OBJECTED-ORIENTED PROGRAMMING REPORT

Programming Language: Java

Team: Yumu Xie(po21744), Hongbing Qiu(mk20661)

Summary

Shortly, there are 83 tests has been passed for the closed part of Scotland Yard task. However, the AI part does not include in this submitted coursework. Here below is thought of this coursework, how to deal with difficulties and team working pattern.

At first glance of team working pattern, it is a little bit different with traditional teamwork pattern, for example giving each member in this team a part of task to do. Teammates had discussion before actual coding separately. After finished, team members held meetings to choose better codes for submission.

Initialisation

At starting part, the first point is to create a class called 'MyGameState' which implemented from 'GameState' interface. Before any operations, the second step is to list attributes needed to use in this class. After that, building a 'MyGameState' constructor and using 'this' keyword to refer to these attributes in constructor is very important before checking any illegal operation of this task. It is illegal to put 'these keywords into attributes after checking step because without referring the current object in a constructor, the whole program will crush.

Getter

In the Getter part, it was implemented some functions such as 'getSetup', 'getPlayers' and so on. Basically, for these starting checking functions and non-null overriding functions, these ways of implements show that how to skilfully construct enhanced 'for' loops to iterate over elements and multiply 'if' statements to get the correct check conditions. And also used collections like set, list, and optional containers to contain elements or add/remove elements into usage. In 'getSetup' function, it only needs to return keyword 'setup' only since this keyword has already been set in listing attributes. It represents 'GameSetup' in 'MyGameState' which is used to the graph and move round in this game. In 'getPlayers' function, it requires us to return players in this method. So, the team chose to create a new set to contain MrX and used enhanced for loop for putting correct number of detectives inside this set before using 'copyOf' method to transfer HashSet type into Immutable Set type. In 'getDetectiveLocation' function, it takes a detective as parameter. So, the team used enhanced for loop for traversing detectives which belongs to list of players type in order to check its identity. If they matched, this method should return location. Otherwise, it needs to return empty collection of this optional type. In ‘getPlayerTickets’ function because this method belongs to Optional<TicketBoard> type. It means this function must get a new TicketBoard for proceeding data. So, creating a new TicketBoard is necessary and there is a ‘getCount’ method which needs to be overridden writing. The logic here is that after checking identity (MrX or detectives) we directly return the corresponding tickets by using ‘getOrDefault’ method. This method is used to initialize tickets. Outside this new TicketBoard, it also needs to check the identity of piece and return this new TicketBoard because inside this TicketBoard, the checking identity program has been written. In ‘getMrXTravelLog’ method, it is also easy as it only needs to return log which has already been defined in the beginning of the ‘MyGameState’ class as attributes.

Advance, Winner and AvailableMoves

For ‘getWinner’, ‘getAvailableMoves’ and ‘advance (Move move)’ functions, they have close relationships but mutually exclusive to some extent. In ‘getWinner’ function, deciding which one (MrX or detectives) can match the winning conditions is vital, for instance MrX is captured, MrX is stucked and detectives cannot move anymore. Before matching these winning conditions, this method should always return empty cause the game is still being processed. In ‘getAvailableMoves’ function the two helping methods ‘makeSingleMoves’ and ‘makeDoubleMoves’ are to help this function for better operations. In these two helping methods, regularly checking whether the adjacent nodes are occupied by other players (detectives) or not and whether player has required tickets for move are necessary. In ‘getAvailableMoves’ function, in order to get rid of mutually exclusive relationship with ‘getWinner’, creating winner to represent it is significant and once ‘getWinner’ function gets winner, ‘getAvailableMoves’ only returns empty which means empty moves. For ‘remaining’ in these functions, it controls which player can move or not in rounds. So, using this to check whether the round turns or not is intelligent. For double moves, MrX should also match two conditions: MrX has double ticket and remaining travel log size should greater or equal to two. For ‘advance (Move move)’ function, it always returns the latest game state to the UI. And it is used to check and determine in each round who can make a move. If in ‘getAvailableMoves’, there is no move inside, it is illegal. Creating a player’s destination visitor pattern is for the sake of better operations. In each round, the player’s (include MrX and detectives) location and ticket should be updated. Besides, ‘remaining’ should be updated, too. It is an important way to determine who can move in the next round. Followed by game rules, in each round, MrX will move first, then detectives move. For the move made by MrX part, the MrX’s log needs to be updated. According to the setup of move round, MrX will show up the tickets used and hide MrX's location, and MrX will show up both location and used tickets in special rounds. So, creating a newLog visitor can produce the new log for the move from MrX, and adding the newLog into the update log which includes the previous log. After that, when a move is made by detectives, updating the player’s location and ticket just like what have done before and replacing the new player in newDetectives list is targeted to reach. Then, giving MrX tickets which detectives have used is crucial. Updating ‘remaining’ adding up all previous remaining and removing those who have made move are good. If the remaining is empty, adding MrX inside can start a new round. But for general logic, executing ‘getAvailableMoves’ method and ‘advance’ at the same time before ‘getWinner’ make program running, because finally determine winner is the last thing in this coding.

Observer Pattern

In the observer pattern, the main purpose is to implement methods from the interface Model. Basically, if game state has any change, the observer pattern should be notified different state. For instance, in ‘chooseMove’ method, advancing the game state and checking whether ‘getWinner’ gets winner or not are the main job of this function. Once ‘getWinner’ gets winner, the observer will be notified to change state to ‘GameOver’. Otherwise, under ‘getWinner’ is empty, the observer will be notified to adjust state to ‘MoveMade’.