Journal homepage: https://itce.com.ua/en

Information Technologies and Computer Engineering **Vol. 21, No. 3. 2024**

Article's History: Received: 20.09.2024 Revised: 22.11.2024 Accepted: 26.12.2024

UDC 004.94 10.63341/vitce/3.2024.108

Analysis of the decision-making algorithm efficiency in complex game environments on the example of Pac-Man

Artem Novikov*

Postgraduate Student
V.N. Karazin Kharkiv National University
61022, 4 Svobody Sq., Kharkiv, Ukraine
https://orcid.org/0009-0004-5914-7098

Volodymyr Yanovskyi

Doctor of Physical and Mathematical Sciences, Professor
"Institute for Single Crystals" of National Academy of Sciences of Ukraine
61072, 60 Nauky Ave., Kharkiv, Ukraine
https://orcid.org/0000-0003-0461-749X

Abstract. Game simulations such as Pac-Man are substantial for testing decision-making algorithms in conditions that mimic real-life scenarios. This creates new opportunities for the development of autonomous systems that can adapt to changing environmental conditions and interact with other agents. The study aimed to compare Expectimax, Monte Carlo Tree Search, and Alpha-Beta Pruning algorithms in the changed conditions of the Pac-Man game to determine the most efficient approach to decision-making in complex environments. For this purpose, simulation modelling was used to evaluate the effectiveness of agents in various game mazes that differ in complexity. The study measured such indicators as the number of points, game time, and percentage of winnings, which were used to assess the effectiveness of algorithms in different situations. The analysis of the experiments determined that the Monte Carlo algorithm is the most effective among the tested methods for solving less complex mazes, confirming quickly optimal path search in simple conditions. The Alpha-Beta Pruning algorithm demonstrated less efficiency, which indicates the need to optimise it for more complex environments. Expectimax demonstrated significantly lower performance, which indicates its limited suitability for complex game mazes. The study demonstrated that increasing the complexity of the mazes significantly reduces the performance of all algorithms, especially with more obstacles, highlighting the importance of developing more robust methods for highly complex environments. Optimising the Monte Carlo and Alpha-Beta Pruning algorithms for complex environments can significantly improve their performance and make them effective for real-world applications in navigation and control of moving devices. The results of this study can be used to develop efficient navigation algorithms for autonomous vehicles, drones and other robotic systems where adaptation to changes in complex environments is critical

Keywords: simulation environments; Expectimax; MCTS; Alpha-Beta Pruning; autonomous navigation

Introduction

The research relevance is determined by the growing need for artificial intelligence (AI) algorithms capable of fast and efficient decision-making in complex, dynamic environments. Such algorithms are critical in industries related to autonomous navigation, drone control, logistics process optimisation, and other areas requiring adaptation to changing conditions and interaction with other agents. They must provide stable performance even when resources are limited. Of particular importance are studies of the adaptability of algorithms in simulation environments that model real-world conditions.

Different approaches to the development of AI algorithms are actively discussed in the scientific literature. For instance, the A.W.R. Ramadhan & D. Udjulawa (2020)

Suggested Citation:
Novikov, A., & Yanovskyi, V. (2024). Analysis of the decision-making algorithm efficiency in complex game environments on the example of Pac-Man. *Information Technologies and Computer Engineering*, 21(3), 108-118. doi: 10.63341/vitce/3.2024.108

*Corresponding author

compared the performance of A* and Dijkstra algorithms in the Pac-Man game, demonstrating the advantage of A* in finding the shortest path due to better cost optimisation and faster execution. The study also highlighted the importance of methodical prototyping to evaluate the performance of algorithms in dynamic environments. N. Salem *et al.* (2024) confirmed the advantages of A* by showing its effectiveness in path planning and cost minimisation tasks in the context of a game. In particular, the paper addressed the role of cost optimisation, and the number of nodes expanded during the search.

Innovations in the creation of environments for testing algorithms are emphasised. K.M. Cheng *et al.* (2024) investigated maze generation algorithms for Pac-Man, in particular, an improved version of the Sidewinder algorithm that significantly improves the creation of dynamic environments tailored to the specifics of the game. The study offers a new approach to the adaptation of traditional algorithms, providing more realistic conditions for evaluating AI strategies.

Monte Carlo Tree Search (MCTS) is one of the most promising algorithms for solving problems in simulation environments. M. Świechowski *et al.* (2023) emphasised the need to adapt MCTS to complex environments, stressing the importance of problem-based modification integration. In particular, the paper explored hybrid approaches for complex games with high branching and real-world applications in transport. W. Li *et al.* (2023) proposed a self-learning version of MCTS (SL-MCTS) that provides faster task completion and improved efficiency in path planning by using a neural network to optimise the search process. This modification significantly improves the algorithm's performance in time-constrained environments. I.F. Lövétei *et al.* (2021) demonstrated the feasibility of MCTS in real-time railway traffic control, where the algorithm proved capable of accommodating multi-factor constraints and producing optimal solutions in the shortest possible time.

Alpha-Beta Pruning was also studied by researchers. For instance, D. Permatasari *et al.* (2022) explored its application to improve NPC performance in the game Triple Triad, proving that this approach significantly increases the chances of winning in multiplayer environments. This confirms the decision-making efficiency of the algorithm in complex environments with many strategic factors. M. Mudda (2022) noted that alpha-beta pruning significantly optimises the calculation time in problems with many possible solutions, especially in two-player games. N. Sharma (2022) emphasised the limitations of Minimax and Alpha-Beta Pruning in cases where opposing agents do not act optimally, which requires adapting these algorithms to environments with uncertainty.

