PACMAN Game AI
Project Description

Dev Patel (dmpatel3), Aditya Karthikeyan (akarthi3), and Jay Joshi (jmjoshi)
Department of Computer Science - CSC584
North Carolina State University
dmpatel3@ncsu.edu — akarthi3@ncsu.edu — jmjoshi@ncsu.edu

Abstract

PACMAN is an action maze game where the player is in control of the main character Pac-Man, and the main objective involves traversing through a maze to eat all of the dots in the maze/level while not getting hit by any of the ghosts moving around in the maze. However, the game is single-player and doesn't have any opponent going against the main player. Because of this, the project will focus on implementing an AI opponent player into the game as a way of introducing a Player vs. AI concept into the game. Implementing the AI player into the game will be done using Python, and most of the implementation will be based off of the PACMAN game code from Giant Jenks' and PACMAN projects from UC Berkeley. Implementing the movement behavior for the AI player will be done using the minimax algorithm along with the alpha-beta pruning algorithm. The baseline of this project will be the vanilla PACMAN game with no AI agents acting on its environment. Finally, the AI player's performance will be evaluated based on the number of victories against a human player along with the average number of dots consumed within a set amount of time and average amount of time taken to complete a maze/level. Overall, this project hopes to enhance the challenge of the game while also potentially introducing solutions for similar problems in other industries.

Introduction

PACMAN is a maze action video game developed by Namco and published by Namco in 1980, and it has become one of the most iconic and influential video games ever. In the game, the player plays as the main character Pacman, who has the ability to consume dots that are placed throughout a maze. While trying to consume all the dots in a maze, ghost opponents are also present and have the ability to kill Pacman if they touch him, so the player has to also worry about dodging the ghosts while eating the dots. Even though the ghosts can kill him, Pacman does have the ability to defeat the ghosts if he consumes a power pellet placed in the maze. Overall, the main objective of the game is to consume all the dots within the given maze/level while dodging all the ghosts in the level in order to avoid getting killed by the ghosts and beat the game, as depicted in Figure 1.

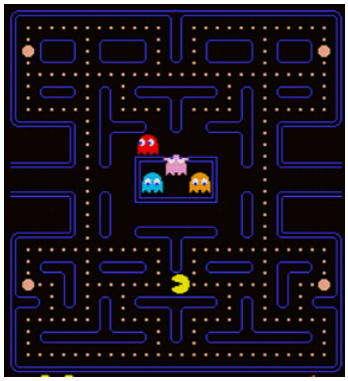

Figure 1: Pacman Game Snippet

Even though the ghosts are enemies controlled by AI aimed at getting the player, the game doesn't have an actual opponent competing against the actual player in terms of getting all the dots in the level. With this project, the project focuses on creating an AI player that acts as an opponent for the main player, with both of them competing for consuming the most dots in the level as a way of introducing a Player vs. AI concept into the Pacman game.

We plan on introducing a multi-agent search agent as an opponent player and letting the agent play the game in an alternate window/environment. The agent tries to beat the game simultaneously as the

human player tries to beat the game in a separate window/environment. We will then compare the performance of both the human and AI players by comparing the amount of dots consumed, the time taken to beat the game, and the result of each iteration.

Project Research

For this project, we looked into various papers and projects for inspiration. PACMAN is an interesting game, in that it provides us with a fairly large field to play with. We can try various AI agents for the player character to compare the differences in the performances. We can also try different AI algorithms for the behaviors of the ghosts. This helps us observe how the Pac-Man character adjusts its performance to deal with new constraints imposed by different ghost behaviors.

The base game of this project is from Giant Jenks' licensed collection of free Python games intended for education and fun. Some of the base game logic and layouts are from 'The PACMAN projects' lectures from UC Berkeley.

The UC Berkeley lectures have great resources to read on implementations of different types of AI agents like search agents, multi-agent search agents, and deep q-learning agents. We decided to use multi-agent search agents, particularly the Minimax agent and Alpha-Beta pruning Minimax agent. Since the game environment of PACMAN has multiple agents operating on it there are scenarios that require coordination between these agents for the player character to survive. The player character has to keep track of ghost characters' positions and also focus on consuming the dots in the layout.

The multi-agent search agents also provide better scalability, adaptability, and robustness. It also allows us to try different layouts of varying sizes and these agents are better to compare performances in these different environments.

