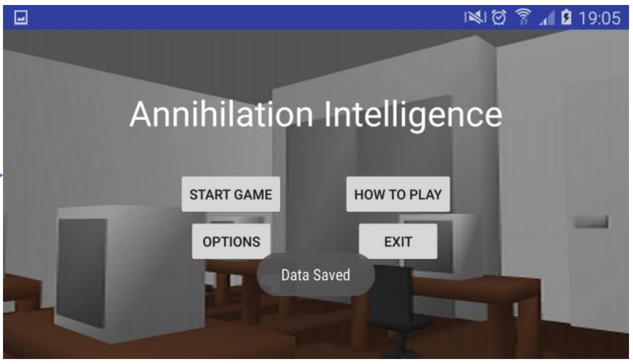


Hmm? I must've fell asleep...
Where is everyone? And what time is it?
I'm on the 3rd floor... I think...
Do I work here? I've gotta get out of here

6A.

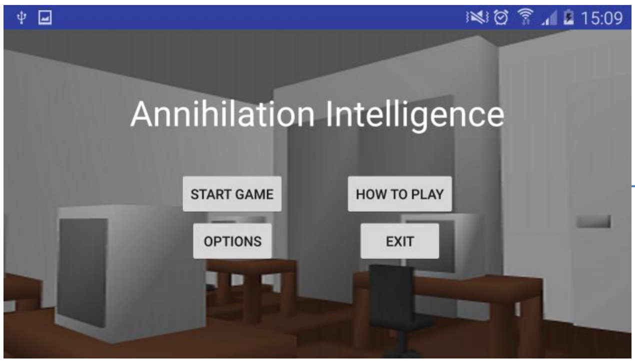
From the Options Activity, pressing the ‘Back’ button will return to the Menu Activity, keeping the data we had before we entered the Options Activity.

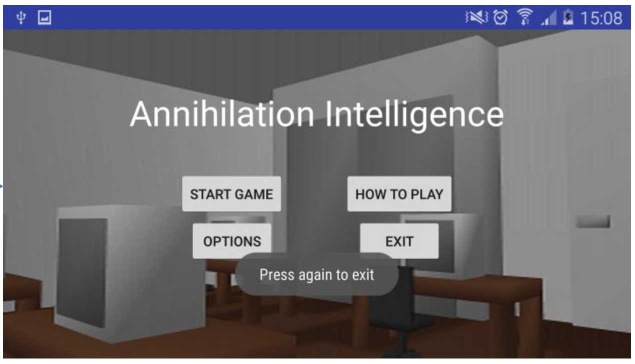




6B.

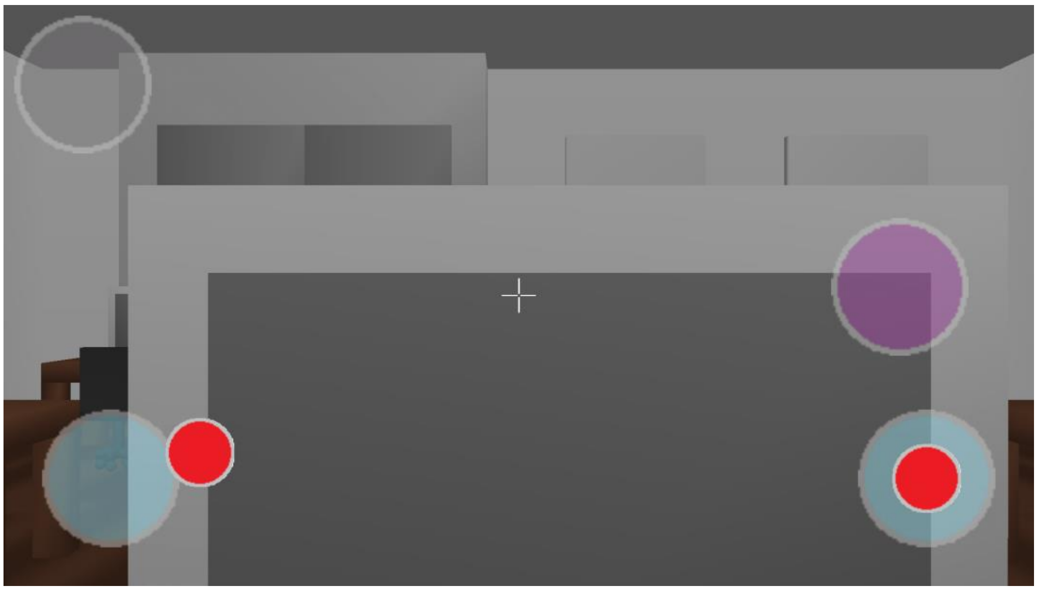
From the Menu Activity, pressing the ‘Back’ button will create a toast message, asking the player “Press again to exit”. This prevents players from accidentally closing the app when pressing the ‘Back’ button.





