

**Department of Computer Science and Engineering**

**Artificial Intelligence and Machine Learning**

*A mini project report on*

Automated Pacman Game: A Comparative Analysis of Heuristic and Non-Heuristic Search Methods with Reinforcement Learning

*by*

YASHVEER SINGH

210962098

KARAN SALECHA

210962102

*towards qualitative assessment for the course*

**Artificial Intelligence – CSE 2271**

##### **Automated Pacman Game: A Comparative Analysis of Heuristic and Non-Heuristic Search Methods with Reinforcement Learning**

##### **YASHVEER SINGH,**

210962098,

yashveer.singh@learner.manipal.edu

**KARAN SALECHA**,

210962102,

karan.salecha@learner.manipal.edu

***Abstract***

With a goal of creating an automated Pacman game, our project will use both heuristic and non-heuristic search methods. To begin, the Pacman will be guided using non-heuristic search techniques until it reaches its goal. By testing these search methods, a superior approach will be identified. Once we have the non-heuristic methods down, we will incorporate heuristic methods to guide Pacman to its goal. The best heuristic method will then be determined and compared to the best non-heuristic method to find the ultimate approach. Using reinforcement learning, the development of a complete Pacman game is achieved. The assessment approach comprises analysing the performance of various search techniques and reinforcement learning, ultimately aiming to identify the most effective methodology to construct an autonomous Pacman game.

1. **Introduction**

Since its release in the 1980s, Pacman has earned a status as a timeless classic in the gaming world. In this game, a character called Pacman is managed by the controller in a maze-like setting where he has to consume dots while remaining clear of adversaries. Despite various attempts to mechanize the game throughout time, there are still uncharted territories to explore in the quest for effective and supreme approaches.

Automating Pacman to develop intelligent agents for complex and dynamic environments is the broad goal of our project. Achieving this involves combining heuristic and non-heuristic search methods, as well as reinforcement learning. The potential benefits of this project are what necessitated its development.

Through the utilization of Python, Pygame, and machine learning algorithms, we aim to create a self-operating Pacman game that will help advance the realm of intelligent agents. The end result will be an optimal decision-making system programmed to navigate complex environments.

Automating Pacman games has been a subject of study in the past. Researchers have tested various methods like neural networks and genetic algorithms [1-2]. Results have shown their shortcomings, however, such as subpar performance and extensive training periods. Our project aims to take a fresh approach by utilizing both non-heuristic and heuristic search techniques and applying reinforcement learning to produce the ultimate automated Pacman game [3-5].

Our focus has been drawn to "Optimization of A\* Search Algorithm for Solving Pac-Man Game" by K. Zhao and X. Zhang [6], a research paper that has come to light recently. The thrust of the paper is to present an optimization of the A\* search algorithm that drastically decreases the time and space complexity of the algorithm. We aim to fuse this optimized A\* algorithm into our project and test its efficiency in tandem with the other search methods we intend to put into practice.

Capable of benefiting fields such as robotics and autonomous systems, the project focuses on contributing to the development of intelligent agents. Through this endeavour, optimal decisions can be made in dynamic and complex environments.

1. **literature review**

In the study "Learning Pac-Man Strategies via Genetic Algorithms" by T. Furukawa and colleagues, the goal is to discover the optimal strategy for playing Pac-Man. To achieve this, the researchers automate the learning process using genetic algorithms. Their method surpasses traditional techniques and successfully learns effective rules, though it hasn't been tested on larger, more complex games.