We also read through research papers like Brennan's *Minimax algorithm and alpha-beta pruning* and Dan Klein's *Teaching Introductory Artificial Intelligence with Pac-Man* to better understand these algorithms for our implementations.

Problem Environment

The PACMAN game was made in Python, and the existing code for the game has implementation for the main Pac-Man player and behavior for consuming the dots in the level along with the AI behavior of the enemy ghosts. This base game is from the UC Berkeley Projects, and it's an open-source project that UC Berkeley allows individuals to use and modify for educational purposes.

We have added the minimax and alpha-beta pruning algorithms on top of the base Python code. In addition, the AI agents for the Pac-Man character use these algorithms to improve their performance compared to an average human player's performance.

For this problem, the task environment for the AI opponent player agent will be defined as:

- Fully Observable: The agent can see the entire layout of any maze along with the positions of all the ghosts at any point in its traversal within the maze
- Stochastic: The movement patterns of some ghosts are random based on a randomly generated probability
- Dynamic: The environment continuously changes as a result of the movement of the ghosts
- Discrete: The agent always has a limited set of movement actions at any given position in a maze
- Sequential: The agent's movement decisions at any point influences the movement decisions that the agent can make later on in the game
- Multi-Agent: The agent will be interacting with a human agent as well as the AI ghost agents
- Competitive: The agent will be competing against the human player to see who can consume all the dots in a maze/level first

Using the existing Pacman game code in Python, the project will primarily focus on adding additional code for adding an opponent player in the game in addition to adding code for implementing the AI behavior of the opponent player. In addition, even though the ghosts have random wandering movement in the game, the AI used for the ghosts themselves isn't much and doesn't really make decisions on what path to take as the ghosts only decide what path to take once they've collided with a wall in the maze.

Because of this, the AI used for the ghosts has the possibility for further improvements to help make the movement of the ghosts more unpredictable and perhaps more challenging for the player, which could be achieved through using path-finding algorithms.

Additionally, code could also be added so that as the player beats the level each time, the AI behavior of the ghosts and opponent player increases and becomes more intense as a way of slowly increasing the game's difficulty with each playthrough the player experiences.

AI Implementation Approach

Since the AI opponent player is meant to play similar to an actual human player in the game, the AI opponent will need to have behavior for moving around in the maze along with consuming the dots in the level. The consuming dots behavior for the AI opponent will be the same as the existing consuming dots behavior for the main player, and the AI player will also be placed into a different maze from the human player's maze, but both mazes will have the same layout. Since the main player and AI opponent will be competing to see which one can consume the all dots in the maze first, there will have to be additional trackers for the number of dots consumed by each player respectively along with the number of existing dots in each maze. For handling the movement behavior of the AI opponent player, the minimax algorithm will serve as the base algorithm, with the alpha-beta pruning algorithm being a second algorithm used to compare with the base algorithm.

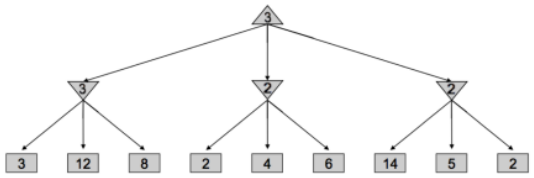

Figure 2: Minimax Algorithm Search Tree Example

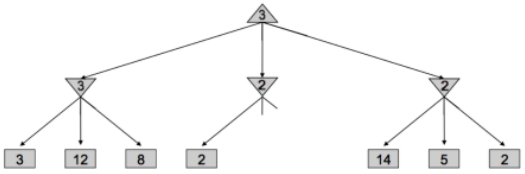

Figure 3: Alpha-Beta Pruning Algorithm Search Tree Example

The minimax algorithm is a game decision-making algorithm that focuses on finding the "best move for a player in a situation where the other player is also playing optimally" (Brennan). The algorithm will examine all possible game states and generate a search tree that assigns a score to each node in the tree. Afterwards, the algorithm represents the root node as a max node that looks for the max value from its children, with the nodes at the next depth being represented as min nodes looking for the min value from its children (Brennan). With each increasing depth in the tree, the nodes alternate between being represented as max and min nodes until reaching the max depth of the tree, where the algorithm then goes through the tree using the minimax rule until reaching a final value for the root node of the tree, as shown in Figure 2. The alpha-beta pruning algorithm is essentially the same thing as the minimax algorithm, however it focuses on trying to "improve the efficiency of the minimax algorithm" by reducing the number of nodes necessary to search through in a given search tree, as seen in Figure 3 (Brennan). With this algorithm, it searches through game states in a similar fashion as the minimax algorithm, but it uses alpha and beta parameters that are initialized to negative and positive infinity respectively and updated when comparing the values of nodes in the tree. When comparing the values of a node