Equally important is the analysis of reinforcement learning models. Y. Cheng *et al.* (2020) proposed an MCTS-MTF algorithm to efficiently manage mixed traffic flows while considering speed and performance. This approach can be adapted to control agents in complex environments, demonstrating the prospects of using MCTS in real-time tasks. B. Wang *et al.* (2020) analysed how convolutional neural networks make decisions in conditions of high complexity using Ms. Pac-Man as an example. The study emphasised the importance of optimising solutions for high-reward tasks, which can be used to create complex simulation models.

However, despite significant progress in the research of these approaches, the adaptation of algorithms to environments with high uncertainty and complexity remains insufficiently studied. Previous studies demonstrate the benefits and comparisons in the Pac-Man game environment but focus on simple environments with few obstacles and limited interaction with other agents. At the same time, such environments do not fully reflect real-world scenarios that include high object density, complex navigation, and the need for rapid decision-making in changing environments.

The modified version of the Pac-Man game proposed in the current study is used as a simulation that models the tasks of unmanned aerial vehicles (UAVs) in two-dimensional environments. Mazes with varying wall densities and moving enemies simulate complex real-world conditions such as navigating through obstacles, optimising a route to multiple targets, and avoiding collisions with other agents. The absence of comparative studies of Expectimax, MCTS, and Alpha-Beta Pruning algorithms in such environments leaves open questions about their effectiveness in tasks that are close to real-world scenarios.

Thus, the present study seeked to fill these gaps by analysing the effectiveness of Expectimax, MCTS, and Alpha-Beta Pruning algorithms in complex environments that simulate real UAV missions. The study aimed to evaluate and compare their performance in achieving the main and additional goals, as well as to analyse their ability to adapt to variable factors. The use of a modified version of the Pac-Man game as a simulation describes in greater detail how these algorithms behave in conditions of increased complexity, which is crucial for their potential application in autonomous systems such as UAV control or other related tasks in dynamic environments.

Materials and Methods

The research material was the Pac-Man video game developed by Namco and released in 1980, modified to test algorithms in changed conditions. The experiment was conducted in the environment of this game to evaluate the effectiveness of three decision-making algorithms: Expectimax, Monte Carlo Tree Search (MCTS), and Alpha-Beta Pruning. Each of these algorithms controlled the behaviour of the main agent (Pac-Man), which had to perform the tasks of finding capsules, avoiding ghosts, and destroying ghosts during their vulnerability. The fourth agent, Random Agent, acted as a baseline for comparing results.

The study was conducted in three levels of complexity of a 50-by-50-cell maze, which differed in the number of walls (10%, 15%, and 20%) and the number of obstacles represented by ghosts. The purpose of introducing different levels of difficulty was to test the adaptability and

efficiency of each algorithm in conditions of different space constraints for manoeuvring. This approach was used to assess how different algorithms cope with the task in conditions where the restriction of freedom of movement is complicated by other factors.

Figure 1 shows the main and additional tasks of Pac-Man. The primary objective of Pac-Man was to collect all strategically important capsules on the map. Additional tasks:

- avoiding ghosts: ghosts act as active opponents chasing Pac-Man, trying to stop the actor;
- destroying frightened ghosts: after collecting the capsule, the ghosts become vulnerable for 10 turns, and Pac-Man can destroy them for extra points.

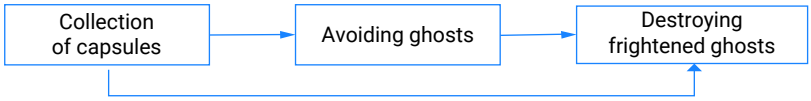

Figure 1. Main and additional tasks of Pac-Man
Source: compiled by the authors based on modified conditions of the Pac-man game

Each of the four agents played 100 games at each difficulty level, and the average score, game time, and winning percentage were recorded during each game. This was used to evaluate the effectiveness of each agent in fulfilling the main and additional goals of the game. Types of agents are shown in Figure 2.

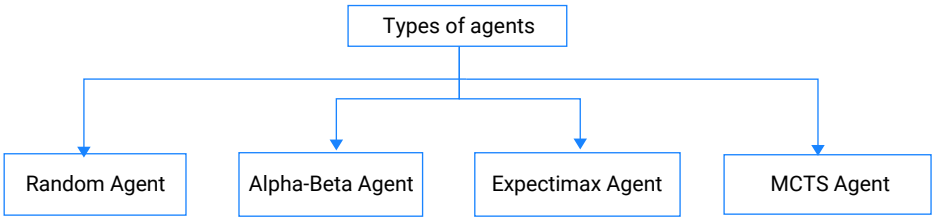

Figure 2. Types of agents used to compare their effectiveness in different Pac-Man game situations
Source: compiled by the authors based on the prepared modelling environment

Random Agent was used as a baseline to compare the results. This agent had no strategy and randomly chose actions during each turn. Its performance was expected to be the worst, as it was unable to respond effectively to game challenges such as ghosts or capsules. The Random Agent was used to evaluate how other agents with more complex algorithms improved the results compared to random decisions. This created a point of contrast to compare different approaches to decision-making in complex environments.

Other agents, **Expectimax**, **Alpha-Beta**, and **MCTS**, used different strategies to achieve the main and additional goals in the mazes. Comparing these agents with **Random Agent** was used to evaluate:

- **Expectimax Agent**: how this agent made decisions based on probable events, given the random behaviour of the ghosts;
- **Alpha-Beta Agent**: How this agent used an alpha-beta cut-off algorithm to reduce the search and reach optimal decisions in complex environments;
- **MCTS Agent**: how this agent used simulations to evaluate possible actions and choose the best strategy, adapting to increasing complexity.

A comparison of the results of these agents was used to determine which of the approaches is most effective in complex mazes with different levels of difficulty and limited resources.

Results and Discussion

For the experiments, it is necessary to examine in detail the four types of agents used to compare their performance in different Pac-Man game situations. Each of the four agents is characterised by its approach to decision-making, which ranges from simple random selection to complex algorithms based on simulations and predictions.

