Jurnal CoreIT, Vol.9, No.1, June 2023
ISSN 2460-738X (Print)
ISSN 2599-3321 (Online)

Pathfinding Solving in Maze Game using Backtracking Algorithms

Tegar Arifin Prasetyo[1*], Rudy Chandra[1], Berliana Simamora[2], Michael Joseph Christian[2], Agus Rokyanto Silaban[2], and Meyliza Veronica Siregar[2]
[1]Information Technology Department, Faculty of Vocational Studies, Institut Teknologi Del, Indonesia
[2] Software Engineering Technology Department, Institut Teknologi Del, Indoneisa
tegar.prasetyo@del.ac.id (*Corresponding Author), rudychandra@del.ac.id, if419016@students.del.ac.id, if419005@students.del.ac.id, if419045@students.del.ac.id, if419058@students.del.ac.id

Abstract.
Games have become a highly sought-after form of entertainment, offering not just leisure, but also the opportunity to sharpen critical thinking skills. A game that contains elements of artificial intelligence requires algorithms for its implementation. One type of game is the maze game, where players are required to find a way out of the maze. The backtracking algorithm was employed to solve this challenge by tracing possible paths recursively. If the path leads to a dead end, it will be backtracked and another path will be explored. The algorithm saves all solutions, continuing the search until the final solution is found. By implementing the backtracking algorithm, the completion time for a single attempt in the maze game was impressively reduced to just 2.16 seconds.

Keywords: Backtracking Algorithm, Maze Game, and Recursive

Received November 2022 / **Revised** May 2023 / **Accepted** June 2023

INTRODUCTION

Games have become a highly sought-after source of entertainment, appealing to all demographics, particularly the youth. The gaming industry is experiencing significant growth, with developers creating increasingly captivating games to attract players. Not only do these games provide entertainment, but they also challenge players to sharpen their problem-solving skills in order to win.The modern computer gaming industry continues to expand and become more intricate each year, with advancements in both map size and the number of units within the industry [1], [2]. A maze-themed game can be just as engaging and mentally stimulating for players, challenging them to use their problem-solving skills. One common issue in games is finding effective pathfinding solutions for non-player characters, which are crucial for their movement and behavior, such as in tower games [3], car racing [4], and educational games [5], and so on.

The implementation of a game that incorporates artificial intelligence elements demands the use of algorithms [6]. The maze game is an example of a game that incorporates artificial intelligence elements. It features barriers or walls that restrict the player's path and has a starting point and a final goal. When players reach a dead end, they must turn around and explore other paths. With multiple potential solutions, this game offers a challenging experience for players. Several algorithms used to solve maze games are closely related to graph theory and are designed to find the shortest path solution from the starting point to the destination, even when multiple solutions exist [7], [8]. The shortest path search problem in games has been addressed using a variety of search algorithms, including the efficient A* search algorithm and the comprehensive Bread-First Search algorithm [9]. Previous studies have utilized the implementation of the Bread-First Search (BFS) and A* algorithms to tackle the pathfinding challenge in maze games [9], [10]. The research has shown that both the Bread-First Search (BFS) and A* algorithms are capable of solving the shortest path problem without encountering bottlenecks, as long as the generator node does not move towards a dead end. The A* algorithm can be utilized in games that are designed with artificial intelligence concepts, such as the Goat Foraging Game [11]. The A* algorithm in the Goat Foraging Game utilizes artificial intelligence to determine the shortest distance between the enemy and the goat. Other studies have demonstrated that the brute force algorithm can solve problems in the shortest amount of time [12]. The Brute Force Algorithm involves evaluating all possible solutions to find the correct one, which results in a longer time frame compared to other algorithms as it thoroughly checks every possible outcome, regardless of its feasibility.

