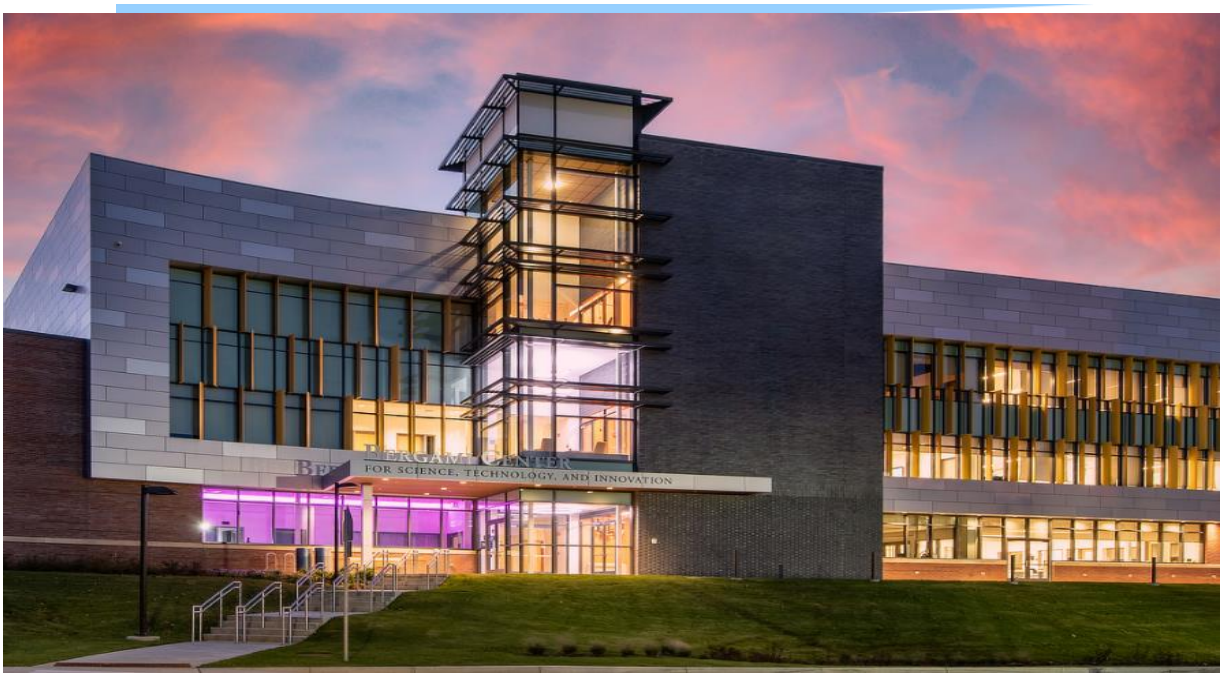
University of New Haven
TAGLIATELA COLLEGE OF ENGINEERING
Electrical & Computer Engineering and Computer Science

Data Science

Comparison of Search-based and Reinforcement Learning Agents in Solving the Game of Snake

