https://doi.org/10.15407/csc.2024.04.019
UDC 004.94

O.A. NOVIKOV, PhD student, Institute of Computer Science and Artificial Intelligence Department of Mathematical Modeling and Data Analysis,
Karazin Kharkiv National University,
Svobody, Sq 4, Kharkiv, Ukraine, 61022,
ORCID: https://orcid.org/0009-0004-5914-7098,
artem.slick@gmail.com

V.V. YANOVSKY, Doctor of Physical and Mathematical Sciences,
Professor of Department of Artificial Intelligence Systems,
"Institute for Single Crystals" of National Academy of Sciences,
Nauky ave. 60, Kharkiv, 61001, Ukraine,
ORCID: https://orcid.org/0000-0003-0461-749X,
Scopus Author ID 7003273794,
yanovsky@isc.kharkov.ua

ANALYSIS OF SEARCH AND MULTI-AGENT ALGORITHMS IN THE PAC-MAN GAME

This paper examines the performance of search and multi-agent algorithms within the context of the Pac-Man game. The game is used as a platform to simulate autonomous system management tasks, where an agent must complete missions in a two-dimensional space while avoiding dynamic obstacles. Classical search algorithms such as A and BFS, along with multi-agent approaches like Alpha-Beta, Expectimax, and Monte Carlo Tree Search (MCTS), are analyzed in terms of their effectiveness under different maze complexities and game conditions. The study explores how maze size, ghost behaviors, and environmental dynamics influence the performance of each algorithm, particularly in terms of execution time, score, and win percentage.*

Keywords: *search algorithms, multi-agent algorithms, Pac-Man, path optimization, Alpha-Beta, Expectimax, MCTS.*

Introduction

The Pac-Man game can be viewed as a simplified simulation of autonomous system management tasks, where agents must perform certain tasks in a two-dimensional space while facing moving obstacles. In this context, the game's maze can be interpreted as a 2D terrain model, where autonomous systems such as drones or ground robots need to optimize their paths, particularly in constrained spaces with obstacles. The game Ms. Pac-Man is an excellent test problem in pursuit-evasion with multiple active opponents, who adapt their chasing strategies based on the state and decisions of Ms. Pac-Man [1]. The scientific interest in this topic is growing, and numerous research directions are actively evolving, including work in the fields of robotics, biology, sociology, and psychology. The field of computational intelligence is particularly active, in part due to popular

Cite: Novikov O.A., Yanovsky V.V. Analysis of Search and Multi-Agent Algorithms in the Pac-Man Game. *Control Systems and Computers*, 2024, 4, 19—33. https://doi.org/10.15407/csc.2024.04.019

academic game competitions involving Ms. Pac-Man [2].

Pac-Man acts as an autonomous agent whose goal is to complete critical tasks-collecting capsules, which can be compared to real-life operations such as drones collecting data or delivering resources to specific points. Capsules can be seen as goals located in the environment that require specific actions such as data collection, servicing, or monitoring certain areas. The agent is defined as a simple finite automaton with a set of rules that regulate the likelihood of the agent's movement, taking into account maze constraints at any given moment [3]. By computing paths in real time, the algorithm can quickly adapt to unexpected opponent movements or dynamic environments [4].

Ghosts, in turn, represent dynamic threats or hostile agents, such as other drones or moving objects, that may interfere with completing the mission. Interaction with such obstacles requires the autonomous agent to constantly evade and adjust its path in real time. In real-world scenarios, this could include algorithm adaptations to avoid interception or collision with hostile or other moving objects. In a study by Tuchník et al. (2018), an experimental comparison was conducted based on a model where agents represent economic entities and can engage in interactions. The model was scaled to different levels of complexity, considering the number of agents and transport nodes. The results indicated significant differences in path planning task completion times depending on the system's complexity levels and model sizes [5].

Thus, the Pac-Man game provides an excellent platform for modeling and testing algorithms that can be applied to real-world autonomous systems operating in complex environments with dynamically changing obstacles. Autonomous agents' interaction with obstacles and the completion of critical tasks require algorithms to optimize paths and adapt to environmental changes, directly corresponding to real-world challenges in robotics and autonomous systems. For example, Said et al. (2021) discussed that related methods treat agents as a high-dimensional complex system and use complete optimal planners (such as A* and its variants). However, the naive approach quickly becomes impractical due to exponential complexity [6]. Recently, specific suboptimal algorithms with polynomial complexity have been proposed for computing actual paths in important sub-tasks of the general problem. These algorithms use single-agent primitives but return sequential paths where only one agent moves at a time [6]. This demonstrates potential applications in large multi-robot systems [7].

In the context of autonomous systems, it is important to consider that in games such as Warcraft III, agent routes may intersect if the agents' plans do not interfere with each other. This means that agents may occupy the same space at different moments in time, but not simultaneously[8]. However, in tasks involving environment coverage or observation, there may be a requirement that agent routes do not overlap, which allows covering more territory in less time, also considering the limited power provided by robot batteries [8].

In scenarios such as emergency rescue, there may be a need for certain areas of the terrain to be checked by specific agents, as the likelihood of people being in those areas is high. To prevent one agent from doing all the work, restrictions may be placed on the route length or plan duration for each agent [8].

Classical Search Algorithms

In classical pathfinding approaches, such as A* and BFS, the goal is to find the shortest path to capsules or other objects in the maze. The A* algorithm uses a heuristic to estimate the distance to the target, which helps optimize search time by reducing the number of nodes considered. This approach is particularly effective in static environments, such as maze walls, but does not account for dynamic threats like ghosts. On the other hand, BFS offers a different approach by ensuring a comprehensive search of all possible paths in the maze. However, its downside is that it can be less efficient in large mazes due to the high search cost. While classical search algorithms are effective for static environments, they lose their efficiency in dynamic environments like the Pac-Man game, where ghosts move and create changing threats. As Hevavasam et al. (2022) point out, many algorithms lack a proper method to adapt to the speeds of moving objects during navigation decision-making [9]. This is especially evident in large and complex mazes, where classical algorithms often lead to locally optimal but not globally efficient solutions, as they do not account for the variable behavior of ghosts.

The study by Lu et al. (2011) proposes an extension to the basic algorithm with the effective use of a multi-dimensional environment representation, allowing quick localization of changed edges and updating the priority queue [10]. This improves both resilience and computational complexity in the worst case compared to classical algorithms.

Multi-Agent Strategies

To solve tasks in dynamic environments, multi-agent algorithms are used, which can model interactions with other agents, such as ghosts. The Expectimax algorithm allows modeling probabilistic actions of ghosts, enabling Pac-Man to predict possible threats and avoid them. This is particularly useful in complex environments where ghosts have unpredictable behavior, such as after collecting a capsule when they become vulnerable.

