

# A\* Maze Runner Project Report

**CPSC 481 Artificial Intelligence Project**

**Team Members: Shayan Darian, Loc Nguyen, Jake Watson, Michael Yu**

**Professor: Mira Kim**

**Course: CPSC 481-01**

**Fall 2023**

## Problem Statement

Our A\* Maze Runner application is an AI based project created using python, tkinter, and utilizes the pathfinding algorithm A\* search. This AI based project was created to change the perspective of students and educators passionate about exploring the complexities of the A\* algorithm. The main objective of our project was to give access to a maze generation, maze solving algorithm, and a feature to visualize the solution of a user given maze. Users can experiment with different maze sizes, wall probability, and establish start/end points. Additionally, the A\* Maze Runner application offers a step by step guidance that shows users which coordinate the maze runner has moved to next. The team has created unique application that can help understand the ways the A\* pathfinding algorithm works and helps create custom mazes that can be used for educational purposes.

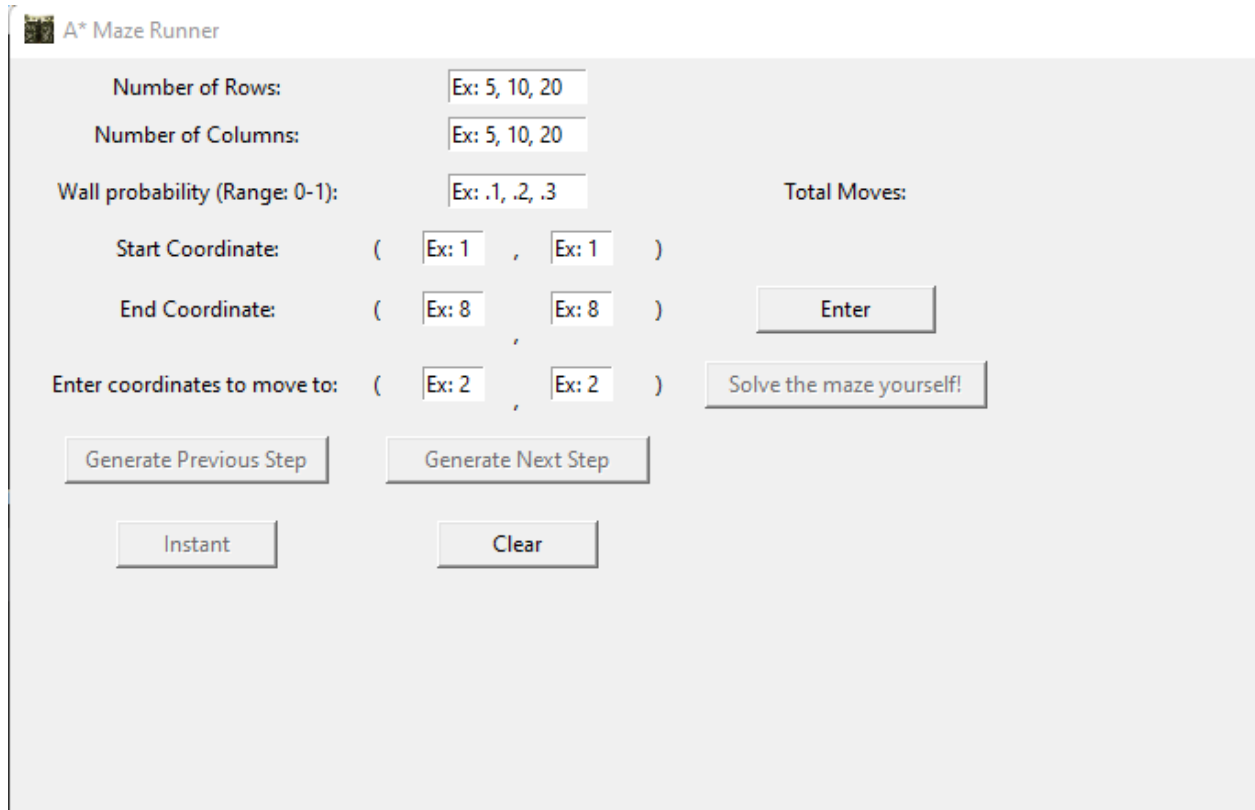
## Approach

For our project, we used Python as the main programming language because many of us were familiar with it. Tkinter was our choice for creating the GUI that the user will be interacting with and the main search algorithm we implemented is the A\* algorithm. How we handled version control was with the use of git and github.

