Duquesne University

‘Lost in the Woods’ Programmer’s Guide

TMT: The Money Team

Devin Dougherty, Lukas Shiley, Jared Konop, Guriqbal Singh, Raj Patel, Patrick Diem

COSC 445W - Software Engineering

Dr. Fadi Weydan

30 April 2021

Section 1.

The program was written in Java and uses the packages 'Swing' and 'AWT' for the GUI and graphical elements to make up the user interface. GitHub is the system that was used to collaborate throughout the developmental process of this program. This software is intended to run on Microsoft Windows x64 bit operating systems.

Section 2. *(Data-Flow Diagram)*

Diagram 1.1: The program begins with an Integer prompt. Depending on the value of the integer input, the program will continue with the appropriate game level. If the input has one of the values; 6, 7, or 8, Game 6-8 will be used and If the input has one of the values; 3, 4, or 5, Game 3-5 will be used. Otherwise, If the input has one of the values; K, 1, or 2, Game K-2 will be used. Depending on the level that has been chosen by the user, the user will receive the appropriate amount of customization abilities.

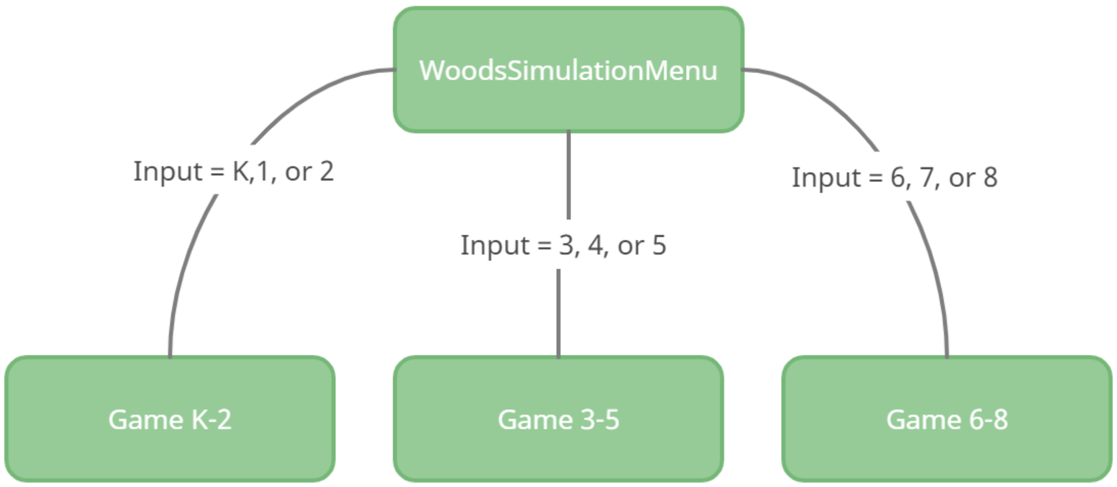


Diagram 1.2.1: (Game K-2) Following the input of the grade level K, 1, or 2, the program will proceed to create the K-2 window. The unique aspect of the K-2 difficulty is that the game's grid is restricted to a square, meaning both the length and width of the grid are equal. Another limitation of the difficulty is the maximum number of people that can be placed in the forest, which is two. GameK2 is an extension of the Game class. The game class includes functions related to the initialization of player characters, starting the game, ending the game, and updating the grid after an action is performed.