However, it should be noted that modeling the possible actions of all ghosts within Expectimax can significantly increase computational complexity, especially in large mazes, limiting the application of this algorithm in real-world scenarios. To address this issue, Alpha-Beta Pruning is used, which optimizes the search by cutting off unnecessary branches of the move tree. As Saffidine et al. (2012) note, this approach uses MiniMax values to prune a subtree when there is confidence that the move will not affect the decision at the root node, allowing Pac-Man to make quick decisions by reducing the number of considered options [11]. The study also notes that after its discovery, the Alpha-Beta method was extended to other types of games and search algorithms, indicating its widespread application in various contexts [11].

Additionally, attention should be paid to algorithms that use simulation approaches, such as Monte Carlo Tree Search (MCTS). This method allows Pac-Man to explore possible actions through simulation of future states. MCTS is an algorithm that finds “optimal” actions by random movements in the action space and building them into a tree structure [12]. The MCTS algorithm strikes a balance between exploring new possibilities and exploiting already known successful strategies. As Lee (2019) notes, “exploitation” involves finding the best solution based on already learned examples, while “exploration” refers to trying a new approach that was previously unknown in the space of learned examples [12].

The success of MCTS depends on balancing exploitation (focusing on promising areas) and exploration (focusing on areas that have not been well studied yet) [13]. The most popular algorithm in the MCTS family, which addresses this dilemma, is Upper Confidence Bound for Trees (UCT), which combines exploitation and exploration to achieve optimal decisions [14].

With each new simulation, the algorithm gradually improves its decisions, making it especially useful in large mazes and complex environments. As Kuipers et al. (2013) noted, when the tree size is small, preference should be given to exploitation, whereas in large trees, it is more appropriate to conduct more exploration [14].

The Importance of Different Approaches

Each of the mentioned approaches has its advantages and disadvantages depending on the game conditions and the maze’s complexity. A* and BFS work well in static environments with predictable obstacles, but when facing dynamic ghosts, they often lack flexibility. Expectimax and AlphaBeta are more adaptive as they can model ghost behavior, but they become less efficient when there are many agents or in large mazes due to limited computational resources. As Felner et al. (2021) note, the classical A* algorithm can be applied to Multi-Agent Path Finding (MAPF) by using a state space representing the location of all agents, but it suffers from exponential growth in the size of the state space and branching factor as the number of agents increases. Multi-agent approaches can provide exponential speedups compared to A* by decomposing the task into independent sub-tasks [15].

MCTS shows the best results in large and complex environments, as it can balance exploring new options and exploiting acquired knowledge through simulations. This algorithm is particularly useful in situations with a large number of possible actions, where other algorithms may suffer significant efficiency losses due to the expanding search space. As Kandelwal et al. (2016) note, over the past decade, Monte Carlo Tree Search (MCTS), and particularly Upper Confidence Bound in Trees (UCT), have proven to be quite effective in large probabilistic planning domains. Research into various backpropagation strategies in MCTS, beyond simple Monte Carlo averaging, can lead to significantly better results in large and complex probabilistic planning tasks [16]. Additionally, Ou et al. (2023) note that MCTS is a cutting-edge algorithm suitable for decision-making in complex environments with adversaries. They introduce important sampling in MCTS to make the search more efficient in ultra-large search spaces [17].

Thus, different algorithms can be used for different types of tasks in the Pac-Man game. The optimal solution depends on the maze structure, the number of ghosts, and the dynamics of their behavior. Using a combination of classical search algorithms and multi-agent strategies allows adaptation to various conditions and ensures success in achieving the main goal. However, as Silva et al. (2018) note, the concept of hyper-heuristics remains under-researched in the analyzed frameworks, and there is a lack of tools that offer optimization process support, such as statistical analysis, parameter self-optimization, and graphical interfaces [18].

Problem Setting

In this version of the Pac-Man game, the primary objective remains unchanged — to collect all capsules on the map while avoiding ghosts. However, several new conditions have been introduced to complicate the game, making it more interesting for testing various decision-making algorithms. These conditions can also be easily interpreted in real-world scenarios, such as drone or autonomous robot management in complex environments.

The main task of Pac-Man (the agent) is to collect capsules, which can be considered as goals or tasks in real-world environments, such as scouting an area or collecting data using drones. Capsules act as critical objects that need to be located and collected. In this context, the maze becomes a two-dimensional terrain model where the agent must optimize its path to complete the task while avoiding collisions with obstacles or other agents.

To more accurately model real situations, additional conditions were introduced that simulate challenges in complex environments. Here are the main game modifications:

Penalties and Rewards:

1. Penalty for inactivity. In real-world tasks like reconnaissance using drones, time is a critical resource. Inactivity or delays in task execution can lead to loss of efficiency or even mission failure. Similar to how Pac-Man loses points for each extra move, real-world systems require time and energy optimization for success.

2. Reward for collecting capsules. Capsules can be treated as critical data collection points or important objects that need to be identified and processed. For example, drones can be programmed to gather information from specific points on the map or perform certain actions in high-concentration object areas.

3. Reward for defeating a scared ghost. A scared ghost in the context of real systems can mean the temporary vulnerability of another agent or system. For instance, this could be when a moving object (another drone or vehicle) becomes vulnerable due to technical problems or limited power, allowing the main agent to take advantage of the moment to neutralize or complete the task. This situation may arise, for example, when one drone temporarily loses connection or power, creating an opportunity for other agents to act quickly.

Ghost Mechanics and Scared States:

1. Ghosts as mobile obstacles: Ghosts in the Pac-Man game can be seen as moving obstacles or enemies in a real-world environment. These could be other autonomous systems (drones, ground robots) or even aggressive elements, such as enemy drones in military scenarios. Their aggressive behavior, which involves chasing Pac-Man, resembles dynamic threats that must be considered when planning routes and avoiding them.

2. Scared state. The vulnerability of ghosts during their scared state can be interpreted as a temporary weakness in a system or agent that can be exploited to achieve a goal. For example, this could relate to moments when other drones or vehicles become temporarily inoperative due to technical problems, allowing the main agent (Pac-Man) to act more aggressively to complete the mission.

In the classic version of Pac-Man, the player simply collects food and avoids ghosts, but in the modified version, additional elements have been added that relate to real-world autonomous system management tasks. The loss of points for each unnecessary move simulates real situations where time and resources are limited. In systems like drones or autonomous robots, every move has its cost — either in terms of energy or time. Inefficient route planning can lead to energy waste or mission failure within the planned timeframe. For example, an autonomous drone conducting monitoring or delivery must constantly optimize its route, as every unnecessary move reduces battery life, which can negatively impact task completion. The same situation occurs in Pac-Man: each inefficient move, for which points are deducted, brings the player closer to defeat, making the search for optimal routes critically important.

