PACMAN Game AI
Final Paper

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 on a separate board as a way of introducing a Player vs. AI concept into the game by comparing the end results. 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 with and without alpha-beta pruning along with A*. The baseline of this project will be the vanilla PACMAN game with no AI agents acting in its environment. Finally, the AI player's performance will be evaluated based on the number of wins achieved in a set of games along with the average score and average amount of time taken to complete a maze/level.

Introduction

In the game PACMAN, 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 Pac-Man 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, Pac-Man 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 level while dodging all the ghosts in said level in order to avoid getting killed by the ghosts and beat the game, as depicted in Figure 1.

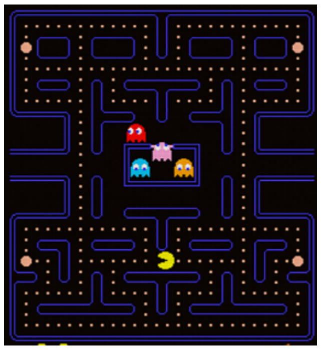

Figure 1: Pacman Game Snippet

Even though the ghosts are 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 will be introducing an AI search agent, both single-agent search agent and multi-agent search, 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 their win rates, average completion/death times, and average scores.

For this project, we referenced lectures and notes from the PACMAN AI course from UC Berkeley. [3] 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 also read through research papers like Brennan's *Minimax algorithm and alpha-beta pruning* [1] and Dan Klein's *Teaching Introductory Artificial Intelligence with Pac-Man* [2] to better understand the algorithms that we were going to use for our implementations.

Game Environment

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 UC Berkeley's PACMAN projects. [3]

The PACMAN game was made in Python, and the existing code for the game has an 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, alpha-beta pruning and A* 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

The project focuses on adding additional code for adding an opponent player in the game and implementing the AI behavior of the opponent player. In regards to ghost characters' behavior in the base game, the ghosts have random wandering movements in the game and they make decisions on what path to take once they've collided with a wall in the maze. The base game also has an additional agent class for the ghosts in the environment called Directional Ghost. Instead of choosing a legal action uniformly at random, the Directional Ghosts choose to pursue the PACMAN character and also try to flee when they are 'scared'.

In addition, the base game also has different sets of layouts of different sizes ranging from small to medium to the original board size. This ensures that testing and learning can be done in various configurations of testing environments. Results from such varying testing configurations provide better insight into the performances of these different agents (A*, minimax, alpha-beta, and keyboard/human) and draw sound conclusions.

In general, an alpha-beta pruning agent should perform better than a minimax agent. Alpha-beta pruning improves upon minimax by eliminating unnecessary branches in the search tree. Data will be collected to test this hypothesis by running these agents side-by-side on separate windows. The agents consider the ghosts in the environment as their adversary rather than the player, and try to maximize their utility. The evaluation function for these agents considers factors like distance to the nearest dot or ghost, and whether ghosts are scared in the current game state.

We also tested the performance of these agents against a Keyboard agent controlled by a human player and a search agent that uses A* algorithm. While minimax focuses on maximizing the evaluation scores of the PACMAN character's actions and minimizing the scores of the ghosts' actions, A* focuses on finding the efficient path to any goal state in a maze-like environment (i.e. PACMAN game environment). Comparing their performances in this context produces interesting results and allows us to draw better conclusions.

AI Implementation Approach

Since the AI opponent player is meant to play like a human player, the AI opponent needs behavior for moving around in the maze along with consuming the dots in the level. The consuming dots behavior for the AI opponent is 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 all the 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, alpha-beta pruning and A* search algorithms are used. This helps us compare the performance of a single-agent search agent with that of a multi-agent search agent in the same PACMAN game environment.

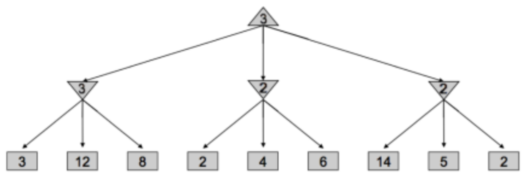

Figure 2: Minimax Algorithm Search Tree Example

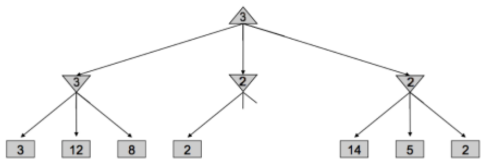

Figure 3: Alpha-Beta Pruning Algorithm Search Tree Example

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". [1] The algorithm will examine all possible game states and generate a search tree that assigns a score to each node in the tree. Afterward, 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. [1] 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. [6]

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. [1] 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. [7] 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. [1] Even though the alpha-beta pruning can be effective in simplifying the minimax algorithm, its effectiveness is dependent on how nodes in a given 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. Two types of evaluation functions were used while testing these algorithms. First, a simple function that scores a possible action by looking at the score of current game state. Second, it considers the Manhattan distance between the player character and the remaining food particles, remaining capsules and ghost characters. So, in this game environment, the ghosts act as adversaries and the pallets/capsules act as rewards. These attributes are used by the evaluation function to score all the possible and legal actions.

