COMP 472 Project 1 Report

Alexia Soucy[1]

[1] 40014822 a_oucy@live.concordia.ca

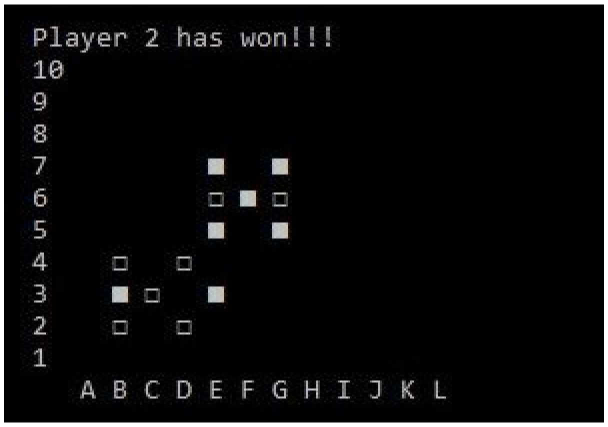
1 Introduction

This report outlines the development of an AI capable of playing a game of X-Rudder against a human player.

1.1 Application

The game of X-Rudder employs a 10 by 12 grid with numbered rows and alphabetized columns (see Fig. 1). Two players have 15 tokens each, in the form of full squares (■) for Player 1 and empty squares (□) for Player 2. The game can be played by a human player against a human or AI player. The players take turns either placing one token on any unoccupied position on the board or moving one token they own to any unoccupied adjacent position (including diagonally). The players can move tokens 30 times between them during a match.

Each player's goal is to form a 3 by 3 X on the board with their own pieces. An opponent's X can be crossed by placing two of one's own tokens on both sides of the X's central token (see Fig. 1). If a player moves a token in such a way that both players have a winning X on the board, the player who made the move wins. Should both players run out of tokens and collectively exhaust their 30 move limit without a winner, the game is declared a tie.



```
Player 2 has won!!!
10
9
8
7          ■   ■
6          □ ■ □
5          ■   ■
4    □   □
3    ■ □   ■
2    □   □
1
   A B C D E F G H I J K L
```

Fig. 1. A snapshot of the X-Rudder board, with a winning X by Player 2 and a crossed X by Player 1

1.2 Technical Notes

The application is written in Python 3.7.4 and uses the NumPy library for its array capabilities. This allows for the game board to function as a two-dimensional array. Additionally, the application makes use of Python's time module to measure the time taken by the AI each turn.

2 Heuristic

This section outlines the logic, implementation, and optimization of the AI. The core of the AI's heuristic is a minimax algorithm processed through a search tree of nodes each representing one gamestate with its relevant values.

2.1 Gamestate Evaluation

The gamestate is evaluated by breaking down the grid into overlapping 3 by 3 subgrids in order to consider every possible location for an X.

For every subgrid, the number of tokens for each player in the X positions (in the corners and center) is tallied. If both players have tokens in these positions, the subgrid's value is considered to be 0 since it is impossible for the current subgrid configuration to result in a win for either player.

Once a subgrid is found where only one player has tokens forming a partial X, the subgrid's value is considered to be the factorial of the token count, as seen in (1) where *n* is the number of tokens forming the partial X. For example, a partial X containing 4 tokens is worth 24 points. A complete X of 5 tokens is worth infinite points, denoting a potential win (if the X is not crossed).

$$V_{sub} = n! \tag{1}$$

Once a partial or total X is found and its value is calculated, the number of tokens crossing the X is tallied for each player. That is, the number of tokens on the first and third position of the subgrid's middle row. Each player having one such token or neither player having one has no effect on the subgrid's value since crossing the X is either is impossible or not in progress. If the player opposite the one whose tokens form the X has one token crossing the X, the subgrid's value is halved. If that player has two such tokens, the subgrid's value is 0 since the X is crossed and cannot result in a win.

If the player whose tokens form a partial or total X is the minimizing player, the subgrid's value is subtracted from the board's value. Otherwise, the subgrid's value is added to the board's value.

This algorithm continues until it has processed each subgrid. Finding a winning subgrid for a player does not abort the algorithm since a move can make both players win. This decision is justified further in subsection 2.3.