Capsules are key objects that must be collected for success. In real scenarios, capsules can be compared to critical information collection points, important data, or areas where certain actions must be performed, such as terrain scanning, delivery, or evacuation. For example, an autonomous drone may have a task to collect data from several points or deliver goods to several predetermined locations. Choosing the right route and task execution order directly affects the overall system efficiency. The priority of capsules in the game reflects a similar situation, where selecting the correct sequence of actions is key to success.

Ghosts in the game act as mobile obstacles or hostile agents that should be avoided, similar to real-world autonomous systems dealing with dynamic threats, such as other drones, vehicles, or other moving objects. The scared state of ghosts in the game can be compared to the temporary vulnerability of hostile agents in real-world scenarios. For example, drones performing missions may encounter other autonomous systems that become temporarily vulnerable (due to technical failures, loss of communication, or other factors). During this period, autonomous systems can not only avoid threats but also effectively neutralize them or use the moment to achieve additional goals, similar to how Pac-Man takes advantage of the scared state of ghosts to score extra points.

Just like in real autonomous systems, Pac-Man collects food in addition to the main capsules, which brings smaller but consistent points. This can be compared to secondary tasks in real autonomous missions, where the system performs additional operations that yield smaller yet important results, such as collecting secondary data or performing simple auxiliary actions that contribute to the overall success of the mission.

The primary goal of this research is to study the effectiveness of various agents in the context of autonomous system management tasks, simulated in the Pac-Man game. To achieve this, we plan to compare the performance of the aforementioned search and multi-agent algorithms in this environment using different versions of 2D terrain models, represented in the game by mazes. We aim to explore how these algorithms perform under various conditions: from simple and less dynamic environments, where path optimization plays a significant role, to dynamic environments with more agents, where interaction with these agents (ghosts) critically affects decision-making.

Key questions we aim to answer:

1. Which algorithms are most effective for completing tasks under different levels of environmental complexity?

2. How do maze complexity and additional conditions (number of enemy agents, terrain size, and number of obstacles) impact the performance of different agents?

3. How effective are search algorithms compared to multi-agent approaches in the context of tasks with varying levels of complexity and dynamism?

To achieve this goal, a series of experiments in mazes of different sizes and complexity will be conducted to compare metrics such as execution time, score achieved, and win percentage for each agent. This will help identify usage patterns for the algorithms and determine which approach is most effective depending on the complexity of the environment and the specific conditions of the task.

Algorithms

AStarSearch (A)* is an efficient pathfinding method that combines actual path cost and heuristics to minimize the distance to the target. In the Pac-Man game, this algorithm helps to collect all capsules by accounting for the maze structure and optimizing the route.

Key components:

1. Priority Queue (fringe). The algorithm uses a priority queue to organize states, allowing it to find optimal solutions with minimal costs. In real autonomous systems, such as drones, this approach is used for route planning while considering limited resources (energy, time).

2. State. The state consists of Pac-Man's position and the list of capsules that remain to be collected. In real systems, this could correspond to the position of a drone and the goals it must reach or process.

3. Heuristic. A* uses heuristics to estimate the distance to the nearest capsule, allowing the algorithm to prioritize closer targets. This is similar to drones using distance estimations to critical points to minimize time and energy.

4. Closed List. The algorithm remembers all visited states, preventing unnecessary returns to them. In real autonomous systems, this reduces resource costs for reprocessing the same points or tasks.

A* is one of the main algorithms for route planning in autonomous systems. For example, drones use it to find the shortest path through complex environments with obstacles and changing conditions. The algorithm helps achieve a balance between speed and resource optimization, which is a key factor in successfully completing missions in real systems.

BFSSearch (Breadth-First Search) is a classic search algorithm that explores all possible paths at the same distance from the starting state, gradually expanding the search area. In the context of Pac-Man, BFS helps find a path to collect all capsules by checking all available options without using heuristics.

Key components:

1. Queue (fringe). BFS uses a queue to organize path searches. The algorithm explores all possible options step by step, ensuring that the first path found is the shortest in terms of moves. In real-world scenarios, this can be useful for navigation in environments where minimizing the number of moves is important, such as in delivery systems or robotics.

2. State: Each state includes Pac-Man's position and the list of capsules that remain to be collected. In real systems, this could be the equivalent of achieving a set of goals in a terrain, such as data collection or task completion in designated areas.

3. No Heuristics. BFS explores all options without additional evaluation. This makes it less optimized for execution time compared to A*, but it guarantees finding the shortest path in terms of moves.

4. Closed List. The algorithm remembers all visited states to avoid re-exploring them, reducing computational complexity and resource costs.

In real systems, BFS can be useful for finding the shortest paths in environments with many options, where the number of moves is more important than resource costs. For example, in urban navigation, autonomous robots can use BFS to explore routes with the fewest maneuvers. BFS is also effective for tasks where all possible options must be explored without prior evaluation, such as in search and exploration of new environments.

RandomAgent is a simple agent that randomly chooses an action from the available options, ignoring the stop action. It does not use any heuristics or path planning algorithms, making it the simplest decision-making option. This agent can be useful as a control, representing the worst possible option or for evaluating the effectiveness of other algorithms.

Key components:

1. Action selection: The agent receives all available actions in the current state and randomly selects one, excluding the stop action. This makes it entirely dependent on randomness, without any strategy or reasoned choice.

2. Lack of planning: RandomAgent does not analyze the current situation or the future consequences of its actions. This means it cannot adapt its strategy to changes in the environment, making it very inefficient in complex tasks. In real systems, such an approach could be analogous to random actions or chaotic behavior in the absence of data or limited resources.

In real systems, a random approach similar to RandomAgent may be used as a control group or to test the boundaries of other algorithms' effectiveness. For example, in modeling autonomous systems, random decisions can show the worst-case scenario, against which the performance of more complex algorithms can be compared. Additionally, such agents may be useful in systems where full optimization is impossible due to a lack of environmental information.

AlphaBetaAgent uses the alpha-beta pruning algorithm for efficient decision-making in the Pac-Man game. This agent acts as a maximizer for Pac-Man and a minimizer for the ghosts, aiming to maximize Pac-Man's score while considering strategies for avoiding ghosts and collecting capsules.

Key components:

1. Alpha-beta pruning: The algorithm prunes unnecessary branches of the decision tree when it can be guaranteed that a particular path will not improve the outcome. This reduces the number of computations without losing decision quality. In real-world scenarios, alpha-beta pruning is applied in many multi-level systems for decision optimization under limited resources.

