COMP 424 Final Project Report

Sagar Nandeshwar (260920948), Miłosz Holeksa(260998827)

April 13, 2022

Abstract

In the report with have discussed the implementation of the Student Agent, theoretical basis of the approach, results and possible improvement.

1 Introduction

The goal of this project is build Student Agent to understand and play Colosseum Survival using AI algorithms discussed in the course. The agent is expected to run in feasible amount of time and limited space.

2 Colosseum Survival

Colosseum Survival! is a 2-player turn-based strategy game in which two players move in an M $\times$M chessboard and put barriers around them until they are separated in two closed zones. M can have a value between 6 and 12. Each player will try to maximize the number of blocks in its zone to win the game.
We have used Mini-max search algorithm to maximize our winning chances using heuristic functions that evaluate the state of the game.

3 Mini-max

Mini-max search algorithm is a backtracking search algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.

The algorithm performs a depth first search to reach leaf and then calculate the score of the leaf using heuristic function discussed later.

The value is then passed on to the parent node, which then take the highest(Max layer) or lowers(Min layer) amongst all the children/leaves and then passed it on it parents.

Max layer represent our agent move that is trying to maximize the score and Min layer represent adversarial move that is trying to minimize the score.

4 Implementation

4.1 Motivation

Our approach for the game is motivated but the tic-tac-toe game as seen in the lecture. We have implemented attacking and defensive used in games Chess. We have also implemented weighted grid that is used in games like Othello.

4.2 Design and structure

We have implemented Mini-max search algorithm with two layers. Due to limited time and space, we have use up to two to layers of mini max. The root of the tree is our starting position and its children will be all the possible moves that current playing agent can take.

Root: We start with the root node, that is our current position of the agent.

Layer 1: Then we create layer 1, i.e. all the possible positions (or state of the game) from the current position.

Layer 2: The we create layer 2, i.e. all the possible position the the opponent can take.

Final score: In the end we calculate the final score of the position of our agent, using Heuristic function, after adversarial have moved.

4.3 Heuristic function

Our strategy in the game is to be attacking and defensive at the same time. After playing multiple time, we found that there are three important factors that affects the game. First, defensive, the number of moves that we can take, second, attacking, the number of moves the adversarial can take, and, third, position on the board. We want to maximize the

the number of moves that we can take and minimize the the number of moves the adversarial can take, taking care of the position of our agent in the during the game.
For the first two strategies we take the linear combination as the heuristic function.

$$score_1 = The\ number\ of\ moves\ that\ we\ can\ take$$
$$score_2 = The\ number\ of\ moves\ that\ adversarial\ can\ take$$
$$score = score_1 - score_2$$

We want to maximize this difference as much as possible.
When we end up in the state of the game where more than one move gives same score, we apply out third strategy. For this we will give preference to those position that are central to the game. Due to time restriction, in actual implementation we just chose random move.

5 Result

5.1 In the current

I made student agent play against the random agent, 50 times; 10 times in 6x6, 10 times in 7x7, 10 times in 8x8, 10 times in 9x9, 10 times in 10x10.

I found the student agent has won 32 matches out of 50.

5.2 Other Approach and their results

- I tried to play the game with more layers in mini-max tree. While increasing the with just one or two more layer doesn't contribute significant difference but when we increase the layers to all-most double more more have show improvement in the performance, especially in small chess board.
- Instead of choosing random move, when the score is same, I added weighted move that favors central position in the board more than the border positions. The implementation doesn't shows any sign of improvement in most of the cases.
- Implementing heuristic function with only defensive strategy. This works better in small chess table (M=4,5,6) but it performs very poorly in larger chess board (M=7,8,9,..)

- Implementing heuristic function with only attacking strategy. This gives have better performance in large chess board (M=9,10,11,12.) but there is no improvement in smaller chess board (M=4,5,6).

6 Advantages and Disadvantages

6.1 Advantages

- Mini-max Algorithm is complete as it reaches leaf in the finite search tree (in our case with depth = 2 and limited breath).
- It is optimal against an optimal opponent.
- acceptable space complexity of O(bm) as DFS is used.

6.2 Disadvantages

- The time complexity of the Minimax algorithm is $O(b^m)$, where b = tree's branch and m depth of the tree. Which slow when max no. of move is large.
- If opponent have more layers/depth in Mini max it could out perform out agent.
- Exploring the entire tree is not possible as there is restriction of time.
- Without any pruning, Search and evaluation of unnecessary nodes worsen the efficiency and the performance of the agent

7 Possible Improvement

- Minimax with Alpha-beta Pruning. This would have largely improved our performance and efficiency of our approach, as it would have allowed us to explore more depth but not searching unnecessary nodes.
- Order of moves in every branch of tree. We implemented properly we could have saved a lot time by completely ignoring worse moves.
- Better Heuristic function that could also calculate the distance from wall and border.
- The randomized Monte Carlo Search tree method could have been used to find moves.