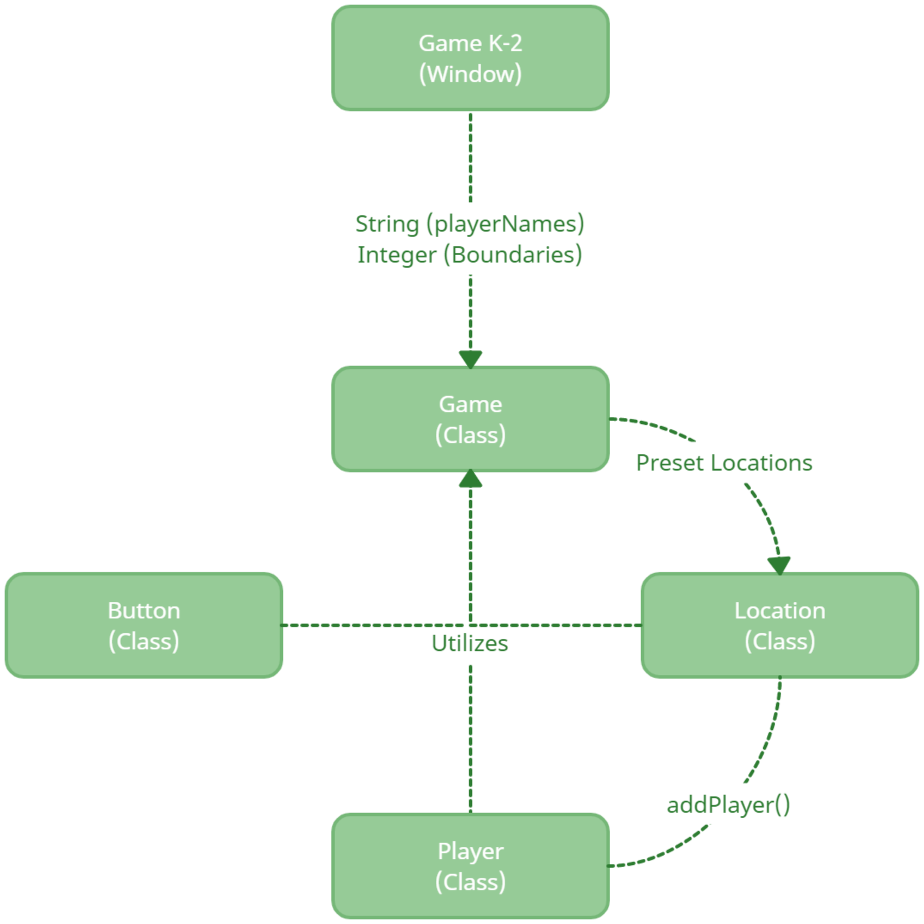


Diagram 1.2.2: (Game 3-5) Following the input of the grade level 3, 4, or 5, the program will proceed to create the 3-5 window. With this difficulty selected, the users can define the bounds of the grid on both the X and Y axis, permitting non-square grids. Additionally, the maximum number of players is raised to 4. These players can also be placed on the grid at the user's discretion. Game35 is an extension of the Game class. The game class includes functions related to the initialization of player characters, starting the game, ending the game, and updating the grid after an action is performed.