2. Max/Min nodes: Each Pac-Man move is a maximizing node, where the algorithm seeks to increase the score (by collecting capsules), while each ghost move is a minimizing node, where the ghosts try to reduce Pac-Man's points (by approaching him). This allows a balance between aggressive and defensive strategies, similar to threat avoidance scenarios in autonomous systems.

3. Evaluation function: The **CapsulesEvaluationFunction** evaluates the current game state based on the distance to the nearest capsules, ghosts, and food. This function provides a balanced approach to avoiding threats and maximizing scores. In real autonomous systems, similar evaluations can be used for route optimization, where it is necessary to avoid dangers while achieving goals.

In real systems where it is crucial to avoid threats while completing tasks, AlphaBetaAgent can be applied to optimize avoidance and goal completion. For example, autonomous drones may use similar strategies to avoid obstacles while maintaining focus on the main tasks. The algorithm allows for effective decision-making in multi-level gaming environments, akin to tasks where optimal decisions must be made under time and resource constraints.

ExpectimaxAgent uses the Expectimax algorithm, allowing Pac-Man to choose actions that maximize his expected score. In this algorithm, Pac-Man's actions maximize the result, while the ghosts make their moves randomly, reflecting the probabilistic nature of their behavior. The algorithm enables Pac-Man to plan his actions by considering both favorable and unfavorable scenarios based on the ghosts' random actions.

Key components:

1. Max node (Pac-Man). In Pac-Man's nodes, the algorithm tries to maximize the score by choosing actions that bring the most points (collecting capsules, food, and avoiding ghosts).

2. Exp node (Ghosts). In the ghost nodes, Expectimax calculates the average outcome based on the assumption that the ghosts choose their actions randomly. This allows Pac-Man to assess the probabilities of each possible scenario and choose the best strategy.

3. Evaluation function. The **CapsulesEvaluationFunction** evaluates the current game state, considering the distance to capsules, food, and ghosts. The algorithm prioritizes capsules but also assesses the risks of approaching ghosts and the opportunities to attack scared ghosts. This provides Pac-Man with a balanced strategy between resource collection and threat avoidance.

Expectimax is suitable for scenarios where the agent faces random events or actions of other agents. For example, in real autonomous systems, Expectimax can be used to plan actions in environments where other participants (or threats) act unpredictably. This is useful for autonomous drones or robots operating in dangerous conditions, where

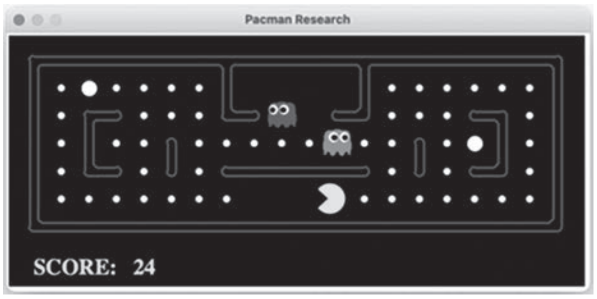

Fig. 1. Small Maze 1 (Optimised) displaying

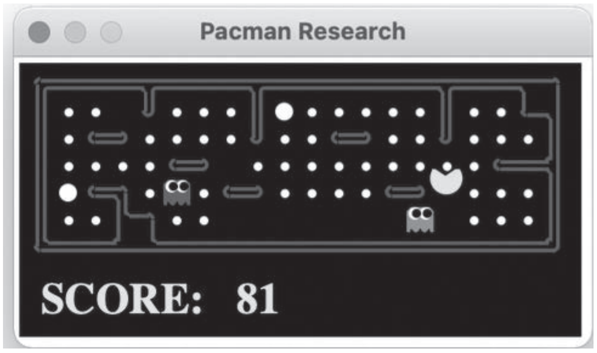

Fig. 2. Small Maze 2 (Non-Optimised) displaying

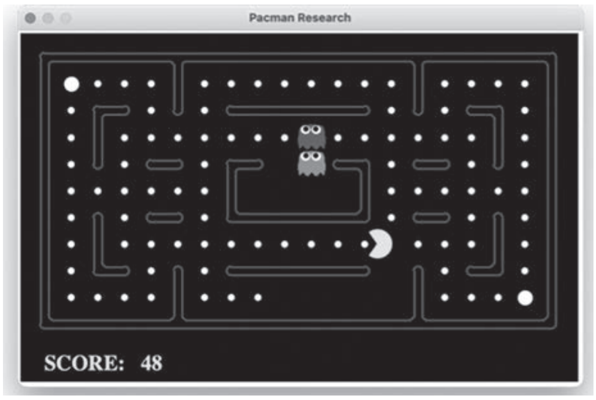

Fig. 3. Medium maze 1 (Optimised) displaying

they must plan actions while considering probable threats or obstacles.

MCTSAgent uses the Monte Carlo Tree Search (MCTS) algorithm to make decisions that efficiently balance the exploration of new opportunities with the exploitation of known strategies. The agent conducts simulations to explore possible strategies in the Pac-Man game, where the ghosts act as dynamic threats.

Key components:

1. Simulations. The agent runs a set number of simulations to explore possible paths and assess the best action. The simulations are based on game situations where Pac-Man avoids ghosts and collects capsules. After the simulations, the action that has been most successful based on the number of visits in the tree is selected.

2. UCT (Upper Confidence Bound for Trees). For selecting the next move during the simulation, MCTS uses the UCB1 policy, which prioritizes not only the most explored nodes but also those that may hide potential gains. This allows the algorithm to efficiently balance between exploring new actions and exploiting already known winning strategies.

3. Evaluation function. The **CapsulesEvaluationMCTSFunction** uses the distance to capsules, food, and ghosts to calculate the effectiveness of Pac-Man's strategy. Capsules and avoiding ghosts are given particular weight, allowing MCTS to find balanced decisions between risks and rewards.

MCTS is an effective tool for decision-making under uncertainty, where the agent must explore many options while evaluating potential gains and risks. In real systems, such as autonomous drones or robots, MCTS can be used to plan actions in complex and dynamic environments, where unknown or variable factors affect the final outcome. MCTS allows for finding optimal solutions by combining a heuristic approach with in-depth exploration of potential scenario developments.

Description and Details of the Experiments

During the experiments, several different maze variations were used, each with its unique structure, number of ghosts, and capsules. The mazes were divided into **optimized** and **non-optimized** categories. Optimized mazes had well-planned paths and convenient passages for Pac-Man, allowing the algorithms to find efficient routes. Non-optimized mazes had chaotic walls and a complex structure, making it significantly harder to find the shortest paths.

Maze types:

• **Small maze 1 (Optimised)**: 7×20, 2 ghosts, 2 objectives (See Fig. 1).

