American University of Armenia, CSE
Artificial Intelligence, Fall 2022

CS 246/346
Final Project Report

Students: Anahit Baghdasaryan, Mane Davtyan

Instructor: Monica Stepanyan

Introduction

This paper takes as an area of research the original game and a series of modifications of Pac-Man as a problem-solving environment to deep dive into the state-space search, and other algorithms, covered during our course CS246, "Introduction to Artificial Intelligence." By doing so, we aim to define our problem as the enhancement of Pac-Man's behavior.
There are a couple of reasons behind our decision to concentrate on Pac-Man. Firstly, it is fun to play and watch, as Pac-Man provides a point in history where video games moved into new territory(Thompson et al., 2008) and perfectly represents retro pop culture. Secondly, this environment supports deterministic/stochastic, fully/partially observable, and adversarial problem settings. Finally, this is an environment that is both intuitive and rich (DeNero & Klein). Pac-Man is intuitive because it consists of objects moving around on a grid; this setting is easy to map onto the general definitions of search problems. Pac-Man is rich because it gives rise to very challenging AI problems.
The main goal of this project is to examine and test methods for the enhancement of Pac-Man's behavior. First, we will represent Pac-Man as a single-agent search problem with several search algorithms and compare the performance of Pac-Man in terms of the algorithm's performance, completeness, and optimality. Second, we will discuss more complicated modifications of Pac-Man (i.e., including nondeterministic ghosts, etc.) and methods that have been used to solve and improve them. Therefore, the paper will consist of two parts. The first part will discuss the original Pac-Man, its modifications, and methods that have been used to solve and improve them, and the second part will concentrate on the methods and solutions for Pac-Man as a single-agent search problem.

Literature Review

Pac-Man was developed by Toru Iwatani and was released by Namco as a coin-operated arcade game in 1980. The game was initially called Puckman but was renamed and redistributed in America by 1981. It soon became the most popular arcade game of all time, even causing a coin shortage in Japan.

Pac-Man

Based on the **"***Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game*" paper, you can see the game rules and description below. (Rohlfshagen, Perez-Liebana, n.d.)

- The game is played on a 2-dimensional maze.
- The player uses the four-way joystick to guide Pac-Man through the maze.
- 240 (non-flashing) pills worth 10 points, 4 (flashing) power pills worth 50 points along the maze.
- The lair in the middle of the maze, where the four ghosts begin, is where they are all free one at a time.
- As each ghost chases Pac-Man, they eat him at first contact.
- Initially, Pac-Man has two extra lives; the game ends when all lives are gone.
- At 10,000 points, a new life is awarded.
- When Pac-Man consumes a power pill, the ghosts temporarily turn blue and become edible for a short while.
- After eating four ghosts in a row, Pac-Man receives a score that doubles: 200, 400, 800, and 1,600.
- 3 050 points can be obtained per power pill eaten (3000 points for eating all the ghosts + 50 points for the power pill)
- Bonus fruits occasionally appearing right below the lair are the final way to get points.
- Eight different fruits range from 100 to 5,000 in value (higher-valued fruits appear in later levels only).
- Pac-Man is the player of the game
- Ghosts are the enemies of the general hero
    - There are four main ghosts in the game; red Blinky, pink Pinky, blue Inky, and orange Clyde
    - In the original game, ghosts have deterministic behavior intended to consume the Pac-Man
    - Depending on the game level, they can operate in three modes; scatter, chase, and frightened.
    - Speed is constant unless they travel through a tunnel or are eaten by Pac-Man.

Such as, the actions of ghosts are predetermined, and it is impossible to get a perfect score. The ghosts differ in how aggressive they are; Clyde, for example, is the most diminutive hazardous ghost and only pursues Pac-Man 20% of the time.
Every ghost has a central area where it mostly moves and spends the majority of its time. Ghosts can move in one of three directions—scatter, chase, or frightened—and a reversal event happens after several transitions between these modes, for example, when moving from scatter mode to chase mode or vice versa. Pac-Man is usually not allowed to reverse, but the situation when

Pac-Man is permitted to reverse is called a reversal event. Unless Pac-Man takes a power pill or an all-ghost-reversal event occurs, ghosts cannot reverse direction.
The ghosts generally move at a constant speed, except when they pass through the tunnel, where they dramatically slow down, and when they encounter edible objects (when they travel at half speed). Additionally, ghosts may quicken as the player approaches the finish of a level (while the speed depends on the type of ghost). These differences in Pac-Man and the ghosts' speeds significantly develop the gameplay and the difference between life and death. For instance, an expert player with ghosts in hot pursuit can look for pill-free paths and rotate several times while taking advantage of the corners.

Ms. Pac-Man

The Pac-Man strategy manuals provide exact patterns the player can follow to increase the game score. These patterns are not only vital in defeating Pac-Man, but they are also a weakness of the game, as noted by Mott in the article "1001 Video Games You Must Play Before You Die": "Lacking any particularly inspiring AI, Pac-Man's pursuers race around the maze, following predictable paths, meaning players can effectively beat the game through memory and timing rather than inventive reactions." (Mott, 2010) Generally, the deterministic nature of the ghosts suggests that there is an exact approach to playing the most effective game. The addition of ghosts changed the situation with nondeterministic behavior in Ms. Pac-Man, a game that featured a female version of the yellow circular hero that was published in 1982: "It is important to note that the monsters are much more intelligent in Ms. Pac-Man than in the original Pac-Man game. This means you cannot always predict what the monsters will do." (Mott, 2010) Ms. Pac-overall Man's gameplay is remarkably similar to Pac- Man's, and the game's goal is still the same.
Nevertheless, despite the ghosts' altered behavior, Ms. Pac-Man also arrived with four new mazes played in rotation.
Furthermore, Clyde was renamed Sue, bonus fruits moved randomly through the maze, and the edible period of ghosts decreased as the game progressed but increased frequently, meaning that Pac-Man can eat the ghosts periodically. However, as the game goes on, this feature starts to fade. See the generalized alterations in the Ms. Pac-Man problem:

- The sizes of the maze are altered a little
- Four new mazes were added to the structure
- A female character was added
- Clyde was renamed Sue
- Ghosts' behavior became nondeterministic
- Moving fruits are added throughout the maze
- Ghosts become edible periodically during the game

Pac-Man might be considered trivial compared to modern-day video games, but the challenge it gives rise to is still relevant to nowadays' humans and machines. In this day and age, enthusiasts still attempt to reach and break the world Pac-Man record; meanwhile, academics are rigorous in developing agents capable of playing the game. (Thompson, McMillan, Levine, Andrew, 2008)

Pac-Man Controllers

The authors of the work "*Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game*" mention that most of the work completed in computational intelligence has been committed to finding better controllers for (Ms) Pac-Man. (Rohlfshagen, Perez-Liebana, n.d.) This includes many methods, such as rule-based ones, tree search or learning, and several nature-inspired (i.e., genetic) algorithms.

1. *Rule-Based*
   Rule-Based Pac-Man controllers include a set of if-then clauses as the main component, which are presented in either a hard-coded way or in a more structured manner. Although these controllers are effective, they require an essential amount of domain knowledge in order to implement the rules, transitions, conditions, and actions. (Rohlfshagen, Perez-Liebana, n.d.)

2. *Tree Search & Monte Carlo*
   Tree search techniques and Monte Carlo simulations, especially MCTS, have shown excellent performance for the agent; however, it needs to have a simulator in order to have a forward model. The forward model is a computational model of spontaneous impulse control that emphasizes the significant role played by comparisons between the purposeful content of our actions and their results.

3. *Evolutionary Algorithms*
   Evolutionary Algorithms are mainly used for creating Pac-Man agents. They include Genetic Programming (evolving tree structures), Grammatical Evolution, and some hybridizations of these methods. The results of the methods are impressive; however, they still require an essential amount of domain knowledge without the dependency on a forward model. This is an important factor that distinguishes them from tree-based search methods. (Rohlfshagen, Perez-Liebana, n.d.)

4. *Artificial Neural Networks*
   Artificial Neural Networks (ANN) were used as a Pac-Man controller implementation tool many times. ANNs are computational abstractions of the human brain and work as universal function approximators. (Rohlfshagen, Perez-Liebana, n.d.) These can be

configured as policy networks that choose actions directly given the current game state or as value networks that rate the value of each state after taking each possible action.

