CSCI 4511W Final Project

Carston Hernke
hernk009@umn.edu

December 19, 2018

1 Abstract

Ms. Pacman is a maze arcade video game that involves guiding a character to collect the most points while avoiding enemies. Ms. Pacman has been used frequently in the field of artificial intelligence to evaluate the effectiveness of new algorithms and approaches. This research will focus on developing an agent that will play the game effectively. Related work is discussed to provide a theoretical overview. An approach to solving the problem by discretizing the environment into nodes with costs is described. The performance of four different search algorithms are then compared for finding the optimal path for Ms. Pacman. These results and potential explanations are then discussed. Finally, future directions for research are presented.

2 Introduction

The goal of this project is to develop an agent that plays the classic Atari game Ms. Pacman (Pacman) competitively. Pacman is a complicated game and there are a few different types of problems to be solved at once. The environment consists of a map on a 2D plane consisting of individual points where food particles are located. The goal is to control the movements of Ms. Pacman around the environment to maximize the number of food particles collected. At the same time, there are 4 ghosts who are chasing Ms. Pacman and must be avoided. The fundamental problem is pathfinding, finding the route that Ms. Pacman should take to get food particles and avoid ghosts. However, since the game is dynamic the agent cannot spend unlimited amounts of time searching for the optimal path. There are other sub-problems, such as predicting the movement of the ghosts (each of whom has a unique strategy) and optimizing when to use the power-ups in the corners. There is has been extensive research conducted to solve these problems and the first section of this paper will focus on briefly summarizing the varied approaches that other researchers have taken in the past. The approach taken for this project will then be described, along with the results and analysis of the experiments conducted.

3 Related work

One of the earliest surveys of the field of Artificial Intelligence was written in 1974 by Nils Nilson [10]. In the paper, Nilson divides the field of Artificial Intelligence into four types: (1) Techniques for modeling knowledge, (2) Techniques for reasoning and problem solving, (3) Techniques for heuristic search, and (4) AI systems and languages. A sublevel of this framework is game-playing problems. When this paper was written in 1974, Nilson described cutting-edge chess algorithms as "still using rather straightforward tree-searching ideas" and also wrote that algorithms "must become substantially better before they can beat humans." [10].

In 1997, Chiang et al. [4] examined the use of different algorithms for conflict detection and resolution, specifically in the context of Air Traffic Management. This is related to the other papers examined, however, the other papers took a much simpler approach to conflict detection - in the context of the Pacman problem the previous 2 approaches boil down to a boolean 'conflict/ no conflict' test at a single point. Neither of the other papers attempt to model or predict where the ghosts will be at a given point in time and whether this predicted point will put them in conflict with a predicted path. This paper describes the use of 2 strategies with their basis in geometric computation. The first involves the use of Delauney Diagrams at each state to keep track of how close planes are to each other and whether the nearest-neighbor distance is less than the allowable limit. The second approach is called geometric hashing and involves discretizing the airspace into squares of the allowable size. Then, to check for conflicts with a given plane at a given time, only neighbor squares need to be examined. Both of these approaches have the benefit of being extremely efficient at finding potential conflicts. This article provides a description and solution to conflict detection, while the other two papers focused exclusively on pathfinding.

In a paper published in 2011, Liu and Ramakrishnan [9] discuss a novel algorithm for pathfinding under constraints called A*Prune. A*Prune. The problem that they are specifically solving is K Multiple-Constrained Shortest-Path (KMCSP), where the goal is to find multiple shortest paths given certain constraints. This algorithm is unique because after every iteration it checks to see if the partial paths have any chance of fulfilling the constraints. If a partial path will not fulfill the constraints, it is pruned and not searched any further. The benefits of the A*Prune algorithm are that it is optimal and complete, as well as able to solve KMSCP. However, one of the main drawbacks of the base A*Prune algorithm it runs in exponential time, not polynomial time. Thus it may not be well suited for dynamic problems like Ms. Pacman. However, there is some discussion in the article about the use of a Bounded-A*Prune (*BA*Prune*) algorithm that can give approximate solutions and may be relevant to the Ms. Pacman Problem.

