Technical Report for Assignment 3 - COMP1100

Aryan Odugoudar

Student Information

- **Name:** Aryan Odugoudar
- **Laboratory Time:** Friday 8:00am - 10:00am
- **Tutors:** Jess Allen and Madeleine Stewart
- **University ID:** u7689173

Contents

1 Introduction

The objective of this technical report is to elucidate the design choices, implementation strategies, and testing methodologies employed in developing the SushiGo AI for COMP1100 Assignment - 3. SushiGo is a card game where players select cards from their hands to form combinations for maximum points. This report discusses the thought process behind the implementation, design decisions, and the rationale behind the chosen algorithms.

2 Analysis of the Program

2.1 Implemented AI Strategies

1. **NoLookahead (Greedy Strategy):**

   The aiGreedy function is a straightforward yet effective approach. It prioritizes immediate gains without considering future consequences. When evaluating possible moves, it calculates their immediate advantage using the `evaluateMove` function. This function, in turn, assesses the potential gain of each move within the current game state, considering in score differentials and card availability. By selecting the move with the highest immediate advantage, the AI maximizes short-term gains, making it ideal for situations where quick decisions are made.

2. **WithLookahead (Minimax Strategy):**

   In contrast, the aiMinimax function employs the minimax algorithm with to go deeper into the game tree. This function evaluates potential moves not only based on their immediate advantages but also by forecasting future states. By recursively exploring the game tree up to a specified depth, the AI anticipates opponents' moves and strategically chooses its own moves. The minimax function, a core component of this strategy, balances maximizing and minimizing moves, aiming to maximize the AI's advantage while minimizing the opponent's. The depth parameter acts as a critical tuning factor, determining the balance between computational complexity and strategic depth.

2.2 Implementation Details

1. **Game State Evaluation (evaluateGameState):**

   The `evaluateGameState` function serves as the important part for both strategies. It calculates a heuristic value for a given game state, reflecting the AI's advantage. The heuristic consists of several factors, including the score difference between players and the disparity in their respective hand sizes. By considering both the current state and potential future scenarios, the AI gains insight into the game's overall trajectory. This heuristic guides the decision-making process, enabling the AI to make informed choices.

2. **Move Evaluation (evaluateMove):**

   Within the `aiGreedy` strategy, the `evaluateMove` function plays a pivotal role. For each potential move, it evaluates the immediate advantage it offers. This evaluation involves simulating the game state after executing the move and computing the resulting score difference. By rapidly assessing moves in this manner, the AI identifies the move with the most immediate benefit, aligning with the greedy strategy's focus on quick gains.

3. **Minimax Algorithm (minimax):**

   The heart of the `aiMinimax` strategy lies in the minimax function. This recursive algorithm explores the game tree, evaluating moves and countermoves up to the specified depth. It balances two fundamental objectives:

4. **Maximizing Moves (for the AI Player):**

   When the algorithm explores the AI player's moves, it aims to maximize the heuristic value. It selects moves that lead to game states with the highest perceived advantage.

5. **Minimizing Moves (for the Opponent):**

   Conversely, when evaluating the opponent's moves, the algorithm seeks to minimize the heuristic value. It assumes the opponent will choose moves that diminish the AI's advantage.
   By recursively applying this logic, the AI anticipates multiple potential outcomes, allowing it to make strategic decisions based on a deeper understanding of the game's dynamics.

3 Rationale and Reflection

3.1 Design Choices and Assumptions:

1. **Greedy vs. Minimax:**

   The decision to implement both greedy and minimax strategies from the inherent trade-off between computational complexity and strategic depth. The greedy strategy offers a rapid decision-making process, ideal for scenarios where immediate gains are pivotal. On the other hand, the minimax strategy sacrifices speed for depth, enabling the AI to consider long-term consequences and opponents' reactions. The coexistence of these strategies ensures adaptability, allowing the AI to choose the most appropriate approach based on the current context.

2. **Heuristic Evaluation:**

   Crafting an effective heuristic for `evaluateGameState` required thoughtful consideration. The chosen factors, including score differentials and hand sizes, were meticulously balanced to provide an accurate representation of the game state's advantage. Fine-tuning these factors was essential to ensure the AI's evaluations align with strategic gameplay.

3.2 Challenges and Solutions

1. **Computational Constraints:**

   One of the primary challenges faced was the computational limitations imposed by lookahead depth. Striking a balance between depth and processing time was crucial. The depth parameter in the minimax strategy offered a solution, allowing the AI's behavior to be tailored based on available computational resources. Careful consideration of this parameter influenced the AI's strategic depth and adaptability.

2. **Heuristic Accuracy:**

   Designing a heuristic capable of capturing the game's complexity presented another hurdle. Iterative refinement and testing were essential to ensure the heuristic accurately reflected the game's dynamics. Balancing factors such as score differences and hand sizes required extensive testing and adjustment to achieve a heuristic that reliably guided the AI's decisions.

4 Testing

4.1 Function-Level Testing:

1. **evaluateGameStateTests:**

   These tests rigorously evaluated the `evaluateGameState` function's accuracy. Various scenarios, including diverse score differentials and hand sizes, were examined to verify the function's ability to assess game states correctly.

2. **evaluateGameStateTests:**

   These tests rigorously evaluated the `evaluateGameState` function's accuracy. Various scenarios, including diverse score differentials and hand sizes, were examined to verify the function's ability to assess game states correctly.

3. **greedyMoveTests:**

   These tests scrutinized the `greedyMove` function's decision-making process. Scenarios emphasizing immediate advantage were explored, ensuring the function consistently selected moves with the highest immediate benefit.

4. **minimaxTests:**

   These tests comprehensively assessed the minimax function's performance. Different lookahead depths and game states were examined to validate the AI's strategic decisions. Edge cases, including game terminations, were considered to confirm the AI's behavior under diverse conditions.

5 Conclusion

In conclusion, this technical report provides an overview of my implementation for Assignment 3 in COMP1100. It discusses the program's features, implementation details, design choices, assumptions, testing and procedures.It successfully meets the requirements of the assignment as expected.