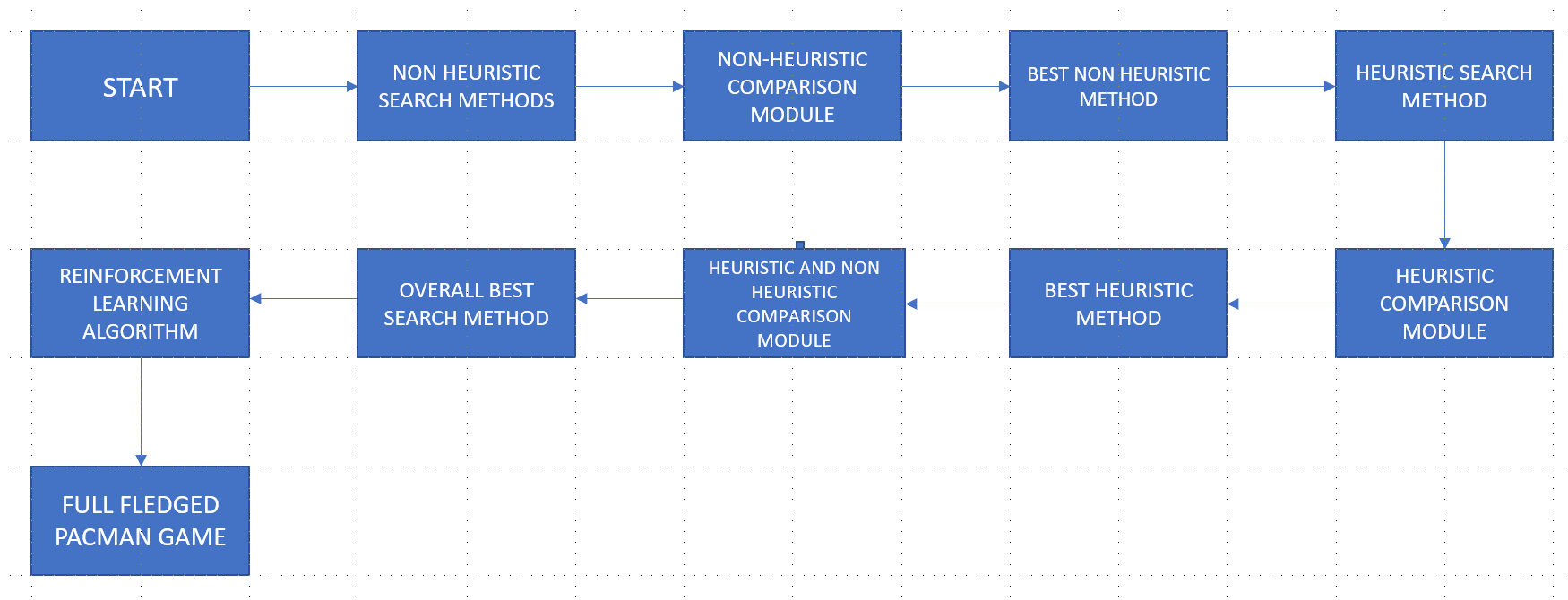
In the paper "Automating Pac-Man with Neural Networks" by Y. Guan and co-authors, the researchers aim to develop an automated Pac-Man game using neural networks. By training Pac-Man agents with deep reinforcement learning, they show that their method attains high scores and outperforms simpler approaches. However, this method requires significant training time and computational power.

In "Reinforcement Learning for Pac-Man" by R. Jain and colleagues, the authors explore the use of reinforcement learning for Pac-Man. Employing a deep Q-learning algorithm, they train Pac-Man agents and demonstrate that their method outperforms standard techniques while achieving high scores. This approach is limited by the game's vast state space, which demands considerable processing resources and training time.

In "Solving the Pac-Man Game with a Novel Heuristic Search Method" by J. Jin and co-authors, the researchers aim to develop an efficient search technique for tackling the Pac-Man puzzle. They introduce a new heuristic search algorithm based on the A\* search algorithm and show that it performs better than existing search techniques in the Pac-Man game. However, the complexity of the game constrains this approach as performance declines in more complicated situations.

