Yun Zhou

GUI Discussion

For our GUI Window, we did not follow the suggested layout and the following is the description of our GUI design.

- The Game board is drawn by using the JPanel. It's in the middle and consist of different type of cells and the corresponding colour. The player recognizes their character through the small circle with single Char like 'S' for Miss Scarlett.
- At the top of our GUI, except the required JMenuBar, we construct several JButtons for players to click on it, like move, suggest, show shortcut keys etc. The reason we add this is these buttons are helpful for players to play. Also, shortcut keys like ALT+W are implemented.
- The suggested footer layout is moved to the Right-hand-side of the window. JSplitPane and JTextArea are used for showing the message as the broadcast. Players can get everything they need from it. For instance, players know the number of movements left, hand cards and who use what refute cards to refute you etc.
- JOptionPane is used for showing the dialogue when it's needed. For example, ask players to get their details, choose the card to refute, broadcast the important message.

There are several disadvantages of our GUI.

- The message showing in the JSplitPane is too messy can be separate out. For example, hand cards can be separated out.
- JButtons can be more tidy and concise. For instance, during the movement state, make accusation/ suggestion button are useless; during the end-turn state, movement buttons are useless and can be subtracted. Implement this point can not only concise the GUI, but also ruled out the unnecessary confusion.

Design Discussion

Due to the MVC design was not taught before, our original design was very bad and it didn't separate the responsibility, thus, our main classes were too long and bloated. We found that it's hard to maintain and add the GUI because our original MVC was always intertwined and came in pairs, *the change of one part of the code is the change of the whole part.* That means, when the data need to update, need to do lots of work.

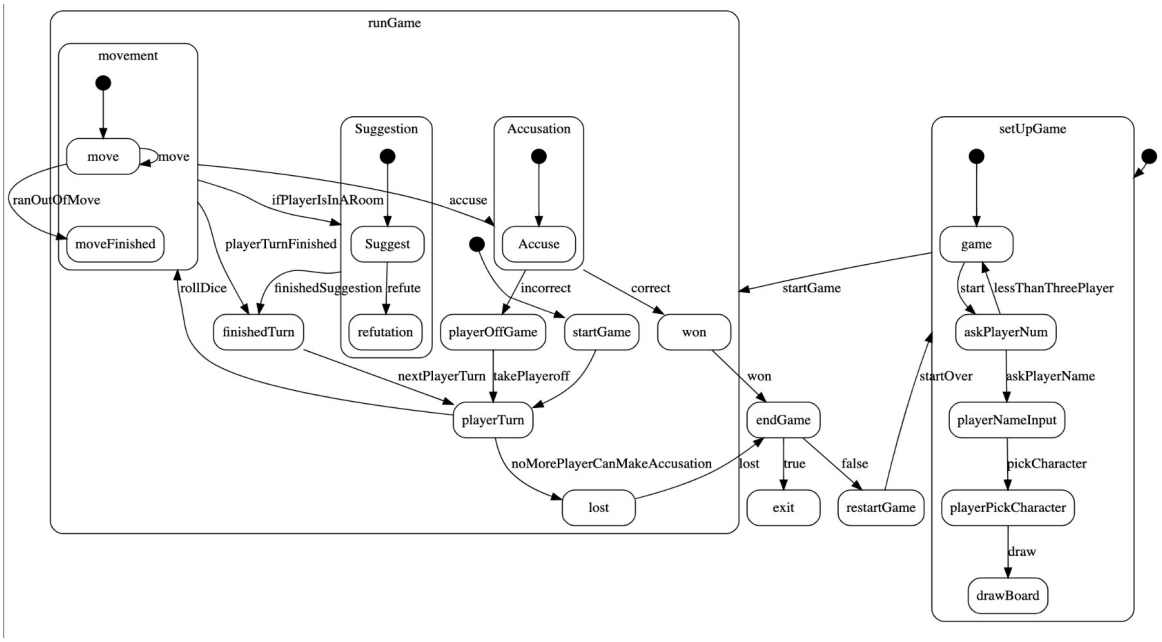
In order for the improvement, we change a lot about the code. The most significant and representative one is the Game class is separated to several and the GameData class is constructed. The GameData class represent the Model of the MVC design. It contains all the data that the Game need, and only has getter and setter methods, it brings the convenient as when the data is updated, just call it, easy and concise.

Obviously, different original design can make it easier to feature the GUI. A good original design always contains the properties of the code legibility and maintainability, that means each main class is concise and when need to add the GUI, there is no need to change the whole part of the code, just need to change the specified part.

For instance, the Model-View-Controller design is the classic good design, the user obtain the information from the View and when the user operates the window to make changes, the Controller is

called to inform the Model that the data need to be updated. When the data is updated, the Controller will be called again to redraw the View of the window. (View -> Controller -> Model -> Controller -> View). As we can see that the Controller plays as the bridge separate the View and the Model perfectly, Model and Controller can only communicate through it. The MVC design let the code of each class maintainable and readable.

Discussion of State Diagram



We have two super-states that covers most of the sub-states, and the remaining states are for whether to start a new game or exit. Our state diagram is pretty easy to understand as the name of the state and the transition are straight forward, the reader can know the functionality through it.

- The first super-state is the setUpGame state, during this state, the sub-states will ask the details of players, that is the number of players, their corresponding nickname and character. After that, it will draw the board and hand out cards.
- When the gameSetUp is finished, it will transit to the runGame super State. Within this state, the second-level super-states are also constructed to cover its main functionality.
    - PlayerTurn State is the state that takes the player who can still play to play in turn. After the dice are rolled, it will transit to the Movement State that allow the player to move the character.
    - After the Movement number is running out, there’re three possible states that can be transited to.
        - It can be transited to finishedTurn State if the player chooses to end the turn. Then, it will transit to playerTurn State again as long as the next player in turn is caught.
        - It can be transited to the Suggestion State if the player is in the room. Within this state, the player makes the suggest by using the current room they are in and other

chosen cards. Other players then will try to refute you with their hand cards. After the Refutation State, it will transit to finishedTurn State.

- It can be transited to the Accusation State if the player chooses to accuse. Accuse is made by selecting the murder cards, and depends on the correction, it can transit to two different states.
  - It can be transited to the won State if the accusation is correct. Then it will transit to the endGame State to ask whether to start a new game or exit.
  - It can be transited to the playerOffGame State if the accusation is incorrect. For this state, the player is out and can only refute others. Then it will transit to playerTurn state to see whether continue the game or not.
    - If noMorePlayerCanMakeAccusation, then it will transit to endGame state, else, the game will continue to roll the dice.