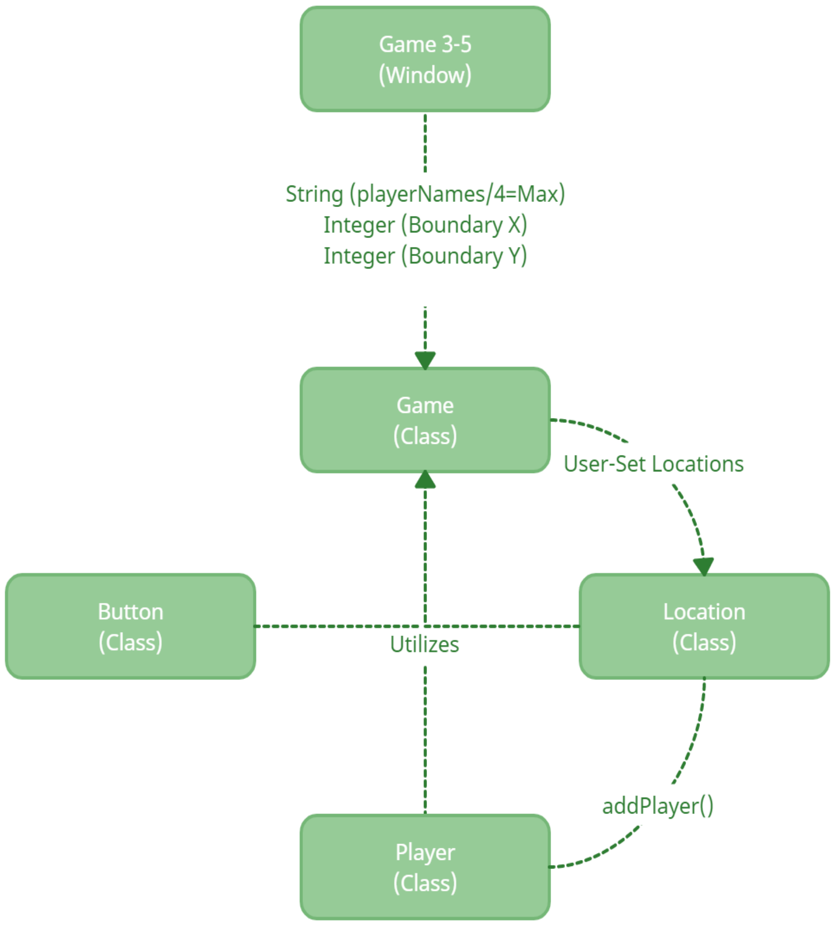
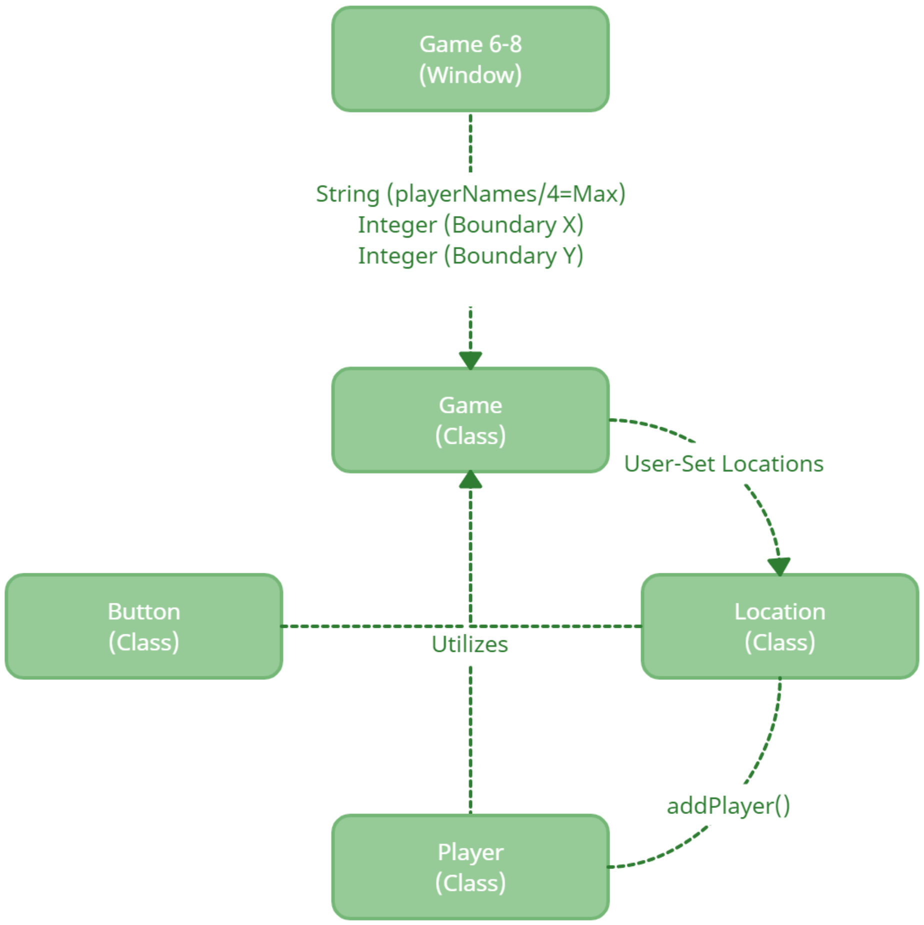


Diagram 1.2.3: (Game 6-8) Following the input of the grade level 6, 7, or 8, the program will proceed to create the 6-8 window. The restrictions of Game68 are the same as that of Game35, however, three unique movement behaviors can be chosen between for the players. The movement of the players can be random, but the player can remain stationary, or the players will remain together after finding each other, assuming there are more than two players currently on the grid. Game68 is an extension of the Game class. The game class includes functions related to the initialization of player characters, starting the game, ending the game, and updating the grid after an action is performed. Movement - Random, Random with Stationary, and Staying Together after Finding Each other.



