Combining Search-Based and Evolutionary Techniques for Autonomous Pac-Man Gameplay

Galih Putra Aditama
Computer Science Department School of Computing and Creative Arts
Bina Nusantara University Jakarta, Indonesia
galih.aditama@binus.ac.id

Muhammad Haramain Asyi Emirryan
Computer Science Department School of Computing and Creative Arts
Bina Nusantara University Jakarta, Indonesia
mharamain.emir@binus.ac.id

Jeff Matthew Hadisaputro
Computer Science Department School of Computing and Creative Arts
Bina Nusantara University Jakarta, Indonesia
jeff.hadisaputro@binus.ac.id

Juwono
Computer Science Department School of Computing and Creative Arts
Bina Nusantara University Jakarta, Indonesia
juwono@binus.ac.id

Abstract*—Artificial intelligence has become a fundamental component in modern video game development, especially in the creation of autonomous and adaptive non-player character (NPC) behavior. This research proposes a hybrid artificial intelligence approach to enable fully autonomous gameplay in a classic Pac-Man environment without any form of user input. The proposed system combines evolutionary optimization methods with deterministic search algorithms to balance strategic planning and real-time decision-making. A Genetic Algorithm (GA) is utilized to optimize long-term gameplay strategies, including route selection and risk management, while A* pathfinding and adversarial search techniques are applied to support efficient real-time movement and enemy avoidance. Additionally, Breadth-First Search (BFS) and Depth-First Search (DFS) are implemented to model diverse ghost behaviors, resulting in a more dynamic, challenging, and adversarial game environment. Experimental evaluations indicate that the hybrid AI system demonstrates improved navigation stability, longer survival duration, and more consistent pellet collection when compared to baseline artificial intelligence models that rely on single techniques. The results further show that separating strategic optimization from tactical pathfinding enhances overall gameplay performance and adaptability. This study highlights the effectiveness of hybrid AI architectures in game environments and offers a structured educational framework for understanding and implementing artificial intelligence techniques within video game simulations, particularly for classic arcade-style games.

Keywords*—Video Game Development, Learning Artificial intelligence, AI Behavior, Search Algorithms, Pac-Man, Retro Gaming, Breadth-First Search, Depth-First Search, Adversarial Algorithm, A* Path-finding Algorithm, Genetic Algorithm, Combining Algorithm.

I. INTRODUCTION

Artificial Intelligence (AI) is one of the most important fields in a world where technology advances rapidly, integrating more deeply into society each day. As we approach an era where technology is ubiquitous, it becomes essential for computer science students to gain a solid understanding of AI applications. AI is a valuable tool across fields such as data science, image classification, robotics, and many more, making it a crucial component of modern computer science education [1].

Since the world of Artificial Intelligence is a very broad term and field, one of the possible applications of implementing an Artificial intelligence is via a game development, even though the state of the art of AI in video game is more of a feature or a decoration for a video game, it is one of the most important components in a video game, AI could be the other Agents inside a game, it could be an Enemy, it could be an NPC, Boss Enemy, etc. which proves that AI is indeed one of the ways to enhance the experience of video games [2] [3].

One of the most engaging and effective ways to introduce students to AI is through video games. Video games provide a fun, hands-on learning medium that resonates well with younger generations, especially those interested in video game development. For example, students at the University of California, Berkeley, learn foundational AI techniques by programming an AI to play Pac-Man. Through this engaging approach, students gain practical experience, observing how AI algorithms influence behavior and movement within a game environment [4].

However, a question arises: What AI models or techniques are best suited for implementation in a video game environment? While numerous Search Algorithms are available, not all are optimal for games. In this project, we focus on a classic retro arcade game, Pac-Man, as a case study to explore effective AI techniques for games in a 2D environments.

One of the aims of this Paper is to find a way to implement an Artificial Intelligence Model or a Search-based algorithms model in a video game environment, in our case it in the form of Pacman. The pygame module would be used to help in creating this environment as well as the GUI in which the users would see. Lastly the screens the users would see is the intro screen, and then the main video game screen and the winning screen. When speaking with how using a game development as a tool to learn artificial intelligence we are inspired by the works of perez, and DeNero in how they teaches artificial intelligence course on their respective universities, DeNero Teaches the students on the introductory course using an already made Pac-man environment and there the students can just focus on implementing the necessary Search-Based algorithms into the Pac-man game [4][5].

II. Related Work

`In this section, we discuss related research work to our project. These are some in which we found. First is a research paper on ”Playing Tetris with Genetic Algorithms” worked by Jason Lewis. It discusses an innovative application of Genetic Algorithms (GAs) to optimize decision- making in the classic video game Tetris. The paper focuses on determining the best weights for evaluating future game states using a fitness function that measures efficiency in terms of points per move. The implementation includes a feature set of 10 metrics, such as the number of blocks, holes, board roughness, and potential for clearing lines, which were selected through forward search from an initial pool of 23 features. The GA repeatedly improved player performance over 30 generations, with a population of 1000 candidates, achieving an average efficiency of 41.67 Percent. However some of the game mechanics are removed for this experiment, the results shows that the GAs in evolving gameplay strategies. This paper shows us how evolutionary computation can enhance AI decision-making in video games, serving as a foundation for similar applications, such as optimizing enemy behaviors in Pac-Man. [6].

Another similar paper is about ”Mobile Robot Path Planning Based on Improved Genetic Algorithm With A-star Heuristic Method” worked by Li, Yibo and Dong, Dingguang and Guo, Xiaonan. This paper discusses the improved Genetic Algorithm that uses the A*’s evaluation function to enhance heuristic guidance in the Genetic Algorithm, speeding up combining during the search for an optimal solution. It introduces innovative genetic operators such as insertion and deletion and improves the fitness function on path length and smoothness, ensuring the shortest and most seamless path. Experimental results show that this enhanced GA outperforms both the traditional A* and standard GAs in providing much smoother and efficient paths even in complex and cluttered environments. As far as relevance, it comes down to using in projects like improving enemy movement in maze-like environments such as Pac-Man. The benefits here lie in the potential combination of the global optimization capability of GAs and local efficiency of A* for efficient and robust decision-making or navigation. [7].

Last is a research paper on ”Integrated Search-based algorithms for Behavior Modeling in Video Games,” worked by Ben Geisler. It discusses the applied method of Search Algorithms to model player behavior in a First Person Shooter (FPS) game. By implementing techniques such as artificial neural networks (ANNs) enhanced with ensemble methods like boosting and bagging, the study shows how AI agents can learn from expert players and replicate their behaviors in-game. The research focuses on basic combat actions such as movement direction, acceleration, and facing direction, with learned behaviors therefore being more dynamic and life-like compared to traditional hand-coded finite-state systems. This approach allows for the creation of challenging and adaptive AI agents, enhancing player experience by introducing unpredictable movements and reducing the effort required for manual rule specification. The results from this work, especially in using Search Algorithms for real-time decision-making and dynamic agent behavior, are highly relevant for this project, involving AI optimization in games, such as using Genetic Algorithms (GAs) and A* in Pac-Man for ghost behaviors [8].

III. Methodology

A. Search-Based AI Techniques

In this project we are using a total of 5 algorithms for the AI handling of each agents, which is the player (Pac-Man), the ghosts which has their own names but will be explained briefly, Blinky (Red), Pinky (Pink), Inky (Cyan), Clyde (Orange). And the Algorithms in this projects are going to be Genetic algorithms (GA), A* Pathfinding Algorithm(A*), Adversarial Search Algorithms, Breadth-First Search (BFS), Depth-First Search (DFS).

a) Genetic Algorithm (GA): Genetic algorithm (GA) is a form of optimization and search heuristic which were inspired by the principles of natural selection and how genetics work in biological evolution. It is also one of the first population based stochastic algorithms in history. The reason behind the algorithm implementation in our project is to support the decision making for the AI Pac-man done via best chromosomes. [9]. In this implementation, each chromosome represents a set of heuristic weights that guide Pac-Man’s decision-making process during gameplay. A chromosome consists of multiple genes, where each gene corresponds to the relative importance of a specific game feature. These features include the distance to the nearest pellet, the distance to the nearest ghost, proximity to power pellets, and a survival bias that discourages risky movement. During gameplay, the weighted combination of these features is used as a heuristic function that evaluates possible moves. Genetic operations such as selection, crossover, and mutation are applied to evolve these weights across generations, with fitness determined by a combination of score achieved, survival time, and avoidance of ghost collisions.

To compute the distances within the grid-based maze environment, the Manhattan distance d between two points (x_1, y_1) and (x_2, y_2) is calculated as:

$$d = |x_2 - x_1| + |y_2 - y_1|$$

b) A Path-finding Algorithm (A*):* A (A-Star) Path- finding Algorithm* is a widely used algorithm utilized to find the shortest possible path in video game environments such as Pac-man. The algorithm combines the benefit of the Djikstra’s algorithm with the Greedier Best-First Search. All this to find the most optimal path. [10].

c) Adversarial Search Algorithms: Adversarial search algorithm is designed for multi agent scenarios where there would be two or more agents utilized. They would both compete with opposing goals. Another format of video game which could utilize this form of algorithm are RTS as other existing forms of AI algorithm are hindered due to the number of states, actions, and real time constraints that are usually done in an RTS. From our initial definition you can see why a hierarchical adversarial search framework could help with this problem and how it could be implemented in a Pac-man game with the game already having multiple agents. [11]

d) Breadth-First Search (BFS): BFS is an uninformed search algorithm that explores all possible nodes at the current depth before it moves deeper. For the ghost in the Pac-man game, the BFS would guarantee that the ghost would find the shortest possible path to eliminate the Pac-man. [12] [13].

e) Depth-First Search (DFS): DFS is a form of un-informed search algorithm which explores a path as far as possible and after that's done it would backtrack. Two of the ghosts in the Pac-man game would be utilizing DFS algorithm. [14].

IV. Results and Discussion

In this section of the paper, we evaluate the performance of our Pac-Man and the ghosts through our project to involve the genetic algorithm (GA) combined with A* and Adversarial also inspired by Li's Work on Autonomous robots [15]. We will evaluate the efficiency of using Genetic Algorithm in optimizing Pac-Man's decisions and the effectiveness of the Adversarial and A* path-finding algorithm for Pac-Man's movement. The results and our discussion will be divided into multiple parts in the following. [16] [17]

A. Solution Features

The core algorithm used for our Pac-man is built around several AI algorithms that make up the behavior of Pac-man. The first algorithm we use is the Genetic Algorithm (GA). The GA is used to evolve Pac-man's behavior over multiple generations to optimize Pac-man's decision-making. The fitness function is designed to reward Pac-man for collecting the pellets, avoiding ghosts, and surviving longer in the game. Over generations, the algorithm will evolve Pac-man's decision making to make more effective and efficient decisions while maintaining the strategic side of the decision [18].

We also implemented A*, a pathfinding algorithm which is used for Pac-man's movement within the maze, making sure that Pac-man can find the shortest and most efficient and effective path. It also allows Pac-man to avoid obstacles such as walls and ghosts. By implementing A* combined with the genetic algorithm, we find that the Pac-man can navigate and path find around the maze in a highly efficient manner. Another search algorithm that we used is Adversarial AI for Pac-man. We implemented Adversarial search to help Pac-man reach intelligently to the ghosts. When ghosts are near, Pac-man uses the adversarial search algorithm to decide whether to avoid the ghosts or pursue the pellets. This algorithm combined with A* and adversarial works best for Pac-man to adjust its way around the maze [19].