Random Agent is the simplest agent that randomly chooses its actions at each step of the game. It does not account for the positions of ghosts or the location of capsules or food. This agent has no algorithms for calculating possible outcomes and does not analyse the state of the game. Its actions depend entirely on a random choice from the available options at each turn. Despite its low efficiency, Random Agent is an essential benchmark for comparison, as it can be used to determine how much more complex algorithms improve performance. It is used as a baseline against which to compare the performance of other agents that use analytical decision-making approaches.

Alpha-Beta Agent is an agent that uses an alpha-beta cut-off algorithm to optimise the search for the best action. The algorithm is based on a minimax approach, where Pac-Man tries to maximise winnings, and ghosts minimise them by acting as opponents. Alpha-beta cutoff reduces the number of possible scenarios by cutting off those that do not affect the final result. This improves the performance of the agent, especially in situations where the

number of options is large, but not all of them are essential to the outcome. The agent assesses the state of the game, including the distance to ghosts and capsules, and makes decisions based on these indicators. The main advantage of Alpha-Beta Agent is its ability to significantly reduce the number of required calculations, which increases its efficiency in medium-complexity environments.

Expectimax Agent simulates the actions of Pac-Man and ghosts based on random events. This agent treats the actions of the ghosts as random and unpredictable. The Expectimax algorithm can be used to estimate the possible outcomes of each action, including the element of randomness in the ghosts' behaviour. This makes Expectimax Agent more flexible in environments where the behaviour of adversaries is changing or difficult to predict. This approach works well in complex environments with a high level of uncertainty, but in more structured environments where ghost behaviour can be predicted, Expectimax may be less effective. The agent attempts to exploit situations where ghosts are in a state of vulnerability after collecting capsules but may not be effective in avoiding ghosts in situations where their behaviour is less random.

MCTS Agent (Monte Carlo Tree Search) is an agent that uses a simulation approach to evaluate possible actions. MCTS performs numerous simulations of each Pac-Man action, attempting to predict the consequences of several moves ahead. Once the simulations are complete, the agent selects the action that has the highest potential for successful completion of the game. In complex mazes, MCTS can use modified heuristics to focus on important elements, such as capsules or ghosts, and plan actions more efficiently. The heuristics include additional criteria such as distance to the nearest capsule or proximity to ghosts, enabling more informed decision-making. This agent demonstrates the highest performance in complex environments, but its effectiveness is largely dependent on the number of simulations. The more simulations the MCTS Agent can run, the more accurate its predictions of the outcomes of actions will be, but increasing the number of simulations also requires more resources and computing time. Depending on the environment, as the number of simulations increases, the results may become worse starting from a certain threshold.

Ghosts in the game are primary opponents for Pac-Man, significantly complicating the task. Their function is not limited to chasing the main character but also includes creating a complex dynamic threat that requires Pac-Man to constantly adapt to changing conditions and make quick decisions. With the help of algorithms that control their behaviour, ghosts add variety and unpredictability to the game, creating situations where any inattention on the part of the player can lead to defeat.

The ghosts are guided by classic algorithms, which renders the behaviour predictable to a certain extent, but their interaction with each other and with Pac-Man complicates the situation. Each ghost has a unique algorithm, which provides diversity in their behaviour and makes them much more challenging as a collective.

The main strategy of the ghosts is to either directly chase Pac-Man or block possible escape routes, depending on the situation on the map. One ghost may move in the direction of the main character, trying to catch Pac-Man, while others may act in different scenarios, for example, one ghost may block paths, another may try to block exits or move to certain points to limit the primary actor's manoeuvrability. Each ghost has a unique behavioural algorithm that includes both active pursuit and strategic route blocking, substantially increasing efficiency in cooperative interaction. Thus, although each ghost acts according to different rules of the game, their collective influence creates an unpredictable and dynamic situation in which Pac-Man is forced to constantly adjust strategies to avoid danger. The types of ghosts are shown in Figure 3.

Figure 3. Ghosts in Pac-Man
Source: compiled by the authors based on Evillasio2 (2022)

Ghosts are guided by the following algorithms:

1. Blinky (Red Ghost): Blinky is the most aggressive ghost and always chases Pac-Man, trying to get as close as possible. Its algorithm involves moving directly to Pac-Man's current location. As Pac-Man collects capsules, Blinky can even accelerate, thus even more threatening. Constant pressure forces Pac-Man to make quick decisions on pathing to avoid being chased.

2. Pinky (Pink Ghost): Pinky acts more strategically, trying to block Pac-Man's path rather than engaging in direct pursuit. The Pinky algorithm involves moving to a point slightly ahead of Pac-Man's current direction. This forces Pac-Man to change route deliberately, as Pinky can easily block the way unless the player takes a different direction.

3. Inky (Blue Ghost): Inky has a more complex behaviour that depends on both Pac-Man's current location and Blinky's position. Its algorithm calculates the direction of movement, which depends on the relative position of the two objects. Because of this, Inky can be both unpredictable and dangerous, especially when Pac-Man is between him and Blinky.

4. Clyde (Orange Ghost): Clyde shows behaviour that changes depending on proximity to Pac-Man. When

Pac-Man is at a great distance, Clyde chases the player, similarly to Blinky. However, as Pac-Man approaches, Clyde suddenly changes direction and moves away. This unpredictable behaviour makes it difficult to predict and complicates Pac-Man route planning.