with the parameters, if the current alpha value is greater than the current beta value, then the branch of the tree with the node can be ignored because that branch is guaranteed to be worse than a branch that's already been evaluated (Brennan). Even though the alpha-beta pruning can be effective with simplifying the minimax algorithm, its effectiveness is dependent on how nodes in a search tree are organized.

To implement the minimax algorithm for the AI player's movement, the possible game states/positions for the AI player to move to relative to the ghosts present in the maze will be recorded along with the utility value for the agent within each state. Using the generated states, the search tree will then be generated and searched through using the minimax rule in order to determine the best course of action for the AI player to take when navigating the maze. Implementing the alpha-beta pruning algorithm for the AI player's movement will essentially be the same implementation as done with the minimax algorithm, but alpha and beta variables will be incorporated and kept tracked during the search process with the generated search tree as a way of speeding up the search process for the AI player movement.

AI Evaluation Methods

Since the AI opponent is meant to play similar to an actual human player, the AI opponent will mainly be trying to consume as many dots as possible and also try to avoid getting hit by any of the ghosts in the level. All the different AI implementations will be evaluated based on various aspects. These include experiments, test environments, baseline and evaluation metrics.

Test Environments - We will test the AI in multiple layouts (boards) of the game with varying difficulty and size. The types of boards that will be used are:

- originalClassic - The original game board
- smallClassic - A small version of the classic board
- mediumClassic - A medium-sized version of the classic board
- trickyClassic - A harder version of the medium-sized classic board

The following figures illustrate the different layouts we plan on using as our test environments.

