Scotland Yard Project Report

The cw-model passes all tests and both GameStateFactory and ModelObserver are fully implanted and functional, below is a summary of the code.

* The first two listed methods are helper functions which create the set of moves for the player in question, they are called by getAvailableMoves and return a set of moves possible by the player. They take into consideration whether the spot is occupied and the current tickets of the player. Written by Jayden.
* The next set of methods are called by this.winner every time a new GameState is created to calculate whether a winner has been found, the 4 different ways for the 2 sides to win have their own specific methods. While they could have been inlined and removed, it feels much easier to read this way. Written by Gabriel.
* The utility helper functions are used to gather specific attributes or to learn specific properties from the code that normally cannot be obtained, e.g. finding the player by their piece or finding out if they are stuck. Within the updateLog method is the visitor pattern in practice used when updating MrX’s log based on the move he has made. Written by Jayden.
* A named inner class, GetDestination, is implemented as there was a lack of an easy way to obtain the destination from a move. It acts as a getter for the destination attribute of a move. This class also utilises the visitor pattern as a Move can either be a SingleMove or a DoubleMove. Written by Gabriel.
* There also is a set of getters used to obtain the various attributes of the GameState class itself. The method getPlayerTickets features a lambda to define the behaviour of getCount from the TicketBoard interface without explicitly creating a class; this method is there to obtain the amount of tickets from the type of ticket. Co-written.
* Like getWinner, getAvailableMoves is called every time a new GameState is created to obtain the set of moves available to the players, this method also keeps track of who’s turn it is. On the second line a check is made to see if a winner has been declared, this necessitated that the winner be found before the moves are calculated; the check is done as the tests do not check if the moves attribute is empty (which would mean a simple ternary operator would pass it), but instead they check if getAvailableMoves returns empty. Co-written.
* The final method that advances the GameState once a move is made; this method updates the log if the move was commenced by MrX, and then updates the remaining list of players. Instead of working on an empty list and adding the players who need to make their move, instead the list is full and the players who made their move are removed. Stuck players are also removed by using removeIf on the list of unmoved pieces. Here, a method reference is used which avoids the repetition of a lambda expression, where this::isStuck references the piecesNotMoved variable as the parameter of isStuck.
* Finally, the build methods create the GameState with testBuild asserting certain properties that must hold e.g. asserting that there are not multiple detectives

----------------------------------PUT MODELOBSERVER DESCRIPTION HERE--------------------------------

The cw-ai is a finished model of an algorithm to be used for MrX when deciding moves, to summarise it obtains a weighted score for all of its possible moves adjusted for various events and returns the best move which is simply the move with the highest score.

The implementation of the scoring is contained within the ScoreMap class wherein all the methods lie, the AI first finds out what moves it possibly has, and then takes into consideration the distance to each detective from its destination using Djikstra’s.

------------------------------------------Describe Djikstra’s here--------------------------------------------------

--------------------------------------------------------------------------------------------------------------------------

The score of each move is then evaluated based on the distance, the ticket used, and the node the AI is ending up at.

First, the score is adjusted based on the distance; originally, a meanDistance was calculated adjust the score, but this was theorised to lead to an issue where if 3 players were far away, and 1 was close, then the threat of the close one would not be big enough, this led to an optimisation and a change to calculating the mean danger of the given location, instead, an average danger is assigned from each detective and a mean is calculated from that, detectives too far away to matter (those who are further away than the rounds remaining) would be ignored.

Furthermore, Djikstra’s was optimised to additionally ignore detectives who were stuck or did not have the required tickets for a move.

Finally, a literal adjustment was made to the score of a location depending on the imminent proximity to a detective, where locations right next to a detective were basically grounded in score to never be considered.

Secondly, the score is adjusted based on the type of transportation, this method uses a visitor to deal with the fact that Move can be a single or a double move. It also uses 2 method references for both parameters of the visitor when evaluating the ticket itself.

------------------------------------------Describe returnScoreSm and returnScoreDm------------------------

Thirdly, the AI evaluates the node it is ending up at, scoring nodes which have more variation higher than those which don’t. It prioritises stations with more transport when it is exposed or in imminent danger.

Finally, the score is returned, and bestMove picks the move with the highest score. This returns to main and executes the move.

Of course, the AI is not perfect, from some play testing, it has been observed that the AI performs well in the early-game when detectives are quite a few distance away, but begins to make errors in the mid-game while in the late-game it loses to human intuitiveness and lack of consideration of it (e.g. if in imminent danger in all possible moves, consider which one the players will most likely not pick)

Overall, this AI would be most likely be suitable as an medium difficulty AI, it is not too hard for the players to catch the AI, but at the same time it is not too easy to catch it out in the early game. Furthermore, the AI does miss some obvious logical conclusions what move it should make.