After Pac-Man eats the capsule, all the ghosts become "spooked" for 10 steps, according to the developed modification of the game. In this state, they start avoiding Pac-Man, while the latter can chase and destroy them for extra points. This phase of the game changes the dynamics, giving Pac-Man a temporary advantage, but requires quick decisions as the frightened ghosts soon return to their usual behaviour. Thus, each ghost in the Pac-Man game has a unique behavioural algorithm that affects the overall difficulty of the game. Interacting with ghosts requires Pac-Man to actively adapt to the situation on the map, which makes them an important element for evaluating the effectiveness of the algorithms used by Pac-Man agents.

To conduct the experiments, three types of mazes were created, which differ in the level of complexity determined by the number of walls. Each maze presented a different set of obstacles for Pac-Man and significantly affects the game strategy, as it makes it more difficult or easier to manoeuvre between the capsules and ghosts. The mazes also reflected real-life scenarios of confined space conditions that mimic situations faced by autonomous agents in real-world environments.

All mazes had the same dimension, with a total size of 50 by 50 playing cells. In each maze, regardless of the complexity, 4 capsules served as the main goals for Pac-Man. The capsules were located in different parts of the maze, which required Pac-Man to carefully plan path to collect them. In addition to the capsules, there were 4 ghosts in the maze that act as active opponents. The ghosts started the game in different parts of the maze, which was a simulation of real-life conditions and aims to balance the game's difficulty directly.

Description of labyrinths:

1. A maze with 10% walls (XLarge maze 1 is shown in Figure 4) – the simplest level of complexity. In this labyrinth, the walls occupy only 10% of the total area, which gives Pac-Man more freedom of movement. The minimum number of obstacles allows Pac-Man to dodge ghosts more effectively and reach the capsules faster. However, simpler mazes also give the ghosts more opportunities for direct pursuit, as Pac-Man cannot easily hide behind walls.

2. A maze with 15% walls (XLarge maze 2 is shown in Figure 5) is a medium difficulty level. As the number of walls increases to 15%, movement becomes more difficult, requiring Pac-Man to make more complex route decisions. More obstacles create more options to avoid the ghosts but also reduce the number of available paths to the capsules. In such conditions, Pac-Man must use the limited routes more efficiently, balancing between protection from ghosts and finding paths to the capsules.

3. The maze with 20% walls (XLarge maze 3 shown in Figure 6) is the most difficult level. In this maze, the walls occupy 20% of the total area, making Pac-Man's movement much more restricted. Pac-Man faces more obstacles on the way to the capsules, which requires more precise planning for each step. Ghosts in this environment become even more dangerous, as Pac-Man has fewer opportunities to evade pursuit, and the choice of safe routes is reduced.

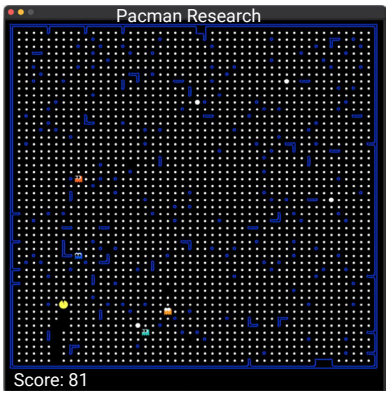

Figure 4. XLarge maze 1 in-game
Source: compiled by the authors in modelling and experimentation application

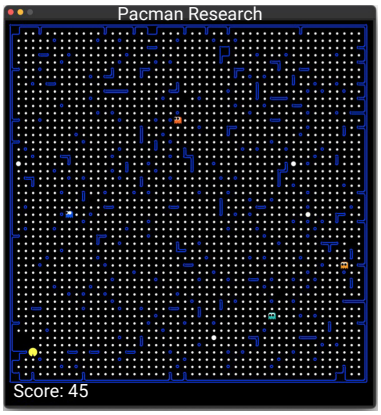

Figure 5. XLarge maze 2 in-game
Source: compiled by the authors in modelling and experimentation application

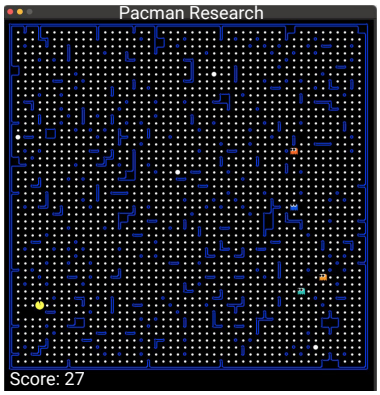

Figure 6. XLarge maze 3 in-game
Source: compiled by the authors in modelling and experimentation application

The experiments showed significant differences in the effectiveness of different agents when the complexity of the mazes changed. Each agent demonstrated unique behavioural properties, prompting certain assumptions regarding strategies and capabilities. An important criterion for evaluating agents was not only their ability to perform the main task of collecting all capsules but also to achieve secondary goals, such as destroying ghosts or collecting food. This was reflected in the number of points because even if Pac-Man did not achieve all the main goals, a high score indicated the agent's effectiveness in achieving additional goals.

Results of testing the effectiveness of agents in mazes

Mazes with fewer walls (10%) proved to be less challenging for the agents (Table 1). A sufficiently open space provided ample opportunities for agents to manoeuvre, while at the same time creating conditions for more aggressive pursuit by ghosts.

Table 1. Results for mazes with 10% of the walls

Agent	Average score	Game time	Percentage of winnings
Random Agent	120.17	29.14	0%
Alpha-Beta Agent	3,174.03	192.96	34%
Expectimax Agent	2,219.85	278.77	8%
MCTS Agent (50 simulations)	3,906.96	484.85	38%
MCTS Agent (101 simulations)	3,383.16	354.64	35%

Source: compiled by the author based on modelling and experiments in the study

The best results were achieved by **MCTS Agent**, which scored an average of 3,906.96 points and had a 38%-win rate with 50 simulations. Increasing the number of simulations to 101 slightly reduced the result (3,383.16 points and 35% of wins). This may indicate that in less complex environments, increasing the number of simulations is not always beneficial, as it can overload the path-making, which becomes less efficient due to excessive exploration of possible options.