<b>Language</b>	<b>Python</b>
<b>GUI</b>	<b>Tkinter</b>
<b>Algorithm</b>	<b>A* algorithm</b>

## Description of the software

Front-end/GUI : This was developed using ‘tkinter’, providing us an interface for users to input their maze parameters such as rows, columns, wall probability, and start/end coordinates. The team has also implemented a feature allowing users to input their own coordinates they want to move to. We used buttons to help users interact with the GUI. Our GUI displays the maze using a grid system where we add labels, buttons, and entry boxes to it.



The screenshot shows the 'A\* Maze Runner' application window. It features a light gray background with a white title bar. The interface includes several input fields and buttons. At the top, there are three input fields for 'Number of Rows:', 'Number of Columns:', and 'Wall probability (Range: 0-1):', each with a placeholder example (Ex: 5, 10, 20; Ex: 5, 10, 20; Ex: .1, .2, .3). To the right of these is a 'Total Moves:' label. Below these are two rows of coordinate inputs: 'Start Coordinate:' and 'End Coordinate:', each with two input boxes (Ex: 1, Ex: 1; Ex: 8, Ex: 8) and a comma separator. Below the end coordinates is a row for 'Enter coordinates to move to:' with two input boxes (Ex: 2, Ex: 2) and a comma separator. To the right of the move coordinates is a button labeled 'Solve the maze yourself!'. Below the coordinate inputs are two buttons: 'Generate Previous Step' and 'Generate Next Step'. At the bottom are two buttons: 'Instant' and 'Clear'. An 'Enter' button is located to the right of the end coordinate inputs.

AI Algorithms: The team implemented the A\*algorithm to find the shortest path within the drawn maze and uses heuristics to optimize the maze solution.

## Description of the buttons

**Enter** : Creates maze with user given maze parameters.

A\* Maze Runner

Number of Rows:

10

Number of Columns:

10

Wall probability (Range: 0-1):

.2

Total Moves:

Start Coordinate:

(

1

,

1

)

End Coordinate:

(

8

,

8

)

Enter

Enter coordinates to move to:

(

Ex: 2

,

Ex: 2

)

Solve the maze yourself!

Generate Previous Step

Generate Next Step

Instant

Clear

Initial Maze:

Shortest Path:

***Solve the maze yourself*** : Allows users to manually solve the maze.

### A\* Maze Runner

Number of Rows:

Number of Columns:

Wall probability (Range: 0-1):  Total Moves: 10

Start Coordinate: (  ,  )

End Coordinate: (  ,  )

Enter coordinates to move to: (  ,  ) 

Solve the maze yourself!

Instant

Clear

Initial Maze:					Shortest Path:				
				■				? ? ? ?	
	■		■		■			■ ? ? ? ?	
	■		■		■			? ■ ? ? ? ?	
			■					? ? ■ ? ? ?	
				■				? ? ? ■ ? ?	
		■			■ ■			? ? ■ ★ ? ?	
	■				■			? ? ? ? ? ? ?	
		■						? ? ? ? ? ? G	

**Generate Next/Previous Step :** Allows users to move on to the next path of the maze or go back.

## A\* Maze Runner

Number of Rows:

Number of Columns:

Wall probability (Range: 0-1):  Total Moves:

Start Coordinate: (  ,  )

