Intermediate Report

Vi-Char Mobile Development Division

Michael **DuBois**
Nathan **Pastor**
Robert **Shapiro**
Davis **Shurbert**
Kirah **Taylor**

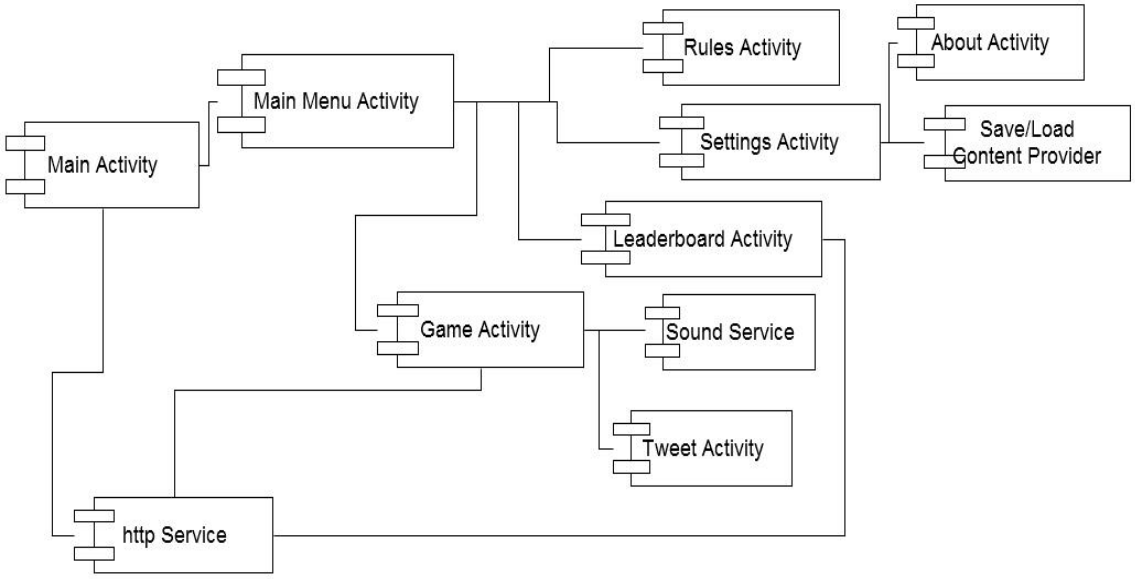
Abstract

This document describes the mobile division's progress thus far in the implementation phase of the project. It will discuss implementation details, a reflection of our current situation and past challenges, and a timeline for the rest of the project.

Android owners who wish to participate in the game will be able to load and install the application onto any mobile phone running Android version 2.2 or higher. The main function of the application is to allow users to view and interact with the game's augmented reality world. The the actual AR view will be implemented by a separate division, but the user's interaction will occur mostly through touch inputs which we implement. These touch inputs will affect the gameworld.

Other functionality includes user profiles. Users will be encouraged to log into our app through Twitter, allowing them to post gamestate altering tweets during the actual gameplay. However, if the user does not have a Twitter account, they can create a ViChar account and their information will be stored on the server. In addition, the user will also be able to access leaderboards and game information before and after a game.

Design Summary



The MainActivity is the first screen the user sees when opening the application. It directs the user to a “Login with Twitter” prompt or a “Login with Vichar” prompt, the Vichar option is for people who don’t want to use twitter for some reason (don’t have account, ect.). After the login is complete, the user is directed to the Main Menu activity, which provides buttons which allow the user navigate to the Game Activity, Rules Activity, Settings Activity, Leaderboard Activity, and back to the opening screen.

The Settings Activity uses the Save/Load Content Provider (Preference Utility) to store information to be persistent across all instances of running the application. Examples of this information might be booleans such as “is_logged_in” and “is_muted” so the user doesn’t have to login to Twitter every time they launch the application or re-mute the game when playing in a quiet setting.