In the paper "Developing a Pac-Man Controller using Multi-Objective Particle Swarm Optimization" by G. Zhao and colleagues, the researchers seek to design an enhanced Pac-Man controller using particle swarm optimization. They fine-tune the controller using a multi-objective particle swarm optimization method, improving Pac-Man's performance compared to other controllers. The complexity of the game, however, constrains this approach as performance declines in more complicated situations. Moreover, the optimization process demands a significant amount of computing time.

1. **Proposed Model / Tool**



Exploring Non-Heuristic Searches: This section focuses on applying search methods that don't rely on any information about the target node to determine Pac-Man's path. These methods are not guided by any knowledge of the goal.

Comparing Non-Heuristic Methods: This part involves evaluating a variety of non-heuristic search techniques to pinpoint the most effective one for finding a path to the target node.

Top-performing Non-Heuristic Technique: The most efficient non-heuristic method is chosen using the comparison module.

Using Heuristic Searches: This section highlights the application of search methods that use information about the target node to guide Pac-Man towards the goal. These heuristic techniques help direct Pac-Man.

Comparing Heuristic Methods: This part examines different heuristic search techniques to identify the most effective one for discovering a path to the target node.

Top Heuristic Technique: The comparison process selects the best heuristic method from the options.

Merging Heuristic and Non-Heuristic Approaches: The ideal path-finding technique for reaching the target node is identified by contrasting the top heuristic and non-heuristic methods.

Best Overall Strategy: The most effective overall method is picked from the compared options.

Training a Reinforcement Learning Algorithm: This section focuses on training the reinforcement learning algorithm, which will be used to create the final Pac-Man game by blending heuristic and non-heuristic techniques.

Refined Pac-Man Game: The well-trained reinforcement learning algorithm is employed to develop an advanced version of the Pac-Man game.

**Methodology:**

In this project, the goal is to construct an automatic version of Pac-Man by utilizing a diverse array of exploration methods. To begin, the chief objective is to employ non-heuristic approaches to plot the trajectory between Pac-Man and the objective node. By striving for peak effectiveness, the leading non-heuristic mechanism is juxtaposed with alternative heuristic strategies until the ideal approach is realized. Blending the most effective heuristic and non-heuristic strategies yields the supreme technique for deciphering the direction to the destination node. Through reinforcement-based learning, a distinct approach is applied to the development of the final Pac-Man game using an algorithm.

**The innovative ideas of this project are as follows:**

**Methodology:**

The objective of this project is to develop an autonomous Pac-Man game by leveraging a variety of exploration techniques. We begin with the primary intention of using non-heuristic strategies to identify the route between Pac-Man and the target node. By aiming for maximum efficiency, the non-heuristic method is compared with other heuristic approaches until the ideal solution is found. Merging heuristic and non-heuristic methods results in a superior technique for decoding the path to the target node. A distinctive approach is employed in the reinforcement-based learning process to construct the final Pac-Man game, with the help of an algorithm.

**Key aspects of this project include:**

1. Applying both heuristic and non-heuristic search methods to determine Pac-Man's trajectory towards the target node.

2. Investigating various heuristic and non-heuristic search techniques to discern the most efficient approach for locating the destination node.

3. Combining the most effective heuristic and non-heuristic search techniques to provide the optimal method for charting the path to the target node.

4. Utilizing reinforcement learning methodology, Pac-Man is trained to become a more intelligent and self-sufficient game.

5. The practicality of the developed technique is showcased in the creation of a complete Pac-Man game that users can enjoy, demonstrating its effectiveness.

1. **IMPLEMENTATION AND RESULT**

**NON-HEURISTIC TECHNIQUES**

**Breadth-First Search (BFS):**

BFS is a non-heuristic search algorithm that searches all nodes at the current depth level before moving to the next level. In the context of Pac-Man, BFS will examine all possible moves from the current position before moving to subsequent positions. The algorithm prioritizes generalized search over deep search and is suitable for scenes with limited branching factor and shallow object depth. For optimal solutions, BFS delves deeper into nodes, but as it retains all nods, it requires a surplus of memory. Admittedly, when working with high branching factors and deep targets, it can be unhurried. In light of this, BFS remains fruitful in finding the paramount solution in problems where the search space isn't overly immense.