TECHNICAL REPORT

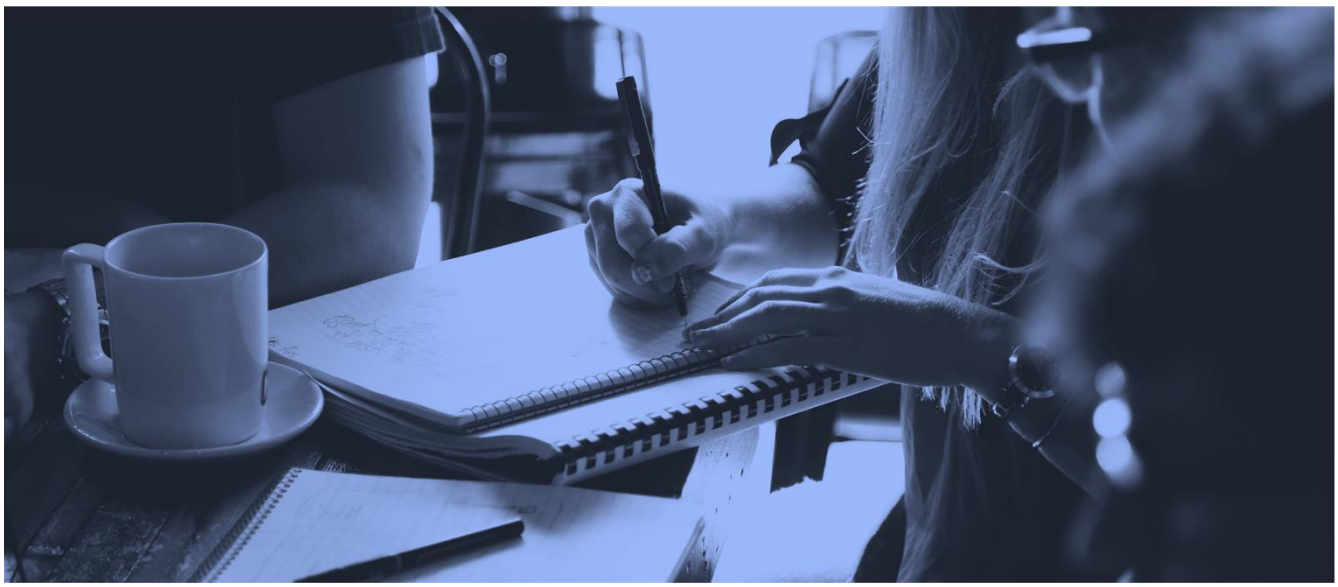


CONTENTS

Comparison of Search-based and Reinforcement Learning Agents in Solving the Game of Snake

Executive Summary

Artificial Intelligence (AI) stands as a fundamental branch in computer science, experiencing a rapid rise in adoption. Its primary application is notably prevalent in the gaming industry, where AI contributes significantly to the development of non-player characters (NPCs). Various methodologies exist for implementing AI in games, with the search method emerging as the most widespread approach. Diverse search algorithms are employed in the implementation of AI, enhancing the intelligence and behavior of NPCs within the gaming environment.



Team Members:
Vamsi Krishna Jammigumpula (Developer)
Venkata Raja Varaprasad Sanayila (ML Engineer)
Deepak Grandhe (Data Engineer)

Questions?
Contact :vsana4@unh.newhaven.edu

Submitted on 12/13/2023

Github link : https://github.com/Venkata-Raja-Vara-Prasad-Sanayila/Snake-Game-by-using-AI-Search-Algorithm

Abstract

Artificial Intelligence (AI) plays a pivotal role in computer science, witnessing widespread adoption and finding significant application in the gaming industry, particularly in the development of non-player characters (NPCs). This study focuses on implementing diverse search algorithms to create an intelligent self-playing snake game, with a scoring system enabling the assessment and analysis of each algorithm's performance.

The research methodology involves a comprehensive literature review, encompassing performance studies on human agents and select algorithms within the context of the snake game. The project aims to identify and evaluate AI-based searching methods, with a specific focus on key algorithms such as Depth-First Search (DFS), Breadth-First Search (BFS), A*, and Uniform Cost Search (UCS). Performance metrics, including game scores achieved by each algorithm, are scrutinized to deduce insights beneficial for advancing AI in video games. The findings underscore the significance of various AI techniques in enhancing the intelligence and behavior of NPCs, contributing to the evolution of AI within gaming environments.

Introduction

Artificial Intelligence (AI) stands at the forefront of Computer Science, emerging as a pivotal field that addresses complex tasks with remarkable efficiency. In the contemporary landscape, AI has become synonymous with problem-solving prowess, employing a myriad of algorithms designed to emulate human behavior [3]. The integration of AI into machines transforms them into computer-controlled entities capable of executing tasks typically associated with intelligent beings. This evolution is marked by characteristics such as learning, adaptability, and the ability to generalize, mirroring the cognitive abilities of humans.