Alpha-Beta Agent achieved a consistent result, scoring 3,174.03 points with a 34%-win rate, which indicates the ability of this agent to effectively reduce the search space using alpha-beta cutoff. This enabled the agent to achieve both primary and secondary goals in less complex environments. However, in more complex mazes, the Alpha-Beta Agent demonstrated limited adaptability due to its dependence on accurate estimates of search depth.

Expectimax Agent scored an average of 2,219.85 points and won only 8% of the games. This indicates its effectiveness in predictable scenarios, but its dependence on random events renders Expectimax less efficient than other agents. Nevertheless, its ability to account for all possible states gives it an advantage in environments with many available moves.

Random Agent performed the worst, scoring an average of 120.17 points and not winning a single game. This confirms that the lack of strategy and random choice of actions make it highly inefficient. This behaviour caused it to frequently fall into ghost traps due to rash movements.

In mazes with a medium number of walls (15%), all agents showed a decrease in performance (Table 2). In this maze, the agents were forced to use more complex strategies to evade the ghosts and reach the capsules.

Table 2. Results for mazes with 15% of the walls

Agent	Average score	Game time	Percentage of winnings
Random Agent	110.07	31.79	0%
Alpha-Beta Agent	2,602.32	698.16	12%
Expectimax Agent	2,183.97	254.42	6%
MCTS Agent (50 simulations)	3,361.54	290.2	19%
MCTS Agent (101 simulations)	3,215.14	599.82	25%

Source: compiled by the author based on modelling and experiments in the study

MCTS Agent continued to lead the leaderboard, scoring 3,361.54 points with 50 simulations and winning 19% of games. Increasing the number of simulations to 101 allowed the agent to increase its winning percentage to 25%. This indicates that in more complex environments, additional simulations improve adaptation. However, the agent requires significant computing resources to achieve high accuracy.

Alpha-Beta Agent achieved an average score of 2,602.32 points with a 12%-win rate, indicating that this agent loses efficiency as the maze complexity increases. Its ability to reduce the search space started to perform worse in conditions with more obstacles. In addition, the agent showed a limited ability to quickly adapt to the changed behaviour of the ghosts.

Expectimax Agent demonstrated even weaker results, scoring 2,183.97 points and winning only 6% of the games. This indicates the limitations of the Expectimax algorithm in more complex environments where the random actions of ghosts become less predictable. The agent often made risky decisions, which led to a loss of points.

Random Agent remained in last place, scoring 110.07 points and not winning a single game. This once again emphasises that randomly choosing actions in difficult conditions is ineffective. In more complex mazes, the agent was even more prone to falling into traps.

Mazes with the largest number of walls (20%) were the most difficult for all agents (Table 3). The agents encountered significant difficulties due to the limited space for evading ghosts and the difficulty of finding optimal routes.

Table 3. Results for mazes with 20% of the walls

Agent	Average score	Game time	Percentage of winnings
Random Agent	183.90	80.48	0%
Alpha-Beta Agent	1,672.59	200.09	2%
Expectimax Agent	1,685.06	253.68	0%
MCTS Agent (50 simulations)	2,042.03	460.64	1%
MCTS Agent (101 simulations)	2,221.03	298.61	2%

Source: compiled by the author based on modelling and experiments in the study

MCTS Agent with 50 simulations performed better than the other agents, winning one game out of 100 and scoring an average of 2,042.03 points. Increasing the number of simulations to 101 resulted in better performance, with the average score rising to 2,221.03 points and the win rate also increasing to 2%. This shows that even in the most challenging conditions, MCTS can benefit from additional simulations, improving its solutions in conditions of increased complexity.

Alpha-Beta Agent achieved an average score of 1,672.59 points with a 2%-win rate, demonstrating a significant decrease in performance in complex mazes with many obstacles. This indicates the limitations of the alpha-beta cutoff strategy, which performs well in simpler environments but struggles in environments with many obstacles. In a maze with 20% walls, this agent often had difficulty predicting effective routes due to the difficulty of dodging ghosts. The high level of obstacles required much more precision in calculations, which the agent failed to achieve.

Expectimax Agent scored an average of 1,685.06 points and did not win a single game, which confirms its limited effectiveness in conditions of high complexity and many random events. In the most complex maze, this agent was unable to adapt its decisions to dynamic changes, which reduced its effectiveness. Many obstacles and limited space significantly reduced Pac-Man's chances of avoiding traps, which led to quick losses.

Random Agent again demonstrated the worst results, scoring only 183.90 points and not winning a single game, which emphasises its complete inefficiency in such conditions. In a maze with 20% walls, the random actions of this agent led to frequent collisions with ghosts, as it could not choose safe routes. Such an environment fully exposed the weakness of this approach, as even minimal planning was absent.

The collected experimental results revealed several interesting aspects of agent performance in different maze complexity conditions. The best results in environments with many walls (20%) were shown by **MCTS Agent** with 101 simulations, which resulted in it winning 2 games out of 100 and scoring 2,221.03 points. This highlights the fact that increasing the number of simulations can improve the performance of an agent in high-complexity environments. However, increasing the number of simulations beyond 101 resulted in significant performance degradation due to a significant increase in the time required to calculate each move. This is especially relevant for complex environments, such as large mazes, where the speed of decision-making is critical. Since the limit on the number of simulations requires more productive computing resources, the current version of MCTS is not optimal for use on power-limited devices such as drones or autonomous robots without additional algorithm modifications.

The other agents also demonstrated high computational resource requirements when calculating the next step, which, as in the case of MCTS, shows a significant increase in the time required to calculate each move. This is especially notable for the Alpha-Beta Agent, which loses efficiency as the maze complexity increases. Therefore, although the Alpha-Beta Pruning algorithm works well in medium-complexity environments, its application in complex environments requires adaptation.

Alpha-Beta Pruning is an improvement of the Minimax algorithm that reduces the number of tested options by skipping some branches of the decision tree that cannot affect the result. This can significantly speed up the search process, but as the complexity of the maze increases (for example, with more possible states or a more complex map structure), the number of required checks increases, which leads to a decrease in the efficiency of the algorithm. Thus, to use this algorithm in complex environments, additional optimisations are usually required or a transition to other methods more adapted to complex conditions.

Lastly, the **Random Agent**, although it had minimal computational requirements, demonstrated its complete inefficiency in all experimental conditions. This agent chooses its actions randomly, without analysing the current state of the game, which can be used only for comparison with other, more complex agents. The low

performance of the Random Agent is used as a benchmark to assess the level of complexity of both the game environment and the tasks faced by other agents. Therefore, Random Agent emphasised the multifactorial nature of the game environment and the scale of the challenges faced by agents in the process of making decisions and implementing strategies. The performance of other agents, such as MCTS (Monte Carlo Tree Search) and Alpha-Beta Pruning, contrasted significantly with the results of Random Agent and demonstrates significantly better performance in the environments shown.

The experimental results showed that MCTS Agent is highly efficient compared to other algorithms, especially in complex environments such as mazes with 20% walls. This trend correlates with the findings of H. Maddipati *et al.* (2020), which emphasised the flexibility of MCTS in dynamic environments. MCTS strikes a balance between research and operation, allowing it to adapt to changing conditions. Similar results were also obtained by N. Pepels *et al.* (2014), demonstrating that MCTS can achieve high performance even in real-time by applying variable tree depth strategies and search tree reuse.

In the present study, MCTS performed best under conditions of increased complexity due to its ability to run numerous simulations to analyse possible options. This was supported by the findings of S. Samothrakis *et al.* (2011), who emphasised that MCTS provides better adaptation in games without a clear end state, such as Ms. Pac-Man. However, as noted by D. Busatto-Gaston *et al.* (2020), the performance of MCTS can be further improved by integrating symbolic cues, which helps to optimise action search and selection.

At the same time, Expectimax Agent demonstrated significantly lower performance compared to MCTS, especially in complex mazes. This is partially consistent with the findings of P.S. Shevtekar *et al.* (2022), who described the limitations of Minimax, particularly its sensitivity to the depth of the search tree. Since Expectimax is a modification of Minimax that accounts for random events, its performance in dynamic environments is significantly reduced. As noted by Y. Zou (2021), Minimax provides optimal results in deterministic environments, while Expectimax adapts better to environments with random events. However, in environments of higher complexity, such as mazes with 20% walls, Expectimax's dependence on randomness and limited ability to predict actions make it less effective, and its performance decreases significantly with increasing maze complexity.

Alpha-Beta Agent, while demonstrating consistent results in medium-complexity environments, loses effectiveness in complex mazes. As noted by S.P. Singhal & M. Sridevi (2019), Alpha-Beta Pruning is optimal for reducing computational costs in deterministic environments, but its performance decreases in large search spaces. Comparison with data by P. Mishra *et al.* (2018) confirmed that search optimisation is crucial for the effective use of Alpha-Beta in complex environments. In addition, the use of parallel architectures proposed by P.S. Shevtekar *et al.* (2022) can reduce these limitations.

The experiments also confirmed the findings of L.M.P. Guerreiro (2021), who emphasised the prospects of using MCTS as an effective agent for creating a dataset and then applying it to training neural networks for real-time tasks. Although the current study did not use combined approaches, such as training neural networks with MCTS, it both confirmed its effectiveness compared to other agents and highlighted weaknesses that require improvement and further research. The modified versions of MCTS described in X. Liu *et al.* (2009), demonstrate similar efficiency in adapting to dynamic conditions by integrating machine learning methods, in particular artificial neural networks. These methods improve convergence speed and optimisation, which may be a further promising direction in complex dynamic environments and the context of UAV missions or related tasks involving robots.

An important conclusion of the study is the confirmation of the high computational requirements of algorithms such as A*, which, as noted by N. Salem *et al.* (2024), demonstrate excellent performance in simple environments, but their application in environments with increased complexity is problematic due to the significant computational cost. An attempt to integrate A* revealed that the algorithm cannot provide stable performance in large mazes due to delays in calculations.

Analysis of the behaviour of agents in the proposed modification of the Pac-Man game demonstrated how different strategies and algorithms can be applied to solve complex real-world problems, such as controlling drones, robotic systems or other moving objects to perform tasks on the ground. This also demonstrated the importance of adapting algorithms to changing conditions and multiple factors that are common in many real-world scenarios, including planning, resource management, automation, or logistics. The effective use of strategic agents in a gaming context not only contributes to the development of artificial intelligence technologies but also provides a valuable tool for solving practical problems where agents must adapt to complex and unpredictable conditions, including limited resources and a changing environment.

The results of the study confirmed that MCTS is the most promising method for complex dynamic environments due to its flexibility and ability to balance research and operation. However, to be used effectively in real-world applications such as UAV control, further improvements are needed to reduce computational costs and improve adaptability. Alpha-Beta Pruning can be used in tasks where it is important to reduce the search space, such as in scenarios with predictable obstacles or a limited set of possible actions, with the potential for possible improvement through additional research.

The potential applications of the investigated algorithms in real-world scenarios, such as UAV missions, include automatic route planning, reconnaissance, collision avoidance, and resource management in energy-limited

environments through high-quality route planning. For instance, MCTS can be used to quickly adapt to changes in the environment, such as the appearance of new obstacles or targets, thanks to its ability to conduct simulations and consider multiple scenarios. This is particularly relevant for missions in complex environments, such as densely populated areas or search and rescue operations, where fast and accurate decision-making is critical.

