International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

Algorithmic Diversity in Maze Generation Comparative Study of Backtracking, Kruskal's, Prim's, and Eller's Algorithms

Ivan Cahyakusuma [1], Wirawan Istiono*[2]

Submitted: 29/01/2024 **Revised:** 07/03/2024 **Accepted:** 15/03/2024

Abstract*:* The creation of content using Procedural Content Generation or PCG is a common occurrence in game development. The utilization of PCG can reduce game development costs and provide players with unique gaming experiences in each session. Among the various existing PCG algorithms, each algorithm generates content with different levels of complexity. This research aims to compare maps generated by the Backtracking Algorithm, Kruskal's Algorithm, Prim's Algorithm, and Eller's Algorithm. Each algorithm will create maze maps of sizes 5x5, 10x10, and 15x15, and their completion times will be measured using the A-Star maze-solver algorithm. Based on the results of this study, mazes created by the Backtracking algorithm are the most complex, with an average completion time of 2.3803 seconds, which is 14.82% longer than the mazes generated by the Eller algorithm. The mazes created by the Eller algorithm come in second as the most complex, with an average completion time of 2.0729 seconds, which is 4.27% longer than those generated by the Kruskal algorithm. The mazes created by the Kruskal algorithm rank third in complexity, with an average completion time of 1.988 seconds, which is 31.84% longer than the ones generated by the Prim algorithm. Lastly, the mazes created by the Prim algorithm are the least complex, with an average completion time of 1.5078 seconds.

Keywords: *Backtracking Algorithm, Eller's Algorithm, Kruskal's Algorithm, Maze, Prim's Algorithm*

1. Introduction

Video games have been rapidly growing as a form of entertainment in recent years. With easy access to the internet and an increasing number of devices capable of playing games, their distribution has expanded rapidly. According to data from We Are Social, a company specializing in internet, social media, and data trends, out of 4.66 billion internet users worldwide, 3.77 billion are gamers [1], [2].

Playing video games involves various tasks and challenges, such as requiring agility and precision in pressing inputs, logical thinking and problem-solving, or tasks related to strategic planning [3]. Though social research on the appeal of video games is still limited, there are indications that the satisfaction derived from playing games is linked to successfully completing these tasks and challenges [4], [5].

However, many games tend to offer the same content when played repeatedly, leading to reduced player satisfaction after multiple playthroughs. In modern game development, the effort and time required to create game content represent a significant portion of the development costs and time. Some popular games, like Minecraft developed by Mojang, utilize Procedural Content Generation (PCG) in creating levels and other content such as in-game items [6].

PCG is a method used to generate game content procedurally. It not only optimizes the game development process and reduces development costs, but also enhances replayability [7]. One of the applications of PCG is in maze generation. Mazes are valuable in level and game design as content that serves both as physical elements within the level and as puzzles for players. Game designers aim to create a well-balanced difficulty curve for their players. If a game designer desires mazes in their game and different maze algorithms produce mazes of varying complexity, knowing which maze algorithm aligns with the game's desired difficulty curve is crucial [8].

The first limitation of this research is that the maze created is a two-dimensional grid maze with four directions, and the level of maze complexity is determined by calculating the time required to solve the maze using the A-Star algorithm. The objective of this research, after successfully implementing the Backtracking, Kruskal, Prim, and Eller algorithms in creating levels for the maze game, is to determine the comparison of maze level complexities generated by the Backtracking, Kruskal, Prim, and Eller algorithms when solved using the A-Star algorithm.

1,2 Universitas Multimedia Nusantara of Informatics Department, Scientia Boulevard, Curug Sangereng, Kelapa. Dua, Tangerang, Banten 15810, Indonesia
2 ORCID ID: 0000-0001-9568-1539
** Corresponding Author Email: wirawan.istiono@umn.ac.id*

2. Materials & Methods

2.1. Backtracking algorithm

Backtracking in maze generation is a randomized version of the Depth-First Search algorithm. This algorithm randomly selects a cell until there are no neighboring cells left to be chosen. Once there are no available neighboring cells, the algorithm backtracks along the path until it finds a cell that still has unvisited neighboring cells [9]. This process of backtracking is repeated continuously until all cells have been selected. The steps involved in the Backtracking Algorithm are as follows [10], [11], [12]:

- Randomly select a starting cell.
- Randomly choose an adjacent cell that has not been visited before. Connect the chosen cell with the previous cell.
- If all adjacent cells have been visited, backtrack to the previous cell until reaching a cell with unvisited neighbors and repeat step two.
- The algorithm ends when the selected cell backtracks to the starting point, and there are no unvisited neighboring cells left.

