WarGPS: An Android Game that Incorporates GPS Location Services

David Nussbaum

dsn516@gmail.com

1. Introduction

As the future of computing moves from the traditional personal computer to that of devices in the mobile realm, the mobile device becomes a more important part of our lives. Since the release of the first iPhone in 2007, mobile devices have become heavily adopted. The adoption rate at which both Android and iOS device have been adopted among users has out-paced the adoption rate of the PC revolution from the 1980s and the Internet boom from the 1990s.[6] Within the smart phone market, adoption of Android smartphones is increasing six times faster than iPhone adoption.[8]

In the spans of 5 years since the first Android phones was released, Android has achieved the largest share of the smart phone market. A major reason for this large share is that the Android OS is used by many manufactures, which each make android phones.

The current generation of Android devices has many different sensors and features to try to improve the users experience with the device. A basic requirement for all smart phones is that they have the ability to get connected through a data network or wifi. These creates great potential for interconnectivity between apps run on different devices Current phones rely on a touch screen for a majority of all user inputs, which is very beneficial to applications. All current mobile devices have location settings built into them. These settings vary from using GPS to figure out your location to using your network to triangulate a rough location. A GPS lock tends to be more accurate and precise. Within Android apps, Google has made it easy for developer to use Google Maps within their app.

For my senior project, I want to explore this evolving field. So, I decided to build an android application. The hope was to use many of the features and sensors to develop an understanding of how they work and how to incorporate them into an app to influence what happens. I was able to incorporated GPS location functionality, server connectivity, and sever communication, as part of my project. The goal of my project was to take a simple game and try to improve it with location-based functionality.

2. Background

Android is an operating system for mobile devices built by Google. To write an application for Android, you must download the Android SDK and an IDE. Android apps are written in Java, so the IDE must support Java. At the current time, Google provides a file with the Android SDK setup in the Eclipse IDE.[2] The consists of the ADT (Android Developer Tools) plugin which is the plugin to Eclipse IDE, Android SDK Tools which is the complete set of development and debugging tools for the Android

SDK including the emulator, Android Platform-tools, and the current Android platform. The Android SDK feature provides developers with features such as downloading older versions of the Android platform for testing, ability create and use emulate devices for app testing, and code example for newer developers.[2] The file can be found at on the Android developer website. Alternative IDEs to Eclipse exist such as Android Studio, which is currently in beta.[3]

Google App Engine is a cloud computing platform for developing and hosting web applications and server backends in Google-managed data centers.[4] App Engine support users to write code in the following languages Java, Python, PHP, and GO. App Engine provides services and features such as Datastore, Google Cloud Storage, Mail, Url Fetch, and Task Queues. App Engine provides a developer web console to monitor and manage the backend if they choose to. To use App Engine, you can use a plugin to install it into Eclipse, which can be found on the app engine website.

For mobile developers, Google Cloud Endpoints provides a simple way to develop a shared web backend.[7] It also provides critical infrastructures eliminating a great deal of work that would otherwise be needed. Furthermore, because the API backend is an App Engine app allowing developers to use all of the App Engine services and features. It is possible to create mobile clients for App Engine backends without Endpoints. However, using Endpoints makes this process easier because it frees you from having to write wrappers to handle communication with App Engine. The client libraries generated by Endpoints allow you to simply make direct API calls. To use Endpoints, you use the same plugin as used for App Engine and the Android SDK has to be installed as well.

3. Game

WarGPS is a game that combines the card game War with geo-location functionality. The game rules for WarGPS are that each player starts out with hand of 26 cards. Before the game, each player selects the location of his or her base. Each turn, both players place their top card into the pile. Based upon the location that they played, a bonus value is calculated based on the distance to the opponent's base. The card value is added to the bonus value to determine a final value. Whichever player has the highest final values wins all cards in the pile. If the both final values are the same, then both players put their next three cards on the pile and then keep playing with the next winner getting the cards in the pile. The two ways of winning are getting all your opponents cards or getting the instant win bonus.

The values for the bonus points follow:

- +1 for within 1000m
- +2 for within 500m
- +3 for within 250m
- +4 for within 100m
- +5 for within 50m
- Instant Win for within 25m

A victory by Instant Win takes affect at the end of a turn. If both players get an Instant Win in the same turn, the game is a tied.

Upon opening the application, an Account Screen is shown. (See Figure 1) It presents all Google accounts on the device. By selecting, the account, you are setting your account for the rest of application.

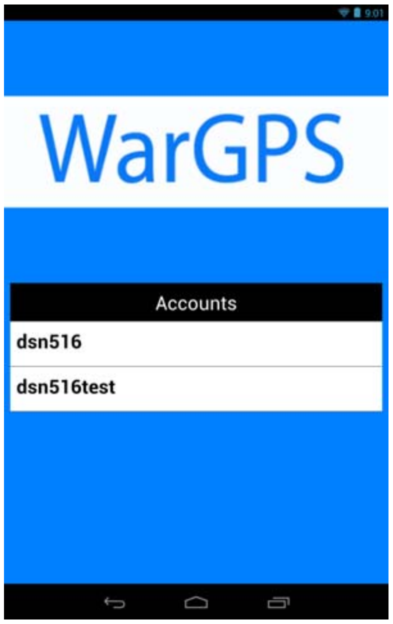


Figure 1: Account Screen (screenshot done on Nexus 7)

After completing the Account Screen, the Main Screen is presented as shown in Figure 2. The Main Screen gives the user the choice of three buttons. The New Game Button takes the user to the Setup Screen. The Rules button brings the user to the rules screen that presents the game rules. The last button is that of Current Games.

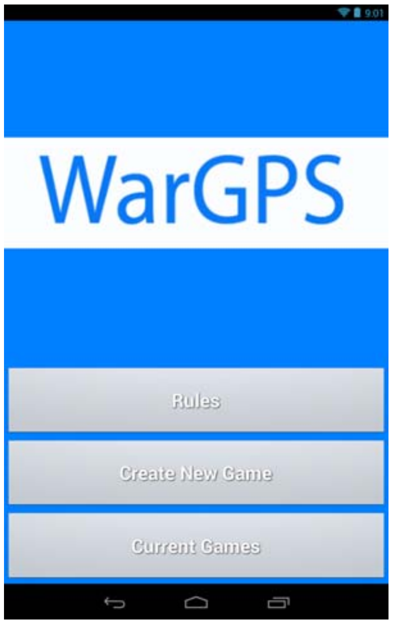


Figure 2: Main Screen

The Main Screen provides a list of all current games through the Current Games button user. (See Figure 3) The list will not be immediately accessible due to amount of time that it takes to get the list of games from the server. When a game is selected from this list, the application will launch the Confirm Screen, Location Screen, or Game Screen depending upon how setup the game is.