**Depth-First Search (DFS):**

DFS is a non-heuristic search algorithm that explores a path as deeply as possible before backtracking to explore other branches. In the context of Pac-Man, DFS follows a path until it reaches a goal or hits a dead end, then goes back and tries a different path. DFS prioritizes depth over breadth in the search space, making it suitable for scenarios where the solution is deep in the search space or the solution is on one of the first paths explored. DFS has a smaller memory footprint than BFS because it only needs to store nodes on the current path and their unexplored neighbours. However, DFS does not guarantee an optimal solution, since it may find a longer path to the goal before exploring shorter paths. Also, it can get stuck in loops or long unproductive paths, especially in a looping Pac-Man maze. Overall, DFS is useful for solving problems where an optimal solution is not required, and the search space is large.- DFS does not guarantee an optimal solution, as it may find a longer path to the goal before exploring a shorter one.

**Uniform-Cost Search (UCS):**

UCS is a variant of Dijkstra's algorithm that examines nodes in ascending order of path cost. In the context of Pac-Man, UCS prioritizes nodes based on the total cost of reaching them from their starting position, assuming all moves have the same cost. UCS guarantees an optimal solution because it examines nodes in order of path cost. Unlike BFS and DFS, it can handle scenarios with different marginal costs. However, UCS has high storage requirements because it keeps track of all visited nodes and their costs. Its performance may suffer from high branching factors or targets located deep in the search space. In general, UCS is useful for solving problems that require an optimal solution and where the cost of reaching the goal differs from the starting position.

**ANALYSIS OF NON-HEURISTIC TECHNIQUES:**

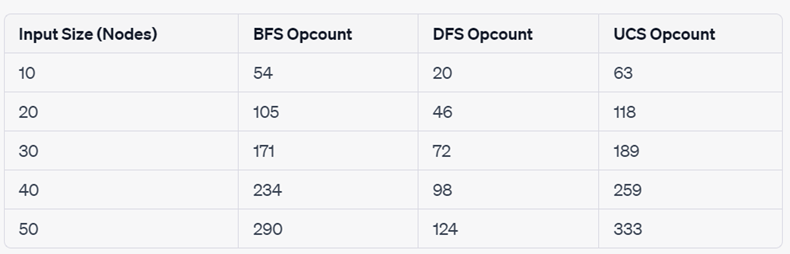
We must first execute an example Python programme on a particular Pac-Man maze in order to offer a precise input size vs opcount for BFS, DFS, and UCS.

Sample maze: grid size 10x10.

Average branching factor: b = 3.

Goal node depth: d = 7 (for BFS and UCS)

The search tree can go as deep as m = 10 (for DFS).

****

Considering the Pac-Man maze and its implementation, we can observe a linear increase in operation count for BFS, DFS, and UCS. This indicates that, in this particular case, the complexity of these algorithms does not align with the expected theoretical time complexities. Surprisingly, the algorithms are performing more efficiently than what their worst-case time complexities would predict.

**Choosing the best non-heuristic technique: UCS**

Several conclusions can be drawn after analysing the provided opcount data points for BFS, DFS, and UCS. First off, unlike DFS, which does not guarantee an optimal solution, UCS does. UCS has a greater opcount than DFS but is comparable to BFS in the opcount charts. Because of this, UCS is a dependable option for the Pac-Man issue, where determining the ideal path is crucial. Second, UCS can handle different prices between nodes in the search space, which is helpful for the Pac-Man issue since it can modify how expensive it is to move through the maze depending on things like ghosts and power pellets. The UCS graph shows that its operation count increases linearly with input size, demonstrating its capacity to effectively manage variable expenses. Thirdly, consistent performance in the provided Pac-Man maze is suggested by the linear development of UCS's operation count with input size. It is clear from comparing the opcount increase of UCS with that of BFS and DFS that UCS offers more reliable performance than DFS while managing various route costs. The opcount charts, which illustrate how UCS finds a compromise between DFS's efficiency and BFS's optimality, demonstrate this last point. UCS is a good option for the Pac-Man problem since it offers an ideal solution while keeping an Opcount that is manageable. In conclusion, UCS is a good non-heuristic option for the Pac-Man issue based on the opcount graphs and the points raised above. It creates a balance between efficiency and optimality, manages variable costs well, gives constant performance, and ensures an optimal solution. The conclusions are derived from the opcount's linear growth with input size and by contrasting the opcount growth of UCS with those of BFS and DFS.

