Systems Document

for

Manhunt

Prepared by Musa V. Ahmed, Ariel Diaz,   
Justin Philips, and Matthew Santiago

Team 3

August 1, 2014



# Table of Contents

[Table of Contents 2](#_Toc394700354)

[Executive Summary 3](#_Toc394700355)

[1. Introduction 4](#_Toc394700356)

[1.1. Purpose of system 4](#_Toc394700357)

[1.2. Scope of system 4](#_Toc394700358)

[1.3. Development Methodology 5](#_Toc394700359)

[1.4. Definitions, Acronyms, and Abbreviations 5](#_Toc394700360)

[1.5. Overview of Document 6](#_Toc394700361)

[2. Current System 7](#_Toc394700362)

[3. Project Plan 7](#_Toc394700363)

[3.1 Project Organization 7](#_Toc394700364)

[3.2 Hardware and Software Requirements 8](#_Toc394700365)

[3.3 Work Breakdown 9](#_Toc394700366)

[4. Requirements of System 10](#_Toc394700367)

[4.1. Functional and Nonfunctional 10](#_Toc394700368)

[4.2. Use case diagram 14](#_Toc394700369)

[4.3. Requirements Analysis 14](#_Toc394700370)

[5. Software Architecture 14](#_Toc394700371)

[5.1. Overview 15](#_Toc394700372)

[5.2. Subsystem Decomposition 16](#_Toc394700373)

[5.3. Hardware and Software Mapping 17](#_Toc394700374)

[5.4. Persistent Data Management 18](#_Toc394700375)

[5.5. Security/Privacy 19](#_Toc394700376)

[6. Detailed Design 19](#_Toc394700377)

[6.1. Overview 19](#_Toc394700378)

[6.2. Object Interaction 24](#_Toc394700379)

[6.3. Detailed Class Design 26](#_Toc394700380)

[7. Testing Process 26](#_Toc394700381)

[7.1. System Tests 27](#_Toc394700382)

[7.2. Subsystem Tests 27](#_Toc394700383)

[7.3. Evaluation of Tests 28](#_Toc394700384)

[7.4. State the testing tools 28](#_Toc394700385)

[8. Glossary 28](#_Toc394700386)

[9. Appendices 29](#_Toc394700387)

[9.1. Appendix A: Project Schedule 29](#_Toc394700388)

[9.2. Appendix B: Use Cases 30](#_Toc394700389)

[9.3. Appendix C: User Interface Designs 37](#_Toc394700390)

[9.4. Appendix D: Class Diagrams 37](#_Toc394700391)

[9.5. Appendix E: Class Interfaces 40](#_Toc394700392)

[9.7. Appendix G: Diary of Meetings and Tasks 47](#_Toc394700393)

# Executive Summary

Manhunt, a more strategic version of tag, is a game that many individuals over the age of 20 may have fond memories of playing during their teenage years. However, the current generation of teenagers have higher expectations of entertainment, thus games like Manhunt have fallen from popularity in le of electronic forms of entertainment. The Manhunt mobile application is designed to bring technology and further engagement into the traditional game of Manhunt in the hopes of re-introducing Manhunt to today’s teenagers in a fun and exciting way.

Traditionally a game of manhunt begins with one predator, tagger, and the remainder of players being prey, hiders. While this concept may be fun for brief instances of running and tagging, it can become dull during drawn out games or long periods of hiding. To stimulate a more engaging game the Manhunt Android application will be developed so that all players in a game may have access to a centralized timekeeping system, a central communication system, scoring, and ways to keep individuals interacting in the game while hiding. Additionally, the application allows for predators greater ease when hunting prey. This development effort will follow the Unified Software Development Process to ensure that the Manhunt game is fully realized as intended and to utilize a systematic, organized process for delivering a usable, reliable, and exciting application.

This document includes a variety of chapters breaking down the various aspects of the project and requirements. The document first introduces the project, purpose of the system, and the project’s scope in detail. Next, an in depth explanation of the implemented system is defined. After that, the project plan is given which includes the project’s organization, hardware and software requirements, and the work breakdown. The succeeding chapter includes the requirements of the system, focusing on both the functional and nonfunctional requirements, use case diagrams, and a description of the requirements analysis performed. After is the software architecture. This contains an overview of the architecture, descriptions of each major subsystem, mapping of both the software and hardware, a description detailing persistent data, and an analysis of the system/s security and privacy. Next is a detailed design of the system. This is an in depth look at the subsystems, object interaction, and class design. Afterwards, the testing process is detailed which includes system and subsystems tests and evaluations as well as the testing tools used. Furthermore, the glossary, which contains a list of terms used in the document is presented. Lastly, the appendix contains a Gantt chart presenting the project schedule, a list of all use cases, user interface designs, class interfaces of the implemented subsystems, code from the test driver, and finally, a list of all meetings and tasks for creating the system.

# 1. Introduction

Manhunt is a demanding, tactical, and exhilarating variation of the game Tag that is extensively played throughout the United States. The Manhunt Android application is a multiplayer mobile extension to the outdated gameplay of Manhunt, adding new features and interactions to make the game more engaging to today’s youth.  In Phase Two of this software development process, the design phase, Manhunt has progressed from a list of requirements to an elegant software design.  With the help of the use cases laid out in Phase One, Phase Two saw development of UML models and the establishment of a well-designed and organized system.

## 1.1. Purpose of system

The traditional game of Manhunt does not utilize modern technology in its gameplay. Its dissociation with modern technology can give it the appearance of being outdated or uninteresting to the current generation of young people. Mobile devices are ubiquitous to the current generation of young people thus providing an accessible medium to introduce modern technological capabilities to Manhunt. Thus, a mobile application which augments the gameplay of traditional Manhunt would facilitate in making Manhunt more appealing to the youth of today.

Manhunt will provide the end user with a platform for the Manhunt game through a mobile application. The user will then be able to play alongside with other players using mobile devices. The system presents the players with features such as a map that contains other players’ locations as well as the ability to gain an advantage over other users by using in-game enhancements referred to as PowerUps.

## 1.2. Scope of system

Team3 shall deliver a functional and engaging mobile application of Manhunt that adheres to the specified functionality requested by the client.

Once the requirements are developed and agreed upon the system design and architecture will be developed around these requirements using systematic and verifiable methodologies to fully realize the client’s vision for the project. Similarly, the system will be developed with a systematic software development model and the final product will result in an Android mobile application; other environments may be supported in the future but will not be supported in this project. Lastly, the Team3 will ensure that the final deliverable is complete, concise, verifiable, and validated using various verification and validation models. The sum of these artifacts will be compiled into the final deliverable along with all required documentation. A complete listing of these deliverables can be found under Project Deliverables in this section.

The final implementation of the application developed will support a core set of features. These features include: a game match which up to 20 players may join, interactive game map that tracks users with beacons, the usage of PowerUps, a tagging feature, and objective designations (prey or predator) on players.

Team3 and this project will not be responsible for the distribution of the application, nor the Android App Market approval process. Additionally, the client will be responsible for purchasing, hosting, and maintaining the central server and these tasks will not be completed within this project’s scope.

## 1.3. Development Methodology

The Unified Software Development Process was chosen as the development methodology for this product given its heavy reliability on use cases. The fact that this methodology is use case driven allows for a stronger connection between the purpose of the application and the finalized product. The process consists of a use case model being specified by an analysis model, realized by a design model, distributed by a deployment model, implemented by an implementation model and finally verified by a test model. A project based on this methodology consists of the following cycles. This methodology consists of four main cycles. Inception culminates once the project sponsor has fully committed to the project. Then the process moves on to elaboration during which the basic architecture of the system is agreed upon and potential risks are identified. Construction follows and ends with a product that, although not fully finished, it is ready to be tested.  Finally a transition cycle introduces the system to its intended users.

## 1.4. Definitions, Acronyms, and Abbreviations

**Actors** - External entities that interact with the system.

**DD -** Design Document.

**IDE -** Integrated Development Environment, is a type of computer software that assists computer programmers in developing software.

**Milestone -** End point of a software process activity.

**Scenario -** A scenario is an instance of a use case describing a concrete set of actions.

**Schedule -** Mapping of tasks onto time.

**SDK -** Software Development Kit.

**SRD -** Software Requirements Document.

**System -** An organized set of communicating parts designed for a specific purpose or a purposeful collection of components that work together to achieve some objectives.

**UML -** Unified modeling Language.

**Use Cases -** Use Cases are general sequences of events that describe all possible actions between actor and the system for a given piece of functionality.

## 1.5. Overview of Document

This Final Deliverable Document is divided into nine main sections which individually contain a number of subsections. Following the introduction, section two, Current System, explains in detail known limitations and problems with the current system. The Project Plan, section three, provides further account of how the project has proceeded, including Gantt charts, hardware requirements, and role specification. Requirements of System follows as section four. Throughout this section, the proposed system is properly introduced along with all of the Functional and Nonfunctional Requirements, use case diagrams for the implemented use cases as well as the Requirements Analysis. Section five, Software Architecture, introduces the software architecture and relates it to the existing use cases that are implemented. This section includes an overview of the subsystem decomposition, hardware and software mapping, persistent data management as well as measures of security and privacy. Section six, Detailed Design, introduces the object design chapter by providing a minimal class diagram for an overview of the design, object interactions by showing sequence diagrams and finally the purpose of each class is explained in further detail in the subsection Detailed Class Design. The Testing process, section seven, presents system and subsystem tests along with their corresponding evaluations. Details regarding the testing tools used during the testing process are presented at the end of section seven. Glossary follows as section eight by presenting proper definitions of domain specific terms used in the document. Section nine, Appendix, contains Appendices A which contains project schedule. It also contains Appendix B which provides all use cases with non-functional requirements, Appendix C which contains the user interface design, Appendix D which presents every detail class diagram, Appendix E which provides the class interfaces, Appendix F which contains the documented code for the test framework used for testing, and finally Appendix G which lists all of the meetings and tasks in a diary. Included at the end of this document, a user guide can be found where a brief description of the application is provided along with the hardware and software requirements. This user guide also includes instructions on how to run the application.

# **2. Current System**

The traditional game of Manhunt consists of one person, the predator, who is chosen to tag everyone else, the prey, who run and hide from the predator. When a prey is tagged by the predator, the tagged prey becomes an additional predator and then assist in tagging the rest of the prey. The traditional game’s goal is to avoid being tagged until everyone else is tagged or the time limit for the game is reached.

Depending on the environment, predators may have difficulty locating the prey. There are no current ways of changing gameplay after game has started given that players abide by the rules specified before the Manhunt game starts. Keeping track of time in the game is difficult since every player has to keep their own time and knowing when the game has ended is not trivial. There is no current way of keeping an elaborate score system. In summary, there is not centralized way to manage each game match and provide a further level of interaction among the players.

# 3. Project Plan

This chapter provides a detailed account of how the project will proceed, including Gantt charts, hardware requirements, and role specification. The Project Organization specifies the role assignments for each project member for every phase of the project. Hardware and Software Requirements specify the necessary hardware and software to implement and operate the Manhunt mobile application as well as the development environment. Within the Work Breakdown each sub-system is broken down into smaller, more manageable project increment allowing for the parallel development of many of the projects sub-systems. Finally, the work breakdown utilizes these small increments to schedule the deliverables, milestones, and tasks along with their dependencies.

## 3.1 Project Organization

The following table displays how the roles are broken down for the three deliverables of the project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Participant** | **Software Requirements Document** | **Design Document** | **System Document** |
| ***Ariel Diaz***  *(786) 361 3525*  *adiaz141@fiu.edu* | Cross-Functional, Meeting Facilitator | Meeting Facilitator, Developer, Cross-Functional | Meeting Facilitator, Developer, Tester |
| ***Mathew Santiago***  *(863) 214 4073*  *msant080@fiu.edu* | Minute Taker & Cross-Functional | Developer & Minute Taker | Minute Taker, Developer, Tester |
| ***Musa Ahmed***  (954) 993 0768  mahme012@fiu.edu | Leader,  Time Keeper | Leader, Developer, Time Keeper | System Architect, Developer, Tester, Time Keeper |
| ***Justin Phillips***  (954) 649-0827  jphil075@fiu.edu | Cross-Functional | Cross-Functional, System Architect | Leader, Developer, Tester |

## 3.2 Hardware and Software Requirements

A central server will be required to act as a communication hub and data processing service to support the game matches. The central server will require commercial bandwidth over the internet and have enough resources to maintain hundreds of concurrent game matches. The server will maintain a database system to track all necessary data.

An Android based mobile device is required to play Manhunt, the device must have a data plan so that the device can periodically query the central server. The device will require a recent version of the Android operating system in-order to ensure the system works as intended.