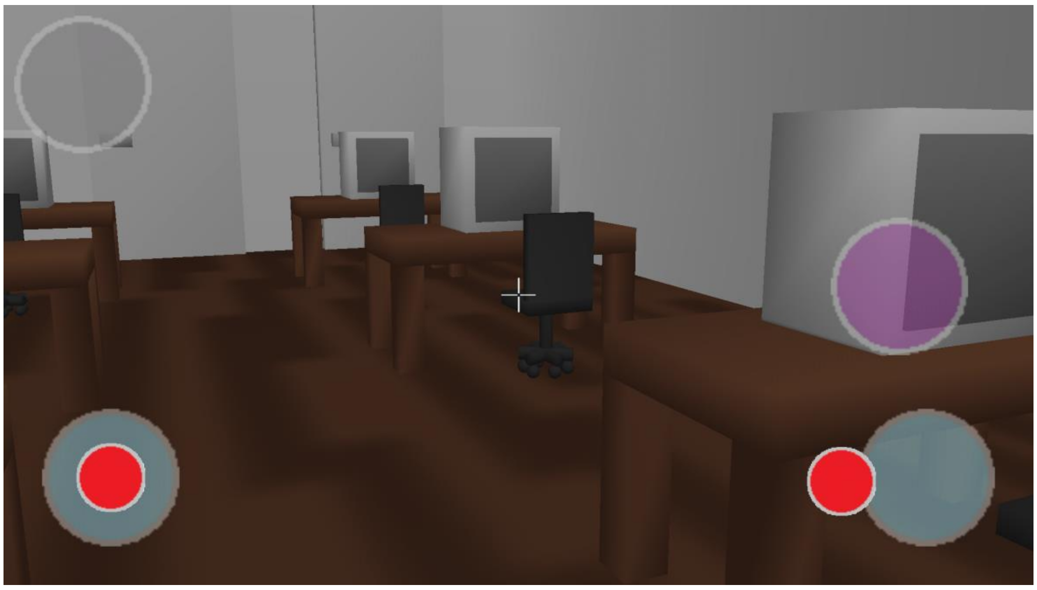
7.

The character moves in the correct direction and speed based on the joystick. Going outside the range of the joystick locks the joystick at the last position.



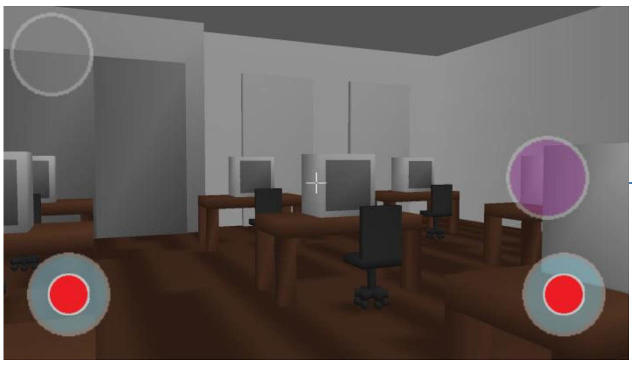
8.

The camera rotates in the correct direction and speed based on the joystick. Going outside the range of the joystick locks the joystick at the last position.

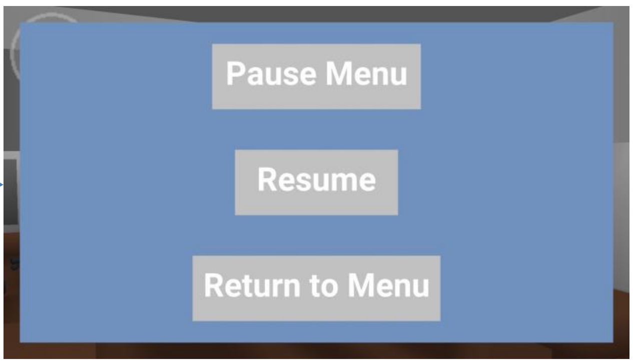


9.

While in the Game Activity, pressing either the Pause button or the ‘Back’ button will open the Pause menu. Pressing ‘Back’ again from here will close the menu, as will as pressing the Resume button.

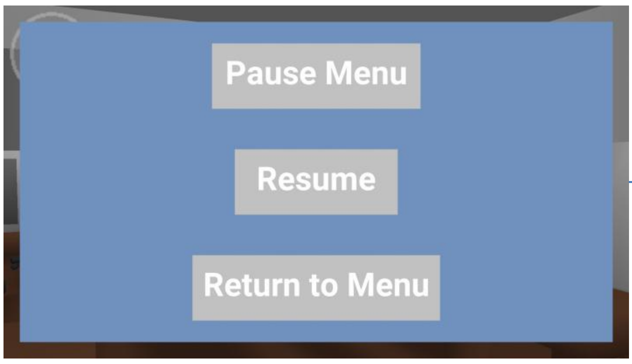


$\longrightarrow$

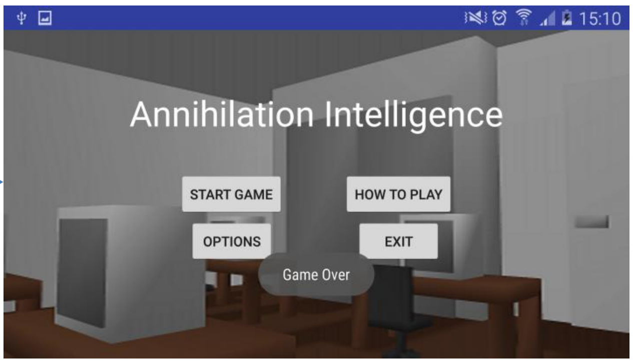


10.

Pressing the ‘Return to Menu’ button ends the game and Game Activity, returning to the Menu Activity and leaving a toast message with “Game Over”, meaning the Menu Activity has acknowledged the end of the game.

