Mr Jack Game Design Report and User Guide

Mac Clevinger and Sarah MacEwan

COMP 3721

November 5, 2018

Mr Jack Game Design Report

Mr. Jack is a two-player game set in the London in the 1800s in which one player (the Detective) attempts to find out which Character the other player (Mr. Jack) is impersonating before they can escape the city. Our implementation of the game was a scaled-down version of the game that allowed for two turns of play, with four possible playable characters, and a smaller gameboard.

**Development and Refactoring**

The underlying architecture of this project has changed a lot from the first attempt at a UML diagram in Milestone 1 to the final implementation in Milestone 3.

When we began outlining code for our implementation, we discovered some fundamental flaws in our original design that we had not accounted for. These issues mainly centred around decisions regarding what classes/objects would know what information. In particular, our previous use case and UML diagrams had not specified clearly whether the Characters and Tiles knew their locations on the gameboard, or if the Board or the GameModel knew that information.

In the original UML class diagram for our GameModel class, we had a method called rules() that we said would “check all the rules” and make sure none of the objects in the game were in an illegal state. This was poor design, and the rules() method ended up getting subdivided into a host of methods that queried different objects and their states in order to ensure that the rules were being followed during gameplay.

Another big change that we made was to make our Character class and our Tile class abstract (in our implementation, we called it MrJackCharacter to avoid overwriting the Java type Character). We made this choice because we wanted all the different kinds of character and tile objects to have similar default behaviours, but with a few special attributes or behaviours to account for each character’s ability and for the different kinds of tiles (buildings, lanterns, manholes, and exits). This seemed best done with an abstract class where we could have the implementations of the base behaviours, and then override certain methods to account for the specializations of each kind of character/tile.

We decided to implement the characters of Miss Stealthy, John Smith, Sir William Gull, and Inspector Lestrade for our version of the game. We wanted to use Miss Stealthy to demonstrate a flexibility in our code that could accounting for characters that move in unexpected ways (ie through squares that should not be legal to move through). Because of the importance of lamps in determining whether Mr. Jack is “seen” or “unseen” during gameplay, we decided to implement John Smith, who can light/unlight lamps around the city. This would ensure that our code could account for different sections of the gameboard being lit or unlit at different times, and help ensure that our win conditions for Mr. Jack were implemented correctly. We chose to implement Sir William Gull, who can swap places with any other character on the board instead of moving because of the challenges this presented for ensuring that we can get character locations on the gameboard and make swaps of character identity/location on the board. Finally, we decided to implement Inspector Lestrade because he can block or unblock an exit tile, preventing Mr. Jack from escaping the board. This blocking/unblocking added complexity to how the game could be won, so it was a good check for ensuring that our win-conditions were properly implemented.

For the GameView, Mac found that she had already implemented a library for a game UI from another course, so we co-opted some of that code to make nice visuals for our version of the Mr. Jack game.

**MVC Architecture**

The main principle behind the model-view-controller (MVC) architecture is to separate the model, or main computation aspects of your program, from the view or user interface (UI). This allows the underlying program to be used with any kind of implementation for a view. In order to ensure this separability, a controller is used as a go-between for the model and view. The model talks to the controller and the view talks to the controller, but the model and view never talk to each other directly. In our implementation, we had three separate classes (GameModel, GameView, GameController) to ensure that the project followed the MVC architecture.

**GameModel:** The GameModel class gathered all the information about the various objects required for our implementation of the game. It made sure the objects’ states were updated according to incoming information from the controller, and that the incoming information adhered to the rules of Mr. Jack. If a rule was violated, a message was sent back to the controller to inform the user to try a different move.

**GameController:** The GameController acted as a go-between for the GameModel and GameView classes. It took the information about the game’s state that it received from the GameModel and translated it into a format that the GameView could understand. It also interpreted information about user actions from the GameView and passed it along to the GameModel to verify if those moves were legal, and if they were, to update the game’s state.

**GameView** The GameView takes in input from the user and displays the effects of those inputs with updates to its visual layout. We decided to implement our view class using awt. User inputs are recorded by various action event listeners, which pass that information on the be interpreted by the GameController. Once the action has been passed from the GameController to the GameModel, and the effects of that action have been calculated, the GameController informs the view to update to whatever new state the game should be in after that action.

**Mr Jack User Guide**

Start a game:

Click “START GAME” on the startup screen.

This will take you to the main game screen, with a board layout and initial starting positions for the characters, and a layout of lamps, manholes, and buildings.

Pick Player:

Whoever is playing as Mr. Jack will be told who they are impersonating (if two players are playing on one computer at this point, the Detective should not look at the screen or they will find out who Mr. Jack is).

Turns:

Choosing a character: Depending on whether it is an even or an odd-numbered turn, either Mr. Jack or the Detective will get to pick the first character of the turn. To pick a character, the player selects the character they want to move from one of the four cars on the sidebar.

Making a move: To move a selected character, the player clicks on the tile they want to move the character to and then on the MOVE button. If it is a legal move, the character’s marker will be moved to that tile. If it is not a legal move, they will be asked to pick a legal move.

Using a special ability: Some character’s special abilities must be used before the character can be moved, some can be used before or after moving, and some can happen during or instead of movement. Once a character has been selected, the possible options for use of the special ability will be made available to the player by clicking on the ABILITY button.

End a game:

By winning: When either Mr. Jack or the Detective has met the win conditions for their role, the game will end and a message informing the users as to who won the game will be displayed.

By force-quitting: To force-quit a game, a user can click the X in the top corner of the view to shut the program. Alternately, they can choose to re-start the game from the MENU button in the bottom corner of the view.