ECE 470

Simplified PacMan Final Report

Team PacMan

By

Doug Cruz	Caelum Dudek	Brent Machado	Chloe Zacharias
V00952906	V00967651	V00913331	V00961884

Table of Contents

Abstract

For our project, we developed a Pacman game utilizing the adversarial search problem approach, with the greedy strategy and the Ghost using A* algorithms. The game was built with a combination of Godot for game creation, python for implementing the greedy algorithms, and a Python relay for communication between the two. The objective was to gain insights into solving NP-complete problems and, with Pacman being a classic game that most people know while also being NP-complete it was perfect for this project. We were able to successfully implement the game and algorithms while demonstrating an understanding of NP-complete problem-solving techniques within a more familiar game context thereby increasing our understanding of AI problem-solving techniques.

Introduction

Video games have been a staple of the computing field since their inception. One of those games was Pacman, and its innovative concept and aesthetic have been capturing the eye of programmers since.

In the original Pacman, there are 4 Ghosts, each with different personalities in the form of simple path-finding algorithms. These adversaries provided the player with many intricate challenges that classic computer algorithms have struggled to solve. With the advent of AI, this problem has multiple ways to be solved using various pathfinding algorithms. In our version of Pacman, we have simplified the game and reduced the number of Ghosts to one, which is representative of the early stages of Pacman. By doing so we can apply various AI methods to direct each player.

Pacman is a classic game that is considered to be NP-complete and therefore very difficult to solve efficiently[1]. By working on problems like Pacman, we can better understand how to solve NP-complete problems, as they are found everywhere and are essential to our modern lives.

Related Work

Pathfinding is a major component when controlling non-player characters (NPCs) within games, with the A* algorithm being a common choice for this [2]. Renowned for its efficiency when navigating old game environments, A* aims to minimize the total estimated solution cost and avoid paths that are already expensive. Despite A*'s presence in pathfinding discussions since 1968[2], it struggles to keep pace with the increasing intricacies of modern game design.

In "A Review on Informed Search Algorithms for Video Games Pathfinding" [2], A. Y. Kapi discusses how A* can be enhanced through various avenues, including improving the heuristic function, optimizing search algorithms and other aspects. One notable enhancement is the Multi Weighted-Heuristic function (MWH), this can improve space complexity but it introduces an increase in time complexity, specifically concerning state-space size [2]. The author acknowledges the need to mitigate this dependency. Additionally, in terms of search algorithms, the author recommends combining search methods. With the best combination of note being A* with Dynamic Pathfinding.

Another paper, "Hybrid Pathfinding in StarCraft" by J. Hagelback [3], presents a different method of pathfinding when priorities change. The paper discusses the changing navigation search algorithms NPCs use when responding to varying environmental conditions. Specifically, the strategy is to use A* when there are no enemies around then transition to flocking with the boids algorithm when enemies approach. However, it was mentioned that the usefulness of this strategy is dependent on the complexity of the NPC and whether or not it needs to surround the enemy.

The networkx library is a valuable tool for conducting intricate graph analysis, offering many different functionalities for these tasks [4]. Most noticeably the library provides an implementation of A*This library helps streamline the process of understanding and visualizing algorithmic behavior within gaming contexts. By using libraries like this developers can streamline the integration of pathfinding capabilities into their games without needing to develop these algorithms from scratch. These libraries help facilitate rapid prototyping and experimentation within game design.

Problem Formulation

The primary goal is to demonstrate the application of artificial intelligence methods in navigating adversarial problems within the world of a simplified PacMan game featuring one Pacman and one Ghost. The challenge lies in controlling both characters using different AI techniques to highlight the interplay between predatory and prey within a constrained environment.

The state space of the game at any given moment is defined by the positions of Pacman and the Ghost on the grid, along with the distribution of pellets across the game board. Walls and other obstacles provide a fixed structure within which the sprites navigate. The initial state begins with Pacman and the Ghost positioned in predetermined locations on the grid. The pellets are also predefined, filling points on the grid.