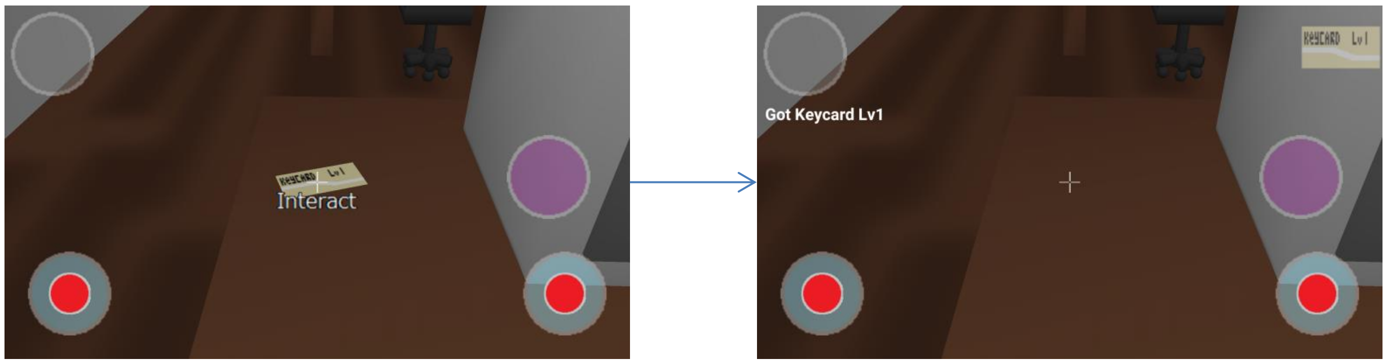


$\longrightarrow$



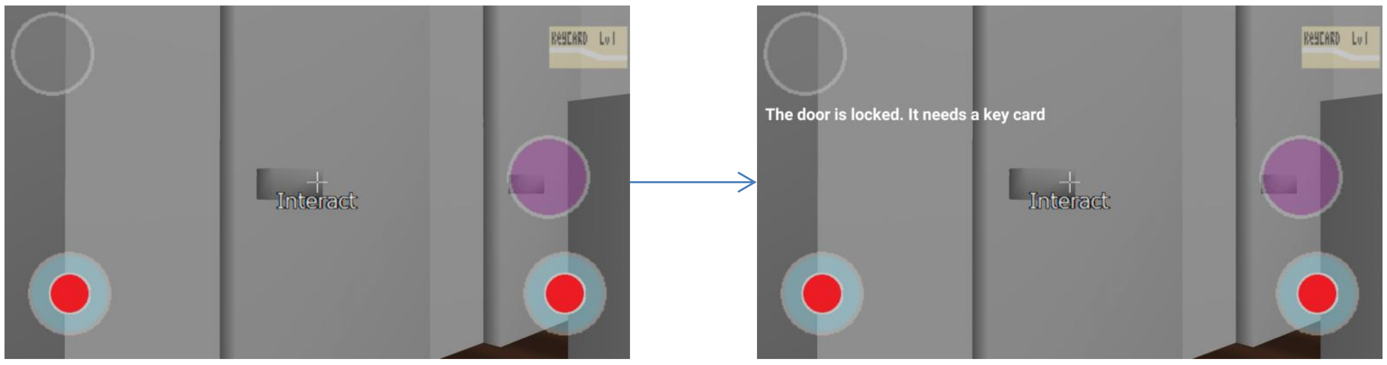
11.

Pressing the Interact button alone does nothing; however, when facing an object that can be picked up, the item appears in the inventory.



Got Keycard Lv1

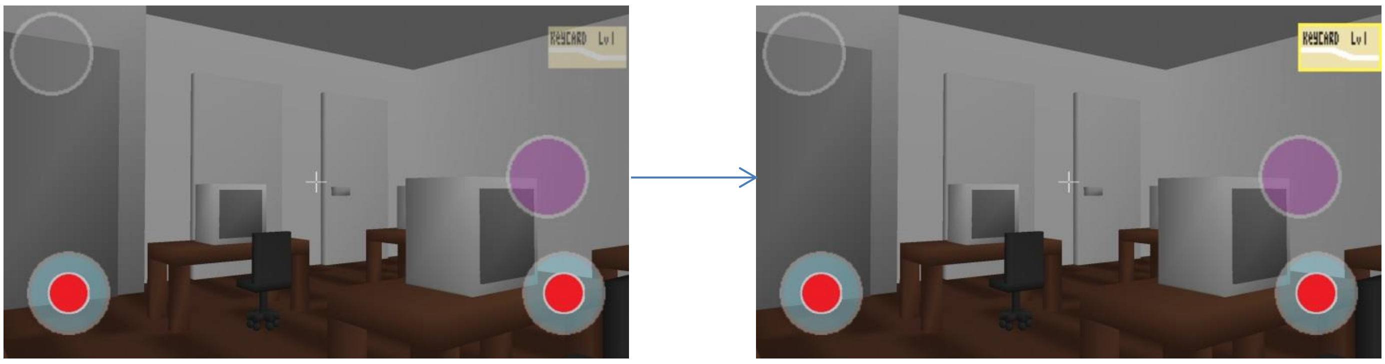
Pressing the Interact button on an object that cannot be picked up, but can be used, allows the interaction with the object to take place.



The door is locked. It needs a key card

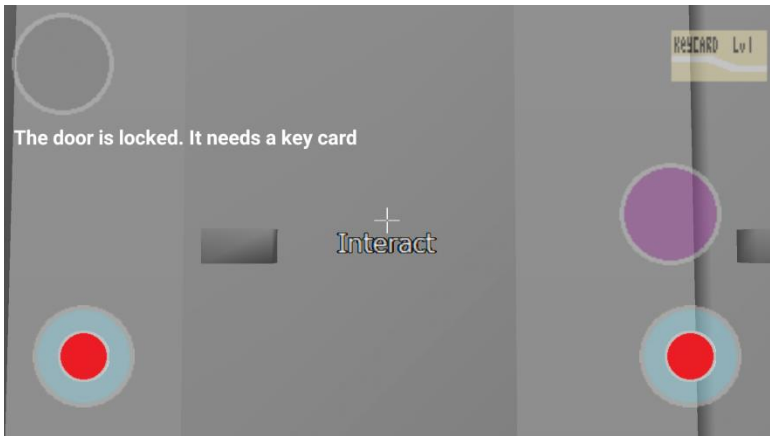
12.

