The Scotland Yard Project
Director's Cut

Lucas Mockeridge
ud21296@bristol.ac.uk

Nikhil Parimi
ob21372@bristol.ac.uk

1 Introduction

1.1 Scotland Yard

Scotland Yard is a board game in which a team of players playing as individual detectives chase a player playing as a criminal called Mister X around a board mapping London. The board contains the locations that players may travel to and from and which mode of transport they can use to do so. Players start with a number of tickets, each of which allows them to travel around the board by taxi, bus, underground, or ferry. Only Mister X can have double-move tickets that allow him to move twice in a round and secret-move tickets that allow him to travel by any mode of transport.

Mister X's moves are hidden, however his location is revealed at regular intervals throughout the game. After each move, Mister X puts the ticket(s) he used in his travel log which is visible to the detectives. If there are fewer than four detectives, the Police are added to make up the numbers. The Police are operated by the detectives collectively and move without tickets. If a detective travels to Mister X's location he must admit defeat but if after 22 rounds Mister X has evaded capture he has won.

1.2 The Scotland Yard Project

The Scotland Yard Project is a summative assignment completed as part of Object-oriented programming and Algorithms I (COMS10017) at the University of Bristol. The assignment is completed using Java, IntelliJ, and git in a pair-programming style and consists of a closed task, an open-ended task, and a three-page report.

The closed task is the implementation of cw-model producing a Java application that allows users to play a simplified version of Scotland Yard. Simplifications include the police and ferry not being modelled and the game ending once Mister X's travel log is full to allow the game to be setup with a variable maximum number of rounds. The open-ended task is the implementation of cw-ai producing an AI that plays as Mister X and an AI that plays as a detective.

2 Summary

2.1 cw-model

A complete implementation of `MyGameStateFactory` and `MyModelFactory` that passes all tests and results in a fully functional GUI.

2.2 cw-ai

A Mister X and a parallelised detective AI that use Monte Carlo Tree Search, with ϵ-greedy playouts using Dijkstra's Shortest Path Algorithm, that exhibit anytime behaviour realised by threading.

3 cw-ai

3.1 Minimax

Minimax is a depth-first-search algorithm used to play 2-player zero-sum games with perfect information under the assumption that players play optimally. A game tree is generated up to a specified depth and the current state of the game is evaluated using a heuristic. One player moves to maximise this heuristic, the other moves to minimise it. The size of the game tree can be reduced by using $\alpha\beta$-pruning to remove moves that will never be chosen.

3.2 Monte Carlo Tree Search

Monte Carlo Tree Search is an anytime aheuristic best-first-search algorithm that balances exploration and exploitation of the search tree effectively. It consists of four stages: Selection, Expansion, Simulation, and Backpropagation. The move corresponding to the immediate child with the highest simulation win rate is played.

- **Selection**: The search tree is traversed from the root node and the child node that maximises UCT is successively selected until a node that has not been fully expanded is reached. UCT stands for Upper Confidence bounds applied to Trees and balances the exploration and exploitation of the search tree
- **Expansion**: If the selected node does not correspond to a terminal state and has been visited before, a child node that represents an available move from the selected node's state is added
- **Simulation**: A simulated playout is performed until a terminal state is reached on either the selected node or it's new child if it was expanded
- **Backpropagation**: The result of the simulation is propagated back up the search tree along the direct path to the root node

3.3 Minimax vs Monte Carlo Tree Search

Monte Carlo Tree Search was chosen over minimax for various reasons. The first being that Monte Carlo Tree Search is an aheuristic algorithm meaning that it does not require any knowledge about the game apart from its rules and end conditions. This is an advantage as deriving a high quality heuristic for both the hider and the seeker would be challenging and require experimentation.

Another advantage of Monte Carlo Tree Search is that it grows the search tree asymmetrically by concentrating on the most promising nodes. This makes it more suitable for games with large branching factors than minimax.

The anytime behaviour minimax can achieve with iterative deepening is considerably less attractive than Monte Carlo Tree Search's anytime behaviour. This is because the computation required for an entire search at an increased depth could be wasted if time runs out before the search finishes.