In the current era of technological advancements, AI has achieved performance levels comparable to human expertise, finding applications in diverse domains, including voice recognition, handwriting analysis, and the development of self-driving cars [20] [21]. While

AI in the real-world addresses a broad spectrum of challenges, its manifestation in the gaming realm presents a distinct landscape within AI research.

Gaming, a prominent area of AI exploration, involves the implementation of AI to craft Non-Player Characters (NPCs) that enhance gaming experiences. The synergy between AI and gaming goes beyond complex algorithms, focusing on the creation of intelligent NPCs that elevate gameplay. Unlike real-world AI applications, game-based AI relies on the application of specific rules and conditions to portray characters as intelligent entities [27].

Despite the strides made in the gaming industry, the emphasis has predominantly been on graphical advancements, with less attention directed toward the intelligence of in-game agents. Successful completion of games by Human Agents has prompted the need for more challenging levels to sustain interest. The design of game levels hinges on algorithmic performance, with superior-performing algorithms earmarked for the implementation of challenging game stages.

Among the array of AI implementations in games, searching algorithms occupy a crucial role. They facilitate the movement of NPCs from one location to another, contributing to the strategic navigation of game environments. However, the intricate process of implementing each searching algorithm in a game poses challenges in terms of time and complexity. This underscores the necessity of identifying and implementing the most efficient searching algorithm to optimize results.

This thesis embarks on a comprehensive exploration of searching algorithms by implementing them in a snake game. Notable algorithms such as Breadth-First Search, Depth-First Search, A* Search, and Uniform Cost Search have been selected for their common usage and applicability as pathfinding algorithms. The ensuing chapters delve into a comparative analysis of the performance of these algorithms, drawing comparisons with Human Agents and each other. Through this investigation, the aim is to discern the most effective searching algorithm for integration into gaming environments, fostering enhanced intelligence and adaptability in NPC behaviors.

Review

The project undertook a thorough literature review, delving into the performance studies of both human agents and selected algorithms within the context of the snake game. The primary objective was to identify and evaluate AI-based searching methods, thereby contributing to the enhancement of artificial intelligence (AI) in video games.

The examination of performance metrics, particularly the game scores attained by different algorithms, played a pivotal role in assessing the effectiveness of AI implementations in the snake game. The chosen algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), A*, and Uniform Cost Search (UCS), underwent rigorous testing to gauge their respective performances.

The deductions drawn from the background research underscored the manifold benefits of employing various AI techniques for the advancement of AI in video games. The project not only sought to analyze the performance of these algorithms but also aimed to unravel insights into their applicability and effectiveness within the gaming environment.

By aligning the goals with empirical testing, the project successfully demonstrated the diverse ways in which AI-based searching methods impact the gameplay experience. The systematic evaluation of DFS, BFS, A*, and UCS against performance metrics provides valuable data that contributes to the ongoing discourse on optimizing AI techniques for video games.

Methodology

The methodology employed in this project encompasses a multifaceted approach to comprehensively explore and evaluate AI-based searching methods within the context of a snake game. The overarching process can be divided into key components, each contributing to the project's objectives:

1. Literature Review:

- A thorough literature review was conducted to delve into the existing body of knowledge concerning the performance of human agents and specific algorithms in the snake game. This initial phase aimed to establish a foundational understanding of the current landscape and identify gaps in knowledge that the project could address.

2. Identification of AI-Based Searching Methods:

- Building on insights gained from the literature review, the project focused on identifying AI-based searching methods that could enhance the intelligence and behavior of non-player characters (NPCs) within the snake game. This step laid the groundwork for the subsequent empirical testing phase.

3. Algorithm Selection:

- Based on the identified AI-based searching methods, a set of algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), A*, and Uniform Cost Search (UCS), were selected for empirical testing. These algorithms were chosen for their common usage and applicability as pathfinding algorithms.

4. Empirical Testing:

conclusion, the project's literature review and empirical testing shed light on the otal role of AI in video games, with a specific focus on searching algorithms. The dings contribute to the broader understanding of AI applications in gaming, hlighting the potential for further advancements and optimizations in video game AI.

