Scotland Yard Report – Philip Mortimer and Jeremy Colfer

Abstract

We produced a complete implementation to the closed task that passes all the tests. For the open task, we produced three AI's that can play as both the detectives and as Mr X. We feel that we have produced an excellent solution to both the closed and open task.

1 - Closed Task

For the closed task, we produced implementations of the GameState and Model interfaces. We made immutability a key principle of our implementation. Our solution passes all the tests and is well documented and highly maintainable. We utilised a range of design patterns like the Model-View-Controller pattern and the visitor pattern to realise our implementation. We feel that we have produced an excellent solution to the closed task.

2 – Open Task

For the open task, we had to produce an AI player that chooses a move to play given a board state. We produced three different AI's (that can play as both Mr X and as a detective). Some of these AI's are trivial (e.g., our easiest difficulty setting simply makes random moves). Others are more complex.

2 a – One Move Lookahead

Our first AI approach was to look one move into the future and use the distance between detectives and Mr X as a way of scoring the game state. This can essentially be thought of as an expectimax search of depth 1.

Scotland Yard is a game with imperfect information (as the detectives do not always know exactly where Mr X is). To account for this, we produced a function that takes a board as an input and calculates all possible locations for Mr X. This is achieved by combining Mr X's travel log with the set of possible nodes Mr X may start from.

We used the mean distance between Mr X and the detectives to score a game state. We employed Dijkstra's algorithm to calculate the distance between any two points on the graph and cached the results. This does come at a cost though, as precomputing distances means that we cannot consider dynamic factors like the ticket count of the players. We produced different functions to calculate the distance between any two adjacent board nodes which somewhat counteracts this issue. For example, nodes connected by train may be classed as more expensive than nodes connected by taxi.

Thus, we produced a one move lookahead that generates all possible board states from the player's point of view. For each of these board states, it makes all possible legal moves and then evaluates the board state using the mean distance between the detectives and Mr X. The move that produces the best mean score from the current player's point of view is then selected.

2 b – Paranoid Minimax Search

Initially, we tried to extend our one move lookahead search into an expectimax search. However, this proved to be remarkably inefficient as the search space was simply too large to search to any meaningful depth. We reduced the search space by assuming that the detectives have access to Mr X's location. This assumption means that this algorithm only works for Mr X moves. We then produced a minimax search using this assumption. This search is a paranoid minimax search.

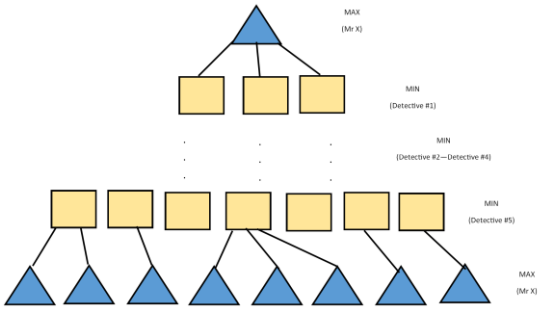

Figure 1 - Paranoid Minimax Search Tree

We used iterative deepening to allow our AI agent to search to the maximum possible depth within the allotted time. We used various move ordering techniques to increase pruning. The most effective move ordering technique proved to be evaluating nodes from the principal variation (PV) at the previous search depth. This is because the best move sequence at depth $d-1$ often proves to be the best sequence at depth d. Additionally, we used the killer heuristic, storing 3 moves per ply. This heuristic keeps track of moves that produce an alpha/beta cut-off at the current search depth. The idea is that if the move produces a cut-off in one branch, it is likely to do so in another branch. We also reused the killer heuristic table from the previous search depth as this is often a good indication of cut-off moves. Finally, we used the history heuristic, which is a depth independent generalisation of the killer heuristic. We assigned each type of move a priority score which results in the following ordering: $pv \rightarrow killer_{current} \rightarrow killer_{previous} \rightarrow history$ (where PV is searched first). Moves within the same priority class are ordered by their evaluation score.

Additionally, move filtering is employed to prevent the AI from evaluating moves that are clearly bad. For example, all double moves are filtered if the AI has any safe single moves. We have also filtered detective moves by having them move in a fixed order instead of in any order. Move filtering helps reduce the branching factor with relatively few drawbacks, thus allowing for deeper searches.