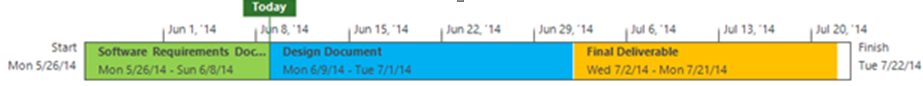
The development stage will require a competent IDE for Android development as well as an Android device with GPS capabilities and data connection. A development environment mimicking that of the central server must be available during the development stage.

|  |  |
| --- | --- |
| **Application Mobile Device Requirements** |  |
| **Hardware** | * An android mobile device * Single-core 1.0 GHz * 1024 MB of RAM, and 5MB free of flash memory * GPS * Cellular data connection |
| **Software** | Android OS, v4.0 (Ice Cream Sandwich) or higher. |
| **Server Requirements** |  |
| **Hardware** | * Intel Xeon E3 1225 v3 64-bit Quad-core or equivalent * 24 GB DDR3 1600 Mhz Unbuffered ECC Memory * 2x 1 TB 7200 rpm Hard Disk Drive * Reliable Gigabit LAN |
| **Software** | * MySQL Server 5.4 * PHP 5.3 and Apache 2.2 |

|  |  |
| --- | --- |
| **Development Workstation Requirements** |  |
| **Hardware** | * Multi-core 64-bit processor * Minimum of 8 GB RAM * Minimum of 100 GB free space * USB 2.0 port for debugging hardware |
| **Software** | * Eclipse IDE with Android SDK Tools. * The Google Maps Geolocation API. * MySQL Server 5.4 * PHP 5.3 and Apache 2.2 |
| **Android Development Requirements** |  |
| **Hardware** | * An android mobile device * Single-core 1.0 GHz * 1024 MB of RAM, and 5MB free of flash memory * GPS * Cellular data connection |
| **Software** | Android OS, v4.0 (Ice Cream Sandwich) or higher |

## 3.3 Work Breakdown

The following a basic project timeline, milestones, and deliverables. Refer to the entire project schedule in Appendix A.



|  |  |
| --- | --- |
| **Project Milestones** | **Project Deliverables** |
| * Introduction Section – 06/06/14 * Project Plan Section – 05/30/14 * Proposed System Section 06/05/14 * Appendix Section – 06/09/14 * Software Requirements Document – 06/09/14 * Design Document – 08/01/14 * Final Deliverable CD – 07/21/14 | * Deliverable 1 - 6/09/14   + Software Requirements Document * Deliverable 2 – 07/02/14   + Design Document   + UML Diagrams * Final Deliverable – 08/01/14   + Software Requirements   + Design   + Implementation   + Test Cases   + User’s Guide * CD with all project material |

# 4. Requirements of System

The Manhunt application gives the traditional Manhunt game an entirely new level interactivity with a digital near real-time map that immerses and engages players. Players stay connected to each other using an in-game chat system that may help organize a strategy to achieve victory or taunt enemies. Scores will be maintained within game matches and players will have the ability to use PowerUps which are purchasable items that enhance the game’s dynamics. The PowerUp “Cloak” allows for the prey’s current location to be hidden for the duration of a minute. At the end of each match the players are presented with a scoreboard showing each player’s achievements during the last match.

## **4.1. Functional and Nonfunctional**

**Implemented Use Cases:**

**UsePowerUp:** The system shall allow the player to dynamically alter the gameplay of every other player using a PowerUp. The system shall present an intuitive user interface to allow the player to select the Power Ups. The system shall allow the player to select a PowerUp within 3 seconds, to not disrupt gameplay. The system shall have a mean time of failure for this use case of 1% for every 24 hours of operation. The system shall invoke the PowerUp for the player within a second of the player tapping on the PowerUp. The system shall be able to handle 20 requests every two minutes.

**ChainPowerUp:** The system shall prevent a player from exploiting PowerUps by using PowerUps back to back. The system does this by applying a cooldown to the usage of PowerUps after one has been invoked. After the cooldown time expires then the player will be able to use PowerUps again. The system shall automatically take care of implementing a cooldown on the use of PowerUps. The system shall have a mean time of failure for this use case of 1 % for every 24 hours of operation given that the gameplay relies on this security use case. The system shall invoke the cooldown within a second after the PowerUp is invoked. The system shall be able to handle 20 requests every second.

**TagPlayer:** The system shall present a tag button which allows a prey to become tagged by a predator when a predator players comes within 5 meters of a prey player. The system shall be able to reliably detect the proximity between the two players. The button shall be displayed within 10 seconds of the two players coming into proximity. The system shall have a mean time of failure for this use case of 5% for every 24 hours of operation. The player shall be able to display this button to at least 2 players at the same time.

**GetTagged:** The system shall transition a tagged prey player into a predator player. After the player presses the button that the system presents the player with to allow getting tagged, the system shall transform the prey within 5 seconds of the button being pressed. The system shall have a mean time of failure of 5% or less for every 24 hours of operation. The system shall reliably detect the proximity between the prey and predator players.

**PlayMatch**: The system shall allow a player to play a Manhunt match. The system shall present the player with an intuitive user interface for the game match. The system shall have a mean time of failure for this use case of 5% for every 24 hours of operation. The system shall refresh the Map Screen automatically every minute. The system shall be able to sustain 20 players at a time.

**BuyPowerUp:** The system shall allow a player to purchase a PowerUp in order to then invoke a PowerUp during gameplay. The system shall provide an intuitive interface to allow the player to buy PowerUps by just pressing a button on the Map Screen. The system shall have a mean time of failure for this use case of 5% or less for every 24 hours of operation. The system shall reward the player with the PowerUp within 5 seconds of the player purchasing the PowerUp.

**SecondDeviceCheat:** The system shall block the map screen with a button to confirm getting tagged as soon as predator player comes within 5 meters of prey player. The system shall reliably identify the proximity between a predator and a prey in order to accurately block the prey’s Map Screen. The system shall have a mean time of failure for this use case of 1% for every 24 hours of operation given that the gameplay relies on this security use case. The system shall block the prey’s Map Screen within 5 seconds of the two devices coming in proximity.

**Unimplemented Use Cases:**

**CreateGameInstance:** The system shall allow a player to create a new game instance. The system shall allow up to 20 players to join. The system shall have a time interval between two and twenty minutes.

**StartGameMatch:** The system shall allow a created game instance to become a started game match by allowing the created to tap the ready button when all players in the game instance have tapped the ready button. The system shall only allow a game match to start if there are at least two players in the game instance. The system shallow allow at most twenty players to be allowed into a game instance.

**JoinGameInstance:** The system shall allow a player to tap the join game button to be placed into an already started game instance. The system shallow only allow up to a maximum of twenty players into any given instance.

**BuyCoins:** The system shallow allow a player to buy coins in allotments of 5, 10, and 20 coins using the Google Wallet payment option to check out. The system shall determine of a payment was processed successfully. The system shall disperse coins within 5 seconds of a payment being received.

**CheckScore:** The system shall allow a player to view his or her score as well as the scores of the top three predators and the current number of prey left in a game match. The system shall provide basic usage for the check score system in the tutorial. The system shall accept a 15% failure rate with in any 24 hour period of operation. The system may accept a downtime of 1 hour per 24 hours of performance. The system shall be updated each time a player’s score increases.

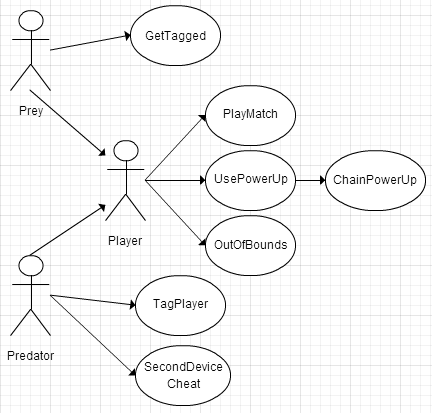
**InGameChat:** The system shall allow players to type into the chat screen and display messages to and from other players. The system shall provide basic usage in the tutorial for this feature. The system shall maintain a mean time for failure of 15% for every 24 hours of operation. The system shall accept up to one hour of downtime per 24 hours of operation. The system shall be able to handle up to 20 players chatting concurrently.

**EndGameInstance:** The system shall allow a game match to end when there are no more prey players present or if there are less than two players in the game match. The system shall reward the appropriate number of coins to players in the match within 5 seconds. The system shall end the match within 10 seconds of detecting that a game match has finished.

**LeaveRejoin:** The system shall appropriately deal with players who attempt to circumvent the tagging system by leaving the game when in danger and rejoining at a later, safer time. The system shall detect a player attempting this as soon as said player attempts to rejoin the match. The system shall erase the player’s score within 5 seconds of rejoining the match.

**OutOfBounds:** The system shall detect if a player goes out of the allowed radius boundary for the game match and the map beacon color shall change to indicate that the player is out of bounds. The system shall reliably change the beacon color for the out of bounds player within a minute of the player going outside the established boundaries. The system shall have a mean time of failure for this use case of 5% or less for every 24 hours of operation given that the gameplay relies on the implementation of this security use case. The system shall be able to identify if a player is out of bounds for up to 20 players.

## **4.2. Use case diagram**



**Figure 4.1**

## 4.3. Requirements Analysis

There are four use case models present in Appendix B. These four models represent the four distinct subsystems of the project. Figure B-1 is the main Manhunt game subsystem. This model was developed with a simple menu driven user interaction in mind. Figure B-2 is the Game Match subsystem. This model consists of the all the features that make up the actual game play of the game except tagging and possible forms of cheating. Figure B-3 is the Tagging subsystem. The Tagging subsystem was purposefully separated from the Game Match subsystem due its modular nature and specific functionality. Finally Figure B-4 is the Device Communication subsystem. Which is designed to consist of all possible problems that can occur from the device.

# **5. Software Architecture**

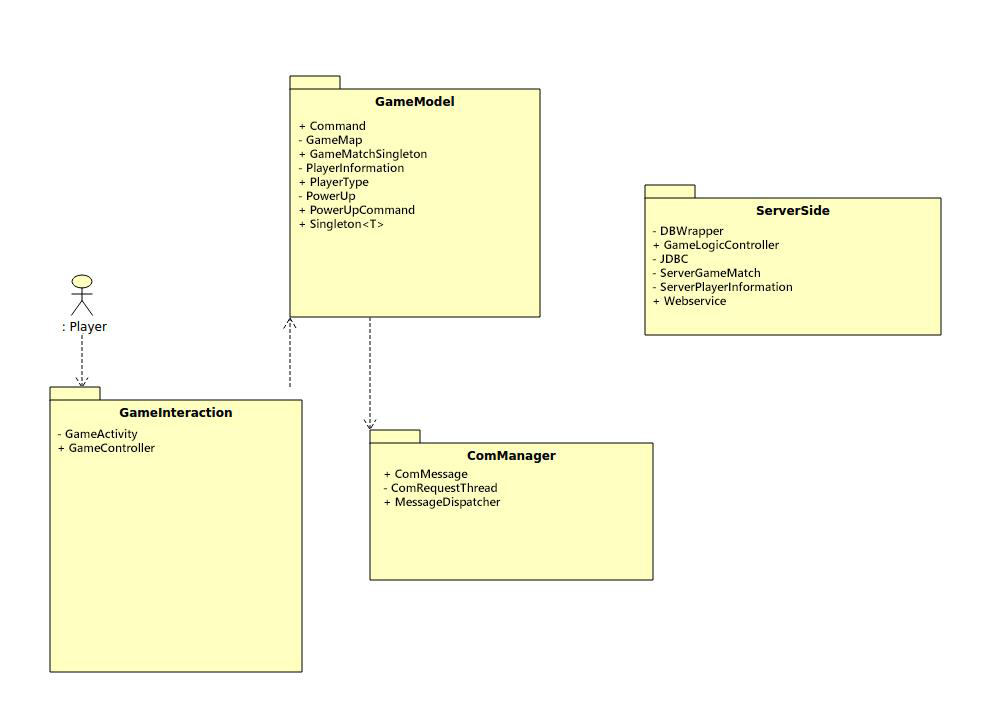
Detailing the software architecture that the system will follow is a crucial step in the development of the project. Subsystem decomposition in terms of responsibilities, dependencies and mapping to hardware will be described by the software architecture that is to be presented in subsections 5.2 and 5.3. Persistent data management and security resources are specified in subsections 5.4 and 5.5 of this chapter.

## 5.1. Overview

The Game Interaction subsystem provides user interface which users interact with and the game’s logic. This subsystem includes event handlers and the main game loop. The Game Model subsystem holds all the models which are the realization of the game, this includes the players in the game, the game match, and a map.