Pressing the item in the inventory causes it to light up, confirming the item has been selected for the next interaction.

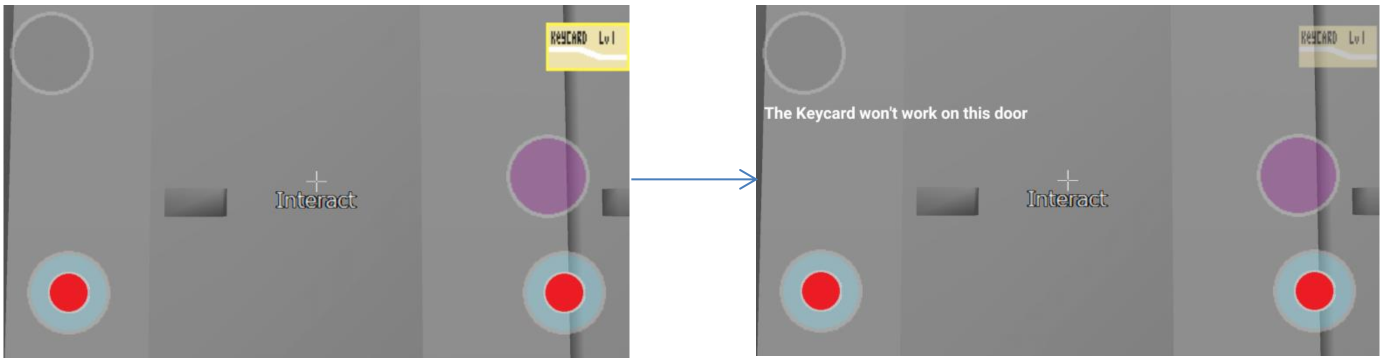


13.

Pressing the Interact button on the door returns a message stating “The door is locked. It needs a key card”.

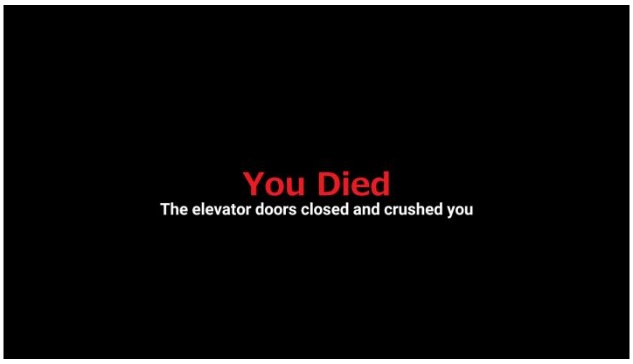

The door is locked. It needs a key card

After selecting an item, pressing the Interact button on the door comes up with a different message, showing that using the selected item was attempted on the door.

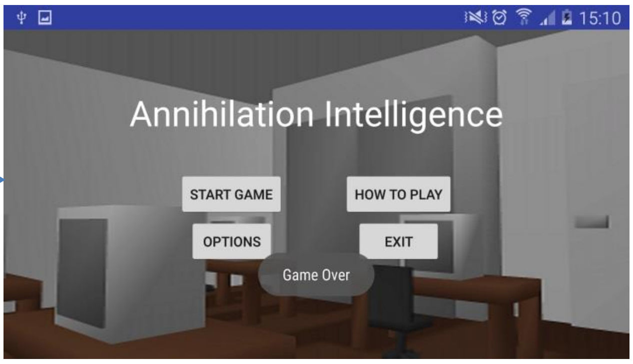

The Keycard won't work on this door

14.

The Game Over state stops any music playing, and plays the sound effect for game over. After a short delay, it returns to the Menu Activity, giving a toast message to confirm the game is over.

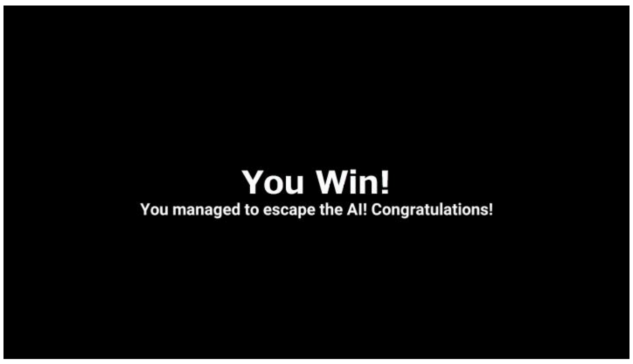


$\rightarrow$

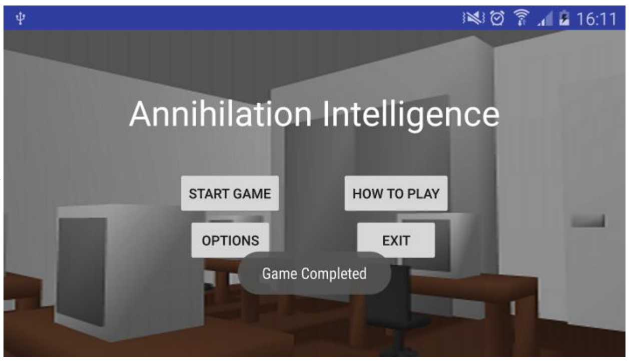


15.

The Win screen stops any music playing, and plays the sound effect for completing the game. After a short delay, it returns to the Menu Activity, giving a toast message to confirm the game has been completed.

