Seasonal Game

Requirements Specification
Spring 2014

Kekoa Taitt
Samantha Brooks
Shane Perine
Aaron Wilt
Table Of Contents

1.0 Introduction ........................................ 1
1.1 Goals and Objectives .............................. 1
1.2 Statement of Scope ................................. 1
1.3 Software context .................................. 2
1.4 Major Constraints ................................. 2
2.0 Usage Scenario .................................... 2
2.1 User profiles ....................................... 2
2.2 Use-Cases ........................................... 3
3.0 Data Model and Description .................... 4
3.1 Data Description .................................... 4
4.0 Functional Model and Description ............ 6
4.1 Description for Functions ....................... 6
4.2 Software and Human Interface Description .... 9
5.0 Behavioral Model and Description ............ 10
5.1 Description for Software Behavior ............. 10
5.2 State Transition Diagram ........................ 13
6.0 Restrictions, Limitations, and Constraints .... 14
7.0 Validation Criteria ................................ 14
7.1 Classes of Tests .................................... 14
7.2 Expected Software Response ..................... 15
7.3 Performance Bounds ............................... 16
1.0 Introduction

This document has been constructed to explain the requirements for creating a mobile game, dubbed *Seasons*, built for an Android device and created using Java. This document will help guide both the developers and possible clients in seeing through how the game will be designed from the bottom up. One can easily refer back to this document for reminders of how the game should in its final form.

1.1 Goals and Objectives

The purpose is to provide the user with a fun and challenging puzzle-solving game that could be played for five minutes or five hours. The user interface should be intuitive and simple to provide the user with easily-accessible gameplay.

1.2 Statement of Scope

The scope of the application will be limited to Android cellular devices and will be coded in Java. The user can interact with the application by using a button interface created specifically for the game. It has left, right, up, and down movement buttons. It also has a button for switching between various season, accessing the inventory and opening up the pause screen. These buttons are presented to the user using the touch screen of android cellular devices.
1.3 Software context

The application is being created specifically for Android; Therefore, it may be distributed as a product on the Google Play Marketplace or downloaded as third party application.

1.4 Major Constraints

A major constraint will be determining which version of the Android OS to create the game around. The different models of phones that operate with the desired Android OS will have to be taken into consideration as well, since phones have different screen sizes and resolutions. There will also be the issue of making sure to only use functions in Java that will port successfully to the Android OS.

2.0 Usage Scenario

This section is dedicated to providing a usage scenario for the software. It is organized around information determined during requirements elicitation and is presented as user profiles and use-cases.

2.1 User profiles

The user would be defined as a player. The player will interact with the game by using the button pad on the android touch screen made specifically for the game.
2.2 Use-Cases

The basic outline of user interaction is listed below:

1.) User starts the game on their Android phone

2.) The game starts up and the main menu (New Game, Continue, and Options) is displayed.

   2a.) If New Game is selected, a new game begins

   2b.) If Continue is selected, the game loads

   2c.) If Options is selected, the option menu is brought up

3.) Player can quit the game, and the game will save automatically
3.0 Data Model and Description

This section describes the information domain for the software. This also shows how the different components of our software interact with each other.

3.1 Data Description

Data objects that will be managed/manipulated by the software are described in this section. These objects are connected through events and are listed in a class diagram.

3.1.1 Data Objects

- Player - The player is the object being manipulated by the user and will also contain the different functions that allow movement in four directions. This data object will also be able to manipulate the in-game data object.
- Items - The item object is an item that the user can pick up and use to complete the puzzle within the room. They also contain functions that will be used to determine how the player interacts with the in-game objects.
• **In-Game Objects** - These are the objects within a level that can be manipulated by the user to solve the current puzzle. This object also contains functions that allow them to be manipulated by the user.

• **Inventory** - This object is used to store any items the player picks up within a level. This will also contain functions that allow the user to use a specific item from within the user's inventory.

