COMP 424 - Project Report

Julien Courbebaisse- 260614548

4/10/2019

Introduction

This report will go over the development process and associated design decisions for the Pentago-Swap autonomous agent. Three agents were developed for this project: a Monte Carlo Tree Search agent, a Minimax agent, and finally a logical agent. The latter logical agent was the one chosen in the end for submission because of its high win rate against both other agents as well as strong performance against human players. The logic behind the agent's decision process will be thoroughly explained throughout this report, and proposed improvements will be made. The report will also briefly go over our Minimax and Monte Carlo agents, although in less depth as they are implementations of classic algorithms with minimal transformations.

A - Logical Agent

1- Motivation

The idea for this agent came from playing numerous games myself. Since this agent was developed last, I had had the opportunity to play many games against the Random Player, Monte Carlo Player, Minimax Player, and other Human Players. This gave me insight about the different ways to win at Pentago-Swap, which are not as numerous as one may think. In other words, the agent relies on a Board Analyzer class to make calculations based on the current state of the board and outputs the best possible move. The logical basis for this agent can be found in depth in the next section.

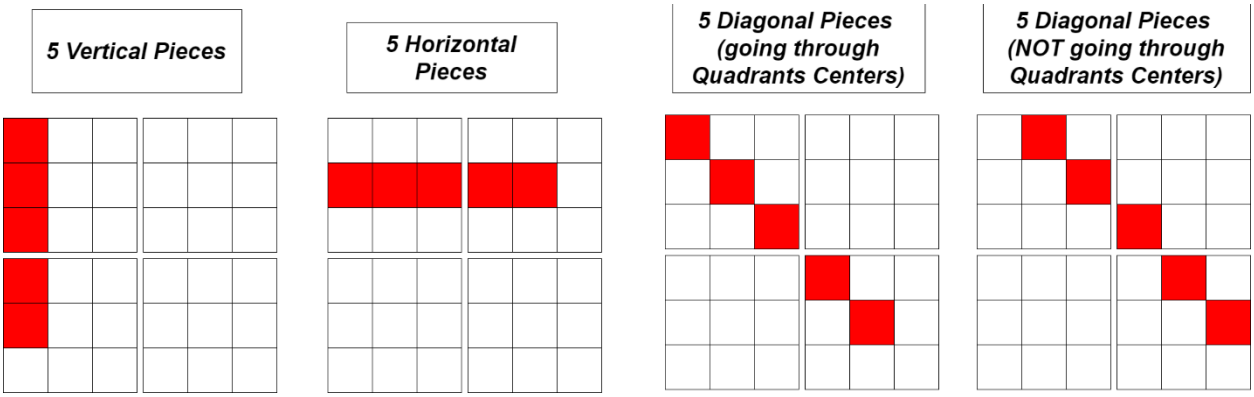
2- Logical Assumptions and Decision Algorithm

2.1 – Basic Insight

The assumptions which form the basis of the logical reasoning rely almost solely on where to place the next piece, not so much how to swap them. To explain why we will first list all the possible ways of winning a game:

- 5 vertical Pieces → 3 in one quadrant and at least 2 in another
- 5 horizontal Pieces →3 in one quadrant and at least 2 in another
- 5 diagonal Pieces WHERE at least one piece goes through the center of a quadrant → 3 in one quadrant and at least 2 in another
- 5 diagonal Pieces WHERE no piece goes through the center of a quadrant → 2 small diagonals in two quadrants (ie diagonals of length 2 which do not go through the center of any quadrant) and one corner piece to link them

Find below a picture of each winning strategy with an example of the pieces associated with it.



2.2 – Assumptions at a Micro Level: Assigning Move Values with Respect to Quadrant

What we have discovered in 2.1 is that there seems to be strong correlation between winning and having 3 pieces lined up diagonally, vertically, or horizontally within the same quadrant. There is also a correlation between having 2 sets of matching "small diagonals" in two different quadrants, which means there must be a correlation between winning and having 1 small diagonal as well. Another assumption is that at each point we are making a play to win in only one of those combinations.

