

Project Part B Report

1. Introduction

Board games have been a popular pastime for centuries, bringing people together and challenging them to exercise their strategic thinking skills. With the rise of artificial intelligence (AI) technology, there has been increasing interest in designing AI agents that can play board games against human opponents. Notable AI agents that have won against international board game champions include DeepBlue which defeated human world champion Gary Kasparov in a six-game match in 1997 (The Guardian, 1996), and DeepMind's AlphaGo defeated the world champion 4-1 in 2016 (DeepMind, 2016). In this report, we present the design of an AI agent that plays a board game, Inflexion, against an opponent. The board game chosen for this task is a classic game of strategy and skill, which requires players to outmanoeuvre their opponent and capture their pieces. The goal of this project is to create an AI agent that can make the best decisions while taking the long term impacts of their actions into account.

2. Approach

2.1 Minimax

The minimax algorithm is used as the base algorithm for the game agent. The minimax algorithm is a recursive method that evaluates all possible moves of a player and chooses the best one assuming that the opponent also chooses their best move, meaning that it assumes the worst-case scenario. The minimax algorithm works well in a zero-sum game, where any gain for a player directly results in a disadvantage for the opponent. The game Inflexion is a zero-sum game since the goal of each player is to remove all opposing players' cells from the board, a player having 1 more cell than its opponent also means that the opponent is on a 1-cell disadvantage. Since each player wants to maximise the number of cells and power they have, while minimising their opponent's cells and power, the minimax algorithm is suitable for playing Inflexion.

A pure minimax algorithm explores the tree of possible moves until the game reaches a terminal state, and then propagates the outcome of each terminal state up the tree using min and max operators. In the game Infexion, this means that every single possible board state in the Infexion game has to be generated at the initial state, which is computationally too expensive. Therefore, the minimax algorithm needs to have a predetermined cutoff depth where the search terminates after that depth (Figure 2.2a). The suitable depth for minimax search that adheres to the time limit was found to be depth 2.

Aside: Calculation of number of possible Infexion board states

Since there are 3 possible states for each cell: (1) empty, (2) occupied by red, (3) occupied by blue and there are 49 cells, the number of possible Infexion board states is calculated as follows: $3^{49} \approx 1.92 \times 10^{19}$.

2.2 Evaluation Function

Cutting off the minimax algorithm prematurely means that the final utility of an action cannot be obtained. Therefore, a heuristic function is required to approximate the utility of a game state. The Infexion game is centred around the concept where a player aims to take over all of its opponent's cells. Therefore, one heuristic that can be used is the difference between the number of cells of the player and its opponent. This heuristic is useful because having more cells compared to the opponent means that the player is closer to winning the game than their opponent:

$$\text{player's power} - \text{opponent's power}$$

Further improvements can be made in incorporating each player's power into the heuristic, since a player's power is correlated with their number of cells and a player's power may also influence the outcome of a game.