DOI: 10.24014/coreit.v9i1.17109

In this research, we aim to find a potential solution by implementing the backtracking algorithm. This algorithm operates by constructing solutions incrementally and then discarding those that are unnecessary. It employs a state-space tree to determine if a solution is valid, and if it is, the process is repeated until the final solution is found. If the solution is invalid, it will be discarded. The backtracking algorithm's unique advantage is its ability to ignore or eliminate invalid solutions efficiently. The objective of this research is to delve into the workings and processes of the backtracking algorithm applied to pathfinding in maze games. It is hoped that the findings will shed new light on the functioning of the backtracking algorithm and contribute to a deeper understanding of its application.

METHODS

The right algorithm to find a way out of the maze is the backtracking algorithm. This algorithm tries a path until it reaches a dead end, then performs the previous step (backwards) until it finds another path, then repeats the path again. To describe a backtracking algorithm, divide the path into a series of steps. One of the steps is to move the unit cell in a certain direction. The directions that can be passed are: up, down, left, and right. A backtracking algorithm is a recursive algorithm that aims to solve a given problem by testing all possible paths to a solution until a solution is found [13]. Each time a path is tested, if no solution is found, the algorithm will return to test another path, and so on until a solution is found or all paths have been tested. In the game Maze, the backtracking algorithm scenario occurs when attempting to find a way out of a maze. Whenever the algorithm reaches a dead end, it will back off and try another path until it finds a way out or all paths have been explored. In general, Backtracking algorithm can be presented in Figure 1.

```
while haven't reached solution do
   if there is a right direction
      so we never move to cell
      in that direction
   then
      move one step in
      that direction
   else
      backtrack steps until there
     is a direction as mentioned
     above
   endif
end while
```

Figure 1. General Backtracking Algorithm Representation

Backtracking algorithm for the Maze problem is the function parameter is M, then the maze is saved (the maze can be implemented as a 0/1 matrix, paths are represented as rows of 0s and walls represented as rows of 1).

```
function
SolveMaze(input M:Maze)->boolean {true if solution found, false if Not}
Declaration
    direction: integer
Algorithm:
if solution found
   return true
else
   for direction of movement (up, down, left, right) do
      move(M, direction)
          {move one step (one cell) in that direction
      if SolveMaze(M) then
          return true
      else
          unmove(M, direction)
          {backtrack}
      endif
   endfor
   return false
   {all direction have been tried, but still deadlocked, then the conclusion:
    no solution}
endif
{up=1, down=2, left=3, right=4}
```

Figure 2. Maze Game Backtracking Algorithm Representation

Based on figure 2, if the recursive call to SolveMaze (M) is feasible, it means that the displacements made lead to the solution. Therefore, the resulting maze image is displayed on the screen. The resulting step motions are printed in reverse order when the recursive call returns, causing a minor problem. To fix this, you need to save the step motion on the stack and print the entire step after the SolveMaze call instead of printing it right away.

The backtracking algorithm for the Maze problem can be thought of as forming a state-space tree. The roots of the tree are the first maze, and the children are the mazes that result from moving one step away from the original maze. We use the programming language Python to simulate the implementation of the backtracking algorithm to solve path finding in the maze game. Python has many features that support functional programming and object-oriented programming [14]. Python is also simple and easy to learn. Python became popular for some reasons: it has a simple syntax that makes it easy to learn and easy to read; it is cross-platform; it supports multi-paradigm; it can be used for multiple tasks (software development and machine learning); it has many libraries; it is open source; and it has a large and active community [15].

RESULT AND DISCUSSION

Maze Game is a puzzle game where players will be set at one point as a start and are required to find a path to get out of the maze at a specified point. To get out of the maze, the player will be confronted with several paths, some of which are dead ends. So to find a solution to this problem, a backtracking algorithm is implemented.

The backtracking algorithm will examine some paths taken to determine whether or not they are qualified.The backtracking algorithm will only select those paths that lead to a solution without checking all possibilities. In other words, paths that do not qualify or do not lead to a solution will not be considered again. Figure 3 show user interface for the backtracking algorithm implementation of the maze game.

