Scotland Yard Project -Report-

Arun Steward (1811148 – cw18903) & Junoh Park (1832704 – ov18075)

<TASK: Part 1>

The game, ‘Scotland Yard’, constructs from the constructor, ‘ScotlandYardModel’. In this constructor, it declares variables which form the game and check if the game has appropriate data to start. Also, it lastly checks if the game is in game-over status by ‘checkGameOver’ method. In the method, all cases of game-over were checked by changing the value of variables, ‘mrxloss’ and ‘mrxwin’. These variables then are used in the method ‘isGameOver’ to confirm the game-over status.

After this step, if there are no problems, it could start the game. The game starts from ‘startRotate’ method. In the beginning, it calls the method, ‘populatemoves’ to fill the set ‘moves’ with the possible movements from the current location. To find possible movements, it first picks out Nodes where a player can go at a time with tickets player has. Then, it removes the Nodes that have detective to decide possible movements finally. After these steps, checking the variable ‘moves’ is not invalid and the first player is MrX are followed. Then the player moves via the method, ‘makeMove’ with ‘this’ as a ‘Consumer<move> callback’ parameter.

The player calls back the ‘accept’ method after having a move in ‘makeMove’ method. In the ‘accept’ method, it checks that the selected movement is possible again with ‘populatemoves’ method. Then, the cases are divided into two, which are the variable, ‘spectators’ is empty or not. Firstly, If it is not empty, it would be divided again into types of the current player, MrX and detectives. If the current player is MrX and try to do Double-Move, the ‘DOUBLE’ ticket would be removed and the ‘onMoveMade’ method which shows made move to spectators follows. Then the game calls ‘visit (DoubleMove m)’ method.

Else if the current player is MrX and try to do Single-Move, then the game calls ‘visit (TicketMove m)’ method and increases ‘round’. Then, the game calls ‘onRoundStarted’ and ‘onMoveMade’ methods which shows the change to spectators.

Lastly, if the current player is detectives, then it also calls ‘visit (TicketMove m)’. These ‘visit’ method removes the used ticket and changes the location of the player and gives MrX removed ticket if the player is a detective. Additionally, in the ‘visit (DoubleMove m)’ method, it calls ‘visit (TicketMove m)’ twice and increases ‘round’ after each call. Also, ‘visit (PassMove m)’ is for detectives who do not have more tickets. Else, if the ‘spectators’ is empty, the steps for spectators are removed.

After these steps, variable ‘currentplayer’ increases for the next player. Also, in the middle of those steps and after them, the ‘checkGameOver’ method is called. So, if the game is over, the method ‘GameOver’ is called and the game is stopped. However, in the game is not over cases, the game calls ‘startRotate’ again if the current player is not MrX. Else, it calls ‘onRotationComplete’ and goes back to the ‘startRotate’ method.

Also, the method, ‘transportInEdge’ is used to change the Transport type into Ticket type and ‘rectifer1’ and ‘rectifier2’ methods work to decrease and increase the value of ‘currentplayer’ for each. Moreover, we updated the last revealed location of MrX in the variable, ‘mrxlastlocation’. These methods, variables and other methods declared fundamentally used in the process of game as well as previous things.

<TASK: Part 2>

The AI player starts in ‘makemove’ method. At the beginning of the method, it calls ‘floydwarshall’ method first. In the method, it makes 2D array, 'graph' which means the connection between every Node. So, the value of the 2D array means the distance between Nodes. Then, with the for loop of every possible movement, the AI first computes the average distance between the detectives and the destination of move through 'floydwarshallget' method. Also, it computes the number of Edges of the destination Nodes, since the number of Edges means the possibilities of escape. Then the biggest sum of the computed values shows the highest probability of victory. Hence, the AI will choose it.

However, in this situation, when one of the distances between destination and detective is 1, it means that AI will be caught next turn. Therefore, to avoid it, at the beginning of computation, the choice will be removed so AI will not choose it.

<Reflection on Achievement>

- Arun

In the early stages of development, the first week or so, progress was very slow. At first the project was extremely daunting due to the relative lack of Java I had seen before starting the process, I distinctly remember not understanding the task for a very long time despite attempting to write code for it; in reflection I believe that I should probably have read the development guidelines a little more carefully. Despite this setback, I was able to make a few decisions at this stage of the project that consequentially saved a lot of time to the overall period of work. The best example of this was to straight away create and use a list of Scotlandyardplayer objects for ease of use and being able to use the iterability that a list allows.

