COMP 472 Project 1 Report

1 Introduction

1.1 Overview

For this project, we used Python to first create a game called X-Rudder. X-Rudder is a two-player game where players take turns placing or moving tokens on a 12 by 10 game board with the goal of creating an X shape made up of 5 tokens on the board without having 2 adversary tokens on its left and right causing a strikethrough. Players have a limited number of tokens to place (15 by default) and limited number of times they can move tokens on the board which is shared by both players (30 total moves by default). Players can place tokens on any open space in the board and moving tokens must be done to an open space that is one space in any direction of the token being moved (including diagonals).

The second aspect of this project was to then create a heuristic based AI, that could play against another player. Given this is a two-player, adversarial game where both players have the same potential moves, Minimax was the specified algorithm to use when state space searching for the AI's next move.

The final aspect of this project was to prepare the heuristic to the best of our ability and test it against the Ai that other students have created. This allowed us to test and analyze our Ai and it's heuristic and how successful it was in the real world

1.2 Technical Details

- Python 3 (3.7 to be specific) was used to create the project.
- The Numpy Library was used to create a matrix which allowed for the representation and manipulation of the game board
- Jetbrains Pycharm IDE was used to develop and debug the project

2 The Heuristic

2.1 Description

To create the heuristic, 2 main aspects had to be considered when creating it. The first aspect was its ability to get as much information from the state of the board so that the Ai could play well consistently. The second part was to make sure it was performant so that the Ai could stay under the 5 second rule and not become disqualified for taking too long.

How I came up with the heuristic

One interesting thing to note about this game is that at a high level, it is quite like tic-tac-toe. For this reason, the basis of the heuristic was built off the general idea shown to us on the lecture slides for adversarial search:

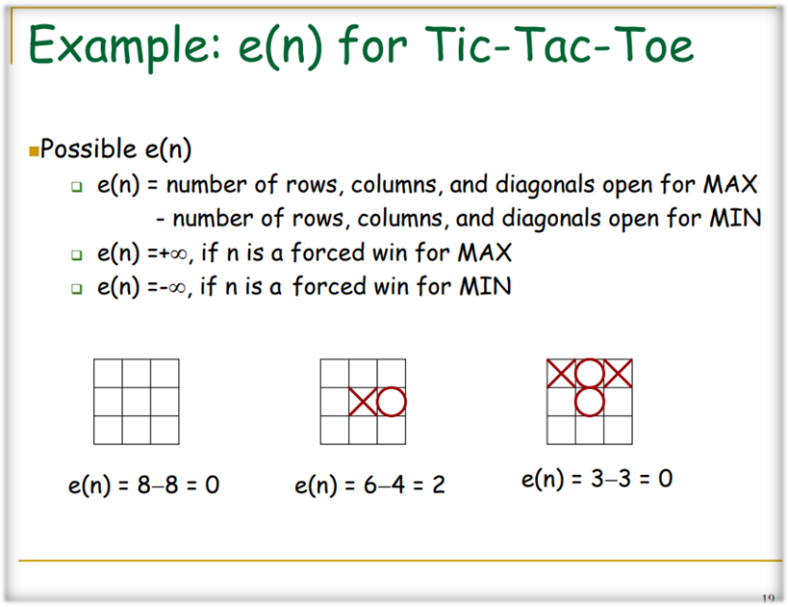

Fig. 1. Slide 19 from adversarial search. Basis of the idea of my heuristic.

The rest of the ideas I came up with (such as weighing the features, further added features, and performance optimization) were mostly found after creating the base heuristic.

Game State

The ability for the Ai to make an informed decision relies upon the data it has that it can analyze so that it could make an informed decision. Here is what I generally stored:

- Game Board
- Total moves left for players
- Players
    - Number (first player, second player)
    - Number of tokens left
    - List of location of tokens added to the board
- Current player's number

The only information left out is the current number of turns, since that does not add any valuable information as to how the Ai should approach the game.

Features and Justification

Completion Status:
The most important thing the Ai checks over anything is if the state being evaluated gives it a win, a loss, or an end game draw. If the state is a win, the state is given infinite score, but if it's a loss, the state is given a negative infinite score. Since the goal is to win the game, this is checked before doing a more general approach to scoring the state.