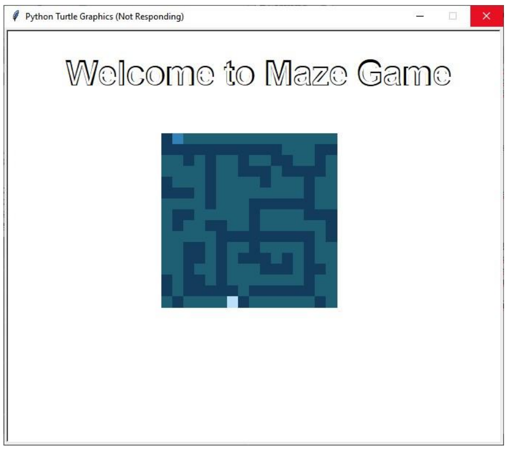

Figure 3. Maze Game UI Implementation

We can find the solution using the state space tree. State space tree roles as follow:

a. Starting point has been set at the end second track.
b. Goal point is set at the bottom in light blue.
c. Paths that can be traversed are dark blue.
d. Paths that can be passed are vertical and horizontal lanes, so players cannot cross paths crosswise or diagonally.

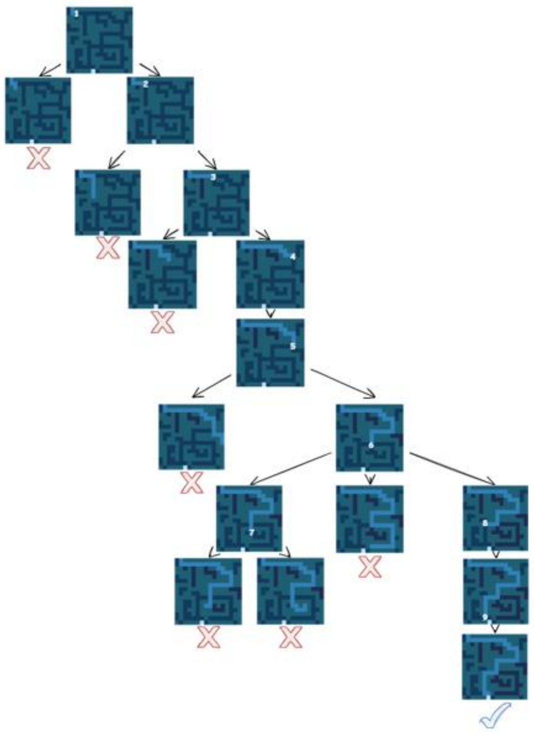

Figure 4. Solution Using State Space Tree

Figure 4 shows the implementation of a backtracking algorithm to solve a maze game problem using a state space tree. The first intersection in the game maze is the root. There are two intersections that lead to the right and down. The downward crossing is not promising because it is a dead end. so that a backtrack will be carried out to the second intersection.

At the second intersection there are two intersections, there are down and to right. Downward crossroad is not promising because of a dead end. So that a backtrack will be carried out towards the third intersection. At the third intersection there are two intersections, there are down and to right. Downward crossing is not promising as it is a dead end. So that a backtrack will be carried out to the fourth intersection. There are two intersections at fourth intersection, there are down and to right. The intersection to right is not passed because it does not lead to solution. So it will continue to fifth intersection. At the fifth intersection there are two intersections, there are to left and right. Righteous crossing is not promising because it is a dead end. So that a backtrack will be carried out to sixth intersection. At the sixth intersection there are three intersections, there are to left, to down and to right. The crossing down promises and continues to the seventh intersection. The seventh intersection has 2 intersections, there are to left and to right. Left crossing is not promising because it is a dead end. So that a backtrack will be carried out towards the intersection to the right, this intersection is also not promising because it is a dead end. So that backtracking is carried out at the seventh and sixth intersections. Next at the sixth intersection trace back to intersection. This crossing is not promising because it is a dead end. So there will be a backtrack to the eighth intersection. At the eighth intersection there are two intersections, there are to up and to down. The upward crossing is not crossed because opposite direction of the final solution. So proceed to the ninth intersection. At the nine intersection there are two intersections, there are to left and to right. Intersection to left is not crossed because opposite to the direction of the final solution. So proceed to the right intersection. So that the path that can be passed to the final solution obtained are1-2-3-4-5-6-8-9.