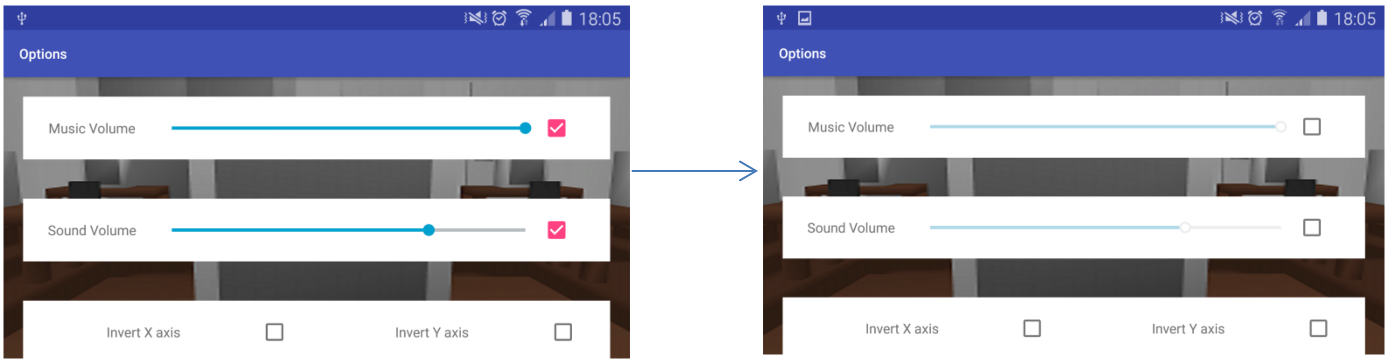


$\rightarrow$



16.

After the changes, changing the check boxes to false will ‘grey out’ the seek bars, rendering them unchangeable until the check boxes are set to true.



17.

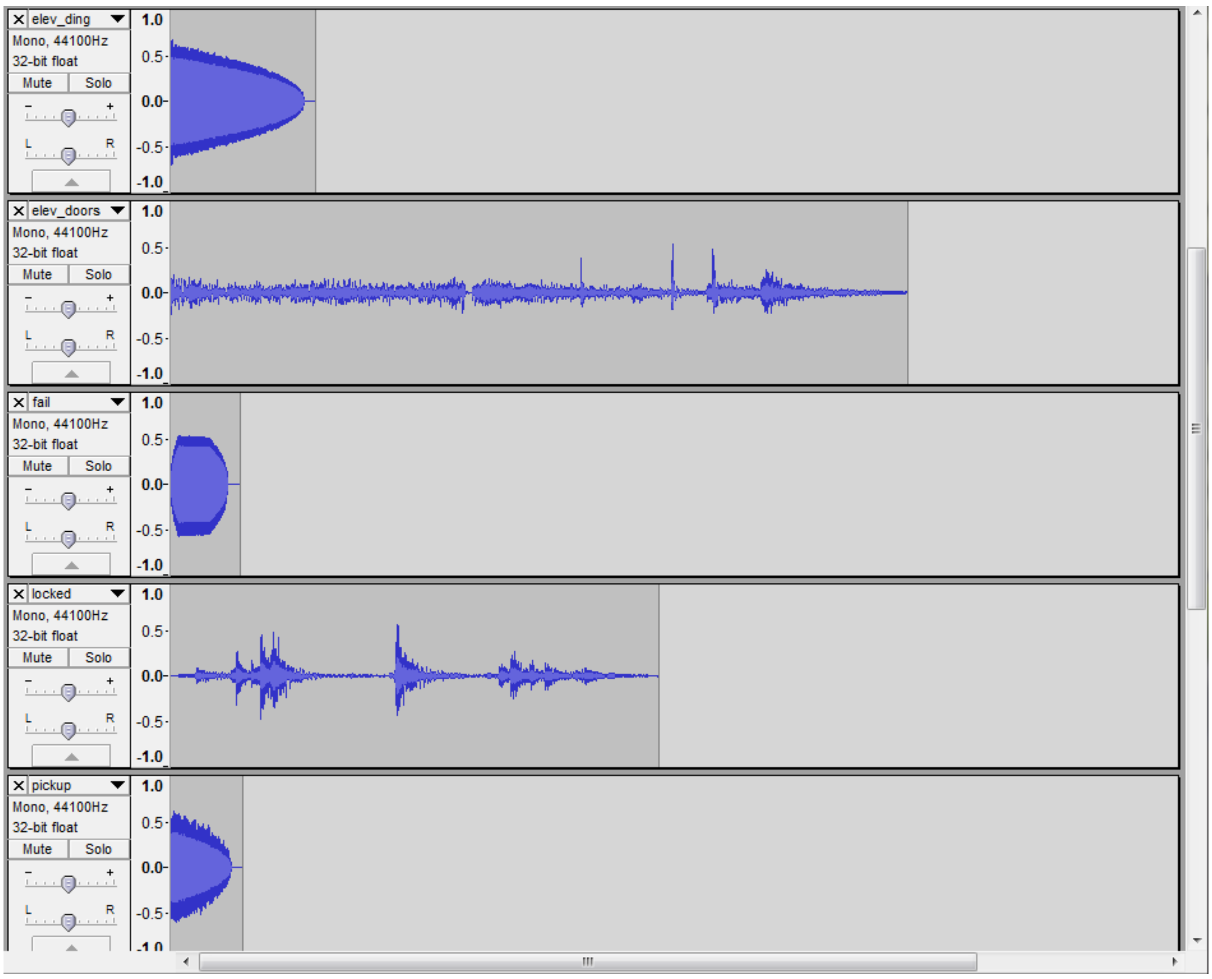
By changing the code to use the setChecked() function of a radio button as opposed to the check() function of the radio group, the output of playing the sound effect and changing the groups is not duplicated, which caused the sound effect to play multiple times.

```java
//Change the orientation using the images
public void OriLeft (View view) {
    RadioGroup rGroup = (RadioGroup) findViewById(R.id.rgOrien);
    rGroup.check(R.id.rLeft);
}

//Change the orientation using the images
public void OriRight (View view) {
    RadioGroup rGroup = (RadioGroup) findViewById(R.id.rgOrien);
    rGroup.check(R.id.rRight);
}
```