Difficult points in the project other than the very start as I have mentioned previously were fairly frequent, usually with wildly varying degrees of time needed to overcome. Several notable points that took a lot of thinking as well as a few misguided attempts were:

Implementing StartRotate; this was a very difficult task to keep track of conceptually, incorporating both the Visitor pattern to deal with the three types of tickets possible, and prior to that the Consumer pattern too in order to allow the user to select a move from th list provided. These two tasks both needed to be completed in order to pass any tests at this stage, making development very unnerving as one did not have any reassurances that one's direction was correct until after all tasks were completed. Nevertheless, having completed both the accept method for the consumer patterns callback and also the three methods in the concrete visitor (the model) to deal with the concrete elements (the moves), many tests simultaneously completed, giving reassurance that I had not strayed from the correct path.

Implementing the valid move logic (populatemoves); this task proved to be far more difficult than anticipated as I quickly realised that doing the logic for a double-move would be very conceptually challenging. At this point I chose to do the solution in the way I have rather the recursive solution that was possible in order to make the code more readable and to prevent having to generate false Scotlandyardview objects in order to achieve the intended goal. The solution that I created was first planned out using a tree diagram which I the translated into code afterwards, this helped in many ways, especially since considering all nine cases was very difficult to keep track of.

Implementing Spectators; this task did not take as long as the two before mentioned, however despite quick initial progress, this stage produced a lot of small bugs in practice when I first started notifying spectators at various points throughout the accept method and concrete visitor methods. These bugs were time consuming to track down and caused considerable amounts of effort to be expended to solve. In doing so the model passed all tests and was "complete".

Unfortunately the completeness of the model was celebrated early, there was still a subtle bug with the round incrementation that caused the game to crash after the first rotation and to misrecord the moves that MrX was making. Luckily with a last minute shift of work, Mr Park managed to find this bug and solve it accordingly, actually completing the model to a fully working standard.

When starting the AI section of the coursework, much was the same as when embarking on the development of the model, combined with my Intellij install suddenly ceasing to work, progress was again initially slow. However, realising that just programming with no plan would lead to sure disaster as it initially did on the valid moves function before, I make a rough plan for the AI to incorporate both a bias based on the valency of the destination node but also the distance from that node to the current positions of the detectives using an algorithm called the Floyd Warshall Algorithm, which required both an interface and extensive modifications to complete. The results of these endeavours led to the fully functional Vision AI, which successfully performs route analysis on the graph from the view of the game and incorporates valency bias. A triumph of this approach was to use the superior (in this case) Floyd Warshall approach in stead of Dijkstra's Algorithm. The reason for this is that by doing things the way I did, using a dynamic programming approach, means for a faster general use case as the information only needs to be calculated once at game start-up in O(n^3) and never again, once done this gives a lookup time of O(1) during the actual game making the AI quick to pick moves in game. Had Dijkstra's been deployed here, it would have lead to a computation of O(n^2) six times per rotation, leading to a slower feeling program.

- Junoh

When I started this coursework, I suffered from understanding it. Although I read the development guidelines, a large number of codes and tasks made me depressed. Since I felt depressed, I think that I tended to evade the full understanding of the codes and mechanisms. And it was one of the big faults. Therefore, without understanding, making codes was the hardest part and made me depressed again, which was a vicious circle. One day, I found that I was avoiding doing this work myself, and I tried to know why. Then, after having a reflection time, I noticed my problem and I tried to understand coursework fully from the beginning. It was quite helpful to me, and coding became more productive than before.

At the working, I put most of my effort into checking game-over, sending information to spectators and implementing the final game. Firstly, when I implemented to check game-over, I needed to consider all cases that can end the game. So, it was not easy to find them and make appropriate codes for it since each case require unique codes to determine it.

Then, when I worked for sending information to spectators and implementing the final game to pass the all test, dividing case by case and sending different data to spectators were complicated. Moreover, although we had finished making code and passed single tests, there were errors at the final tests of the whole game. Therefore, I reorganised all part again. Since there was a problem in order of codes, I broke many for loops and if statements. Also, when I broke those statements, most of the previously passed tests were stopped too. Therefore, I wrote orders and sequences of the game on the paper and tried to reorganise it. After those works, I finally passed all tests and became possible to play the game.

In AI parts, Arun’s idea was good, so I followed his one. By following it, I added the part that AI does not choose the point where the distance between detectives is 1. Although our AI works well, I feel that more computing and idea are needed to make perfect and outstanding AI. So, if I have a chance, I would like to evolve the AI more than now.