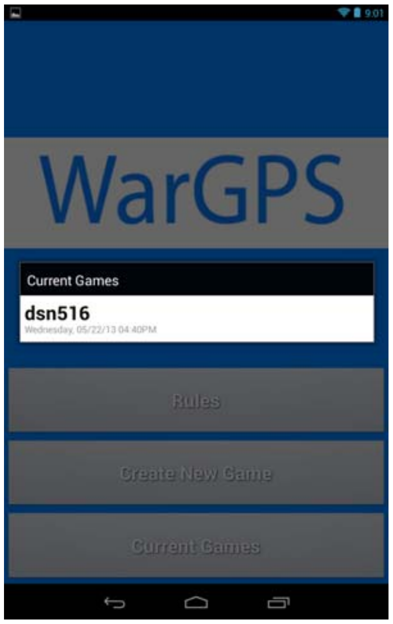
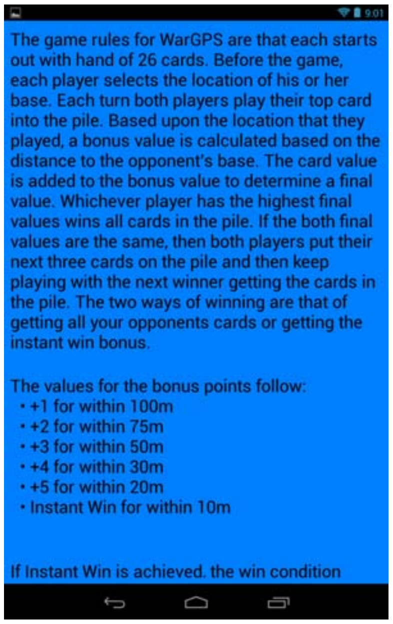


Figure 3: Main Screen with Current Games open

The Rules Screen is meant to provide the user with information about how to use the app, if they get confused. (See Figure 4) The instructions shown are similar to those included in this section.



The game rules for WarGPS are that each starts out with hand of 26 cards. Before the game, each player selects the location of his or her base. Each turn both players play their top card into the pile. Based upon the location that they played, a bonus value is calculated based on the distance to the opponent's base. The card value is added to the bonus value to determine a final value. Whichever player has the highest final values wins all cards in the pile. If the both final values are the same, then both players put their next three cards on the pile and then keep playing with the next winner getting the cards in the pile. The two ways of winning are that of getting all your opponents cards or getting the instant win bonus.

The values for the bonus points follow:
- +1 for within 100m
- +2 for within 75m
- +3 for within 50m
- +4 for within 30m
- +5 for within 20m
- Instant Win for within 10m

If Instant Win is achieved, the win condition

Figure 4: Rules Screen

The Setup Screen is in charge of opponent selection by user name and selection of game grid location as shown Figure 5. The map allows the user to select a point that serves as the center of the game grid circle. This circle serves as grid of allowed locations. The opponent selection is a text view where you type the username in. Once both actions have been performed, the done button will activate allowing the user to register. When the Done button is selected, a server communication occurs to add a new game onto server. If the server returns successful, the application will move to the Location Select screen. Upon server failure, a notification will be thrown up stating that the opponent player does not exist.

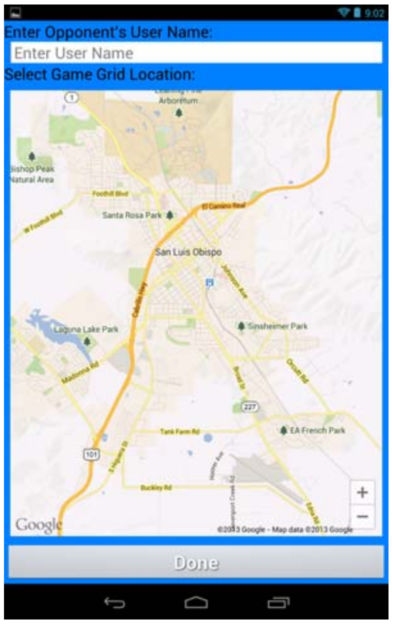


Figure 5: Setup Screen

The Confirm Screen is in charge of the opponent user confirming both the users playing and the game grid location. The map shows the circle that represents the game grid.   The textbox lists the other user playing the game. This screen presents two choices via two buttons. The Cancel button makes a request to the server to delete this game and then the application moves to the Main Screen. The Confirm button makes a server request that confirms the game. Once the request is complete, the application launches the Location Screen. The Confirm Screen can be seen in Figure 6.

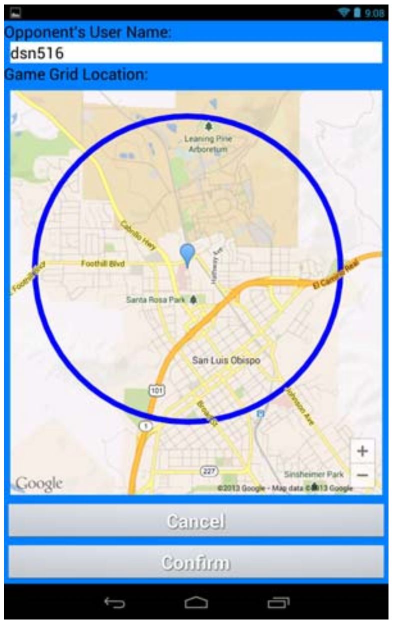


Figure 6: Confirm Screen

The Location Select lets the user select their opponent's goal location as shown in Figure 7. The goal location must be within the valid area within the game grid. As a result, the map will only allow you to select within the circle on the map. When the done button is clicked without goal being set, a notification is shown telling the user to set a location. If the done button is clicked with the goal set, a server call is made to add the goal to the game. Once complete, the user is taken to the game screen.