The HttpService binds to necessary activities to allow communication with the web server. Right now this service only binds to (stays running while the activity is running) the game activity, but this activity can track touches and continuously update coordinates to the server. Eventually, the HttpService will also bind to the Leaderboard to receive username, score, and date information.

The rules activity is simply a text view at this point, but hopefully it will be extended to a full tutorial of our game and possibly be able to receive updates to the

rules by our http Service, through the web server. Similarly, the about activity just contains a string of the names of members of the mobile group, but ideally it will accept text from the web server so every class member can put something on the “about” screen.

The sound Service, not yet implemented, will bind to all activities to allow for constant sound throughout the user experience. Simultaneously, its main goal is to add sound to the game activity in the form of ambience, music, or event triggered sounds. The Tweet Activity is a Fragment Activity that is also attached to the Game Activity, which means it can be pulled up from the bottom of the screen at any time during game play. This will allow the user to post tweets while playing the game.

The GameActivity houses the logic for displaying and interacting with the game. In addition to managing our UI and game-related subviews, the GameActivity will act as a controller that manages network interchanges between our app and the server with the assistance of either the HttpService, or the SocketService.

This project utilizes the basic Android architecture and application components, including activities, services, intent objects, and content providers. All these components will reside in one Java package (edu.pugetsound.vichar). However, in the interest of promoting modularity, we intend to integrate the AR module as a separate Android library package, if feasible -- there are some limitations on what resources a library can carry along with it, but it should work for our needs. At the very least we will maintain the AR view as a separate Fragment view -- a child Activity-subclass that maintains its own logic. Beyond the possible addition of a SocketService for TCP socket communication in the GameActivity, our original architecture design remains mostly unchanged.