2.2 Optimization

The AI was required by project specifications to have a decision making delay of less than 5 seconds. After making it play multiple games against itself, it was found that the average delay with the minimax algorithm outlined above was 15.61 seconds with a search depth limit of 2.

The implementation of alpha-beta pruning, wherein branches that mathematically cannot provide an acceptable result are pruned [1], cut the delay down to an average of approximately 3.27 seconds, but the AI still took longer than 5 seconds to make decisions with increased frequency when approaching the halfway point of the game before eventually falling off.

To further optimize the algorithm, an alternative evaluation process was employed. In a typical minimax algorithm, the leaves of the search tree are evaluated and their value backpropagated as necessary. [2] In this case, the algorithm evaluates the tree's root, then passes that value on to its descendants recursively. Each descendant that is not a leaf updates its own value by only considering the subgrids that surround the last move made.

In total, 7 subgrids are processed since the algorithm must consider the effect of the move on corners as well as the center of potential Xs as well as determine if it is crossing an X partially or completely (see Fig. 2). This is repeated twice for a placement, to undo those subgrids' values before the token's placement before adding in the new value. For movements, the process is repeated to account for the token's previous position.

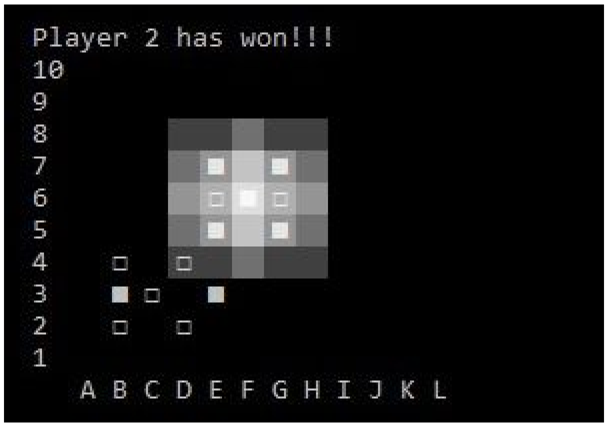

```
Player 2 has won!!!
10
9
8
7
6
5
4
3
2
1
   A B C D E F G H I J K L
```

Fig. 2. A graphical representation of the overlapping subgrids evaluated per update at 6F.

The number of subgrid evaluations needed by the standard and optimized algorithms can be predicted through complexity equations (2) and (3) where n is the number of children each node has, d is the search depth limit, m is the number of subgrids needed for a full evaluation, j is the number of subgrids necessary for an update, r is the number of repetitions is needed to properly update the value (2 for placements, 4 for movements). Due to the difference between m and rj, C_{opt} generates a value one order of magnitude less than C_{std} at a search depth of 2.

$$C_{std} = n^d \times m \tag{2}$$

$$C_{opt} = n^d \times rj + m \tag{3}$$

Once leaves have updated their own values, they then backpropagate it as necessary, updating the tree per the minimax algorithm with alpha-beta pruning.

In addition to this, the process of generating a node's children recursively was changed to explore options starting at the board positions closest to the last move, based on the assumption that this could provide a slight increase in speed as moves made in reaction to the opponent's might have a greater impact on the gamestate and therefore optimize the alpha-beta pruning process. These changes lower the average decision delay to 0.22 seconds at a depth limit of 2, but were not impactful enough to limit the delay to 5 seconds at a depth limit of 3. Therefore, the depth limit was not changed.