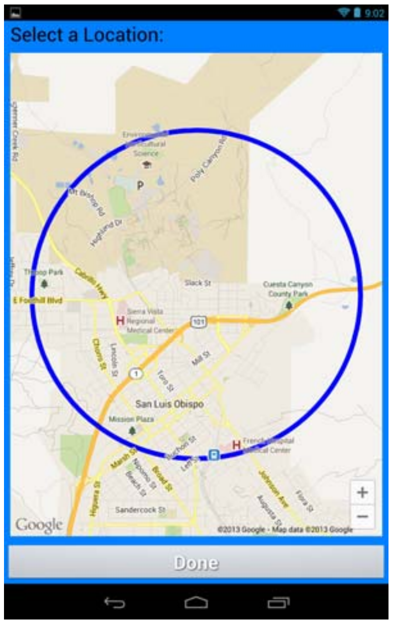
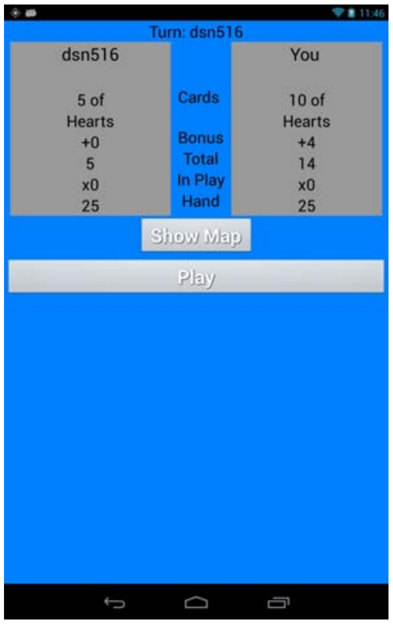


Figure 7: Location Screen

The Game Screen is where the game is actually played as shown in Figure 8. The screen contains the game information including turn, most recent card in play for each player, most recent bonus value for each player, total value for each player, current the number of cards in play for each player, and the hand size for each player. The total is the sum of the card value plus the bonus. The screen shows two buttons on the screen. The Show Map button takes the user to the Map screen. While the Play button can only be pressed if it is the users turn. When pressed, notifications will be shown if GPS is off, no GPS location established, or GPS location out of the grid. If none of these conditions are triggered, the game gets the user's location and calculates a bonus. Then, it updates and sends the information to the server. When complete, the game information will update to show the play that occurred.



Turn: dsn516

dsn516		You
5 of Hearts	Cards	10 of Hearts
+0	Bonus	+4
5	Total	14
x0	In Play	x0
25	Hand	25

Figure 8: Game Screen

The Map screen shows a map with the user's current location and all previous points that the user has played from. (See Figure 9) The previous points provide information about their bonus values in two ways. The first ways is that the marker color is dependent on the bonus value, so that each bonus value has a different color. The colors for each bonus value are:

- Azure for +0
- Blue for +1
- Violet for +2
- Magenta for +3
- Orange for +4
- Rose for +5
- Yellow for Instant Win

Second, the bonus values can also be viewed by clicking the marker to see the bonus value of that marker. The Done button takes the user back to the Game Screen.

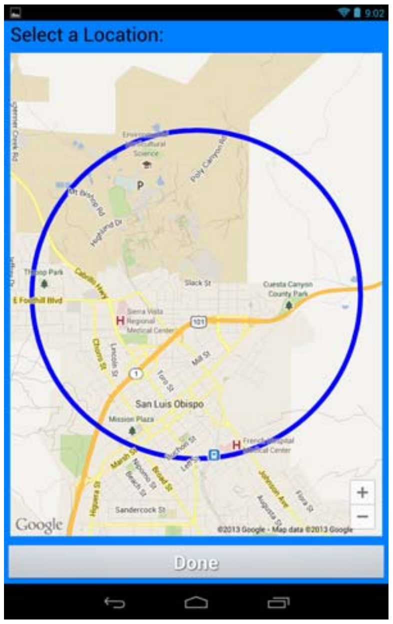


Figure 9: Map Screen

4. Design/Implementation

To make the Android game support competition between multiple users, a server for communication is required. This requires the application to consists of two sides: an Android side and Server side.

4.1. Android Side

The android side consists of the pieces of the application that the user will interact with directly. The android side consists of a number of activities that are connected to one another. A general overview structure of how the activities are connected to each other is expressed in Figure 10. Whenever a server request is shown in the flowchart, this means that a call was made to server. We will now go through the activities individually in greater detail.