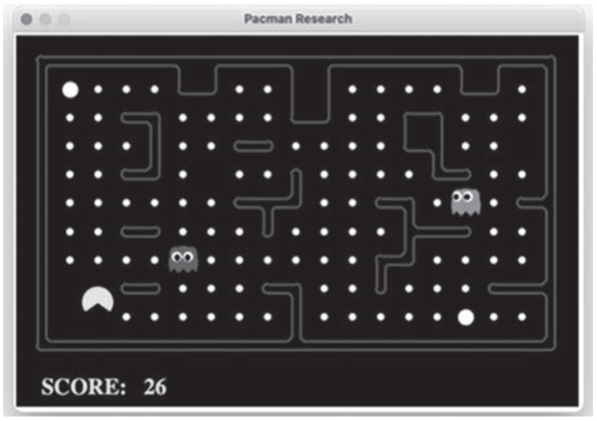

Fig. 4. Medium maze 2 (Non-Optimised) displaying

• **Small maze 2 (Non-Optimised)**: 7×20, 2 ghosts, 2 objectives (See Fig. 2).
• **Medium maze 1 (Optimised)**: 11×20, 2 ghosts, 2 objectives (See Fig. 3).
• **Medium maze 2 (Non-Optimised)**: 11×20, 2 ghosts, 2 objectives (See Fig. 4).
• **Large maze 1 (Original classic, Optimised)**: 27×28, 4 ghosts, 4 objectives (See Fig. 5).
• **Large maze 2 (Non-Optimised)**: 27×28, 4 ghosts, 5 objectives (See Fig. 6).

The game conditions remained the same for all mazes:

• Penalty for inactivity: 1 point per move.
• Collecting a capsule: +500 points.
• Eating a ghost: +200 points.
• Collecting food: +10 points.
• Ghosts' scared timer: 10 moves after collecting a capsule.

To evaluate the agents' efficiency, the following metrics were used:

1. Average score: The total number of points the agent earned during the game. This includes points for capsules, food, and neutralized ghosts, considering penalties for inefficient actions.

2. Time: The amount of time the agent took to complete the game or reach the final goal (collecting all capsules or losing).

3. Win percentage: The percentage of games where Pac-Man won, i.e., collected all capsules without getting caught by the ghosts.

These metrics allow for a comparison of the performance of different agents across various mazes and game conditions.

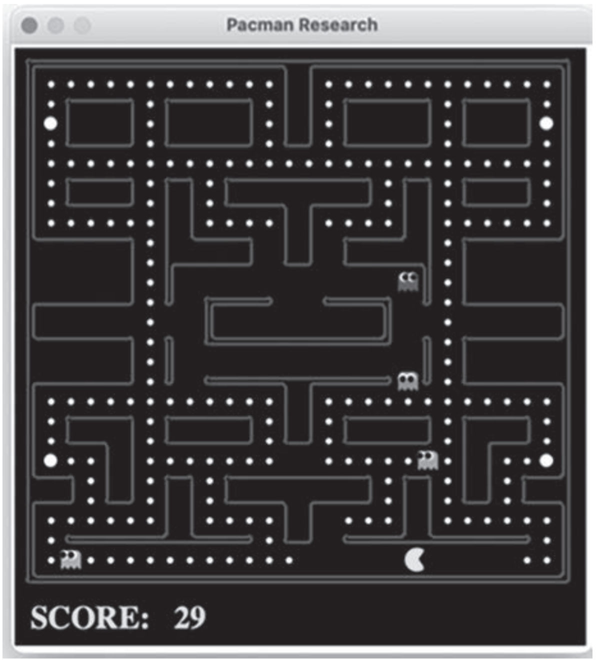

Fig. 5. Large maze 1 (Optimised) displaying

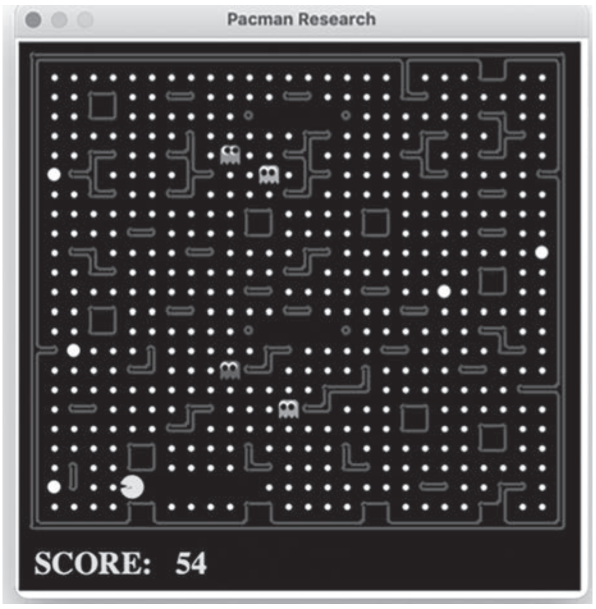

Fig. 6. Large maze 2 (Non-Optimised) displaying

The experiments were conducted with Pac-Man facing the classic ghost agents from the original game. The ghosts followed the classic Pac-Man rules, chasing the player and blocking paths, adding dynamic complexity to each experiment.

These agents followed predefined strategies, presenting challenges for Pac-Man and influencing the effectiveness of each algorithm.

For each maze and agent, 100 runs were conducted to gather enough data for performance analysis. The only variable in each experiment was the Pac-Man agent and its settings.

For agents using MCTS, the number of simulations was adjusted to several levels: 50, 101, 200, 300, 400, 500 simulations. This allowed for an assessment of how the number of simulations affected the agent's performance across different mazes. Other game parameters remained unchanged.

Results

Small Maze 1 (Optimised) — 7×20, 2 ghosts, 2 capsules

In this maze, **AStarSearch** and **BFS** showed almost identical results, with both agents having the same win percentage. **Random Agent** acted chaotically, leading to low results (Table 1).

Expectimax performed the best among multi-agent algorithms, surpassing **AlphaBeta** in both average score and win percentage. **MCTS Agent**, with the current number of simulations, had the worst results among the agents, except for Random Agent (Table 2).

Small Maze 2 (Non-Optimised) — 7×20, 2 ghosts, 2 capsules

AStarSearch Agent demonstrated better results in terms of score and win percentage compared to **BFSSearch Agent**, though the gap between them was minimal. **Random Agent**, as usual, showed poor results (Table 3). **Expectimax Agent** was significantly more effective than **AlphaBeta Agent**, while **MCTS Agent** had a lower average score and win percentage compared to other agents (Table 4).