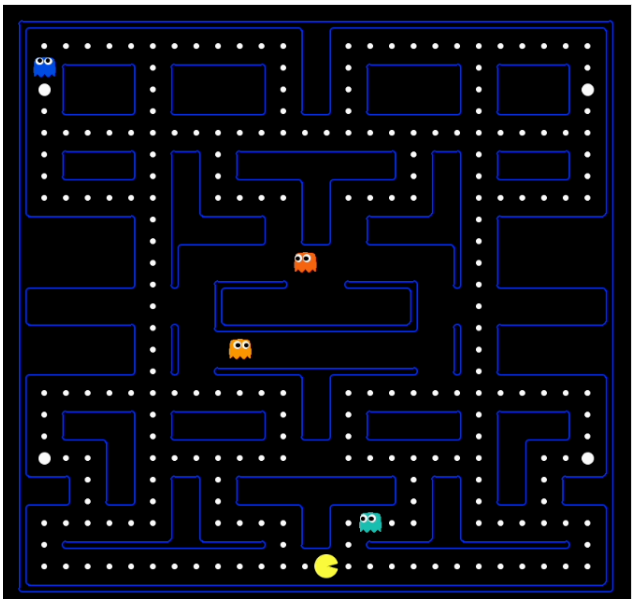

Figure 4: Original Classic Maze Layout

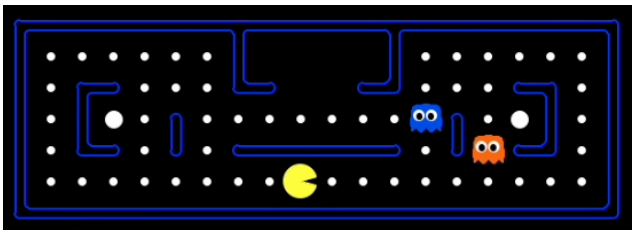

Figure 5: Small Classic Maze Layout

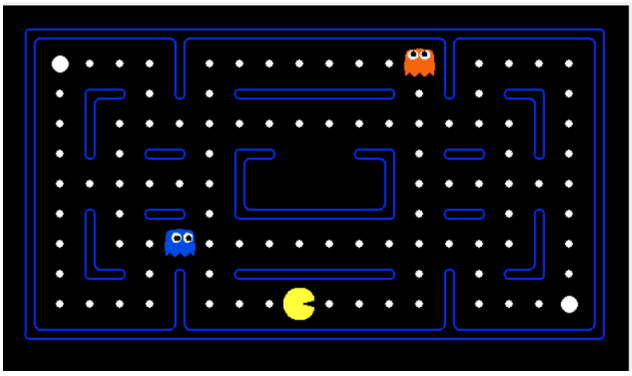

Figure 6: Medium Classic Maze Layout

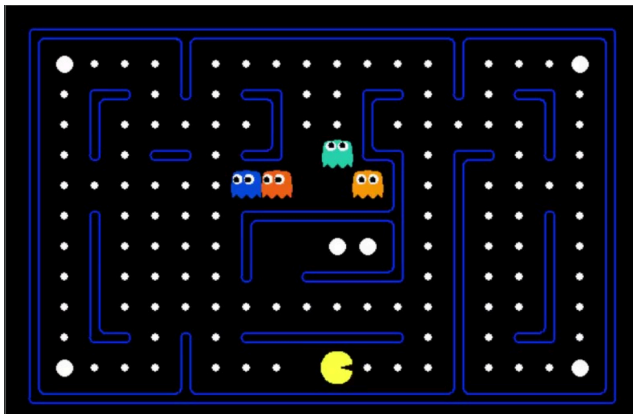

Figure 7: Tricky Classic Maze Layout

Baseline - We will consider the performance of an average human player as the baseline.

Metrics - The following metrics will be considered for evaluation:

1. Consumed dots - The total dots/fruits consumed by Pac-Man
2. Deaths - Number of times Pacman gets caught by the ghosts in a specific map before clearing it (The game will be restarted till either the player wins or till a certain number of games is reached)
3. Win rate - The ratio of wins against total games played with a human player
4. Completion time - Time taken to consume all the dots by the AI player
5. Algorithm specific metrics - We will compare minimax with and without alpha-beta pruning by tracking the decision time taken for both and comparing their averages

Experimentation Methodology - Each of the implementations will be tested against all combinations of the above mentioned environment to find out metrics for each of them.

We plan on testing the performance of a human player versus an AI agent running with the minimax algorithm and alpha-beta pruning algorithm separately. Additionally, we will also run iterations pitting the two AI agents against each other.

We plan on running these different agents simultaneously at first to see who beats the game first. Afterwards, we plan on running them separately multiple times to see how the AI agents perform in different situations including using different layouts and changing the number of ghost enemies present in the maze.

Other testing conditions include different attributes of the AI multi-agent search algorithms as we can observe different behaviors using different depth values and analyze them against different layouts or on the same layout. This will allow us to see how a maze's difficulty impacts the performance of the algorithms and how well the agents can adapt to different environments. This in turn will provide us with enough information to draw a conclusion on which algorithm translates into better performance for the AI player.

Overall, using these metrics can provide useful insight into the AI's performance compared with an average player's performance in the game, which can be helpful for determining what needs to be changed with the AI behavior in order to get the AI player to have an equal/similar performance to an average human player in the game.

Project Significance

By adding an AI opponent player into the game, the project aims to achieve an enhanced single-player experience for the PACMAN game by adding challenge for the main player through the AI opponent. Furthermore, the PACMAN game environment is multi-agent, dynamic and stochastic, so our solution to this problem might have applications to other domains such as autonomous vehicles, robotics or other video games consisting of similar environments.

References

Brennan, A. 2023. Minimax algorithm and alpha-beta pruning. https://medium.com/@aaronbrennan.brennan/minimax-algorithm-and-alpha-beta-pruning-646beb01566c. Accessed: 2024-03-31.

DeNero, J., & Klein, D. 2010. Teaching Introductory Artificial Intelligence with Pac-Man. *Proceedings of the AAAI Conference on Artificial Intelligence*, *24*(3), 1885-1889. https://doi.org/10.1609/aaai.v24i3.18829. Accessed: 2024-03-31.

UC Berkeley. The Pac-Man Projects. http://ai.berkeley.edu/project_overview.html. Accessed: 2024-03-31.