Both Pacman and the Ghost can move in four directions: up, down, left and right, subject to the constraints imposed by the walls and boundaries of the game board. The transition model is a change in the position of each sprite on the grid. This change is deterministic, based on the chosen direction of movement. The goal state for Pacman is to collect all the pellets, while the goal state for the Ghost is to catch Pacman. Lastly, the path cost incurs a uniform cost for each step, and it is the number of steps taken. For Pacman, the optimal path minimizes the total steps to collect all the pellets while for the Ghost, the cost is minimized by catching Pacman in the fewest steps.

Methodology

To solve the problem of creating a Pacman AI, we first created a copy of the game in the Godot game engine. While Godot 4 is still in the growing phases, we learned valuable information on how it works and this allowed us to modify it sufficiently to control it with our AI process. Once the game was functional, there came the problem of how to implement the type of inter-process communication that would be used to connect it with our AI process.

The AI process, which is used to direct the players(Ghost and Pacman), works best if it drives the program by sending movement instructions to the game. Unfortunately, Godot 4 cannot act as a REST server, but it can act as a client. To overcome this problem we introduced an intermediary relay server, which can turn polled requests from the game into a form that allows the AI process to drive the game asynchronously as seen below(Figure 1,2,3).

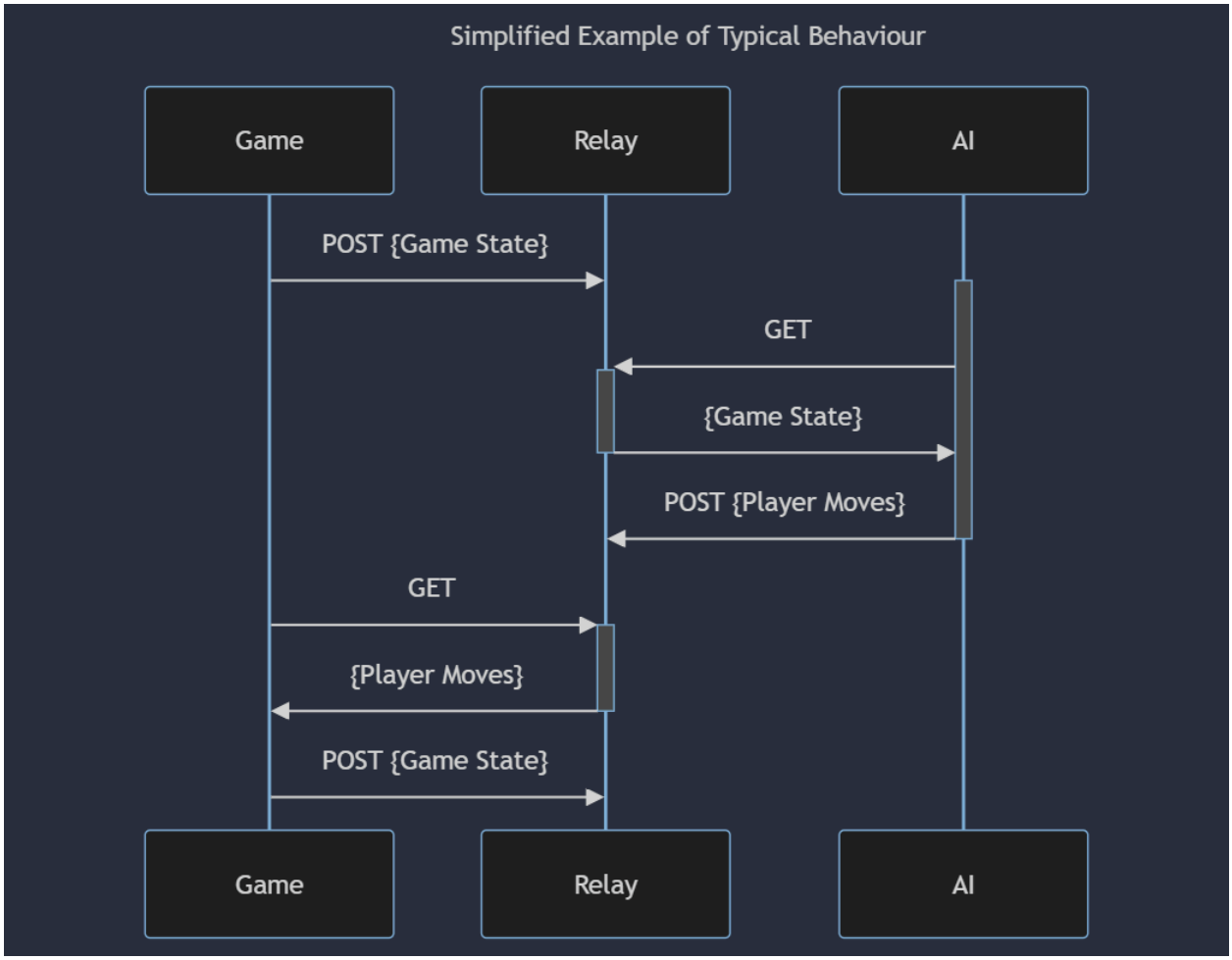

Figure 1, Simplified Example of Typical Behaviour of controlling the game with the AI

