VINUNIVERSITY

Project 1: Tic Tac Toe

Vu Ai Thanh - V202200674

Q1: Minimax Player

Evaluate the performance and optimality of the Minimax Player by playing a series of games against different types of opponents: a Random Player, itself (Minimax vs. Minimax), and a Human Player.

Minimax vs. Random Player

```
Final Scoreboard:
Minimax Player wins 87/100 games
Random Player wins 0/100 games
Draws 13/100 games

Minimax Player average move duration: 0.12 seconds
Random Player average move duration: 0.00 seconds
Total evaluation time: 42.84 seconds
```

Minimax vs. Minimax Player

```
Final Scoreboard:
Minimax Player wins 0/100 games
Minimax Player wins 0/100 games
Draws 100/100 games

Minimax Player average move duration: 0.11 seconds
Total evaluation time: 101.56 seconds
```

Minimax vs. Human Player

```
Final Scoreboard:
Minimax Player wins 0/10 games
Human Player wins 0/10 games
Draws 10/10 games
```

Conclusion

The results of the games indicate that the Minimax Player consistently plays optimall. It consistently avoids losses and makes the best possible moves to either win or draw games. Its performance against Random, Human, and itself shows it always finds the optimal strategy, ensuring no defeats.

Q2: AlphaBeta Player

Play 20 games between AlphaBeta and Minimax Player, the final result is as bellow:

```
Final Scoreboard:
Alpha-Beta Player wins 0/20 games
Minimax Player wins 0/20 games
Draws 20/20 games

Alpha-Beta Player average move duration: 0.08 seconds
Minimax Player average move duration: 0.09 seconds
Total evaluation time: 15.40 seconds
```

The AlphaBeta Player gave the same performance as the Minimax Player in terms of playing optimally, with all 20 games resulting in draws. However, Alpha-Beta pruning does result in an improvement in runtime. The average move duration for the AlphaBeta Player was 0.08 seconds, slightly faster than the Minimax Player's 0.09 seconds.

Q3: Monte Carlo Tree Search

Using the default value of NUM_SIMULATIONS = 5000 and an exploration constant of $c = \sqrt{2}$, the MCTS Player plays optimally, matching the performance of the Minimax/AlphaBeta Player in terms of optimal play, as evidenced by the 100% draw rate in their games. However, the MCTS Player takes longer to make a move, averaging 0.43 seconds per move compared to the AlphaBeta Player's 0.09 seconds.

```
Final Scoreboard:
MCTS Player wins 0/100 games
Alpha-Beta Player wins 0/100 games
Draws 100/100 games

MCTS Player average move duration: 0.43 seconds
Alpha-Beta Player average move duration: 0.09 seconds
Total evaluation time: 233.67 seconds
```

Try to increase the number of simulations from 100 to 10000

- NUM_SIMULATIONS = 100

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 12/50 games
Draws 38/50 games

MCTS Player average move duration: 0.01 seconds
Alpha-Beta Player average move duration: 0.08 seconds
Total evaluation time: 19.83 seconds
```

- NUM_SIMULATIONS = 250

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 9/50 games
Draws 41/50 games

MCTS Player average move duration: 0.02 seconds
Alpha-Beta Player average move duration: 0.09 seconds
Total evaluation time: 25.82 seconds
```

- NUM_SIMULATIONS = 750

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 3/50 games
Draws 47/50 games

MCTS Player average move duration: 0.06 seconds
Alpha-Beta Player average move duration: 0.08 seconds
Total evaluation time: 29.80 seconds
```

- NUM_SIMULATIONS = 1000

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 1/50 games
Draws 49/50 games

MCTS Player average move duration: 0.08 seconds
Alpha-Beta Player average move duration: 0.08 seconds
Total evaluation time: 35.47 seconds
```

- NUM_SIMULATIONS = 2000

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 0/50 games
Draws 50/50 games

MCTS Player average move duration: 0.15 seconds
Alpha-Beta Player average move duration: 0.08 seconds
Total evaluation time: 51.14 seconds
```

- NUM_SIMULATIONS > 2000, the MCTS player consistently makes the best move, although it takes more time to do so.

Increasing the number of simulations from 100 to 10,000 significantly impacts the performance of the MCTS Player. The smaller the NUM_SIMULATIONS value, the less optimal the MCTS Player performs, though it makes moves faster. As NUM_SIMULATIONS increases, the MCTS Player becomes more optimal, closely approximating the behavior of a Minimax agent. Essentially, as NUM_SIMULATIONS approaches infinity, the MCTS Player's decision-making becomes nearly identical to that of a Minimax agent, trading off increased computation time for improved performance.

How does c impact the performance?

- Smaller c values (e.g., 0.5): With a lower exploration constant, the MCTS Player will favor exploitation over exploration, focusing more on the moves that have yielded high rewards in the past. This may lead to faster but potentially suboptimal decisions, especially in complex game states where exploring new moves could uncover better strategies.

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 7/50 games
Draws 43/50 games

MCTS Player average move duration: 0.14 seconds
Alpha-Beta Player average move duration: 0.08 seconds
Total evaluation time: 46.74 seconds
```

- Default $c = \sqrt{2}$: At this value, there is a balanced trade-off between exploration and exploitation. The MCTS Player can effectively explore new moves while still considering past successes, leading to well-rounded performance in a variety of game scenarios.