Number of Player Tokens on Board:
The player is given a small score boost for adding tokens to the board. This is to encourage the Ai to place tokens at the start of the game. Around halfway through, this changes to score in the opposite matter, more tokens causing lower score. This is to encourage the Ai to instead save tokens if an equivalent move is good enough, so that the Ai can allow the opponent to use up its tokens first and take advantage of the situation

Number of Possible Wins and Closeness to Win (Main Heuristic):

This is my main heuristic, and the one I implied was inspired by the tic-tac-toe slide. Basically, for any token on the board, I look for all possible positions of it such that it can give the player a win. The closer it is to creating the X shape (the number of tokens in the assumed token position), the greater the score is amplified (exponentially). If the possible win configuration is covered by opponent tokens, a negative score is amplified. A small positive score is given for open spaces, while negative score is also added for spaces that are out the bounds of the board.

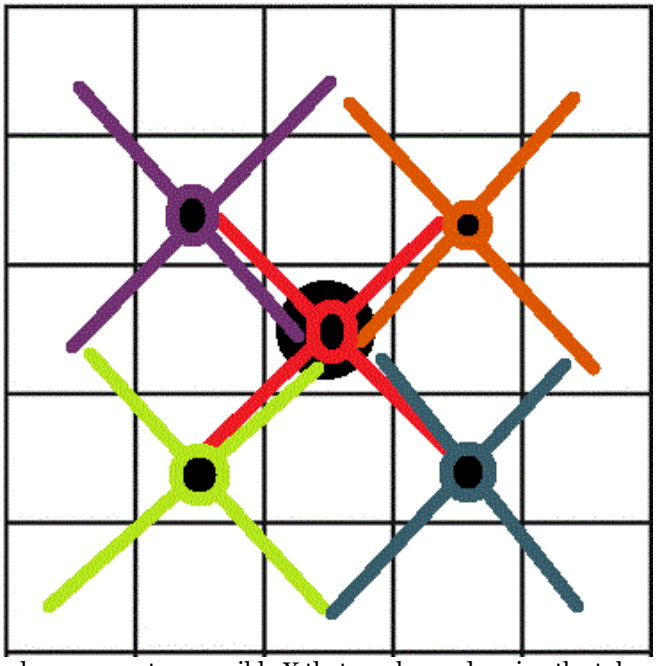

Fig. 2. Each color represents a possible X that can be made using the token at the center.

This is done so that the Ai pushes to create an X shape which is what will allow it to win the game. It also lets the Ai determine how close the enemy player is to creating X shapes making it more defensive in such scenarios. This also favors token placements such that the maximum win possibilities are created (for example, an edge of the board is disfavored since less X shapes can be created.

Pyramid Strategy Closeness:

In this game, there is a possible configuration, that allows players to create a scenario such that the other player can only block one of two possible wins. While it is great to get closer and closer to an X shape, once the shape is found it can easily be blocked. I added this heuristic and added a heavy weight on it so that my Ai would try to set up tokens in such a way to make this scenario occur. I figured this out when I noticed my Ai was doing something similar with my main heuristic when I set higher depths. So, in turn, this was essentially a compromise and way to add a strategy the Ai seemed to approach without the added performance hit of too much depth.

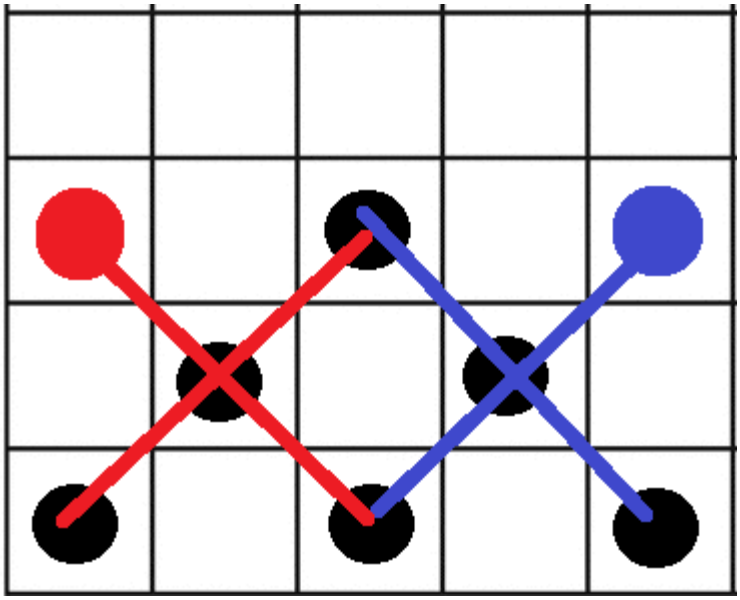

Fig. 3. Given the configuration of the black tokens (The Pyramid), there are 2 win possibilities here

Further discussions about optimizing performance is described in the “Difficulties” section below.