Thus, the results of the study not only emphasised the potential of the presented algorithms to perform complex tasks in dynamic environments but also demonstrated the need for further improvement to increase efficiency in real-world applications, such as UAV missions and other automated control systems.

Conclusions

The study conducted a series of experiments in three types of mazes with different levels of complexity, which varied in the number of walls and size. Each maze modelled realistic navigation conditions, in particular, simulated situations requiring route variability and the need to evade competitors (in the form of ghosts) chasing Pac-Man. The data on the average score, game duration, and winning percentage were collected, which was used to analyse in-depth the features of each algorithm in specific conditions. The study determined that the MCTS algorithm was most effective in mazes of lower complexity, as its ability to simulate allows for a quick assessment of potential options and the selection of the most effective paths. At the same time, as the complexity of the maze increased, the effectiveness of MCTS decreased, as the increase in the number of possible options complicated the decision-making process, requiring more resources. This shows the importance of further optimising this algorithm to adapt to complex scenarios. Alpha-Beta Pruning performed reasonably well in simpler environments, but its effectiveness gradually decreased with the increasing complexity of the environment, similar to MCTS. The limitations of Alpha-Beta Pruning in such situations may be related to the need to expand its ability to quickly analyse alternatives.

Expectimax performed consistently poorly compared to the other algorithms, especially in more complex environments, which is possibly caused by reliance on random events that limited its effectiveness in complex scenarios where rapid adaptation to conditions is crucial. This algorithm was the least effective for simple mazes, but its performance dropped sharply in environments requiring flexibility and evasion strategies. Thus, the data obtained concluded that it is important to choose an appropriate algorithm depending on the level of task complexity.

An important aspect of this research was that its results can be used to further develop navigation algorithms for autonomous systems, such as drones or robotic platforms, operating in complex dynamic environments. The use of algorithms with the ability to rapidly adapt and optimise solutions under conditions of limited resources and high levels of uncertainty is particularly relevant for such scenarios. MCTS has demonstrated the potential for use in autonomous navigation systems, where adaptation to unpredictable environmental changes is critical.

Further research could address the integration of machine learning techniques to improve the MCTS Agent, which has demonstrated the best adaptability in complex environments. Combining MCTS with neural networks can provide more efficient state estimation and reduce the need for many simulations, thus lowering computational costs. In the future, the integration of machine learning into MCTS can strike a balance between the speed and accuracy of decision-making required to deal with dynamic environments with many variables. Such improvements can render MCTS more efficient and suitable for use in tasks close to real-world scenarios, such as automating the navigation of drones, autonomous robots, or other robotic systems in dynamic environments.

Acknowledgements

None.

Conflict of Interest

The authors declare no conflict of interest.

References

[1] Busatto-Gaston, D., Chakraborty, D., & Raskin, J.-F. (2020). Monte Carlo tree search guided by symbolic advice for MDPs. In *31st international conference on concurrency theory (CONCUR 2020)*. *Leibniz international proceedings in informatics (LIPIcs)* (Vol. 171, pp. 40:1-40:24). Schloss Dagstuhl: Leibniz-Zentrum für Informatik. doi: 10.4230/LIPIcs.CONCUR.2020.40.

[2] Cheng, K. M., Liu, H., & Dou, X. (2024). Randomized Pacman maze generation algorithm. *Applied and Computational Engineering*, 42, 156-162. doi: 10.54254/2755-2721/42/20230771.

[3] Cheng, Y., Hu, X., Tang, Q., Qi, H., & Yang, H. (2020). Monte Carlo Tree search-based mixed traffic flow control algorithm for arterial intersections. *Transportation Research Record*, 2674(8), 167-178. doi: 10.1177/0361198120919746.

[4] Evillasio2. (2022). *The Pac-Man Ghosts Alt*. *DeviantArt*. Retrieved from https://www.deviantart.com/evilasio2/art/The-Pac-Man-Ghosts-Alt-916669643.

[5] Guerreiro, J.M.P. (2021). *Learning agent in the Ms. Pac-Man vs Ghosts game*. (Master's Thesis, Instituto Superior Técnico, Lisboa, Portugal).

[6] Li, W., Liu, Y., Ma, Y., Xu, K., Qiu, J., & Gan, Z. (2023). A self-learning Monte Carlo tree search algorithm for robot path planning. *Frontiers in Neurorobotics*, 17. doi: 10.3389/fnbot.2023.1039644.

[7] Liu, X., Li, Y., He, S., Fu, Y., Yang, J., Ji, D., & Chen, Y. (2009). To create intelligent adaptive game opponent by using Monte-Carlo for the game of Pac-Man. In *Fifth international conference on natural computation* (pp. 598-602). Tianjian: IEEE. doi: 10.1109/ICNC.2009.633.

[8] Lövétei, I. F., Kővári, B., & Bécsi, T. (2021). MCTS based approach for solving real-time railway rescheduling problem. *Periodica Polytechnica Transportation Engineering*, 49(3), 283-291. doi: 10.3311/PPtr.18584.

[9] Maddipati, H., Kundurthi, A., Raaj, P., Srilatha, K., & Surapaneni, R. (2020). Artificial Intelligence based Pacman Game. *International Journal of Innovative Technology and Exploring Engineering*, 9, 140-144. doi: 10.35940/ijitee.I6975.079920.

[10] Mishra, P., Patel, V., Mittal, P., & Patni, J.C. (2018). Algorithm analysis tool based on execution time-input instance-based runtime performance benchmarking. In *International conference on recent developments in science, technology, humanities and management – 2017* (pp. 27-30). Kuala Lumpur: SRD.