2.2. Kruskal's Algorithm

The Kruskal's algorithm was developed by mathematician and computer scientist Joseph Kruskal in 1956 to create minimal spanning trees. In a graph or grid, where each connecting path between cells is assigned a cost, the Kruskal's algorithm repeatedly selects the connecting paths between 2 cells based on their costs to connect all cells without forming a loop [13]. The steps involved in Kruskal's Algorithm for creating a maze are as follows [14], [15]:

- Assign letters to all cells to indicate their respective sets.
- Randomly choose one wall between two cells. If the two cells belong to different sets, merge those two sets into one and connect the two cells.
- Repeat step two until all cells are part of the same set.

2.3. Prim's Algorithm

The Prim's algorithm was first developed in 1930 by the Czech mathematician named Vojtěch Jarník, but it received its name from Robert C. Prim, a computer scientist who rediscovered it independently in 1957. In a graph or grid, where each connecting path between cells is assigned a cost, the Prim's algorithm randomly selects one cell and chooses connecting paths based on the cost of connecting neighboring cells to the selected cell [16]. Then, the Prim's algorithm repeats the process of selecting connecting paths from all previously selected cells until all cells are interconnected. The steps involved in Prim's Algorithm for creating a maze are as follows [17]:

- Randomly choose a starting cell, which becomes part of the maze set.
- Mark all neighboring cells of the maze set as border cells.
- Randomly select one border cell and connect it to an adjacent cell in the maze set. This newly selected cell becomes part of the maze set.
- Repeat steps two and three until all cells are included in the maze set.

2.4. Eller's Algorithm

The Eller's algorithm was discovered by Marlin Eller in 1982. The Eller's algorithm works by creating rows one by one, with each cell belonging to a different set. Then, neighboring cells are randomly connected to form one set. Next, at least one cell from each existing set is randomly selected to be connected to cells in the new row below [18]. On the next row, all cells without a set will receive a new set. The process of merging sets in each row is repeated continuously until the desired grid size is reached, and on the last row, all sets will be connected. The steps involved in Eller's Algorithm for creating a maze are as follows [19]:

- Assign letters to all cells in the first row to indicate their respective sets.
- Randomly establish connections between adjacent cells, resulting in the merger of the two connected sets into a single set.
- Randomly connect cells to cells in the next row; the connected cells will join the set of the cell above. Each set must connect at least one cell.
- Assign letters to cells in the new row that are not connected to different sets in the existing row.
- Repeat steps two, three, and four until reaching the last row.
- On the last row, connect all cells in different sets, leaving only one set throughout the entire maze.

3. Methodology

The research methodology used in comparing the maze levels created by the Backtracking, Kruskal, Prim, and Eller algorithms in the maze game is as follows. The first step is literature review, where the researcher conducts theoretical research on mazes and the algorithms to be used in maze generation. The sought-after theories include Maze, Procedural Content Generation, Backtracking Algorithm, Kruskal's Algorithm, Prim's Algorithm, Eller's Algorithm, and A-Star Algorithm. Next is the design phase, where the program design for maze comparison, the workflow for maze generation using each algorithm, and the workflow for the A-Star algorithm used to compare maze complexity are established [20]. The subsequent step is implementation, where a program is developed based on the designed algorithms for maze generation, using the Unity game engine.

Then, the next step is testing, where the mazes created by the algorithms are tested using the A-Star algorithm. The A-Star algorithm is employed to solve mazes of various sizes generated by each algorithm. The A-Star algorithm measures the number of steps and time required to solve each maze [21]. This is followed by the evaluation phase, where a comparison is made between the number of steps and time obtained from the previous testing stage. The results of testing each algorithm are compared to measure the relative complexity of the mazes created by each algorithm.