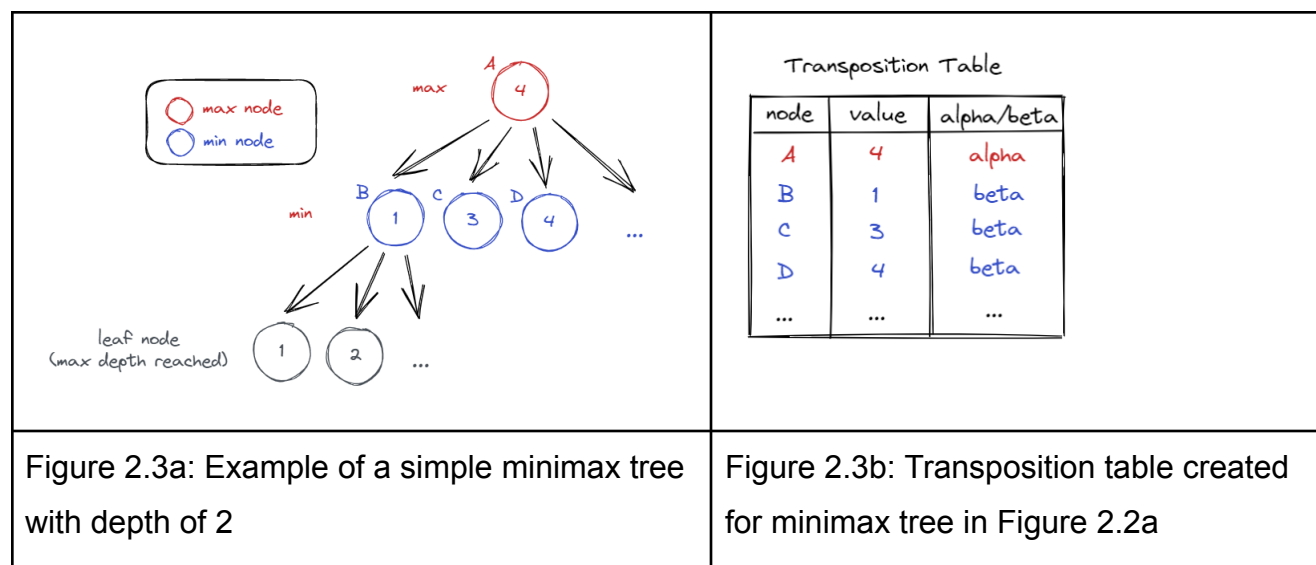
2.3 Improvements

To make the minimax search more efficient, we implemented alpha-beta pruning when performing the search. This allows us to identify and prune branches within the search tree that will definitely not impact the final outcome of the search. This is done by

keeping track of the alpha and beta values of each subtree, which are the smallest and largest possible values that a particular node's cost can be respectively, and pruning the subtree if it has a beta value greater than its parent's alpha value. This has the potential of significantly reducing the tree size since we are pruning entire subtrees from our search. Implementing alpha-beta pruning to the minimax search allowed the search tree to expand until depth 3 within the time limit, as opposed to depth 2 that was achieved by pure minimax.

A further optimization to our minimax algorithm is by maintaining a hash table of previously explored board states, called a transposition table (Qi et al., 2020). The intuition of maintaining a transposition table is that we can use our previous evaluations to aid with our current minimax search. Whenever we get an evaluation value for a board, we add the board state, along with its value to the transposition table (Figure 2.2b). This way, if our algorithm ever explores a state that we have previously explored, we can yield the output directly without needing to explore its subtree. This allows us to speed up our search significantly as we can prevent searching the same state multiple times.

Transposition tables were used with alpha-beta pruning as well. To do this, we store a flag for each board state specifying whether that state previously updated the alpha value, the beta value or is the final outcome after alpha-beta pruning as shown in Figure 2.3b (Qi et al., 2020, p.200). This way, when we reference this state again, we can update the alpha or beta value of the current search tree based on whether the entry in the table was an alpha-cutoff or a beta cut-off. Alongside saving the exploration time of subtrees of a regular transposition table, using transposition tables with alpha-beta pruning also narrows the alpha beta range, allowing more pruning to occur.



3. Performance Evaluation

3.1 Evaluation Function

When deciding between evaluation functions, the performance of 2 functions was compared against each other, using identical minimax algorithms, both with depth 3. The 2 functions that were compared against each other are:

$$(\text{red power} - \text{blue power}) \text{ --- (a)}$$

$$(\text{red power} - \text{blue power}) + (\text{red cells} - \text{blue cells}) \text{ --- (b)}$$

When playing both algorithms against each other for 200 rounds, with agents of each function alternating between being the red or blue player, it was found that the agent using function (a) wins 63% of the games. This shows that both evaluation functions have similar performance, with function (a) having an advantage over function (b). Therefore, it was concluded that function (b) is better than function (a) in estimating the relative utility of a board state. Moreover, it was also observed that both agents had more wins as the red player compared to being the blue player. This shows that there is advantage of going first in the game Infexion, as the red player always goes first.