[11] Mudda, M. (2022). Tic Tac Toe by Minimax Alpha-Beta Pruning using Arduino. *International Journal for Research in Applied Science and Engineering Technology,* 10(2), 157-165. doi: 10.22214/ijraset.2022.40115.

[12] Pepels, T., Winands, M. H. M., & Lanctot, M. (2014). Real-time Monte Carlo Tree search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3), 245-257. doi: 10.1109/TCIAIG.2013.2291577.

[13] Permatasari, B.D., Haryanto, H., Zuni Astuti, E., & Dolphina, E. (2022). Peningkatan kemenangan non-playable character dalam permainan triple triad Menggunakan Alpha-Beta Pruning. *Jurnal Komputasi*, 10(1). doi: 10.23960/komputasi.v10i1.2952.

[14] Ramadhan, A. W. R., & Udjulawa, D. (2020). Perbandingan algoritma dijkstra dan algoritma a star pada permainan Pac-Man. *Jurnal Algoritme*, 1(1), 12-20. doi: 10.35957/algoritme.v1i1.411.

[15] Salem, N., Haneya, H., Balbaid, H., & Asrar, M. (2024). Exploring the maze: A comparative study of path finding algorithms for PAC-Man game. In *21st learning and technology conference (L&T)* (pp. 92-97). Jeddah: IEEE. doi: 10.1109/LT60077.2024.10469459.

[16] Samothrakis, S., Robles, D., & Lucas, S. (2011). Fast approximate Max-n Monte Carlo Tree search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2), 142-154. doi: 10.1109/TCIAIG.2011.2144597.

[17] Sharma, N. (2022). *Introduction to artificial intelligence.* Retrieved from https://inst.eecs.berkeley.edu/~cs188/fa22/assets/notes/cs188-fa22-note06.pdf.

[18] Shevtekar, P.S., Malpe, M. & Bhaila, M. (2022). Analysis of game tree search algorithms using Minimax Algorithm and Alpha-Beta Pruning. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, 8(6), 328-333. doi: 10.32628/CSEIT1228644.

[19] Singhal, S.P., & Sridevi, M. (2019). Comparative study of performance of parallel alpha Beta Pruning for different architectures. In *2019 IEEE 9th international conference on advanced computing (IACC)* (pp. 115-119). Tiruchirappalli: IEEE. doi: 10.1109/IACC48062.2019.8971591.

[20] Świechowski, M., Godlewski, K., Sawicki, B., & Mańdziuk, J. (2023). Monte Carlo Tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56, 2497-2562. doi: 10.1007/s10462-022-10228-y.

[21] Wang, B., Ma, R., Kuang, J., & Zhang, Y. (2020). How decisions are made in brains: Unpack “black box” of CNN with Ms. Pac-Man video game. *IEEE Access*, 8, 142446-142458. doi: 10.1109/ACCESS.2020.3013645.

[22] Zou, Y. (2021). General Pacman AI: Game agent with tree search, adversarial search and model-based RL algorithms. In *2nd International conference on big data & artificial intelligence & software engineering (ICBASE)* (pp. 253-260). Zhuhai: IEEE. doi: 10.1109/ICBASE53849.2021.00053.

Аналіз ефективності алгоритмів прийняття рішень в умовах складних ігрових середовищ на прикладі Pac-Man

Артем Новіков

Аспірант
Харківський національний університет ім. В.Н. Каразіна
61022, Майдан Свободи, 4, м. Харків, Україна
https://orcid.org/0009-0004-5914-7098

Володимир Яновський

Доктор фізико-математичних наук, професор
«Інститут монокристалів» Національної Академії наук України
61072, пр-т Науки, 60, м. Харків, Україна
https://orcid.org/0000-0003-0461-749X

Анотація. Ігрові симуляції типу Pac-Man є важливим інструментом для тестування алгоритмів прийняття рішень в умовах, що імітують реальні сценарії. Це відкриває нові можливості для розробки автономних систем, які можуть адаптуватися до мінливих умов навколишнього середовища та взаємодіяти з іншими агентами. Метою цієї роботи було порівняння алгоритмів Expectimax, Monte Carlo Tree Search та Alpha-Beta Pruning у змінених умовах гри Pac-Man для визначення найбільш ефективного підходу до прийняття рішень у складних середовищах. Для цього було використано імітаційне моделювання для оцінки ефективності роботи агентів у різних ігрових лабіринтах, що відрізняються за складністю. У дослідженні вимірювалися такі показники, як кількість балів, час гри та відсоток виграшів, що дозволило оцінити ефективність алгоритмів у різних ситуаціях. Аналіз проведених експериментів продемонстрував, що алгоритм Монте-Карло є найбільш ефективним серед протестованих методів для вирішення менш складних лабіринтів, підтверджуючи його здатність швидко знаходити оптимальні шляхи в простих умовах. Алгоритм Alpha-Beta Pruning показав меншу ефективність, що вказує на необхідність його оптимізації для роботи в більш складних середовищах. Expectimax продемонстрував значно нижчу продуктивність, що свідчить про його обмежену придатність для складних ігрових лабіринтів. Дослідження показало, що збільшення складності лабіринтів значно знижує продуктивність усіх алгоритмів, особливо при більшій кількості перешкод, підкреслюючи важливість розробки більш стійких методів для роботи у високоскладних умовах. Оптимізація алгоритмів Монте-Карло та Alpha-Beta Pruning для роботи в складних середовищах може суттєво покращити їх результативність та зробити їх ефективними для реальних застосувань у навігації та керуванні рухомими пристроями. Результати цього дослідження можуть бути використані для розробки ефективних алгоритмів навігації для автономних транспортних засобів, дронів та інших робототехнічних систем, де адаптація до змін у складних умовах є критично важливою

Ключові слова: симуляційні середовища; Expectimax; MCTS; Alpha-Beta Pruning; автономна навігація