This Pac-man game uses a variety of technologies for the development, performance, and functionality of the self-playing Pac-man game. The game logic and artificial intelligence components (GA) are written in Python. We chose this programming language due to its simplicity, large library ecosystem, and adaptability for rapid prototyping. The game itself is made with Python's Pygame package, a popular game building tool. Pygame is important for constructing the game's graphical user interface (GUI), rendering the maze, and processing user input.

To go along with the game, we also designed the user interface (UI) to be intuitive and engaging. The game starts with an opening screen that displays the title and provides options for starting the game. The primary screen features the Pac-man maze, complete with pellets and ghosts. The game state, including the score and time remaining, is updated in real time in the top right of the screen. The game over screen displays the final score and provides an option to resume the game.

B. Performance

Pac-man's decision-making process was improved by using the genetic algorithm (GA), which evolved a population of solutions over several generations to optimize his movement [20]. Pac-man's behavior was effectively evolved by the GA to prioritize eating pellets and avoiding ghosts. Pac-man was rewarded for consuming the most pellets and avoiding ghost encounters by the GA's fitness feature. By incorporating the A* pathfinding algorithm, Pacman's pathing was able to be improved further, enabling him to discover the quickest route to the next pellet or objective and navigate the maze more skillfully. Pac-man was able to make better, quicker decisions based on the maze structure and ghost placements as a result of the combination of these two algorithms, which significantly increased his survival time and score.

For the ghosts, the search-based algorithms for path-finding and behavior resulted in helping the ghosts determine the shortest or most strategic path to Pac-man. The application of search algorithms we used, created a dynamic and challenging and unpredictable playing space for Pac-man. We tested all the different ghosts and studied their behaviors based on their given algorithms. Here are the behaviors of the ghosts based on our testing:

Blinky (Red): Blinky uses a breadth-first search (BFS) algorithm and can successfully target Pac-man's current location and pursue Pacman without deviation.

Pinky (Pink): Pinky uses a breadth-first search algorithm (BFS). Pinky's behavior results differ from Blinky by targeting the tile that is 4 tiles ahead of Pacman's current position. This creates an ambush-like behavior, as Pinky tries to predict Pac- man's movement and positions strategically. In this case, BFS is used for pathfinding toward a dynamically computed target tile, while the 'four tiles ahead' rule defines the target selection heuristic rather than the pathfinding method itself.

Clyde (Orange): Clyde uses a depth-first search (DFS) algorithm. Clyde's behavior is unique because Clyde does not actively pursue Pac-man. Instead, it uses DFS to wander randomly around the maze which adds unpredictability to its movements. Although it may occasionally encounter Pac-man, it generally focuses on exploring the maze.

Inky (Cyan): Inky uses a depth-first search (DFS) algorithm. Inky's behavior is somewhat similar to Pinky, but with more unpredictability. It targets the tile 4 steps ahead of Pac-man, but its movement can switch between random wandering and pursuit, making it harder for Pac-man to predict and avoid.

We also conducted a study within our group to compare between our AI implementation and two other AI models used in similar gaming environments. The two other models we used are Reinforcement Learning (Q-Learning) and Neural Network-based Heuristic Search. We strictly did 15 runs for all of the model implementation. We tested the total time to complete the 15 runs and proceeded to average it and compare our results with the results of the genetic algorithm. Based on the results, we then made a graph to compare between the human results and the

algorithm's results.