Furthermore, the backtracking algorithm implementation uses the Python programming language.

```
import turtle
import random import randint
from turtle import Turtle, Screen
```

Figure 5. Library Used

Library in figure 5 is a graphical library used to create user interface of Maze Game. Turtle is a python library used to create canvas. Where in the canvas will later be able to draw user interface for the maze. Radiant is used to generate random numbers of integer type with a specified range.

```
def box(intDim):
   point.begin_fill()
   point.forward(intDim)
   point.left(90)
   point.forward(intDim)
   point.left(90)
   point.forward(intDim)
   point.left(90)
   point.forward(intDim)
   point.end_fill()
```

Figure 6. Path and Wall Arrangement

In figure 6 it is a function to draw paths and walls in the maze game using fill function. Point 0 refers to the right, 90 up, 180 left and 270 down.

```
def box(intDim):

palette = [“#133B5C”, “#1E5F74”, “#BBE1FA”,
“#3282B8”, “#1D2D50”]

maze =[[0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1)]
maze.append([0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0])
maze.append([1,1,0,1,0,1,1,0,1,1,0,0,1,1,0,1])
maze.append([1,1,1,1,0,1,1,0,0,1,0,0,0,0,0,1])
maze.append([0,1,1,1,0,1,1,1,1,0,1,1,1,0,1,1])
maze.append([0,0,0,1,0,1,1,1,1,1,1,1,1,0,1,1])
maze.append([1,1,1,1,0,1,1,1,0,0,0,0,0,0,1,1])
maze.append([1,1,1,1,0,1,1,1,0,0,0,0,0,0,1,1])
maze.append([1,0,0,1,1,1,1,1,0,1,1,1,1,0,0,0])
maze.append([1,0,1,1,0,0,1,1,0,1,1,1,1,1,1,0])
maze.append([1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,0])
maze.append([1,1,0,0,1,0,1,1,0,1,1,1,1,0,1,1])
maze.append([1,1,0,0,1,0,1,0,0,0,1,0,1,0,1,1])
maze.append([1,1,0,1,1,0,1,1,1,0,0,0,1,0,0,1])
maze.append([0,1,0,0,1,0,1,1,1,1,1,1,1,0,1,1])
maze.append([0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1])
maze.append([1,0,1,1,1,1,2,0,1,1,1,1,1,0,1,1])
```

Figure 7. Path and Wall in Maze Game

Figure 7 is code snippet for PixelArt saved in list. To determine paths and walls, 0 is path, 1 is wall and 2 is end point.

```
def drawMaze(maze):
 boxSize = 15
 point.penup()
 point.goto(-130,130)
 point.setheading(0)
 for i in range (0, len(maze)):
  for j in range (0, len(maze[i])):
    point.color(palette[maze[i][j]])
    box(boxSize)
    point.penup()
    point.forward(boxSize)
    point.pendown()
  point.setheading(270)
  point.penup()
  point.forward(boxSize)
  point.setheading(180)
  point.forward(boxSize*len(maze[i]))
  point.setheading(0)
point.pendown()
```

Figure 8. Starting Point Position Setting

Code snippet in figure 8 is the starting point position in upper left area of screen.

```
def exploreMaze(maze, row, col):
  if maze[row][col] == 2;
   return True
  elif maze[row][col] == 0;
   maze[row][col] = 3
   point.clear()
   drawMaze(maze)
   point.getscreen().update()
   if row<len(maze)-1:
    if exploreMaze(maze, row+1, col):
return True
```