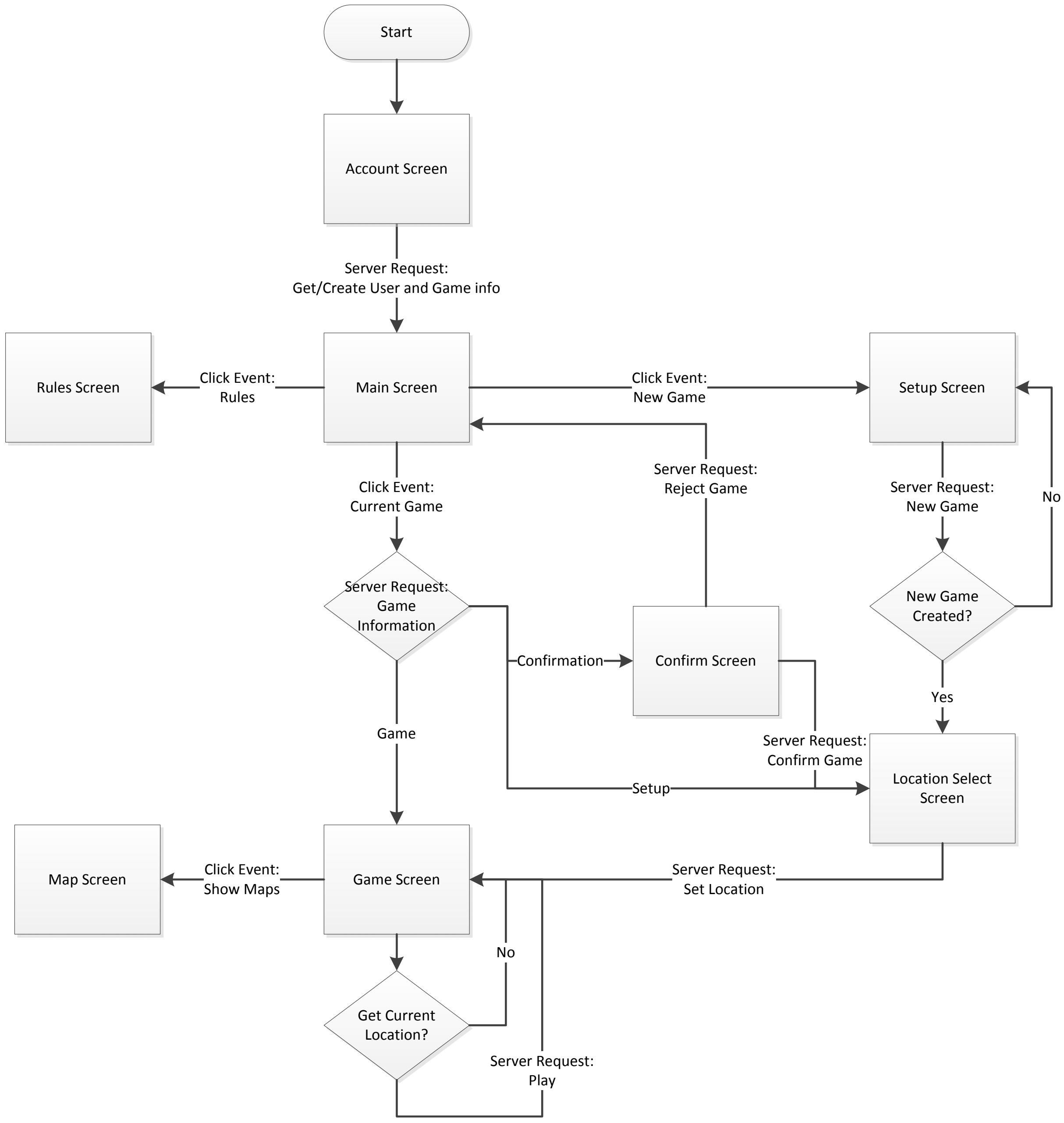


Figure 10: Android Side Flowchart

Early in the design of the application, a need for authentication between players presented itself. To deal with the problem a solution was conceived to use the Google account for authentication, since all Android devices are required to have one. In developing the Google account recognition, the potential of multiple Google Accounts occurs. To resolve this problem, an Account Screen was created, as the first screen in the application so that the user could choice which account that they want to use.

Within the Main Screen, the user is given a choice of where they want to go within the application. The base design and elements where decided upon early on. Most of the UI elements such as buttons and images were coded statically. The list of current activities had to be implemented dynamically, since the games can’t be predicted. As a result, a ListView is used to display the information. The ListView uses a custom ArrayAdapter that shows the opponents name, the time of last move, and the current turn for each element within the list.

To populate the ListView, the information must be downloaded from the server. To complete this task, three server calls are made to deal. The first server call, insertUserLookup, is used to make the user as valid player within the game. The second server call, queryGameInfo, is used to get a list of all games where the user is the first player. The third server call, queryGameInfo, is used to get a list of all games where the user is the second player. The second and third calls combine to create a full list of current games. The amount of time taken by the server calls can be excessive. Due to the time consuming nature of server calls, they are all run on different Async Tasks rather than the main thread in a hope to decrease the lag time. As a result, there is detection for when the calls are completed before letting the current game button be clicked. These server calls tend to be the slowest in the application. Due to the need of updating this list often to show changes, these server calls are also called on when the activity is restarted.

Upon a list item being selected, the GameInfo needs to be checked to see the state of the game. Many of these states are different parts of the setup. The first state is that of configured unset. This results in the Confirm Screen being called. The second state is that of grid location not set.   This state causes in the Location Screen being called. If both of these values are set then, the game state is triggered. The game state calls Game Screen. The flowchart of Main Screen can be seen in Figure 11.

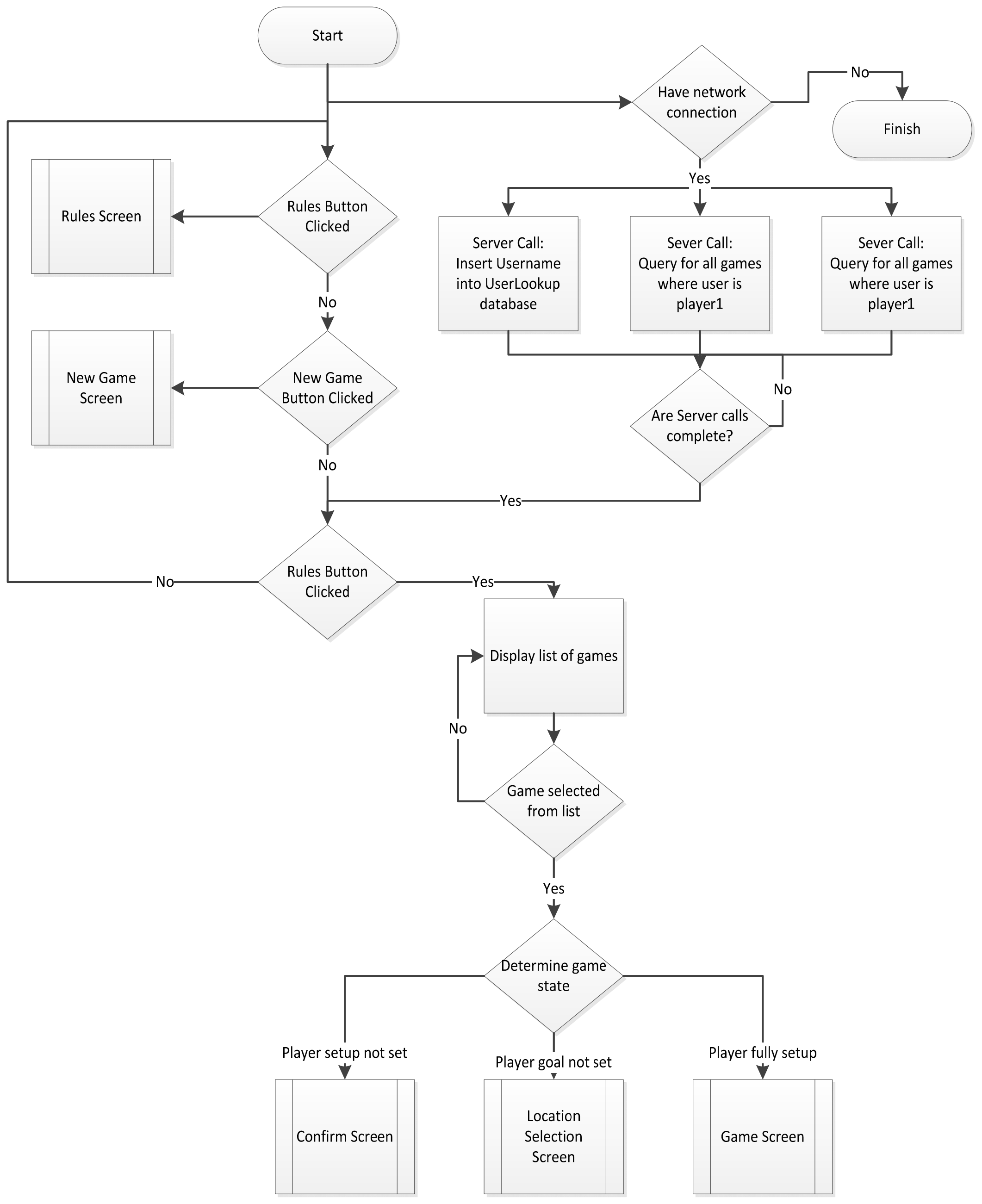


Figure 11: Main Screen Flow Diagram

Early in the design the need for Google Maps and location integrations was realized. To figure out potential problems, a simple app was designed to use Google Maps and location services to explore the features that WarGPS would have. This features consisted of:

- accessing and setting up Google Maps Android API V2[5]
- creating a map
- placing a marker and the properties related to the marker

- using GPS to get the device's current location and detect if the current location has been found yet
- using a listener to get latitude and longitude of location from a spot clicked on the map
- drawing a circle on the map around a latitude longitude location
- calculating distance between two latitude longitude locations and understanding the units of the result
- turning off screen rotation
- turning off action bar

The knowledge of brought obtained from this app provided the skills easily implement Google Maps into many activities. Within Setup Screen, a map is set up that shows a circle with radius 3600m around a user selected point.

Within the Setup Screen, the ability of game creation exists as expressed in Figure 12. Three items are needed to make a game. Those items are two different player names and a game grid location. Upon the game being created, a shuffled deck of cards is created and 26 cards are dealt to each player's hand. The gameId is set as 0. Once all this has been completed, a server call, insertGameInfo, is called. This call adds the game to the GameInfo database. On success, the call returns the game back with a new valid permanent gameId. Otherwise, the call returns a null value meaning that opponent player is not on the list of active players.

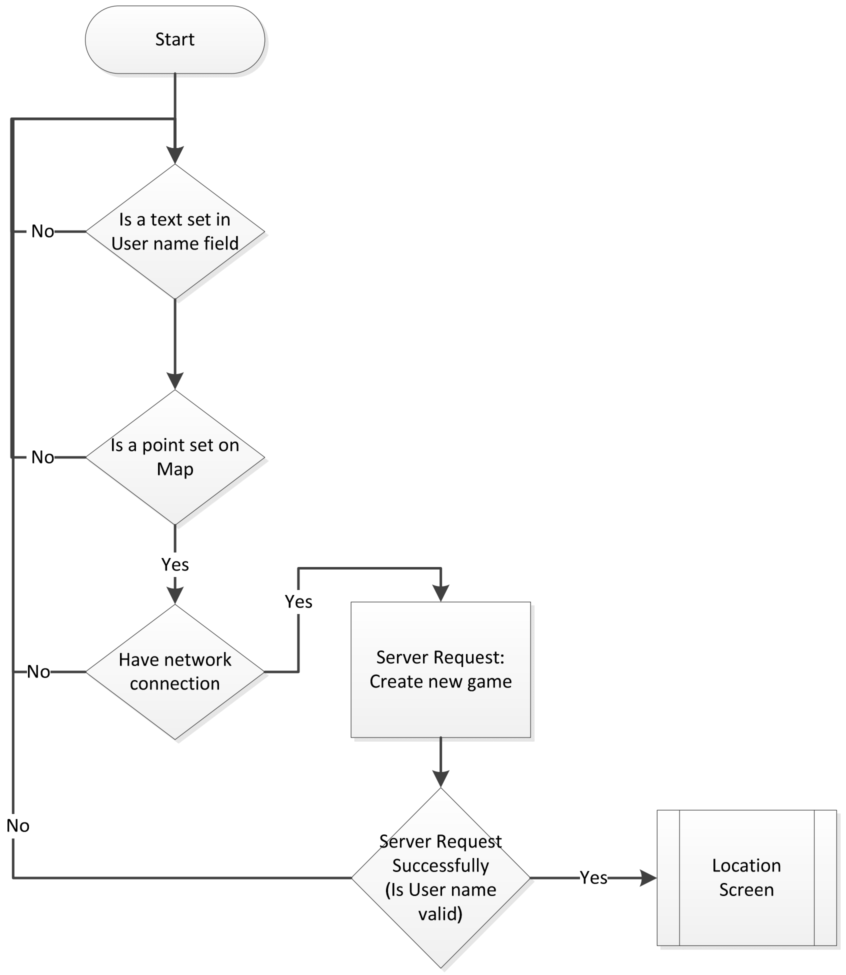

Start
Is a text set in User name field
No
Is a point set on Map
No
Yes
Have network connection
No
Yes
Server Request: Create new game
No
Server Request Successfully (Is User name valid)
Yes
Location Screen

Figure 12: Setup Screen Flowchart

Originally the plan was to pass the game data between the different activities using intents. Unfortunately due to issues, just the player name and gameId are passed across to other activities using intents.   To get the full game information, the gameId is used to make a get request to server for this game data. This get call is only important to activities that make a update server call, such as Game Screen and Confirm Screen.

As development progressed on the application, the need for a Confirm Screen presented itself. It is an opposing screen to the Setup Screen, so that its UI was made to look as close to Setup Screen, as possible while serving its specific purpose. The screen presents the possibility of confirming or deleting a game as shown in Figure 13. In terms of confirming, two server calls are made. The first server, getGameInfo, is a request for a specific game. Once this call is completed, the player configure is set. Then the second server call, updateGameInfo, is called so that changes made are on the database.

