ScotlandYard Model Report

Model Development

The first task that we needed to tackle was the Scotland Yard model constructor. This method is used by the game to create a list of players used throughout the entire model. The constructor also checks that no two players have the same colour when the game starts ,detectives are not on the same location and all players have their respective tickets.

Next, we had to implement the methods from the ScotlandYardView interface, this made us realise that we should have a list of detectives so that we could deal with the detectives easily, however, we were not able to implement all the methods as we first needed to implement the logic of the game.

Valid Moves

This had then brought us to the finding a player's valid moves. As there were several circumstances that we needed to check for, we decided to split the problem into smaller parts so that we could approach each situation separately. This ended up with us having several smaller methods rather than having one large method where we compute the valid moves . for example, we have a method to find all the edges around a player ; then we have methods to filter the location and tickets. We then convert these edges to moves as such , this would then return all the possible ticket moves for all players, this was our getMoves () method.

In addition, we then needed to find the double moves for mrX. We first needed to check whether mrX had any double tickets left and that it also wasn't the penultimate round. We then go through each move using Downcasting and call get moves on it so that we are able to find the next corresponding move. This made an understandable shortcode which help in dealing with the tests in a much more simpler way.

GameOver

We wanted to follow the same style that we used for valid moves for the game over tests. Hence, we had separate methods for each winning condition. This made it a lot easier to check when the game was over and also returning the winning set of players as well.

Accept

In the accept method, we check whether a move that is about to be made is valid or not and we check this by seeing if the move that is about to be made is in our set of valid moves if this isn't the case the program will throw and inform the players that an illegal move was played. If not we will then visit the move. For a passmove, all that we needed to do was inform the spectators.

However, for a ticket move, we needed to ensure that the player's location was updated and the ticket removed. In addition, for a ticket move, we needed to check if the player was a detective and if so the ticket that they had just used should be given to MrX. Furthermore, if the current round is a reveal round, MrX's location should be updated with his current location.

The logic for a double move is very similar to a ticket move, however here we don't need to check whether or not the player making the move is a detective as they wouldn't have any double move. However, there were more factors that we needed to take into consideration . For example, checking for a revealing round twice removing three tickets whilst also taking into account the order of every call made on these functions.

One of the biggest problems we came across was updating the last location when it was a double move. We found that the location that the spectators tests were expecting were not always the updated last location after a double move. Our solution was to store the last location before changes were made and create the double move that was to be notified accordingly ;having another variable that would hold the last location after the first round and second round. Therefore before the spectators were notified for the first move, the last location would be set to the value of the variable after the first round only and then set to the value of the variable after the second round (the actual last location after the double move is completed) when the spectators notify the second move. Thus enabling us to have a fully working Game model.

AI Development

Our main idea in constructing the MrX Ai was to implement a scoring function which will attribute a score to each move in the set of moves MrX could use and then sample through the moves to simply choose the best move to play by calling it in the "callback.accept" method .We decided to use a HashMap structure made up of the moves and their respective scores.

The first step in doing this was to create a copy class of "ScotlandLardPlayer" named "PlayerConfiguration" in the AI package. Doing so we could be able to have access to a mutable player from which we could get useful information such as MrX and detectives locations. Furthermore, We created a Score Class which mainly holds the logic and design to attribute scores to MrX moves .We started by adding some helper functions from our Model such as getMoves () and mrX Moves () changed to accustom the Score class . This would help us to easily get a List of players and the Moves of MrX .

For a start we decided to assign scores to moves based on the number of nodes each move makes available for the next move.Because The move type is unknown when called we decided to use the visitor's pattern to deal with the different type of moves. The ScoreMove() function simply call visit on each move. The logic for scoring the moves is implemented in their visit method .

Moreover, we decided Secret tickets and Double tickets should only be used After a Reveal round ;as such we implemented selectDouble () and selectSecret () which picks the best

Doublemove and Secretmove accordingly. In addition , we created a method singleMoves () which selects the best Taxi , Bus or Underground move . Using a HashMap was very beneficial as all we had to do is extract the key with the highest value. The method chooseMove () deals with the selection of the Move that MrX will play. If the next round to come is after a reveal round , It select Double or Secret ticket, else it would get a normal ticket move.

After that we decided to implement the Dijkstra's Algorithm to score the moves based on distances from detectives. This was implemented in a separate class and was instantiated in the Score class from which the shortestpath () method was called in the move visitor methods. For each move whose shortest path to a detective was less than 4 we would assign a negative score. As such , the score will take into consideration global distances to detectives. The Ai class encapsulated the Score class simply call on the best move in the call back method.

We ended with a Distance One move-ahead Ai for MrX which is able of choosing appropriate moves.

Reflections

We believe that the way we approached the model and solved the problems was well done. By splitting our problems into smaller parts making it easier to tackle, while also getting a better understanding of the problem. However, we came across some repetition of code with having very similar for loops within our model. This is something we would like to improve in the future.

Our initial thought about the data structure that would hold the scores was a map of the edges on the graph and the corresponding score . We later on realised that this resulted in the absence of moves with secret ticket . We spent a lot of time coming to that realisation and this was time that could be used in implementing the Mini Max algorithm to make the AI more robust . This lead to comprehension that debugging a UI is very tedious when it comes to its interface and output interactions . As a result simplification should always be done where possible to minimise errors.

By using a Map of Moves and scores it was easier to call function directly on the moves done and attribute scores accordingly using visitor pattern. We also used the getclass() method in some of the intricate parts of functions like selectDouble () and selectSecret () where we are not necessarily doing something on the object itself. We think we could have avoided that by probably using overloading on methods that would take the different object class or using the decorative pattern to hide the use of getclass() or " instance of " for example .

To conclude, this Coursework has really strengthen our skills in java programming as well as sharpening our team work skills and we are happy with the work we have been able to pull through.