The Communication subsystem manages the transfer of data between the Game Interaction, Game Model subsystems and the Server Side subsystem. This is handled using an outgoing queue of messages, once sent a callback function provided on invocation handles the response data.

The Server Side subsystem takes requests from active clients which send data to be processed. The Server Side subsystem then processes all of the data collected from the clients and responds with the results.

The Client/Server and Model View Controller are the architectural patterns chosen to implement the system. Client/Server was selected as the primary pattern because at the greatest scale, the system will have many clients sending and receiving information from a Server that will perform most of the operations. In addition, the clients should have no knowledge on how the logic is implemented in the server, and the Client/Server architectural pattern allows for this abstraction. As a secondary architectural pattern, the system implements Model View Controller (MVC) because the system relies heavily on a user interface, and the MVC structure allows for the logic and the objects to be separated from the actual user interface. This architectural pattern was also chosen because Android is already setup in a similar way given that Activities resemble controllers, Layouts resemble Views and our classes for the game resembled Models.

## 5.2. Subsystem Decomposition

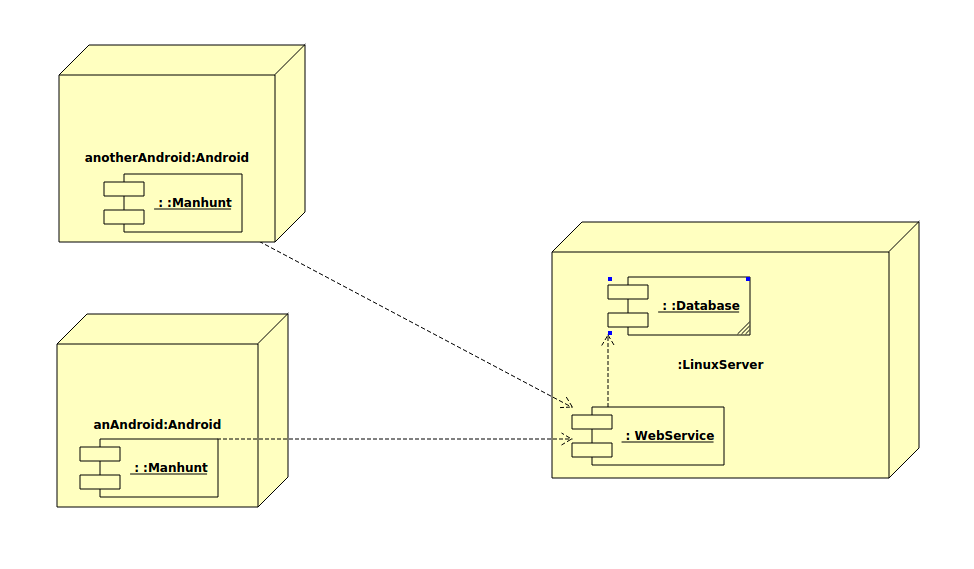
The Manhunt system is broken down into two major subsystems: the client android application subsystem and the server subsystem. The client subsystem contains mostly passes information that deals with user interaction and passes commands to the server so that other clients in the same game instance can interact and store persistent information onto the server. The server subsystem contains most of the logic for the system. This logic is usually activated by the client activity and is received from the client subsystem.

The Client subsystem contains the following classes: GameModel and GameInteraction shown in Figures 6.2 and 6.3. The GameModel class incorporates a majority of the data that the player will interact with. This includes player information, the game match singleton, the game map, and power ups. This information is often obtained from the server’s persistent database. The GameInteraction class allows the player to interact with other players. While most of the logic lies on the server, the client is often the invoker. Commands to the server include showTagButton, tapTagButton, startGameMatch, tapPowerUpButton, displayPowerUpList, and buyPowerUp.

The Server subsystem is decomposed into the following classes: DbWrapper, JDBC, GameLogicController, Webservice, Player, and GameMatch. The DbWrapper is mostly involved with player interaction with a game match. It can add and remove, ban and unban, and update players as well as create and remove a game match. The GameLogicController class contains a list of all game matches, and controls the maximum number of players per match, and maximum game instances. Its logic allows it to update player information, remove a player from a match, and create or remove a game instance. The Player class contains variables for a player’s email address, current latitude and longitude position, unique identifier, and status. It also has getters and setters for each of these variables. The GameMatch contains a list of players in a given match, the start time of the match, the match’s unique identifier, and whether the match is active or not. This class can add and remove players from a match, and return a list of every player, a boolean of whether the game is active, and the match’s identifier. Finally the Webservice class allows the client to interact with each of these classes. It contains a listener for commands, and a list of methods available.

## 5.3. Hardware and Software Mapping

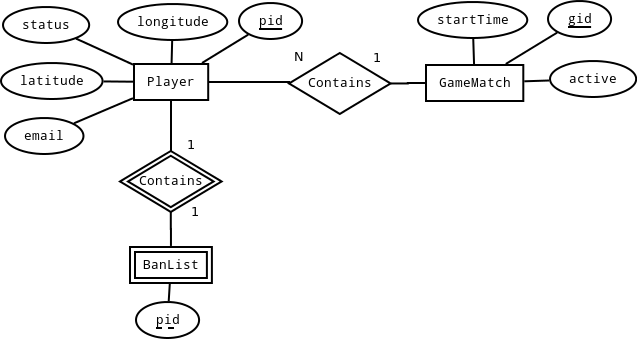
As shown in Figure 5.1, the four subsystems that compose the Manhunt application are mapped to hardware as follows. The GameInteraction, GameModel and Communication subsystems compose the client side of the application which is mapped to an Android mobile device. A centralized server and database will be mapped to our ServerSide subsystem. This is a many-to-one mapping, given that several client subsystems will be able to connect to a single server subsystem. The system requires KitKat 4.4.x as the Android version for the Android device mapped to the client side of the application. However, there are no requirements tying the operating system of the server that will be mapped to the server side of the application given that it’s running a web service and dependencies of this kind do not exist in this case. However the system requires a PostgreSQL 9.1 database to be mapped to the server side of the application.



**Figure 5.1**

## 5.4. Persistent Data Management

There are two categories of persistent data. Firstly the player authentication data with is held by Google and secondly all the game data held in the ServerSide subsystem. The ServerSide subsystem data is mainly information about each player, information about each game, and a list of players who are banned. The ServerSide subsystem data is stored in a Postgres relational database as a collection of records in tables. For a general overview of the table structure see Figure 2 for an entity-relationship diagram.



**Figure 5.2**

## 5.5. Security/Privacy

Security is a key component for keeping a game like Manhunt fun and enjoyable for all players. In order to maintain a secure system data, especially persistent data, must be kept hidden from unauthorized systems. Most of the data is kept safely stored away from the users, on a server.

As for locally stored persistent data, the player’s power ups will be stored on their device. To maintain a healthy gameplay and reduce risk for abuse, this data will be encrypted.

Another key security component is user authentication. As a result of the system being Android based, Google takes control of user authentication. As long as the system can obtain the required data from the Google Servers, the user will have access to his/her Manhunt account. Any user who fails to provide required data, or provides duplicate authentication information will be added to the user ban list. The user ban list will be globally accessed. Any users on this list will not be able to log into the system and participate in any games.

# 6. Detailed Design

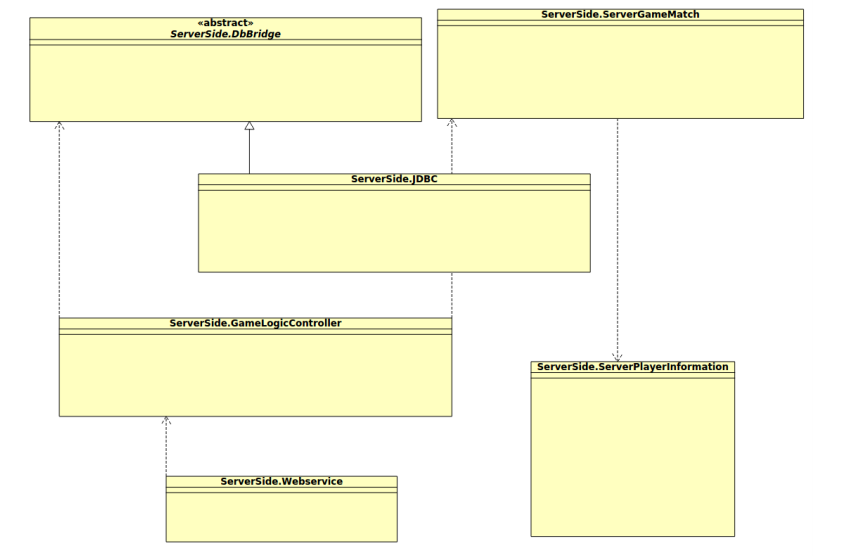
Object design allows us to plan our logic in order to attain better flow within the program. The system decomposition into objects allows us to explain how these objects interact with one another throughout the system. This allows us to build a robust overall structure while meeting the initial purpose of the system.

Subsection 6.1 shows an overview of the system in terms of subsystems and their respective classes by providing a minimal class diagram. Subsection 6.2 joins the subsystems with their corresponding use cases in the form of sequence diagrams, which allows us to visualize interactions between objects in the system. Finally, subsection 6.3 explains the purpose of each class and appropriate references to the class diagram and code section are provided.

## 6.1. Overview

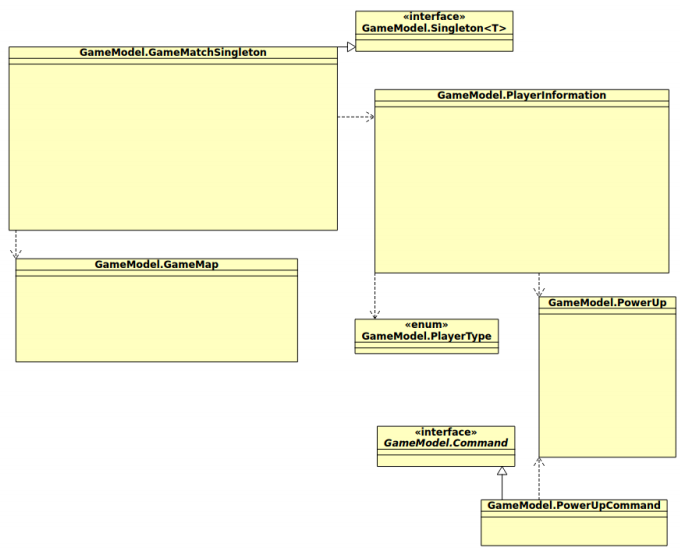
The system is broken into two major subsystems, the client and the server. The client consist of

GameInteraction, GameModel, and ComManager packages; while the server simply implements ServerSide package. The three design patterns used in the minimal class diagrams are Singleton, command, and Adapter (Wrapper).



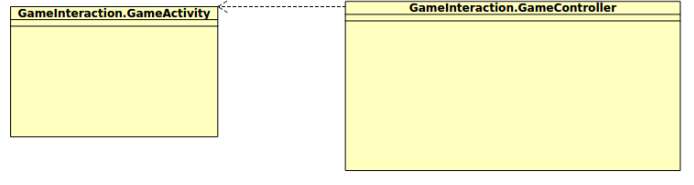
**Figure 6.1**

* **ServerSide**This package implements the server component. Allowing for a web service to listen for request, the game logic controller to handle the request, and back to the web service for returning the response to the client.
  + **dBWrapper**This class implements the adapter design pattern as an interface. It allows for the persistent storage operations to be implemented abstractly, essentially decoupling the storage mechanism from the server.
  + **JDBC**The implementation for JDBC database connection via the dbWrapper interface. Contains the SQL for all the calls.
  + **GameMatch**This is the server’s version of game match, this calls will be handled as a collection of matches with data the server needs for each match.
  + **GameLogicController**The main controller and logic handler for server, it handles all of the request via the webservice.
  + **Player**Player object for the server, tracking player data the server needs. This class will typically be handled as a collection within each game match.
  + **WebWervice**The web service provides a connection interface for the server to communicate with clients through. This is a platform independent protocol which allows for remote invocation of methods.