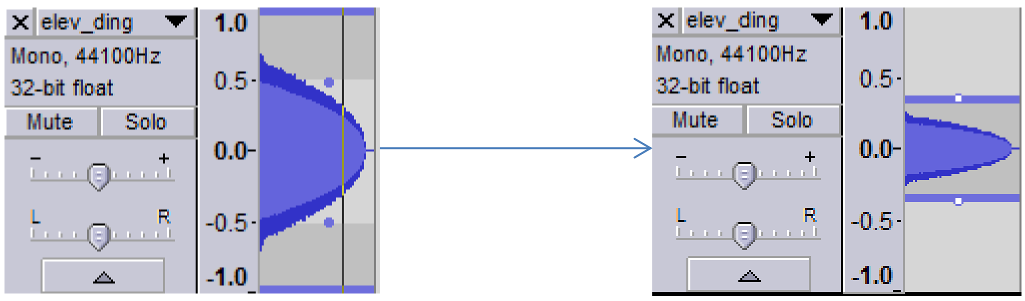
```java
//Change the orientation using the images
public void OriLeft (View view) {
    RadioButton rButton = (RadioButton) findViewById(R.id.rLeft);
    rButton.setChecked(true);
}

//Change the orientation using the images
public void OriRight (View view) {
    RadioButton rButton = (RadioButton) findViewById(R.id.rRight);
    rButton.setChecked(true);
}
```

18.

An example of some of the sound effects in Audacity. Certain sounds such as the 'elev_ding' sound effect has a large amplitude (height of wavelength) as opposed to the 'locked' sound effect, with a much smaller amplitude.

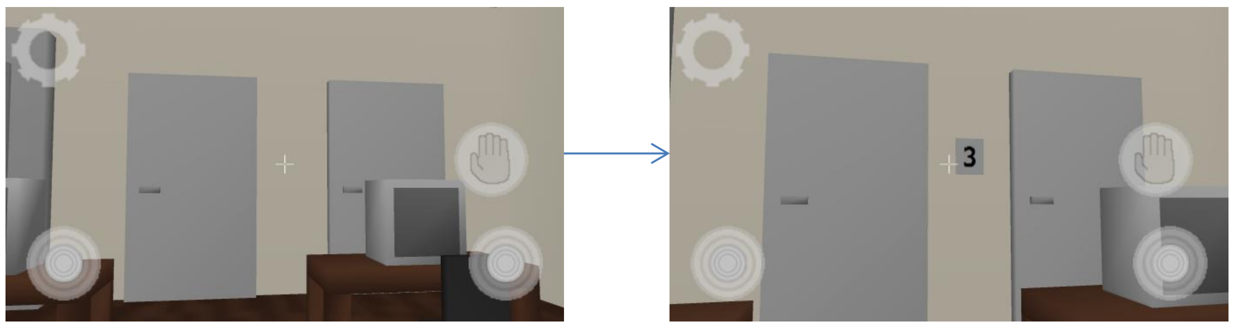


Using Audacity, the amplitude can be very easily adjusted, dragging the amplitude to the desired amount.



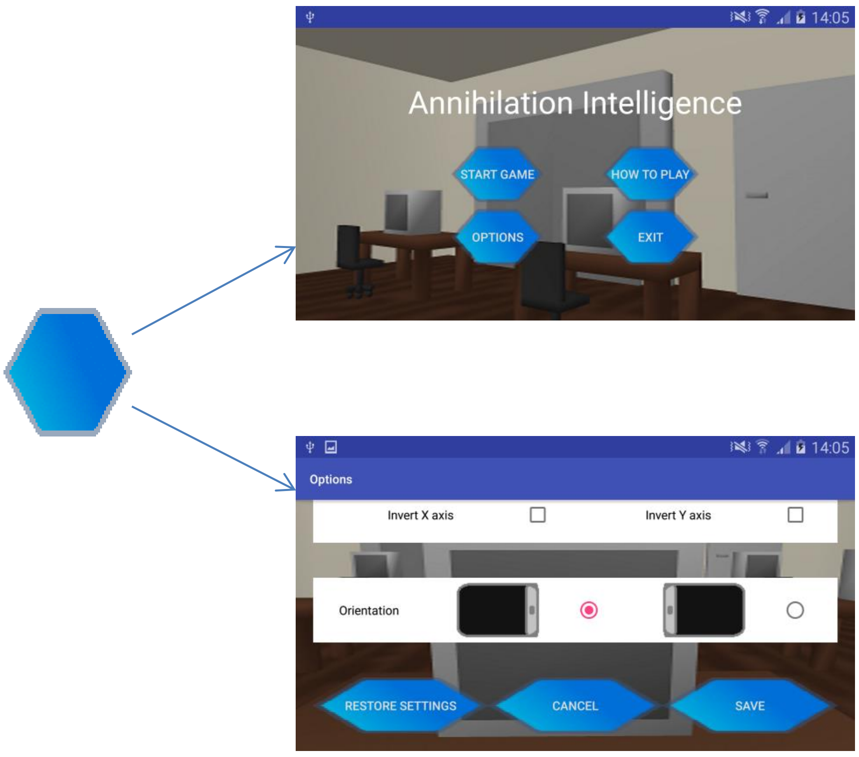
19.

Below are the changes to improve player navigation, with a basic sign of the current floor the player is on.