Section 3. MDD Diagram

3.1 - *External Libraries*

The external libraries utilized for this project include both Java Swing and AWT, also known as Abstract Window Toolkit. Java Swing is an extension upon AWT, which adds additional functionality to the existing API. The additional features include but are not limited to platform-independent components, much more lightweight components, support for pluggable look and feel, and a much wider array of components to utilize. AWT on the other hand provides the foundation for the components that are utilized in the program. AWT Components can be found throughout this program. A non-exhaustive list of these components contains buttons, labels, containers, windows, sliders, frames, and panels. An extension of the JButton object is included in this program to quickly store relevant X and Y coordinate data.

3.2 - *Front-End Classes*

The driver class for the program is *WoodsSimulationMenu*. The responsibility of the driver class is the creation of the initial window as well as the creation of relevant UI elements. The UI elements can receive input regarding the grade level, the size of the grid on both axes, the number of players, and the behavior of said players. The driver class is also responsible for launching the game windows for the various difficulty levels.

The *GameK2* class is responsible for the creation of the game window for the K-12 difficulty. The restrictions of the K-12 difficulty include the grid limitation to a square shape (X = Y). The player count is also limited to only two as well as the spawn location for each player being pre-set to opposite corners of the grid. The players will proceed to select a random direction to move each turn. Upon locating the other player, the game will conclude, and the user will be presented with an end game splash screen as well as statistics.

The *Game35* class is responsible for the creation of the game window for the 3-5 difficulty. The restrictions of the 3-5 difficulty are laxer. In this difficulty the user can define the length and width of the grid, allowing the shape of the grid to go beyond a simple square shape. The maximum player count has been expanded to support up to 4 players, and the program permits the user to choose the starting location for each player. Again, the player characters will move about randomly until all 4 players have located each other. When this happens, the user will be presented with an end-game splash screen as well as relevant statistics.

The *Game68* class is responsible for the creation of the game window for the 6-8 difficulty. The restrictions are the same as the 3-5 difficulty, except the user now can choose between multiple different wandering behaviors. These behaviors are as follows:

Random - The players will wander randomly until all players have located each other.

Random with Stationary - The players will wander randomly until all players have located each other, but also have the option to remain stationary as opposed to being forced to move each turn.

Stay Together - The players will wander randomly until they have located each other, if two players have found each other, they will remain together until the rest of the players have been located. This is only applicable to games with over 2 players.

3.3 - *Back-End Classes*

The back end of the program includes classes created for various functions. At the core is the Game class, which contains methods that will be utilized in every game window. These functions are:

initializePlayers(String[] playerNames) - Responsible for the creation of all player characters as requested by the user.

startGame() - Begins the game update loop, which is responsible for looping through each turn in the game until the game has ended.

gameUpdate() - Updates the game by selecting a random move location for each player.

setGameSpeed() - Sets the speed at which the game will update.

