Android Mystery Application
Requirements Definition
Version 1.0

Jennifer Coston
Jordan Hitrik
Caleb Gutshall
Rebecca Rush

2/16/2012
7.1 Classes of Tests ................................................................. 30
7.2 Expected Software Response .................................................. 30
7.3 Performance Bounds ........................................................... 31
Works Cited ............................................................................ 32
Table of Figures

Figure 1: Use Case Diagram for the one user ................................................................. 7
Figure 2: Entity Relationship Diagram............................................................................. 11
Figure 3: Sample Data Dictionary.................................................................................... 12
Figure 4: Mystery Game State Diagram........................................................................... 28
1.0 Introduction

This document defines the requirements for the Android Mystery Game. This mobile application will provide the user with hours of entertainment. The user will be able to play this game multiple times because of on the spot story generation. Android applications have several ways to be downloaded on to the phone. This application will be available on the android market and also hosted on a team website. The application will be made to run on Android’s newest operating system, Android 4.0 (Ice Cream Sandwich). Mobile phones running the Android OS are not made to fit one specification, so this application will have to be adjustable to fit on a variety of the pones out there on the market.

1.1 Goals and objectives

This android mobile application’s goal is to provide entertainment to the user. The game should be easy and fun to play. The length of the game will be manageable but the user will have the option to save and continue later. This game will bring old school mystery games to the modern mobile game world.

1.2 Statement of scope

The scope of the project will be limited to the features of JavaFX and what the Android 4.0 OS can handle. The input to the application will be limited to simple inputs for user info and solving the mystery and a lot of pre-defined inputs that the user can pick from throughout the game. The output from the application is the display of the game on the phone and the possibility of an outside leader board on a separate website or the option to post your game status to Facebook.

1.3 Software context

This software application is intended as a commercial product. The application must meet the gaming needs of the typical android mobile gamer. Since there is really no particular customer in mind, the game must appeal to the widest audience to entice people to want to download it on their phone.
1.4 Major constraints

The Android 4.0 OS will limit the amount of functionality of this application to what the operating system can handle. The layout of the game is constrained to what will work well on multiple android phone styles. Creating an outside leader board will pose issues of connecting with a server to post results. If the option to post results to Facebook is allowed, there are concerns with making the app compatible. Finally the amount of storage space on the phone itself will limit the games saving capabilities.
2.0 Usage scenario

This section provides a usage scenario for the software. It organized information collected during requirements elicitation into use-cases.

2.1 User profiles

There is only one category of user for this system, the player.

2.2 Use-cases

![Use Case Diagram for the one user](image)

Figure 1: Use Case Diagram for the one user
The user will access the mystery game using their Android phone. When the game starts the user will be presented with a menu. From this menu the user will be presented with five options: to start a new game, to continue a saved game, to create a player, to view the help, or to quit. If the user chooses to start a new game, a new screen will be displayed asking the user to specify the length of the game and to select their player. After these choices are made, the map will be displayed and play can begin. If the user chooses to continue a saved game they will be directed back to the map where they can view their progress and continue playing. By selecting the option to create a player, the user will be directed to a screen where they can enter their name and gender as well as selecting the image for their character. This information will be used to personalize each player’s experience with the story. The help option will provide the user with basic instructions on the rules of the game. The final option quit will exit the program.

To play the game the user will select the first room from the map. The player will be presented with a background story followed by an image of the room. While in the room, the player will be able to talk to any characters and to solve puzzles. This will reward the player with clues to the mystery and items that will be needed to solve future puzzles. All of the clues will be recorded in a logbook that can be accessed by the player at any time. When the user is finished with the room another room will be unlocked. The player will be returned to the map where they can pick their next destination, save the game, and return to the main menu. After the user has successfully traversed through all of the rooms and solved all of the puzzles they will be presented with a screen asking them to solve the mystery. Based on their choice, a concluding message will be displayed.
3.0 Data Model and Description

3.1 Data Description

The Mystery game for android will use a variety of data objects during the completion of a game. These data objects will help to store game states and simplify the process of controlling and saving the game. They allow for modularity and variability of where different objects will appear throughout the game.