The final step is report writing, where a detailed report is written, describing the entire research process and the findings obtained from this study. In this research, a program is designed to serve as a platform for algorithms to create a maze. Within this program, users can choose the size of the maze to be generated and the algorithm to be used in maze generation. Users can also observe the maze-solving process using the A-Star algorithm and obtain information about the time required to solve the maze created by each algorithm.

In Fig. 1, the workflow of the Maze Generation scene is depicted, which appears after selecting an algorithm from the main menu. Within the maze generation scene, the program displays three buttons: Menu, Generate Maze, and Solve Maze. When the Generate Maze button is pressed, the program executes the algorithm previously chosen in the main menu to create a maze and displays it on the screen. When the Solve Maze button is pressed, the program runs the A-Star algorithm to solve the existing maze and displays the solved maze and the time taken to solve it on the screen.

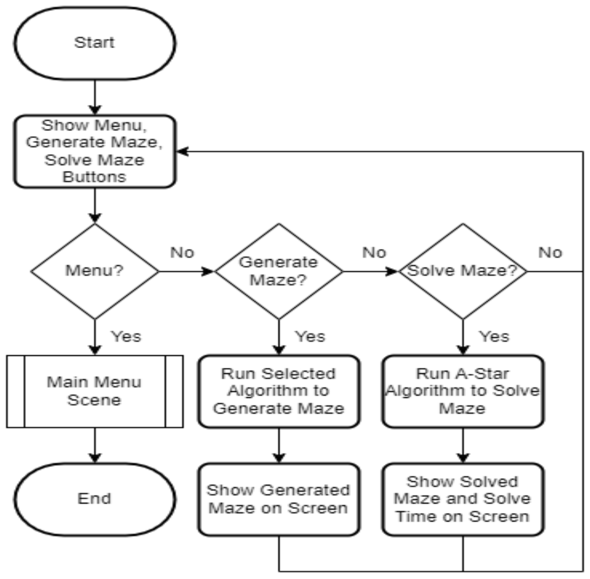

Start
Show Menu, Generate Maze, Solve Maze Buttons
Menu?
No
Generate Maze?
No
Solve Maze?
No
Yes
Yes
Yes
Main Menu Scene
Run Selected Algorithm to Generate Maze
Run A-Star Algorithm to Solve Maze
End
Show Generated Maze on Screen
Show Solved Maze and Solve Time on Screen

Fig. 1. Flowchart Maze Generation Scene

Fig. 2 depicts the sequential steps of the Backtracking method. Prior to the commencement of the algorithm, the computer generates cells for the maze according to the specified maze size. To begin with, the method employs a random selection process to designate one cell as the Start cell. The adjacent cells to the selected cell are examined, and any unvisited cells among them are identified as potential cells for visitation. From the candidate cells, the algorithm randomly chooses one cell to visit. This cell is then connected to the previous cell, and the previous cell is noted as the source. This visiting process is repeated until the algorithm reaches a cell that has no more unvisited target cells. If there are no more target cells to visit, the algorithm will backtrack towards the source cell until it reaches a cell with unvisited target cells and then perform the visiting process again. If the Backtracking algorithm returns to the Start cell and there are no more cells that can be visited, it means that all cells in the maze are connected, and the algorithm is complete.