- The selected algorithms underwent rigorous empirical testing within the snake game environment. Performance metrics, such as game scores achieved by each algorithm, were systematically evaluated to gauge their effectiveness in navigating the game space.

5. Performance Comparison:

- A comparative analysis was conducted to assess the performance of each algorithm in relation to one another and, notably, against the performance of Human Agents.

This step aimed to discern the strengths and limitations of each searching method in the specific gaming context.

6. Insights and Conclusions:

- The project culminated in the synthesis of insights derived from both the literature review and empirical testing. These insights were used to draw conclusions regarding the efficacy of AI-based searching methods in the snake game, providing valuable information for the advancement of AI in video games.

Overall, the methodology employed in this project is designed to provide a comprehensive understanding of the performance and applicability of AI-based searching methods in the dynamic and interactive realm of video games. Through a systematic approach, the project contributes to the ongoing discourse on optimizing AI techniques for enhanced gaming experiences.

Depth-First Search

The Depth-First Search (DFS) technique explores a vertex to its deepest level before backtracking through unexplored nodes until the initial vertex is reached. Also referred to as an edge-based method, DFS operates on a last-in-first-out stack data structure. The algorithm employs recursion until it reaches the goal node, traversing edges twice and each vertex only once along the path. Its space requirement is linear, avoiding the exponential increase seen in Breadth-First Search for deep node searches. The time complexity of the DFS algorithm is O(bd), and its space complexity is O(b*n), where 'b' is the branching factor and 'n' is the number of levels [9].

Advantages:

1. It demonstrates efficiency by using minimal time when the target is in close proximity, stopping the search once the goal node is reached.

2. In worst-case scenarios, DFS offers the possibility of reaching the target state through multiple paths.

3. The primary appeal of DFS lies in its linear space requirement during the search process.

Breadth-First Search

The Breadth-First Search (BFS) technique operates by systematically traversing or exploring the deepest nodes before moving on to adjacent nodes. This method involves exploring the search tree level by level, expanding all nodes at each level before progressing to the next level and ultimately reaching the goal. Consequently,

BFS is often referred to as a level-by-level traversal technique. In this approach, all solutions for each node are exhaustively determined, ensuring the discovery of the optimal solution.

BFS utilizes a queue data structure to store values, maintaining a First In, First Out (FIFO) principle. Even in the presence of cycles in the graph, BFS employs an array to store visited vertices in the search tree. The time and space complexity of the Breadth-First Search is denoted as O(bd + 1) and O(bd + 1), respectively, where 'b' represents the branching factor and 'n' is the number of levels.

Advantages:

1. The Breadth-First Search minimizes the number of steps required to reach the goal node in the search tree, making it an efficient approach.

2. This technique guarantees the existence of a path to reach the goal node in the search tree.

A* Search

The A* search algorithm amalgamates the favorable aspects of Uniform Cost Search and pure Heuristic Search, aiming to achieve both optimal solutions and completeness of the path, along with optimal efficiency in state space search trees. This algorithm endeavors to discover the optimal path from a starting node to a goal node by iteratively extending the tree of nodes until the goal state is reached.

Throughout its iterations, the A* search algorithm extends the path by employing cost estimation, facilitating progress toward the goal node. The cost estimation formula, denoted as f(n) = g(n) + h(n), encompasses the cost of the path from the source node to node 'n' (g(n)), the estimated heuristic cost from the target node to 'n' (h(n)), and the total optimal cost of a path passing through node 'n' (f(n)) [7]. If the heuristic function is admissible, the A* search algorithm consistently selects the least costly path from the start to the goal node.

Utilizing a priority queue data structure, the A* search removes lower f values at each step, updates the adjacent nodes' f(n) and g(n) values accordingly, and iterates until the f(n) value is lower than any node in the queue. The algorithm's time complexity is O(bd), and its space complexity is O(bd) [19].