Ali, Munir, and Farooqi [2] examine the Pacman game specifically. In contrast to Liu and Ramakrishnan [9], this article focuses on simpler techniques for solving the Ms. Pacman game. The paper proposes 4 different strategies, named *random, greedy, rule-based with greed, and rule-based with look-ahead.* The random policy picks a purely random move. The greedy policy picks whatever direction has the highest number of food pellets available, with no concern for the location of ghosts. The other two rule-based policies are a bit more advanced. The first one makes sure that the chosen path has the most pallets and no ghosts. The second looks ahead to the second move and makes sure that it has pallets and that there are no ghosts. After testing each of the strategies, the authors concluded that the rule-based strategies were most effective.

These approaches are similar to those examined by Thomas et al. in 2008 [13]. In that paper, the researchers compared a random strategy, a greedy-random strategy that took the path with the largest number of pills, and a greedy-lookahead strategy that searched for the shortest path to the closest pills. Thomas et al. found that the greedy look-ahead strategy generated the highest average score [13] These strategies are a bit different from those described in the A*Prune algorithm [9].

These strategies are a bit different from those described in the A*Prune algorithm [9]. The A*Prune algorithm by default performs BFS without any limits on depth, so it will explore much more space (and take much more time) than the rule-based strategies described in this article. However, this increased search has the benefit of finding problems that are 'over the horizon' of rule-based search.

Alhejali and Lucas take a different approach to the problem, employing genetic algorithms to evolve Pacman agents who react the best to different situations. The genetic algorithm was allowed to run over 50 iterations and the resulting most successful agent scored an average of over 15,000 points per game [1]. The researchers also found that this genetic algorithm approach was able to be generalized to the 4 different maps available in Ms. Pacman.

Another approach to determining the best move in the Ms. Pacman game is to employ Monte-Carlo Tree Search (MCTS), which is a tree-based algorithm that involves using simulations to predict the reward of certain paths. Ikehata and Ito explored this approach in their 2011 research [8]. The researchers used a MCTS algorithm to predict the probability of being trapped by 'pincer' moves, where Ms. Pacman is surrounded by ghosts and cannot escape. The research found that the MCTS algorithm produced a significant improvement in performance when compared to existing methods [8]. Pepels et al. also applied MCTS to the Ms. Pacman game, but rather than using it to predict pincer moves specifically, the researchers used the MCTS algorithm to evaluate and choose all of Ms. Pacman's moves in real-time [11]. Because of the real-time nature of the game and the limited search time, the researchers used a novel strategy of re-using information between moves and decaying this information over time by a set parameter. This allowed the algorithm to save time by not having to re-examine the entire tree for every move. The researchers found that the combination of MCTS and re-using information resulted in a powerful agent that places highly in Ms. Pacman rankings [11].

Foderaro et al. also examine the problem of creating an agent that can play Ms. Pacman [7] in their 2017 research. Their research is differentiated by the approach that they took to modeling the problem. Foderaro et al. used a computational geometry approach known as cell decomposition to discretize the environment into cells. The relationship between these cells was then represented using a connectivity tree/graph [7]. This graph was then used to generate the possible paths that Ms. Pacman could take.

4 Approach

The basic approach will be to use breadth-first search to explore a tree of the possible moves for Ms. Pacman, with the primary goal of avoiding ghosts and the secondary goal of accumulating as many dots as possible.

4.1 Actuators

Moves available to Ms. Pacman include *right, left, up, down.* Depending on the state of the game, not all moves will be available. A map of the state space was created, which identifies which spaces are allowed to be moved to (Figure 1). A function was developed that takes a position as input, uses the map to identify available moves, and then returns the possible moves. Actions are then transmitted to a Stella Atari 2600 Emulator [3] by calling a Python script from the Open AI Gym library [5].