3.1.1 Data objects

- Game – The game is the main data object of the android application. The game object will be created when a user wants to play a new game. The game is considered the container for all other data objects needing to be stored throughout that particular game being played. The game will store items such as the story generated for this game, the rooms being attended, and the user playing the game.

- Room – The room is an important data object that corresponds to the mystery being solved during a particular game. The room contains objects such as items, clues, puzzles, characters, and pieces of the story. The room will hold start states and end states. In a particular room, there may be more than one start and end state, which means the user, can visit the room more than once and learn different things from the same room. The rooms will be generated with their data to store when a new game is started allowing the rooms to change what data they are storing.

- Item – The item data object stores a particular item that the user will have to retrieve and use in order to obtain some type of help for the game. This is a low level data object because it does not store any other data objects in it.

- Clue – The clue data object stores a particular clue that is useful to the user to help solve the mystery. The clue is also a low level data object.

- Puzzle – The puzzle data object stores a variety of puzzles that the user will solve in order to obtain things such as a clue or an item. The puzzle must store the answer as well as what to do if the user gets the puzzle right or wrong.

- Character – The character is a data object that stores information about the other characters throughout the game. These characters will be able to talk to the user in
different rooms. Not only will they store their different conversations to the user but they will also store the different appropriate responses that the user can say.

- **Story** – The story data object contains the generated story for a particular game. Each game will have a story generated at the initial start of the game. This data object is important so that the user will have the same story throughout the game.

- **Quest Log** – The quest log is a higher level data object that is used as a container for a lot of smaller data objects. It contains the rooms entered, characters talked to, items found, and clues given throughout the game. The log can be used a reference to help solve the mystery at the end of the game.

- **User** – The user is a main data object that stores information about the person playing the game. Information stored would be username, password, email address, and character image. When a new game is started the user has the option to use their predefined user information or to create a new user. The users name will be used throughout the game to make the user feel more a part of the game.

- **Leader Board** – The leader board will store users and their game to create a ranking system that hopes to invoke users to want to play better at the game. This will keep a predetermined amount of users in the top 15 ranking spots for the game. This is not a necessary data object for the game but helps to improve upon the extra capabilities of the game.

### 3.1.2 Relationships

The relationships among the data object can be seen in greater detail in section 3.1.3 Figure 1. A new game or save game can be retrieved from the main menu. Each game will contain a user, quest log, rooms, and story. A user will contain information about the specific person playing the game. The quest log will chronologically record the happenings in the game. Each room will contain a collection of items, clues, puzzles, characters, and pieces of the story. The entire story and ending will need to be stored because it is generated when the story is started. These relationships work together to create a multidimensional game.
3.1.3 Complete data model

![Entity Relationship Diagram]

Figure 2: Entity Relationship Diagram
3.1.4 Data dictionary

The data dictionary will be kept in an excel spreadsheet that describes the basic organization of the database and the data that will be stored to an xml file on the users phone. A small example of what the data dictionary organization looks like follows below. The bold items are the bigger “containers” and the others are smaller items within them.

<table>
<thead>
<tr>
<th>Data</th>
<th>Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>varchar</td>
</tr>
<tr>
<td><strong>Quest Log</strong></td>
<td>Log Object</td>
</tr>
<tr>
<td>quest log id</td>
<td>integer</td>
</tr>
<tr>
<td><strong>Room</strong></td>
<td>Room Object</td>
</tr>
<tr>
<td>room id</td>
<td>integer</td>
</tr>
<tr>
<td><strong>Character</strong></td>
<td>Character Object</td>
</tr>
<tr>
<td>character id</td>
<td>integer</td>
</tr>
<tr>
<td>name</td>
<td>varchar</td>
</tr>
<tr>
<td>conversations</td>
<td>varchar</td>
</tr>
<tr>
<td>responses</td>
<td>varchar</td>
</tr>
<tr>
<td><strong>Item</strong></td>
<td>Item Object</td>
</tr>
<tr>
<td>item id</td>
<td>integer</td>
</tr>
<tr>
<td>feature</td>
<td>varchar</td>
</tr>
<tr>
<td>purpose</td>
<td>varchar</td>
</tr>
<tr>
<td><strong>Story</strong></td>
<td>Story Object</td>
</tr>
<tr>
<td>story id</td>
<td>integer</td>
</tr>
<tr>
<td>story pieces</td>
<td>varchar</td>
</tr>
<tr>
<td>result</td>
<td>varchar</td>
</tr>
<tr>
<td><strong>Clue</strong></td>
<td>Clue Object</td>
</tr>
<tr>
<td>clue id</td>
<td>integer</td>
</tr>
<tr>
<td>clue</td>
<td>varchar</td>
</tr>
<tr>
<td><strong>Puzzle</strong></td>
<td>Puzzle Object</td>
</tr>
<tr>
<td>puzzle id</td>
<td>integer</td>
</tr>
<tr>
<td>answer</td>
<td>varchar</td>
</tr>
<tr>
<td>responses</td>
<td>varchar</td>
</tr>
</tbody>
</table>