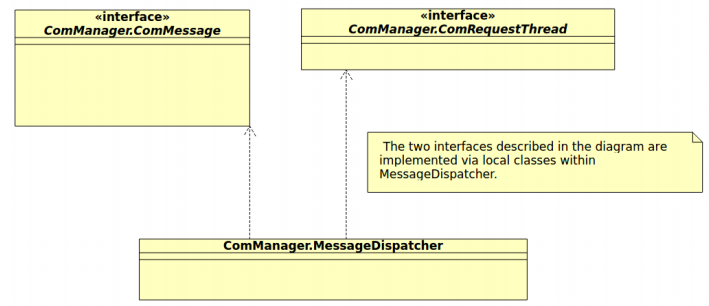
**Figure 6.2**

* **GameModel**The GameModel acts as the model for the client. It is the realization of the entities that exist on the server and it updates or receives data syncing its classes via the ComManager package.
  + **GameMap**This class implements Google’s Map API and represents the state of the map. Its responsibilities are to update the map entity every time new data is received, determine who is out of bounds, and which type of Google marker will be used depending on prey or predator.
  + **Player**Player information stores all the information about a player including location, prey or predator, and etc...
  + **GameMatchSingleton**The game match is a singleton design pattern which contains the match state an information. There can be only one game match per client so the singleton pattern ensures that the class finds any instantiated game match and returns it, else creates an instance.
  + **PowerUp**This class represents powerups, the powerup name, icon, and state information.
  + **PowerUpCommand**This interface class implements the command pattern on powerups deferring execution until it can be delegated to another thread as not to block the UI. Also, provides a common method of execution for all powerups.
  + **Command**The command pattern interface, it simple provides a method called “execute” for the above calls to implement.



**Figure 6.3**

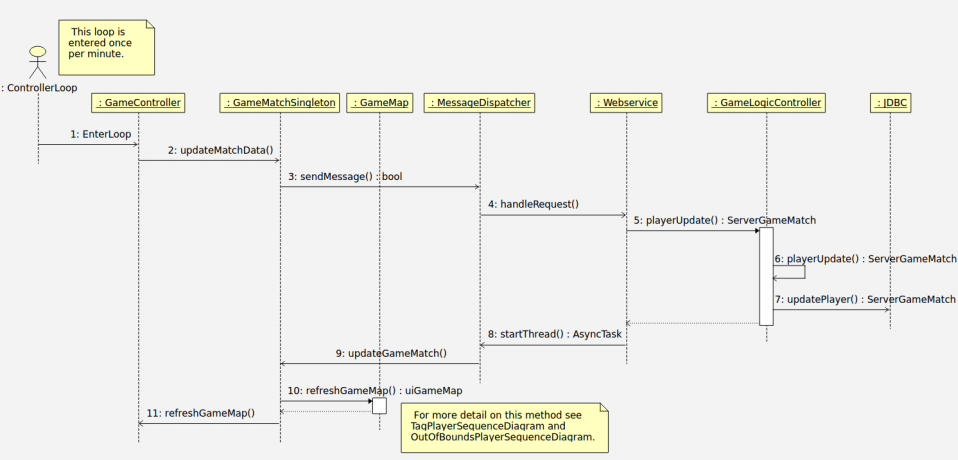
* **GameInteraction**This package works as both the view and controller within the client.
  + **GameActivity**  
    An Android Activity class which interacts with the UI, responsible for visual UI changes, screen transitions, and UI event handlers. This is the view.
  + **GameController**  
    The main logic controller class, it drives the entire client operation. Its responsibilities are the main game loop, passing view interactions to the model and model updates to the view.

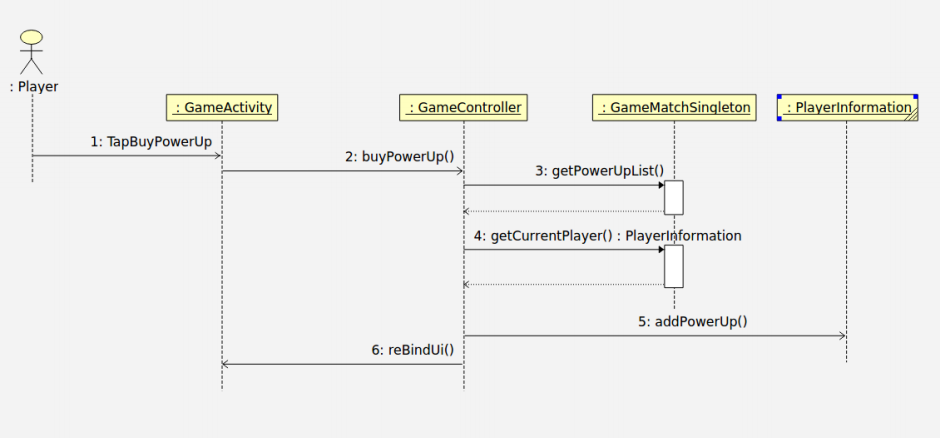


**Figure 6.4**

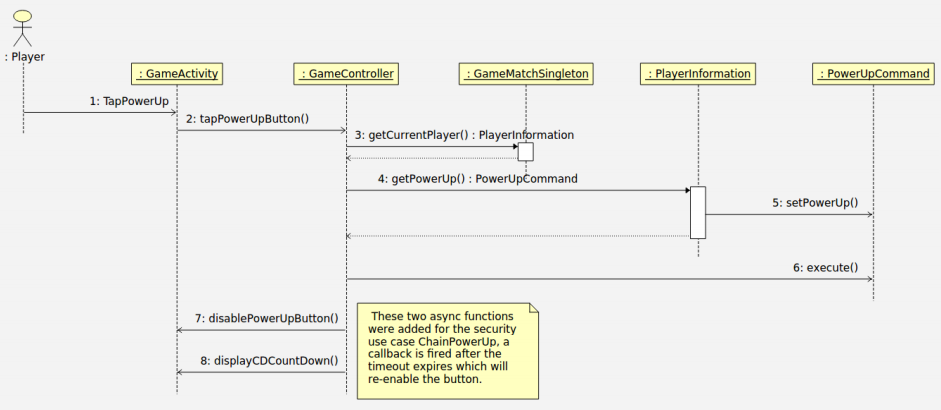
* **ComManager**This package handles all messages to and from the server on the client. It maintains a queue for outgoing messages and handles each response to a request it makes with a lambda expression provided by whomever is sending the message. This class is built to work with a web service, though there is an interface for message threads to abstract the coupling to other communication protocols.
  + **ComDispatcher**The dispatcher maintains the message queue, spawns message threads, and maintains the connection info.
  + **ComMessage**  
    This class is an interface representing messages, these are to be implemented as local class within the GameMatchSingleton for the various types of messages which will be required. It will need a method string, a json data string, and a lambda expression for callback.
  + **ComRequestThread**  
    This class implements AsyncTask, Androids threaded task class, to make a call to the web service and then handle the web service's response with the provided lambda expression.

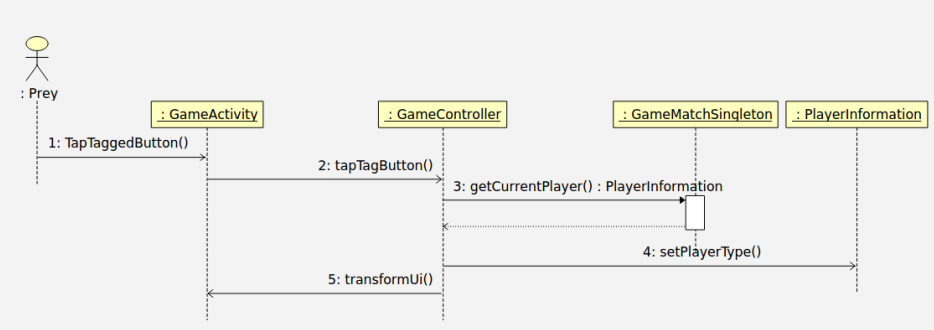
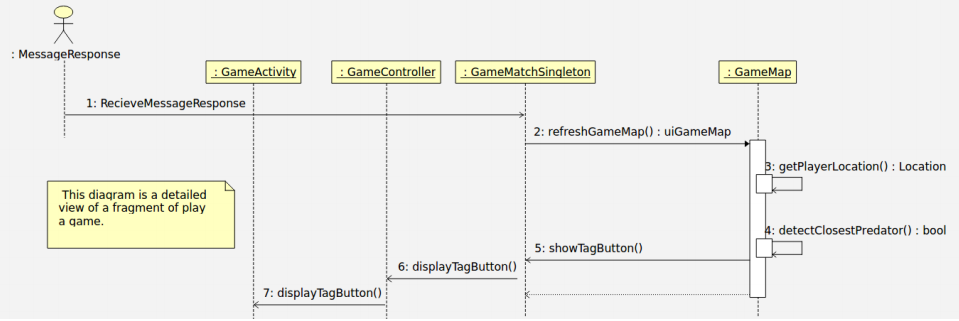
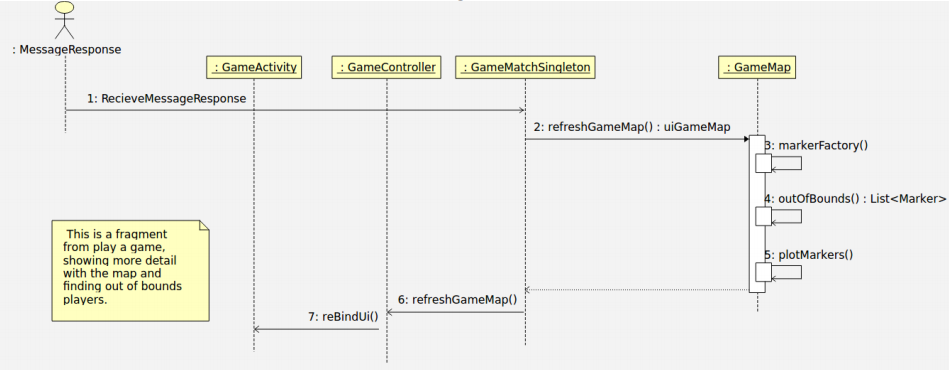
## 6.2. Object Interaction

**Figure 6.5**



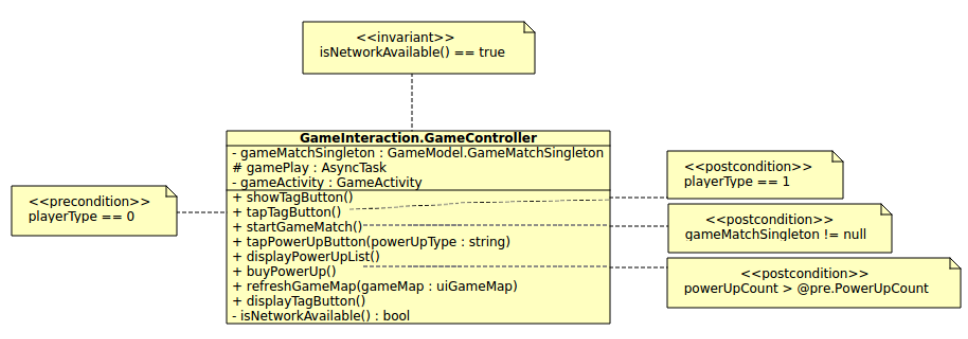
**Figure 6.6**



**Figure 6.7** **Figure 6.8** **Figure 6.9** **Figure 6.10**

## 6.3. Detailed Class Design

– Explain the purpose of each class and reference appropriate class diagram in Appendix D and the code in Appendix E.  Indicate the patterns used in designing the system.



**Figure 6.11**

# 7. Testing Process

Although it is one of the last steps of the software development process, testing of the system is easily considered the most crucial step as well. This chapter will go in depth with the testing that was performed on the Manhunt system. In Section 7.1, the tests involved with the Manhunt system as a whole are performed while section 7.2 details the subsystems tests performed. In section 7.3, the evaluation of the testing performed on the system is analyzed. Section 7.4 contains the tools used to test the system and subsystems.