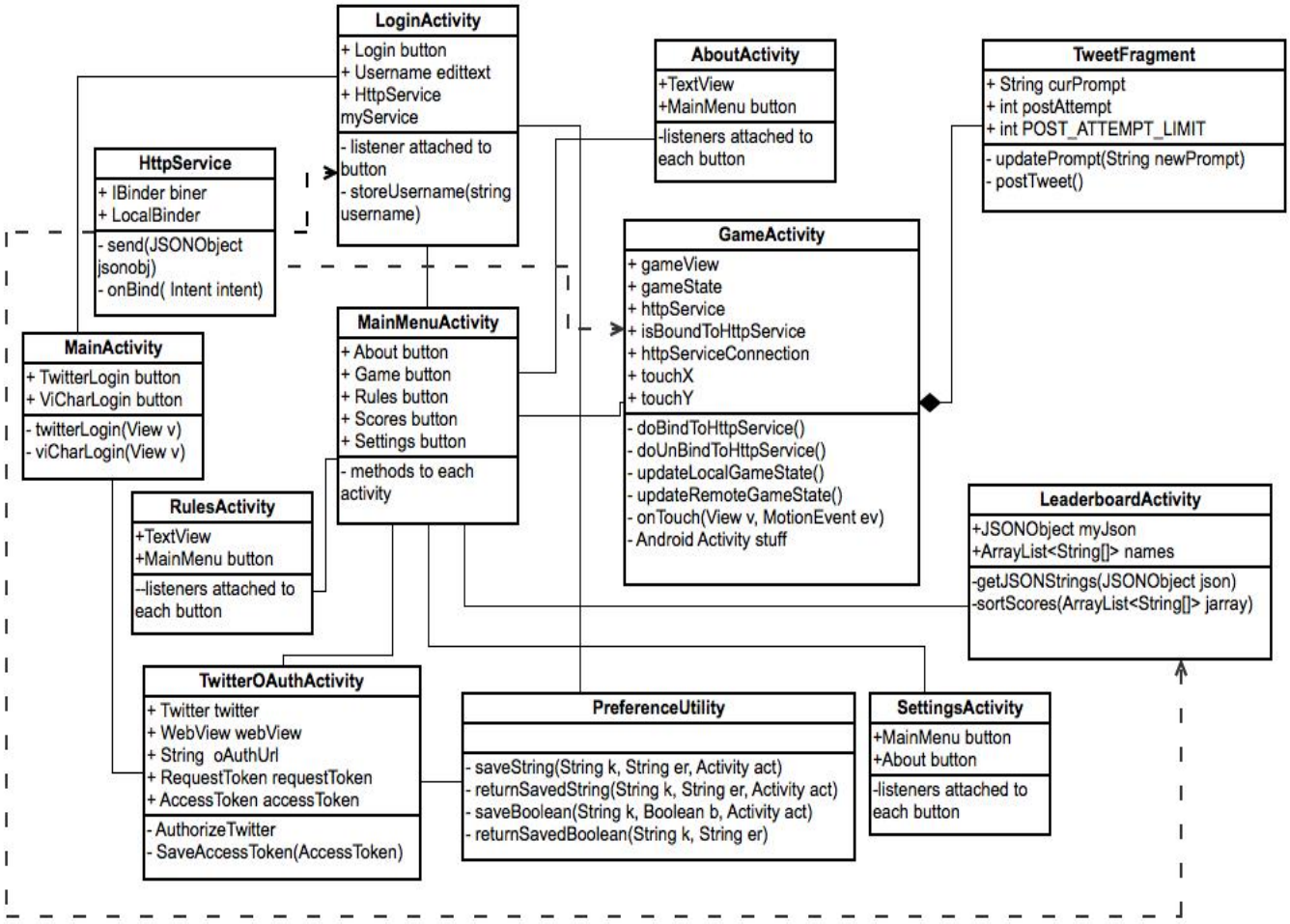
Install Documentation

Currently, the application (and associated documentation) must be retrieved from the ‘mobile’ Github repository. This is a private account, so the organization administrator must first grant permission to those seeking the application. Once a developer has this access, they can fork or clone our source code with a Git client, or they can download it in a compressed folder via the ZIP button in the Github web interface.

With the source code on one’s local machine, there are several ways of installing it on a device. The steps we will outline here require the Android SDK and Eclipse with the Android SDK plugin (Links for these downloads and installation instructions can

be found below in the glossary). After these are installed, our code can be loaded into Eclipse via File > Import > Android > Existing Android Code into Workspace. Once done, the application can be run in an Android virtual device or on a physical Android device if available. (Instructions for setting up an Android device can be found below in the glossary.)

Implementation Details



This section will follow the main use case for our program, and explain the implementation details corresponding to each section of the program.

The first screen that the user sees on startup is the login page. Here the user can choose to either login through Twitter or through the ViChar server. If the user chooses the former, they are presented with the Twitter login page.

This page, ***TwitterOAuthActivity***, is implemented through a dedicated activity. Notably, several methods of this activity extend AsyncTask methods. AsyncTask is included in the Android library and provides a simple means for implementing worker threads. This is used here because as of Android 3.0, http transfers cannot be executed in the UI thread. AsyncTask provides methods for moving sets of instructions, in this case OAuth requests, into separate worker threads. Furthermore, we are using an external library (Twitter4j) to execute the guts of the OAuth http authentication.

After a successful authentication, the OAuth access token and access secret are stored to persistent storage on the device. These two values can later be used to post tweets on behalf of the user, or bypass the login screen during the next game session. These values are stored in key/value pairs using the Android SharedPreferences class, which provides a means of storing primitive types, in this case Strings. We implement this class in our project with the ***PreferencesUtility***.

If instead the user chooses to login through the ViChar server, they will be presented with the ***LoginActivity*** which will give the user the ability to input a username which will soon be able to be sent to the server for authentication via our ***HttpService***. Once this is implemented, if the username does not pass authentication, the user will be prompted to select new usernames until they have one which is unique, and it will then be stored, just like the OAuth access token and access secret of the Twitter login method, via the PreferencesUtility.

The HttpService is a simple service that can accept data that needs to be passed to the server and request data from the server. Its methods accept JSON objects, which are then taken by the HttpService and sent to the server using the simple pre-existing class HttpClient and the method HttpPost (both from the Apache library that comes bundled with Android) in a separate worker thread.