3 Analysis

3.1 Personal Results and Analysis

When playing with my Ai on its own, I thought it was working ok. Playing with it in general helped me improve it along the way. In general, I found that my Ai was good at making sure it didn’t lose, and if I played in the expected optimal matter, I would generally tie with my Ai. The main blind spot I found about the Ai though was when I used an unexpected strategy which the Ai was not really looking for.

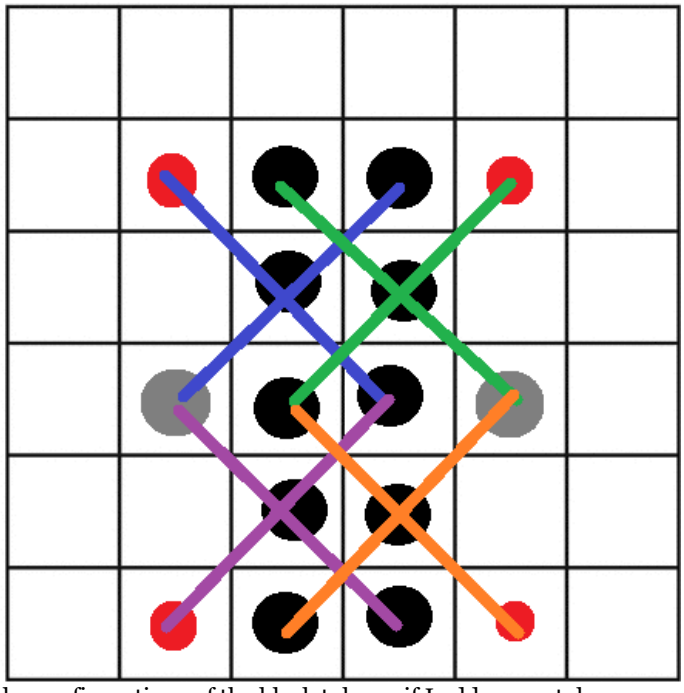

Fig. 4. Given the configurations of the black tokens, if I add a gray token, many possible wins come up

I think what ended up happening was that the above configuration is not amazing in the eyes of my Ai since the pillar set up of tokens doesn't really create many potential X shapes. Once a gray token is added as shown above though, the Ai is at a forced loss.

I think overall this was an ok compromise as the AI assumes an optimal opponent and this configuration of tokens isn't really great for a normal game, but valuable if specifically playing against this AI.

3.2 Tournament Results and Analysis

Teams played	AI Starts First Result	Ai Starts Second Result
1	Draw	Win
2	Win	Win
3	Win	Win

Table. 1. Tournament Results

In the tournament, my Ai did pretty good. Oddly it seemed to play better when it started second, and I'm not sure why. I think the fact that my Ai was strict on ensuring a it didn't lose was helpful, since it seemed to mainly win once the placing of tokens was exhausted. While the Ai was bad at going out of its way to find a win in such a state with all tokens on the board, it was good at letting the other Ai do so until it found an opening as a result of the other Ai trying to find such a win. The strategy of saving tokens halfway through also helped the Ai having tokens left to add after the other Ai exhausted its 15 tokens, although the weight could have been stronger to save more tokens as it usually only saved around 1 or 2.

4 Difficulties

The main struggle with the Ai was trying to get the Minimax to depth 3. Going from a depth of 2 to 3 was important in helping improve its decisions. Here are the compromises and strategies I used to help with performance. In general, the bad performance was not the result of the heuristic calculation, but simply the result of possible state explosions. I figured that was such when I replaced my heuristic with a random equation only, and generally getting similar performance.

- Alpha-Beta Pruning was implemented. This was helpful in reducing the number of generated states in the Minimax tree
- The Ai won't consider adding tokens if players run out of tokens, this both improves performance and prevents errors
- The Ai only considers moving of its own tokens once it places half its tokens. Once both players exhaust tokens, it considers all possible movements. This was done because moves create a lot of states and are generally less useful at the start of the game.
- The Ai only considers adding tokens that are near other tokens only. This heavily reduces the number of states created which always tended to be useless as the Ai seemed to always add tokens to get closer to an X or block the opponent, both of which rely on adding tokens near others
- The Ai only adds token in diagonal positions around tokens. This greater than halves the potential additions near tokens, since the board generally resembled a bunch of diagonals. Since my Ai was always choosing such moves, I added this condition and was able to keep a consistent depth of 3 with a good enough calculation time

5 Team Contributions

I worked alone, so work didn't have to be divided up across multiple team members. I did want to note though that I took an iterative approach when adding code to the project and used local code history to manage my changes.

References

The only thing I referred to was that one slide in the lecture. Everything else simply came from coming up with my own ideas or learning from playing with the Ai. So, I have no references.