*Figure 3: Sample Data Dictionary*
4.0 Functional Model and Description

4.1.0 Menu Functions:

StartNewGame() :

**Description:** This function starts a new game for the user to play through

**Processing narrative:** This function will ask for a User object and create a Log object for that user, and then initialize a Game Object. It will then invoke the functions generateStory(), generateMap(), and displayMap() with will be locally stored in the Game Object. This will start a Game state.

**Inputs:** User Object

**Outputs:** A New Game State is initialized.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system. A user Object will need to exist in order to start a new game.

4.1.1 Game Functions:

generateStory() :

**Description:** Generates Story Content for the user.

**Processing narrative:** This function will randomly create a Mystery in varchar format and a result of the story in varchar.

**Inputs:** name

**Outputs:** None.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
generateMap() :

**Description:** Generates a Map and Rooms for the User to explore.

**Processing narrative:** Creates a Map Object and Randomly fills it with Room Objects each which contain a Puzzle Object for the user to solve.

**Inputs:** None

**Outputs:** None

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

saveGame() :

**Description:** Saves the current game state to a xml file.

**Processing narrative:** Prompts the user for a name to save the file as then Writes the current Game Object to a XML file which contains all of the current games data.

**Inputs:** None

**Outputs:** Game XML file.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
displayMap() :

**Description:** Displays a graphical interpretation of the Game Map

**Processing narrative:** Using tools giving with Java will display a 2D interpretation of the Map Object along with Buttons for the user it interact with to solve the mystery.

**Inputs:** Map Object

**Outputs:** 2D map to the system Screen

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

selectRoom() :

**Description:** Gives the user the option to enter a room.

**Processing narrative:** Prompts the user if they would like to enter the room, if they do it will initialize a Puzzle Object to be solved. Others it takes them back to the Map.

**Inputs:** None.

**Outputs:** None.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
displayStory() :

**Description:** Displays the Mystery and Story content on the Screen

**Processing narrative:** Takes data from the generated story and displays it to the screen.

**Inputs:** None.

**Outputs:** Text display of the story.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

playPuzzle() :

**Description:** Provides the user with the options to solve the puzzle.

**Processing narrative:** Gives tools for solving a Puzzle Object based off of the Puzzle Object.

**Inputs:** Vary

**Outputs:** None.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
getReward() :

**Description:** Adds a Item to the User

**Processing narrative:** When a puzzle is solved the user may receive a reward this is randomly determined. The reward is stored as a Item of the user.

**Inputs:** Boolean Condition.

**Outputs:** Item Object.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

recordLog():

**Description:** Records a log of the current actions in the Story

**Processing narrative:** When the user leaves the room, the events that took place in that room are saved in the log object.

**Inputs:** Elements of Room

**Outputs:** None

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
accessLog():

Description: Displays the Quest Log.

Processing narrative: Displays all the actions that are contained in the Quest Log.

Inputs: None.

Outputs: Graphical display of the quest log.

Performance Issues: Performance will be based off of the space and memory available at the time.

Design Constraints: The function will need to work within the constraints of a touch interface and graphic abilities of the system.

unlockRoom():

Description: Checks for Item and unlocks room it item is contained.

