# 2. The Engineering Project

## 2.1 Health and Safety

Doing this last...

## 2.2 Engineering Professionalism

Also doing this last...

## 2.3 Project Management

Also doing this last...

## 2.4 Individual Contributions

### 2.4.1 Project Contributions

Contributions here...

### 2.4.2 Report Contributions

More contributions here...

# 3. Background and Terminology

## 3.1 Android

Android is a set of software which includes a mobile Operating System (OS), middleware, and required applications. Android is open source software (OSS), meaning the source code is freely available to the general public to use and modify so long as the licensing restrictions are adhered to.

### 3.1.1 Operating System

The Android architecture is layered in four key categories: applications, framework, libraries, and kernel. An additional fifth layer, the runtime layer, contains core libraries essential in any application or program, as well as the Dalvic Virtual Machine (DVM). [sl1]

The Android core libraries allow the Java programming language to be seamlessly implemented and used on Android mobile devices alongside the DVM. DVM is a non-licensed implementation of the Java VM which was built for mobile devices; it is small, efficient and optimized for mobile devices such as the HTC Desire. [sl1]

The libraries included in the Android software development kit (SDK) are:

* Libc: Standard C library;
* SSL: Secure Socket Layer;
* SGL: Custom engine library for 2D images;
* OpenGL/ES: An engine library for 3D images;
* Media: Library which deals with all multimedia intents;
* SQLite: Deals with SQL databases, used for data storage;
* WebKit: Library for web-based applications, such as browsers;
* FreeType: Deals with images and vectors; and
* Surface Manager: Provides surface holders for various applications. [sl2]

Of the nine listed libraries, two were used extensively (OpenGL/ES and Surface Manager) and will be discussed in further detail in sections 8.1 and 8.2.

### 3.1.2 Emulator

Android applications can be run either on the physical device or on the Android Emulator, which is shipped with the Android SDK. The main benefits of utilizing the Android Emulator are twofold:

* Integrates very well with the Eclipse Integrated Development Environment (IDE); and
* Eliminates a fair amount of “real-world” testing, as the application programming interface (API) is nearly identical to a physical mobile device.

Of course, there are some functionalities which even the Android Emulator does not implement, such as the built-in gyroscope of more modern mobile devices. This particular issue is covered in section 3.1.3 below.

### 3.1.3 Additional Libraries

In order to properly test the tilt functionality of the Android mobile device without access to such hardware, an open source project by OpenIntents was utilized. As mentioned in section 3.1.2, the Android Emulator does not implement or support the gyroscope functionality which the RoboWars project has taken advantage of. Fortunately, OpenIntents has already solves this problem by implementing their own open source project – Sensor Simulator – which directly addresses this issue.

The Sensor Simulator project allows the designer to “simulate sensor data with the mouse in real time. It currently supports accelerometer, compass, orientation, and temperature sensors, where the behavior can be customized through various settings.” [sl3] As the API for this particular library is nearly identical to the API of the actual Sensor Event Listener class, modifying source code to use this library in place of its hardware counterpart is extremely easy and involves modifying only a couple of lines of code. [sl3]

# 8. Android Client Implementation

## 8.1 Android Libraries and Application Layout

### 8.1.1 Android Libraries

As mentioned previously, the two main libraries utilized in this project (on the client side) are the OpenGL/ES library and the Surface Manager library.

#### 8.1.1.1 Surface Manager

Android’s surface manager is quite elegant in that it does not draw directly to the screen buffer, but rather forms the entire screen layout before drawing anything at all. In doing so, the developer has much more flexibility in terms of implementing graphical/visual effects such as transparent windows and transitions.

There are many built-in view layouts, such as a RelativeLayout, LinearLayout, and TableLayout, all of which were utilized in the RoboWars project. All of the layouts support all of the built-in widgets, and are very easy to understand, implement and use.

#### 8.1.1.2 OpenGL/ES

The OpenGL/ES library shipped with Android is extremely useful, versatile and relatively well-documented. Although support specifically for OpenGL/ES is slightly more difficult to find than other Android documentation, OpenGL/ES conforms to the OpenGL 1.5 standard, which is very well documented and widely used. Section 8.2 discusses further detail about OpenGL/ES.

### 8.1.2 Application Layout

The layout of the RoboWars Android application is relatively simple. The majority of the view components and characteristics are stored in an XML file; this approach keeps the source code clean and simple, and at the same time improves reusability. The remainder of the classes are generally models, for storing data, and controllers, for handling user input. Below is a list of classes and a brief overview of their functionality and purpose:

***Views***

* RoboWars.java: This is the main application class that is loaded and executed when the Android application is run. Its purpose is to identify the view (in this case, the file main.xml), handle tilt functions, and to make calls to OpenGL methods to render 3D objects on to the screen. It also inheritably handles basic user-device interaction, such as touch screen events, button presses, and so on.
* ImageStreamView.java: A relatively simple class which is used for displaying a bitmap image (sent from the MediaClient) updated in real time. Essentially the surface holder for live video feed.

