Overview

We were able to recycle much of the code from Assignment 1, as we designed that to make the transistion from Text UI to Graphical UI as simple as possible. This meant that we just needed to re-implement our UI interface to use a GUI instead of the console, which meant several methods needed minimal changes. One issue we faced was that the old interface did not allow for event driven interaction, so we had to change some of the core Cludeo code to make the GUI work. This meant that our TextUI interface no longer works, which was disappointing but necessary.

Ben and I used a GIT repository hosted on GitHub to work on the assignment together, as we were both confident with using Git as opposed to SVN from the previous assignment. Our repository can be viewed at https://github.com/bagedevimo/SWEN222-Cluedo.

GUI

The GUI has two main parts to it, the display of the board and everything on it, and the control panel. This was achieved by having two objects added to the main JFrame, a canvas for the board and a JPanel for the control panel. It also has a JMenu above the board, from which the notebook and cards of the player can be viewed, and the game can also be exited from this menu.

The control panel uses a GridBagLayout as its LayoutManager, which allowed for more flexibility in positioning the components than a BorderLayout. I do not completely understand how this layout works, as Ben was the one who took care the addition of components to the control panel. The panel has three main sections, the display of the dice on the right, the display of the information for players (in particular, the player who's turn it is currently) in the center, and the buttons to open the information windows on the left.

The Cards and Checklist buttons both create a new JFrame when clicked. Depending on which button is pushed, several JLabels are added to this JFrame. For the cards window, these JLabels each represent one of the cards the player was dealt at the start of the game. For the checklist window each JLabel corresponds to one of the cards the player has seen during the course of the game. Initially these are all of the cards the player was dealt, but as the game progresses and the player has some Guesses proven false, various cards are added to this list.

When the game is started, several dialog boxes are shown. The first is a welcome message, the second asks the user how many human players the game should have, and the rest ask each player which character they would like to control for the game. Players are not able to select characters that have been chosen by other players already.

At several other points during the game, dialog boxes are shown to gather information from the player. The most notable of these is when a player is making a guess. Two dialog boxes are show, the first has a list of all the characters in the game and asks the user to choose which one they think was involved in the murder, and the second has a list of weapons for the user to choose the murder weapon from. Ideally, these two dialog boxes should have been combined into one dialog box, with two drop down menus. We couldn't work out how to do this using the default Swing dialog boxes, so would have had to make our own.

Dialog message boxes were also used to display messages to the user as they played the game, such as which card they have been shown after their guess (if any), and to display if an accusation made was correct or not. These did not take any input from the user, just simply displayed a message.

The board

To draw the board, we used a canvas object. Firstly, the board image was drawn at position (0,0), and then all of the token on the board were drawn on top of the board image. The positioning of these tokens was determined by a grid overlay implemented, which lined up with the positioning of the tiles on the board image. This was less than elegant, as each board tile had a width of 16.5 pixels, resulting in some gaps between our grid tiles due to rounding. The origin of the grid was not at (0,0), but at about (14,14) to account for the border on the board and the wider first row and column on the board.

When a user clicks on the board, the first thing that occurs is that the grid position that they click is determined, by using the grid overlay we made. Secondly, we worked out if the player was able to move their piece their within the allowed number of turns. We used a path-find algorithm similar to Dijkstra's for this, and the available squares were shown by adding a transparent green color to the tile. We displayed rooms that were able to be visited by painting them with a transparent blue to differentiate them from hallway tiles. A known issue we have is that movement with the squares in a room only costs nothing when entering the room. When a player goes to exit the room, they have to use some of their allowed movements to move from their position inside the room to the door, rather than exiting from the door and using all of their movements there.

If the player is allowed to move to the position they selected, their token is moved there, and if they are in a room, they can make a guess. If they are still in a hallway, their turn ends and the next players turn starts. If the player cannot reach the tile they selected, nothing happens, and the user has to select a tile that they can reach.