Sometime before or after the game is played, the user can access the ***LeaderboardActivity*** to view past scores and the usernames associated with these scores. As of now this activity takes a dummy, one dimensional JSONObject, sorts it according to the values, and puts the sorted keys and values in a table. The code is designed, as of now, to ignore “bad” JSONObjects to avoid crashing the program. However, it will implement a more robust JSON parsing solution once we develop a general use JSON parser for all the data our app consumes from the server.

The ***GameActivty*** represents the main game view. It implements the Android FrameLayout, which stacks views consecutively in the order they are added to the layout. This allows us to place the AR view at the lowest z-index and overlay our own UI components over top of it. In order to subjugate the AR Module, we have encouraged the AR team to modularize their code within an Android Fragment -- a subclass of Activity which can be owned by another Activity. As we have not yet integrated with AR, the GameActivity currently houses a PlaceholderFragment. In order to implement Fragments as far back as Android 2.2, we have drawn on the Android support library, which packages backwards compatible FragmentActivity and Fragment classes in the final application file (apk).

The GameActivity currently registers itself as the OnTouchListener for all its subviews -- including the AR/PlaceholderFragment. This allows it to first delegate touch actions to the game-server component, then handle the necessary callbacks for internal components, like the AR module, upon receiving a response from the server. For now, the GameActivity is bound to the HttpService, through which it makes rudimentary POST and GET requests. However, in the next iteration, we may need to bind it instead to a ***SocketService*** which connects to a TCP websocket that maintains an open connection. Making this change will undoubtedly improve performance, so it remains a high priority to coordinate with the web server group to get the Input/OutputStream functions in the SocketService to working order.

The “general layout” describes the way in which the user navigates between the activities. The biggest issues with this come from not only supporting various versions of Android, but making the app as uniform as possible between different versions. An example of this is the use of the Action bar vs. built in menus or buttons. The Action bar is generally considered the best way to navigate between screens, however it wasn’t introduced until version 4.1 or so. We are currently attempting to bring in an external library or support package in order to implement the Action bar in 2.2.

Another consideration in the implementation of the user interface is whether to implement a View in .xml or .java. There are advantages and disadvantages to both, but basically any layout information specified in the .xml cannot be changed programmatically. So anything that must be updated while the app is running must be done in .java, the leader board table for example. The .xml is fairly easy to use and is perfect for buttons and text in the about activity. We use the .xml for Views that are

constant every time we run the application.

Procedures Followed

Thus far, each member of our team has implemented different components. Kirah has finished working on the structure of the HttpService and its post method and is now working on implementing the ViChar login and differentiation user inputs. Robert has been working on the get method in the HttpService. With help from Michael, Nathan has implemented the TwitterOAuthActivity, and he is now working on the tweet fragment which will reside in the GameActivity. Michael has been implementing the GameActivity and SocketService. Davis has been working on the general layout and user interfaces, the LeaderboardActivity, and is responsible for all the Activities that aren't covered above.

With some difficulty, we have been using the git-flow model for version control. Commands for this model can only be executed from the shell, and the learning curve has been rather steep for most group members. Fortunately, Michael has substantial experience with this model and has been thoroughly helpful.

We haven't done extensive testing on any of our code yet, the focus still being the implementation of our core components. However, the testing that has been done has occurred mostly on the Android Virtual Device (AVD), with some verification also occurring on the actual Android devices. We do, however, have a defined set of procedures that we will follow for issue tracking, and which provide a consistent format across all issues (bugs, features etc). These procedures are explained on our Github Wiki.

Lastly, our code is currently loosely following the Android coding style. This will be adhered to more thoroughly in the future, and existing code will be modified to fit the prescribed guidelines.

Reflection

Generally, the consensus is that we should have started familiarizing ourselves with the Android development environment and git (and in some cases Eclipse in general, as its use came with several unexpected and time consuming issues) a lot sooner than we did. As a result, we were slower starting off the implementation component of our system's development than we would have liked.