3.4 Domain Knowledge

We decided to incorporate domain knowledge into the playouts to make them as close as possible to playouts amongst humans and hence return more realistic results. Despite being able to perform fewer iterations, we felt this would allow us to build a more accurate statistic faster. ϵ-greedy playouts were used to incorporate domain knowledge.

ϵ-greedy playouts balance exploration and exploitation by playing randomly with a probability of ϵ or using domain knowledge with a probability of $1 - \epsilon$. "Systematic testing showed that the best results are achieved with $\epsilon = 0.1$ for the hider and $\epsilon = 0.2$ for the seekers"[1, p. 289].

Move filtering is the easiest way to incorporate domain knowledge into ϵ-greedy playouts. Mister X's moves were filtered so that moves which lead to defeat in the next round were discounted. Double moves are only considered if there isn't a single move that doesn't lead to defeat in the next round. Secret moves are not considered during the rounds before Mister X's location is first revealed, the round before Mister X's location is revealed, or rounds where Mister X can only move by taxi. However, if a secret move makes use of the ferry it is considered. A detective's moves were filtered so that moves which would leave them unable to move in the next round were discounted. After move filtering, Mister X chooses the move that maximises this heuristic

$$|L_v| + \deg(v) + (\min_{i \in D} d_{v,i})^2 \tag{1}$$

where v is Mister X's destination, L_v is the set of all Mister X's possible locations after moving to v, and $d_{v,i}$ is the shortest path from v to i. $d_{v,i}$ is computed using Dijkstra's Shortest Path Algorithm.

To make the Mister X AI more risk averse, if the detectives have a winning move it is always played. Otherwise they use a more strategic approach. In the rounds before Mister X's location is revealed they move to the best connected station that they can. If Mister X's location has been revealed in the round they pick the closest move to that location after move filtering. If Mister X's location is hidden they pick the closest move to the possible location that maximises (1) after move filtering.

As the detective AI plays the game as if it had perfect information, the detectives choose the closest move to Mister X's assumed location using Dijkstra's Shortest Path Algorithm after move filtering.

3.5 Mister X AI

The Mister X AI treats the detectives collectively as a single player to allow for a deeper search tree to be generated and a reduced space complexity. It has also been designed to be as risk averse as possible. Node expansion uses the same move filtering as the ϵ-greedy playouts so is biased towards child nodes whose move would result in defeat but also uses domain knowledge to discount moves.

3.6 Detective AI

Monte Carlo Tree Search requires perfect information. From the detectives perspective, the game has imperfect information as Mister X's location is usually hidden. Our solution was to calculate the possible locations of Mister X using the travel log, use (1) to select the n most probable locations, run the algorithm using each of these locations as if the game had perfect information and combine the results to select a best move.

The n most probable locations are selected using the same heuristic that Mister X uses to evaluate a destination. Running the algorithm for each of these locations is performed concurrently using root parallelisation. For each of the n possible locations a `GameState` is created and used to run the algorithm on a separate thread. n will be limited by the number of threads a machine can run simultaneously without causing performance issues.

In the rounds before Mister X's location is first revealed, the detectives move to the best connected station that they can.

3.7 Threading

The anytime behavior of Monte Carlo Tree Search was implemented by suspending the execution of ai-thread-0 whilst running the main loop of the algorithm on a separate thread. Just before the time to pick a move is up, the thread running the main loop of the algorithm is interrupted and execution of ai-thread-0 resumes. It was decided that the benefit of running the algorithm for as long as possible outweighed the cost of creating a new thread.

3.8 Areas for Improvement

3.8.1 Parallelisation

Monte Carlo Tree Search can be parallelised using leaf, root, or tree parallelisation. In leaf parallelisation multiple playouts are each performed on separate threads concurrently. Root parallelisation aims to produce better results by running the algorithm on separate threads concurrently and then combining the results. This was the approach used to run the algorithm for different possible locations of Mister X concurrently in the Detective AI. Tree parallelisation also involves running the algorithm on separate threads concurrently but to build the same search tree.

Whether the quality of the statistic built outweighs the cost of the threading overhead would need to be determined before incorporating any of the parallelisation methods into the Mister X AI. However, root parallelisation would be the easiest to implement as it avoids the race conditions that may arise in the other methods of parallelisation.

References

[1] Pim Nijssen and Mark HM Winands. Monte carlo tree search for the hide-and-seek game scotland yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):289, 2012.