• **Background** - This object is used to display the current level the user is in and contains functions that display where the other data objects will be located within our level.

• **Game Loop** - This data object is the apex of our hierarchy that will be directing and updating the game based on user input. This will initialize all of our data objects and maintain them while the game is being played. This will also be where any updates that are needed will be controlled.
3.1.2 Complete Data Model

4.0 Functional Model and Description

This section is dedicated to explaining the various aspects of a function, design constraints, and various interfaces. These functions are needed to create a playable game that can be continuously updated as the user plays.

4.1 Description for Function
openMenu();

**Description:** It pulls up the user menu.

**Processing Narrative:** When the application initializes, openMenu() displays the menu, allowing the user to select what he/she wishes to do.

**Inputs:** N/A

**Outputs:** N/A

**Performance Issues:** Must not use too much of the system memory.

**Design Constraints:** Must be compatible with the Android Operating System and function properly with various models of Android phones.

saveGame();

**Description:** It saves information regarding a current session of gameplay.

**Processing Narrative:** When called, saveGame() writes all important information regarding the current game session to a text document.

**Inputs:** The important information will be taken in to be stored.

**Outputs:** A text document that contains information regarding information on the game session will be stored.

**Performance Issues:** Save file must not use too much memory because it will be stored on the cell phone.

**Design Constraints:** Must be compatible with the Android Operating System and function properly with various models of Android phones.
createLevel();

Description: A pre-created level will appear after going through a door.

Processing Narrative: Once called createLevel(), the level is displayed on the screen.

Inputs: The integer associated with the level will be taken in.

Outputs: When the level is complete, the level counter will increment and be returned.

Performance Issues: Must not use too much system memory.

Design Constraints: Must be compatible with the Android Operating System and function properly with various models of Android phones.

movement();

Description: It allows the player to move about the level.

Processing Narrative: Once called movement(), the current location integer will be incremented or decremented to the new location integer.

Inputs: The current location integer will be taken in.

Outputs: The new location will be returned.

Performance Issues: Must appear instantaneous and smooth to the user.

Design Constraints: Must be compatible with the Android Operating System and function properly with various models of Android phones.
changeSeason();

**Description:** As the button is pressed, the active season is cycled through the various available seasons.

**Processing Narrative:** When called `changeSeason()`, the next element in the season array will be used to change the season in the game.

**Inputs:** The current season in the cycle will be given.

**Outputs:** The next season in the cycle would be returned.

**Performance Issues:** Must be a smooth transition and not use too much of the system memory.

**Design Constraints:** Must be compatible with the Android Operating System and function properly with various models of Android phones.

loadGame();

**Description:** It loads information regarding a previous session of gameplay.

**Processing Narrative:** When called, `loadGame()` reads the dedicated text file and creates a state containing all required information to start a game similar to the previous session.

**Inputs:** The text file that contains information on the player’s last session.

**Outputs:** Level and items corresponding to the previous game session.

**Performance Issues:** Must not take up too much system memory. Should be called during initialization of the application.
Design Constraints: Must be compatible with the Android Operating System and function properly with various models of Android phones.

4.2 Software and Human Interface Description

The main menu will feature buttons to start a new game, to continue an existing game, to exit, and possibly to view a list of options. By clicking the new game or continue buttons, a list of levels will be displayed for the user to play. Once a level is selected, the user will be able to use buttons on the screen to move up, move down, move left, move right, change the season, use an item, and view the pause menu.

5.0 Behavioral Model and Description

Throughout Seasons, our hero is going to be looking for various items to venture through a garden. He will be able to move left, right, up, and down throughout the level. He will also have the ability to change seasons on command to assist with solving puzzles presented in the various levels.

5.1 Description for Software Behavior

This section consists of events and states of the game. These events and states help define what will be going on in-game and how the user will be able to interact with the environment within the level.
5.1.1 Events

- **Change the Season** - This event will change the map, barriers, and objects on the map to the next season in the sequence. The season sequence will be spring, summer, fall, and winter. The user can cycle through these at his or her discretion. No movement will be allowed during season changes.