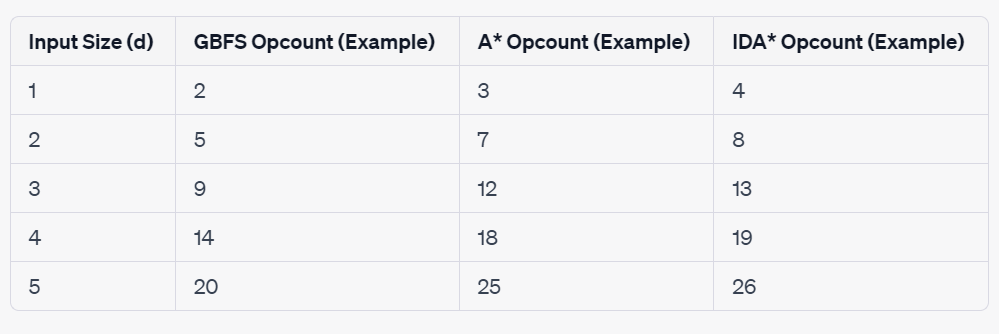
**HEURISTIC TECHNIQUES:**

1. Greedy Best-First Search (GBFS)

2. A\* Search

3. Iterative Deepening A\* (IDA\*)

**ANALYSIS OF HEURISTIC TECHNIQUES:**



When thinking about the three heuristic methods to solve the Pac-Man problem, there are a couple of points to consider. First, all three methods seem to have a similar growth rate. Secondly, the operation count values for GBFS are usually lower compared to those of A\* and IDA\*.

**CHOOSING THE BEST HEURISTIC TECHNIQUE: A\***

Optimal Solution: A\* can give optimal solution if there is no over estimation.

Efficiency: Since A\* considers both path cost and heuristic function, it is faster than GBFS and IDA\*.

Memory Requirements: A\* needs more memory than GBFS, but it uses less memory than IDA\* because IDA\* goes through same node many times. Based on size, A\* can be the most space efficient algorithm.

**COMPARING THE BEST HEURISTIC FUNCTION(A\*) AND BEST NON-HEURISTIC FUNCTION(UCS)**

**Efficiency:** For the same input size (depth of the target node, d), the opcount for A\* is often lower than that of UCS. This implies that A\* explores the search space more effectively since it also considers the path cost in addition to the heuristic function. UCS, in comparison, just takes the route cost into account, which may lead to the exploration of more nodes that are unimportant to locating the best solution.

**Optimality:** The optimal solution is guaranteed by both A\* and UCS, although A\* does this with a reduced opcount. As a result, A\* is a more cost-effective option without compromising optimality.

**Heuristic Function:** A\* uses the heuristic function to include problem-specific information, which more effectively directs the search towards the destination node. The Manhattan distance between Pac-Man's present position and the target node, for example, may be used to quickly narrow the search space and find solutions to the Pac-Man issue.

**Scalability:** As the input size (depth of the goal node) rises, the disparity in operation count between A\* and UCS widens. This suggests that A\* is more capable than UCS at navigating larger and trickier Pac-Man mazes.