3.2 Agent's Performance

The evaluation of the algorithm's performance was done by playing the minimax agent against different agents, using identical evaluation functions if needed, against each

other and comparing the number of wins each agent has over its opponent. The agents that were played were (i) a greedy algorithm, which is minimax of depth 1, and (ii) a random agent. To prevent all games from being identical and yielding identical results, randomness was introduced to the game by making the initial move of both players random. After 200 games of playing against agent (i), it was found that the minimax agent won in 93% of the games, whether it is red or blue. For the games where the minimax agent has lost against agent (i), it was found that the game ended in very few turn counts. Therefore, these losses can be attributed to bad random selection of initial moves, resulting in the agent being put in a difficult position to play from. Against agent (ii), the minimax agent won all of the 100 matches that were played. This shows that the minimax algorithm is outperforming the baseline algorithms and is the best performing agent among the three agents considered.

4. Supporting work

4.1 Evaluation

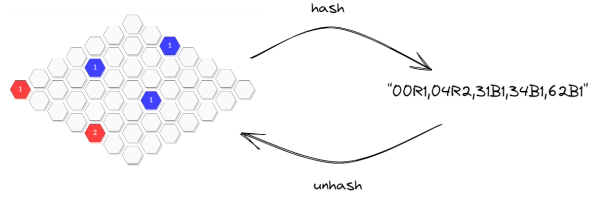
We created an additional program in `./evaluate.py` that plays 100 matches between two agents and evaluates the win rate percentage of each agent as well as the number of moves it took for games to reach terminal states.

4.2 Monte Carlo Tree Search

Other than the minimax algorithm, we also experimented with Monte Carlo Tree Search as described in the lectures and Figure 4.2a; the code of which can be found in `./mcts/mcts.py`.

Despite our efforts of varying the number of MCTS simulations, precomputing simulations and storing states in a csv transposition file for lookup during runtime, and varying different simulation techniques; we found that our MCTS Agent was constantly outperformed by our Minimax Agent. This, in addition to bottlenecks in terms of time and space complexity (ie. not enough time to compute a reasonable number of simulations

and not enough memory to store a reasonably large number of precomputed states), forced us to abandon our MCTS approach and focus on Minimax.

<p>Algorithm 1 General MCTS approach.</p> <hr/> <pre> function MCTSSEARCH(s_0) create root node v_0 with state s_0 while within computational budget do $v_l \leftarrow \text{TREEPOLICY}(v_0)$ $\Delta \leftarrow \text{DEFAULTPOLICY}(s(v_l))$ BACKUP(v_l, Δ) return $a(\text{BESTCHILD}(v_0))$ </pre> <hr/>	
<p>Figure 4.2a: Pseudocode of an Iteration of MCTS (Browne et. al, 2012, p.6)</p>	<p>Figure 2.3c: Hashing and Unhashing performed to minimise memory required to store states in the transposition csv</p>

5. Conclusion

In this report, we have presented a design for an artificial intelligence agent to play the board game Infexion. The minimax algorithm was used as the core of the agent's decision making process, and has been enhanced with alpha beta pruning and transposition table to improve its efficiency and performance. It was shown that the minimax agent can play the game at a high level of skill, and performs better than baseline, naive agents. We hope that this report can inspire further research and development in the field of artificial intelligence and board games.

References

- DeepMind. (n.d.). AlphaGo. Retrieved [8 May 2023], from
<https://www.deepmind.com/research/highlighted-research/alphago>
- Deep Blue computer beats world chess champion (1996, February 12). *The Guardian*.
<https://www.theguardian.com/sport/2021/feb/12/deep-blue-computer-beats-kasparov-chess-1996>
- Qi, Z., Huang, X., Shen, Y., & Shi, J. (2020). Optimization of Connect6 based on Principal Variation Search and Transposition Tables Algorithms. In 2020 *Chinese Control And Decision Conference (CCDC)* (pp. 198-203). Hefei, China: IEEE.
<https://doi.org/10.1109/CCDC49329.2020.9163922>
- Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1-43. <https://doi.org/10.1109/TCIAIG.2012.2186810>