East West University

Department of CSE

Course Code and Name: **CSE 366 Artificial Intelligence** Section: **02**
Lab 6 Report
Semester and Year: **Fall 2024**

Name of Student, ID	Course Instructor information:
Md. Bayzid Ohin (2021-3-60-288)	**Dr. Raihan Ul Islam** Associate Professor Department of Computer Science & Engineering

Report on AlphaBetaAgent

Introduction

The AlphaBetaAgent is an advanced AI that improves upon the traditional Minimax algorithm by implementing alpha-beta pruning. This enhancement allows the agent to make efficient and strategic decisions in adversarial multi-agent environments, such as Pacman. By pruning unnecessary branches of the game tree, the agent significantly reduces the number of states it evaluates, making it both faster and more effective

Understanding Alpha-Beta Pruning

The Minimax algorithm is a classic decision-making approach used in two-player games. However, its biggest limitation is that it evaluates all possible moves, which becomes computationally expensive. Alpha-beta pruning introduces two key thresholds

- Alpha (α): The best score that the maximizing player (Pacman) can guarantee so far
- Beta (β): The best score that the minimizing player (ghosts) can guarantee so far

If a certain branch of the game tree cannot provide a better outcome than the existing alpha or beta values, that branch is pruned—eliminating unnecessary computations

Implementation Overview

The AlphaBetaAgent is built upon the MultiAgentSearchAgent class and operates in a multi-agent setup where Pacman maximizes its score and ghosts minimize it. The implementation consists of several key methods

1. getAction(gameState)

- Serves as the entry point for the agent's decision-making process
- Initiates the alpha-beta pruning search
- Returns the best action for Pacman based on computed utility

2. maximize(state, depth, alpha, beta)

- Evaluates Pacman's possible moves
- Selects the move with the highest score
- Updates alpha (α) and prunes branches where further exploration is unnecessary

3. minimize(state, depth, agentIndex, alpha, beta)

- Evaluates ghosts' possible moves
- Updates beta (β) and prunes unnecessary branches
- Ensures that each ghost minimizes Pacman’s score

4. Depth Handling and Evaluation

- The agent operates with a fixed search depth to limit computation time
- The evaluation function is used to score non-terminal states, providing a heuristic measure of state desirability
- Once the depth limit is reached or a terminal state is encountered, the evaluation function is used to return a score

Key Observations and Advantages

Efficiency Boost

Alpha-beta pruning significantly improves the efficiency of decision-making by reducing the number of game states explored, making the agent much faster than a standard Minimax agent

Better Decision-Making

By pruning unnecessary branches, the agent focuses on the most promising moves, improving its ability to navigate complex situations

Multi-Agent Adaptability

The algorithm is well-suited for Pacman, handling interactions between multiple agents (Pacman and ghosts) in a structured manner

Challenges and Considerations

Evaluation Function Impact

The agent's effectiveness depends heavily on the quality of the evaluation function. A weak heuristic can lead to suboptimal decisions

Fixed Depth Limitation

A fixed search depth ensures computational feasibility but may limit the agent's ability to anticipate long-term consequences

Conclusion and Future Enhancements

The AlphaBetaAgent is a powerful implementation of adversarial search, offering significant improvements over traditional Minimax. By efficiently pruning the game tree, the agent balances speed and accuracy, making it well-suited for complex environments like Pacman

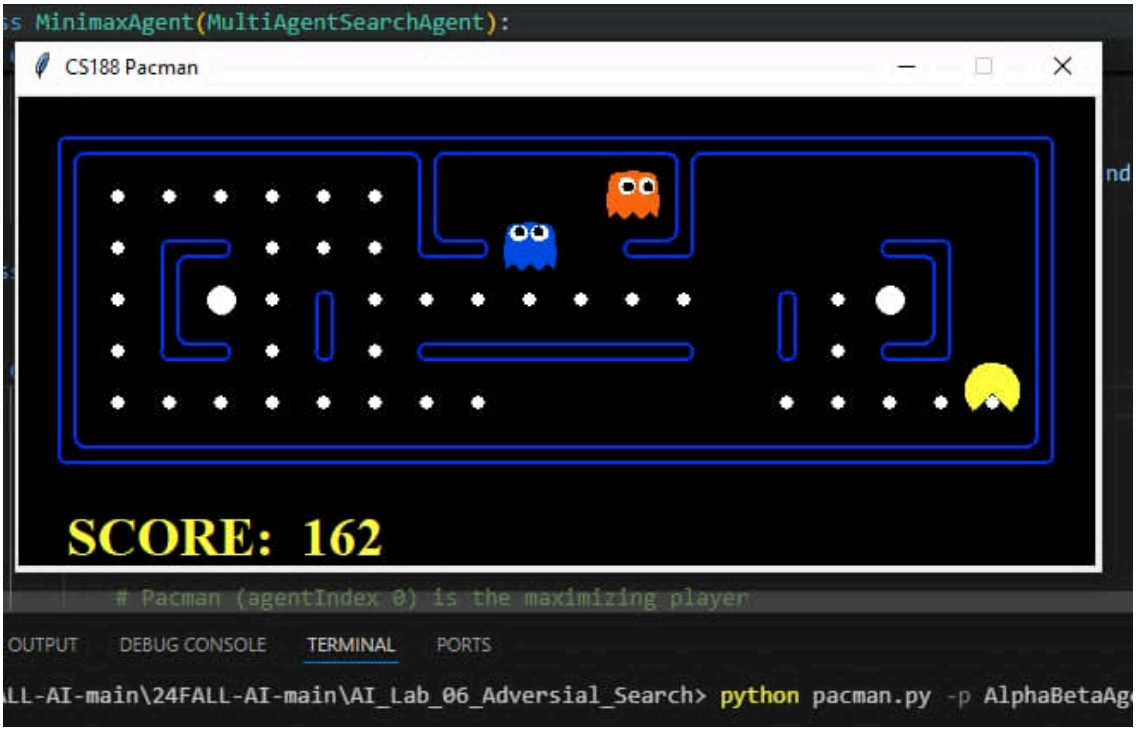
Future Enhancements

- Improving the Evaluation Function: Implementing a more sophisticated heuristic to better predict game outcomes
- Adaptive Depth Search: Allowing dynamic depth adjustment based on game complexity
- Learning-Based Enhancements: Incorporating reinforcement learning techniques to refine decision-making over time

By refining these aspects, the AlphaBetaAgent can become even more efficient and intelligent, leading to smarter gameplay strategies and faster execution times

Sample Output

The output generated by the AlphaBetaAgent during execution.



The image above shows the output when running the command: python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic