Architecture

After deciding on the project requirements, we went about deciding which ones played a significant feature in deciding the architecture of our final product. We grouped them into 3 main groups: technical contraints, project constraints and quality attributes. We settled on these requirements as our **Architectural Drivers:**

- Technical Constraints (1.x)
  - (1.1) Mobile application
  - (1.2) Offline functionality
  - (1.3) 3D graphics
- Project Constraints (2.x)
  - (2.1) Java based project
  - (2.1) Due 1st May 2020
- Quality Attributes (3.x)
  - (3.1) High poly 3D models (Lisence Free)
  - (3.2) Smooth performance (Across multiple devices)
    - (3.2.1) Constant high framerate
    - (3.2.2) Clear UI
  - (3.3) Concrete fucntionality testing
  - (3.4) Concrete system testing

In order to fulfill these architectural drivers, we employed the following programs:

- Android Studio
- Processing 3
- Maya

Key libraries to the architecture of the project include:

- PApplet
- Apache Commons .CSV library
- JUnit

Software

Android Studio:

Android Studio fulfilled a large portion of the architectural drivers as a Java based (2.1), mobile application (1.1) development and distribution platform (2.4). Android Studio also provide a plethora of mobile application support (1.1) such as touchscreen functionality libraries. It also provides a plethora of UI tools which allow us to create and operate the separated **MainActivity** front end systems (which include systems such as our main menu, level select, settings menu)(3.2.2). As well UI tools and libraries, it provides seemless support of both modular (JUnit)(3.3) and system testing (APK build testing)(3.4).

Processing 3

Due to the Java based constraints, we have chosen to use Processing 3 as our 3D graphical engine. Fortunately, it was very easily integrated into a larger Android application environment, making it ideal for mobile application development (1.1). Moreover, known for its efficiency, we thought it would be quite good in keeping graphical rendering overheads to a minimum, especially key when designing software to run on smaller, less powerful mobile processors (3.2.1). Processing also provided seamless support for universal 3D model file types (3.1).

Maya

Due to the experience of some of our team members, we decided on using a 3D modelling software to create our very own 3D models (3.1). This eliminated the problem of procuring license-free models. Maya is a software with quite a lot of learning support which made the learning curve a lot easier for the teammembers in charge of developing models to develop high 3D assets within the time frame of the project (2.2). Moreover, Maya's exporting system included a file type which was fully supported by processing which made translating those models into our graphics engine easy. Maya also created low impact model files which would help a lot when it comes to rendering performance in tandem with Processing 3.