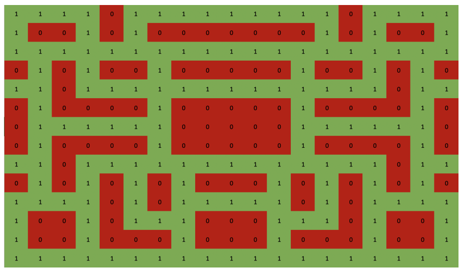

Figure 1: A map of the state space, indicating which spaces are valid

4.2 Environment

Ms. Pacman takes place on a fully-observable map, as seen in Figure 2 The output of the game is a an RGB array with shape (210, 160, 3). This environment was discretized into a 19x14 grid of XY coordinates. This 16x21 grid captures every dot in a unique square and allows for simplified tracking of Pacman and the Ghosts.

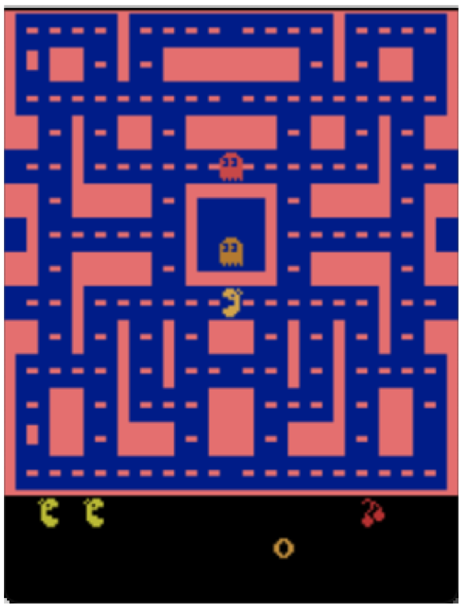
Figure 2: Rendered output of the Stella Atari 2600 Emulator

4.3 Sensors

The sensor is a RGB image of the [virtual] screen, provided by the Stella Atari 2600 Emulator. This image is provided as an RGB array with shape (210, 160, 3). The only other information provided by the emulator is an integer value of the current score of the game and an integer value of the number of lives remaining. No information on the location of Pacman or the Ghosts is provided by the emulator. These locations must be derived from the raw output.

4.3.1 Locating Pacman and the Ghosts

The main character in Ms. Pacman is Ms. Pacman herself, who is being chased by 4 different ghosts (Figure Because no location information for either Pacman or the Ghosts is provided by the emulator, this information must be derived from the raw output. This is done by exploiting the fact that each ghost, as well as Pacman, is a unique color (Table 1). These colors only appear in the game on these characters. To determine the location of each character, the RGB array is searched for each color and an array of locations with this color is returned. A centroid function is then applied

	R	G	B
Ms. Pacman	210	164	74
Ghost 1	198	89	179
Ghost 2	84	184	153
Ghost 3	180	122	48
Ghost 4	200	72	72

Table 1: RGB color values for Ms. Pacman and the 4 ghosts

to this array points with color X. The result of this centroid function is then discretized to the 19x14 grid (see Environment). Although a rather unconventional approach, this yields highly accurate locations for Pacman and the Ghosts and is fast to compute.

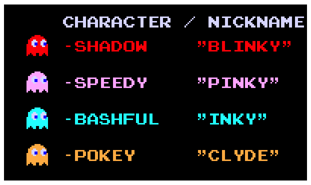

Figure 3: The 4 different ghosts in Pacman. Taken from *https://en.wikipedia.org/wiki/Ghosts_(Pac-Man)*

4.4 Performance

The ultimate goal of the Ms. Pacman game is to get the highest score. A variety of actions contributes to increasing the score, including eating dots, eating big dots, eating ghosts when they are on the defensive, and eating the candy that randomly appears in the middle of the screen. For this project, performance was defined by the amount of time that Pacman survived, rather than the score that the game returned. I chose to use this measure of performance because the strategy that I developed seeks to avoid ghosts rather than attempting to collect every dot.