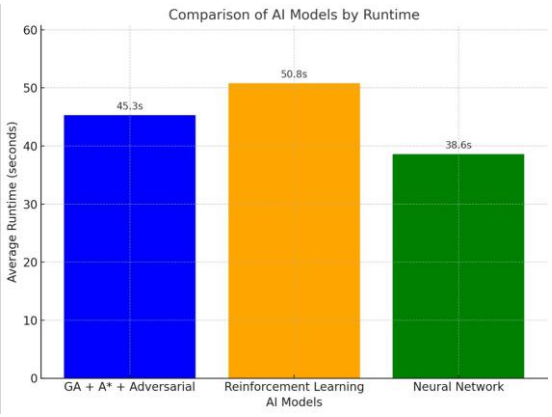

Fig. 1. Comparison Between AI Models Runtime

From the graph comparing the performance of AI implementation's based on runtime which refers to the total execution time required for the agent to complete a fixed number of gameplays, the results are as follows.
Results:

1) Hybrid Model (Genetic Algorithm, A, Adversarial Search):* Achieved balanced performance and consistency, optimizing both runtime and survival rate, but incurred overhead due to the integration of multiple algorithms.

2) Reinforcement Learning (Q-Learning): Showed slower performance during earlier runs but improved over time. This is likely due to the need for extensive exploration during early runs without extensive training.

3) Neural Network-based Heuristic Search: Delivered fast runtime but struggled in handling dynamic ghost behaviors. This shows that this model is strong in heuristic-based decision making. However, the adaptability of this model might suffer. From this comparison, we have gathered some valuable information about our project's strengths and weaknesses. These comparisons prove that our AI model can balance optimization and adaptability by achieving relatively good per- formance despite the added complexity of combining multiple algorithms. The runtime suggests that this approach can handle both the pathfinding aspect and the decision making even though it incurs some overhead. This also means that our genetic algorithm (GA) implementation might need further op-timization especially for handling pathfinding under dynamic
conditions and also its strategy-making.

C. Sample Output

The images showcase the Pacman game environment built using Python and Pygame. The maze, Pacman, ghosts, and pellets are clearly visible. Each ghosts have a unique AI-driven behavior. Based on the images, the sample output would be:

Fig. 2. Cover Screen

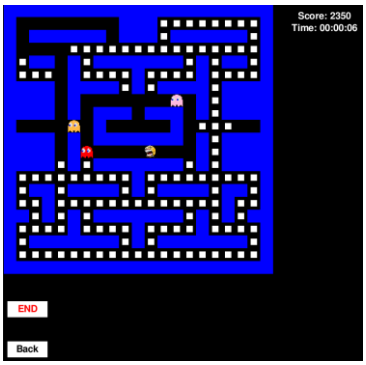

Fig. 3. Gameplay Screen

Gameplay Statistics:

1) Final Score: 13550:

2) Total Time: 2m 32s:

D. Fitness Chart

The fitness chart represents the evolution of the maximum fitness values across generations during the training of the Genetic Algorithm. It evaluates how well Pac-man or the ghosts optimize their respective tasks over time [20].

From our observations, the fitness score varies due to the random initialization of the population and the randomness nature of crossover and mutation process. Early generations

