SWEN225 Assignment 2 - Cluedo GUI

Vaibhav Ekambaram 300472561

User Interface

The user interface created for this assignment was made using Java Swing, focused around the GUI class. When the game is run, an instance of this class is created. The update display method keeps track of the game state then displays the appropriate elements. Elements are added to panels (such as the actionPanel or handPanel), which are added to the window frame through the main JPanel which contains all other panels. Elements are initialised and added to panels in the GUI constructor, then their visibility is updated using setter methods when needed. The constructor also contains the listeners for buttons, mouse behaviour, and keypresses.

The interface was designed to behave in a similar way to past assignments, with an action panel on the left of the screen, next to the main panel displaying the board, with characters and weapons displayed. These action panel buttons execute various methods, through their associated action listeners. The main game panel was constructed out of the previously existing board class, which contained a two-dimensional array, holding all the positions on the board. When adding a GUI to the program, all that was needed was to associate each position with a rectangle drawn in the window. After this, the game players are similarly drawn on the screen. Weapons are drawn differently, using an ImageIcon which loads from file when the weapon cards are constructed. Another panel is positioned at the bottom of the screen displaying the cards in the current player's hand. This is implemented using a swing panel with a grid layout using a fixed value of rows and an adaptive value (0) of columns. This allows for the panel to resize based on the hand size without cards clipping off the end of the screen. A small addition to this feature was the ability to hover over a player card for more information. If a user positions their mouse over the card, a popup will appear, showing the card type.

To help improve code simplicity, the code for the popup menus for player setup, suggestions and accusation functionality has been separated from the main GUI class. These popup windows make use of swing option panes, allowing for a popup, independent of the main window. For these windows, elements such as radio buttons, combo boxes and text fields were added to a panel which is then sent to the optionPane instance.

Keyboard shortcuts for key functions were an extra addition, as an alternative to pressing buttons to transition state or to move player pieces around the board. For example, rather than pressing the "Roll Dice" button then using the mouse to move their piece, a user may press the 'r' key then use their keyboard arrow keys to move their player.

Game Design

The majority of the code for the original command-line based program has been retained. However several changes were required to accommodate a graphical user interface.

The first distinct change was the depreciation of scanners to check for inputs. Due to the previous text-based limitation, significant portions of the scanner code was dedicated to validating user input. In the new input based code, these scanners were able to be easily substituted with OptionPane dialogue boxes called from the view packages. This was made even simpler by separating model and view classes. Due to the methods built into the ComboBox, RadioButton and TextFields, input validation was significantly simplified. For example, when asking for the number of game players, rather than checking if the input was a single number then checking the number was between three to six, the number could be extracted as one of three options in a combo box.

One slight problem occurred when replacing scanners with the new OptionPane was the possibility of a user being able to confirm an option pane, without entering an input. This was particularly the case for the player token setup screen, where a user must select a token before continuing. If a token was not selected and the options pane was closed, this could lead to significant problems with the game setup. This was relatively simple to resolve by decrementing the number of players looped through in the player setup.

A significant issue occurred when implementing the GUI with the board constantly flickering. This occurred due to the screen constantly updating with the weapon image icons and wall lines unable to be redrawn at the same speed. This proved to be an extensive problem which required changing the handling of the main game loop. In the original text-based iteration of the program, the main game loop was accomplished by using a while loop, which kept running until the game was won. However, in addition to the flickering issue, setting the visibility of different GUI elements using this loop appeared to be inefficient. This issue was resolved by replacing the main game loop with a state-based system, where elements are updated depending on the state. The board view was then updated on these state changes, or when a user moved their player token on the board.

States

The game uses a state-based system to process the main game loop. This consists of a main and a substate. The main state is split into three sections, delineating the game setup, running, and a game-over state. The substate then governs the running state, which is split into stages for player movement and action. Movement is the period in which a user may roll a dice and move. Therefore, this stage terminates once the current player has used their generated number of moves, or manually overridden using the “Finished” button. From here, the substate transitions to the action stage, where a player can make suggestions and accusations or passed. After these have concluded the substate will then return to the movement state. This is unless a successful accusation has been made, wherein the game will then enter the finished state, where the game will cease to run.