Figure 9. Code Program Maze Game Using Backtracking

```
if row>0:
  if exploreMaze(maze, row-1, col):
    return True
 if col<len(maze[row])-1:
  if exploreMaze(maze, row, col+1):
    return True
 if col>0:
  if exploreMaze(maze, row, col-1):
    return True
 maze[row][col] = 4
 point.clear()
 drawMaze(maze)
 point.getscreen().update()
print("Backtrack")
```

Figure 10. Code Program Maze Game Using Backtracking

Figures 9 and 10 are backtracking/recursive functions to check all possible paths to the end point or exit of the maze. If an exit or destination is found, it returns true. If the path is empty then it will not be traced.

```
def text(message, x, y, size):
   FONT = (‘Arial’, size, ‘normal’)
   point.penup()
   point.goto(x,y)
point.write(message, align = “left”, font = FONT)
```

Figure 11. Setting for Messages

Code snippet in figure 11 is a code snippet to set style of message when you have finished exploring the maze path.

```
solved = exploreMaze(maze, 0, 1)
if solved:
   print(“Jalan Keluar dari Labirin berhasil Ditemukan”)
   text(“Jalan Keluar dari Labirin berhasil Ditemukan”, -250, -150, 20)
else:
   print(“Jalan Keluar dari Labirin Gagal Ditemukan”)
text(“Jalan Keluar dari Labirin Gagal Ditemukan”, -130, -150, 20)
```

Figure 12. Message if Solution Found

Code snippet from figure 12 is the code snippet for message when you have finished tracing path.

```
drawMaze(maze)
```

Figure 13. Maze Drawing Function

Figure 13 is a code snippet to describe the maze. Figure 14 is the output displayed in completing Maze Game using backtracking algorithm with the solution finding speed time is 2.16 seconds for one trial.

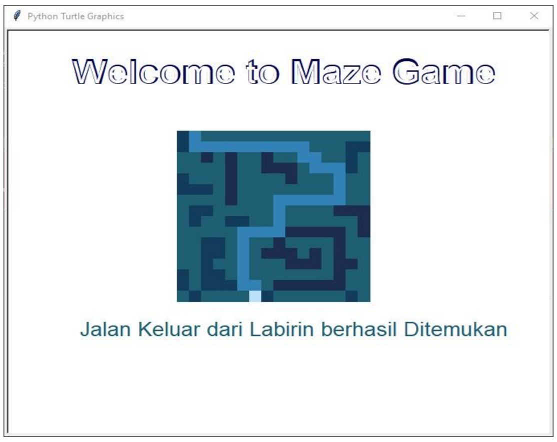

Figure 14. Solution Using State Space Tree

CONCLUSION

The maze game's challenge of finding a path to reach the end point can be effectively addressed through the utilization of the backtracking algorithm. The implementation of the backtracking algorithm is straightforward, and its results can be seen in the player's ability to reach the goal and exit the maze.

However, there is scope for further improvement, such as incorporating artificial intelligence algorithms and exploring various in-game scenarios, to enhance the scalability of the maze game solution.

REFERENCES

[1] Z. Abd Algfoor, M. S. Sunar, and H. Kolivand, “A comprehensive study on pathfinding techniques for robotics and video games,” *International Journal of Computer Games Technology*, vol. 2015, 2015, doi: 10.1155/2015/736138.

[2] R. Graham, H. McCabe, and S. Sheridan, “Pathfinding in Computer Games,” *The ITB Journal*, vol. 4, no. 2, p. 6, Nov. 2015, doi: 10.21427/D7ZQ9J.

[3] G. Teixeira Galam, T. P. Remedio, and M. A. Dias, “Viral Infection Genetic Algorithm with Dynamic Infectability for Pathfinding in a Tower Defense Game,” *Brazilian Symposium on Games and Digital Entertainment, SBGAMES*, vol. 2019-October, pp. 198–207, Oct. 2019, doi: 10.1109/SBGAMES.2019.00034.

