Scotland Yard Game

Sriram Cheedi
dz23405@bristol.ac.uk

Aruna Kumara Aritha Nisalvin
qs23940@bristol.ac.uk

1. Overview

We were able to pass all the tests in the CW-Model , and also implemented a MrX AI, Minimax Ai for MrX using Minimax algorithm with alpha-beta pruning.

1.1 CW-Model

MyGameStateFactory

Getters: First we have implemented all the getters. We have provided all the basic checks so that the given parameters don't hold null and we have also added player's tickets. We implemented the getwinner() function which has all the conditions for winning which passes all the tests. For getAvailableMoves() we have written 2 helper functions – one for Mr. X and another for the Detectives. The helper functions get the moves for Mr. X and Detectives from makeSingleMoves() and makeDoubleMoves().

makeSingleMoves(): This function gets the moves for both Mr. X and detectives. It checks whether the destination is same as the detectives' location, if it is the same it adds the moves. It also gets the moves for Mr. X's SECRET ticket.

makeDoubleMoves(): This function gets moves for Mr. X, It checks whether the location is occupied by the detectives and also generates all possible double moves while simultaneously looking for the availability of the tickets. This includes all double moves with SECRET tickets.

Advance Move: It includes the visitor patterns for single and double moves which updates the travel log of Mr. X, remaining players based on ticket availability and available tickets. The travel log for Mr. X checks all the possible cases with the double move with the visibility of Mr. X, whether Mr. X's move is visible or hidden affects the detectives' strategies. In the update tickets method if a detective makes a move , their number of tickets are decreased and given to Mr. X and if Mr. X makes a move only his tickets are updated.

MyModelFactory

getCurrentBoard(): Returns the current game state. This method provides encapsulated access to the game state.

registerObserver(Observer observer): Adds an observer to the set. If the observer is already registered, it throws an 'IllegalArgumentException'. This method ensures that each observer is only registered once.

unregisteredObserver(Observer observer): Removes an observer from the set. It throws an 'IllegalArgumentException' if the observer is not registered. This method allows observers to unsubscribe from receiving updates.

getObservers(): Returns the set of all registered observers, allowing external entities to query which observers are currently registered.

chooseMove(Move move): Advances the game state using the provided move. It updates the 'gamestate' by calling 'advance' method. After updating the state, it checks whether the game is over (by checking if 'getWinner()' returns a non-empty result). It then notifies all observers of the change using the 'Observer.Event' enum to describe what type of event occurred.

1.2 CW – AI

Overview
This AI is designed to play the role of Mr. X, which uses Dijkstra algorithm to calculate minimum distance between two node and scoring function. We have also implemented MiniMax algorithm using alpha-beta pruning which also uses Dijkstra algorithm with a different scoring function to optimize Mr. X's movement.

Dijkstra
We have utilized this algorithm for finding the shortest path from a specified start node to a target node within a graph.
It initializes a list dist where all nodes are set to MAX (a high value indicating they are initially unreachable), except the start node which is set to 0. It repeatedly finds the unvisited node with the smallest distance (currNode). The loop terminates when this node is the target node or when no unvisited nodes remain. For each neighbour of currNode, the function calculates a potential shorter path (alt) and updates the neighbor's distance if this new path is shorter.
Marking Visited: After processing, currNode is marked as visited by setting its distance to MAX. It returns the shortest distance to the target node if reachable, otherwise returns null if the target node is unreachable from the start node.

My Ai
MRXscore(): This function calculates a score for a given move for Mr. X. It calculates the location of Mr. X after the move and then computes a score based on various factors such as distance, freedom, and ticket scores. Then it prints out the move and its corresponding score.

MoveRate(): This function calculates a score multiplier based on the distance score. If the distance score is negative, it returns double the negative distance score indicating an increased penalty for risky positions. Otherwise, it returns the distance score raised to the power of 1.05, which adjusts for more sensitive growth.

freedom(): This function calculates the freedom score for Mr. X at a given location. It counts the number of escape routes available for Mr. X from the given location, considering whether the adjacent nodes are safe from detectives' proximity. It assigns a higher weight (1.5 times) to the number of escape routes found.

Distance(): This function calculates the distance score based on Mr. X’s moves and the detectives’ positions. It iterates through detectives, calculating their distance from their position to Mr. X’s move. It then computes a penalty based on the distance, reducing the score if the distance is short.

Penalty(): This method calculates the penalty for a move based on the distance between the destination and the detectives’ locations. It assigns a penalty of -200 if the distance is 1.

TicketScore(): This method calculates the ticket score for a move based on the types of tickets used and the risk level at the destination. It assigns different scores for each ticket type, considering the risk level and the distance from detectives.

pickMove(): This method is the main entry point for the AI to pick the best move. It calculates the score for each available move using the MRXscore method and selects the move with the highest score. It iterates through all available moves, calculates the score for each move, and selects the move with the highest score.

MiniMax

isMrXRemaining Method: Checks if Mr. X still has available moves left on the board.

detectiveMoves Method: Recursively applies all possible detective moves until Mr. X has moves remaining.

minimax Method: The minimax algorithm with alpha-beta pruning. Evaluates the best move for Mr. X by recursively exploring the game tree to a certain depth. At each level of recursion, it alternates between maximizing Mr. X's score and minimizing the detectives' score. Alpha-beta pruning is used to reduce the number of nodes evaluated, improving performance.

score Method: Computes the score for a given move by Mr. X. Considers factors such as connectivity, distance from detectives, and ticket penalties. TicketScore Method: Calculates the penalty score for using certain types of tickets based on the distance from detectives.

freedom Method: Calculates a connectivity score for Mr. X based on the number of adjacent nodes.

Distance(): This function calculates the distance score based on Mr. X’s moves and the detectives’ positions. It iterates through detectives, calculating their distance from their position to Mr. X’s move. It then computes a penalty based on the distance, reducing the score if the distance is short.

Penalty(): This method calculates the penalty for a move based on the distance between the destination and the detectives’ locations. It assigns a penalty of -200 if the distance is 1.

pickMove Method: The main method that selects the best move for Mr. X. Iterates through all available moves, applies the minimax algorithm to evaluate each move's score, and selects the move with the highest score.

Possible Improvements

Dijkstra

We can improve the efficiency of the Dijkstra algorithm by using priority queue which would typically prioritize nodes based on the shortest known distance, thereby reducing the overall computational complexity.

Scoring

A better and faster scoring function can be implemented by encouraging SECRET and DOUBLE tickets. Currently, the freedom method computes escape routes by checking adjacency and safety for each node. This might be inefficient if there are many nodes and players. Consider caching the detectives' locations and using more efficient data structures like sets for quicker lookups.

MiniMax

Implemeting a method which prioritise moves that are likely to be better can prune branches earlier which enhance the alpha – beta pruning. A better scoring function can be implemented with better ticket penalty values. A better Distance function can be implemented calculates accurately. We can give more depth by adjusting the ticket values and optimise the scoring function.