5. *Neuro-Evolutionary Approaches*
   This particular approach is a trendy approach to controller development. It hybridizes evolutionary algorithms and neural networks. Evolution is used to improve the weights, topologies, and reward functions of artificial neural networks, and literature shows that it can be further hybridized with other techniques. (Rohlfshagen, Perez-Liebana, n.d.)

6. *Reinforcement Learning*
   Reinforcement Learning describes another set of offline learning methods that require episodic learning. This approach works pretty well on Ms. Pac-Man because of the stochastic nature and unpredictable behavior of ghosts there. The approaches described here mainly used a form of Q-Learning or TDL. Most of the work performed with RL methods utilizes different frameworks to the competitions, making general comparisons with other methods less visible. One of the difficulties of these methods is the large state space and number of features which occur in Ms. Pac-Man, making the method work for this specific modification. (Rohlfshagen, Perez-Liebana, n.d.)

Work accomplished in 2003 by Gallagher and Ryan in "Learning to play Pac-Man: an evolutionary, rule-based approach" includes hand-coded rules for gameplay in a reduced version of the original Pac-Man (Gallagher, 2003)

- No power pills
- One deterministic ghost
- Specific parameters of rules trained by an evolutionary algorithm

The learning process was triumphant, with the authors deliberating the benefits and drawbacks of the approach. "One interesting observation is that the agent can consider only localized information, an important factor in our observations of Pac-Man work to date." (Gallagher, 2003)

The work performed by Fitzgerald and Congdon in "RAMP: a Rule-based Agent for Ms. Pac-Man" use a rule-based controller, RAMP (Fitzgerald, Congdon, 2009). The decision to choose a certain path is made based on higher-level conditions and actions. The set of conditions contains the following information

- the number of ghosts that are "close" or "very close" to Ms. Pac-Man
- the number of remaining power pills, and whether ghosts are edible

The conditions also differentiate between the mazes and level and the progression throughout the level to apply the most fitting set of rules. The set of actions is composed of Graze, Eat, Evade and Run, and multiple parameters are used to construct the complete set of rules. Essentially, the conflict is solved when multiple rules are applied simultaneously.

In order to develop the performance of agents, training algorithms were applied using machine learning practices as well. Many scholars agree with the machine learning approaches and claim that such practices are better applied to a small scope of problems because only the strategy of avoiding ghosts has to be depicted. (Thompson et al., 2008). The paper "An Evaluation of the Benefits of Look-Ahead in Pac-Man" discusses the hypothesis that planning ahead and creating paths in the maze while using a reactive control to escape the clutches of the ghosts is beneficial and can be applied to generate more intelligent agents than the original Pac-Man. (Thompson et al., 2008) The authors have used Pac-Man as a base but applied some critical differences, like stochastic ghosts. The controller suggested is constructed from a knowledge base and a graph model of 7 the maze. The knowledge base is a series of rules, and the overall decision-making is enabled by the use of a Finite State Machine (FSM). The FSM has three states: Normal, Ghost is Close and Energised. Three different strategies are considered: Random, Greedy-Random, and Greedy-Lookahead. The first two just make a move based on the current state of the maze. The third one performs a search in certain situations and subsequently employs A_Star Search to find the paths to the targets identified during the search. Ghost avoidance is solved explicitly in the FSM. The experiments compared all three controllers as well as a range of human players and found that the lookahead player, although not quite as good as the best human player, significantly outperforms the other two controllers. (Thompson et al., 2008)

Koza has improved one of the earliest works to improve Pac-Man's performance and assess the effectiveness of genetic programming for task prioritization. This work operated on a modified version of the Pac-Man domain, with a maze that replicated the first level of Ms. Pac-Man. (Thompson et al., 2008) The ghost behaviors were not the same as in the original game. All four ghosts use the same strategy. Using a series of predefined control primitives for perception and action control, agents can generate solutions to solve certain mazes at the human novice level, such as structure-dependent solutions (Thompson et al., 2008).

