SWEN225 – Assignment 2 Reflection
greenthom – 300536064

GUI Discussion

All classes with their Swing components:

- Board class:
  - JPanel[][]: Represents individual cells on the game board.
  - BorderFactory: Enhances visual distinction by creating borders around panels.

- CharacterSelectionScreen class:
  - JPanel: Creates sections for character selection, with each player's selection presented within a panel.
  - JButton: Initiates the game once all players have been selected.
  - JRadioButton: Allows players to choose characters. A selected character becomes unclickable for other players.
  - BorderFactory: Used for titled borders around player panels.
  - ActionListener: Listens to radio button clicks.
  - GridLayout: Organizes player panels in a grid format.

- GameGUI class:
  - JFrame: The main game window that holds various panels.
  - JPanel: Represents different game screens or stages.
  - JMenuBar: Provides options for starting a new game or exiting.
  - JOptionPane: Displays confirmation dialogs for starting a new game or exiting.

- GamePanel class:
  - JFrame: Used to create multiple windows, including winFrame, incorrectFrame, gameLostFrame, solveDialog, and refutationDialog.
  - JDialog: Smaller window for prompts during interactions.
  - JPanel: Holds buttons in the button frame.
  - JButton: Creates clickable buttons for various actions.
  - JLabel: Displays player moves information.
  - JTextArea: Used for the information window to display and organize information.
  - JScrollPane: Enables scrolling within the information window.
  - JComboBox: Provides game cards for player guessing and solve attempts.
  - JPanel[][]: Allows access to specific rows and columns on the board, mainly for player refuting.

- StartingScreen class:
  - JPanel: Presents the starting screen in the window.
  - JLabel: Displays "Welcome to Hobby Detectives" at the top.
  - JButton: Creates a button to start the game.

Strategic Placement and Purpose:

- JPanels used to organize sections of game, while JFrames and JDialogs were utilized for game interactions and information display.
- JButtons placed within JPanel and JDialog: perform actions: moving on the board, making guesses, and solving attempts.
- JLabels were used to display important game information, such as player moves, welcome messages, guess and solve attempt.
- JTextArea and JScrollPane used together: organize/display extensive information to player.
- JComboBox was integrated for player guesses/solve attempts, providing selection mechanism.
- JPanel[][] facilitated access to specific board positions and cells.

Justification:

- Chosen components match their respective roles: JPanels for layout, JButtons for actions, JLabels for information, JTextArea for extensive text display, and JComboBox for selection.
- JDialogs were preferred for smaller interactions to avoid cluttering main game screen.
- JScrollPane and JTextArea ensured that large amounts of information could be displayed effectively.
- JComboBox was suitable for providing a selection mechanism for player guesses and solve attempts.

Design Discussion

- Transitioning from a textual output to GUI posed several challenges. One challenge was managing player movement in and out of estates. The original code's handling of player removal upon entering estates and exit from either door proved complicated to adapt in the GUI framework. We resolved this by aligning estate exits with the door through which players entered.

- In the original code if a character was inside of an estate, it would not print them there, but just give them a prompt stating they are in that estate. In the GUI framework, we incorporated the player being presented in the estate.

- Another challenge was the refutation process. When a player's character was guessed it would teleport the player into that estate the guessing player is in. The main challenge was when the teleported player exited the estate he was now teleported in, making sure he was then exiting that estate he was in previously, since we had assigned a specific door for a player once an estate was entered. We solved this through when a player is transported, we assign that player the same exit door as the guessing player.

- Implementing card and other game functionality from a text-based UI to a GUI was relatively straightforward, as was integrating player-related features.

- The adoption of the Model-View-Controller architecture facilitated the integration of GUI into existing code. By separating game mechanics and presentation, we established a well-organized structure. This allowed us to focus on GUI presentation without disrupting core mechanics. Modifying GUI elements had minimal impact on underlying logic, enhancing maintainability and facilitating interface adjustments while coding. The MVC framework promoted clear communication between different components, streamlining development across team members' areas.

- MVC also positions us well for future code enhancements, ensuring that evolving the codebase will be efficient without compromising the existing GUI.

- Transitioning from textual UI to GUI involved addressing challenges related to movement mechanics, estate interactions, and maintaining consistency across different components. Leveraging MVC architecture proved vital in seamlessly integrating the GUI, supporting a well-structured and adaptable codebase.

State Diagram Discussion

In our state diagram, we've mapped out the software's progression through states, showing the key stages of user interaction and gameplay:

- The initial state is the starting screen; user initiates the game by pressing the "Start Game" button.
- Transitions to the character selection state; users' select their characters within this state.
- Transitions to main game state; featuring the presentation of the board and players:
    - Player to move.
    - Player receiving number of moves transitioning to "Player waiting to move."
    - Player moving inside an estate and not attempting solve, transitions too "Can guess, Can solve".
    - Player not inside an estate transitions to "Can solve"
    - (Can Guess, Can Solve) Player guessed transitions to "Player to move."
    - (Can Guess, Can Solve/Can Solve) Player solved incorrectly removes player from arrayList and transitions to "Player to move"
- All players incorrectly solve transitions to losing panel.
- A correct solve attempt results in a transition to the winning panel.

This state diagram represents software behaviour across user interactions and gameplay. It aligns states with user actions. The state diagram serves as a guide for implementing the GUI's functionality, ensuring a user-friendly interface.