SWEN225 Assignment 2 Reflection

For my reflection, I will be splitting the analysis into three parts, as per the assignment brief: GUI discussion, design discussion, and discussion of the state diagram.

GUI Discussion

To implement a GUI for Cluedo, my group decided to go with the recommendation provided by the assignment brief at first. We designed the model initially using a whiteboard and drew up the example provided. We then went through each of the requirements, in turn, discussing how we could implement them in our program to make sure that we hit on all the required points. We decided to split our GUI components into JPanels to make it easier to manage and then split these panels down further, giving the GUI a hierarchy to it. We later changed to Canvas objects but still kept the hierarchy associated with the JPanels.
We discussed at length about what the board would look like, going back and forth between the rooms being detailed and copied from pictures, or whether they should just be vector graphics on the screen. The latter is what we chose, due to it being easier to implement on the board array. We did pick nice sprites for the weapons and character objects, and for the cards that were displayed too. We also decided to colour the text of the buttons grey when they were useless, for instance, the suggestion button when in a hallway.
In the discussion, we also had to take into account the feature we wanted to include in order to tick off the extension part of the assignment. We decided that implementing shortcut keys would be the easiest, but by the end of the assignment we ended up doing 3 of the extensions. At first, we had the cards being displayed at the bottom of the screen, but I realised that most screens are landscape, so we shifted that stuff to the side to make the board bigger in the window.

Design Discussion

When modifying the program to suit a graphic based output as opposed to a textual input/output, we discovered several challenges that we had to deal with. For me, this was especially hard, since I built virtually all of the output for the text-based version of the assignment. This meant that I was less familiar than everyone else with the rest of the code since my contributions to the first assignment were essentially mostly overwritten in this assignment.
Another issue we all faced was that no one had had any prior experience with swing, which we needed to use to create the GUI. This was a relatively steep learning curve for me, as there are lots of objects that you need to use to build a fully functioning GUI, all of which I was unfamiliar with prior to starting this. In our original code, I built the output to be relatively separate from the rest of the project, which made it easier to build the GUI. I think that this is good coding, as it provides a separation between output and logic. This is something we got to explore more in this assignment, as we were introduced to the Model View Controller (MVC) design style. I must admit that we wrote most of the code in our GUI class initially, and once it was all working we then implemented the model view controller

approach, which in hindsight we should have started out doing. Also, I believe that if we had taken a similar model view controller approach in assignment one, then the transition from text-based to GUI could have been a lot easier
This was another big challenge we faced, extracting the model, view, and controller code from our big GUI class. This was made more difficult by the fact that it is quite hard for multiple to work on this at the same time, so one person had to do it all. The final challenge was that the actual logic code had to be reconfigured to accept a different input than before, as now input came from radio buttons as opposed to text input, but this wasn't that hard to do.

Discussion of State Diagram

Our program has various states which it can be in, which we discussed as a group. There are a total of nine states in our program, of which six are inside a superstate called game running. The three states outside this are 'Game Over', 'Game Ready', and 'Game Initialise'. Inside the superstate 'Game Running' exists the following states: 'Before Move', 'After Move', 'Suggestion', Accusation', and 'Next Player'. It is worth noting that the movement has been split into two states to better represent what is happening in our code.
Next, we have transition guards. There are too many to go through in any detail in this report, but I will summarise it. Lots of the guards are dependent on user input, like clicking a button to transfer between states, or when the player submits a suggestion by clicking on the radio buttons. The guards also ensure that certain actions are inaccessible to players depending on where they are and what they are doing, for instance, to transition to the 'Suggestion' state the variable 'currentPlayerCanMakeSuggestion' must be true.
Furthermore, we also have transition actions, which are things that need to happen when the game makes a transition between one state and another. For instance, to move from 'Next Player' to 'Before Move' the player must start their turn.
Finally, the 'Game Running' superstate allows for the program to ensure that when the game is not running, no game logic can run, that is the game cannot be played.

Conclusion

In conclusion, I was very satisfied with my team's work, and think my group produced a very good version of Cluedo. I think we worked well together and am pleased with the final product. I believe that we worked much more efficiently this time around as well.