Processing narrative: Check is the user as the correct item if he does the room is now available to enter on the Map.

Inputs: Item Object

Outputs: None.

Performance Issues: Performance will be based off of the space and memory available at the time.

Design Constraints: The function will need to work within the constraints of a touch interface and graphic abilities of the system.
solveMystery():

**Description:** Checks clues to see if Mystery is solved.

**Processing narrative:** If the correct clues have been collected and given this will say the mystery is solved and the game is over.

**Inputs:** The user’s clues.

**Outputs:** A Congratulations display to the screen.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

continueGame():

**Description:** List current saved games and allows user to select one.

**Processing narrative**

**Inputs**

**Outputs**

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
4.1.2 Save Functions:

loadStory() :

   Description: Loads the saved story from an XML file.

   Processing narrative
   Inputs
   Outputs

   Performance Issues: Performance will be based off of the space and memory available at the time.

   Design Constraints: The function will need to work within the constraints of a touch interface and graphic abilities of the system.

loadMap() :

   Description: Loads the current map from an XML file.

   Processing narrative: Regenerates the Map Object based on the XML file it is given.

   Inputs: XML file
   Outputs: None.

   Performance Issues: Performance will be based off of the space and memory available at the time.

   Design Constraints: The function will need to work within the constraints of a touch interface and graphic abilities of the system.

createPlayer() :

   Description: List functions to create a new Player

   Processing narrative: Displays a list of functions that will create a the user.

   Inputs: None.
   Outputs: None.

   Performance Issues: Performance will be based off of the space and memory available at the time.

   Design Constraints: The function will need to work within the constraints of a touch interface and graphic abilities of the system.
4.1.3 Accounts Functions:

setName():

**Description:** Allows user to enter the Player the name.

**Processing narrative:** Prompts the user to enter the name of there character in String format.

**Inputs:** A String

**Outputs:** None.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

setGender():

**Description:** Allows the user to select the Gender of the Player.

**Processing narrative:** Lets the user set weather there character is male or female.

**Inputs:** List of Options.

**Outputs:** None

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
setProfile():

**Description:** Allows the user to select an image to represent the Player in the game.

**Processing narrative:** Ask the user for an image file on the system's hard drive and assigns it to the user's profile.

**Inputs:** A image file.

**Outputs:** None.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

viewHelp():

**Description:** Will display a screen explaining the actions available in the game and the rules of the game.

**Processing narrative:** Prints the rules and action available in the game to the screen.

**Inputs:** None

**Outputs:** Displays a graphical list of the game rules.

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.

quit():

**Description:** Exits the Game application.

**Processing narrative:** Closes the application using the OS system features.

**Inputs:** None

**Outputs:** None

**Performance Issues:** Performance will be based off of the space and memory available at the time.

**Design Constraints:** The function will need to work within the constraints of a touch interface and graphic abilities of the system.
4.2.0 Software Interface Description
This section describes the manner with which the software will interface with things external to itself.

4.2.1 External System Interfaces
The chief interface between the software and an external product occurs with the Android operating system that runs the mobile phone. System calls to this operating system will be necessary; all calls will occur through the Java programming language, and they will adhere to the API constraints of the Java modules used.

4.2.2 Human Interface
Due to the mobile platform, the human interface is naturally the phone’s touch-sensitive screen. Android development modules create the link between human touches and gestures on the screen and the underlying actions at the software level. Nearly all interfacing between the user and the software will occur via Android’s touch click sensing to navigate through graphical menus and elements. A limited amount of the touch-sensitive on-screen keyboard provided by the operating system will also be utilized.
5.0 Behavioral Model and Description

The following constitutes a description of how the software transitions through defined states in response to user-initiated events, and the behavior that the software exhibits within these states.

5.1 Description of Software Behavior

These subsections contain list that give the overall states of the system and the events to which the system responds with state changes.