Medium Maze 1 (Optimised) — 11×20, 2 ghosts, 2 capsules

In the medium maze, search agents like **AStarSearch** and **BFSSearch** demonstrated a low win rate, though their execution time was optimal (Table 5). **Expectimax Agent** was the most effective in this maze, achieving the highest win rate and slightly better performance than the **AlphaBeta Agent**. **MCTS Agent** showed a noticeably lower win rate compared to these agents but still significantly outperformed the search algorithms (Table 6).

Medium Maze 2 (Non-Optimised) — 11×20, 2 ghosts, 2 capsules

In the non-optimized medium maze, the **BFSSearch Agent** outperformed the **AStarSearch Agent**

Table 1. **Results of search algorithms in Small Maze 1 (Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AStarSearch Agent	1063.27	4.12	37
BFSSearch Agent	1071.49	4.08	37
Random Agent	35.88	2.66	0

Table 2. **Results of multi-agent algorithms in Small Maze 1 (Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AlphaBeta Agent	1272.29	7.09	62
Expectimax Agent	1410.85	7.91	72
MCTS Agent (101 simulations)	585.18	7.59	12

Table 3. **Results of search algorithms in Small Maze 2 (Non-Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AStarSearch Agent	963.92	3.62	49
BFSSearch Agent	936.20	3.58	40
Random Agent	61.73	2.68	0

Table 4. **Results of multi-agent algorithms in Small Maze 2 (Non-Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AlphaBeta Agent	1098.27	7.24	17
Expectimax Agent	1212.56	17.03	44
MCTS Agent (101 simulations)	784.41	9.28	9

in terms of win percentage. Random Agent, once again, did not show significant results (Table 7). Multi-agent algorithms: the **Expectimax Agent** demonstrated higher performance than other multi-agent algorithms, showing nearly twice the efficiency of the **AlphaBeta Agent**. **MCTS Agent** results were the least effective, with a significant lag in each performance metric (Table 8).

Large Maze 1 (Optimised) —
27×28, 4 ghosts, 4 capsules

In the large maze, **BFSSearch Agent** showed a slightly higher win percentage compared to **AStarSearch**, despite similar results in average score. **Random Agent** was ineffective, as expected (Table 9). **AlphaBeta Agent** achieved the highest average score, win percentage, and time. **Expectimax** had a better time and showed improved results in each parameter compared to **MCTS Agent** with 50 simulations (Table 10).

Large Maze 2 (Non-Optimised) —
27×28, 4 ghosts, 5 capsules

Search Algorithms: **AStarSearch Agent** and **BFSSearch Agent** demonstrated extremely high efficiency in the non-optimized large maze, with a high win percentage (Table 11).

Multi-Agent Algorithms: **Expectimax Agent** showed the highest average score among multi-agent algorithms. **MCTS Agent** demonstrated the highest win percentage and comparable results in terms of time and average score, which is a significant outcome considering the overall win percentage (Table 12).

Table 5. **Results of search algorithms in Medium Maze 1 (Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AStarSearch Agent	847.12	4.84	5
BFSSearch Agent	863.19	4.86	6
Random Agent	86.13	5.77	0

Table 6. **Results of multi-agent algorithms in Medium Maze 1 (Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AlphaBeta Agent	1427.44	9.12	68
Expectimax Agent	1424.39	9.46	75
MCTS Agent (101 simulations)	1093.71	17.95	34

Table 7. **Results of search algorithms in Medium Maze 2 (Non-Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AStarSearch Agent	1078.98	4.95	54
BFSSearch Agent	1115.53	4.98	61
Random Agent	43.54	2.48	0

Table 8. **Results of multi-agent algorithms in Medium Maze 2 (Non-Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AlphaBeta Agent	989.38	7.73	18
Expectimax Agent	1129.20	10.61	30
MCTS Agent (101 simulations)	766.24	15.31	5

Table 9. **Results of search algorithms in Large Maze 1 (Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AStarSearch Agent	2156.62	18.08	31
BFSSearch Agent	2166.65	18.09	43
Random Agent	80.63	14.31	0

Table 10. **Results of multi-agent algorithms in Large Maze 1 (Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AlphaBeta Agent	2382.58	28.56	25
Expectimax Agent	2272.06	33.45	11
MCTS Agent (50 simulations)	1255.41	36.23	1

Table 11. **Results of search algorithms in Large Maze 2 (Non-Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AStarSearch Agent	2854.78	15.44	77
BFSSearch Agent	2836.62	17.08	72
Random Agent	62.61	4.31	0

Table 12. **Results of multi-agent algorithms in Large Maze 2 (Non-Optimised)**

Agent	Average Score	Time	Win Percentage (%)
AlphaBeta Agent	2985.85	21.27	14
Expectimax Agent	3307.09	30.23	17
MCTS Agent (50 simulations)	3109.16	37.12	29

General Observations:

1. Multi-Agent Algorithms (**Expectimax**, **AlphaBeta**, **MCTS**) consistently demonstrate higher efficiency in larger mazes, where the search space is broader and interaction with ghosts is more complex.

2. Search Algorithms (**AStarSearch**, **BFS**) show high performance in smaller mazes, where strategic evasion of ghosts is less crucial.

3. Random Agent is the least effective, highlighting the need for well-founded strategies to effectively achieve the goal in Pac-Man.

These results may help identify the most effective algorithms depending on the environment complexity and the number of simulations.

Conclusions

The analysis of the conducted experiments leads to several key conclusions regarding the choice of algorithms based on the task type. For search tasks in optimized environments, search algorithms such as **AStarSearch** and **BFSSearch** demonstrate high efficiency. These algorithms perform well in smaller mazes where interacting with a large number of dynamic threats, like ghosts, is not necessary. **AStarSearch** delivers better results in optimized mazes due to its effective consideration of distances to objectives, while **BFSSearch** is competitive in more complex, non-optimized environments.

Multi-agent algorithms such as **AlphaBeta**, **Expectimax**, and **MCTS** perform better in complex environments with dynamic threats where the probable behavior of ghosts must be considered. **Expectimax** demonstrates the best results in conditions where ghost behavior can change randomly, especially in small and medium-sized mazes. **AlphaBeta** is suitable for situations where minimizing risks is necessary by avoiding direct collisions with ghosts. **MCTS** proves most effective in large mazes due to its ability to conduct numerous simulations, allowing it to find optimal solutions in highly complex environments with many possible scenarios.

The performance analysis of search and multi-agent algorithms shows that while search algorithms perform well in optimized environments, their effectiveness decreases in dynamic conditions involving interaction with ghosts. Multi-agent algorithms are more flexible and effective in complex situations where it is necessary to avoid threats while collecting capsules.

Random Agent, which acts randomly, showed the worst results and served as a control variant for comparing with other algorithms. The results show that even the simplest search algorithms, such as **AStarSearch**, significantly outperform random strategies, while multi-agent algorithms provide the highest efficiency in complex and dynamic environments.