Kalyanpur's and Simon's "Pacman using Genetic Algorithms and Neural Networks" paper applies a genetic algorithm to improve ghost strategies. The article took a domain similar to the original Pac-Man domain, while the solutions obtained were in the form of a list of directions for ghosts to traverse. To find proper crossovers and mutation rates, neural networks were used based on experimental data from the game world. (Kalyanpur, Simon, 2001)

The project of 2007 written by Gallagher and Ledwich in "Evolving Pac-Man Players: Can We Learn from Raw Input?" attempts a lower-level approach; it uses the same reduced domain, and the agent applies neuroevolution. (Gallagher, Ledwich, 2007) The agent succeeds in gaining basic skills, such as it has very constrained information on the game rules and the game state and a reasonably straightforward fitness function. (Gallagher, Ledwich, 2007)

Method

The Pac-Man agent will find paths through his maze world, both to reach a particular location and to collect non-flashing pills efficiently. We will build general search algorithms such as BFS, DFS, UCS, A* and apply them to Pacman scenarios. We will compare the performance of Pac-Man in terms of the algorithm's performance, completeness, and optimality.
Considering the fact that such experiments were done before, we will also compare our results with the prior ones.
In order to develop an environment that includes mazes of different sizes, non-flashing pills, and our moving agent Pac Man we will use Python 2.7 programming language. Code sources for creating the environment are available online, provided by the University of California, Berkeley, in the scope of the course CS188, Introduction to Artificial intelligence; hence we are planning to use them as well as to perform modifications in order to get different mazes. After we implement the search algorithms, we will use those on the developed environment with the modified mazes and compare the eventual results with the initial ones.

About the Structure of Files (Provided by UC, Berkeley)

The code for this project consists of several Python files. (Joshkarlin, n.d.)

Files to edit:
search.py, Where all of the search algorithms will reside.
searchAgents.py, Where all search-based agents will reside.

Files you might want to look at:
pacman.py The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.
game.py The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
util.py Useful data structures for implementing search algorithms.

Supporting files:

graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents

Evaluation

To find the optimal path through the maze we have considered solving the problem with the help of a problem-solving agent, to let the agent eat all the dots in fewer steps possible. In designing this intelligent agent, we implemented various uninformed search algorithms covered during the semester. (DFS, BFS, UCS). Although these algorithms can solve the problems, they are not that efficient; thus, we have applied informed search as well. (A* search with heuristics Manhattan distance and Euclidian distance)

We have run all the mentioned searches and recorded the number of expanded nodes, scores, and total cost while defining our main agent to be the Pac-Man. We have also included several pills (goals) throughout the maze, thus doing the Pac-Man search for them. For this certain reason, we have added new classes for each of the search algorithms to collect all the pills in the maze. In our code, the A* search was run by two heuristic functions, with Manhattan and Euclidian distances. Besides, we have generated new mazes of three different sizes, tiny, medium, and big, and run the searches on both the original and newly generated mazes. The mazes were generated by the online tool cited below. (dCode, n.d.)

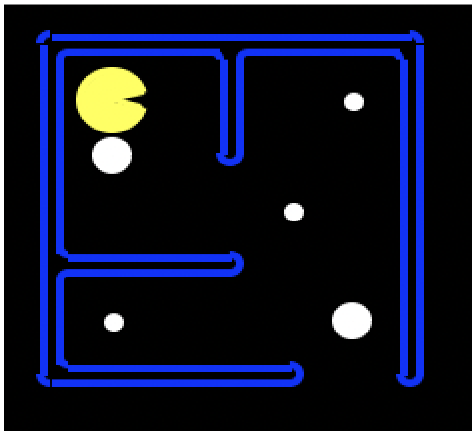

Figure 1.1.