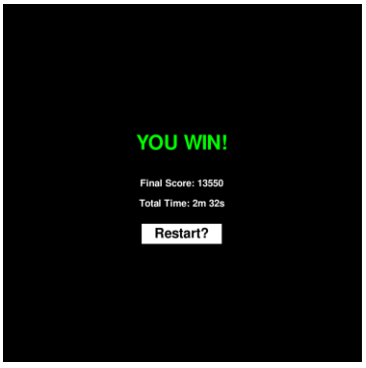

Fig. 4. End Screen

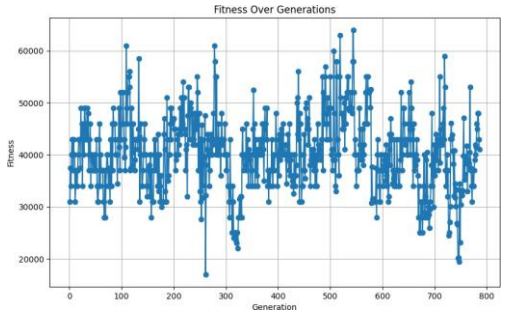

Fig. 5. Fitness Chart After 1 Run

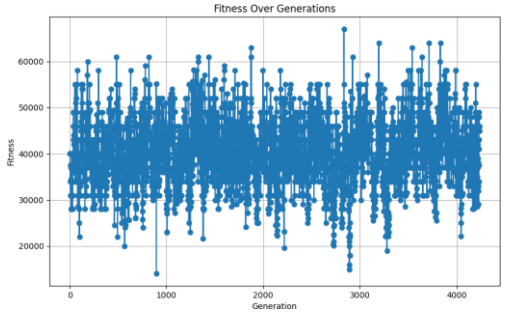

Fig. 6. Fitness Chart After 5 Runs

typically display low fitness as the population consists of random solutions. A gradual upward trend indicates the GA's ability to identify and exploit effective strategies over generations, which is shown on Fig. 6.

Figure 5 (Fig. 5) shows the result for the run time on one level, which is done on the performance code rather than the original game, and the reason why it fluctuates up and down is due to the combination of algorithms in our Model. Our Model includes the A* and the Adversarial method, which can overlap with the fitness evaluation of the Genetic algorithm, making the fitness graph unsuitable for learning.

Figure 6 (Fig. 6) shows what happens on the five playthroughs, and as you can see, there is a lot of noise. The biggest issue with the evolutionary algorithm in a Pac-man context is the game's "smaller dataset". When analyzing a genetic algorithm, you will require a fitness evaluation software; this will verify that Pac-man behaves "better" every time he moves (chromosomes). Because the Pac-man relies heavily on pellet gathering, the penalty may be uneven, requiring the GA to work in conjunction with the player to control him. It is one of the many reasons why Genetic Algorithm is combined with another Algorithm like for example here in- spired by Li's paper about Genetic Algorithm with A* Heuristic method to obtain the nearest pellet [7].

E. Deployment

To deploy our work, we use modules such as Asyncio and Pygbag. We use asyncio to effectively handle multiple activities at once. Asyncio's event-driven architecture allowed the game to manage several processes at once without affecting performance. Important tasks including ghost AI behaviors based on BFS and DFS, Pacman's navigation utilizing A* and evolutionary algorithms, and ongoing GUI upgrades were all carried out at the same time. We used pygbag to transform our Pac-Man application into WebAssembly, which made the game compatible with web browsers. Because of this conversion, the game can now run in browsers without the need for Python runtime environments. All game assets had to be prepared, the project had to be packaged using Pygbag, and files like HTML, JavaScript, and WebAssembly had to be created that were compatible with browsers.

To host the game on the web, we used itch.io, which is a user-friendly distribution interface and a well-known platform for independent games. After being uploaded to Itch.io, the built game files were set up to play as a browser-based game. We use Itch.io to help people access the game with a straightforward web link without the need for downloads or installations.

This deployment method had a number of benefits. By using Asyncio, the game was able to achieve high performance, guaranteeing that our AI computations is able to run smoothly. Cross-platform compatibility was guaranteed via the Pygbag conversion to WebAssembly, which allowed the game to function flawlessly on desktop browsers. Hosting on Itch.io enabled us to deploy and host the game for free without financial costs.