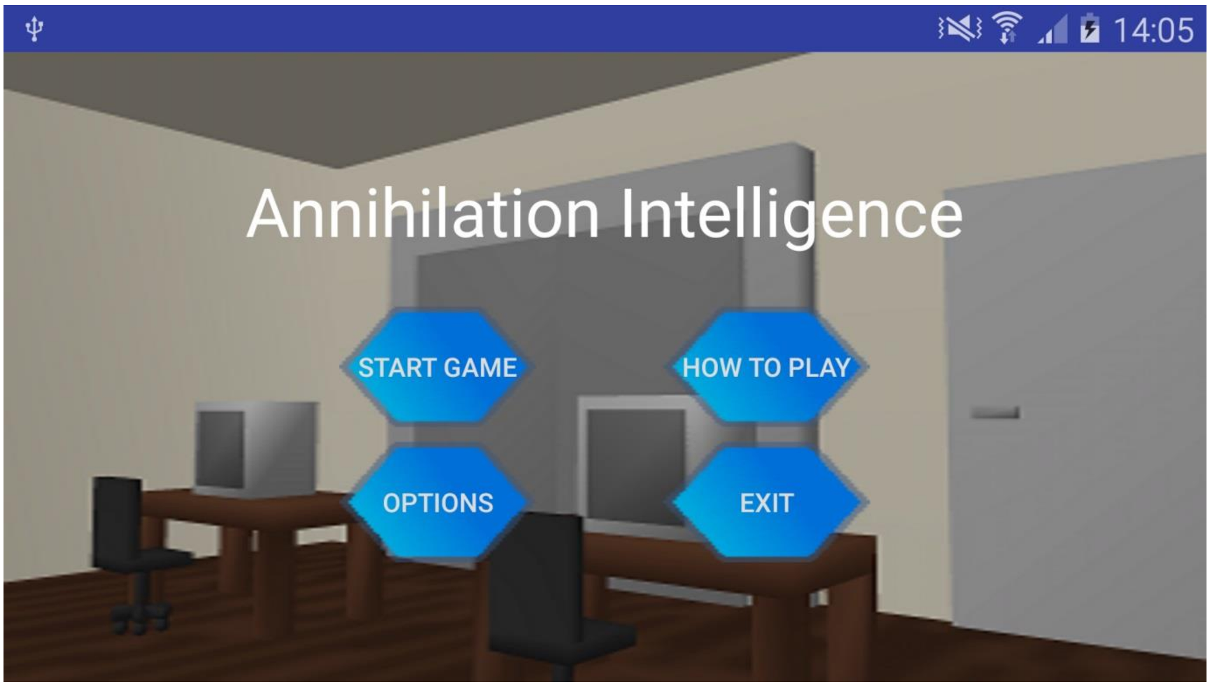
20.

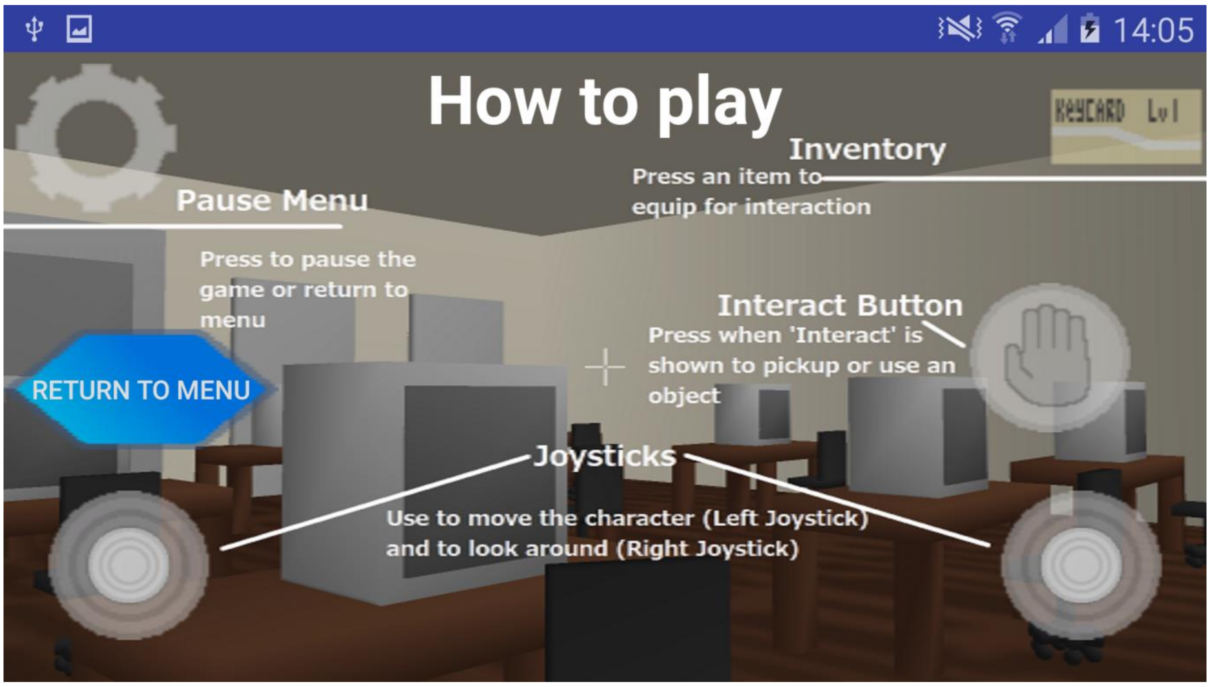
The button texture, designed as a square to save on memory usage, can be set and stretched to fit the text of the button.