A* search algorithm is one of the more popular techniques used in path-finding and graph traversals. A* combines a cost function that measures the actual cost with a heuristic function that estimates the remaining cost to the goal, and the performance of an A* algorithm heavily relies on the heuristic function used. In our implementation, we have used Manhattan distance. Positions with closer food items will have a lesser distance compared to more distant ones. To account for ghosts, the heuristic function assigns a large penalty if a ghost is nearby, essentially deterring pacman from going closer to the ghost. [5]

AI Evaluation Methods

Since the AI opponent is meant to play similarly 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, and 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

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

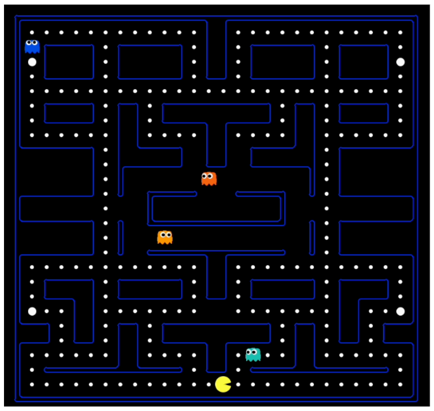
Figure 4: Original Classic Maze Layout

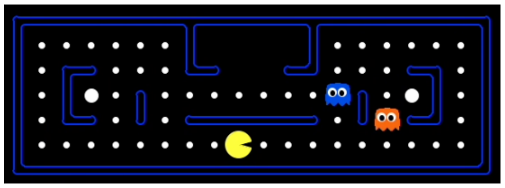
Figure 5: Small Classic Maze Layout

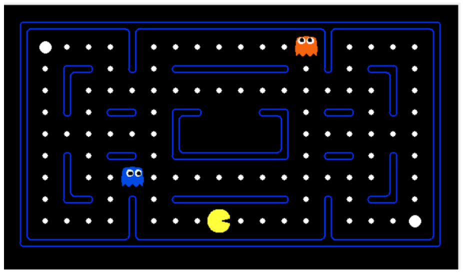
Figure 6: Medium Classic Maze Layout

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

Metrics - The following metrics will be considered for evaluation:

1. Average Score - PACMAN character is scored on its performance (+1 point when consuming food, -1 when no food is consumed, +10 for consuming a scared ghost, +500 for winning and -500 for losing)
2. Win Rate - The ratio of wins against total games played (Generally 10 as used with the tests)
3. Average Time - Time taken by the AI player to complete or die in a game

Experimentation Methodology - Each of the implementations were tested against all combinations of the algorithms and maze layouts.

We test the performance of a human player versus an AI agent running with either the minimax, alpha-beta pruning and A* algorithms separately. Additionally, we also ran iterations pitting the two agents against each other and against an A* agent to compare how a single-agent search agent fare against a multi-agent search agent and observe which agent better adapts to this environment.

We run these different agents simultaneously at first to see who beats the game first. Afterwards, we run them separately multiple times to see how the AI agents perform in different maze environments.

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.

Results

[FYI: Highlighted rows indicate averages for each listed table]

Maze = originalClassic, Algorithm = Minimax

Average Score	Average Time (seconds)	Win Rate (%)
2448.3	412.1	5/10 (50%)
2273.7	367.2	4/10 (40%)
2521.9	430.3	5/10 (50%)
2461.4	470.9	6/10 (60%)
3110.0	495.2	8/10 (80%)
2563.1	435.1	56%

Maze = originalClassic, Algorithm = Alpha-Beta Pruning

Average Score	Average Time (seconds)	Win Rate (%)
2076.9	210.4	4/10 (40%)
2586.0	235.8	7/10 (70%)
2476.3	220.8	7/10 (70%)
2296.7	221.5	6/10 (60%)
2835.6	289.5	8/10 (80%)
2454.3	235.6	64%

Maze = originalClassic, Algorithm = AStar

Average Score	Average Time (seconds)	Win Rate (%)
2693.0	149.9	9/10 (90%)
2384.4	154.1	6/10 (60%)
2487.1	176.5	7/10 (70%)
2578.2	168.8	8/10 (80%)
2213.3	170.4	7/10 (70%)
2471.2	163.9	74%