From these assumptions our Board Analyzer calculates a Potential Score for each move for each winning scheme: Vertical, Horizontal, and Diagonal. Notice that for this step of the algorithm the diagonal potentials aggregates both Diagonal wining schemes for simplicity but will adjust later during the macro game analysis where the whole board will be considered. These potentials are stored in 3D integer arrays so that the game board is broken down into quadrants.

```
private int[][][] verticalPotentials = new int[4][3][3];
private int[][][] horizontalPotentials = new int[4][3][3];
private int[][][] diagonalPotentials = new int[4][3][3];
```

I will try to explain as clearly as possible how the potential for each move is calculated for each case. Here is the general insight on how this works. If we look at a blank square in the game and try to calculate its Vertical Potential then we look at the other two pieces in that column, in that quadrant of course. If they are both empty, then we can conclude that although this column is empty, we could still use it to line up 3 pieces and therefore its vertical potential is 3. However consider the case that we look at the column and see that we already have one piece and that the other piece is empty. Then the potential could be 3, but obviously the value of this move should be bigger than the value of the previously considered move since it would get us more pieces in the same row. From this we derive a formula that says value = (max potential number of pieces line up) + (number of pieces already lined up). So in our later case we could still potentially align 3 pieces and we already have 1, therefore our vertical potential is 4 for this piece.

We need to add one final check to this. Consider the example for vertical potential again. If the middle piece of the column is owned by our opponent, then the vertical potential for the remaining pieces in the column will be 0. This is because we know that we will be able to line up at most 1 piece, and we know from 2.1 that a single vertical piece in a quadrant has no winning potential from our own perspective. For complete Vertical Potential calculations, the reader is encouraged to check the UpdateVerticalPotentials() method since some bias was later added to those values but the main logic remains the same.

The Horizontal Potential calculation works similarly to the Vertical Potential calculation and we will assume it does not need to be reexplained here. Basically it is still value = (max potential number of pieces line up) + (number of pieces already lined up), for each Piece.

Now the diagonal potential is a little different because it aggregates potentials for 6 winning schemes. With the same value equation as before, we start by calculating the potential score values for a Negative Slope Diagonal (top left, middle, bottom right). The calculated values are added to the diagonal potentials. Now the same thing is done for a Positive Slope Diagonal (bottom left, center, top right). Again the values are added. Now the 4 remaining Pieces(left middle, top middle, right middle, and bottom middle) are all scanned for “small diagonal” potentials (having the pieces lined up diagonally that do not go through the center of the quadrant). There are 4 of those potential diagonals to check for, and we assign values with the same formula as before.

To clarify all this information, see below printouts of the potential values the board analyzer calculates at different stages of the game:

Stage 0: There are no pieces on the board yet

```
Printing Board
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
-------------------------

Printing horizontal potentials
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
Printing vertical potentials
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
Printing diagonal potentials
|3|2|3|3|2|3|
|2|6|2|2|6|2|
|3|2|3|3|2|3|
|3|2|3|3|2|3|
|2|6|2|2|6|2|
|3|2|3|3|2|3|
```

Stage 1: There are is 1 piece on the board (which is not ours)

```
Printing Board
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   | w ||   |   |   |
-------------------------

Printing horizontal potentials
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|2|2|0|3|3|3|
Printing vertical potentials
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|2|3|3|3|
|3|3|2|3|3|3|
|3|3|0|3|3|3|
Printing diagonal potentials
|3|2|3|3|2|3|
|2|6|2|2|6|2|
|3|2|3|3|2|3|
|2|2|3|3|2|3|
|2|5|2|2|6|2|
|3|2|0|3|2|3|
```