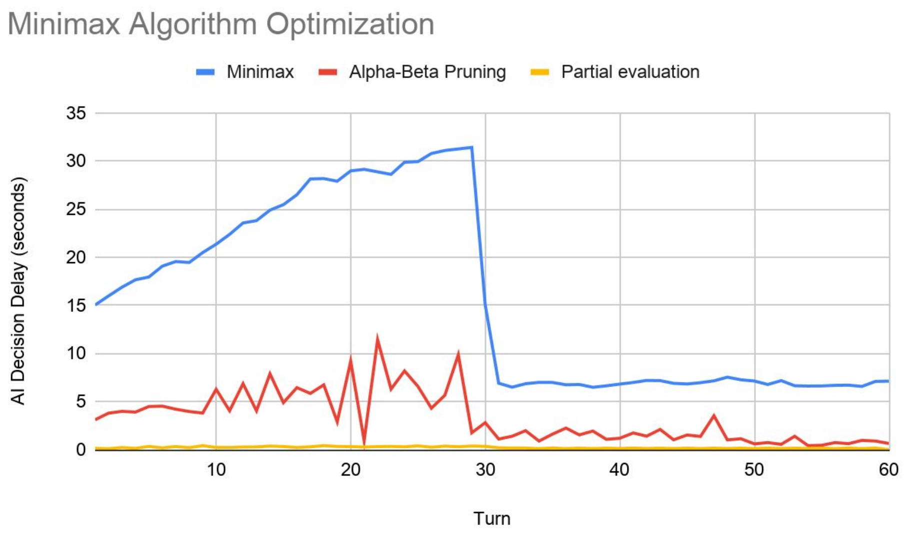


Fig. 3. Minimax Algorithm Optimization methods

2.3 Considerations

During development, there was consideration of making the optimized evaluation scheme take into account winning states by the opponent to abort the evaluation algorithm, but this would have ignored token movements that would have provoked a win for both players and could have generated false negatives (and the total board value would have become 0, which may not be considered a winning state by the heuristic). Additional logic checks could have been implemented to only discard such a branch if it was not provoked by a

Among other optimization methods considered was to set a greater search depth limit, which would normally go over the time limit of 5 seconds per turn, and make the AI abort its search and pick its best option so far once the time limit was reached. This was based on the assumption that the reactive tree ordering mechanism described in section 2.2 would make the best choices appear at the start of the search and the later choices would be unnecessary to explore. In practice, however, this caused the AI to make suboptimal choices.

The other attempted optimization method was to prune a branch as soon as a child was found to be a winning move for the opponent, but this did not provide any observable improvement in performance.

3 Results

The AI was tested for its functionality and behaviour first against a human player at each step of its development and then in its final form in a tournament setting where it played against 3 different AI. This section outlines the results of this testing.

3.1 Human vs AI Testing

The AI was tested against a human player for its behaviour and it was found to behave in a suitably strategic manner, often times prioritizing its own X formation until the human player has a partial X made up of 3 tokens.

It places a high amount of priority on placing new tokens rather than moving them, but will move tokens on occasion if that is optimal. In practice, however, it has a tendency to expend its 15 tokens in the first half of the match, and then to move tokens in a defensive manner.

Against a human, this presents a fatal flaw as the human player can have the foresight to place tokens in a relatively loose cluster to form an X after the AI runs out of tokens. Alternatively, the human player can conserve its tokens and prioritize movements to face little resistance from the AI in the later portion of the game. Given more time, this would have been remedied by altering the heuristic's weight on movements versus token placements.

3.2 AI vs AI Tournament

In the tournament, the AI played a total of three rounds against different AIs, each composed of two matches where player 1 and player 2 exchanged places.

In the first round, the AI won both matches by default, as the opposing AI made illegal moves in both games.

In the second round, the AI played against an AI with a search depth of 4, employing a hard time limit as described in section 2.3. In both games, player 2 won.

In the third round, the AI played against an older version of the above AI that had a search depth of 2 instead, but the algorithm proved to be flawed and this AI won both matches early on, leaving it with a total of 5 victories and 1 defeat. It behaved as expected and no problems were encountered.

4 Challenges

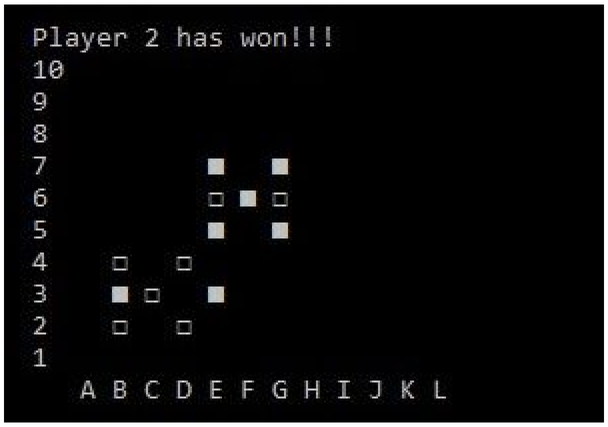
The main challenge faced during this AI's development was minimizing its decision delay to never surpass 5 seconds. This proved quite difficult and at times counterintuitive when alpha-beta pruning, the conventional minimax optimization scheme, had been used and proved insufficient. This was finally overcome as outlined in section 2.2.