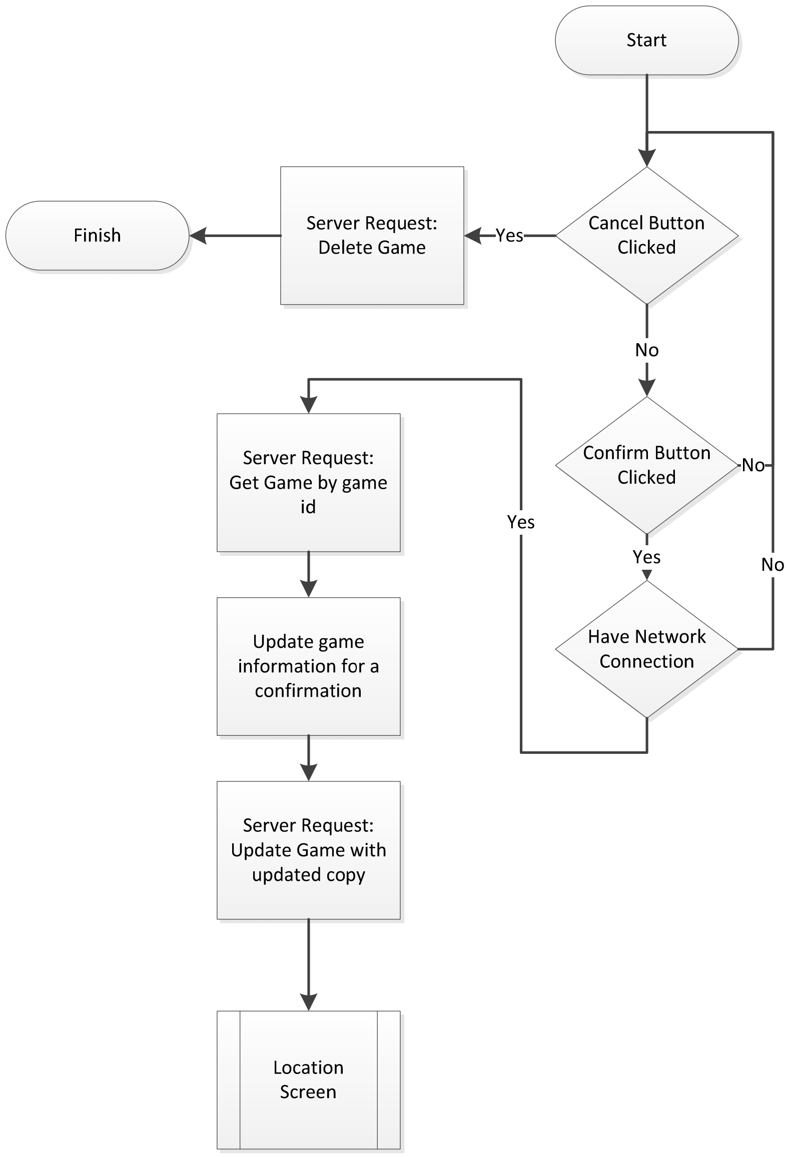

Start
Finish
Server Request: Delete Game
Yes
Cancel Button Clicked
No
Server Request: Get Game by game id
Confirm Button Clicked
No
Yes
Yes
No
Update game information for a confirmation
Have Network Connection
Server Request: Update Game with updated copy
Location Screen

Figure 13: Confirm Screen Flowchart

The Location Screen was built very similar to that of the Setup Screen. They both have map functionality and won't progress to the next activity until a location is selected. Unlike the Setup Screen map, the Location Screen map show a fixed circle of radius 3600m around the game grid and the user can only select values within that circle. The calculation of whether or not you are in the circle is actually latitude longitude distance function figuring out the distance between the goal and your location. This function returns a value in meters.

On the Game Screen start, a server call, getGameId, is made to get the game. Once this is completed, the information is used to populate the UI. The “Play” button will only be active on the players turn. On “Play” button clicked, the GPS is used to make sure that you are within the valid game area. On success, the bonus value is calculated based on the GPS location compared to the goal location.

Within the GameInfo, an update function is called. This function holds major logic of what a play is and what it does. It first checks whether the game is completely setup. If this fails the logic stops. Otherwise, the player turn is changed. If both players have gone, then the war logic triggers to figure out which player won the cards. By taking the card value and adding the current location bonus to it for each player. If one player has all the cards and none are still in play, then that player wins. If all of both players’ cards are in play, then the game is a tie. In the case of one player setting an Instant Win, that player wins the game. In the case of both players setting an Instant Win, the game would result in a tie. In all the cases of win or tie, the finished variable is set to true and the winner, if any, is recorded in the winner variable. Once these variables are set no new plays can be made to this game. After update function completes, the update GameInfo server call is made and updated data is used to repopulate the UI with new information. The Game Screen is shown in full in the flowchart in Figure 14.