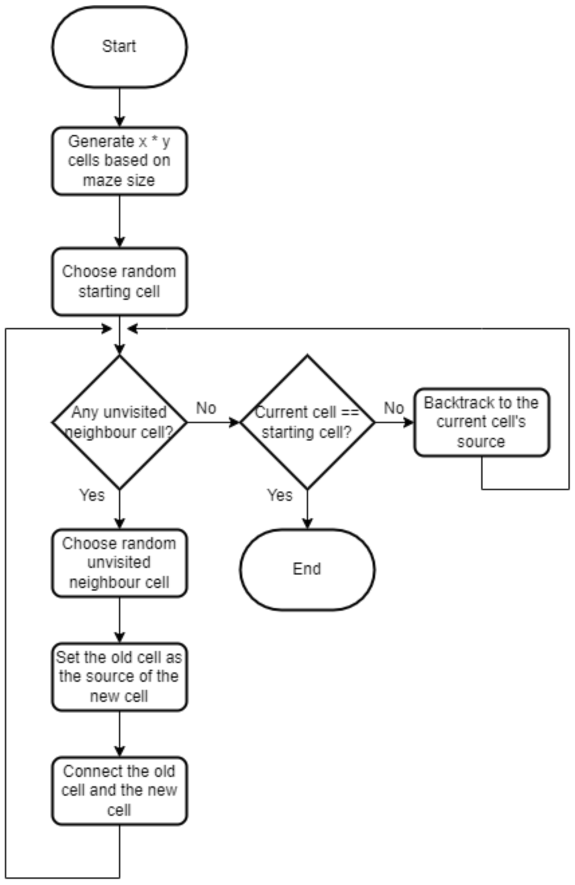

Start
Generate x * y cells based on maze size
Choose random starting cell
Any unvisited neighbour cell?
No
Current cell == starting cell?
No
Backtrack to the current cell's source
Yes
Yes
Choose random unvisited neighbour cell
End
Set the old cell as the source of the new cell
Connect the old cell and the new cell

Fig. 2. Modified Backtracking Algorithm Flowchart For Maze Generation

Fig. 3 illustrates the workflow of the Kruskal algorithm. Before the algorithm begins, the program creates cells for the maze based on the desired maze size. Firstly, this algorithm assigns different set numbers to all cells and keeps track of the number of sets present. The algorithm then randomly selects two adjacent cells. If these two cells belong to different sets, the algorithm connects them and merges the two sets into one. This random selection and merging process are repeated until there is only one set remaining throughout the entire maze, indicating that all cells in the maze are connected, and the algorithm is complete.

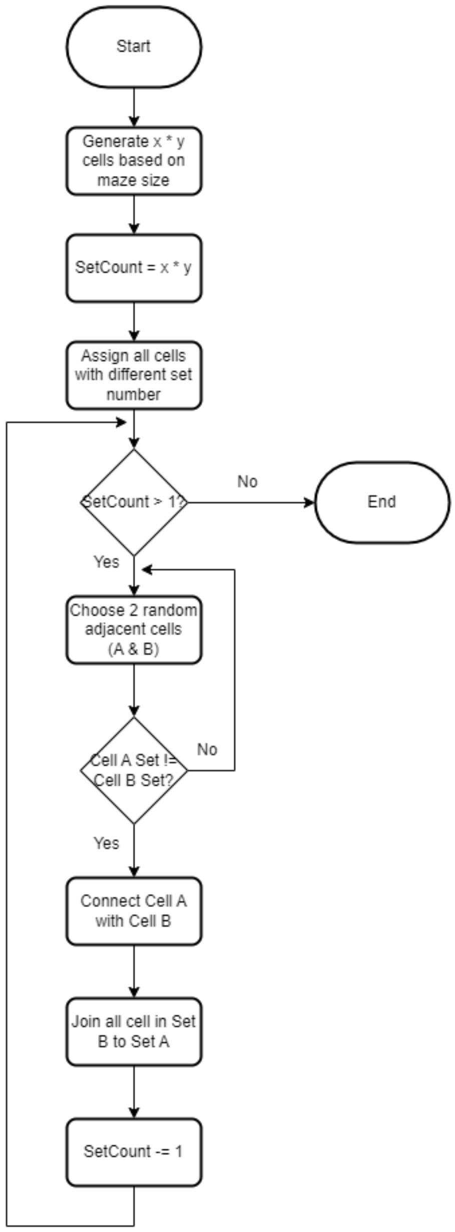

Start
Generate x * y cells based on maze size
SetCount = x * y
Assign all cells with different set number
SetCount > 1?
No
End
Yes
Choose 2 random adjacent cells (A & B)
Cell A Set != Cell B Set?
No
Yes
Connect Cell A with Cell B
Join all cell in Set B to Set A
SetCount -= 1

Fig. 3. Modified Kruskal Algorithm Flowchart For Maze Generation

The flowchart in Fig. 4 illustrates the workflow of the Prim algorithm. Before the algorithm starts, the program creates cells for the maze based on the desired maze size. Firstly, this algorithm designates all cells as outside cells. The algorithm then randomly selects one cell to be the first inside cell. All outside cells adjacent to any inside cell will be converted into edge cells. The algorithm selects one existing edge cell to connect with an inside cell and transforms that edge cell into an inside cell. This process of changing and merging edge cells is repeated until all edge cells are used up, indicating that all cells in the maze are connected, and the algorithm is complete.