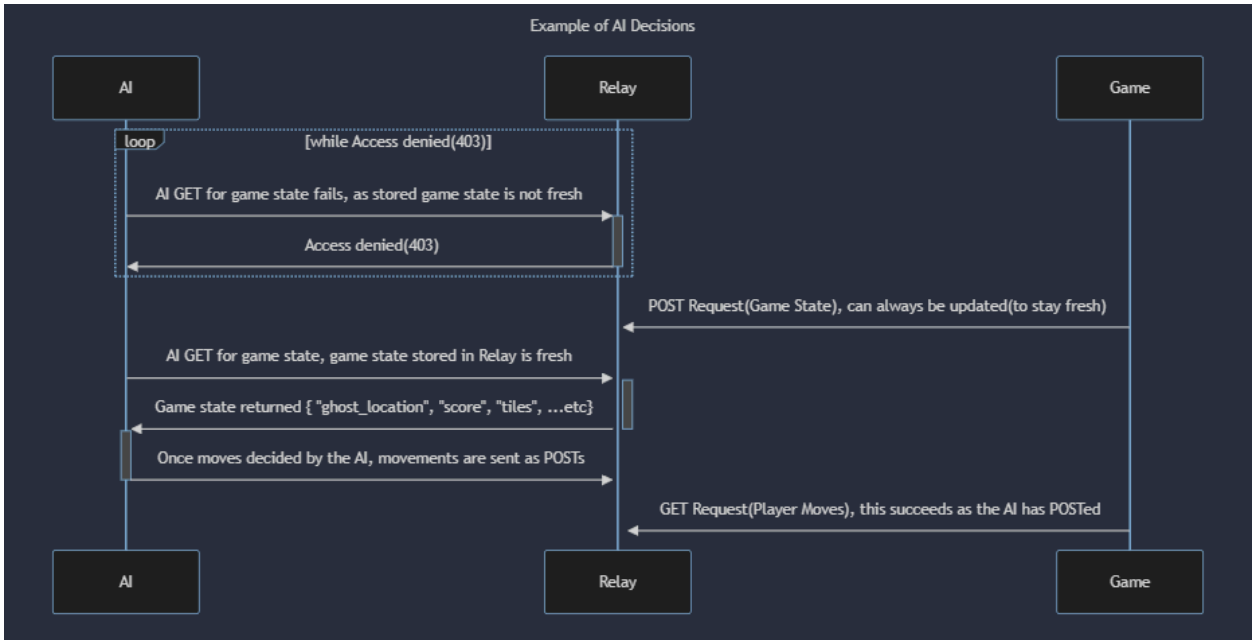

Figure 2, Example of AI Process Decisions

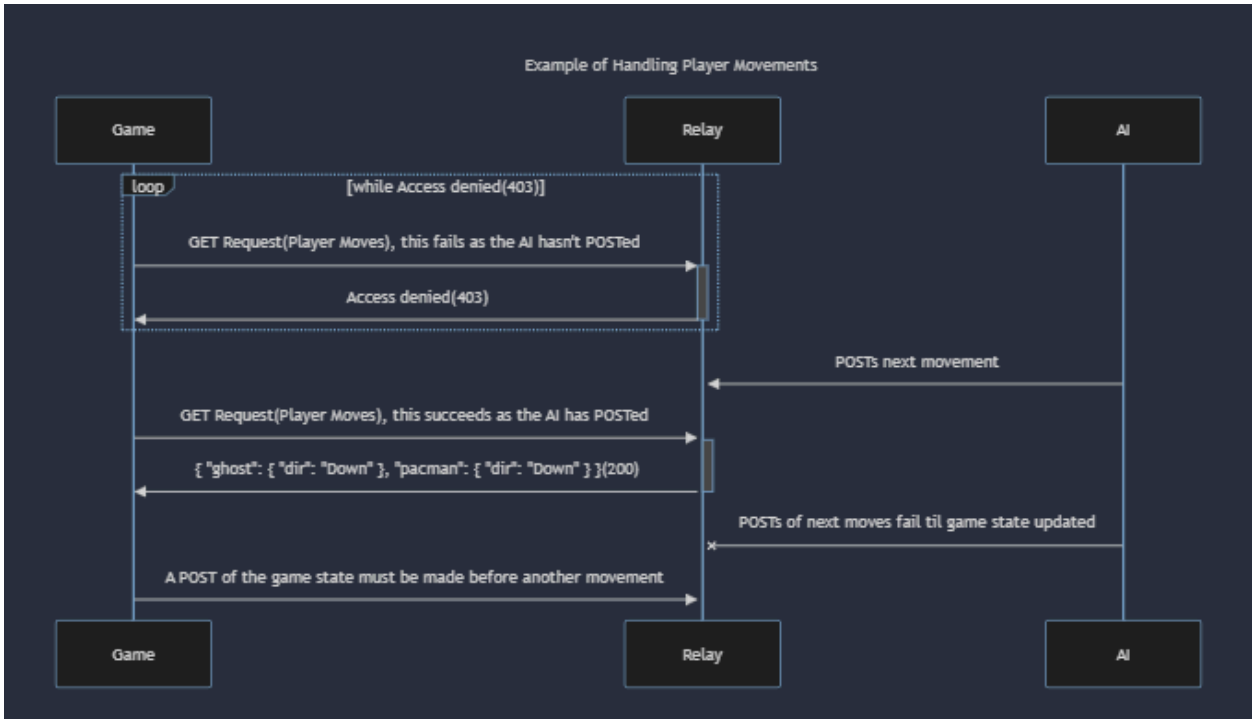

Figure 3, Example of Handling Player Movements

This approach solves most of the anomalous behaviour that we experienced when trying other methods, such as sharing memory or having the AI process also be the server for communication.

Once we had solved the issue of communication we then needed to convert the current state of the game, which was sent as a JSON text string, into Python objects that could be used to perform our analysis and then provide directions to the game for our players. Using a library called networkx, we were

able to convert our maze, a 2d array of numbers representing tiles, into a graph which can be traversed using any algorithm of our choosing. Our AI uses the A* and a modified greedy algorithm to determine the next direction of the Ghost and Pacman respectively. The determined instructions are delivered to the game, and each direction moves the players a single tile in a given direction. The directions are continuously sent until either the game is won and the pellets have all been eaten, or Pacman is caught by the Ghost. As we experimented, we necessarily edited the heuristic functions of each player until we got results that we were happy with.

Within the unit test labelled “test_navigate_to_Pacman”, the A* algorithms works by firstly storing the Pacman location and the Ghost location, then calling the networkx method for A* with the graph, these mentioned locations, and a heuristic function. Our heuristic function leverages the Euclidean distance between the location of Pacman and the location of the Ghost. The Euclidean distance is a measure of the straight-line distance between two points.