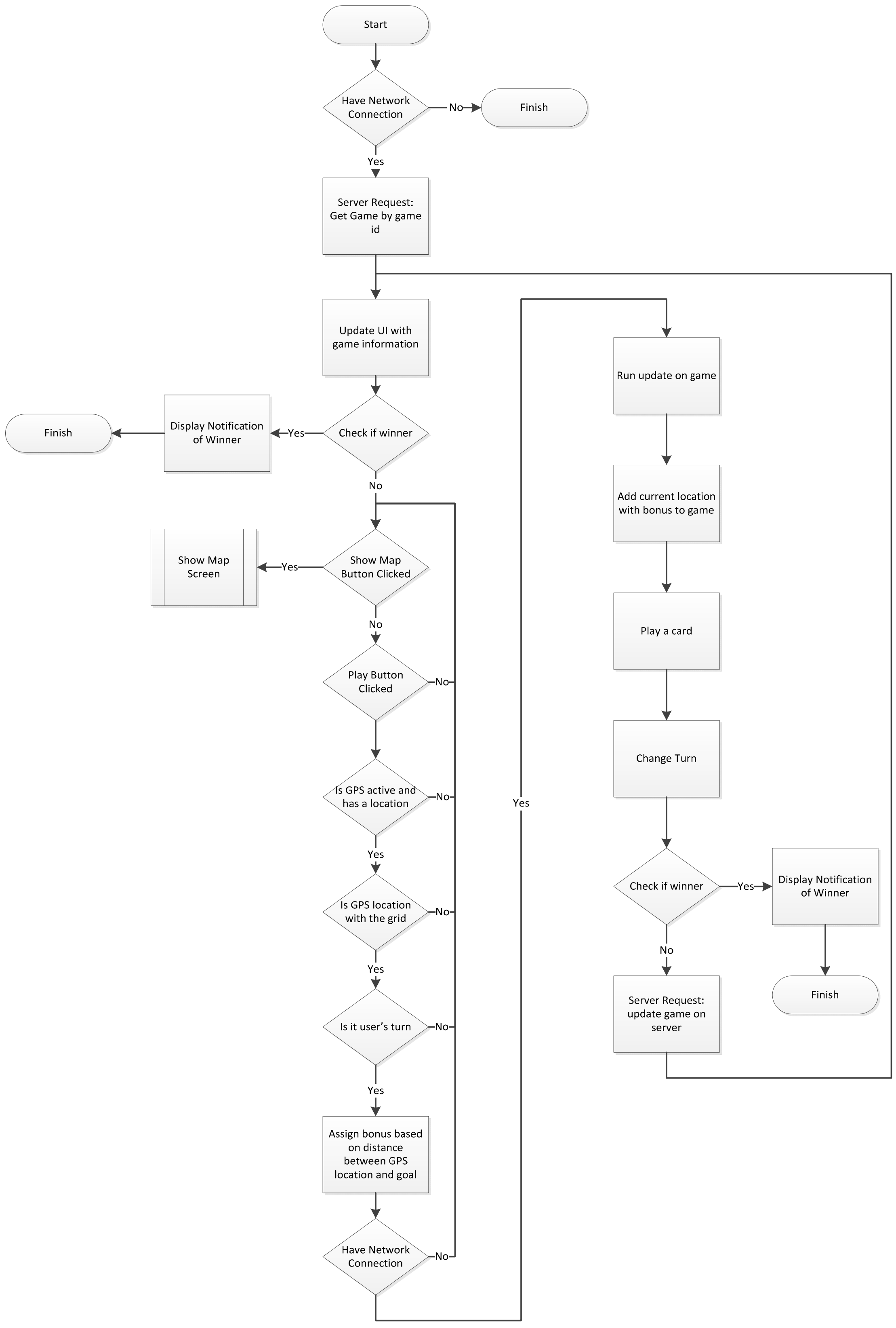

Start
Have Network Connection
No
Finish
Yes
Server Request: Get Game by game id
Update UI with game information
Check if winner
Yes
Display Notification of Winner
Finish
No
Show Map Button Clicked
Yes
Show Map Screen
No
Play Button Clicked
No
Is GPS active and has a location
No
Yes
Is GPS location with the grid
No
Yes
Is it user's turn
No
Yes
Assign bonus based on distance between GPS location and goal
Have Network Connection
No
Yes
Run update on game
Add current location with bonus to game
Play a card
Change Turn
Check if winner
Yes
Display Notification of Winner
Finish
No
Server Request: update game on server

Figure 14: Game Screen Flowchart

4.2. Server Side

The server side is designed to act as means of storing game state between devices. For the server side, the key functionality is to verify a request and hold game information. The server side was written in Java using Google App Engine[4] for the server and Google Cloud Endpoints[7] for the connection between server and device.

In terms of storage, App Engine generates a database from a Java class. On the server side, there exist 2 different databases. The first database, UserLookup, is meant to contain a list of all possible active players that a game can be started with. An entry in UserLookup contains a String for the userName.

The second database, GameLookup, is meant to hold the current state of all game. An entry in GameInfo contains:

- gameId (unique identifier/key for each game)
- count (session key/ increments by 1 every time server update is called)
- user1 (name of the player that created the game)
- player1 (player1 specific information)
  - goal (Latitude and Longitude values of the goal place by opponent)
  - location (array of Latitude, Longitude, and Bonus values of places played)
  - hand (ArrayList of the cards in the hand)
  - inPlay (ArrayList of the cards currently in play)
  - setUp (whether the player has setup yet)
- user2 (name of the player that didn't created the game)
- player2 (player2 specific information with same structure as player1)
- grid (Latitude and Longitude values of center of the playable center)
- turn (which players turn it is)
- lastTime (the time of the last play/update)
- configured (game setup complete)
- finished (whether game is finished)
- winner (name of player that wins)

By a request from the user, the eclipse will create base API calls for you as well and the configuration of the application to make these calls. The API calls that were use were a combination of new calls and modified existing. On the server side, communication is done thorough a list of generated API calls:

- queryGameInfo – API call that takes in a legitimate query and returns the results if any to the client
- insertGameInfo – API call that adds game data to database and assigns a game Id. If one of the players does not exist will not add the data.
- getGameInfo – API that returns a game based on given a id value
- removeGameInfo – API call that removes a game based on given a game Id
- updateGameInfo – API call that updates an existing game with new updated data only if the incremented count of the new data is greater than the count of the existing data
- insertUserLookup – API call that add user to database of allow users

For the communication between the two sides, it is very important that procedures for communication are established and implemented correctly. When the android side wants to create a game it makes insertGamenfo call to the server side as show in Figure 15. The result of the call can be a success that a game was created or a failure. This call is only need to be made on a single client. If a second player makes this call a second game will be created.

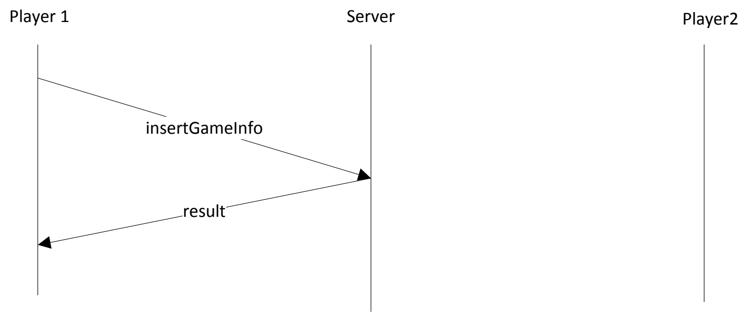


Figure 15: Server Client Diagram of Game Creation

For the Confirm Screen and Location Screen, there is chance that if the getGameInfo for each player are made before either update, you could overwrite the first update with the second update. (See Figure 16) To deal with this problem, a count variable was added to the GameInfo.   The purpose of it is to increase every time an update occurs and if the count on a GameInfo data that is being update is not the same as the one on the server then the game will reject the second update and thus prevent the first update form being overwritten. The second user can just make another getGameInfo.

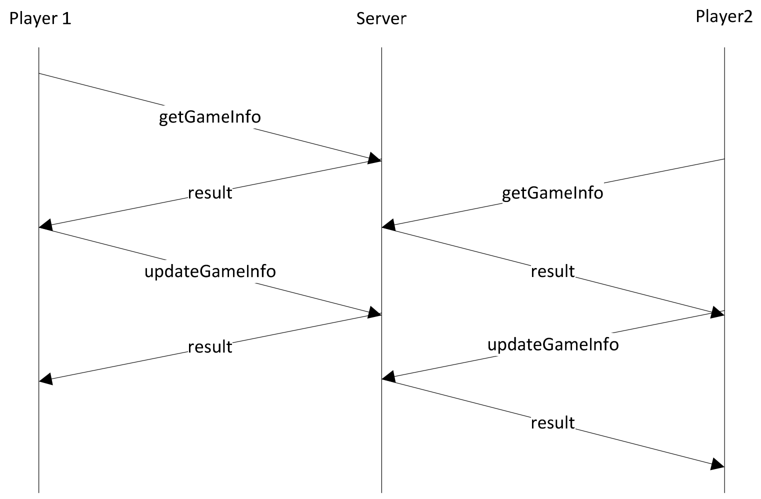


Figure 16: Server Client Diagram of Multiple Overlapping Game Updates

For the Game Screen, this problem does not occur. Instead of using the count variable, a user is only allowed to play/update if it is that use’s turn. (See Figure 17) As a result you don’t get overlaps due to this check occurring on the Android side.