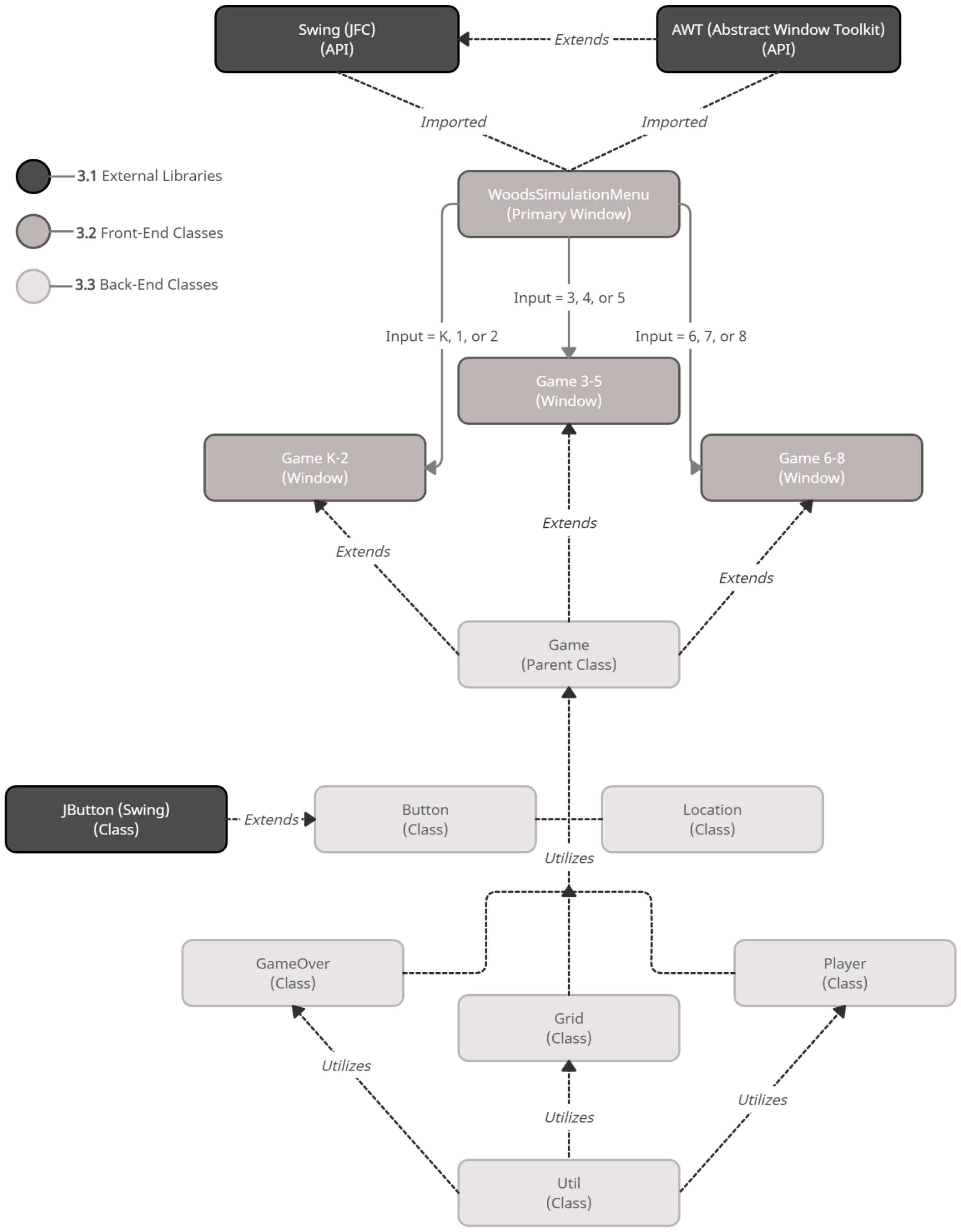
updateGrid() - Responsible for updating the visuals presented to the user regarding the grid.

checkForWin() - Has no actual functionality in the Game class, as the conditions for a win are different in each subclass. Exists solely to be overwritten by the subclass.

playerPlaced(int x, int y) - Has no defined functionality in the Game class. Called when a player is placed on the grid.

Each difficulty is a subclass of the game class. Each inherits the methods listed above.

Stemming from the game class, the Location class is responsible for tracking how many players are currently located at a singular grid location. The Grid class is responsible for managing data displayed to the user. This class heavily utilizes the Java swing library. The Player class handles statistical data regarding the performance of a player, such as how long it took to find another, etc. The player class is also responsible for handling where the player decides to move, moving the player, and setting the player location. Finally, the Util class is responsible for handling the creation of image icons from resources as well as scaling said image icons.



Section 4.

Installation Instructions: To run the WoodsSimulationMenu.jar file you will need to open the command prompt. Once the command prompt is opened then you will need to go to the directory the WoodsSimulationMenu.jar file is saved in.

Method 1: Simply double-click WoodsSimulationMenu.jar to launch the program.

Method 2: First, press the Win key and type 'cmd,' press enter to launch the Command Prompt.

Second, change the directory to the same location as the jar file. For example, if the project was saved to the desktop, you type the following command followed by the return key:

```
cd desktop
```

Finally, to run the WoodsSimulationMenu file you will need to type the following command followed by the return key:

```
java -jar WoodsSimulationMenu.jar
```

The 'Lost in the Woods' simulation menu should execute and await input from the user.