## 7.1. System Tests

|  |  |
| --- | --- |
| **Test ID:** | UsePowerUp1 – Sunny Day |
| **Purpose:** | To test that the powerup |
| **Test Setup:** | * The Manhunt application must be loaded * User must be in a game match |
| **Input:** | User uses a power up, Cloak |
| **Expected Output:** | User’s map location no longer shows up on the screen |

|  |  |
| --- | --- |
| **Test ID:** | TagPlayer – Sunny Day |
| **Purpose:** | Test powerup cooldown is in place |
| **Test Setup:** | * The Manhunt application must be loaded * User must be in a game match * Player has used a powerup |
| **Input:** | Tap on a power up |
| **Expected Output:** | Power up not used |

|  |  |
| --- | --- |
| **Test ID:** | GetTagged – Sunny Day |
| **Purpose:** | Test that tagged button works |
| **Test Setup:** | * The Manhunt application must be loaded * User must be in a game match * Prey must be close to prey |
| **Input:** | Player taps on get tagged button |
| **Expected Output:** | Player becomes a predator |

|  |  |
| --- | --- |
| **Test ID:** | PlayMatch – Sunny Day |
| **Purpose:** | Tests that a player can play a match |
| **Test Setup:** | * The Manhunt application must be loaded * User must be in a game match |
| **Input:** | Player taps on start game button |
| **Expected Output:** | Player is added to game match |

|  |  |
| --- | --- |
| **Test ID:** | SecondDeviceCheat – Sunny Day |
| **Purpose:** | Tests that a player cannot use a second device to cheat |
| **Test Setup:** | * The Manhunt application must be loaded * User must be in a game match |
| **Input:** | Player connects to the game match on another phone nearby |
| **Expected Output:** | Tag button covers screen of the prey phone |

## 7.2. Subsystem Tests

|  |  |
| --- | --- |
| **Test ID:** | PowerUp |
| **Purpose:** | To test that the powerup |
| **Test Setup:** | * The Manhunt application must be loaded * User must be in a game match |
| **Input:** | User uses a power up, Cloak |
| **Expected Output:** | User’s map location no longer shows up on the screen |

## 7.3. Evaluation of Tests

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Results** | **Actual Results** |
| UsePowerUp | Pass | Game marker removed from map |
| TagPlayer | Pass | Prey screen covered by get tagged button |
| GetTagged | Pass | Prey player changed to predator player |
| PlayMatch | Pass | Player has been added to match |
| SecondDeviceCheat | Pass | Prey screen covered by get tagged button |
| ChainPowerUp | Pass | Use Power up button disabled |

## 7.4. State the testing tools

The following is a list of the development tools used for the testing of the system:

|  |  |
| --- | --- |
| **Development Tool** | **Use** |
| **EclEmma** | Looks at code coverage by monitoring lines. Gives a ratio of line coverage. This help |
| **Gradle** | Project configuration and dependency manage tool |
| **Maven** | Project configuration and dependency manage tool |

# 8. Glossary

**Actors** - Prey, Predator, Player

**Android device** - Smartphone that uses Android as its operating system.

**Application** - The Manhunt system.

**Beacon** - representation of player on the map.

**Boundary Radius**- area where game is supposed to take place.

* **Cloak** - PowerUp that hides player’s visibility.
* **Decoy** - Spawns new instance of the player on the map for a limited duration of time.

**Coins -** In-game currency that allows you to buy Power-Ups.

**Cool-down** - Period of wait time before a **Power-Up** can be used after a prior **Power-Up** was used.

**Game** - Manhunt Application

**Game Full -** When a game has 20 people in the match at one time.

**Game Instance** - Lobby with players and score statistics

**Game Match** - Current game being played

**Game Session** - Duration Manhunt application is open

**Hunted** - **Prey** player(s) that hiding/running from the **Predator**.

**Hunting** - Job of the **Predator** to find and **Tag** the **Prey** player(s).

**Match** - playable part of game?

**Match Duration** - Time limit for a match specified by the amount of players

**Player** - Actor who is playing Manhunt before game match begins.

**Power-Up** - Consumable game item containing a use cool-down timer and finite quantity.

**PowerUps chaining** - using PowerUps back to back.

**Predator** - Player who is hunting.

**Prey** - Player being hunted.

**Screens**

* **Lobby Screen** - Screen where all players wait until match is started.
* **Main Menu Screen** - Screen the player is presented with by default when the game starts.
* **Map Screen** - Screen where **preys’** location is displayed.
* **Purchase Coins Screen** - Screen where **coins** are purchased.
* **Power-Ups Catalog Screen** - List of purchasable **Power-Ups**, a prompt is displayed on purchase.
* **Post-Game Screen** - Screen where game results are displayed.
* **Splash Screen** - Screen that displays the game logo while the game is initially started.

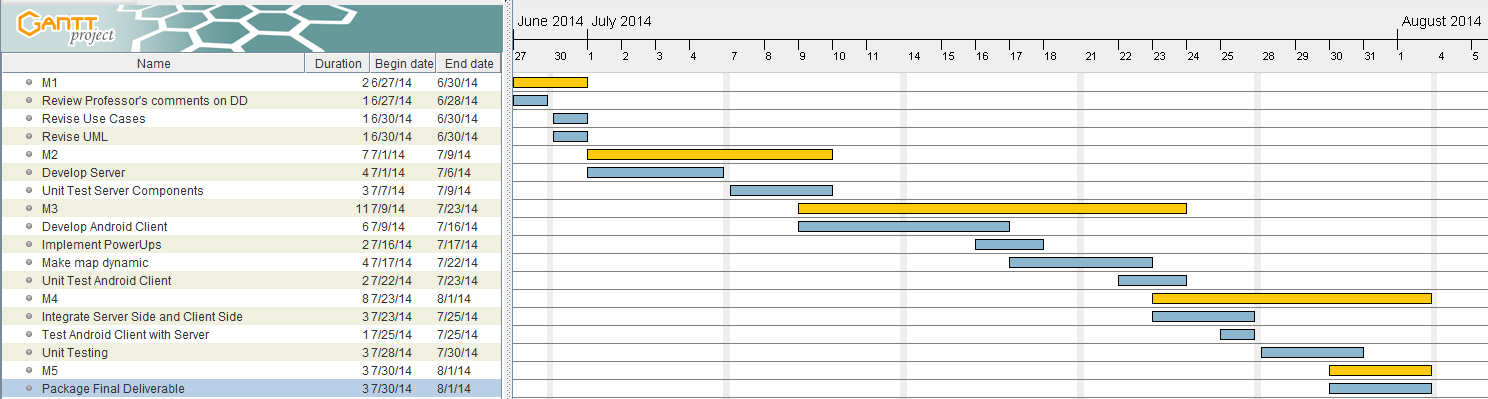
**Session -** Duration that the app is open

**Tag** - The system determines when a **Predator** player is close to a **Prey** player and sends a request to the **Prey** confirming whether or they have been “hunted”. After receiving confirmation the system transfers the **Prey** player to a **Predator** player.

**Tagging** - Performing a Tag.

# 9. Appendices

## 9.1. Appendix A: Project Schedule



## 9.2. Appendix B: Use Cases

|  |  |
| --- | --- |
| **Play Match** | |
| **Use Case ID** | Team3\_PlayMatch |
| **Use Case Level** |  |
| **Scenario** | The Player plays a Game Match. |
| **Actor** | Player |
| **Pre-conditions** | 1. Player has the Game installed and running on their mobile device. 2. Player has started a Game Match. 3. Player has strong GPS signal. 4. Player has a strong Data connection. |
| **Description** | 1. Use case begins when the Player transitions into the match from the Lobby Screen. 2. The system shall display a connection message. 3. The system shall show the Player the Map Screen along with a slide out menu on the left side. 4. The system shall retrieve and populate the Map Screen with the locations of the other Players based on whether the Player is either a Predator or Prey Player. 5. The system shall continue to update and refresh the Map Screen repeatedly at a set time interval. 6. Use case ends when the Game Match ends and the Post-Game Screen is displayed. |
| **Relevant Requirements** | The Player is not attempting to enter the match from a commercial airplane. |
| **Post-conditions** | 1. The Game Match was successfully played. 2. The Post-Game Screen is displayed. |
| **Alternative Courses of Actions** | 1. In step D.2 if the system does not have a strong GPS signal or Data connection then it will display an error message. 2. In step D.5 if the system losses GPS signal or Data connection for a specific duration it will display an error message. |
| **Exceptions** | 1. The Player has reached the Map Screen, however the system is not displaying any information on the Map Screen. 2. The Player’s Map Screen does not update information. 3. The Player’s Map Screen shows an incorrect location. 4. The system is displaying the wrong data on Map Screen for either Predator or Prey Players |
| **Related Use Case(s)** | 1. Team3\_JoinAMatch 2. Team3\_InGameChat 3. Team3\_CheckScore 4. Team3\_UsePowerUp 5. Team3\_TagPlayer |
| **Decision Support** | |
| **Frequency** | A Player will play an average of five matches per Game Instance. |
| **Criticality** | High. The system is not functioning properly if a Player cannot play a Game Match. |
| **Risk** | High. Implementing this use case requires the use of the Google Maps API with the integration of GPS services and a Data connection. |
| **Constraints** | Usability  Intuitive use by touch.  Basic usage should be in the tutorial.  Reliability  Mean time to Failure – 15% failures for every 24 hours of operation is within acceptable limits.  Availability – Downtime of one hour per 24 hours is acceptable.  Performance  The Map Screen should refresh automatically every minute.  The Map Screen should be able to sustain 20 Players at a time.  Supportability  The Map Screen should properly display on all types of devices and screen sizes.  Predator Specific Constraints   1. Predator cannot use PowerUps that involve manipulating the Predator’s location. 2. Predator’s Map Screen refreshes Prey locations less often. 3. Predator’s Map Screen beacons are less accurate than those on the Prey’s Map Screen. |
| **Modification History** | |
| **Owner** | Musa V. Ahmed |
| **Initiation Date** | 06/01/14 |
| **Date Last Modified** | 06/02/14 |

|  |  |
| --- | --- |
| **Buy PowerUp** | |
| **Use Case ID** | Team3\_BuyPowerUp |
| **Use Case Level** |  |
| **Scenario** | Player purchases a PowerUp to be used during a Game Match. |
| **Actor** | Player |
| **Pre-conditions** | 1. Player has the Game installed and running on their mobile device. 2. Player must be on the Main Menu Screen. |
| **Description** | 1. Use case begins when the Player taps the “Buy PowerUp” button. 2. The system shall then display a screen with all the available PowerUps for sale along with a description of every individual PowerUp. 3. The Players select a PowerUp. 4. The system shall then ask confirmation for the purchase of the PowerUp. 5. The Player confirms the purchase of the PowerUp. 6. The system shall then check if the Player has enough Coins to carry out the transaction. 7. The system shall then reward the Player with the requested PowerUp. 8. The system shall then retrieve the corresponding Coins from the Player. 9. Use case ends when the system displays a message with the newly purchased PowerUp and the amount of coins withdrawn from the Player’s account. |
| **Relevant Requirements** |  |
| **Post-conditions** | 1. Player will have the selected PowerUp in their PowerUp inventory for future matches, Coin will be deducted from their account. |
| **Alternative Courses of Actions** | 1. If a PowerUp fails to be purchased and Coins were withdrawn, the system shall reimburse the Player with the withdrawn Coins. 2. If the Player does not have enough Coins to purchase the PowerUp, the system will display a message saying “Insufficient coins, buy more?” with a “Buy Coins!” button. 3. If a PowerUp fails to be purchased and no Coins were withdrawn, the system will notify the player and ask the Player to try again. |
| **Exceptions** | 1. Coins are deducted and PowerUp was not transferred to Player’s account. 2. Coins are not deducted and PowerUp was not transferred to Player’s account. |
| **Related Use Case(s)** | 1. Team3\_BuyPowerUp |
| **Decision Support** | |
| **Frequency** | Indeterminate |
| **Criticality** | Low. Game is not dependent on PowerUps. |
| **Risk** | Medium. Implementing this use case requires the Game to communicate with the server. |
| **Constraints** | 1. Player has sufficient Coins to buy PowerUp. |
| **Modification History** | |
| **Owner** | Ariel Diaz |
| **Initiation Date** | 05/31/14 |
| **Date Last Modified** | 06/05/14 |

|  |  |
| --- | --- |
| **Use PowerUp** | |
| **Use Case ID** | Team3\_UsePowerUp |
| **Use Case Level** |  |
| **QQScenario** | Player disrupts or alters the Map Screen of other Players using a PowerUp. |
| **Actor** | Player |
| **Pre-conditions** | 1. Player has slid the pull out menu from the Map Screen. 2. Player must have available PowerUps. |
| **Description** | 1. Use case begins when the Player taps the accordion PowerUp heading in the pull-out menu. 2. The system shall provide the Player with a screen with all of the PowerUps available to the Player. 3. The Player shall then tap the PowerUp they desire. 4. The system shall then grant the Player with the features they selected PowerUp contains. 5. Use case ends when the pull-out menu slides back in, showing the Map Screen. |
| **Relevant Requirements** |  |
| **Post-conditions** | 1. Player will cause some disruption or alteration to the Map Screen based on the PowerUp selected for a duration of time specified by the specific PowerUp. |
| **Alternative Courses of Actions** | 1. Player slides pull-out menu back in without selecting a PowerUp. |
| **Exceptions** | 1. PowerUp is consumed and did have its intended effect. 2. PowerUp duration is longer than specified than in the PowerUp catalog. 3. PowerUp duration is shorter than specified than in the PowerUp catalog. |
| **Related Use Case(s)** | 1. Team3\_BuyPowerUp |
| **Decision Support** | |
| **Frequency** | Approximately once every Game Match. |
| **Criticality** | Low. Game is not dependent on PowerUps. |
| **Risk** | Medium. Implementing this use case requires the Game to communicate with the server. |
| **Constraints** | 1. Player must have available PowerUp. 2. After using PowerUp the player undergoes a two minute cool down period. |
| **Modification History** | |
| **Owner** | Ariel Diaz |
| **Initiation Date** | 05/31/14 |
| **Date Last Modified** | 06/05/14 |