Advantages:

1. A* search finds the best path in minimal time, effectively reducing time complexity.

2. It employs a linear data structure to store f(n) values.

3. Time complexity is minimized with heuristic evaluation functions during node searches.

4. The algorithm explores only a limited number of nodes, enhancing efficiency.

Uniformed Cost Search

Uniform Cost Search is employed to discover the minimum cost traversal within a graph tree, making it a key algorithm in state space search. Utilizing a priority queue, it systematically determines the minimum cost of adjacent nodes. The algorithm systematically backtracks, exploring all potential paths to the destination, and subsequently selects the minimum cost path from the root to the destination. Also known as Dijkstra's single-source shortest path algorithm, Uniform Cost Search compares and chooses optimal costs while exploring paths in a graph or tree.

The algorithm employs the formula $c(m) = c(n) + c(n, m)$ for calculating the cost of each node, where $c(m)$ is the cost of the current node, $c(n)$ is the cost of the previous node, and $c(n, m)$ is the distance cost from node n to m. Its time and space complexity are denoted as $O(b^1 + \frac{C^*}{\varepsilon})$ and $O(b^1 + \frac{C^*}{\varepsilon})$, respectively, where C^* represents the optimal solution cost, and each activity has a cost of at least ε.

Advantages:

1. It efficiently identifies optimal solutions by considering the minimum cost for each node.

2. Space utilization is minimized by focusing on minimum cost nodes.

3. Time complexity is reduced by selecting minimum nodes for exploration.

Disadvantages:

1. The algorithm explores multiple paths to achieve the minimum cost traversal from the root to the destination, leading to increased time complexity.

2. Space complexity escalates when a path fails to find the optimal solution.

3. If the chosen path fails to yield the optimal solution, the algorithm compares it with the previous solution, introducing additional space complexity and execution time.

Results

Upon conducting tests and consolidating data, it was evident that BFS exhibited the superior performance across the board. This conclusion was drawn by employing our scoring system, represented as:

$$\text{finalScore} = \left(\frac{\text{score}}{\text{actions}}\right) \times 100$$

Here, "score" denotes the number of food items eaten, and "actions" encompasses the cumulative count of all actions taken until a failure occurred. Emphasizing the significance of score over actions was intentional, as an algorithm achieving a higher score signifies greater success. This prioritization ensures that algorithms are assessed based on their effectiveness in achieving the goal, rather than solely on the number of actions performed. For instance, while DFS may generate numerous actions, resulting in a low score, it is ranked lower due to its lower overall success in comparison to other algorithms.

BFS ACTIONS: 1318
DFS ACTIONS: 4127
ASTAR ACTIONS: 1463
UCS ACTIONS: 1391

RAW BFS SCORE: 42
RAW DFS SCORE: 87
RAW ASTAR SCORE: 88
RAW UCS SCORE: 87

CALC BFS SCORE: 1.0176883935061787
CALC DFS SCORE: 6.600910470409711
CALC ASTAR SCORE: 6.015037593984962
CALC UCS SCORE: 6.2544931703810205

RUN NUMBER: 0 DATE:2023-12-01 21:19:47.228020
BFS ACTIONS: 1093
DFS ACTIONS: 4781
ASTAR ACTIONS: 1737
UCS ACTIONS: 1122

RAW BFS SCORE: 38
RAW DFS SCORE: 78
RAW ASTAR SCORE: 107
RAW UCS SCORE: 72

CALC BFS SCORE: 0.794812800669316
CALC DFS SCORE: 7.136322049405306
CALC ASTAR SCORE: 6.160046056419113
CALC UCS SCORE: 6.417112299465241

RUN NUMBER: 0 DATE:2023-12-01 21:34:08.952507
BFS ACTIONS: 375
DFS ACTIONS: 3721
ASTAR ACTIONS: 395
UCS ACTIONS: 393

RAW BFS SCORE: 37
RAW DFS SCORE: 34
RAW ASTAR SCORE: 36
RAW UCS SCORE: 38

CALC BFS SCORE: 0.9943563558183284
CALC DFS SCORE: 9.066666666666666
CALC ASTAR SCORE: 9.113924050632912
CALC UCS SCORE: 9.669211195928753