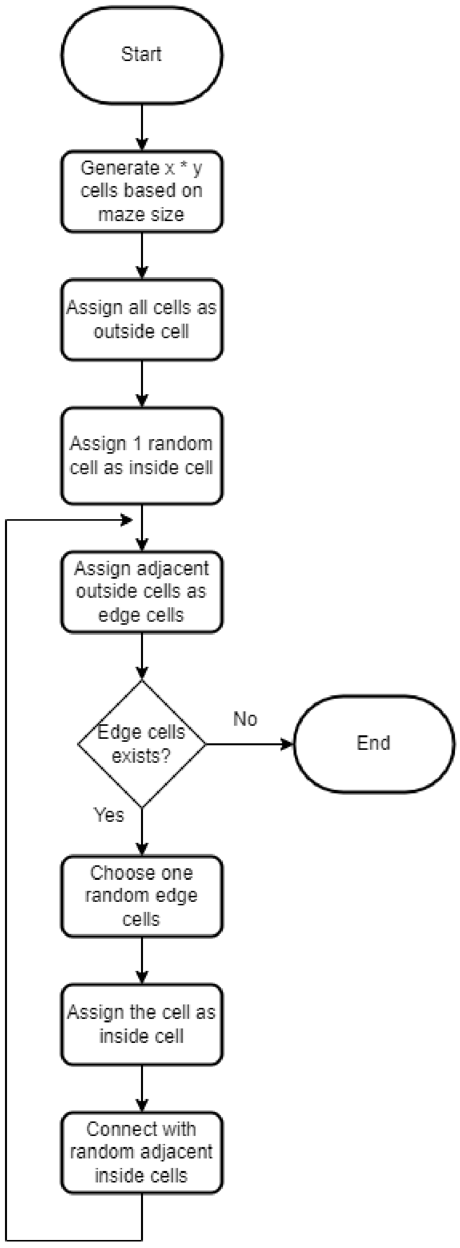

Start
Generate x * y cells based on maze size
Assign all cells as outside cell
Assign 1 random cell as inside cell
Assign adjacent outside cells as edge cells
Edge cells exists?
No
End
Yes
Choose one random edge cells
Assign the cell as inside cell
Connect with random adjacent inside cells

Fig. 4. Modified Prim's Algorithm Flowchart for Maze Generation

Fig. 4 illustrates the workflow of the Eller algorithm. Before the algorithm starts, the program creates a maze based on the desired maze size. Firstly, the algorithm initiates the process from the first row. It assigns different set numbers to cells in this row. Then, random connections are made between two cells with different sets, and those sets are merged. For each existing set, one cell is connected to a cell in the next row. This process of assigning sets, connecting cells, and merging sets is repeated until reaching one row before the last row. In the last row, sets are continuously merged until only one set remains, indicating that all cells in the maze are connected, and the algorithm is complete.

4. Results & Discussion

An application was designed to visualize the comparison of the backtracking, Kruskal, Prim, and Eller algorithms, based on the completed study methods and flowchart design.

In this research, mazes created by the Backtracking, Kruskal, Eller, and Prim algorithms will be tested in various sizes. The tested maze specifications include 30 mazes of size 5x5, 30 mazes of size 10x10, and 30 mazes of size 15x15 for each algorithm. The variables taken into consideration in these tests are as follows:

- No: The sequence number of the maze.
- Maze Node: The total number of nodes present in the maze.
- Maze Length: The exact length of the path from the starting point to the ending point.
- Total Step: The number of steps taken by the A-Star algorithm to solve the maze.
- Solve Time: The amount of time in seconds taken by the A-Star algorithm to solve the maze, measured in seconds.