Within the unit test labelled “test_greedy_algo”, the modified greedy algorithm works by again storing the Pacman location, the Ghost location, and the locations of every single pellet within the graph. We then obtain the K shortest paths between Pacman and the Pellet nodes defined using Dijkstra’s Algorithm. Utilizing Dijkstra’s is a safe choice since there are no negative weights within the graphs. Secondly, we calculate the Manhatten distance between the final node of each of the K shortest paths(to pellets) and to the Ghost. The Manhatten distance is a metric used to measure the distance between two points in a grid-based distance, so only up, left, down, and right, rather than the straight line distance. Given the shortest paths, and the Manhattan distance, we lastly define a weighted heuristic function using scaling factors, and we take the path with the best heuristic score.

Results and Discussions

The A* algorithm for navigating the Ghost toward Pacman by leveraging the Euclidean distance as the heuristic function resulted in efficient and dynamic pathfinding. By using the Euclidean distance as opposed to the Manhatten distance, the algorithm prioritizes moving the Ghost in a direction the direct distance to Pacman, potentially leading to cornering or more unpredictable engagement between the Ghost and Pacman. A disadvantage of the A* algorithm is that the heuristic does not account for the remaining pellets, so later in the game it is easier for Pacman to outmaneuver the Ghost with large, spacious movement.

Utilizing the modified greedy algorithm for Pacman indicated several qualitative results. The navigation for Pacman was relatively safe because the heuristic considers the Manhattan distances, and prioritizes such with a scaling factor, therefore leading Pacman to be more likely to avoid paths that would lead to close encounters with the Ghost. Secondly, because the K shortest paths are considered, Pacman has a higher potential to explore different areas of the maze rather than strictly following the shortest path. A more sophisticated AI for the Ghost could help avoid predictable and exploitable patterns. A main disadvantage of using the modified greedy algorithm is that later in the game when many of the shortest paths are actually quite long, the Ghost will move a lot between the start and end of the path traversal so the performance is not adequate. This is because the Ghost traversal is not accounted for in the heuristic. The biggest issue with, the Greedy approach is using the property of local optimums. This leads to the AI making short-term optimal decisions which may not be optimal later on due to map characteristics, pellet locations, and future Ghost locations.

To fix this, we would have to employ a more sophisticated AI algorithm such as Minimax, as it considers more factors such as dead ends, and future Ghost movements. However, many of these adversarial algorithms require the game to be turn-based, meaning that some modified and nuanced version would need to be created.