Invert X axis ☐
Invert Y axis ☐
Orientation ● ○

5.7 Final Development Screenshots



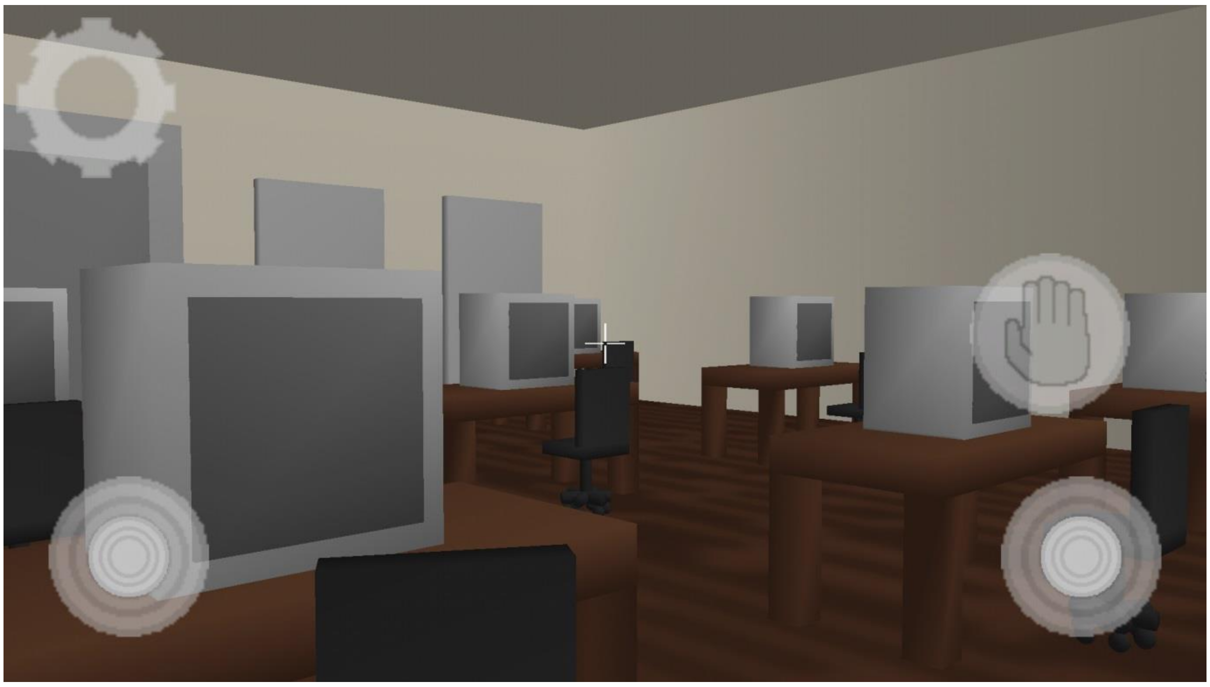


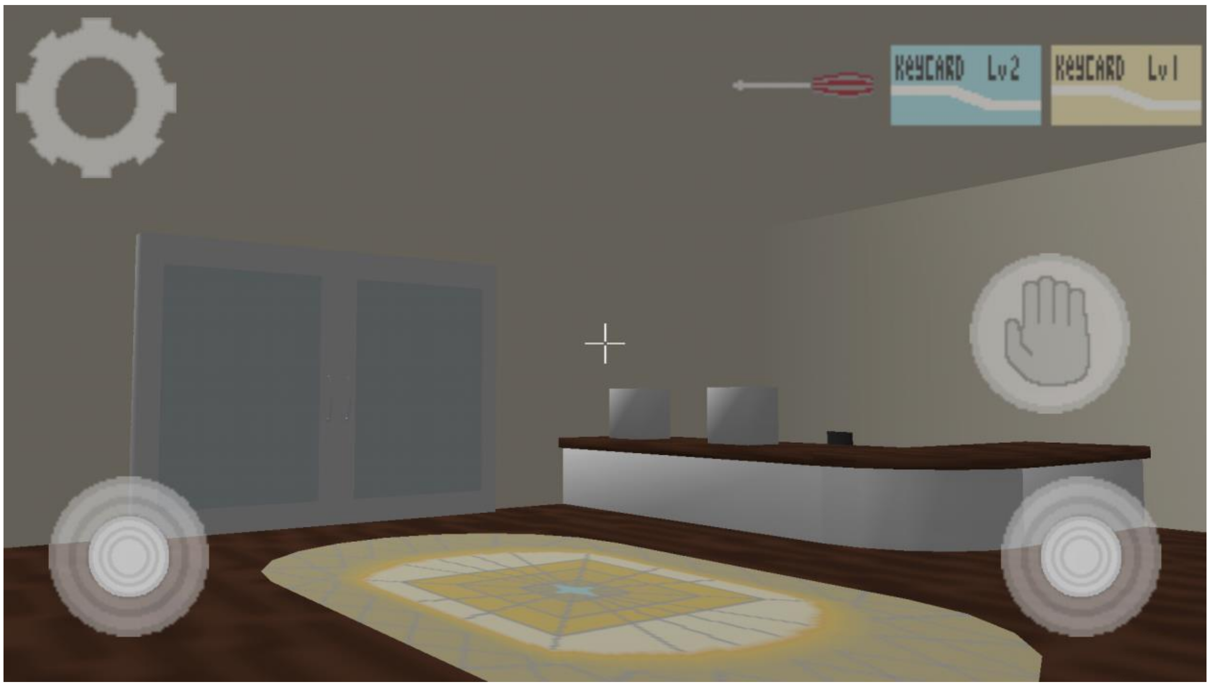
Press an item to equip for interaction

Press to pause the game or return to menu

Press when 'Interact' is shown to pickup or use an object

Use to move the character (Left Joystick) and to look around (Right Joystick)




KEYCARD Lv2
KEYCARD Lv1