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) Playability
        - (3.2.1) Constant high framerate
        - (3.2.2) Clear UI
        - (3.2.3) Interactabilty
    - (3.3) Concrete fucntionality 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:

- Porcessing Core
- 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 and UI tools which allow us to create and operate front end systems (which include 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) as well as our graphics engine (1.3).

Processing 3

We chose to use Processing 3 as our 3D graphical engine as it was easily integrated into a larger Android application, due to it also being java-based (2.1), making it ideal for mobile 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

We decided on using a Maya software to create our very own 3D models (3.1) to eliminate the problem of procuring license-free models. Maya is an easy to learn piece of software, allowing the teammembers in charge of developing models to develop quality 3D assets within the time frame of the project (2.2).

Libraries

Processsing

The Processing Core library provides graphical engine (3.1) fucntionality to java based projects. Importing this allowed us access to all of Processing's functionality to create the 3D rendering loops in the game (1.1).

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

To accomplish Test Driven Development, we used the JUnit Testing library in order to test independent features of the game (i.e. our Physics Engine) (3.3).

Figure 1: Architecture Diagram

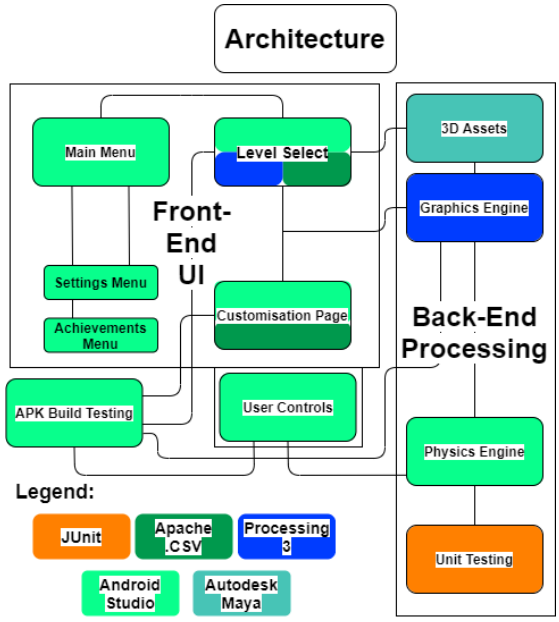

Figure 1 shows the different front and back end systems we will be creating. Our main front-end system will be the **Main Menu**, providing links to the **Settings, Achievements, Level Select** and **Customsations** systems (3.2.2). All will utilise the touchscreen navigation libraries present in Android Studios to enable user interaction (1.1, 3.2).

User decisiosn in **Level Select** and **Customisations** will interact the various CSV parsers we implemented (using Apache .CSV) in order to determin the state of the loaded game; which in turn will dictate the **3D Models** used by the **Graphics Engine** to display the playable level(3.1, 3.2.3).

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 on top of the game render to register user controls, passing 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 operations which calculate things like position, acceleration and the presence of collisions/ interactions (3.2.3).