Thus, the choice of algorithm for Pac-Man depends on the maze's complexity and the necessity of accounting for ghost actions. In optimized and simplified conditions, search algorithms are more effective, while for more dynamic environments and complex mazes, multi-agent approaches are the most promising. Prospects for improving the algorithms include increasing the number of simulations for **MCTS** and improving heuristics for search algorithms to make them more adaptable to complex environments.

REFERENCES

1. Foderaro, G., Swingler, A., Ferrari, S. (2017) "A Model-Based Approach to Optimizing Ms. Pac-Man Game Strategies in Real Time." *IEEE Transactions on Computational Intelligence and AI in Games*, 9 (2), pp. 153–165. DOI: https://doi.org/10.1109/TCIAIG.2016.2523508
2. Rohlfshagen, P., Liu, J., Perez-Liebana, D., Lucas, S.M. (2018) "Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game." *IEEE Transactions on Games*, 10 (3), pp. 233–256. DOI: https://doi.org/10.1109/TG.2017.2737145
3. Gallagher, M., Ryan, A. (2003) "Learning to play Pac-Man: an evolutionary, rule-based approach." The 2003 Congress on Evolutionary Computation, Canberra, ACT, Australia, pp. 2462–2469, Vol. 4. doi: https://doi.org/10.1109/CEC.2003.1299397
4. Foderaro, G., Raju, V., Ferrari, S. (2011) "A cell decomposition approach to online evasive path planning and the video game Ms. Pac-Man." 2011 IEEE International Symposium on Intelligent Control, Denver, CO, USA, pp. 191–197. DOI: https://doi.org/10.1109/ISIC.2011.6045414
5. Tucnik, P., Nachazel, T., Cech, P., Bures, V. (2018) "Comparative analysis of selected path-planning approaches in large-scale multi-agent-based environments." Expert Systems with Applications, 113, pp. 415–427. DOI: https://doi.org/10.1016/j.eswa.2018.07.001
6. Sajid, Q., Luna, R., Bekris, K. (2012) "Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives." Proceedings of the 5th Annual Symposium on Combinatorial Search, pp. 88–96. DOI: http://doi.org/10.1609/socs.v3i1.18243.
7. Saccon, E., Palopoli, L., Roveri, M. (2023) "Comparing Multi-Agent Path Finding Algorithms in a Real Industrial Scenario." In: Dovier, A., Montanari, A., Orlandini, A. (eds) AIxIA 2022 – Advances in Artificial Intelligence. AIxIA 2022. Lecture Notes in Computer Science, vol 13796. Springer, Cham. DOI: https://doi.org/10.1007/978-3-031-27181-6_13
8. Erdem, E., Kisa, D., Oztok, U., Schüller, P. (2013) "A General Formal Framework for Pathfinding Problems with Multiple Agents." Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013.
9. Hewawasam, H., Ibrahim, Y., Appuhamillage, G. (2022) "Past, Present and Future of Path-Planning Algorithms for Mobile Robot Navigation in Dynamic Environments." IEEE Open Journal of the Industrial Electronics Society, 3 (1), pp. 353–365. DOI: http://doi.org/10.1109/OJIES.2022.3179617
10. Lu, Y., Huo, X., Arslan, O., Tsiotras, P. (2012) "Incremental Multi-Scale Search Algorithm for Dynamic Path Planning With Low Worst-Case Complexity." IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 41, pp. 1556–1570. DOI: http://doi.org/10.1109/TSMCB.2011.2157493
11. Saffidine, A., Finnsson, H., Buro, M. (2012) "Alpha-Beta Pruning for Games with Simultaneous Moves." Proceedings of the National Conference on Artificial Intelligence, 1.
12. Lee, S. (2019) "Exploring to Learn Winning Strategy." International Conferences Interfaces and Human Computer Interaction, Game and Entertainment Technologies, and Computer Graphics, Vsualization, Computer Vision and Image Processing, pp. 377–380. DOI: http://dx.doi.org/10.33965/g2019_201906C052.
13. Dam, T., D'Eramo, C., Peters, J., Pajarinen, J. (2022) "A Unified Perspective on Value Backup and Exploration in Monte-Carlo Tree Search." ArXiv abs/2202.07071.
14. Mirsoleimani, S.A., Plaat, A., van den Herik, J., Vermaseren, J.A.M. (2015) "Ensemble UCT Needs High Exploitation." ArXiv abs/1509.08434, 2015.
15. Felner, A., Stern, R., Shimony, S.E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N. R., Wagner, G., Surynek, P. (2021) "Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges." Symposium on Combinatorial Search. DOI: https://doi.org/10.1609/socs.v8i1.18423
16. Khandelwal, P., Liebman, E., Niekum, S., Scott, A., Stone, P. (2016) "On the Analysis of Complex Backup Strategies in Monte Carlo Tree Search." International Conference on Machine Learning, pp. 1319–1328.
17. Ou, T., Cao, J., Lu, Y., Wang, Y.-P., Wu, X. (2023) "A New Decision-Making Approach via Monte Carlo Tree Search and A2C." 2023 3rd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), pp. 204–210. DOI: https://doi.org/10.1109/CEI60616.2023.10527918
18. Silva, M.A.L., de Souza, S.R., Souza, M.J.F., de França Filho, M.F. (2018) "Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis." Applied Soft Computing, 71, pp. 433–459. DOI: https://doi.org/10.1016/j.asoc.2018.06.050

