COMP 421
Final Project Report

Houkai Qian 260849921 April 13, 2020

Professor: Jackie CK Cheung
McGill University

Contents

Introduction

Pentago-Twist is a variant of game Pentago. The game is played on a 6×6 board, and the board could be divided into four 3×3 sub-boards (quadrants). Two players each holds a color pieces(either black or white), taking turn, place a piece on an unoccupied space on the board, and then rotate one of quadrant by 90 degree clockwise or flip the quadrant horizontally. A simple example is showed in Figure 1 and Figure 2[1]. The goal of the game is to have 5 pieces in a line after a player's turn. If no player wins when the board is full or two player wins at same time, the game is draw.

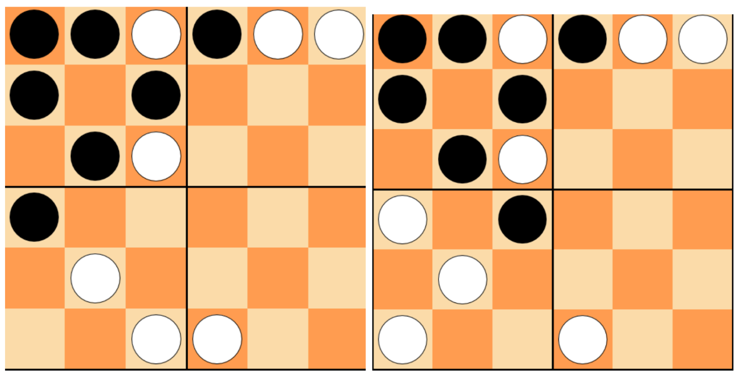


Figure 1: White player places its move at (0,5) and choose the third Quadrant to rotate 90 degrees clockwise.

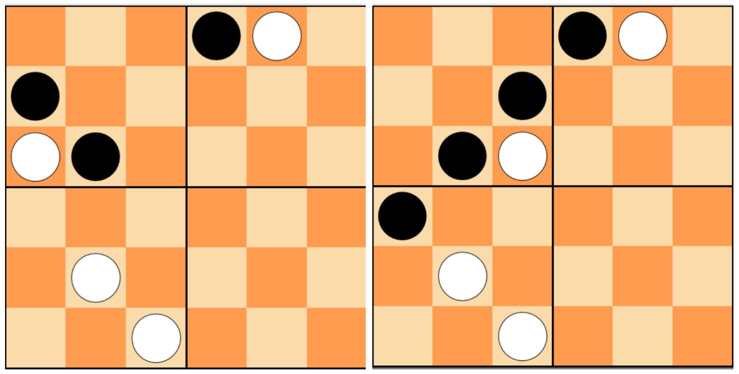


Figure 2: Black player places its move at (0,3) and choose the first Quadrant to flip horizontally.

For reference, original Pentago game is also analyzed for this project. Although Pentago's rules are simple, the game states are hard to be predicted by human because of the involvement of quadrants rotations and flipping. Therefore Pentago and similar games are ideal for developing AI. Pentago has been strongly solved with help of a supercomputer at NERSC, and it is a first-player-win game with 3,009,081,623,421,558 possible positions with symmetries removed[2].

[1] *Figures are directly from comp_424_project_detials.pdf

[2] https://perfect-pentago.net/details.html#algorithms

Motivation With Theoretical Basis

Challenges

Each turn Pentago-Twist requires similar amount of actions as original Pentago game does, So Pentago-Twist could also have more than 3×10^{15} positions. Therefore, it's impossible to let AI agent evaluate every position of the game.

Pentago-Twist is also a very unpopular variant of Pentago game, there is not much information about this game. One of the quadrant will be rotated or flipped in each turn, the game states will not be easy to be predicted by human. So, it's hard to came up a good evaluation function for this large amount of moves in the game without a long time study.

Therefore, the motivation of this project is to implement an AI agent of the high branching factor game without a precise evaluation function.

Selecting Monte Carlo Tree Search (MCTS)

Since Monte Carlo Tree Search algorithm could allow AI agent to choose a promising move without having a good evaluation function nor traversing all positions.

A list of legal moves that AI agent can choose next are the child nodes of root in Monte Carlo Tree. MCTS selects each nodes, run the tree policy to choose a promising leaf node of existing tree node to expand, then run the default policy on the promising node. By recording the back propagated win rate of the child nodes, MCTS could make a choice of next move by choosing the win rate of each node. The pseudocode is following:

```
ArrayList moves <-- all legal moves that current state
MCST.root.children <-- moves
promising_node <-- find one most promising node from root.children
promising_node.child e<-- expand the node by add a child to the promising node;
run default policy on the child node:
while(!e.state.gameover()){
  e.state.process(e.legal_move)
}
e.back_propgate()
```

Upper Confidence Tree

To select a promising node in tree policy. The technique of UCT is added to the MCTS.

Using one node's score *s*, run times *t*, and its parent's run times, UCT value Q can be calculated as: (c is selected to be sqrt(2))

$$Q' = Q(s,a) + c\sqrt{\frac{\log n(s)}{n(s,a)}}$$

Where Q(s,a) is the win rate of the node and n(s) can be got from the parent's run times.

$$Q(s,a) = \frac{WinScore}{RunTimes}$$

By using UCT, we can get benefit of not only selecting a good move each time, but also exploring other moves that AI agent haven't visited often.

Implementation

Code Structure

There are 4 major classes in this project

MoveCoder is responsible for converting between a PentagoMove object and a 5-digit integer.