|  |  |
| --- | --- |
| **Get Tagged** | |
| **Use Case ID** | Team3\_GetTagged |
| **Use Case Level** |  |
| **Scenario** | Prey type Player is Tagged and transitioned into a Predator type Player. |
| **Actor** | Prey Player, Predator Player |
| **Pre-conditions** | 1. Player must be in a Game Match. 2. Prey and Predator must be in near vicinity. |
| **Description** | 1. Use case begins when the Prey receives a prompt asking whether they were tagged and to press the button “Tagged!”. 2. The Prey taps the button “Tagged!”. 3. The system shall transform the Prey into a Predator. 4. The system changes the UI layout to be that of a Predator UI. 5. The system sends an announcement to all players that a Prey was tagged. 6. The use case ends when the former Prey sees a Predator UI screen. |
| **Relevant Requirements** |  |
| **Post-conditions** | 1. The system registers the Prey as a Predator. 2. The Prey Map Screen changes to that of a Predator. |
| **Alternative Courses of Actions** | 1. If the prey does not tap the “Tagged!” button they will continue as prey until this use case begins again. 2. If more than 5 meters distance is found by the system before the prey has tapped the “Tagged!” button then the button disappears. |
| **Exceptions** |  |
| **Related Use Case(s)** | 1. Team3\_PlayMatch 2. Team3\_DataConnectionLoss 3. Team3\_GPSSignalLoss |
| **Decision Support** | |
| **Frequency** | Dependent how often the Prey is in the vicinity of the Predator. |
| **Criticality** | High. As this is core functionality. |
| **Risk** | High. Implementation requires extension use of Google Maps API as well as other advanced geolocation algorithms. |
| **Constraints** | The Prey and Predator must be within 5 meters of each other for the Prey to receive the prompt. |
| **Modification History** | |
| **Owner** | Justin Philips |
| **Initiation Date** | 06/03/14 |
| **Date Last Modified** | 06/06/14 |

|  |  |
| --- | --- |
| **Chain Power Up** | |
| **Use Case ID** | Team3\_ChainPowerUp |
| **Use Case Level** | High-Level |
| **Scenario** | The system shall prevent a player from exploiting PowerUps by using PowerUps back to back. This system is required due to the potential misuse that could occur by a player exploiting PowerUps such that they maintain a PowerUp effect for the entire duration of the game. |
| **Actor** | Prey Player, Predator Player |
| **Pre-conditions** | 1. Player must be in a Game Match. 2. Player must have at least one PowerUp in the player’s inventory. |
| **Description** | 1. Use case begins when the player uses a PowerUp from the in game pull-out menu described in Team3\_UsingPowerUp use case. 2. The system shall immediately invoke a 2 minute cool down on the player’s remaining PowerUps as the PowerUp used affect is applied to the map. 3. Use case ends when the 2 minute cool down expires and the player is free to use their remaining PowerUps. |
| **Relevant Requirements** |  |
| **Post-conditions** | 1. The system has removed the cool down from the players PowerUps. |
| **Alternative Courses of Actions** |  |
| **Exceptions** |  |
| **Related Use Case(s)** | 1. Team3\_UsePowerUp |
| **Decision Support** | |
| **Frequency** | Dependent on the player’s discretion of use and the number of PowerUps a player has purchased. |
| **Criticality** | Medium. Prevents exploitation of core gameplay features. |
| **Risk** | Low, no other cases are dependent on this case. |
| **Constraints** | 1. Cool down must last 2 minutes |
| **Modification History** | |
| **Owner** | Justin Philips |
| **Initiation Date** | 06/03/14 |
| **Date Last Modified** | 06/06/14 |

## 9.3. Appendix C: User Interface Designs

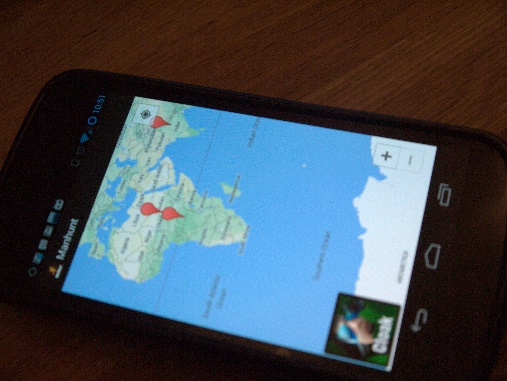
**Start Screen:**

User presses start button to join the game match instance.