- **Move Up** - This event will move the player up one block. If a moveable block is in the way and there is no boundary directly past the moveable block, it will also move one block up. A non-moveable block directly above the player will keep the user in a still state.

- **Move Down** - This event will move the player down one block. If a moveable block is in the way and there is no boundary directly past the moveable block, it will also move one block down. A non-moveable block directly below the player will keep the user in a still state.

- **Move Left** - This event will move the player left one block. If a moveable block is in the way and there is no boundary directly past the moveable block, it will also move one block left. A non-moveable block directly to the left of the player will keep the user in a still state.
- **Move Right** - This event will move the player right one block. If a moveable block is in the way and there is no boundary directly past the moveable block, it will also move one block right. A non-moveable block directly to the right the player will keep the user in a still state.

- **Use Item** - This event will allow the player to affect the blocks directly bordering the character's current space depending on which item is being held.

5.1.2 States

- **Still** - This state is the initial and resting state. The user can change the weather, use an item, or view the pause menu from this state.

- **Move Up** - In this state, the user’s character will move toward the top of the map by one space, barring that there are no barriers in the way. If there is a movable object in the way, the object is moved toward the top of the map by one space as well. If there is a non-moveable barrier in the way, the user’s character remains in the still state.
- **Move Down** - In this state, the user’s character will move toward the bottom of the map by one space, barring that there are no barriers in the way. If there is a movable object in the way, the object is moved toward the bottom of the map by one space as well. If there is a non-moveable barrier in the way, the user’s character remains in the still state.

- **Move Left** - In this state, the user’s character will move toward the left side of the map by one space, barring that there are no barriers in the way. If there is a movable object in the way, the object is moved toward the left side of the map by one space as well. If there is a non-moveable barrier in the way, the user’s character remains in the still state.

- **Move Right** - In this state, the user’s character will move toward the right side of the map by one space, barring that there are no barriers in the way. If there is a movable object in the way, the object is moved toward the right side of the map by one space as well. If there is a non-moveable barrier in the way, the user’s character remains in the still state.
6.0 Restrictions, Limitations, and Constraints

Accessibility: The game will be accessible by displaying a how-to introduction at the start of a new game.

Hardware Diversity: Many android phones do not have buttons implemented. The game will have touchscreen options for those without buttons.
7.0 Validation Criteria

The game needs a way of testing whether or not the game is satisfactory for the average player and to weed out any glitches. Here are our methods of testing.

7.1 Classes of Tests

7.1.1 Market Game Testing

People who commonly play android or other mobile-based games will be asked to try a brief demo of the game. They will be asked what they liked, didn’t like, and other suggestions they may have for the developers. This will be the most important part of testing, as this will clue the developers in on what people want out of a mobile game.

7.2 Expected Software Response

These responses will be used to further smooth the game out as well as reinforce positive aspects of “Seasons”.

7.2.1 Likes

The testers will give a very short description of what they liked in the game. Testers will most likely not be able to pinpoint exactly what it is that they liked, but from what they are able to say, it is possible to come up with inferences that can help with the design.
7.2.2 Dislikes

People are able to more easily pinpoint what they dislike than what they like, so this is where the meat of the testing comes in. This will tell the developers where they may want to tweak some parts of the game in order to appeal to more players.

7.2.3 Suggestions

Each person will want something different from the game, so it is important to take in all that the development team can from player suggestions. They may have a really neat idea that the team never even thought of but are able to easily implement. This is not limited only to game mechanics, but can be spread out to aesthetics as well.

7.2.4 Glitches

No program is ever perfect, so testers are bound to find glitches with the game. It is important to find these issues and address them as quickly as possible so that they are not leaked into the full release.

7.3 Performance Bounds

The basic demo will not use graphics that are too taxing in development, and the demo will have only a few rooms to it, allowing for the tester to see different rooms that require multiple seasons to progress.