RUN NUMBER: 0 DATE:2023-12-01 21:34:35.764781
BFS ACTIONS: 947

DFS ACTIONS: 3300
ASTAR ACTIONS: 1350
UCS ACTIONS: 1137

RAW BFS SCORE: 45
RAW DFS SCORE: 76
RAW ASTAR SCORE: 96
RAW UCS SCORE: 79

CALC BFS SCORE: 1.3636363636363635
CALC DFS SCORE: 8.025343189017951
CALC ASTAR SCORE: 7.111111111111111
CALC UCS SCORE: 6.948109058927001

RUN NUMBER: 0 DATE:2023-12-01 21:34:50.708190

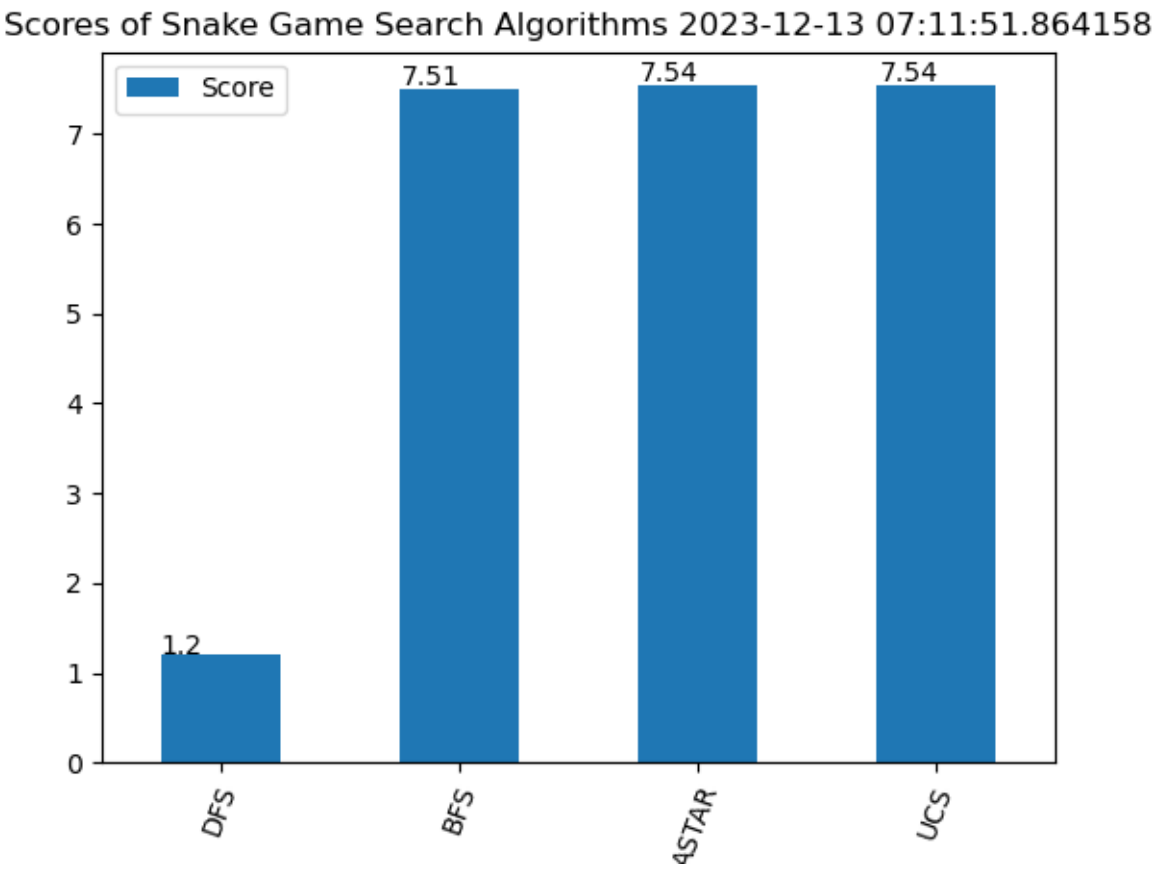


Fig. Comparison of Scores for AI Algorithms

Overall, BFS performs the best

Summary:

- DFS: runs slowly with lots of actions, always lowest food score
- BFS: runs quickly and linearly, good food score, low actions
- Astar: runs quickly and diagonally, good food score, more actions
- UCS: runs quickly and diagonally, good food score, more actions

Conclusion

In conclusion, the comprehensive analysis and evaluation of different searching algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), A* (Astar), and Uniform Cost Search (UCS), within the context of a snake game, have provided valuable insights into their respective strengths and weaknesses.