The program is developed by using C# programming language with Unity game engine. The mazes cells and passages are shown with simple rectangle sprites with the start cell colored green and end cell colored red as shown in Fig. 5.

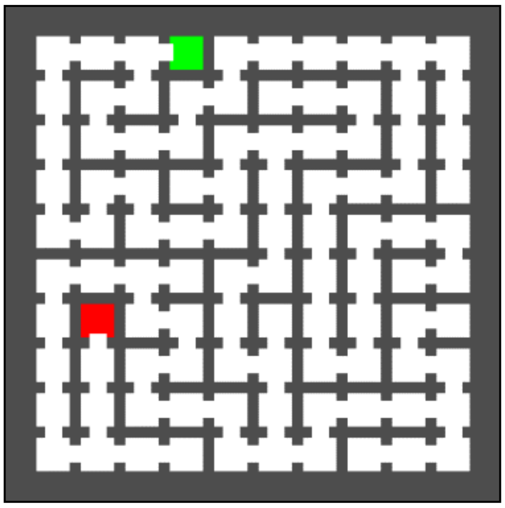


Fig. 5. An example of maze generated by Backtracking Algorithm

The labyrinth solution procedure using the A-Star Algorithm is depicted in Fig. 6. Each cell traversed by the algorithm will be labeled with a white color and a numerical value indicating its cost. The designated route from the starting point to the destination will be indicated by a color spectrum ranging from green to red.

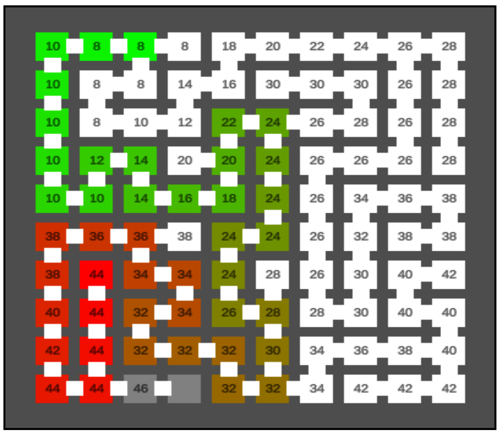
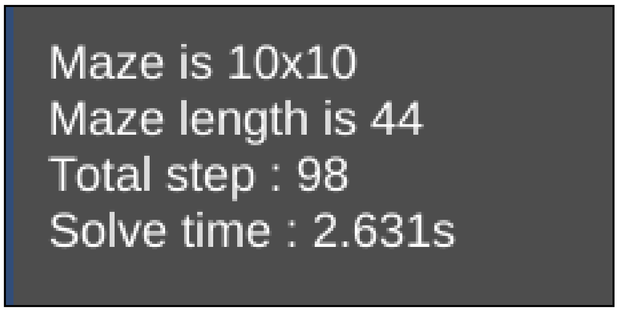


Fig. 6. An example of maze traversed by A-Star Algorithm

After the A-Star Algorithm finishes solving the maze, the data of the maze will be recorded and shown in the screen as shown in Fig. 7.



Maze is 10x10
Maze length is 44
Total step : 98
Solve time : 2.631s

Fig. 7. Recorded data shown after maze is successfully solved

The comparison of each algorithms is done by generating 30 mazes of each size of each algorithms. The average solve time on each algorithm is then calculated and compared to other algorithm in the same size category. The average results of the mazes generated is shown in Table 1.

Table 1. Average results of 30 mazes of each size of each algorithms

Maze	Average Maze Length	Average Total Step	Average Solve Time
Backtracking 5x5	17.9	23.6333	0.2299s
Backtracking 10x10	61.3	98.5333	1.0014s
Backtracking 15x15	114.467	221.233	2.3803s
Kruskal 5x5	10.7	18.3	0.1719s
Kruskal 10x10	25.467	78.2	0.7854s
Kruskal 15x15	46.333	188.4	1.9880s
Prim 5x5	9.7333	18.2666	0.1721s

Prim 10x10	21.633	67.9	0.6840s
Prim 15x15	32.167	141.733	1.5078s
Eller 5x5	12.633	21.4333	0.2041s
Eller 10x10	30.133	91.2333	0.9223s
Eller 15x15	47.766	194.033	2.0729s

Fig. 8 displays the comparative analysis of maze-solving algorithms in the 15x15 size category.

