Initial Report

Dylan Ahearn, Aubrey Conners, Christopher Siems, Samuel Szymanski

2024 February 19

1 Abstract

We aim to implement the classic arcade game, Pac-Man, along with sets of selectable algorithms controlling both Pac-Man and the ghosts. The user of this program will be able to set up instances of the game selecting which algorithms will control Pac-Man and the ghosts. These algorithms will not necessarily be good at playing Pac-Man; some may play well, others may not. Our algorithms will be run against each other and their performance will be tracked. For Pac-Man algorithms, we will track performance metrics such as time to level completion, deaths, score, and ratios between these metrics. For ghost algorithms, we will track performance metrics such as number of Pac-Man kills, and time between Pac-Man kills. We will use Python's Pandas and Matplotlib libraries for data collection and processing. Our primary goal is to use our knowledge of programming and understanding of algorithmic fundamentals to explore the practicality of different algorithms when applied. Through our early research, we have found existing algorithms built for Pac-Man and the ghosts that we intend to include alongside our homebrewed algorithms as points of reference. We are still in the brainstorming phase for our algorithms, but some early ideas include an algorithm to pick random spaces on the board and move to them (the BogoSort of Pac-Man algorithms), using BFS to search the maze for the earliest possible win, algorithms blind to the layout of the board and/or the locations of the enemies, more courageous and more fearful algorithms, etc. The project will primarily be built in Python and use libraries such as Pygame (a Python library for making 2D games), Pandas (for tracking data on how well each algorithm performs), Matplotlib (to visualize the data produced with Pandas), and potentially Pytorch (to power machine learning models). We aim to leverage our existing skills to overcome obstacles related to algorithm implementation and game development. This project gives our group the opportunity to apply what we are learning in Algorithms, adapt to emerging challenges, deepen our understanding of computer science concepts, and improve our collaboration abilities. Time permitting we also plan on exploring the potential of procedural generation of the mazes and machine learning models trained to play Pac-Man or control the ghosts.

2 Motivation & Concrete Problem Statement

Our goals for this project include:

- Exercising our programming and broader software development skills.
- Creating a functional and playable implementation of Pac-Man with at least one maze for the algorithms to run through and all four ghosts (Blinky, Pinky, Inky, and Clyde).
- Building several algorithms for both Pac-Man and the ghosts, at least three for Pac-Man and three for the ghosts.
- Demonstrating our learning of algorithm concepts.
- Tracking algorithm performance to determine which algorithms play Pac-Man or act as a ghost, best. Data on time to win, score, deaths, kills, etc. will be tracked and analyzed using Pandas and Matplotlib.

And if time permits:

- Developing a procedural generation algorithm for the mazes.
- Constructing a machine learning model for controlling Pac-Man and/or the ghosts.

The primary beneficiaries of this project are us the developers. This project exists mostly as a fun exercise in the application of the computer science ideas we have been learning for years. That is the source of our interest in building the product we are working on and the source of its value.

With a lot of hard work, this project could become a full-fledged game/toy suitable for sale on digital marketplaces such as Steam, or with more work ported to other platforms such as mobile. Alternatively, the project could be made open source with the goal of crowd sourcing a even greater set of algorithms for Pac-Man and the ghosts.

3 Related Work

30 Weird Chess Algorithms: Elo World - A funny video by PhD computer scientist Thomas Murphy, known online as tom7. In the video Tom creates a similar project with 30 chess algorithms of dubious skill and, like us, Murphy pits these algorithms against each other and tracks their performance. This is a primary inspiration for this project. We will be doing with Pac-Man what Murphy did with chess[12].

Playing Pacman with Multi-Agents Adversarial Search - An article describing a project very similar to the one we are currently building. The author built a set of algorithms for playing Pac-Man, measured their performance, and compared the algorithms. This is a great reference for us as this author has built a very similar product to what we are trying to make currently[10].

Pacman in Python with Pygame - A simple Pac-Man implementation written in Python using Pygame. We could use this as a reference for the game implementation itself. This is solely the Pac-Man game

without any sort of built in maze traversal algorithms, the ghosts even follow set paths. There are many Pac-Man implementations on the internet for us to reference, we selected this one to be our primary reference due to it being built in Python with Pygame and due to it simplicity. Because it is so simple we will be able to effectively and quickly modify it for our needs[6].

Pacman Python Pygame - Another simple Pac-Man implementation written in Python using Pygame. This is implementation will serve as a secondary reference for the basic structure of the game. Much like the previous reference, this implementation was selected due to it using the same technology stack that we will be using and due to its simplicity[7].

Programming a Pac-Man in Python Introduction - Another Pac-Man implementation in Python with Pygame. This implementation is different from the previous two in a significant way. Instead of using a grid system, this implementation uses a node graph system to understand and process the maze. For some of our algorithms it could make more sense to use a node graph to process the maze than a grid. For that reason, we are including it in our references[9].

The Pacman Projects - A collection of resources used to teach UC Berkeley's CS188, Introduction to Artificial Intelligence, class that uses Pac-Man as a method to teach both traditional algorithms and basic artificial intelligence concepts. This is a great reference for educating ourselves on known implementable algorithms for Pac-Man such as BFS, DFS, and A* to name a few[5].

