Project 3: Build Game Playing Agent

Experimental Results & Report

By: Sarp Karamarti

Six experiments, referred to as E1 to E6, have been conducted for this project this project. In E1 to E4 α-β has been tested as a search algorithm, with E1 and E2 having a depth of 3 and E3 and E4 using iterative deepening. Furthermore, E1 and E3 have been deployed with a simple heuristic as the evaluation function, namely the #mymoves - #opponentmoves heuristic. E2 and E4 on the other hand were tested with an advanced heuristic that can be calculated as follows:

```
If game_state.ply_cnt < 30:
        return #mymoves - #opponentmoves - distance(player,opponent)

Else If game_state.ply_cnt < 50:
        return #mymoves - #opponentmoves - distance(p,o) - distance(p,m)

Else:
        return #mymoves - #opponentmoves - distance(p,m)
```

The idea behind the advanced heuristic is that the agent has additionally to the simple heuristic, the strategy to keep the distance to the opponent short and chasing him. After the game advanced, the strategy changes to also keep the distance short to the middle of the board. In the final stage of the game, the agent's strategy is to keep the number of his moves compared to the opponent's high and the distance to the middle of the board short. For all experiments, E1 to E4, 50 rounds with the fair matches setting have been played against the MiniMax opponent and a time limit of 150 ms. As we can see in Table 1, The advanced heuristic has a slight advantage in terms of the win rate over the simple heuristic. Interestingly, this advantage becomes even less when using iterative deepening. This might lead to the assumption, that when higher depths are available this heuristic becomes less advantageous over the simple heuristic. In order to validate this assumption E1 and E2 have been replicated twice with depth 2 and depth 4 for the agent and the opponent. The results, however, show quite the opposite pattern as depicted as expected previously. The reason for this could be that iterative deepening has already such an edge over simple minimax that a better heuristic does not increase the result much.

In E5 and E6 an implementation of the monte carlo tree search (mcts) method was evaluated. The algorithm is based on the Pseudocode provided in the additional material of the lectures. What's really interesting here is, that the mcts algorithm only outshines the simple minimax with depth

3 with significantly higher time limits than used in the previous experiments. The reason for this is probably that in each iteration, new statistics get generated and the accuracy of the statistics becomes better the longer the time limit per move is. However, one big advantage of mcts is that it is a aheuristic and a simple algorithm able to play complex games.

The overall conclusion for this project is that a α-β implementation with iterative deepening performs best with a limited time limit per move and simple games with smaller trees. Considering more complex games that have a bigger search space mcts might be the better choice.

Table 1 – Conducted experiments

	E1	E2	E3	E4	E5	E6
Search Algorithm	α-β	α-β	α-β	α-β	MCTS	MCTS
depth	3	3	id	id	–	–
heuristic	simple	advanced	simple	advanced	–	–
Time Limit	150ms	150ms	150ms	150ms	300ms	5000ms
Rounds	50	50	50	50	50	50
Opponent	MiniMax	MiniMax	MiniMax	MiniMax	MiniMax	MiniMax
Winrate	50.0%	55.5%	81.5%	80.0%	51.0%	78.0%

Table 2 – Variations of E1 and E2

	$E1_{d=2}$	$E2_{d=2}$	$E1_{d=4}$	$E2_{d=4}$
Winrate	50.0%	52.5%	50%	61.0%