MCTnode implements the nodes of Monte Carlo tree.

MCTree is responsible for AI agent to find the next move in each turn.

Evaluation class is to evaluate the next possible moves and gives each move a bias score.

AI agent will run MCTS with UCT in MCTree first and get a win rate for every next move from the current state. Then all the moves will be passed to Evaluation class to get a bias number. By add up both bias number and win rate, the node with largest number will be selected to be the next move.

Memory Saving

Since memory is limited for AI agent, and what will cost most memory is the Monte Carlo tree. In order to decrease the size of each node in MCTree. The PentagoMove object will be converted to a 5-digit integer called "codedmove" by using MoveCoder. Codedmove's digits are made by player id, x and y coordinates, aswap and bsawp,1 digit for each value.

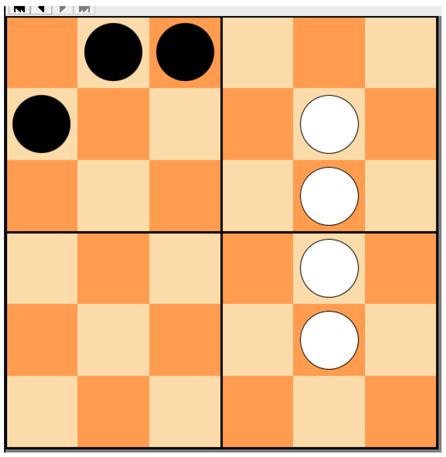
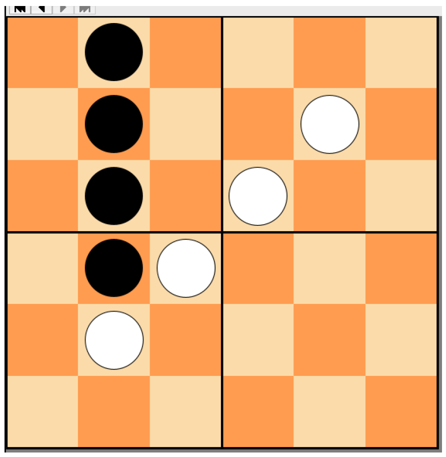
There is also no state is stored in node of Monte Carlo tree to save the memory. To retrieve the state of each node, the series of moves from root node to the current node will be applied to the start state the get the node's state.

Performance Improvement

After implementing the MCTS, the performance of the agent is very poor. It is observed that the agent does not know what is a good move, and it even will not select the winning move sometimes. To improve the performance, the Evaluation class is implement to add some bias number to the decision factor of the agent.

There are 3 type of moves will be rewarded in evaluation function;

1. The move that can lead the agent to win. By simulating moves one step further, the agent is able to detect the winning move. Any winning move will be added a large amount of bias number.
2. The move that placing a piece on the edge of each quadrant near the centre of game board, specifically on the third row, fourth row, third column and fourth column. It is observed that any patten that is hard to be blocked are on the edge.
3. If there are 3 opponent's pieces are connected in a line, it is biased to place pieces on both sides of them. It is found that if there are 4 opponent's pieces are connected in a line and both side of edge are free (examples below), the game state is an absolutely LOSE for the agent.

Absolutely LOSE for black

Analysis

Advantage

The main advantage of the approach is memory saving. By using Monte Carlo algorithm, the AI agent can select a reasonable move without access huge amount of nodes comparing to Minimax tree search. Techniques of converting move object to a 5 digit integer and not storing state object in nodes also decrease the size of Monte Carlo tree a lot. In the performance aspect, opponent 3 pieces detection, one step further checking and bias function in Evaluation class make the agent smarter.

Disadvantage

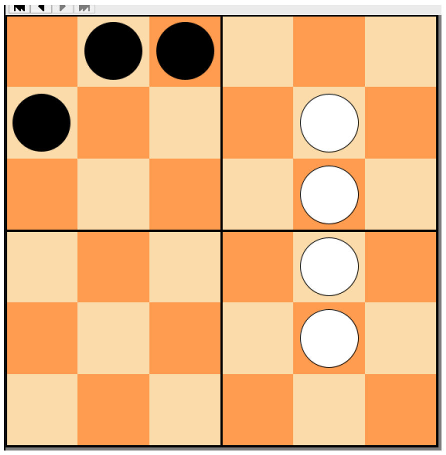
The biggest problem of this AI agent is the lack of bad move detection. The AI agent is not able to detect any moves that can leads to an absolutely win for opponent. Therefore the agent cannot block the opponent's move if the opponent find the winning moves. This problem will be the main reason why it cannot beat many other AI agent in this game since most AI agent will have one or more steps checking for winning moves,

Another problem will be the calling MoveCoder frequently since coded integer moves cannot be passed to many functions. Frequently converting PentagoMoves and integer will cost a lot of time and memory.

Further Improvement

Implementation of 2 or more steps-forward checking on both winning and losing state will made a huge improvement to this agent. The agent will have the ability to filter out all the move that can lead to the loss, and apply the move that leads to the absolutely win.

it is also doable to convert PentagoBoard PentagoState object into number representation class to save the memory. For example the state object of the following state can be converted into a serial numbers: 0 for empty, 1 for black and 2 for white



===> 011000100020000020000020000020000000

By this way, applying pattern recognition on the state will be much easier.

The last improvement can be made is to make good use of the 10 MB data that can be loaded into the agent in the setup phase of the tournament. There are a lot useful pattern can be stored and used to improve the performance of the agent.