Along this vein we have wasted some time in “false starts” with several of our components. For example, Twitter authorization was initially built into an alert dialog, an implementation that failed. If we had invested more time into planning the actual architecture of our project, we may potentially have saved some time in these areas.

Now, however, our implementation phase is in full swing. Now that we are familiar with Android Java, the SDK, and have a more visceral understanding of the project’s requirements, we will be quicker and more effective in our implementation.

That said, many significant challenges remain ahead. Aside from the general Android UI and social networking tasks we always anticipated, the prospect of integration with AR and the webserver suggests new responsibilities. It seems likely now that our team will have to manage packet loss and game-time synchronization between the game-server component and our mobile-AR component. In turn, we may need to implement our own animation loop to manage the Augmented Reality view (depending, of course, on whether or not the AR module will have it’s own animation loop). Similarly, in response to the need to manage and parse arbitrary game data from JSON, we may also need to implement some sort of game update function that will pull game entity templates and 3d assets from the game-server component to avoid a more precarious code-duplication solution. In the coming days, our team will work hard to sort out these implementation details with members from the other teams.

Timeline

Our current project schedule, in the form of a Gantt chart, appears at the end of this document.

Our timeline has gone through some significant changes as ideas have been refined and new knowledge of how the application will need to function has come to light. Multiple goals, new and old, have already been met, such as creating a menu system and integrating with the server, and some that had previously seemed daunting now obviously won’t take as much time, such as touch inputs and sound. However, many continue and look as if they shall need to be long running, such as the game activity, which is highly complex and robust and will need much more time and effort than other activities. The menu system will also likely take a longer amount of time because although it sounds like a much simpler task, it includes such necessities as layout styling, which is made time consuming by the fact that other groups and persons need to be

consulted with so that the theme of the overall game is consistent across modules.

Glossary and References

A Service is a component which runs in the background, without interacting with the user. Every developer can create new Services in his application. Services support true multitasking for Android, as they can run in their own process. If you use threads in *Activities* they are still connected to the life-cycle of *Activities* and the Android system may decide to terminate them at any point in point. (http://www.vogella.com/articles/AndroidServices/article.html#service_overview)

There are two kinds of services. A bound service attaches itself to an activity and is running and accessible as long as the activity its bound to is running. A start-stop service has one specific task and terminates when completed. A service can bind to more than one activity.

A fragment is a subclass of an activity. It is basically an activity that another activity can own. An example of this is the ability to pull down a text input screen from the game activity in order to post tweets.

A View is an instance of the android.View class, which "Provides classes that expose basic user interface classes that handle screen layout and interaction with the user." Essentially, views are what you *see* when you run an Activity. (http://developer.android.com/reference/android/view/package-summary.html)

TCP, or Transmission Control Protocol, is one of the core services of the Internet Protocol Suite. It is used by major internet applications and provides reliable, connection-based data stream service.

An OnTouchListener is an "interface definition for a callback to be invoked when a touch event is dispatched to this view." (From Android website)

JSON, or JavaScript Object Notation, is a common design for data exchanged supported by the Android library.

Download Links:

Eclipse Juno for Java http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junosr1

Android SDK

http://developer.android.com/sdk/index.html

Hardware Android Device Setup Instructions:

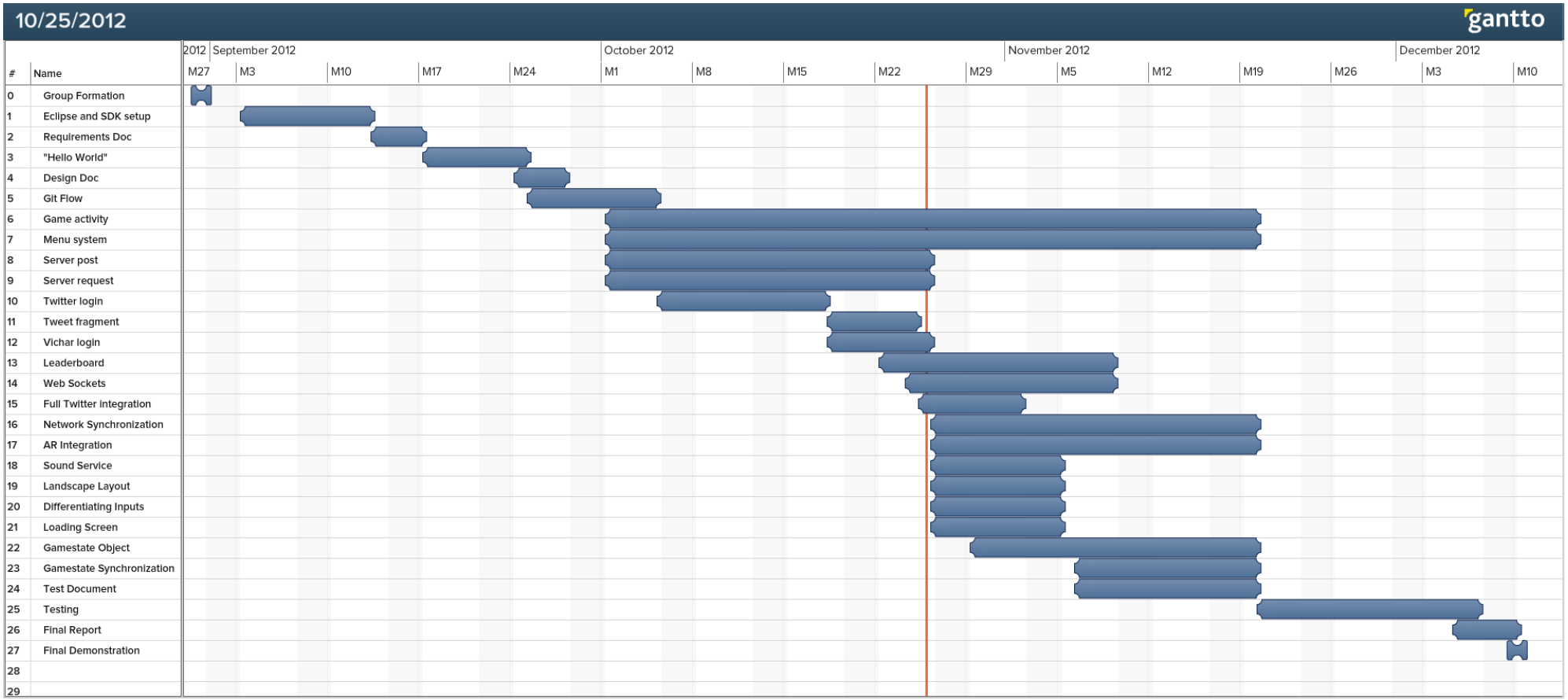
http://developer.android.com/tools/device.html

Eclipse Android SDK plugin Setup Instructions

http://developer.android.com/sdk/installing/installing-adt.html

Twitter4j (Android OAuth Java Library)

http://twitter4j.org/en/index.html


10/25/2012
gantto
2012
September 2012
October 2012
November 2012
December 2012
M27
M3
M10
M17
M24
M1
M8
M15
M22
M29
M5
M12
M19
M26
M3
M10
Name
0 Group Formation
1 Eclipse and SDK setup
2 Requirements Doc
3 "Hello World"
4 Design Doc
5 Git Flow
6 Game activity
7 Menu system
8 Server post
9 Server request
10 Twitter login
11 Tweet fragment
12 Vichar login
13 Leaderboard
14 Web Sockets
15 Full Twitter integration
16 Network Synchronization
17 AR Integration
18 Sound Service
19 Landscape Layout
20 Differentiating Inputs
21 Loading Screen
22 Gamestate Object
23 Gamestate Synchronization
24 Test Document
25 Testing
26 Final Report
27 Final Demonstration
28
29