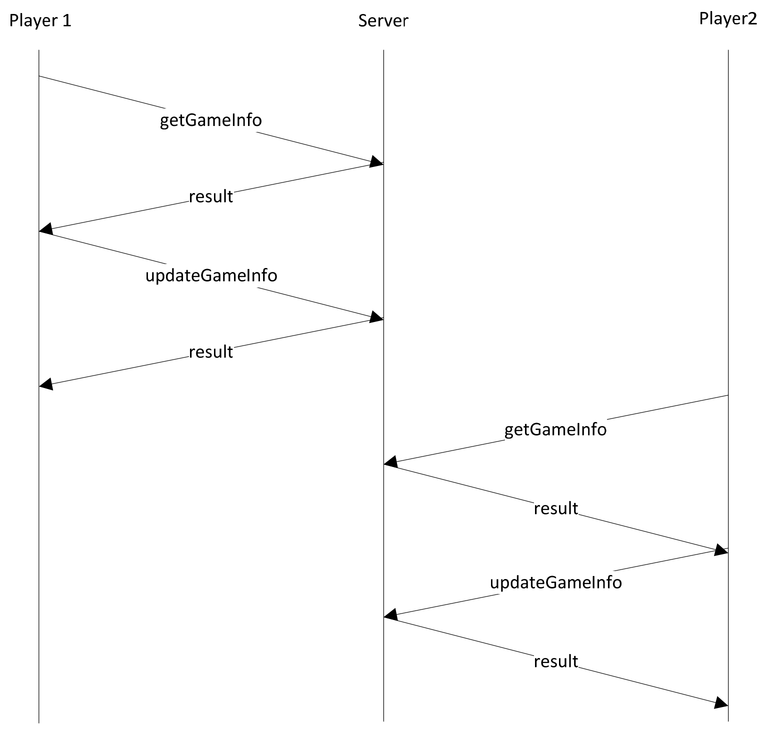


Figure 17: Server Client Diagram of Multiple Non-overlapping Game Updates

5. Analysis

After being allowed to use the app, a test group of five participants were given a survey. When asked how fun they thought the app was their answers varied from fun to so-so. These answers were split pretty evenly with 2 participants saying fun and 3 participants saying so-so. When asked about how intuitive the interface is, the response was an almost unanimous response of somewhat intuitive. One participant thought the interface was very intuitive. When asked if they would download this app, they all responded as maybe. When asked if they would pay for this app, they all responded no. This matches what I felt about this app. When asked what things that they would change about the app, the number one response was to improve the connection for the Game Screen, so that the user doesn't have to reload the Game Screen to see changes. Other suggestions were add a random opponent option, different modes, adding GPS locate button to all the maps, and better directions.

6. Related Work

Ingress is a location based online multiplayer game with two teams. Each team can gain and hold territory by complete mission by going to certain location. My project varies from this because the point of my game is to figure out where the set location without it being given to you. The strengths of the game are that it presents a clean UI and is intuitive to use.

GeoEmpires: GeoCaching Battle is a location based online multiplayer game where you conquer real world territories, collect tolls from your opponents, and build your own empire. My project varies from this because the point of my game is to figure out where the set location without it being given to you. The strengths of this game are that it is well put together.

Tourality Free GPS Challenges is a location based game where the player has to find geographic spots within their area before their opponents to earn points some sort of points. My project varies from this because in my game the player does not know the location that the spot is placed and the card game war is integrated on top of the location functionality. The strength of the game is that it presents a clean simple UI.

7. Conclusion

Starting from a position of never having any experience writing android apps or servers, I gain a lot of knowledge and skills. My approach of break the project into smaller steps to accomplish and then rolling them in all together made the project easier to accomplish. I learned how to write an android app and how to create a server using App Engine. Also, I learned how to use Android Google Maps API within an app and use location functionality. I discovered how to enable communication app and server. I learned how to use online server management tools and how to perform a survey on an application.

8. Future Work

In the future, there are many improvements that I would make to this app. I would improve the app GUI in particular the Game Screen. I would have the players reset their goals after 10 plays. I would modify the Account Screen so that it put account name into shared preferences. This would allow the account screen to only appear on the first time that game started and after that it would go straight to the Main Screen. The last thing that I would add is a way for the server to send notification to the devices that there has been an update or that it is the players turn. If the notification is received while not in the app in the Game Screen, then a notification would appear in the notification bar stating that it is your turn. If the notification is received in the Game Screen, then make a server call to get current game information. To implement, this I would use Google Cloud Messenger, a tool that allows a server to send a message to an android device or a group of devices.

9. References

1. “Android Developer," [online] 2013, http://developer.android.com/index.html (Accessed: June 7 2013).
2. “Android SDK | Android Developers," [online] 2013, http://developer.android.com/sdk/index.html (Accessed: June 7 2013).
3. “Getting Started with Android Studio | Android Developers," [online] 2013, http://developer.android.com/sdk/installing/studio.html (Accessed: June 7 2013).
4. “Google App Engine — Google Developers," [online] 2013, https://developers.google.com/appengine/ (Accessed: June 7 2013).
5. “Google Maps Android API v2 — Google Developers," [online] 2013, https://developers.google.com/maps/documentation/android/ (Accessed: June 7 2013).
6. Mlot, Stephanie. “Smartphone Adoption Rate Fastest in Tech History,” (PCMAG), [online] August 27, 2012. http://www.pcmag.com/article2/0,2817,2408960,00.asp (Accessed: June 7 2013).
7. “Overview of Google Cloud Endpoints - Google App Engine — Google Developers," [online] 2013, https://developers.google.com/appengine/docs/java/endpoints/ (Accessed: June 7 2013).
8. Reed, Brad. “Android adoption is growing 6x faster than iPhone,” (BGR), [online] November 6, 2012. http://bgr.com/2012/11/06/android-adoption-outpacing-iphone/ (Accessed: June 7 2013).