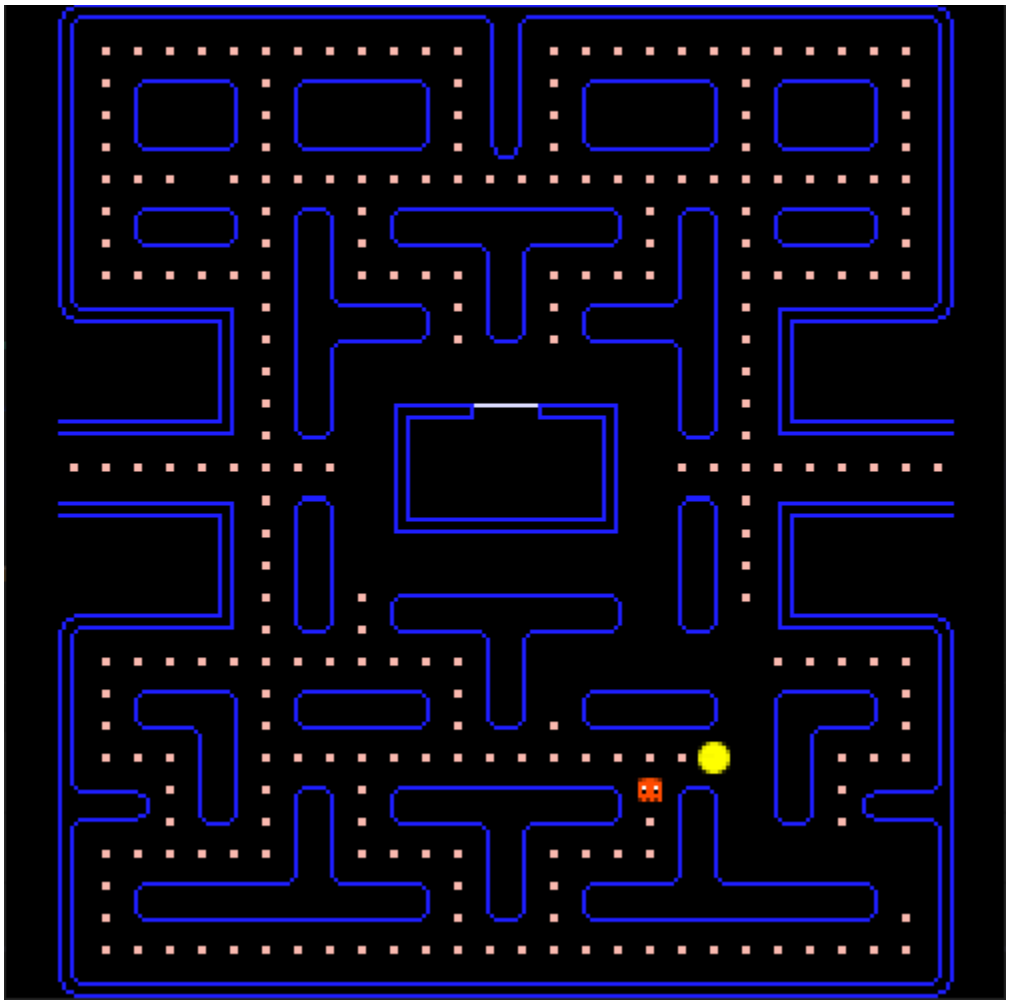

Figure 4, Example of short-term gains, not making optimal overall choices

Conclusion

Our exploration of using AI within the classic Pacman game has resulted in interesting insights about the potential for navigating and solving adversarial problems. By simplifying the game environment to focus on the interplay between a single Pacman and a single Ghost, we've demonstrated how A* and a modified greedy algorithm interplay against each other in a strategic setting and the difficult challenges of expectations from the heuristic function.

While our Pacman, and Ghost were not the most intelligent of AI. They were successfully able to succeed in their base functions, even if their interplay may not have been the most optimal. Additionally, we learned a lot about the importance of problem sub-structures and gained knowledge with networking and coding in Godot and Python.

Future Work

There are many things that can be explored further to enhance the game or improve our project; some of them being:

Enhanced AI Behavior: Try different heuristics and evaluation functions for A* to improve the decision-making of Pacman and the Ghost. This could include adding things like power pellets that allow Pacman to attack the Ghost and then adding that to the evaluation. Also, we could look into prioritizing parts of the maze, as well as using specific adversarial AIs such as Minimax.

More Levels: We could add more levels and explore what happens when Pacman or the Ghost is able to move faster. Also, we can look at how the size of the maze affects the behavior of the AI.

Add more Ghosts: Adding more Ghosts to the game that use different heuristic functions to see if they are able to work together to catch Pacman. This would also increase the complexity of the game and allow us to observe how multiple Ghosts interact and coordinate movements.

GitHub

This is the GitHub we used to house all our code:
https://github.com/CaelumD25/networked_Pacman

References

[1] Emerging Technology from the arXiv, “Pac-Man Proved NP-Hard By Computational Complexity Theory,” *MIT Technology Review*, Jan. 26, 2012. Accessed: Feb. 19, 2024. [Online]. Available: https://www.technologyreview.com/2012/01/26/188193/pac-man-proved-np-hard-by-computational-complexity-theory/

[2] A. Y. Kapi, “A review on informed search algorithms for video games pathfinding,” *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 3, pp. 2756–2764, Jun. 2020. doi:10.30534/ijatcse/2020/42932020

[3] J. Hagelback, “Hybrid Pathfinding in StarCraft,” IEEE transactions on computational intelligence and AI in games., vol. 8, no. 4, pp. 319–324, 2016, doi: 10.1109/TCIAIG.2015.2414447.

[4] “What is NetworkX?,” *NVIDIA Data Science Glossary*, 2023. https://www.nvidia.com/en-us/glossary/networkx/