Received 21.08.2024

ЛІТЕРАТУРА

1. *Foderaro G., Swingler A., Ferrari S.* "A Model-Based Approach to Optimizing Ms. Pac-Man Game Strategies in Real Time." IEEE Transactions on Computational Intelligence and AI in Games, 2017, 9 (2), pp. 153-165. DOI: https://doi.org/10.1109/TCIAIG.2016.2523508
2. *Rohlfshagen P., Liu J., Perez-Liebana D., Lucas S. M*. "Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game." IEEE Transactions on Games, 2018, 10 (3), pp. 233-256. DOI: https://doi.org/10.1109/TG.2017.2737145
3. *Gallagher M., Ryan A*. "Learning to play Pac-Man: an evolutionary, rule-based approach." The 2003 Congress on Evolutionary Computation, 2003, Canberra, ACT, Australia, pp. 2462-2469, Vol. 4. doi: https://doi.org/10.1109/CEC.2003.1299397
4. *Foderaro G., Raju V., Ferrari S.* "A cell decomposition approach to online evasive path planning and the video game Ms. Pac-Man." 2011 IEEE International Symposium on Intelligent Control, 2011, Denver, CO, USA, pp. 191-197. DOI: https://doi.org/10.1109/ISIC.2011.6045414
5. *Tucnik P., Nachazel T., Cech P., Bures V.* "Comparative analysis of selected path-planning approaches in large-scale multi-agent-based environments." Expert Systems with Applications, 2018, 113, pp. 415-427. DOI: https://doi.org/10.1016/j.eswa.2018.07.001
6. *Sajid Q., Luna R., Bekris K*. "Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives." Proceedings of the 5th Annual Symposium on Combinatorial Search, 2012, pp. 88-96. DOI: http://doi.org/10.1609/socs.v3i1.18243
7. *Saccon E., Palopoli L., Roveri M.* "Comparing Multi-Agent Path Finding Algorithms in a Real Industrial Scenario." In: Dovier, A., Montanari, A., Orlandini, A. (eds) AIxIA 2022 – Advances in Artificial Intelligence. AIxIA 2022. Lecture Notes in Computer Science, vol 13796. Springer, Cham, 2023. DOI: https://doi.org/10.1007/978-3-031-27181-6_13
8. *Erdem E., Kisa D., Oztok U., Schüller P.* "A General Formal Framework for Pathfinding Problems with Multiple Agents." Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013.
9. *Hewawasam H., Ibrahim Y., Appuhamillage G.* "Past, Present and Future of Path-Planning Algorithms for Mobile Robot Navigation in Dynamic Environments." IEEE Open Journal of the Industrial Electronics Society, 2022, 3 (1), pp. 353-365. DOI: http://doi.org/10.1109/OJIES.2022.3179617
10. *Lu Y., Huo X., Arslan O., Tsiotras P.* "Incremental Multi-Scale Search Algorithm for Dynamic Path Planning With Low Worst-Case Complexity." IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2012, 41, pp. 1556-1570. DOI: http://doi.org/10.1109/TSMCB.2011.2157493
11. *Saffidine A., Finnsson H., Buro M*. "Alpha-Beta Pruning for Games with Simultaneous Moves." Proceedings of the National Conference on Artificial Intelligence, 2012, 1.
12. *Lee S. "Exploring to Learn Winning Strategy.*" International Conferences Interfaces and Human Computer Interaction, Game and Entertainment Technologies, and Computer Graphics, Visualization, Computer Vision and Image Processing, 2019, pp. 377-380. DOI: http://dx.doi.org/10.33965/g2019_201906C052
13. *Dam T., D'Eramo C., Peters J., Pajarinen J.* "A Unified Perspective on Value Backup and Exploration in Monte-Carlo Tree Search." ArXiv abs/2202.07071, 2022.
14. *Mirsoleimani S. A., Plaat A., van den Herik J., Vermaseren J. A. M.* "Ensemble UCT Needs High Exploitation." ArXiv abs/1509.08434, 2015.
15. *Felner A., Stern R., Shimony S. E., Boyarski E., Goldenberg M., Sharon G., Sturtevant, N. R., Wagner, G., Surynek, P.* "Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges." Symposium on Combinatorial Search, 2021. DOI: https://doi.org/10.1609/socs.v8i1.18423
16. *Khandelwal P., Liebman E., Niekum S., Scott A., Stone P.* "On the Analysis of Complex Backup Strategies in Monte Carlo Tree Search." International Conference on Machine Learning, 2016, pp. 1319-1328.
17. *Ou T., Cao J., Lu Y., Wang Y.-P., Wu, X.* "A New Decision-Making Approach via Monte Carlo Tree Search and A2C." 2023 3rd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), 2023, pp. 204-210. DOI: https://doi.org/10.1109/CEI60616.2023.10527918
18. *Silva M. A. L., Souza S. R. de, Freitas Souza M. J., Felizardo de França Filho M.* "Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis." Applied Soft Computing, 2018, 71, pp. 433-459. DOI: https://doi.org/10.1016/j.asoc.2018.06.050

Надійшла 21.08.2024

А.О. Новіков, аспірант ННІ комп'ютерних наук та штучного інтелекту
кафедри математичного моделювання та аналізу даних,
Харківський національний університет ім. В.Н. Каразіна,
пл. Свободи, 4, Харків, Україна, 61022,
ORCID: https://orcid.org/0009-0004-5914-7098,
artem.slick@gmail.com

В.В. Яновський, доктор фіз.-мат. наук, професор, завідувач теоретичним відділом,
"Інститут монокристалів" Національної Академії наук України,
просп. Науки, 60, Харків, Україна, 61001,
ORCID: https://orcid.org/0000-0003-0461-749X; Scopus Author ID 7003273794,
yanovsky@isc.kharkov.ua

АНАЛІЗ ПОШУКОВИХ І МУЛЬТИАГЕНТНИХ АЛГОРИТМІВ У ГРІ PAC-MAN

Вступ. У даній роботі досліджується ефективність пошукових та мультиагентних алгоритмів у контексті гри *Pac-Man*. Гра Pac-Man моделює завдання управління автономними системами у двовимірному середовищі, де агент стикається з динамічними перешкодами. Використання класичних пошукових алгоритмів, таких як *A**, *BFS*, а також мультиагентних підходів, таких як *Alpha-Beta*, *Expectimax* та *Monte Carlo Tree Search* (*MCTS*), дозволяє дослідити різні стратегії для вирішення задач оптимізації шляху та ухилення від динамічних загроз (привидів).

Мета статті. Метою даного дослідження є аналіз продуктивності різних алгоритмів у різних за складністю лабіринтах, зокрема за показниками середнього балу, часу виконання та відсотка перемог. Це дослідження спрямовано на порівняння ефективності пошукових та мультиагентних алгоритмів в умовах змінного середовища.

Методи. У дослідженні використано системний підхід та метод експериментального моделювання. Ефективність алгоритмів оцінювалася шляхом проведення численних експериментів у лабіринтах з різним рівнем складності.

Результати. Отримані результати показують, що пошукові алгоритми (*A** та *BFS*) демонструють високу продуктивність у менш динамічних середовищах, тоді як мультиагентні алгоритми *(Expectimax*, *Alpha-Beta*, *MCTS*) більш ефективні в складніших лабіринтах з багатьма динамічними загрозами. *Expectimax* продемонстрував найкращі результати в середовищах із випадковими діями супротивників, тоді як *MCTS* показав високу продуктивність у великих та складних середовищах.

Висновки. Дослідження виявило, що вибір алгоритму залежить від складності лабіринту та необхідності врахування дій динамічних супротивників. Пошукові алгоритми є ефективними у спрощених умовах, тоді як мультиагентні підходи є перспективними для складних середовищ з динамічними загрозами.

Ключові слова: *алгоритми пошуку, мультиагентні алгоритми, Pac-Man, оптимізація шляху, Alpha-Beta, Expectimax, MCTS.*