Original tinyMaze

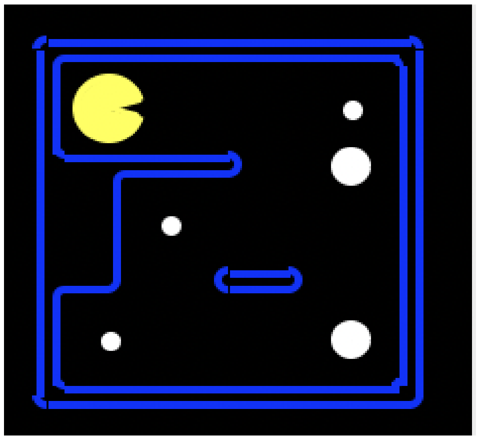

Figure 1.2

Modified tinyMaze

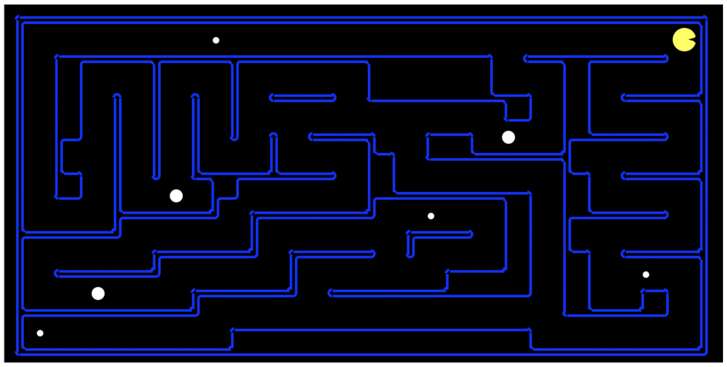

Figure 2.1.

Original mediumMaze

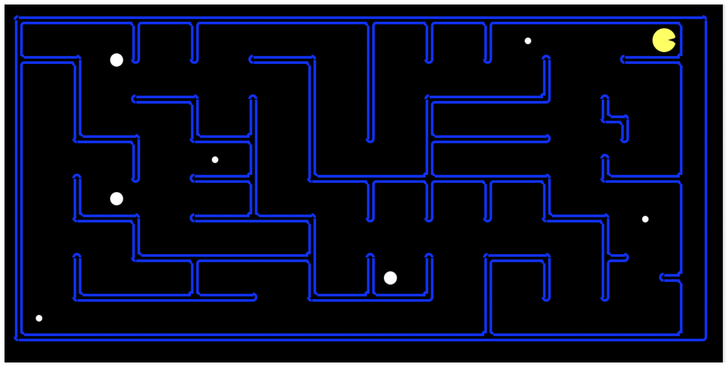

Figure 2.2

Modified mediumMaze

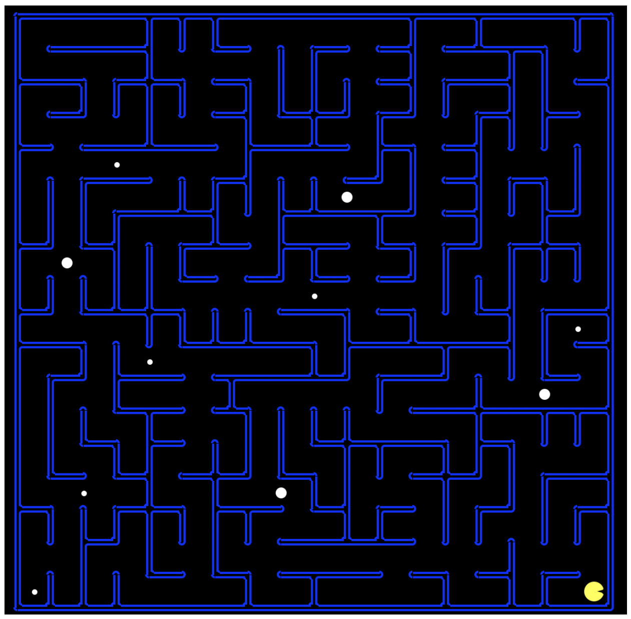

Figure 3.1.