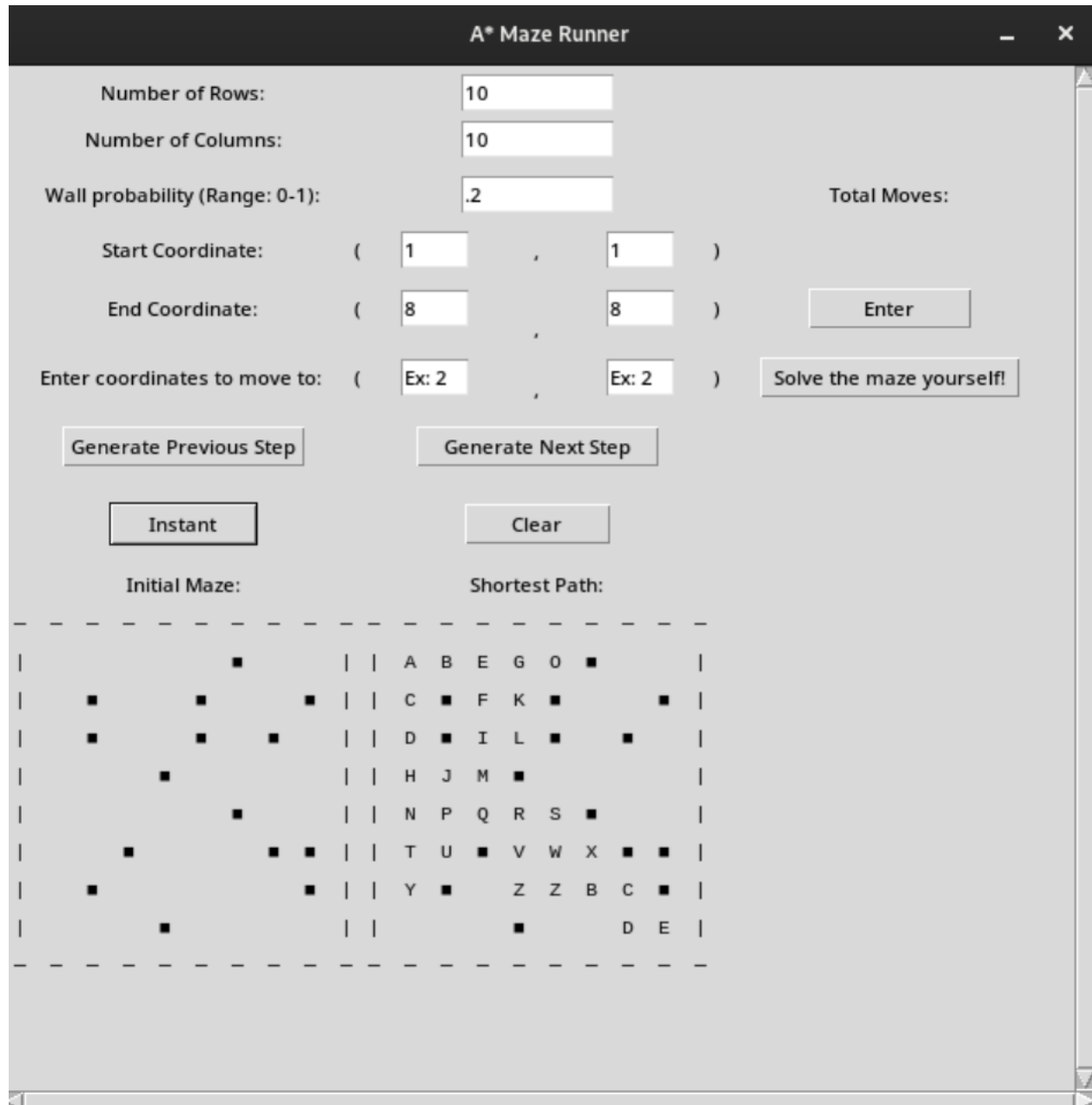
End Coordinate: (  ,  )

Enter coordinates to move to: (  ,  )

Initial Maze:

Shortest Path:

**Instant** : Shows the completed path of the maze.



The image shows a software window titled "A\* Maze Runner". It contains several input fields and buttons. At the top, there are fields for "Number of Rows:" (10), "Number of Columns:" (10), and "Wall probability (Range: 0-1):" (.2). To the right of these is a "Total Moves:" label. Below these are fields for "Start Coordinate:" (( 1 , 1 )) and "End Coordinate:" (( 8 , 8 )). Further down is a field for "Enter coordinates to move to:" (( Ex: 2 , Ex: 2 )). To the right of this field is a button labeled "Enter". Below the coordinate fields are two buttons: "Generate Previous Step" and "Generate Next Step". At the bottom of the input section are two buttons: "Instant" and "Clear". The main area of the window displays two 10x10 grids. The left grid is labeled "Initial Maze:" and shows a maze with black squares representing walls. The right grid is labeled "Shortest Path:" and shows the same maze with letters A through Z indicating the path from start to end. The path starts at (1,1) and ends at (8,8).