DFS: While DFS may be characterized by slow execution with a high number of actions, it consistently yields the lowest food score, making it less optimal for the snake game.

BFS: On the other hand, BFS stands out as the most effective algorithm, demonstrating quick and linear execution, achieving a commendable food score, and minimizing the number of actions.

Astar: A* combines speed with diagonal movement, resulting in a good food score, albeit with a higher number of actions compared to BFS.

UCS: Similar to A*, UCS operates swiftly with diagonal movement, achieving a satisfactory food score but involving more actions.

Considering the prioritization of score over actions in the evaluation, BFS emerges as the preferred choice for the snake game. Its ability to strike a balance between speed, linearity, and efficiency in achieving a high food score with minimal actions positions it as the most suitable algorithm for this gaming context.

The findings emphasize the importance of selecting an algorithm that aligns with the specific requirements of the snake game, where achieving a high score is paramount. The optimization of BFS in this regard highlights its potential for enhancing the gaming experience by efficiently navigating the snake through the game environment.

Contributions/References

[1] Finite state machines (are boring). https://martindevans.me/heist-game/2013/04/16/Finite-State-Machines-(Are-Boring)/. (Accessed on 10/05/2020).

[2] Snake (video game genre) - wikipedia. https://en.wikipedia.org/wiki/Snake_(video_game_genre). (Accessed on 09/13/2020).

[3] Rehman Butt. *Performance Comparison of AI Algorithms: Anytime Algo- rithms*. 2008.

[4] Rehman Butt and Stefan J Johansson. Where do we go now? anytime algo- rithms for path planning. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 248–255, 2009.

[5] Murray S Campbell and T. Anthony Marsland. A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20(4):347–367, 1983.

[6] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. Evo- lutionary algorithms for solving multi-objective problems. 5, 2007.

[7] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *In- ternational Journal of Computer Science and Network Security*, 11(1):125–130,2011.

[8] Rina Dechter and Itay Meiri. Experimental evaluation of preprocessing algo- rithms for constraint satisfaction problems. *Artificial Intelligence*, 68(2):211– 241, 1994.

[9] R Gayathri. Comparative analysis of various uninformed searching algorithms in ai, 2019.

[10] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *ICML*, pages 327–334. Citeseer, 2000.

[11] Carla P Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff. Randomizationin backtrack search: Exploiting heavy-tailed profiles for solving hard schedulingproblems. pages 208–213, 1998.

[12] Marcus Östergren Göransson. *Minimax Based Kalaha AI*. 2013.

[13] Wojciech Jaśkowski, Krzysztof Krawiec, and Bartosz Wieloch. Evolving strategy for a probabilistic game of imperfect information using genetic programming. *Genetic Programming and Evolvable Machines*, 9(4):281–294, 2008.

[14] Daniel Johnson and Janet Wiles. Computer games with intelligence. In *10th IEEE International Conference on Fuzzy Systems.(Cat. No. 01CH37297)*, vol- ume 3, pages 1355–1358. IEEE, 2001.

[15] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[16] Richard E Korf. *Real-time heuristic search*, volume 42. Elsevier, 1990.

[17] Richard E Korf. Improved limited discrepancy search. pages 286–291, 1996.

[18] Daniel R Kunkle. Solving the 8 puzzle in a minimum number of moves: An application of the a* algorithm. *Introduction to Artificial Intelligence*, 2001.

[19] Alberto Martelli. On the complexity of admissible search algorithms. *Artificial Intelligence*, 8(1):1–13, 1977.