4.4.1 Cost Function

When searching graphs for an optimal path, it is necessary to define the costs of traveling to each node. In Pacman there is no specific cost for traveling from one space to another, however, to discourage unneeded back-and-forth I set the cost of travelling from one space to another to be 1. A non-zero value is necessary to find a realistic solution. Being eaten by a ghost causes Pacman to lose a turn, so I set the cost of moving to a space that is occupied by a ghost to be 100.

4.5 Gameplay

At each step in the game, the following process is used:

1. The RGB matrix representing the current game state is retrieved.
2. Based on the RGB matrix, the locations of Pacman and each of the Ghosts is identified.
3. The location of the ghosts is used to update a graph containing the costs of traveling to each node (See Cost Function).
4. The closest uneaten super dot is identified as the goal.
5. A search algorithm is used to find the optimal route between Pacman's current location and the super dot, avoiding ghosts along the way (Figure 4)
6. The route returned by Uniform Cost Search is examined to find the necessary next move for Pacman.
7. The move is executed, and the process repeats until there are no more dots on the level or Pacman is eaten by a ghost.

Path from (4,6) to (1,1): ***Node 4,6 - Node 4,5 - Node 4,4 - Node 3,4 - Node 2,4 - Node 1,4 - Node 1,3 - Node 1,2 - Node 1,1***

Figure 4: An example of a path generated for Pacman by the search algorithm.

5 Experiment Design and Results

The steps described in the Gameplay section were implemented in a script written in Python. The program was designed to leverage the Undirected-Graph and GraphProblem data structures contained in the *AIMA Python* library [6]. The AIMA Python library contains implementations of the algorithms described in Artificial Intelligence: A Modern Approach [12].

	Average elapsed time
Breadth First Graph Search	1.39ms
Depth First Graph Search	0.50ms
Uniform Cost Search	2.46ms
Bidirectional Search	0.10ms

Table 2: A comparison of the time performance of various algorithms in finding the optimal path to a corner. Results are the average of 154 iterations.

By writing the Pacman-related code to interface with the data structures and algorithms in the AIMA Python library, the algorithms do not need to be implemented again, which reduces the risk of errors. It also greatly reduces the amount of time needed to write the code for the algorithms, making it relatively easy to test different algorithms. To begin, four different algorithms were used to find the optimal route between Pacman's current location and the closest super dot. The performance of the each of the algorithms, measured in milliseconds, is show in Figure 2. Performance was ranked in terms of speed because the nature of the Ms. Pacman game requires that decisions be made quickly.

6 Analysis

Very surprisingly, each of the algorithms found the optimal path to a corner in under 3ms. The game of Ms. Pacman requires moves to be made within 40ms [11]. Each of these algorithms would have easily been under this cap even when accounting for significant overhead (finding the characters and building the search graph). The very fast search speed for each of the algorithms can likely be explained by the relatively small graph size consisting of 161 nodes and an average branching factor of 3. The graph is small because of the fixed node goal, rather than attempting to find the most optimal destination, a far more open-ended problem. This particularly explains the extremely fast performance of bidirectional search. The original idea for this project was to implement Monte Carlo Tree Search (MCTS) in order to find the best path within the 40ms timeframe. This plan was based on the idea that Ms. Pacman requires decisions to be made extremely quickly and it seemed unrealistic for algorithms such as Breath First Graph Search to find a solution within that short timeframe. Thus, it was hypothesized that using MCTS and simulating playouts rather than actually fully searching the graph would give significant performance benefits. However, after applying the relatively basic search algorithms to the problem it is clear that their performance is suitable for the problem. Each of the 4 algorithms was far under the 40ms decision-time cap. Because these algorithms show fast performance and are guaranteed to find optimal solutions, it is not necessary to employ MCTS for this problem. Monte-Carlo Tree Search relies on simulating playouts to predict the performance of certain branches. It is well suited for problems with a high branching rate and limited time. However, because MCTS relies on simulations, it does not guarantee finding the optimal solution.