***Controllers***

* TcpClient.java: An active class which handles incoming information from the application server, as well as sending out updates of the current models. All critical information is sent via serialized objects over the input/output stream; non-critical information, such as chat messages, are sent as UTF-encoded Strings.
* MediaClient.java: The MediaClient is responsible for handling incoming UDP packets regarding the live video stream. Its sole purpose is to handle these packets and to reconstruct the video feed.

***Models***

* LobbyModel.java: Keeps track of all users and robots in the lobby, as well as all chat and game events, which are displayed in the chat interface.
* ClientGameModel.java: Keeps track of all game entities and the state of the current game. This includes players, various obstacles, projectiles, as well as locations, life, ammunition, etc.

***Passive***

* Player.java: Used to store in-game player information, such as their name. Used by the ClientGameModel.
* User.java: Used to store out-of-game player information, such as name, ping, flags, etc. Used by LobbyModel.

***OpenGL***

* OpenGLRenderer.java: A custom OpenGL renderer class which takes a particular Mesh (see below) and renders it on to the OpenGL view (set up in RoboWars.java).
* Group.java: A simple storage and access class to manage a group of Mesh objects (see below).
* Mesh.java: A super class which can be implemented to form any type of 3D polygon. Used to map vertices, textures, and indices, and defines how to draw the object. Also controls the position and the rotation of the object.
* Cube.java, Plane.java: Polygon classes which have extended the functionalities of the Mesh class.

## 8.2 OpenGL Rendering

OpenGL/ES is a cross-platform API that renders both 2D and 3D graphics on embedded systems. Theoretically, any OpenGL program created under the OpenGL 1.5 standard (or earlier) should work in OpenGL/ES, therefore there exists plenty of documentation for OpenGL/ES that is relevant to mobile developers. [sl4]

Rendering 3D objects in OpenGL is relatively straight-forward, yet takes some time to learn and adjust to. An OpenGL drawing starts as a set of vertices, which define the corners of the polygon. Next, a set of index triplets are supplied; these define the order in which to connect the vertices. When texturing is enabled, another set of indices must be supplied, these ones relative to the polygon and not an absolute coordinate system. Once the shape of the polygon is defined, it must be placed and rotated. It is easier to think of the OpenGL surface as moving and not the actual polygons; placing a polygon at (0,0,0) will place it exactly where the previous polygon was placed. Further, once one polygon has been rotated and placed, all subsequent polygons will retain the same rotation. This is due to the fact that the polygons are not being rotated at all, but the OpenGL surface is.

## Development Issues and Solutions

### Android Emulator

Two prominent issues with the Android Emulator came up, both during implementation and during testing. The first, and most easily handled, was the fact that the Android Emulator has no built-in gyroscope emulator or stub. As the RoboWars mobile application depends on the tilt controls for steering during a session, either the application needed to be tested on a physical mobile phone, or an alternative software solution was needed. After some research, an open source library, named SensorSimulator, was found and used. Not only was this library very well documented, it also turned out to be very easy to implement into any existing project. Assuming source code already exists in the project which uses the hardware gyroscope, only a couple of lines of code require modification in order to switch between the SensorSimulator and the actual hardware gyroscope. The application, as seen in Appendix X, allows the user to control the tilt of the phone along all three axes through a software interface. The application communicates with the RoboWars project through a TCP connection, so information transfer was not an issue, as it was already fully implemented and working.

The second, and more prominent issue regarding the Android Emulator, is the way it handles the OpenGL interface. Unfortunately, the emulator only draws the very first frame of an OpenGL scene, which means that none of the rendering or moving of objects can be tested over the Android Emulator. Unfortunately a solution was never found for this problem; the remainder of OpenGL testing was done through actual Android hardware as opposed to the emulator.

# 9. Testing

## 9.4 Android Client Testing

The Android client was tested mainly through the use of the Android Emulator software, and later via physical mobile devices. During the first half of the project, in testing various elements of the design of the Android application such as scrollbar functionality, button presses, and basic user interaction, it was crucial to take advantage of the Android Emulator; this allowed for quick, easy and efficient testing of the application’s interface without any prior knowledge or experience in Android application development. <expand further>.

# References

[sl1] *An Introduction to Android* Huang Huguang, Inha University. Nov 2, 2009.

[sl2] <http://developer.android.com/guide/basics/what-is-android.html> Accessed 23 March 2011.

[sl3] <http://code.google.com/p/openintents/wiki/SensorSimulator> Accessed 23 March 2011.

[sl4] <http://www.khronos.org/opengles/> Accessed 23 March 2011.