5.1.1 Events

Due to the interactive graphical nature of the software, events are encapsulated within appropriately named buttons. As a result, the following can perhaps be seen as a list of the buttons that have state-change implications. These actions that result in the system changing its state, and therefore behavior, are as follows:

- Create New Game – From the main menu, the user decides to start a fresh playthrough. The state changes to the New Game state.
- Resume Saved Game – From the main menu, the user decides to resume a playthrough from a previous session. The state changes to the Load Game state.
- View Leader Board – The user wishes to view help; the game transitions from the main menu to the leader board screen.
- View Help – The user wishes to view help; the game transitions from the main menu to the help screen.
- Return to Main Menu – This event can occur in the Leader Board, Help, New Game, and Load Game. It results in the state transitioning from the current state to the Main Menu state. The Quit With/Without Saving events have similar implications.
- Input New Character Information – When the user elects to create a new character from the New Game state, the state is changed to the Create Character state.
- Cancel New Character – Cancelling the character creation from the Create Character state transitions the software back to the New Game state.
- Confirm New Character – Confirming the character creation from the Create Character state transitions the software to the map state and gameplay begins.
- Select Game to Resume – Selecting a valid previous game save from the Load Game state causes a state change to the map state and gameplay begins.
- Enter Room – On the map, selecting a valid room has the effect of transitioning the game to the playing state.
- View Map – In the playing state, the user may transition the game back to the map state for navigation through the rooms.
- View In-Game Menu – In the playing state, the user may transition the game to the In-Game Menu state for reviewing information about the story or save/quit functions.
- Read Logbook – While in the In-Game Menu state, the user may cause a change to the logbook state.
- Check Inventory – While in the In-Game Menu state, the user may cause a change to the inventory state.
- Save/Quit – While in the In-Game Menu state, the user may cause a change to the Save/Quit state.
- Quit Without Saving – One option in the Save/Quit state is to exit the gameplay back to the Main Menu state without saving game information from the current session.
- Save and Quit – Another option in the Save/Quit state is to exit the gameplay back to the Main Menu state after saving game information from the current session.
- Save and Continue – A third option in the Save/Quit state is to return to In-Game Menu state after saving game information from the current session.
- Return to In-Game Menu – From the Save/Quit, Inventory, and Logbook states, the user may return to the In-Game Menu state without making any changes to the software. This refers to the cancel option in the Save/Quit state, but the Save and Continue event has similar implications.

5.1.2 States
Distinct modes of system behavior can be observed in the following states:
Main Menu – The game starts in this state from launch. The user is given the option to start a new game, resume a saved game, view the leader board, view gameplay help, or quit. Each button triggers an event which results in a state change to the appropriate state. Quit is not a state; rather, this option terminates the game.

New Game – A new game state is characterized by the option to select existing character information or to input this information differently for the new playthrough. The former option leads to the map state, while the latter must reach this state indirectly by first passing into the character creation state. The previous state is also reachable if the user selects the “return to main menu” button.

Create Character – The user is given prompts for the textual information and graphical information that define his or her character in the story. Such information includes a name and an image file. If the user chooses to confirm the information given, the game moves to the map state. The previous state is also reachable without storing entered information if the user selects the “cancel” button.

Load Game – In the load game state, the software present the user with up to three game sessions persisted from previous plays. If there are less than three or perhaps none, the appropriate buttons will read “empty”. Selecting a valid option causes the system to load the saved information into the game for play to resume at the last point and a transition to the map state. The previous state is also reachable if the user selects the “return to main menu” button.

Leader Board – High scores from players are retrieved from storage and displayed textually for review by the user. The previous state is the only destination state, and it is accessed when the user selects the “return to main menu” button.

Help – The help state displays help text and graphics to the user for instructional purposes. The previous state is the only destination state, and it is accessed when the user selects the “return to main menu” button.

Playing – The majority of the user’s time is spent in this state. All dialog options, character interaction, and other interaction between the user’s character and the encompassing mystery story happen here. Appropriate textual and graphical
display components facilitate these interactions which are handled within the state. Two state changes are possible using the map button or the in-game menu button.

- **Map** – The map state is linked to the playing state because significant changes in the content given to the user in the playing state are not possible without navigation in the map state. Linked rooms are shown for the user to select and enter; this entrance is limited, however, by the current state of the destination room (locked or unlocked). The user selects his or her current room or successfully selects a linked room to transition to the playing state. Unsuccessfully selecting a room due to that room’s state is handled within the map state.