Stage 2: There are is 2 pieces on the board (we are white here)

```
Printing Board
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
-------------------------
|   |   | b ||   |   |   |
|   |   |   ||   |   |   |
|   |   | w ||   |   |   |
-------------------------

Printing horizontal potentials
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|2|2|0|3|3|3|
|3|3|3|3|3|3|
|4|4|0|3|3|3|
Printing vertical potentials
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|3|0|3|3|3|
|3|3|3|3|3|3|
|3|3|0|3|3|3|
Printing diagonal potentials
|3|2|3|3|2|3|
|2|6|2|2|6|2|
|3|2|3|3|2|3|
```

```
|4|2|0|3|2|3|
|2|6|2|2|6|2|
|2|2|0|3|2|3|
```

Notice that at each point the value of the piece in the corresponding scheme is value = (max potential number of pieces line up) + (number of pieces already lined up). Except for the diagonal center piece which has the potential of lining up 3 in two different diagonals, and therefore its value is 6.

2.3 – Superposing our Potentials

Before we start analyzing the board at a Macro Level we need to aggregate our calculated potentials somehow. Two implementations were tested to determine the best way: superposing and summing the potentials at each piece. It turned out that the summation gave results that were inconsistent at times with the best next move to play. In turn I opted for a superposition of all potential matrices, where the strongest potential value is saved for each piece. This showed good results and few errors in terms of piece placement priority. This gives some validation to one of our hypotheses which is that at each time we can only try to win in one given direction and that the best move amongst all those schemes should be chosen. This idea is more straightforward than the previous section but see below an example anyways:

Stage 0 : no Pieces placed

```
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
-------------------------
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
|   |   |   ||   |   |   |
-------------------------

Printing superposed potentials
|3|3|3|3|3|3|
|3|6|3|3|6|3|
|3|3|3|3|3|3|
|3|3|3|3|3|3|
|3|6|3|3|6|3|
|3|3|3|3|3|3|
```

2.4 – Updating Potentials with Macro Analysis (considering other quadrants)

This part focuses on updating our superposed potentials matrix by looking at all the quadrants. Again this is done on 3 levels: Diagonal, Vertical, and Horizontal. We will start by explaining how our Vertical Macro Potentials are updated, which works in the same fashion as our Horizontal Macro Potentials.

First our algorithm will look to check if any Quadrant has 3 vertically aligned pieces (our color) using a method called has3Vertical(). Skip through details, this method will determine if we have 3 vertically aligned pieces in any quadrant. If such combination is found, then it will return the quadrant index as well as whether the aligned Pieces were in a left, right, or middle column. If it does not find any such combination, we simply do not update our potentials. If we do find one however, then we know that

we are in a good place to win, and now all that is left to do is find another quadrant where we could align 2 pieces vertically in the same column. Now following our value equation from the Micro Analysis, we create a new value function where the max potential value is now 5. In other words, all the middle pieces of the corresponding column in other quadrants are given a lot of priority, a lot of value is added. The other pieces in those corresponding columns are also given more priority but not as much since a middle piece can align 2 pieces ion 2 different ways which has more utility. Exact potential increments at this point use a little more bias to be practical in a real game but the idea is generally the same. Note that Horizontal Macro Potentials works in the same manner and will not be reexplained.

Diagonal Macro Potentials work a little differently. Again it covers 6 winning schemes, our 2 Positive and Negative slope Diagonals, and the 4 small diagonal schemes (which do not go through any quadrant centers). The Positive and Negative Slope Diagonal work in the same idea as the Horizontal and Vertical Macro Potential updates, by increasing the potential of moves on corresponding diagonals in other quadrants. The small diagonal macro updates work a little differently. First note that our agent is not naturally inclined to playing this way and will rarely put itself in a position where its tries to win through these schemes, however you will see later that keeping track of this is still very useful from a defensive perspective. In short, if we detect that we have 1 small diagonal (2 pieces) in one quadrant and 1 other piece in another quadrant with the potential of making another corresponding small diagonal, then we give a lot of priority to form that second corresponding small diagonal because it is a good indicator that we will win once we have the two small diagonals. At that point we would only need the 1 corner piece to win.