7 Future Work

There are a number of future directions that can be taken to build and improve on this project. While this project took into account only the current location of the ghosts when planning an optimal route, a future improvement would be to predict the movements of the ghosts into the future. Each ghost has its own playing style, so they would each need to be modeled individually. With a prediction of where each ghost will be at an arbitrary time in the future, Pacman could adopt more nuanced strategies. Right now the current algorithm always avoids any path that crosses a ghost, even if the ghost is many tiles away and poses no immediate threat. Another improvement that could be made to this project would be keeping track of which dots have been consumed. Currently the algorithm decides on a path to the closest corner 'super dot' and eats dots randomly along the way. There is no strategy for eating all of the dots in the most efficient way, but it is assumed that this will happen eventually after enough moves. An improvement would be to keep a map with information on the locations of dots and use this information to discount the path cost to incentivize the selection of paths that would collect the most dots. Another future improvement would be to utilize the teleport-tunnels on each side of the map. In this project there are no available moves mapped to those spaces. However, in the future the graph of the map could be modified so that those teleport-tunnels connect to each other and have a low path cost. This would introduce further complexity, however. Finally, Monte-Carlo Tree Search could be implemented, but the search space would need to be expanded further to leverage the power of the algorithm to estimate many millions of possible paths. Currently the graph of possible moves is small enough that basic graph search methods are effective.

8 Conclusion

Research into pathfinding and adversarial game-playing has a relatively long history for computer science and is still ongoing today. Approaches have improved drastically from when Nils Nilson wrote his survey paper on the subject in 1974. Ms. Pacman specifically has emerged as a very popular problem for advancing research on the subject. There are dozens of research paper comparing different approaches to solving the problems posed by the game. Undertaking this project gave me a much better understanding of the inherent complexity and messiness of applying fundamental artificial intelligence concepts to a non-trivial problem. In the future, it will be very interesting to see how advances in artificial intelligence continue to improve the way that classic games like Ms. Pacman are played.

References

[1] A. M. Alhejali and S. M. Lucas. Evolving diverse Ms. Pac-Man playing agents using genetic programming. *2010 UK Workshop on Computational Intelligence, UKCI 2010*, (October 2010), 2010.

[2] R. B. Ali, M. Ali, and A. H. Farooqi. Analysis of rule based look-ahead strategy using Pacman testbed. *Proceedings - 2011 IEEE International Conference on Computer Science and Automation Engineering, CSAE 2011*, 3(May):480–483, 2011.

[3] S. A. Bradford Mott. Stella. urlhttps://stella-emu.github.io/, 2018.

[4] Y.-j. Chiang, J. T. Klosowski, and J. S. B. Mitchell. Geometric algorithms for conflict detection/resolution in air traffic management. *Decision and Control*, pages 1–6, 1997.

[5] O. Contributors. Mspacman-v0. urlhttps://gym.openai.com/envs/MsPacman-v0/, 2018.

[6] V. Contributors. Aima python. urlhttps://github.com/aimacode/aima-python, 2013.

[7] G. Foderaro, A. Swingler, and S. Ferrari. A model-based approach to optimizing Ms. Pac-Man game strategies in real time. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):153–165, 2017.

[8] N. Ikehata and T. Ito. Monte-Carlo Tree Search in Ms. Pac-Man. *011 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 39–46, 2011.

[9] G. Liu and K. G. Ramakrishnan. A * Prune : An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints. *Ieee Infocom*, pages 743–749, 2001.

[10] N. J. Nilson. A Survey of Artificial Intelligence. *Information Processing*, 74:778–801, 1974.

[11] T. Pepels, M. H. Winands, and M. Lanctot. Real-time monte carlo tree search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):245–257, 2014.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[13] T. Thomas, M. Lewis, L. John, and A. Alastair. An evaluation of the benefits of look-ahead in pac-man. *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, pages 310–315, 2008.