COMP 424 - Artificial Intelligence

Final Project Report

Arian Omidi - 260835976

Course Instructor:
Prof. Jackie Cheung

April 14, 2021

Contents

List of Tables

List of Figures

1 Introduction

The main goal of this project was to implement an AI agent for the game Pentago Twist, a variation on the popular game Pentago. Our agent must be able to win against a random player and compete against other students agents with the time constraint of 2 seconds per move and a memory constraint of 500MB of RAM.

2 Design and Motivation

Many different AI agents were considered for this project, including a Min-Max algorithm with Alpha-Beta Pruning and a logical agent. Ultimately however, an AI agent implemented by a Monte Carlo Tree Search (MCTS) with the enhancements of Upper Confidence Trees and Tree Reuse was selected. This decision was largely influenced by the constraints of the project. Firstly, the MCTS algorithm can be terminated at any moment and return the best result found so far, resulting in an agent with a low likelihood of timeouts and disqualifications. Furthermore, due to the high branching factor of Pentago Twist - 288 unique first moves - MCTS can go much deeper than a Min-Max algorithm, potentially finding moves which are more beneficial in the long run. Lastly, given the new variation of Pentago, creating a near-optimal evaluation function for Min-Max would be very time intensive and challenging and due to the random nature of MCTS, the agent would be more competitive and versatile against other agents with differing evaluation criteria.

2.1 Data Structures

Tree and Node data structures were developed to store game states, visit and win counts, and allow tree search functionality. The Node class has the following fields along with constructors and getters:

```java
PentagoBoardState state;
PentagoMove move;
Node parent;
List<Node> children;
int visitCount;
double winScore;
```

A method was also implemented to determine if two nodes are equal. This method compares a nodes board state, turn number, and player to move with the other and is used to identify which sub-trees could be pruned during tree reuse, see Section 2.3.

2.2 Tree Policy

The tree policy used for selecting a node for expansion and simulation was implemented by Upper Confidence Trees, see Section 3. UCT balances the exploitation and exploration such that more promising nodes are selected, while ensuring no node become the victim of starvation.

For this project, the exploration parameter was set to $\frac{\sqrt{2}}{7}$ to favour exploitation of better moves which enables the robot to play for more positional games. This value was empirically chosen by simulating 20 games between agents with varying exploration parameters and selecting then best performing agent, see Section 5.

2.3 Tree Reuse

In the standard MCTS algorithm, the search tree built during a move is discarded at the end of the turn and a new tree is built from scratch on the next turn. However, part of the information in the tree may still be useful for the next turn as it can be used to better guide the search [1]. Thus, by saving the tree and pruning the irrelevant nodes the agent can simulate more nodes, improving the best move estimate. The tree is pruned at the start of a turn by iterating over the children of the root and finding the node which corresponds to the current state of the board. This node is set as the new root and the old root, the siblings of the new root and their subtrees, which are not relevant anymore, are discarded. Below is a code snippet of the implementation:

```java
public void pruneTree(PentagoBoardState state) {
    for (int i = 0; i < root.children.size(); i++) {
        if (equalsBoard(root.children.get(i).state, state)){
            this.root = root.children.get(i);
            break;
        }
    }
    this.root.parent = null;
}
```

3 Theoretical Basis

The Monte Carlo Tree Search algorithm, first developed in 2006 by Rémi Coulom, estimates the best move in a position by averaging the outcome of several random continuations[2].

Monte Carlo Tree Search consists of four main processes: selection, expansion, simulation, and back-propagation.

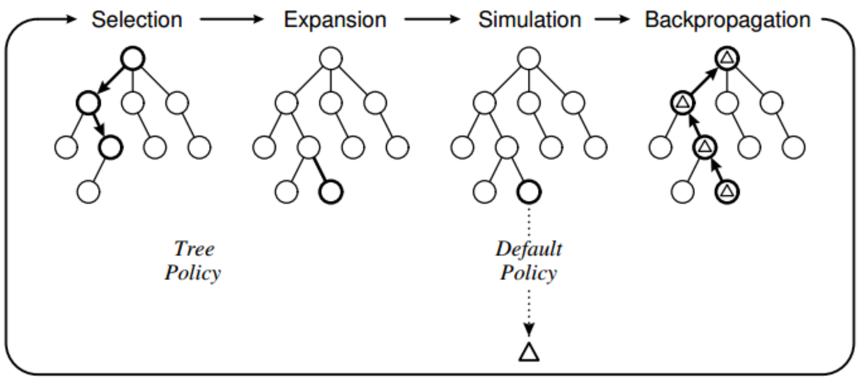

Figure 1: Phases of the Monte Carlo Tree Search

Selection

During selection, the most promising leaf, in the explored tree is selected by using Upper Confidence Bound for trees (UCT). UCT is a weighted balance between the exploitation of promising nodes and the exploration of new nodes, ensuring that no node becomes a victim of starvation while selecting a well performing node. The UCT value of a node is calculated as follows:

$$UCT_i = X_i + C\sqrt{\frac{\ln n_p}{n_i}},$$

where X_i is the win ratio of node i, n_p is the total number of simulations of the parent node, n_i is the number of simulations of node i, and C is an exploration parameter - theoretical equivalent to $\sqrt{2}$ but empirically selected in practice[3].