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 0/50 games
Draws 50/50 games

MCTS Player average move duration: 0.15 seconds
Alpha-Beta Player average move duration: 0.08 seconds
Total evaluation time: 51.14 seconds
```

- Larger c values: With a higher exploration constant, the MCTS Player will emphasize exploration, giving more weight to less-visited moves. This can be beneficial in highly complex or poorly understood game states where discovering new strategies is crucial. It may also result in longer decision times and potentially weaker performance in situations where exploitation of known good moves is more advantageous.

```
Final Scoreboard:
MCTS Player wins 0/50 games
Alpha-Beta Player wins 2/50 games
Draws 48/50 games

MCTS Player average move duration: 0.16 seconds
Alpha-Beta Player average move duration: 0.09 seconds
Total evaluation time: 54.55 seconds
```

A better rollout policy

A more effective rollout policy for a complex game like Gomoku would involve using game-specific heuristics instead of random moves. For example, the policy could prioritize moves that block the opponent's threats, extend the player's own sequences, and focus on controlling key positions on the board.

Q4: Tabular Q-learning

Define a MDP for the game

States S

Each state represents a possible configuration of the Tic-Tac-Toe board. There are $3^9 = 19683$ possible states, considering all possible combinations of X, O, and empty cells on the 3x3 grid.

Actions A

Actions represent the possible moves that a player can make in a given state. Each action corresponds to placing either an X or an O in an empty cell on the board. There are $9 - \|X\| - \|O\|$ possible actions in a state, where $\|X\|$ and $\|O\|$ are the number of X's and O's already placed on the board, respectively.

Transition Model P

The transition is deterministic. After a player makes a move, the game progresses to a new state corresponding to the updated board configuration.

Reward Model R

- $R(s, a) = 1$ if the action a leads to a win for the player making the move.
- $R(s, a) = -1$ if the action a leads to a loss for the player making the move.
- $R(s, a) = 0$ if the game continues without win or loss.

Q-Learning vs. every other Player

In the following games, the Q-learning player was trained with the following parameters:

- NUM_EPISODES = 200000
- LEARNING_RATE = 0.2
- DISCOUNT_FACTOR = 0.2
- EXPLORATION_RATE = 0.1

and using Q-Learning player as transfer_player.

- Q-Learning vs. itself

```
Final Scoreboard:
Q-learning Player wins 0/100 games
Q-learning Player wins 0/100 games
Draws 100/100 games

Q-learning Player average move duration: 0.00 seconds
Total evaluation time: 0.33 seconds
```

- Q-Learning vs. Minimax Player

```
Final Scoreboard:
Q-learning Player wins 0/100 games
Minimax Player wins 0/100 games
Draws 100/100 games

Q-learning Player average move duration: 0.00 seconds
Minimax Player average move duration: 0.11 seconds
Total evaluation time: 49.61 seconds
```

- Q-Learning vs. AlphaBeta Player

```
Final Scoreboard:
Q-learning Player wins 0/100 games
Alpha-Beta Player wins 0/100 games
Draws 100/100 games

Q-learning Player average move duration: 0.00 seconds
Alpha-Beta Player average move duration: 0.10 seconds
Total evaluation time: 45.78 seconds
```

- Q-Learning vs. Random Player

```
Final Scoreboard:
Q-learning Player wins 93/100 games
Random Player wins 0/100 games
Draws 7/100 games

Q-learning Player average move duration: 0.00 seconds
Random Player average move duration: 0.00 seconds
Total evaluation time: 0.23 seconds
```

In the recorded 100 games between the Q-Learning agent and other agents, it demonstrated strong performance. When playing with itself (Q-learning), Minimax, and AlphaBeta player, both agents exhibited

optimal decision-making, resulting in a 100% draw rate. Against the Random Player, the Q-Learning agent was winning 93% of the games and drawing 7%.

Hyperparameter tuning

Observations

- Number of Training Epochs: More training epochs allow the Q-learning algorithm to explore and learn from a greater variety of game states, leading to a more comprehensive understanding of optimal strategies.
- Discount Factor: A high discount factor gives more weight to future rewards, potentially causing the agent to prioritize long-term gains over immediate ones. This can lead to suboptimal decisions, especially in games where short-term gains are crucial for success.
- Learning Rate: A high learning rate can result in rapid adjustments to the Q-values, potentially causing the agent to overfit to specific experiences or to oscillate between strategies without fully converging to an optimal solution.
- Exploration Rate: A high exploration rate encourages the agent to explore new actions, which can be beneficial for discovering optimal strategies. However, if the exploration rate is too high, the agent may spend too much time exploring less promising actions, leading to suboptimal performance.

A good combination of hyperparameters for Tic Tac Toe would involve setting a large number of training episodes to allow for thorough learning. Additionally, choosing a low learning rate, discount factor, and exploration rate would help ensure stable and effective learning, as these values strike a balance between exploring new actions and exploiting learned strategies. Therefore, my choice would be:

- NUM_EPISODES = 200000
- LEARNING_RATE = 0.2
- DISCOUNT_FACTOR = 0.2
- EXPLORATION_RATE = 0.1