Libraries

PApplet

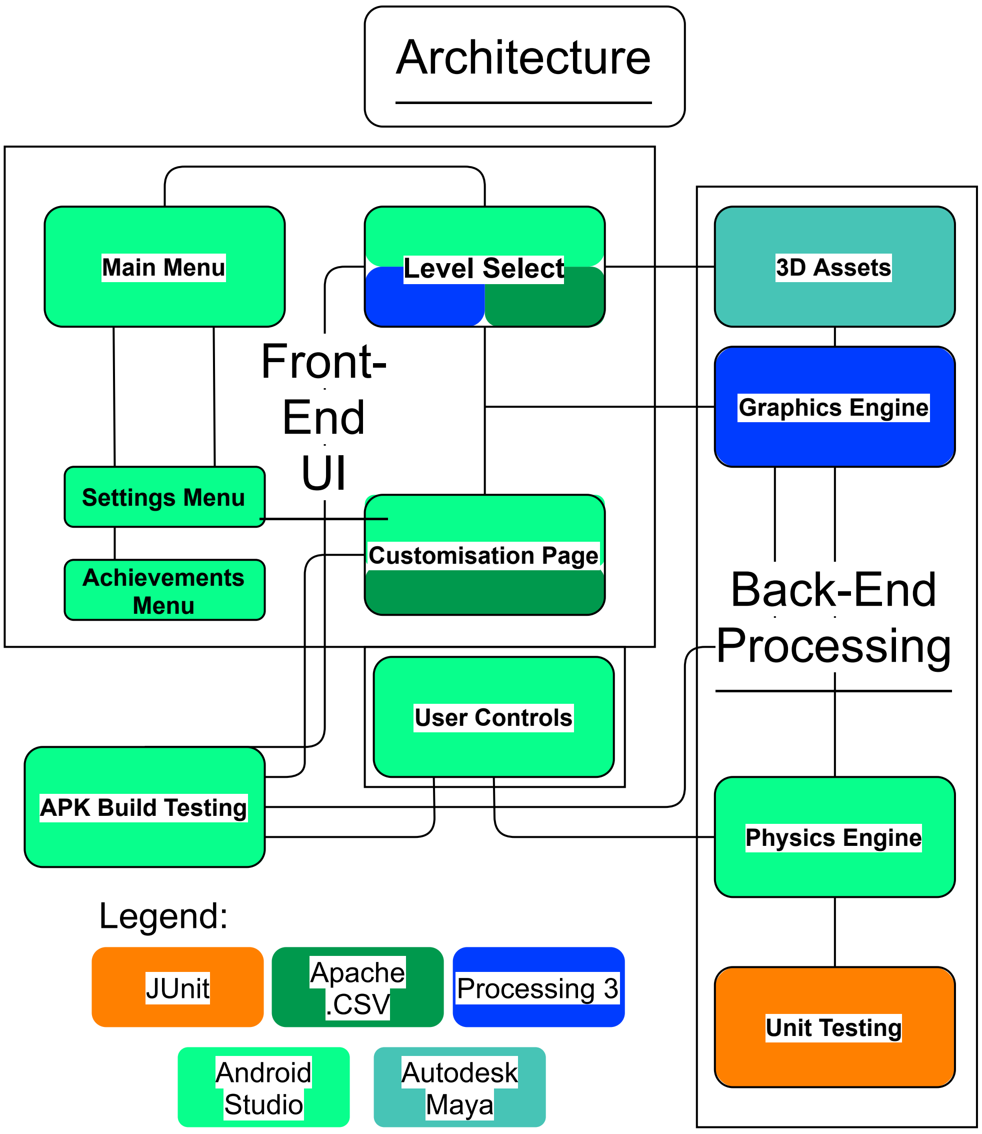
The PApplet library provides the Processing.core (3.1, 3.2) fucntionality to java based projects. Importing this allowed us access to all of processing's functionality as a graphics engine in our Android Studio environment (1.1) resulting in the graphical engine backend of our game (which we dubbed **GameSketch**).

Apache Commons .CSV

We planned to utilise the universality of .CSV files as a means of efficient level creation and asset arrangements in our game (2.1). This library provided us with the necessary .CSV parsers we would need.

JUnit Testing

We used the JUnit Testing library to create tests in order to uphold the Test Driven Development approach we wanted to take. It allowed us to create class specific tests which would ensure the integrity of the methods we were creating and using (3.3).



```
Figure 1: Architecture Diagram
```

Figure 1 shows the different front and back end systems we will be creating. Our front-end systems will include the **Main Menu** which provides links to the **Settings, Achievements, Level Select** and **Customsations** systems. All will utilise the touchscreen navigation libraries present in Android Studios to enable user interaction. Interactions in **Level Select** and **Customisations** will interact with the various Apache .CSV parsers we will be using in order to determin the state of the loaded game; which in turn will interact with the **3D Models** and **Graphics Engine** to display our game. **User Controls** is in its own box as it is a unique user interaction, only working in parallel with game rendering. An Android UI is overlayed the game render to read user controls which pass into the **Physics Engine** to be processed in the back-end. The **Physics Engine** uses the inputs from the **User Control** overlay in order to calculate the state of the game every frame through a series of mathematical functions which calculate things like position, acceleration and the presence of collisions/ interactions. Due to the independent operation of these functions, the **Physics Engine** was the only module which could be tested via **Unit Tests** as all other aspects were intrinsicly tied to external systems like the **Graphics Engine**/ **3D Models** making them difficult to test in isolation. The rest of the systems developed were tested in both functionality and integration via **APK Build Testing** due to their dependencies. **APK Build Testing** allowed us to create and test builds of the game, using system displays to test interactions between systems and to also test gameplay feel (a testing criteria which is quite important due to the nature of our product).