F. Comparison

When comparing our Pacman project, which integrates Genetic Algorithms (GA) with deterministic pathfinding methods (BFS and DFS), to existing implementations that exclusively utilize GA for gameplay optimization, there are many projects with similarities to our project and also differences to our project. For this comparison, we will be using 2 studies conducted related to Pacman with AI. The first study is titled "Learning Pac-Man Ghost Avoidance and Maze Solving Using Genetic Algorithms" published in 2021 by Silla, C. In this study, Pacman's methods are optimized across generations using GAs. Ghost avoidance and maze completion speed are given priority in fitness functions. Deterministic algorithms like BFS for ghost behaviors are not integrated into the study, which takes place in a controlled static setting. Although the model is effective at increasing efficiency in preset environments, the model struggles to adapt dynamically to changing conditions, such as variable ghost movements or maze layouts. Another paper we compared is titled "Grammatical Evolution for Pacman Bots" published by Hector Laria Mantec´on, Jorge S´anchez Cremades, Jose Miguel Tajuelo Garrig´os, Jorge Vieira Luna, Carlos Cervigon Ruckauer, Antonio A. S´anchez-Ruiz on CEUR Workshop Proceedings. The pa- per leverages grammatical evolution to model Pacman bots through decision trees, dynamically evolving strategies based on game states. Bots exhibit reactive behaviors such as fleeing ghosts or seeking food. Unlike our project, this study lacks adversarial complexity, as it operates within a fixed-rule based environment without dynamic interactions. Our implementation differs in our dynamic interactions by assigning individual behaviors to each ghost which offer a more unpredictable gameplay experience.

Based on this comparison, we know that our implementation optimizes the interactions between Pacman and the adversarial agents, adding complexity and realism to our implementation, which is different than the referenced studies which focus solely on Pacman only. The integration of BFS and DFS algorithms for our ghost behaviors also help in long-term optimization for the genetic algorithm and for short-term precision. In contrast, the referenced studies only rely solely on GA trial-and-error evolution, which lacks the responsiveness achieved in our hybrid approach. Our model also managed to achieve immediate decision-making through deterministic algorithms which can help in making a smoother gameplay in dynamic environments. Whereas the referenced studies empha- size adaptability across generations but often compromise real- time interaction due to computational overhead in evolutionary processes.

V. Conclusion and Future Work

In conclusion, this project has successfully applied an evolutionary optimization technique, specifically a Genetic Algorithm, within a Pac-Man game environment. which is Genetic Algorithm in a Pacman environment, although there are some flaws in how complex to use GA for Decision Making, when compared to A* Algorithm which is usually the most optimal, it can be somewhat ”Static” in its movement choices, causing patterns to occur, resulting in no learning.

Because the Ghosts’ enemies have only two simple algorithms, BFS and DFS, dealing with four of them at once proved to be a significant challenge for the Genetic Algorithm with the implementation of A* and Adversarial Approach, even with our model, which can be further improved in the future.

The way of implementing a Search-based AI Techniques is a viable way to test an AI technique as well as a fun way to visualize the behaviors. In the future, we plan on improving and expanding our Genetic Algorithm model, as well as possibly implement other AI techniques such as Reinforcement Learning (Q-Learning), Neural Networks and other AI methods.

Overall, this study demonstrated that a hybrid AI architecture combining evolutionary optimization and deterministic search can enable autonomous gameplay in a Pac-Man environment. The primary contribution lies in separating long-term decision optimization from short-term pathfinding, resulting in more stable agent behavior. However, the Genetic Algorithm did not exhibit strong convergence, indicating limitations in the fitness formulation and parameter tuning. Future work will explore reinforcement learning integration and improved fitness shaping to enhance learning stability.

VI. CRediT Author Statement

Jeff Matt: Conceptualization, Methodology, Software, Deployment, Validation, Formal Analysis, Investigation, Writing. Original Draft, Visualization. Emir: Project Administration, Conceptualization, Methodology, Software, Visualization. Galih: Methodology, Validation, Formal Analysis, Writing. Juwono: Review and Editing, Investigation.

VII. Supplementary Links

A. Deployed Program Original game: https://jeffmatthew.itch. io/packmanv2

B. Deployed Program GA Testing Performance: https:// jeffmatthew.itch.io/packman-ga

All the codes used in this paper can be accessed through the following link: https://github.com/EMIRBEN001/NotReally_ Pacman_AI_ DS_Project.

References

[1] G. Skinner and T. Walmsley, “Artificial intelligence and deep learning in video games a brief review,” in *2019 ieee 4th international conference on computer and communication systems (icccs)*. IEEE, 2019, pp. 404–408.