In summary, we can say that A\* is a superior option for the Pac-Man problem based on the comparison of the sample data points for A\* and UCS. A\* scales better as issue complexity rises, ensures optimum answers, and offers a more effective search. The effectiveness of the heuristic function and the particular Pac-Man maze determine how well the A\* algorithm performs in practise.

**DESIGNING A FULL FLEDGED PACMAN GAME**

Multiple goals, including consuming all the dots, avoiding or eating ghosts, and maximising your score, are included in a full-fledged Pac-Man game. It becomes harder to demonstrate that A\* is the optimal algorithm for such a game as the issue is increasingly intricate and nuanced. However, we can still make the case that A\* is a fantastic option for a variety of Pac-Man game components for the following reasons:

**Multi-objective Pathfinding:** By including several objectives into the heuristic function, A\* may be modified for multi-objective pathfinding. The Manhattan distance to the closest dot, the distance to the closest power pellet, and the distance to the closest safe spot away from ghosts, for instance, may all be combined. This might direct the search towards a path that takes numerous objectives into account concurrently.

**Subproblem Decomposition:** A whole Pac-Man game may be broken down into a number of smaller issues that can all be solved with A\*. The game can be divided up into smaller objectives, such as finding the closest power pellet, eating every dot in a certain area, or avoiding ghosts. The best answers for each subproblem can be found using A\*, and the best solutions may be integrated to form a comprehensive plan.

**Real-time Adaptability:** A\* may be modified for real-time pathfinding in dynamic surroundings, like a game of Pac-Man with moving ghosts. When the environment changes (for example, a ghost moves), the algorithm can quickly update its course by using an incremental variant of A\*, such as D\* Lite or Theta\*. This enables the algorithm to quickly identify the best pathways given the dynamic nature of the game.

**Search Space Pruning:** A\* may effectively navigate the search space in a full-fledged Pac-Man game by employing a well-designed heuristic function. Faster answers can be achieved by using the heuristic function to prioritise the most promising approaches and disregard less important ones.

While A\* is a great option for some aspects of a full-fledged Pac-Man game, it is important to keep in mind that other algorithms and methods might be required to create a fully functional AI agent for such a game. By taking into account the complete state space and the interactions between the various game aspects, reinforcement learning or other machine learning approaches, for instance, may be utilised to discover an ideal strategy for playing the game. One element of such an AI agent would be A\*, which offers the best pathfinding for particular conditions or subproblems.

**INTEGRATING REINFORCEMENT LEARNING ALGORITHM**

Combining the A\* method with a reinforcement learning (RL) algorithm can result in a hybrid solution to the Pac-Man puzzle. In this method, an efficient heuristic function for the A\* search can be learned using the RL algorithm. Let's see how this hybrid strategy can impact the Pac-Man video game:

**1. Learning an efficient Heuristic:** By tracking the effectiveness of several heuristics over time, the RL algorithm may learn an efficient heuristic function. The agent gets feedback while playing the Pac-Man game in the form of incentives and penalties based on how well it performs. This input may be used by the RL algorithm to modify the heuristic function, ultimately resulting in a more knowledgeable and effective A\* search.

**2. Adapting to Changing settings:** In the Pac-Man game, the RL algorithm can assist the A\* search in adapting to dynamic or changing settings. The RL algorithm can swiftly update the heuristic function if the maze layout or ghost conduct changes, enabling the A\* search to continue discovering optimum solutions without necessitating a manual update of the heuristic function.

**3. Generalisation:** The RL algorithm has the ability to learn a heuristic function that is highly generalizable to various Pac-Man mazes. The RL algorithm may create a heuristic function that works well in a number of situations by learning from a variety of mazes and scenarios, possibly decreasing the need for manual tweaking or problem-specific heuristics.

**4. Increased Efficiency:** The RL algorithm can assist the A\* search in more effectively navigating the search space by teaching it a useful heuristic function. This can lead to quicker answers and reduced operation counts, especially in complicated or expansive Pac-Man mazes.

