Midterm Report

Dylan Ahearn, Aubrey Conners, Christopher Siems, Samuel Szymanski

2024 March 18

System Overview

Our project is built around three main processes. These parts are the testing of our set of Pac-Man algorithms, the collection of data about the performance of each Pac-Man algorithm, and the processing of that data. The first process is having each algorithm run many times through a standard game of Pac-Man. The second process is collecting data from each algorithm about their performance for that individual run and storing that data in a dataset. And the final process is using the dataset we have developed to create graphs of the performance of each algorithm and extract key data points from that stored performance.

System Architecture

There are four components that will make up the whole project.

The Game

Probably most obvious component is the Pac-Man game itself. The Pac-Man version that we are using is a faithful Pac-Man implementation built in Python using Pygame that was forked and modified for our purposes from an open-source Pygame Pac-Man implementation built by Sebastian Rohaut, usawa/pypacman on GitHub. This implementation is a very high-quality clone of Pac-Man that was easy to set up and begin modifying for our purposes. The primary change we have made to Rohaut's Pac-Man is editing the code to allow instances of the Pac-Man object, that represents Pac-Man himself, to be constructed with an algorithm passed as an argument in the constructor.

The Algorithms

This component is really a set of components. This is the set of algorithms that will be used to control Pac-Man. Each algorithm will be contained within a class representing that algorithm and containing

all its logic. Instances of these algorithm classes will be instantiated and passed as arguments to the Pac-Man constructor to set that algorithm as the controller for that instance of Pac-Man. Some, but not all, ideas we are moving forward with to implement in our algorithms are:

- Randomness. To be used as a baseline of comparison. How well does Pac-Man perform when he is moving randomly? (Spoiler: Not well!)
- Predictable patterns. Algorithms that play the game in predictable ways.
- Choosing points. Algorithms that select points in the maze to move to, based on some criteria, and move to them using some pathfinding algorithm.
- Breadth First Search. Will serve two purposes in our project, to serve as a point-to-point pathfinding algorithm and a modified version will serve as a complete Pac-Man algorithm.
- A*. One of the algorithms suitable for point-to-point pathfinding.
- Dijkstras algorithm. One of the algorithms suitable for point-to-point pathfinding.

Data Collection

Data about the performance of each algorithm will be collected at the end of each algorithm's attempt at playing Pac-Man. This data will be collected into a Comma Separated Values (CSV) file or multiple CSV files for later reference using Python's Pandas library. Some of the data points we intend to track include:

- Success. Did Pac-Man win?
- Lives lost. The number of lives Pac-Man lost
- Points scored. The number of points Pac-Man scored
- Pacgums collected. The number of Pacgums (the dots) Pac-Man collected.

Data Processing

The aforementioned data will later be processed into a smaller dataset of key performance metrics for each algorithm and a set of graphs meant to graphically display the performance of each algorithm and compare said performance. These actions will be completed using the Python libraries Pandas, Scipy, and Pyplot from Matplotlib. Some performance metrics we intend to collect from the data include:

- Means
- Medians
- Modes
- Ranges
- Standard Deviations
- Quartiles

And some graphs we intend to produce include:

- Histograms
- Scatter plots
- Bar charts

Component Interaction

There are three interactions between the components explained in the prior section.

Game and Algorithm

The game interacts with the algorithms. This is handled by instantiating instances of the algorithms and passing them into the `main()` function in the `pypacman.py` file. From there they are passed into the constructor for the game itself and the instance of the game is assigned to a variable inside the algorithm object, this gives the algorithm access to the game's state. The algorithm is then passed into Pac-Man's constructor and assigned as Pac-Man's algorithm for that run of the game. The algorithm's `setup()` method is then called to perform any calculations necessary before the game begins. From that point, the algorithm's `get_dir()` method is called after each move to get the direction that the algorithm would have Pac-Man face for the next move.

Game and Datasets

The game interacts with the datasets. After each run of the game, the game will spit out information about the performance of Pac-Man in during that game. This information will be compiled into a dataset in the form of a Pandas DataFrame and then appended to the end of a CSV file tracking the statistics of all algorithm runs thus far.

Datasets and Data Processor

The processing script interacts with the dataset to produce a smaller dataset of the key values enumerated in the prior section for each algorithm and produces a set of graphs depicting information about the performance of each algorithm and graphs comparing the performance of each algorithm.

Milestone Delivery

We have adhered to our proposed timeline. We have achieved the completion of the foundational coding of the game structure and have advanced into algorithm implementation slightly ahead of our anticipated schedule. Thus far we have successfully implemented a Pac-Man game that is suitable for plugging our algorithms into and we have begun familiarizing ourselves with the systems by developing simple algorithms. We have also begun work on developing BFS and A* algorithms for Pac-Man. This progress lays a foundation for the further development and refinement of our project.

Questions

Why did we choose a particular language/framework?

We chose to build our project in Python for a few reasons.

- Our group was most familiar with coding in Python.
- The Pygame library provides much of what we need for this project.
- The Python community is big and provided us with many models to base our project off of.

We were able to find a faithful Pac-Man implementation on GitHub for us to use as the base of our project. The benefits of not having to develop our own game of Pac-Man are obvious. It gives us more time to focus on implementing our algorithms and assuring us that there will be minimal debugging later in the coding process. However, using someone else's codebase came with its own set of challenges, some of the code was very obscure, and the developer did not provide adequate comments for us to immediately understand how the code worked. We have ascertained how the code functions at this point, and have made the necessary edits for us to focus solely on algorithm development at this point.

Have we met our milestones?

We have been keeping up with our proposed timeline in the initial report. We have finished working on the game part of the project and have progressed to the implementation of algorithms a little bit ahead of schedule. We have for the time being, however, shrunk our scope for the project. We will solely be focussing on developing algorithms for Pac-Man, some of our other ideas (algorithms for the ghosts, procedural generation, machine learning) are too much to also work on while we build the Pac-Man algorithms. All-in-all, we are on track and not worried about failing to meet any deadlines set for us in the future.

Have we encountered any challenges?

Thus far we have encountered two main problems:

- Learning the Pac-Man implementation that we ended up using took bit of time. Anyone who codes knows that code written by someone else without adequate documentation just looks like gibberish. To overcome this problem when integrating the Pac-Man implementation will be using we just had to take time trying to learn the code, and had to communicate with each other when we figured something out.
- Each algorithm has to be able to be passed into the `main()` function of the Pac-Man game and work from there with no more needed input. Devising the best way to do this took some thought. What we arrived at was standardizing the format of each algorithm. Each algorithm is a Python file containing the algorithm within a Python class that contains a variable to store the game for referencing the game state, a `setup()` method for running any necessary pre-game calculations, and a `get_dir()` method for returning the direction Pac-Man should face at that moment. Because each algorithm works the same the game can interact with them all in the same way.

Do we need to change our milestones?

At this time we do not need we need to make adjustments to our milestones.

Typeset with LaTeX.