Maze = originalClassic, Algorithm = Keyboard/Human

Average Score	Average Time (seconds)	Win Rate (%)
2075.6	107.0	7/10 (70%)
1844.4	91.8	5/10 (50%)
1984.8	99.1	6/10 (60%)
2056.9	101.4	7/10 (70%)
1924.7	92.3	6/10 (60%)
1977.3	98.3	62%

Maze = smallClassic, Algorithm = Minimax

Average Score	Average Time (seconds)	Win Rate (%)
709.6	21.2	4/10 (40%)
930.4	30.2	5/10 (50%)
1234.7	30.9	8/10 (80%)
453.9	453.9	2/10 (20%)
853.4	34.6	5/10 (50%)
836.4	114.2	48%

Maze = smallClassic, Algorithm = Alpha-Beta Pruning

Average Score	Average Time (seconds)	Win Rate (%)
673.6	20.4	4/10 (40%)
291.7	15.6	3/10 (30%)
756.3	23.8	5/10 (50%)
178.6	17.1	1/10 (10%)
360.8	17.9	3/10 (30%)
452.2	18.9	32%

Maze = smallClassic, Algorithm = AStar

Average Score	Average Time (seconds)	Win Rate (%)
565.2	14.2	6/10 (60%)
666.2	13.7	7/10 (70%)
850.9	14.8	9/10 (90%)
1030.7	18.7	10/10 (100%)
948.1	15.2	8/10 (80%)
812.2	15.3	80%

Maze = smallClassic, Algorithm = Keyboard/Human

Average Score	Average Time (seconds)	Win Rate (%)
439.4	30.8	4/10 (40%)
320.2	42.2	4/10 (40%)
287.6	39.0	3/10 (30%)
333.7	39.0	5/10 (50%)
411.8	44.3	4/10 (40%)
358.5	39.0	40%

Maze = mediumClassic, Algorithm = Minimax

Average Score	Average Time (seconds)	Win Rate (%)
523.2	34.7	2/10 (20%)
309.3	20.8	1/10 (10%)
1889.3	57.2	10/10 (100%)
1650.7	71.2	9/10 (90%)
1643.6	52.7	8/10 (80%)
1203.2	47.3	60%

Maze = mediumClassic, Algorithm = Alpha-Beta Pruning

Average Score	Average Time (seconds)	Win Rate (%)
1050.5	37.7	4/10 (40%)
1065.7	33.1	4/10 (40%)
1541.1	50.2	7/10 (70%)
1495.2	49.8	7/10 (70%)
1048.9	44.5	5/10 (50%)
1240.3	43.1	54%

Maze = mediumClassic, Algorithm = AStar

Average Score	Average Time (seconds)	Win Rate (%)
1584.1	28.7	9/10 (90%)
1438.2	30.1	8/10 (80%)
1209.4	24.2	7/10 (70%)
1358.2	28.4	8/10 (80%)
1362.3	28.2	8/10 (80%)
1390.4	27.9	80%

Maze = mediumClassic, Algorithm = Keyboard/Human

Average Score	Average Time (seconds)	Win Rate (%)
698.5	25.1	4/10 (40%)
478.5	25.6	2/10 (20%)
472.6	16.8	4/10 (40%)
482.8	23.4	3/10 (30%)
696.6	24.2	4/10 (40%)
565.8	23.0	34%

Observations

Each algorithm was tested for 5 rounds, each consisting of 10 runs, and these tests were run on 3 different layouts, so 600 total games were used for testing these algorithms. With the minimax and alpha-beta algorithms in particular, they were tested with a depth value of 3.

Looking at the overall performance of A*, it consistently wins more games when compared to other algorithms and the human player. This is due to the fact that the heuristic function provides an easy way for Pac-Man to avoid ghosts. [7] Just by running away from ghosts when they are near and trying to pick the closest route to the next food pallet, A* is able to win in simpler scenarios, which allows A* to have a quick win with relatively low time taken and processing. Since the heuristic is limited to avoiding ghosts, even though A* has higher winning rates, we see that the score is comparable to other algorithms with lower winning rates. This is because the score gets a significant bump up by consuming a bunch of ghosts while navigating the maze.

One interesting observation about minimax and alpha-beta algorithms can be inferred from comparing small, medium, and original layouts. For smaller layouts, the minimax performs better than alpha-beta while for larger layouts, alpha-beta trumps over minimax. One possible explanation could be that the game tree is not large enough in smaller layouts for the alpha-beta algorithm to produce a significant difference in the performance of the agents. Because of this, the basic score evaluation function produces sub-par performance as expected for both minimax and alpha-beta algorithms. This is because the basic score evaluation function only considers the current game state and doesn't consider ghost positions or capsule positions.