Maze-solving algorithm - A Wikipedia page summarizing a few common and famous maze solving algorithms, not necessarily Pac-Man algorithms, that could be modified and implemented in our project. Ideally we would create implementations for each of these algorithms. Some algorithms summarized on the page are the Hand on Wall Rule (suitable for a Pac-Man that could never win or easy to dodge ghosts), the Pledge Algorithm, Trémaux's algorithm (suitable for Pac-Man or the ghosts if reworked), Maze-routing algorithm, among others[1].

Breadth-first search - A Wikipedia page summarizing the ideas behind BFS, one of the most suitable algorithms for searching mazes in general. BFS works by searching each possible path at the same time until a target condition is met. This page also summarizes complexity and completeness for BFS. We will definitely use BFS in this project[3].

Depth-first search - A Wikipedia page for summarizing the ideas behind DFS. DFS is likely less suitable for optimal path finding than BFS, however, it could still provide an interesting comparison for the other algorithms. DFS works by searching each path completely before moving onto the next path. Like the wikipedia page for BFS, this page provides complexity and completeness of the algorithms. DFS could be used in procedural generation for the mazes[4].

A* search algorithm - A Wikipedia page summarizing the A* path finding algorithm. This is a common algorithm used for ghosts in Pac-Man implementations. This algorithm is more complex than BFS or DFS and will require greater research, however it seems promising for quickly computing optimal paths. We will investigate this algorithm for potential usage in our project[2].

Maze generator and solver - A Python project to procedurally generate mazes. To be used as reference for maze generation later on in the project. Not necessarily a Pac-Man arena generator but could be repurposed to do such a thing[8].

Deep Reinforcement Learning - A Python project teaching a machine to play Snake using Pytorch Deep Q-Learning. We may use this and other projects like it to learn the machine learning knowledge necessary to build Pac-Man playing models[11].

4 Potential Challenges

Some of the potential challenges we expect to face include:

Implementing Pac-Man itself. While creating the game in Python is a well documented task that we will most likely be able to accomplish relatively quickly if we approach the problem well, it is still likely we will run into trouble during implementation. To alleviate this troubles we will not be shy about using our references and resources for building solutions for the solved problem that is Pac-Man in Python.

Building the game so as to accommodate all the algorithms we want to implement may cause compatibility issues. Each algorithm will have slight or great differences in how they operate. This means that we will have to build systems to accommodate each algorithm. As an example, one algorithm may process the level as a grid of spaces, whereas another may understand the level as a node graph. We must make sure that systems are in place for both of these algorithms to function.

Another problem we are likely to run into is figuring out how exactly we should translate our conceptual algorithms to code. We will use all resources available to solve this issue, our references, classes, the internet generally, and our group mates.

Current skills:

- Python
- Pygame
- Pandas
- Matplotlib
- Github
- LaTeX
- Recursion
- Iteration
- Backtracking
- Pseudocode
- Agile
- Self-sufficiency

Skills to develop:

- Pytorch
- Procedural generation
- Machine learning
- BFS
- DFS
- Division of labor

5 Timeline

February 19	Initial report due
Late February	Algorithm conceptualization and building basic Pac-Man implementation
Early March	Begin implementing algorithms
March 18	Midterm report due
Early April	Explore possibility of using procedural generation and/or machine learning
Week of April 22	Final presentation given
Week of April 22	Final report due

References

[1] Maze-solving algorithm, December 2023. URL: https://en.wikipedia.org/wiki/Maze-solving_algorithm.

[2] A* search algorithm, February 2024. URL: https://en.wikipedia.org/wiki/A*_search_algorithm.

[3] Breadth-first search, January 2024. URL: https://en.wikipedia.org/wiki/Breadth-first_search.

[4] Depth-first search, January 2024. URL: https://en.wikipedia.org/wiki/Depth-first_search.

[5] Pieter Abbeel, John DeNero, and Dan Klein. The pac-man projects, September 2014. URL: http://ai.berkeley.edu/project_overview.htmlhttp://ai.berkeley.edu/project_overview.html.

[6] hbokmann. Pacman in python with pygame, October 2021. URL: https://github.com/hbokmann/Pacman.

[7] Jan Jileček. Creating pac-man clone in python in 300 lines of code or less— part 1, December 2022. URL: https://github.com/janjilecek/pacman_python_pygame.

[8] jostbr. Maze generator and solver, April 2021. URL: https://github.com/jostbr/pymaze.

[9] JustToThePoint. Programming a pac-man in python introduction, May 2022. URL: https://justtothepoint.com/code/pacman/.

[10] Davide Liu. Playing pacman with multi-agents adversarial search, February 2020. URL: https://techs0uls.wordpress.com/2020/02/13/playing-pacman-with-multi-agents-adversarial-search/.

[11] maurock. Deep reinforcement learning, January 2021. URL: https://github.com/maurock/snake-ga.

[12] tom7. 30 weird chess algorithms: Elo world, July 2021. URL: https://www.youtube.com/watch?v=DpXy041BIlA&t=12s.

Typeset with LaTeX.