Original bigMaze

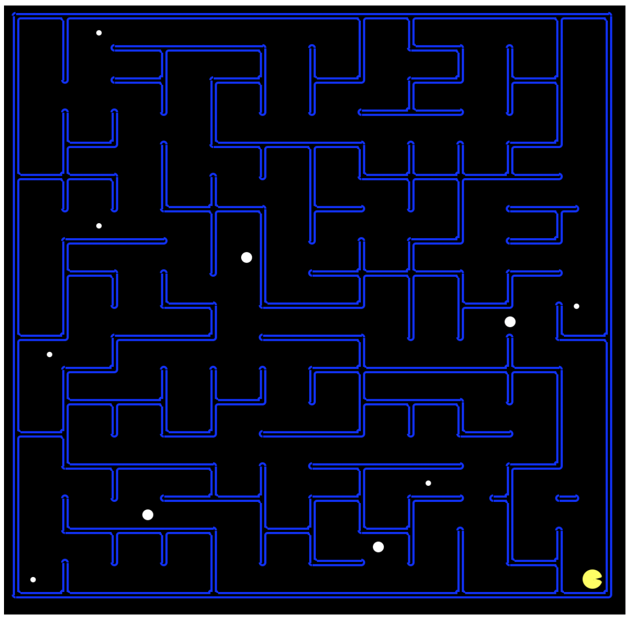

Figure 3.2

Modified bigMaze

Depth First Search

DFS starts at the root node and through depth along each branch before backtracking. Depth First Search works with LIFO, last in first out queues, expanding the most recent node generated. In Table 1.1 and Table 1.2, you can see the results of DFS completed on the original maze of the code source and the maze generated by our team members. As for tiny mazes in the original version, the solution costs less than the modified version, fewer nodes are expanded, and the total score is a bit higher. In the medium-sized mazes, the solution of the modified version costs less, fewer nodes are expanded, and the total score is nearly greater by 100 than in the original version. Finally, for the big mazes, the solution found for the original maze costs much less, the nodes generated are less by nearly 500, and the total score is almost 10 times greater than in the modified version.

Original:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	20	23	510.0
mediumMaze	340	581	200.0
bigMaze	242	1045	318.0

Table 1.1

Modified:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	26	41	504.0
mediumMaze	228	357	312.0
bigMaze	528	1583	32.0

Table 1.2

Breadth First Search

Unlike the DFS, BFS expands the nodes on the breadth of the path. Keeping the nodes in the FIFO queue, breadth-first search expands the first node of the queue. Tables 2.1 and 2.2 show all results depicted from BFS on the Pac-Man with several pills on the mazes. In the tiny mazes, the solution obtained for the original version has a lower total cost than in the modified version, but the number of expanded nodes and the total score is a little higher than in the original version. For the Medium mazes, the solution cost and the total score are higher, while the number of expanded nodes is less in the modified version compared to the original. As for the big mazes, the original maze has to exceed the total cost in comparison to the modified one, while the score is higher in the modified one, and the expanded nodes are way bigger than the original.

Original:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	12	58	518.0
mediumMaze	14	2777	396.0
bigMaze	242	2891	318.0

Table 2.1

Modified:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	16	54	514.0
mediumMaze	87	2255	453.0
bigMaze	216	26039	344.0

Table 2.2

Uniform Cost Search

This specific search expands the nodes with the lowest path cost. It stores the nodes in the priority queue and then pops out the least valued node from the queue. Tables 3.1 and 3.2 represent the results of this search algorithm for the original and the modified mazes. We see that for the tiny mazes, the solution cost for the original maze is less, the number of nodes expanded is greater, and the total score is a bit higher than for the modified maze. In the case of medium-sized mazes, the solution for the original maze has a higher total cost, the number of nodes expanded is greater, and the score is less than in the modified maze. Finally, for the big mazes, the solution obtained for the original maze has a higher cost, however, in the modified version number of nodes expanded is almost ten times greater, and the total score is a little higher.

Original:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	12	58	518.0
mediumMaze	144	2777	396.0
bigMaze	242	2891	318.0

Table 3.1

Modified:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	16	54	514.0
mediumMaze	87	2255	453.0
bigMaze	216	26039	344.0

Table 3.2

A* search

A* search expands the nodes with the least value of evaluation function, which is equal to the sum of path cost and heuristic function.

$$f(x) = g(x) + h(x)$$

For this specific informed search, we have used two types of heuristics, Manhattan and Euclidian distances. We ran the program on all original and modified mazes with both of the mentioned heuristics used.

Tables 4.1, 4.2, 5.1, and 5.2 represent the results for the manhattan and euclidian heuristics used with A* search.

In the case of the Manhattan distance heuristic, the solution cost for the tiny mazes is smaller for the original maze, as well as the number of expanded nodes, while the total score is slightly higher in the original. Medium mazes' cost of the solution is smaller in the modified version, as well as the number of expanded nodes, while, again, the score is greater than in the original version. Furthermore, for the big mazes, the solution cost of the original one exceeds the modified one, while the total score is smaller in the original, and the number of nodes is significantly smaller than in the modified version.

For the Euclidian distance heuristic, we got the same results as for the Manhattan distance heuristic. Turns out that in this specific game with those specific mazes generated, it does not matter if we use Manhattan or the Euclidian distances as our heuristics.