Selection begins from the root of the exploration tree and recursively selects the child with the highest UCT value until a leaf is reached.

Expansion

This selected node is then expanded and a child node is added to the explored tree for each of the legal moves from the current state. These child nodes are initialized with $X_i, n_i = 0$ and the states for these nodes are generated.

Simulation

A child, w, of the selected node is then chosen arbitrarily and a game is simulated from this node. The moves are selected with respect to a playout policy, either light or heavy. Light playouts consist of selecting moves at random while heavy playouts select moves by applying heuristics which is more computationally expensive. The simulation terminates once the outcome of the state can be determined - win, loss, or draw.

Back Propagation

The final phase of the Monte Carlo Tree Search is back-propagation. The results of the simulation is used to update the visit and win counts of all the nodes on the path from the root to w.

These four phases are repeated until an amount of time has elapsed, the move corresponding to the child of the root with the highest win rate is selected. MCTS with UCT converges to the optimal result as time per move increases.

4 Advantages and Disadvantages

4.1 Advantages

The Monte Carlo Search Tree algorithm has many advantages over other algorithms such as Min-Max. Firstly, MCTS does not require an explicit evaluation function as it randomly expands and simulates promising nodes. This is beneficial as I am not an expert at Pentago Twist and do not know which board states are preferable to others, thus any evaluation function written will most likely be sub-optimal. Since Min-Max will find the optimal move relative to the evaluation function, the agent implemented with it might make a move which leads to a lost position. Moreover, MCTS only expands the nodes with the largest UCT value, concentrating the algorithm on more promising sub-trees. This makes it advantageous in games with high branching factors such as Pentago Twist as the search can go to deeper than Min-Max. MCTS can also be interrupted at any time and return the most promising move found. This means we can optimize our turn time while ensuring that we will not incur a timeout penalty or disqualification.

4.2 Disadvantages

One of the major disadvantages is that if the position is sensitive and there exists only a couple of paths that lead to a win, MCTS can be beaten by an expert player. This occurs because with random simulation it is unlikely to find these specific paths and MCTS will likely end up in a lost position. This becomes more prevalent the deeper the winning combination is. Furthermore, unlike Min-Max[1], MCTS does not guarantee the best move. Again, due to its random simulations and lack of a heuristic, it is possible that a bad combination of moves are selected and for the agent to misevaluate certain moves or positions.

5 Alternatives and Results

Two agents attempted for this project: a standard MCTS agent and a MCTS agent with a tree reuse policy. These agents were set head-to-head for 50 games to determine which agent was ultimately more successful. The agent with the tree reuse policy prevailed, winning 29 out of the 50 games, a 58% win rate. Furthermore, a wide variety of UCT exploration parameters were tested, see Section 3. Four different exploration parameters were tested by playing 20 games against a baseline constant of 1, the results are shown in Table 1. As can be seen, the most successful exploration parameter was $C = \frac{\sqrt{2}}{7}$ as it won all 20 games against the baseline.

Thus, the MCTS agent with tree reuse and a UTC exploration parameter of $C = \frac{\sqrt{2}}{7}$ was selected as the final version.

$UCTExplorationParameter$	Win Rate
$C = \sqrt{2}$	0.55
$C = \frac{\sqrt{2}}{4}$	0.75
$C = \frac{\sqrt{2}}{7}$	1.00
$C = \frac{\sqrt{2}}{10}$	0.85

Table 1: Results from 20 Games of Differing UTC Exploration Parameters against a Baseline of $C = 1$

[1]Min-Max guarantees to return the best move relative to an evaluation function up to a certain depth.

6 Improvements

There are many changes that can be made to improve the performance of the agent. Firstly, we could implement an inexpensive evaluation function to use during playouts. By adding such a heuristic, the agent can explore nodes that are more likely to be played in the game, better informing the agent about the game. Another improvement that can be made is creating an opening move database for Pentago Twist. This would enable the agent to make prepared and optimal moves for the opening which would result in a winning or drawn position. Lastly, we could implement a transposition table to be able to look up the values of equivalent position reached through different move orders. This would allow information sharing between nodes and improve our best move estimate.

References

[1] C. Sironi, “Monte-carlo tree search for artificial general intelligence in games,” Nov 2019.

[2] R. Coulom, “Efficient selectivity and backup operators in monte-carlo tree search,” *Computers and Games Lecture Notes in Computer Science*, p. 72–83, May 2006.

[3] Baeldung, “Monte carlo tree search for tic-tac-toe game,” Oct 2020.