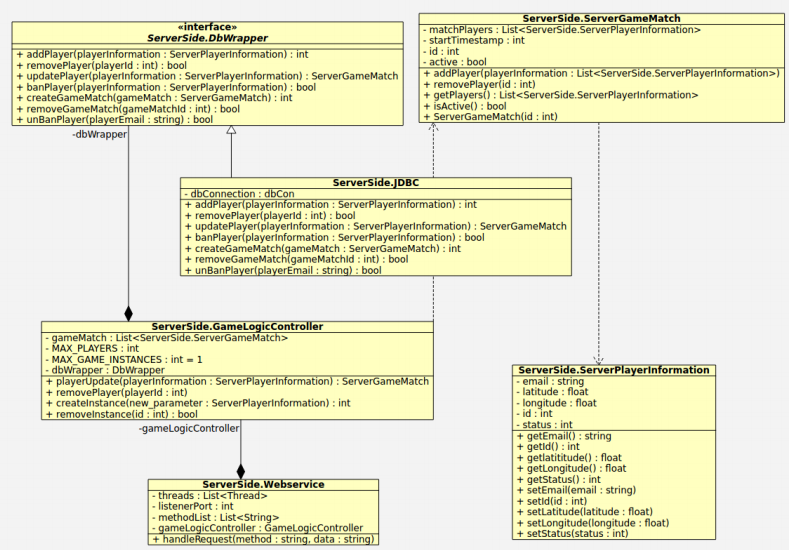


**Game Match Screen:**

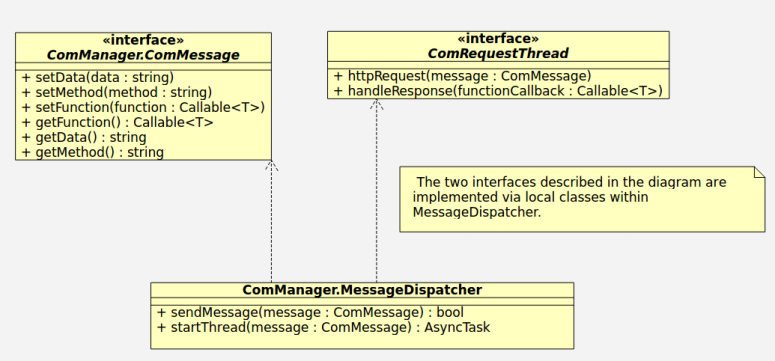
Screen displayed during a game match

****

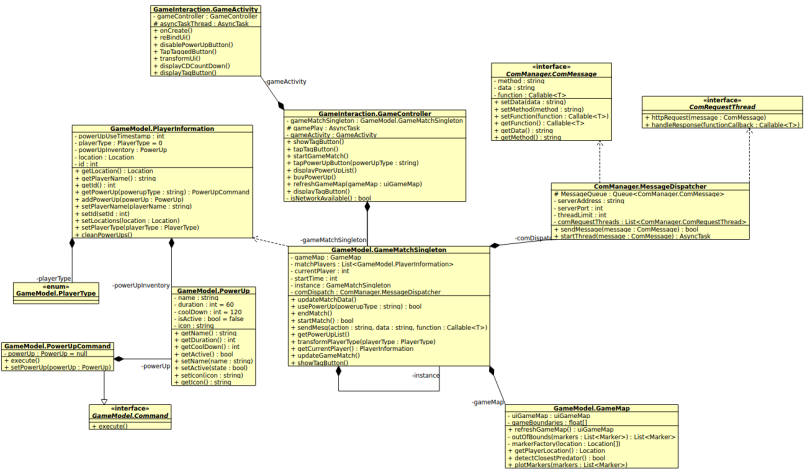
## 9.4. Appendix D: Class Diagrams



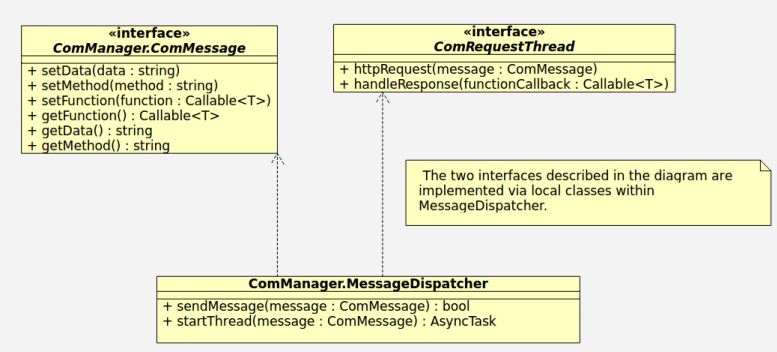
**Figure D.1**



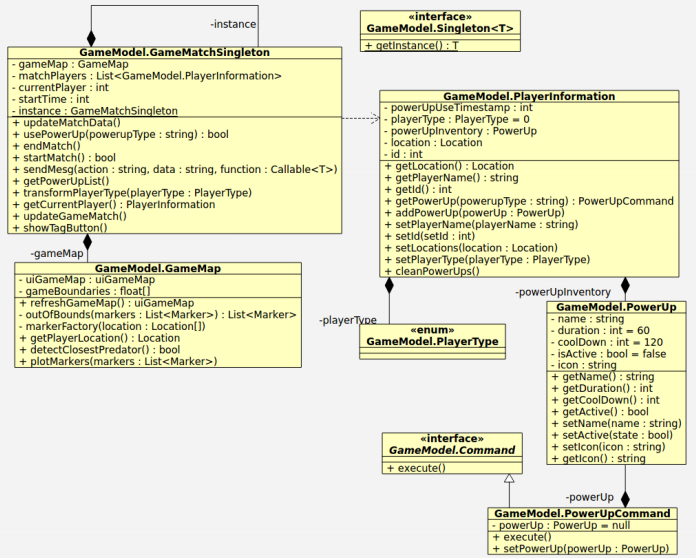
**Figure D.2**



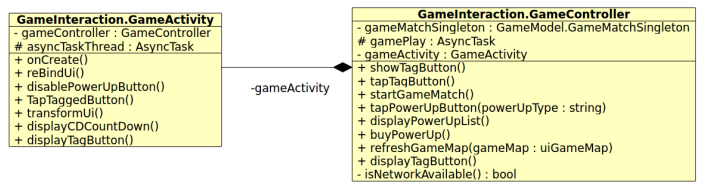
**Figure D.3**



**Figure D.4**

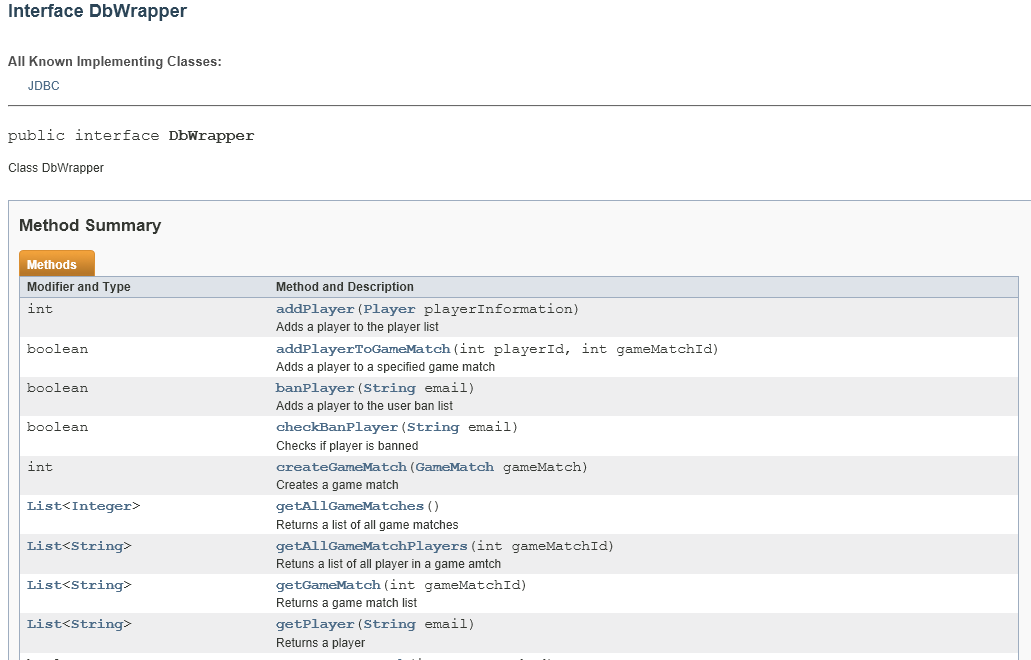
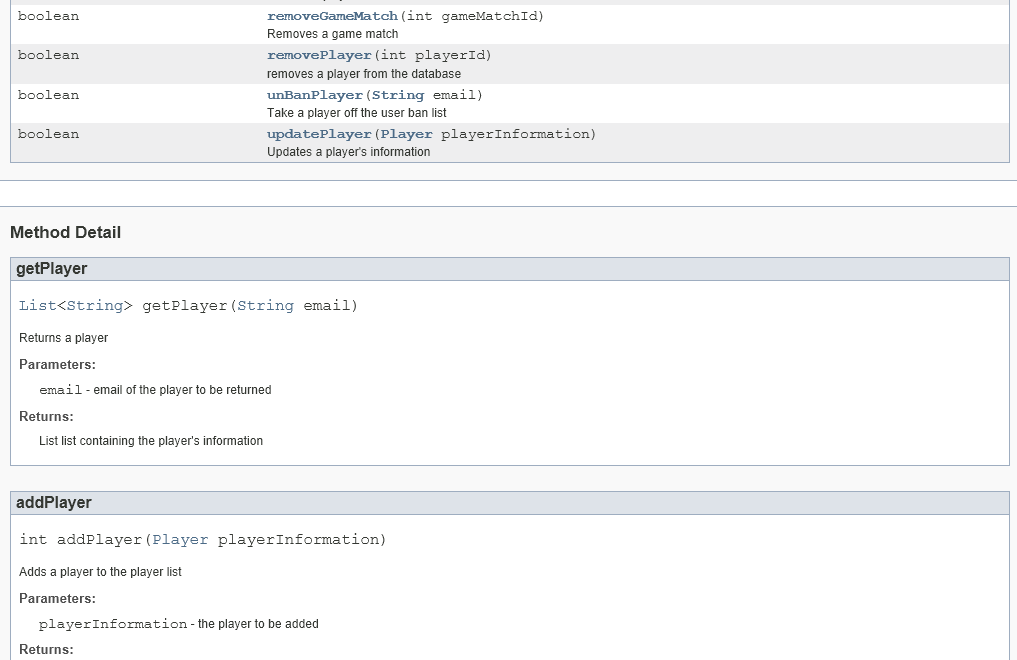
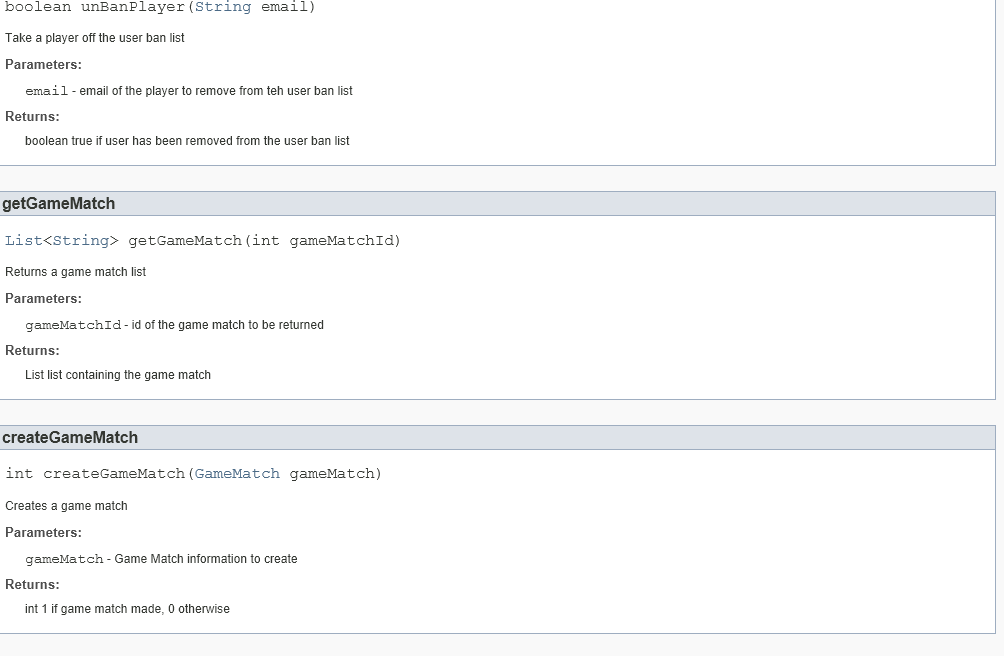
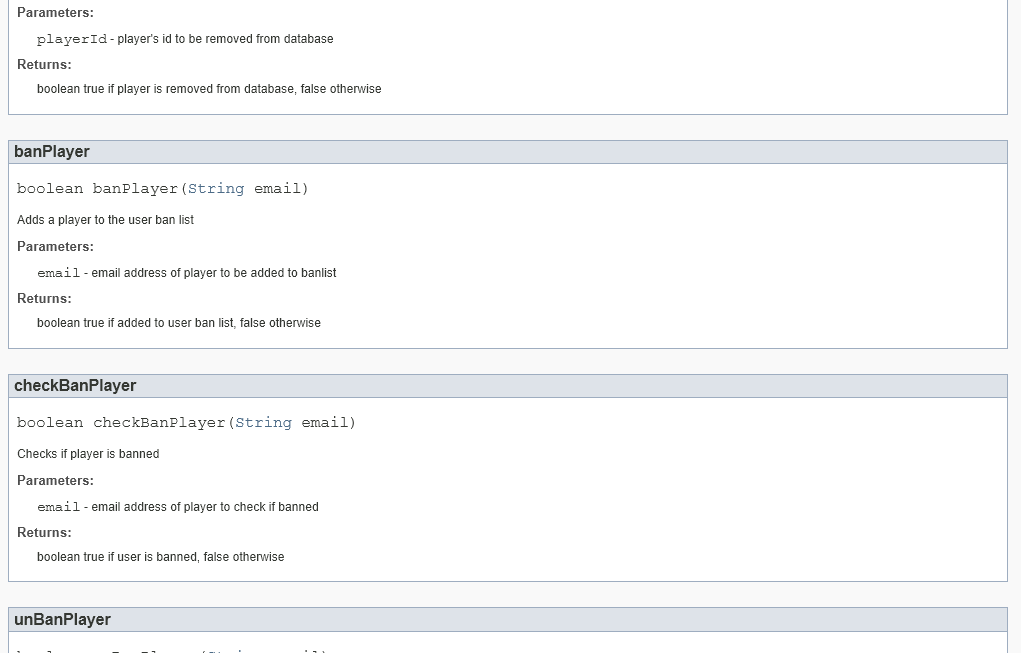
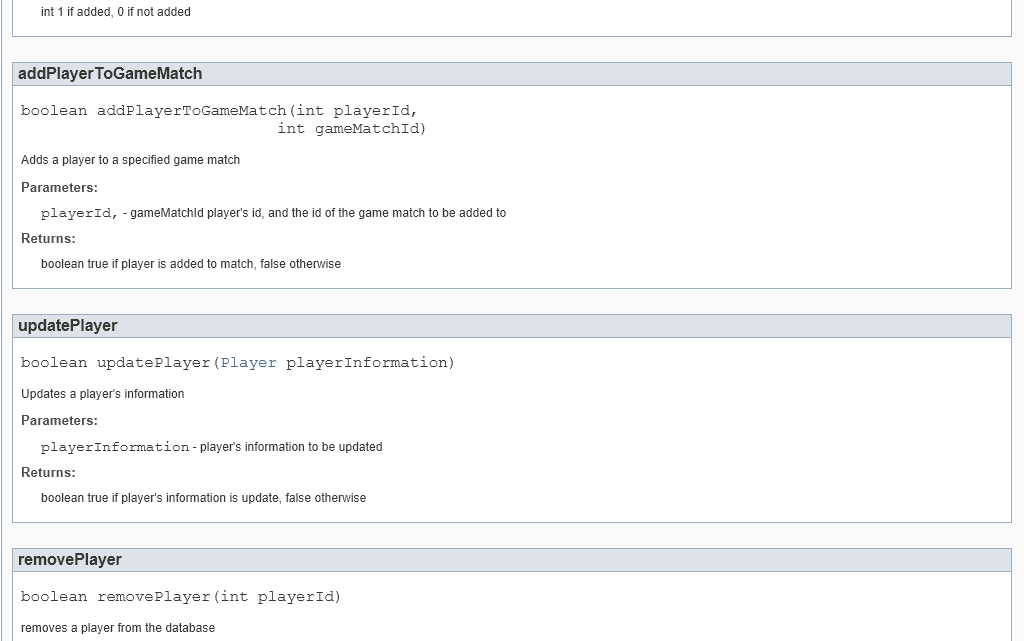
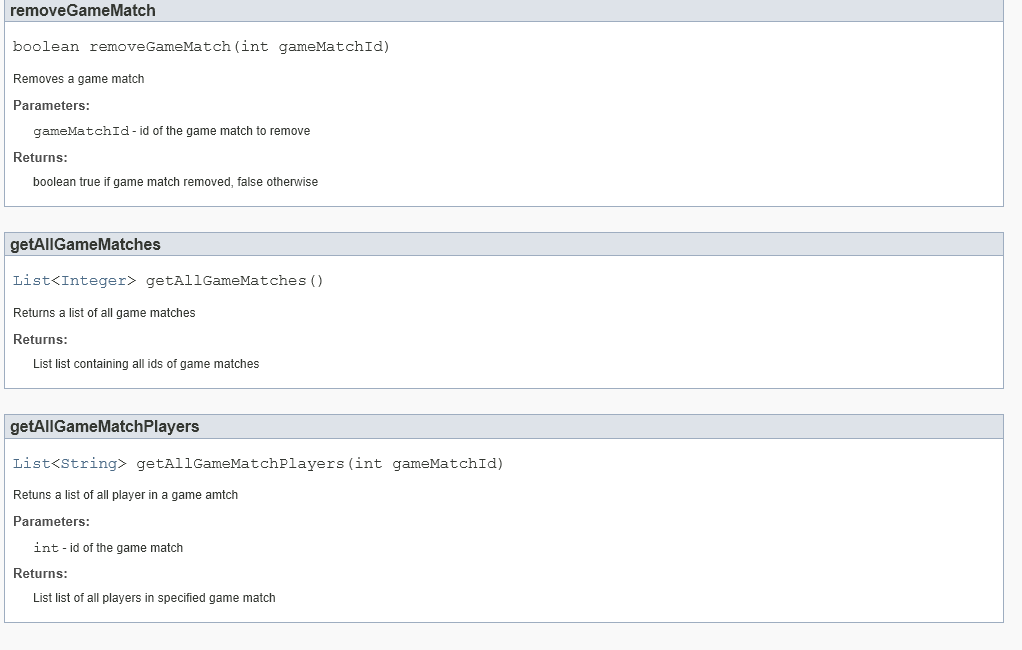


**Figure D.5**



**Figure D.6**

## 9.5. Appendix E: Class Interfaces

      9.6. Appendix F: Test Driver

package com.fiu.manhunt.test.driver;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.PrintWriter;

import java.io.UnsupportedEncodingException;

import java.util.Random;

import com.fiu.manhunt.data.JDBC;

import com.fiu.manhunt.server.PlayerMessageData;

import com.fiu.manhunt.test.modules.ControllerTest;

import com.google.gson.Gson;

public class TestDriver {

private static final int NUM\_TESTS = 3;

private static Random \_rnd;

/\*\*

\* @param args

\*/

public static void main(String[] args) {

\_rnd = new Random();

\_rnd.setSeed(0);

try {

JDBC db = new JDBC();

TestController();

} catch (Exception E) {

E.printStackTrace();

System.out.println("Something major broke.");

}

}

/\*\*

\* Controller for tester

\*/

private static void TestController() throws FileNotFoundException,

UnsupportedEncodingException {

ControllerTest ct = new ControllerTest();

PrintWriter writer = createLog("ControllerTest");

PlayerMessageData pmd = new PlayerMessageData();