[4] Y. Sazaki, H. Satria, and M. Syahroyni, “Comparison of A∗ and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game,” *Proceeding of 2017 11th International Conference on Telecommunication Systems Services and Applications, TSSA 2017*, vol. 2018-January, pp. 1–6, Jan. 2018, doi: 10.1109/TSSA.2017.8272918.

[5] D. Kurniadi, A. Mulvani, and R. S. Maolani, “Implementation of Pathfinding Algorithm in Sundanese Land History Educational Game,” *2021 2nd International Conference on Innovative and Creative Information Technology, ICITech 2021*, pp. 145–150, Sep. 2021, doi: 10.1109/ICITECH50181.2021.9590181.

[6] W. Westera *et al.*, “Artificial intelligence moving serious gaming: Presenting reusable game AI components,” *Educ Inf Technol (Dordr)*, vol. 25, no. 1, pp. 351–380, Jan. 2020, doi: 10.1007/S10639-019-09968-2.

[7] I. Lagzi, S. Soh, P. J. Wesson, K. P. Browne, and B. A. Grzybowski, “Maze solving by chemotactic droplets,” *J Am Chem Soc*, vol. 132, no. 4, pp. 1198–1199, Feb. 2010, doi: 10.1021/JA9076793/SUPPL_FILE/JA9076793_SI_006.AVI.

[8] V. S. Gordon and Z. Matley, “Evolving sparse direction maps for maze pathfinding,” *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, vol. 1, pp. 835–838, 2004, doi: 10.1109/CEC.2004.1330947.

[9] N. H. Barnouti, S. S. M. Al-Dabbagh, and M. A. Sahib Naser, “Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm,” *Journal of Computer and Communications*, vol. 04, no. 11, pp. 15–25, 2016, doi: 10.4236/JCC.2016.411002.

[10] B. V. Indriyono and Widyatmoko, “Optimization of Breadth-First Search Algorithm for Path Solutions in Mazyin Games,” *International Journal of Artificial Intelligence & Robotics (IJAIR)*, vol. 3, no. 2, pp. 58–66, Nov. 2021, doi: 10.25139/IJAIR.V3I2.4256.

[11] P. Harsani, I. Mulyana, and D. Zakaria, “Fuzzy logic and A* algorithm implementation on goat foraging games,” *IOP Conf Ser Mater Sci Eng*, vol. 332, no. 1, p. 012054, Mar. 2018, doi: 10.1088/1757-899X/332/1/012054.

[12] I. Ariyanti, M. A. Ganiardi, and U. Oktari, “Mobile Application Searching of the Shortest Route on Delivery Order of CV. Alfa Fresh With Brute Force Algorithm,” *Logic : Jurnal Rancang Bangun dan Teknologi*, vol. 19, no. 3, pp. 120–130, Nov. 2019, doi: 10.31940/LOGIC.V19I3.1437.

[13] A. Erguzen and E. Erdal, “SUDOKU SOLUTION WITH BACKTRACKING ALGORITHM,” *International Journal of Advances in Electronics and Computer Science*, no. 7, pp. 2394–2835, 2020, Accessed: Feb. 10, 2023. [Online]. Available: http://iraj.inSudokuSolutionwithBacktrackingAlgorithm

[14] J. M. Chambers, “Object-Oriented Programming, Functional Programming and R,” *https://doi.org/10.1214/13-STS452*, vol. 29, no. 2, pp. 167–180, May 2014, doi: 10.1214/13-STS452.

[15] N. Thaker and A. Shukla, “Python as Multi Paradigm Programming Language,” *Int J Comput Appl*, vol. 177, no. 31, pp. 38–42, Jan. 2020, doi: 10.5120/IJCA2020919775.