Manhattan Distance heuristic

Original:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	12	40	518.0
mediumMaze	144	2340	396.0
bigMaze	242	2374	318.0

Table 4.1

Modified:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	16	43	514
mediumMaze	87	1757	453
bigMaze	216	23869	344.0

Table 4.2

Euclidian Distance heuristic

Original:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	12	40	518.0
mediumMaze	144	2340	396.0
bigMaze	242	2374	318.0

Table 5.1

Modified:

Maze Form	Total Cost	Nodes Expanded	Score
tinyMaze	16	43	514.0
mediumMaze	87	1757	453.0
bigMaze	216	23869	344.0

Table 5.2

Conclusion

In a nutshell, in our final project, we have discovered the historical background of the well-known arcade game Pac-Man. We have analyzed various methods for creating Pac-Man controllers and scientific works conducted based on those methods. Afterward, we inspected the game in the sense of a problem-solving environment. Our group applied uninformed and informed searches to define the performance of Pac-Man, and we defined pills all over the maze as our goal states. As a result, out of all search algorithms, we have found that the BFS, UCS, and A* searches have exactly the same total scores and solution costs. Such as, BFS is a specific case of UCS, when all its path costs are equal, they perform exactly the same, and for the A* search turned out that the heuristics do not change results in this case. Meanwhile, the A* search has expanded less nodes compared to BFS and UCS. Scores generated by DFS are more or else similar to the scores generated by the other search algorithms. Although, the DFS expands the least amount of nodes out of all searches. Thus, in a sense of space, DFS is the best-performing search, while the A* is the best one in a sense of achieving the goal with the higher score possible, and less number of nodes expanded.

References:

A. Fitzgerald and C. Congdon, "RAMP: a Rule-based Agent for Ms. Pac-Man," in Evolutionary Computation, 2009. CEC'09. IEEE Congress on. IEEE, 2008, pp. 2646–2653.

A. Kalyanpur and M. Simon, "Pacman using Genetic Algorithms and Neural Networks," Retrieved from http://www.ece.umd.edu/˜adityak/Pacman.pdf (19/06/03), 2001.

dCode. (n.d.). *Maze Generator (perfect) - online text ASCII Labyrinth*. (Perfect) - Online Text ASCII Labyrinth. Retrieved December 10, 2022, from https://www.dcode.fr/maze-generator

DeNero, J., & Klein, D. (n.d.). (working paper). *Teaching Introductory Artificial Intelligence with Pac-Man*.

DeNero, J., Klein, D., & Abbeel, P. (n.d.). *The Pac-Man Projects*. Berkeley Ai Materials. Retrieved November 7, 2022, from http://ai.berkeley.edu/project_overview.html

Joshkarlin. (n.d.). *Joshkarlin/CS188-project-1: In this project, your Pacman agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. you will build general search algorithms and apply them to Pacman scenarios.* GitHub. Retrieved December 10, 2022, from https://github.com/joshkarlin/CS188-Project-1

J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

M. Gallagher and M. Ledwich, "Evolving Pac-Man Players: Can We Learn from Raw Input?" Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on, pp. 282–287, 2007.

M. Gallagher and A. Ryan, "Learning to play Pac-Man: an evolutionary, rule-based approach," Evolutionary Computation, 2003. CEC'03. The 2003 Congress on, vol. 4, 2003.

N. Tziortziotis, K. Tziortziotis, and K. Blekas, "Play Ms. Pac-Man using an Advanced Reinforcement Learning Agent," in Hellenic Conference on Artificial Intelligence. Springer, 2014,

Project 1: Search in Pacman. (n.d.). Retrieved November 14, 2022, from https://inst.eecs.berkeley.edu/~cs188/sp11/projects/search/search.html

Rohlfshagen, P., Perez-Liebana, D., Liu, J., & Lucas, S. M. (n.d.). *Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game*. Retrieved November 7, 2022, from http://www.diego-perez.net/papers/PacManConquersAcademia.pdf

S. Lucas, "Evolving a neural network location evaluator to play ms. pac-man," Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 203–210, 2005.

Thompson, T., McMillan, L., Levine, J., & Andrew, A. (2008). An evaluation of the benefits of lookahead in pac-man. *2008 IEEE Symposium On Computational Intelligence and Games*. https://doi.org/10.1109/cig.2008.5035655

T. Mott, 1001 Video Games You Must Play Before You Die. Cassell Illustrated, 2010.