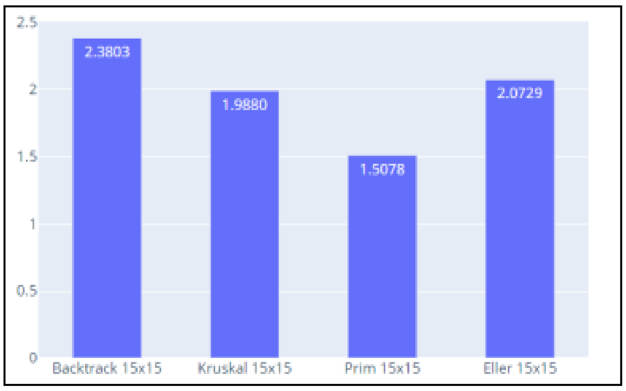


Fig. 8. Average maze solve time of each algorithm in the 15x15 size category

By analyzing the average time it takes for each algorithm to solve a maze, we may determine that the ranking of maze difficulty is as follows: Backtracking, Eller's, Kruskal's, and Prim's. On average, solving backtracking mazes required 14.82% more time compared to Eller's mazes. On average, Eller's mazes required 4.27% longer time to solve compared to Kruskal's mazes. Kruskal's mazes required, on average, 31.84% additional time to solve compared to Prim's. The disparity in average solve time is directly proportional to the size of the mazes.

5. Conclusion

Based on the conducted research, the implementation of the Backtracking, Kruskal, Prim, and Eller algorithms has been successfully accomplished in generating maze levels. Based on the analysis of the complexity of the mazes created by each algorithm, it can be concluded that concerning the maze-solving time with the A-Star algorithm for a size of 15x15, mazes generated by the Backtracking algorithm are the most complex, with an average time of 2.3803 seconds, which is 14.82% greater than the mazes generated by the Eller algorithm. The mazes created by the Eller algorithm are the second most complex, with an average time of 2.0729 seconds, which is 4.27% greater than the mazes generated by the Kruskal algorithm. The mazes created by the Kruskal algorithm are the third most complex, with an average time of 1.988 seconds, which is 31.84% greater than the mazes generated by the Prim algorithm. The mazes created by the Prim algorithm are the least complex, with an average time of 1.5078 seconds

Acknowledgements

Thank you to Universitas Multimedia Nusantara in Indonesia for providing a space for academics to conduct this journal research. Hopefully, this research will contribute significantly to the growth of technology in Indonesia.

Author contributions

Ivan Cahyakusuma: Task executor, Methodology, Software developer, Informatics department **Wirawan Istiono:** Research mentor, Corresponding research author, editor's research report, Informatics department.

Conflicts of interest

I declare that there is no conflict of interest in this research.

References

[1] I. S.-E. Otobe, “Innovations in the video game industry,” *Innov. Syst. Process. Asia*, no. 94, pp. 1–34, 2007, [Online]. Available: http://asia.stanford.edu/events/fall07/slides/otobe.pdf

[2] M.-P. Määttä, “Growth and Future of Video Game Industry,” in *Proceedings of the 2022 International Conference on Economics, Smart Finance and Contemporary Trade*, 2019, p. 31. doi: 10.2991/978-94-6463-052-7_9.

[3] B. Chaarani, J. Ortigara, D. Yuan, H. Loso, A. Potter, and H. P. Garavan, “Association of Video Gaming with Cognitive Performance among Children,” *JAMA Netw. Open*, vol. 5, no. 10, p. E2235721, 2022, doi: 10.1001/jamanetworkopen.2022.35721.

[4] C. S. Green and D. Bavelier, “Action video game training for cognitive enhancement,” *Curr. Opin. Behav. Sci.*, vol. 4, pp. 103–108, 2015, doi: 10.1016/j.cobeha.2015.04.012.

[5] I. Granic, A. Lobel, and R. C. M. E. Engels, “The benefits of playing video games,” *Am. Psychol.*, vol. 69, no. 1, pp. 66–78, 2014, doi: 10.1037/a0034857.