- **In-Game Menu** – The user is given the option to view a logbook, check his or her inventory, save and/or quit, or return to the game. Each of these describes the destination state reached from the event associated with each button.

- **Logbook** – A textual narrative of the user’s exploits within the game are displayed for the user to navigate. Clues received, puzzles solved, and other textual artifacts from the mystery accumulate here as user progresses. The only linked state is the previous state, the in-game menu, which is also the destination state once the user presses the “return to menu” button.

- **Inventory** – A graphical and textual list of items currently held by the user’s character is displayed. As in the logbook state, the only linked state is the previous state, the in-game menu. This is again the destination state once the user presses the “return to menu” button.

- **Save/Quit** – Options for “save and continue”, “save and quit”, and “quit without saving” are presented. Each option constitutes a predictable event and state transition, the destination of which is the main menu state for the latter two and the in-game menu state for the first. The “return to menu” button is also presented to allow the user to return to the previous state without performing any of the previous transitions.
5.2 State Transition Diagrams

Figure 4: Mystery Game State Diagram
6.0 Restrictions, Limitations, and Constraints

Most of the special issues that will impact the specification, design, or implementation of the software are derived through the use of the Android platform. The following list describes these issues.

1. Since the Android is a phone, the main operating system must have the ability to interrupt the game to receive a call. This will impact the method that is used to save the game.
2. The application will need to be digitally signed with a certificate that identifies the developer.
3. The application will have an icon that is used to launch the game. This icon will need to meet the guidelines set forth in the Android developer’s IDE.
4. All logging and debugging options will need to be disabled before the application is built.
5. To allow for a wider compatibility the application may need to add the Android’s support library.
6. The system will be built as a release-ready .apk file.
7.0 Validation Criteria

7.0 Validation Criteria
Certain tests must be passable by the software to ensure that operation is consistent and correct. Any errors must also be handled appropriately.

7.1 Classes of Tests
- Invalid Character Name – This occurs when a user enters a character name that is too long, too short, or contains invalid characters.
- Incorrect XML Format – The XML save data becomes invalid if it does not conform to tag and body usage specifications. This includes incorrect placement of tags, exclusion of required tags, inclusion of incorrect tags, or missing or erroneous data.
- Phone Call Interruption – Phone calls preempt any running games on mobile platforms. No loss of data or placement within the game should occur.
- Text Message Interruption – Text messages preempt any running games on mobile platforms. No loss of data or placement within the game should occur.
- Email/Alert Interruption – Emails, low battery signals, and other alerts preempt any running games on mobile platforms. No loss of data or placement within the game should occur.
- Saving a Fourth Game – Since only three save slots are allowed, saving a fourth game would require overwriting a previous save.
- Incorrect Leader Board/Other File – Similar to saved games, any persistent data such as the user character and the leader board files could become corrupt.

7.2 Expected Software Response
- Invalid user information – The user will be alerted that the information is correct, and prompts will be brought back up for reentry.
- Invalid XML – The game will alert the user that the game save is corrupt. The user will be told to select another saved game.
Phone Call – The game should remain running and allow for resume at the same place as before the interruption. This includes no loss of data despite the process switching to the background.

Text Message - The game should remain running and allow for resume at the same place as before the interruption. This includes no loss of data despite the process switching to the background.

Other Alert - The game should remain running and allow for resume at the same place as before the interruption. This includes no loss of data despite the process switching to the background.

Saving a Fourth Game – At save time, the game should prompt for confirmation due to the need to overwrite a previous game. The user should then be able to select which game to overwrite and successfully create a new save.

Incorrect Leader Board /Other File – The software should tell the user about the problem and elegantly replace the bad file. Problems with user information should be accompanied by the option to replace the bad information.

7.3 Performance Bounds

While no formal bounds are given for performance of the software, all interactive games must respond quickly to user input. As such, the software is expected to have a reasonable response time to all appropriate events so as to not detriment the gameplay.
Works Cited