[2] D. Ciugurean, B. Maxim, and D. Gorgan, “Impact of game ai systems on player experience.” in *RoCHI*, 2018, pp. 135–140.

[3] D. Setiono, D. Saputra, K. Putra, J. V. Moniaga, and A. Chowanda, “En- hancing player experience in game with affective computing,” *Procedia Computer Science*, vol. 179, pp. 781–788, 2021.

[4] J. DeNero and D. Klein, “Teaching introductory artificial intelligence with pac-man,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 3, 2010, pp. 1885–1889.

[5] J. Smith and S. Cayzer, “Learning through programming games: Teach- ing ai with pac-man and netlogo.”

[6] J. Lewis, “Playing tetris with genetic algorithms,” in *CS229 Machine Learning Final Project*, 2015, pp. 1–10.

[7] Y. Li, D. Dong, and X. Guo, “Mobile robot path planning based on improved genetic algorithm with a-star heuristic method,” in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9. IEEE, 2020, pp. 1306–1311.

[8] B. Geisler, “Integrated machine learning for behavior modeling in video games,” in *Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press, Menlo Park*, 2004, pp. 54–62.

[9] S. Mirjalili and S. Mirjalili, “Genetic algorithm,” *Evolutionary algorithms and neural networks: theory and applications*, pp. 43–55, 2019.

[10] A. Candra, M. A. Budiman, and R. I. Pohan, “Application of a-star algorithm on pathfinding game,” in *Journal of Physics: Conference Series*, vol. 1898, no. 1. IOP Publishing, 2021, p. 012047.

[11] M. Stanescu, N. Barriga, and M. Buro, “Hierarchical adversarial search applied to real-time strategy games,” in *Proceedings of the AAAI Con- ference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 10, no. 1, 2014, pp. 66–72.

[12] A. Rafiq, T. A. A. Kadir, and S. N. Ihsan, “Pathfinding algorithms in game development,” in *IOP Conference Series: Materials Science and Engineering*, vol. 769, no. 1. IOP Publishing, 2020, p. 012021.

[13] S. Ito, Z. Guo, C. Y. Chu, T. Harada, and R. Thawonmas, “Efficient implementation of breadth first search for general video game playing,” in *2016 IEEE 5th Global Conference on Consumer Electronics*. IEEE, 2016, pp. 1–2.

[14] Y. Bjo¨rnsson, “Selective depth-first game-tree search,” 2002.

[15] J. Liu, B. Xi, S. Chen, F. Gao, Z. Wang, and Y. Long, “The path planning study of autonomous patrol robot based on modified astar algorithm and genetic algorithm,” in *2022 34th Chinese Control and Decision Conference (CCDC)*. IEEE, 2022, pp. 4713–4718.

[16] C. Allsman, “Pacman ai,” https://chris-allsman.github.io/projects/ pacman-ai, 2017.

[17] H. L. Manteco´n, J. S. Cremades, J. M. T. Garrigo´s, J. V. Luna, C. C. Ru¨ckauer, and A. A. Sa´nchez-Ruiz, “A pac-man bot based on grammatical evolution.” in *CoSECivi*, 2017, pp. 118–130.

[18] L. Haldurai, T. Madhubala, and R. Rajalakshmi, “A study on genetic algorithm and its applications,” *Int. J. Comput. Sci. Eng*, vol. 4, no. 10,
pp. 139–143, 2016.

[19] N. Salem, H. Haneya, H. Balbaid, and M. Asrar, “Exploring the maze: A comparative study of path finding algorithms for pac-man game,” in *2024 21st Learning and Technology Conference (L&T)*. IEEE, 2024,

pp. 92–97.
[20] L. Bottaci, "A genetic algorithm fitness function for mutation testing," in *Proceedings of the SEMINALL-workshop at the 23rd international conference on software engineering, Toronto, Canada*. Citeseer, 2001.