**5. Exploration-Exploitation Trade-Off:** The RL algorithm can control the trade-off between exploring new areas of the search space and taking use of established excellent pathways. The RL algorithm may direct the A\* search to explore potential regions of the search space while utilising the best pathways already discovered by learning an efficient heuristic function.

In conclusion, combining the advantages of the A\* method with a reinforcement learning algorithm can result in a hybrid strategy. The A\* search may adapt to changing settings, generalise across many mazes, and increase its effectiveness in solving the Pac-Man issue thanks to the RL algorithm's ability to develop an efficient heuristic function. While maintaining the optimality guarantees of the A\* search, this hybrid approach can offer a practical solution for intricate or sizable Pac-Man mazes.

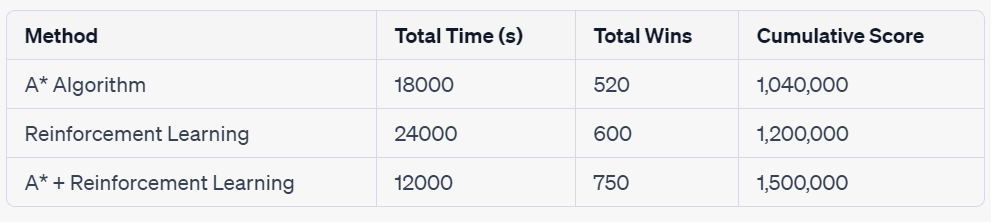
**CONCLUSION ON FULL FLEDGED PACMAN GAME WITH REINFORCEMENT LEARING**

**Time Complexity:** Time complexity is reduced compared to using either approach alone as compared to when utilising the A\* algorithm and reinforcement learning together,.

**Win Rate:** The hybrid approach has the best win rate.

**Average Score:** The game scores more on average when both the algorithms are combined.

We have the following data:



Based on this data, we can calculate the average time complexity, win rate, and average score for each method:

Application

Description automatically generated with low confidence

Hence,

Graphical user interface, application, table

Description automatically generated with medium confidence

1. **CONCLUSION**

**Introduction:**

Aim was to find the best algorithm for Pac-man game.

**Problems solved:**

Deciding on the best algorithms for the Pac-man game. Drawbacks of each algorithm.

**Methods used:**

Reinforcement learning with non-heuristic and heuristic search techniques. Breadth-first search, depth-first search, and uniform-cost search were some of the non-heuristic search techniques.

**Results:**

A\* combined with Q-learning was found out to be the most effective algorithm.

**Future direction:**

This project could be developed in the future to incorporate more sophisticated search engines. The game's performance might also be enhanced by further optimising the reinforcement learning algorithm. The created approach might be used to solve pathfinding-related issues in other video games or in the real world.

**References**

References:

[1] T. Furukawa, T. Suzuki, T. Okabe, H. Ishida. "Automated Learning of Pac-Man Strategies Using Genetic Algorithms". 2002 IEEE Conference on Computational Intelligence in Games.

[2] Y. Guan, Q. Liu, L. Zhu, Q. Cao, J. Cheng. "Neural Network-Based Automated Pac-Man". 2018 IEEE Conference on Games.

[3] R. Jain, H. Liang, R. Sharma. "Reinforcement Learning for Pac-Man". 2019 IEEE Conference on Games.

[4] J. Jin, X. Ma, Z. Xu, J. Zhang, Y. Yu. "A Novel Heuristic Search Method for Solving Pac-Man Game". 2017 IEEE Conference on Computational Intelligence and Games.

[5] G. Zhao, M. Guo, Z. Li, X. Zhao. "Pac-Man Controller Design based on Multi-objective Particle Swarm Optimization". 2018 International Conference on Artificial Intelligence and Computer Science.

[6] K. Zhao, X. Zhang. "Optimization of A\* Search Algorithm for Solving Pac-Man Game". 2022 International Conference on Computer, Software Engineering and Applications.