But with a better evaluation function, we can observe improvement in the win rates as well as the average score for each run. While comparing the average scores and the time taken for each agent to complete a run using both algorithms, we can observe that alpha-beta spends its time more efficiently than the minimax algorithm.

We also observed that in some rare cases, A* might get stuck in an indefinite loop trying to pick the best path. This can be caused by placement of ghosts in particular locations, such that the path cost for multiple paths comes about to be equal, which leads to it to keep evaluating the paths endlessly. Scores are deducted for every move with no reward, so the algorithm is not functioning optimally if the character wanders around a part of the layout with no clear idea of what action to take or which direction to move.

By looking into some runs of the minimax algorithm, the characters have lower scores even if they have more time spent. Sometimes, lower scores are observed for runs with higher win rates, in the case of the minimax algorithm. It is because the character wastes time wandering around the environment when using the minimax algorithm. This provides an edge to the alpha-beta algorithm as this kind of wandering behavior is mostly avoided due to some nodes of the game tree being pruned by the algorithm.

The runtimes of A* are considerably smaller than minimax and alpha-beta algorithms, which was because both multi-agent search algorithms take less risky moves at all times. While testing, the character typically kept its distance from the ghosts when using minimax or alpha-beta. However, when using A*, the agent usually got closer to the ghosts since avoiding ghosts was handled with the heuristic.

For testing the performance of human players, observations from 3 players playing the game were recorded. By comparing the win rates, the performance can be evaluated as average. The scores are a bit lower in smaller and tricky layouts when compared to the original layout. In the cases of smaller layouts, the room for error is very small with very little room to breathe. Humans do not have an evaluation function calculating scores, taking into account every attribute in the environment that might affect the performance, for every possible move to choose the best among them. So the performance can be lower when compared to these AI search algorithms. But the performance in the original layout is on par with these agents, except maybe the A* agent.

Conclusion

The agent using the A* algorithm performed the best across all the layouts, with its win rate and average scores being consistently higher than the other agents. This was likely because the heuristic used for the A* algorithm finds the optimal path efficiently compared to the other algorithms. The heuristic guides the search towards more promising paths and helps the agent make the best decision at each move.

While the adversarial search property of both minimax and alpha-beta search algorithms can help anticipate ghost movement and mitigate the risks, the uncertainty in the game environment reduces the efficiency of those algorithms. Even though the A* algorithm typically got the highest win rates and average scores, the human player tended to get lower average times, implying that the human player still tended to either complete the maze or die in the level faster compared to the A* agent.

As expected, the alpha-beta algorithm performed better than the minimax algorithm in the original layout since alpha-beta search typically produces efficient pruning and reduced search space. Despite that, the minimax algorithm managed to slightly outperform in smaller layouts because the effects of pruning weren't fully realized in a small search space/game tree. The performance of AI algorithms were on par with the performance of human players, and in smaller layouts the AI agents consistently outperformed human players.

Improvements can be made in the heuristic function of the A* algorithm and the evaluation function of the minimax and alpha-beta algorithms. For instance, the impact of ghost proximity and the accessibility of the remaining pellets/capsules could be considered when evaluating a possible action in the heuristic function of A* algorithm. Some dynamic ghost behavior handling like how long the ghosts should be scared or their proximity to pellets/capsules could also be considered for finer tuning the parameters used for the evaluation functions in the minimax and alpha-beta pruning algorithms.

References

[1] 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.
[2] 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.
[3] UC Berkeley. The Pac-Man Projects. http://ai.berkeley.edu/project_overview.html. Accessed: 2024-03-31.
[4] Nosrati, Masoud, Ronak Karimi, and Hojat Allah Hasanvand. "Investigation of the*(star) search algorithms: Characteristics, methods and approaches." *World Applied Programming* 2.4 (2012): 251-256.
[5] Y. Zou, "General Pacman AI: Game Agent With Tree Search, Adversarial Search And Model-Based RL Algorithms," in 2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE), Zhuhai, China, 2021 pp. 253-260.
[6] Bruce W. Ballard,The *-minimax search procedure for trees containing chance nodes,Artificial Intelligence,Volume 21, Issue 3,1983,Pages 327-350,
[7] J. Schaeffer, "The history heuristic and alpha-beta search enhancements in practice," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 11, pp. 1203-1212, Nov. 1989, doi: 10.1109/34.42858. keywords: {History;Minimax techniques;Iterative algorithms;Decision trees;Councils;Testing},
[8] Korf, Richard E. "Artificial intelligence search algorithms." (1999): 22-17.