PlayerMessageData.PlayerData pd = pmd.new PlayerData();

Gson gson = new Gson();

// good test cases

for (int i = 0; i < NUM\_TESTS; i++) {

// test within a range of four e-mails (this way we hit 4 new

// players and Num\_test - 4 updates)

pd.set\_email("test" + (i % 4) + "@somewhere.com");

// random locations.

pd.set\_lat((float) 1.2234 \* (float) getRandomRange(1, 25));

pd.set\_long((float) 1.2234 \* (float) getRandomRange(5, 40));

// expected input

String playerJsonData = gson.toJson(pd);

writer.println("\n--Test " + i + " input--\n");

writer.println(playerJsonData);

// write out results

writer.println("\n--Test " + i + " output--\n");

writer.println(ct.TestPlayerUpdate(playerJsonData));

}

writer.close();

/\*

// bad test cases

for (int i = 0; i < NUM\_TESTS; i++) {

// test with valid e-mail, invalid e-mail, and no e-mail

// test1@somewhere.com should be in ban list.

if (i % 3 == 0)

pd.set\_email("test" + (i % 4) + "@somewhere.com");

else if (i % 3 == 1)

pd.set\_email("test" + (i % 4));

else

pd.set\_email("");

pd.set\_lat((float) 1.2234 \* (float) getRandomRange(1, 25));

pd.set\_long((float) 1.2234 \* (float) getRandomRange(5, 40));

// expected input

String playerJsonData = gson.toJson(pd);

writer.println("\n--Bad E-mail Test " + i + " input--\n");

writer.println(playerJsonData);

// write out results

writer.println("\n--Bad E-mail Test " + i + " output--\n");

writer.println(ct.TestPlayerUpdate(playerJsonData));

}

// bad localtion

for (int i = 0; i < NUM\_TESTS; i++) {

pd.set\_email("test" + (i % 4) + "@somewhere.com");

if (i % 3 == 0) {

pd.set\_lat(-1);

pd.set\_long(-1);

}

else if (i % 3 == 1) {

pd.set\_lat(0);

pd.set\_long(0);

}

else {

pd.set\_lat((float)1000.0 \* (i % 5));

pd.set\_long((float)1000.0 \* (i % 5));

}

// expected input

String playerJsonData = gson.toJson(pd);

writer.println("\n--Bad E-mail Test " + i + " input--\n");

writer.println(playerJsonData);

// write out results

writer.println("\n--Bad E-mail Test " + i + " output--\n");

writer.println(ct.TestPlayerUpdate(playerJsonData));

} \*/

writer.close();

}

// support functions

/\*\*

\* @params prefix the name of the output document created

\* @return writer for printing output document

\*/

private static PrintWriter createLog(String prefix)

throws FileNotFoundException, UnsupportedEncodingException {

int i = 0;

String fileName = null;

while (i >= 0 && i <= 100) {

File log = new File(prefix + "-" + i++ + ".txt");

if (!log.exists()) {

fileName = log.getName();

break;

}

}

PrintWriter writer = new PrintWriter(fileName, "UTF-8");

return writer;

}

private static int getRandomRange(int min, int max) {

return (\_rnd.nextInt((max - min) + 1) + min);

}

}

## 9.7. Appendix G: Diary of Meetings and Tasks

|  |  |
| --- | --- |
| **Date** | May 17, 2014 |
| **Location** | John C. Comfort Laboratory |
| **Start time** | 4:30pm |
| **End time** | 5:00pm |
| **In attendance** | Ariel Diaz  Matthew Santiago |
| **Late** | None |
| **Agenda** | Brainstorming session |
| **Summary of Discussion** | 4:30 Came up with several ideas for projects.  5:00 Meeting Dismissed |
| **Assigned tasks** | Everyone  Continue brainstorming ideas |

|  |  |
| --- | --- |
| **Date** | May 18, 2014 |
| **Location** | John C. Comfort Laboratory |
| **Start time** | 5:00pm |
| **End time** | 6:03pm |
| **In attendance** | Ariel Diaz  Justin Phillips  ~~Will Rodriguez~~  Matthew Santiago |
| **Late** | None |
| **Agenda** | Choose idea for Project |
| **Summary of Discussion** | 5:00 Continued Brainstorming ideas for projects  5:20 Made list of all viable ideas and narrowed down  5:45 Started process for choosing project  6:00 Project idea chosen  6:00 Meeting Dismissed |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | May 23, 2014 |
| **Location** | John C. Comfort Laboratory |
| **Start time** | 4:30pm |
| **End time** | 5:28pm |
| **In attendance** | Musa Ahmed  Ariel Diaz  Justin Phillips  Matthew Santiago |
| **Late** | None |
| **Agenda** | System Conceptualization |
| **Summary of Discussion** | 4:30 Discussed system requirements  4:45 Discussed Tagging process  5:03 Assigned tasks among members  5:28 Meeting Dismissed |
| **Assigned tasks** | Appendix A  Who: Musa Ahmed, Justin Phillips  When: May 27th  Hardware and Software Requirements  Who: Ariel Diaz, Matthew Santiago  When: May 30th  Project Organization  Who: Musa Ahmed, Justin Phillips  When: May 31st  Scope of System  Who: Justin Phillips  When: June 3rd |

|  |  |
| --- | --- |
| **Date** | May 28, 2014 |
| **Location** | John C. Comfort Laboratory |
| **Start time** | 4:30pm |
| **End time** | 5:03pm |
| **In attendance** | Musa Ahmed  Ariel Diaz  Justin Phillips  Matthew Santiago |
| **Late** | None |
| **Agenda** | System Requirements discussion |
| **Summary of Discussion** | 4:30 Discussed system requirements  5:03 Meeting Dismissed |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | May 30, 2014 |
| **Location** | John C. Comfort Laboratory |
| **Start time** | 5:00pm |
| **End time** | 5:45m |
| **In attendance** | Ariel Diaz  Matthew Santiago |
| **Late** | None |
| **Agenda** | Discussing Tagging system |
| **Summary of Discussion** | 5:00 Finalized Tagging system  5:45 Meeting Dismissed |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | June 2, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 9:00pm |
| **End time** | 10:45pm |
| **In attendance** | Musa Ahmed  Ariel Diaz  Justin Phillips |
| **Late** | Matthew Santiago |
| **Agenda** | Use Cases |
| **Summary of Discussion** | 9:00 Brainstorm Use Cases  9:35 Assigned use cases to group members  10:20 Came up with Use Case structure  10:45 Meeting Dismissed |
| **Assigned tasks** | Matthew Santiago  CellDataLoss, AvoidTag, LeaveRejoinCheat, BuyCoins, LeaveLobby,  Musa Ahmed  HunterCheats2Devices, PlayAMatch, InGameChat, CheckStore  Ariel Diaz  UsingPowerUp, DisplayingOutOfBoundsPlayer, LeaveGameMatch,  BuyingPowerUp, CreatingGameInstance  Justin Phillips  ChainingItemUse, GettingTagged, EndAGame, StartGameMatch,  JoinAnExistingGame |

|  |  |
| --- | --- |
| **Date** | June 7, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 11:00am |
| **End time** | 3:36pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed |
| **Late** | None |
| **Agenda** | SRD |
| **Summary of Discussion** | 11:00 Starting piecing together SRD  12:03 Peer Review of Use Cases  1:11 Peer review of other finished sections  2:20 Break  2:40 Finish System Scope  3:01 Finish System Purpose  3:36 Dismiss Meeting |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | June 8, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 11:00am |
| **End time** | 4:39pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed |
| **Late** | Matthew Santiago |
| **Agenda** | SRD |
| **Summary of Discussion** | 11:00 Gathering Meeting Diaries  12:03 Peer Review of Use Cases  1:11 Peer review of other finished sections  2:20 Break  2:40 Finish System Scope  3:01 Finish System Purpose  3:35 Final review and editing of document  4:00 Finish design of document  4:39 Dismiss Meeting |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | June 18, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 11:00am |
| **End time** | 1:00pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | Design Document |
| **Summary of Discussion** | 11:00 Work Breakdown  12:03 Task creation  12:40 Task due dates  1:00   Meeting End |
| **Assigned tasks** | Musa Create SCRUM            Assign tasks among group |

|  |  |
| --- | --- |
| **Date** | June 22, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 11:00am |
| **End time** | 1:00pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | SCRUM Review |
| **Summary of Discussion** | 11:00 Review SCRUM  11:30 Rearrange tasks assignments  12:15 Validate due dates  1:00   Meeting End |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | June 24, 2014 |
| **Location** | JCCL |
| **Start time** | 8:30pm |
| **End time** | 9:30pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | Design Document |
| **Summary of Discussion** | 8:30   Teammate Check-up  8:50   Individual tasks  9:30   Meeting End |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | June 27, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 11:00am |
| **End time** | 1:00pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | Design Document |
| **Summary of Discussion** | 11:00 Individual tasks  12:30 Peer Editing of completed sections  1:00   Meeting End |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | July 03, 2014 |
| **Location** | Skype (remote) |
| **Start time** | 9:00am |
| **End time** | 11:30pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed |
| **Late** | Matthew Santiago |
| **Agenda** | Sequence Diagrams |
| **Summary of Discussion** | 9:00  Work Breakdown  10:03 Task creation  11:00 Diagram assignments  11:30 Meeting End |
| **Assigned tasks** | Musa Server Subsystem  Ariel   Game Model Subsystem  Matthew        Menu Subsystem  Justin Game Match Subsystem |

|  |  |
| --- | --- |
| **Date** | July 05, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 2:00pm |
| **End time** | 10:30pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | Design Document |
| **Summary of Discussion** | 2:00   System design rework  3:00   Sequence Diagrams  5:30   Finish introduction sections  6:30   Peer editing of completed sections  7:00   Break  9:30   Continue peer editing  10:20 Discuss relief of delayed due date  10:30 Meeting End |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | July 06, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 1:00pm |
| **End time** | 4:00pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | Design Document |
| **Summary of Discussion** | 1:00   Finalization of Design Document sections  3:00   Piece together document  4:00   End Meeting |
| **Assigned tasks** | Presentation Breakdown |

|  |  |
| --- | --- |
| **Date** | July 12, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 12:00 pm |
| **End time** | 1:00 pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** | None |
| **Agenda** | Manhunt Coding |
| **Summary of Discussion** | 12:00 Coding Updates 12:30 Git repository set up  1:00 Meeting End |
| **Assigned tasks** | All: Obtain access to GitHub Repository |

|  |  |
| --- | --- |
| **Date** | July 21, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 2:00 pm |
| **End time** | 3:00 pm |
| **In attendance** | Justin Phillips  Matthew Santiago |
| **Late** | None |
| **Agenda** | Manhunt Responsibilities |
| **Summary of Discussion** | 2:00 Assign Areas to begin coding  3:00 Meeting End |
| **Assigned tasks** | Ariel Manhunt Android Coding  Justin ServerSide: General  Matthew ServerSide: GameMatch and Player  Musa Serverside: JDBC and DBWrapper |

|  |  |
| --- | --- |
| **Date** | July 26, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 2:00 pm |
| **End time** | 9:00 pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** | None |
| **Agenda** | Manhunt Coding |
| **Summary of Discussion** | 2:00 Coding Updates  2:30 Individual Coding  4:00 Break  6:00 Coding Updates  6:30 Individual Coding  8:00 TeamViewer started, Manhunt Coding  9:00 Meeting End |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | July 27, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 2:00 am |
| **End time** | 8:00 pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** |  |
| **Agenda** | Manhunt Coding |
| **Summary of Discussion** |  |
| **Assigned tasks** | None |

|  |  |
| --- | --- |
| **Date** | July 30, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 9:30 pm |
| **End time** | 10:30 pm |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed  Matthew Santiago |
| **Late** | None |
| **Agenda** | Manhunt Coding |
| **Summary of Discussion** | 9:30 Coding Updates  10:00 Start Test Drivers  10:30 Meeting End |
| **Assigned tasks** | Matthew Final Document |

|  |  |
| --- | --- |
| **Date** | July 31, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** |  |
| **End time** |  |
| **In attendance** | Ariel Diaz  Justin Phillips  Musa Ahmed |
| **Late** |  |
| **Agenda** | Manhunt Coding |
| **Summary of Discussion** |  |
| **Assigned tasks** |  |

|  |  |
| --- | --- |
| **Date** | August 1, 2014 |
| **Location** | Google Hangouts (remote) |
| **Start time** | 4:00 |
| **End time** | 12:00 |
| **In attendance** |  |
| **Late** |  |
| **Agenda** |  |
| **Summary of Discussion** | 11:00 Submit Package |
| **Assigned tasks** | All Rejoice |