For the final version, I decided to structure the project similarly to how we structured our mobile game project in the IMAT 2608 Mobile Games module from second year. This separates the project into Activities, Classes and Views. Activities contains the different ‘screens’ of the game, such as the start-up screen, the options screen, etc. Classes contains the java classes we use to define objects, for example the player, certain objects in the world, etc. Lastly the Views are how the game is displayed. The game uses a surface view to render and draw objects.

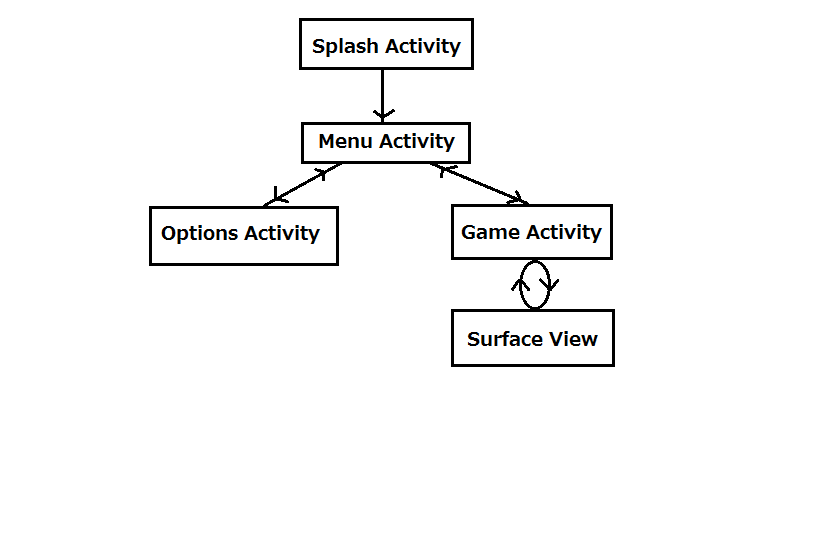
## Activities

The currently planned activities are:

* Splash Activity – This will be the starting screen, which is used to load certain parts of the game, or prepare files for the game
* Menu Activity – This is the main menu the player will see, and will allow access to other activities
* Game Activity – The game itself, this will then have a surface view to render and play the game in
* Options Activity – This will have the editable options for the player, such as changing the volume of music or sound effects, or changes to controls of the game

### Activity Life-cycle

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 we enter the Menu Activity. This is the main hub from which we can access other activities. 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.

We can 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 to settings to the last used settings. 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, then passes it to the Surface View to process. When the game ends, the Surface View will return this to the Game Activity, and it will return to the Menu Activity.

## 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.

## Flow of data through Activities

### Intents

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, we can set certain data to also be passed, by changing the Intent, which is essentially the new activity. By doing this we can also recover data from a child activity back to its parent.

For example, the options data is 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. We cannot create a pointer for an integer for example, and pass the pointer through the activities. Because of this, each activity has a local version of any passed data, and starting or ending and 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.

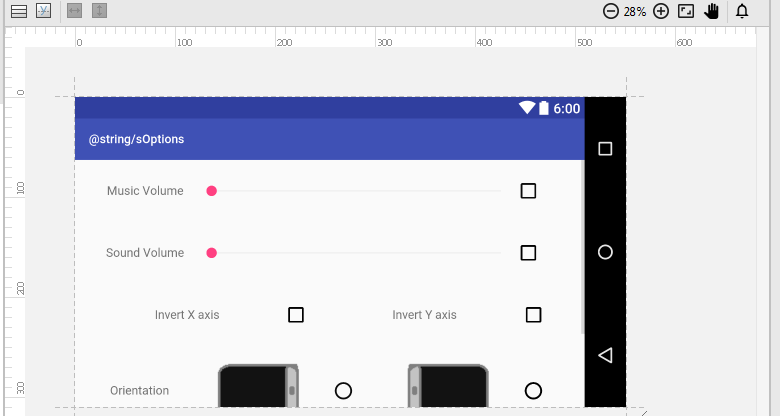


Fig. 1 – The Activity Layout for the Options Activity

In Fig.1 we can see the activity layout for the Options Activity. The various check boxes, radio boxes and seek bars can be accessed with Java pointers, searching for the ID of a widget. Using this we can access and change these widgets from the Activity code. Widgets such as buttons are given an onClick variable, which becomes a function in the Activity code.

### Singleton Classes

Data can also be passed through a singleton class, 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 is an example of a singleton class, 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.

### Why data flow is important

The setup of data to be used throughout the app is extremely important. We save loading times 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.

## Write up away from Computer

Write ups for various topics, done away from a computer (such as phone or tablet or laptop) which needs to be checked through.

### Seperating the Game Activity and Game Sufave View.

The game activity originally had the surface view integrated into the code although this made it easy to use variable between the two, this also made the code very messy and not very maintainable. I seperated this into an activity and a surface view class, however, this could potential break the code, and j-PCT may not actually work when doing this. To avoid changing the main code, i created a branch on Github, to test seperating the two apart.

With a branch on Github, im able to test different things and push them to the repository, without worry of replacing the original code, bu essentially having a seperate repository. Once im done with this branch, i can either merge the two, saving the changes ive dont with the branch, or it can be deleted, if any changes i 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 pf affecting other or getting affected by others changes.

### j-PCT World axis

J-PCT uses its own coordinate axis, which means transforming an object in these coordinates can be difficult. Using Maya to create a scene using objects, transforming in j-PCT would require us to reverse the y and z translations, and to reverse the x and z rotations to set the objects in the right place. Instead of manually reversing the correct numbers, i created a class (check name) which will allow us to pass translation or rotation vectors (as SimpleVector variables) and convert them to the correct format. This can be used as we use j-PCTs transform functions.

### Designing in Maya and loading the scene

For creating a scene, I used maya to position objects and to rotate them to create a scene. The translations and rotations can then be taken to the project, either by manually adding it as part of the code, or by having a scene loader as part of the project, which could read in a file and parse the data.

This is how the scene loader works. Similar to the Media class, it is a singleton which can be used in the surface view to load the object, load the textures, and to load the positions and rotations. This is somewhat similar to how we loaded objects in the Games Programming module, parsing in a file with all the necessary information.

### Puzzles

The game use a few puzzles as part of the gameplay, requiring the player to solve them to continue with the game and story. When the player interacts with a puzzle, the view changes to the puzzle, (can either continue using current controls, or be able to use 'touch screen' as it were.

The puzzles need to be somewhat short and consise. They arent the main part of the game, more of an add-on or chance to show my skills. They'll be probably logical/ mathematical, probably in the form of moving or using shapes? (Look at 999 or prof layton for inspiration. Research of puzzles? Only if a lot of spare time).