[6] M. Beukman, C. W. Cleghorn, and S. James, “Procedural content generation using neuroevolution and novelty search for diverse video game levels,” *GECCO 2022 - Proc. 2022 Genet. Evol. Comput. Conf.*, pp. 1028–1037, 2022, doi: 10.1145/3512290.3528701.

[7] J. Togelius *et al.*, “Procedural Content Generation : Goals, Challenges and Actionable Steps,” *Artif. Comput. Intell. Games*, vol. 6, pp. 61–75, 2013, [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4351%5Cnhttp://drops.dagstuhl.de/opus/volltexte/2013/4336/

[8] A. Chia, “The artist and the automaton in digital game

production," *Convergence*, vol. 28, no. 2, pp. 389–412, 2022, doi: 10.1177/13548565221076434.

[9] B. A. Hassan and T. A. Rashid, "Operational framework for recent advances in backtracking search optimisation algorithm: A systematic review and performance evaluation," *Appl. Math. Comput.*, vol. 370, 2020, doi: 10.1016/j.amc.2019.124919.

[10] F. Rossi, P. van Beek, and T. Walsh, "Handbook of Constraint Programming Introduction," *Handb. Constraint Program.*, pp. 1–955, 2006.

[11] D. C. Schmidt and L. E. Druffel, "A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices," *J. ACM*, vol. 23, no. 3, pp. 433–445, 1976, doi: 10.1145/321958.321963.

[12] G. Kondrak and P. Van Beek, "A theoretical evaluation of selected backtracking algorithms," *Artif. Intell.*, vol. 89, no. 1–2, pp. 365–387, 1997, doi: 10.1016/s0004-3702(96)00027-6.

[13] H. Li, Q. Xia, and Y. Wang, "Research and Improvement of Kruskal Algorithm," *J. Comput. Commun.*, vol. 05, no. 12, pp. 63–69, 2017, doi: 10.4236/jcc.2017.512007.

[14] S. Marković, J. Novakovic, and A. Marković, "Generating and solving of the maze by using Kruskal's and Floyd's algorithm," *Int. Sci. Conf. "UNITECH 2019" – Gabrovo*, no. November, pp. 315–319, 2019.

[15] D. Jakus, R. Čađenović, J. Vasilj, and P. Sarajčev, "Optimal reconfiguration of distribution networks using hybrid heuristic-genetic algorithm," *Energies*, vol. 13, no. 7, 2020, doi: 10.3390/en13071544.

[16] Wamiliana, M. Usman, Warsono, Warsito, and J. I. Daoud, "Using modification of Prim's algorithm and GNU Octave and to solve the multiperiods installation problem," *IIUM Eng. J.*, vol. 21, no. 1, pp. 100–112, 2020, doi: 10.31436/iiumej.v21i1.1088.

[17] M. Iqbal, A. P. U. Siahaan, N. Elizabeth, and D. Purwanto, "Prim's Algorithm for Optimizing Fiber Optic Trajectory Planning," *Int. J. Sci. Res. Sci. Technol.*, vol. 3, no. September, pp. 504–509, 2017, [Online]. Available: https://www.researchgate.net/publication/319349574

[18] C. Science, "Fundamentals of Maze Generation," *Fundam. Program. Comput. Sci.*, pp. 1–6.

[19] P. A. Newell, J. Aycock, and K. M. Biittner, "Still Entombed After All These Years: The continuing twists and turns of a maze game," *Internet Archaeol.*, no. 60, 2022, doi: 10.11141/ia.59.3.

[20] O. R. Chandra and W. Istiono, "A-star Optimization with Heap-sort Algorithm on NPC Character," *Indian J. Sci. Technol.*, vol. 15, no. 35, pp. 1722–1731, 2022, doi: 10.17485/ijst/v15i35.857.

[21] G. T. Kumala and W. Istiono, "Comparison of Flow Field and A-Star Algorithm for Pathfinding in Tower Defense Game," *Int. J. Multidiscip. Res. Anal.*, vol. 5, no. 9, pp. 2445–2453, 2022, doi: 10.47191/ijmra/v5-i9-20.