Additionally, finding an adequate strategy for the heuristic was difficult at first. Counting partial Xs and taking crossings into account proved to be the most effective approach.

5 References

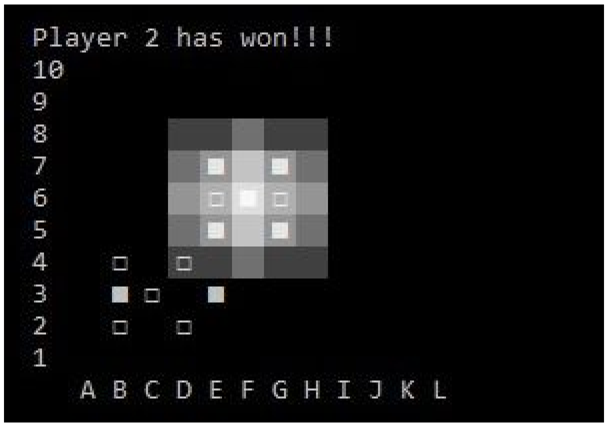
1. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach. 3rd edn., pp. 165, Pearson Education, Upper Saddle River, New Jersey 07458 (2010)
2. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach. 3rd edn., pp. 167-169, Pearson Education, Upper Saddle River, New Jersey 07458 (2010)

6 List of Figures



```
Player 2 has won!!!
10
9
8
7          ■   ■
6          □ ■ □
5          ■   ■
4    □   □
3    ■ □   ■
2    □   □
1
   A B C D E F G H I J K L
```

Fig. 1. A snapshot of the X-Rudder board, with a winning X by Player 2 and a crossed X by Player 1



```
Player 2 has won!!!
10
9
8
7          ■   ■
6          □ ■ □
5          ■   ■
4    □   □
3    ■ □   ■
2    □   □
1
   A B C D E F G H I J K L
```

Fig. 2. A graphical representation of the overlapping subgrids evaluated per update at 6F.

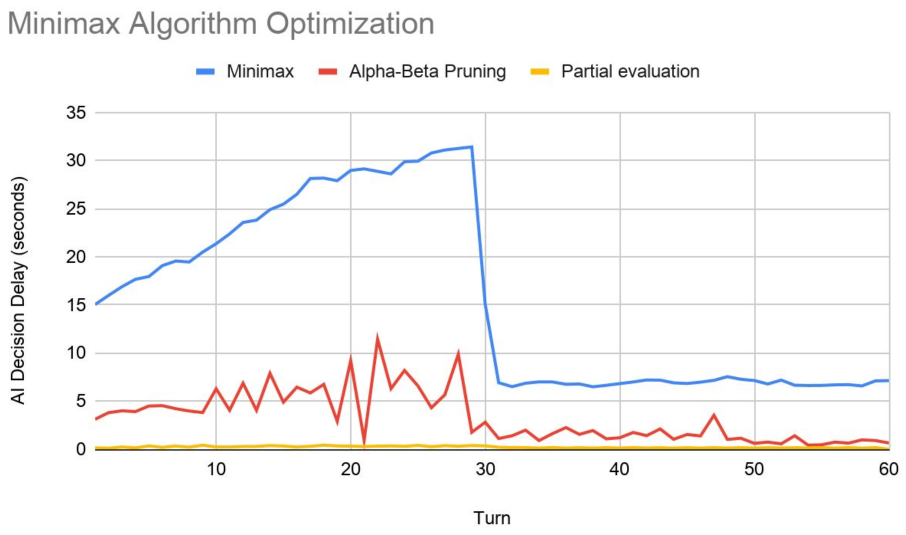


Fig. 3. Minimax Algorithm Optimization methods