This is the end of our board analysis process, at least in terms of potential calculation from an offensive perspective. The piece with the highest potential score would be our best offensive move.

2.5 – Defense through Switching Perspectives

The motivation for this phase is that the algorithm worked quite well if its opponent did not try to beat it playing very greedily. For example, if it did not start and its opponent directly tried to line up 5 pieces on the bottom row, then it would proceed to playing offensively and lose. I tried to think of defensive schemes to implement but everything I tried either broke the potentials system or did not work. Then I realized that if I was the other player then those greedy moves would quickly skyrocket my potential values. This is where switching perspectives comes in. We save our calculated potentials and refresh all our potential matrices and we switch the color the algorithm thinks it plays with.

Now the algorithm reruns steps 2.2-2.4 from the opponent perspective. We now have 2 full potential matrices from both perspectives which we superpose to obtain a final potential matrix (keeping the biggest value like in 2.3). This now means that if a move would get an opponent closer to winning than any move we could play ourselves, we will just occupy that spot to block our greedy opponent. This makes the Agent a strong tic tac toe player where it will try to not allow the opponent to align 3 pieces in a quadrant. Furthermore, it would also block our opponent from trying to win through a greedy “small diagonal” scheme.

2.6 – Finding Best Swap and Returning a Move

At this point the algorithm has a good idea of where to place the next Piece. However, it must also decide on a swap to play. First note that at the beginning of each turn, our algorithm will check all the moves it can play and directly return that move if it is a winning move. In that regard, the algorithm only needs to be 1 swap away from potentially winning, if that makes sense. Second the algorithm will also never return a move if this move is "suicide" i.e. if the other player could win in one move following this move (except of course if it is already doomed from a minimax perspective).

The algorithm then has a basic swap mechanism. It is based on the idea that a Negative Slope Diagonal is only useful if it is in the top left or bottom right quadrant, and vice versa. The algorithm will therefore give more priority to a swap which achieves this, otherwise it will return any random swap.

3- Algorithm weaknesses and Proposed Improvements

The agent is overall quite strong and hard to beat as a human. However, one of its weaknesses is its swapping mechanism. The current mechanism is very basic and there are many more factors that would need to be taken into account. I think a complete swapping potential mechanism could be implemented. Another weakness of this agent is that some bias was used in calculating potentials, both at a micro and macro level. I believe a genetic algorithm approach with mutations and natural selection could help strengthen the agent and calculate the most rewarding potential values for all the above-mentioned combinations and pieces.

B- Monte Carlo Agent

This agent uses a classic Monte Carlo Tree Search algorithm to sample outcomes from each move. It uses a UCB1 score to balance exploration and exploitation. The first 4 moves are programmed to play in the quadrant centers if possible. Also if it sees a winning move it will return it. It beats the random player around 90% of the time and wins around 5% of games against the logical agent too.

C- Minimax Agent

This agent uses a classic Minimax algorithm. However, it showed very poor performance and development was not continued past a basic implementation. The poor performance can probably be explained by the fact that different subtrees can have very different depth and the approach was just not practical for this game given the time limit imposed on each move.

Summary and Conclusion

The logical agent developed performs overall quite well. It is a tic tac toe master, where it will almost never lose in a greedy battle. It executes its decision making process very quickly, considered the amount of calculations done. It is very good at recognizing where to place the best piece, opting between defense and offense each time. The values associated with each move however are sometimes not perfect. I have identified that almost all losses by this agent come from the opponent playing in the bottom right quadrant. That is because the algorithm chooses the first move with max value and will sometimes fail to recognize that it should play defensively, and that is because it does not know how to choose a move if multiple moves have the same value at the end of the calculations. This being said it performs well against the other two agents developed and it generally knows how to reconcile offense and defense.