Comp 424 - Project Report

Ruichao Zhang

March 30, 2015

1 Introduction

For the Comp 424 final project, I implemented an AIPlayer for omweso Game. Omweso is a board game similar to mancala.

Omweso requires a board of 32 pits arranged with 8 pits wide and 4 pits deep. Each player has 32 seeds need to play initially on their territory that is 16 pits close to their side of the board. Game starts with first player chosen at random. After initialize the board, players take turns. The player selects one of their pits and starts sowing counter clockwise, placing one seed in each pit until running out of seeds. At end of a sowing, if the pit was previously empty, turn is over, If the pit was occupied, and player meet the condition of capturing (with both of the opponent's pit opposite are occupied). Then player take the seeds from opponent' pit and start sowing from the previous starting place. If previous 2 condition, player take all seed in the pits and continues sowing. In this project, the player wins when opponent has no move to play.

The design idea of the Ai bot is originally from another popular board game Tic tac toe. Tic Tac Toe is easily solvable by the algorithm Minmax that is used particularly to solve board game problem. The essential idea of Minmax algorithm is like dynamic programming. The player choose the best move that will maximized his win potential while consider the opponent will minimized his. MinMax allows the player to foresee the move several steps in the future, which grants the advantage toward himself. In the practical perspective, Minmax is widely used and well documented; therefore this easy implement approach is chosen for project needs.

Numerous trials fail on increasing the depth as well as improving the algorithm (using alpha beta pruning). In the end, the final version of the working code is using Minmax (depth 4).

Theoretical basis of MinMax

The minmax algorithm uses the recursive computation to make the decision base on the current state. The minmax values are then backed up through the tree. The bottom leaves are using the evaluation function or the heuristic function to determine its value.

The minmax algorithm uses a depth-first approach in search the game tree. The time complexity of the algorithm is 0(bm) given b is the legal move at each state, and m the maximum depth.

The algorithm can be improved using the alpha beta pruning approach. It keeps track of the min max value of each state and stop exploring the node that is not necessary. It the branching factor is b , search depth is d, search space of minmax is $O(b^d)$, while the alpha-beta yield optimal O(b^d/2) and average case is O(3d/4).

Advantage of MinMax

MinMax algorithm is commonly used to solve observable deterministic board game problems. Within certain depth, it performs very efficient and fast. It uses relatively less memory space. Combines with a good heuristic function, it will yield a promising win rate. In this project, the heuristic function is set to be the number difference between current player board's seed and initial board. More seed on the board means more likely to win.

Disadvantage of MinMax

For each move, using MinMax will need to recompute the previous computed result. The algorithm will be more effective if keeps track of pre-computed result. Due to the huge branch factor, Minmax will easily exhaust the computation resource. Therefore using such algorithm may not yield the optimal solution due to this depth limitation. Moreover, this search is heavily dependent on the heuristic. The alpha—beta algorithm is designed to select a good move base on calculating the bounds on the values of all the legal moves. It can be problematic if the design of the heuristic is not accurate since it will then not go through those moves that have lower heuristic value, therefore leads to a not optimal solution.

Alternative approach

The Monte Carlo tree search is a heuristic search algorithm that can be use to search through the future possible state of the board. The MCTs chooses the most promising moves base on the expansion and random sampling of the searching tree. There are four steps in MCTs, Descent phase, Rollout phase, Update phase and Growth Phase(Pineau.J).

In the Descent phase, it uses minmax strategy to expand to certain depth. Use the heuristic estimation of the child node to decide with node will be selected in the later rollout phase.

In the Rollout, once the node is selected, do a Monte Carlo simulation to the end of the game.

In the update phase, pass back the information up to the tree and update the information of the selected node

In the Growth phase, the first state of the rollout is added to the tree. Base on the value of these first states, MCTs select the more promising state and expand that state and do simulation.

MCTs is harder to implement to integrate into the project. After MinMax expansion, MCTs simulation is added to heuristic function to generate a better estimated of the child value. The selecting strategy, which is the essential of the MCTs is using an evaluation function to compute confidence on both the current state and action it will select afterward. In this project, such strategy is too difficult to implement, my simulation is chosen to run in all possible moves, which results in the timeout of the search. (similar to Pure Monte Carlo)

Even though MCTs fails due to its complexity, it is more precise and efficient compared to MinMax on its rollout phase and selecting strategy. It allows a further depth to the end of the game, so it will be unaffected by the branching factor. It is more flexible since it can be interrupt at any time and return the current most promising move.

Pure Monte Carlo game search is an alterative version of MCTs, it applied to the finitely moves board game. On all possible move, it simulates k games, and selected the move with the best score.

Improvement of the AI Player

Additional technology can be used to further improve the AI player. First, Multithreads programming can be useful to improve efficiency. Since the algorithm is using depth-first approach, the different node can be selected simultaneously, in return it shortens the computation time. Second, Memory can be use to store pre-compute move result. So when computing the MinMax for the next run, we can search through this memory tree and just compute one depth further away. In term of game strategy, the heuristic can be weight more precise by adding more factors, for example, less seed in the same column. In the end, adds the MCTs simulation. In order to reach the best possible AI Player, different factors will need to be considered to balance between this random trials and best-worst strategy.

References

Russell, S., & Norvig, P. (2010). Chapter 5 Adversarial Search. In *Artificial intelligence: A modern approach* (3rd ed.). Englewood Cliffs, N.J.: Prentice Hall.

Pineau, J. (Director) (2015, January 28). Monte-Carlo Tree Search. Lecture conducted from Joelle Pineau.