To evaluate a board state at a leaf node, we use the following equation:

$$s_{hider} = 90 \min_{i \in D}(d_{MrX}, i) + t_{MrX} + 2|L| + 10\frac{\sum_{i \in D}(d_{MrX}, i)}{|D|}$$

Equation 1 – The evaluation function used by the paranoid minimax search

$\min_{i \in D}(d_{MrX}, i)$ refers to the distance between the closest detective and Mr X. t_{MrX} refers to the number of secret tickets Mr X has. $|L|$ refers to the number of possible locations Mr X may be at from the detectives' point of view. $\frac{\sum_{i \in D}(d_{MrX}, i)}{|D|}$ is the mean distance between Mr X and the detectives. Note that the $|L|$ term is particularly important as it encourages Mr X to obscure his location, thus helping to offset some of the drawbacks that come from making the paranoid assumption. Nijssen and Winands [1] successfully applied a similar methodology when constructing their own minimax-based Mr X AI.

3 – Evaluation of Work

Regarding the open task, we feel that we have produced three distinct AI players which play at well-adjusted levels of skill. We feel that this is evidenced by Table 1.

Table 1 - Mr X Win rate for all difficulty settings (95% CI). For Mr X Hard setting, AI was given one second of thinking time.

Detective AI	**Mr X AI**		
	Easy	Medium	Hard
Easy	$65.7\% \pm 2.9$	$92.8\% \pm 1.6$	$99.0\% \pm 2.0$
Medium	$2.2\% \pm 0.9$	$15.0\% \pm 2.2$	$89.0\% \pm 6.2$
Hard	$0.1\% \pm 0.2$	$3.4\% \pm 1.1$	$55.7\% \pm 3.1$

We were pleasantly surprised at how effectively our one move lookahead AI agent played. The AI is simplistic in that it always moves towards the mean location of Mr X. However, this strategy proves to be highly effective. Both of us use this strategy when playing Scotland Yard as it is a very human approach. The AI opponent has an edge on human players as it able to deploy this strategy with perfect precision. However, the AI opponent does suffer from its inability to look further ahead into the future. This can sometimes result in the AI moving to nodes that it will get stuck at as it does not have the necessary tickets to leave the node. For future enhancements, I would look to implement an expectimax search. Despite our simplistic approach, the AI is rather difficult to beat, which I feel demonstrates that we have succeeded in achieving our goals.

Our paranoid minimax search AI for Mr X is a much more advanced AI, and we are extremely pleased with the results achieved. The AI can search up to 13 moves into the future, looking at over 10 million possible board states. Our paranoid search makes a few costly assumptions to achieve these results. Firstly, it filters out a range of moves which the AI should not consider. Secondly, it assumes that the detectives know where Mr X is. The first assumption is justified, as experiments indicate that our move filtering tends to increase the win rate and search depth of our AI. The second assumption is by far the biggest drawback to our approach. The alternative to this would be to construct an expectimax tree. This tree would be much larger and thus the AI would be able to search far fewer moves into the future. We feel that searching 13 moves into the future using a paranoid assumption is better than searching 2 moves into the future without it. This extended search depth is particularly important as detectives often make 5 moves in a row, which means that rather deep searches are needed to calculate good moves.

The drawbacks that come from making the paranoid assumption are partially mitigated by our evaluation function for a board state, which rewards Mr X for making his location as ambiguous as possible. To improve our Mr X AI, we would look to finetune the relative weights of each component of the evaluation function. To further improve the search, we would look to create a more lightweight representation of a game state that leads to faster move generation.

If we were to enhance our AI opponents further, we would look to employ Monte Carlo Tree Search, which has been shown to outperform minimax-based agents at Scotland Yard [1]. We would also look to implement the “Player of Games” framework outlined by Schmid, Martin et al. [2], which is well-suited to imperfect information games.

Overall, we feel that we have produced strong AI agents that play Scotland Yard very well.

References

[1] Nijssen, J.A.M. & Winands, Mark. (2012). Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard. IEEE Transactions on Computational Intelligence and AI in Games. 4. 282 - 294. 10.1109/TCIAIG.2012.2210424.

[2] Schmid, Martin, et al. (2021). Player of games. arXiv preprint arXiv:2112.03178.