Number of Rows: 10

Number of Columns: 10

Wall probability (Range: 0-1): .2

Total Moves:

Start Coordinate: ( 1 , 1 )

End Coordinate: ( 8 , 8 )

Enter coordinates to move to: ( Ex: 2 , Ex: 2 )

Enter

Generate Previous Step

Generate Next Step

Instant

Clear

Initial Maze:

Shortest Path:


**Clear** : Resets the GUI.

## Included Files

AstarMazeRunner.py : This is our main script where Tkinter is used to create the applications' UI and handle user interaction.

Astar\_gen.py : This file was the maze generation code and A\* pathfinding algorithm. Its main purpose is to create a custom maze and generate the solution for the maze.

generate\_maze() : Creates a maze as a 2D list based on the users' parameters. Based on the users' wall probability, creates walls within the custom maze.

print\_maze() : This prints the maze to the console and prints each row with spaces in between to create a visual representation of the user created maze.

heuristic() : This calculates the heuristic cost between two points(current and goal). We utilized the euclidean distance formula to get the distance between the two points and are able to estimate the cost of reaching the goal node from the current node.

astar\_pathfind\_gen(): This is our function that executes the A\* search algorithm to find a path from the users' given start coordinate and goal coordinates. It utilizes priority queue to keep track of nodes to explore based on their estimated cost. The function stops when the goal has been found or when there are no more nodes to explore. This also helps generate the visualization of the A\* search path step by step.



`finish()`: This function uses `astar_pathfind_gen()` to find the path of the custom maze created by the user. It modifies the generated maze to include the solution path. If the goal is unreachable with the custom maze, this will return “None” to the user but if there is a path, it will modify the initial given maze.

`scroll.py` : What `scroll.py` does is it allows us to implement a vertical and horizontal scroll bar which will allow for a dynamically changing GUI. This is important because if a maze that is too big for the screen is generated, the maze would be cut off and the user would have no way of viewing the whole maze. This file allows us to solve that problem.

## **Evaluation metrics**

Our team tested multiple input cases within A\* maze generator and have added error messages in some of those cases. Some include where the maze generated does not have a solid path from the given coordinates, due to the wall probability, and once the maze solution is generated, it will tell the user that there is no solution path. We also had test cases with a large maze boundary and encountered performance issues. These weren't fixed but the team has been able to make the larger mazes look better visually, allowing better user interaction.

## **Future work**

If the scope of the project did not end and we were allotted more time, there are a couple of features that we would try to implement. One feature we would add is how our generated mazes are displayed. Instead of having an ascii based image we would have colored images instead. To achieve this, we would try and incorporate the python module, pygame. This would allow us to easily display images which would allow us to change how the maze appears.

Another addition to the project is an option to change how the maze is completed. Instead of using the A\* algorithm, an option to solve the maze with the BFS (Breadth First Search) or DFS (Depth First Search) algorithms. This would allow the user to compare and contrast the similarities and differences of each search algorithm as well as learn how each algorithm works.

Lastly, we would be interested in implementing an import system where the user could import their own custom maze and the application would be able to create the solved path for the custom maze for them.

## **Conclusion**

One main lesson we learned from this project was when we were dealing with passing lists to functions. We ran into a problem when we were switching between pressing the “instant” button and the “generate next/back” button; the displayed maze was incorrect. We learned that when passing a list into a function, python decides to pass by reference and not by value. So our solution was to import the copy library in order to be able to create a distinct copy of a list. Another lesson we learned was that heapq, the python implementation of a priority queue, is very limited. There is no built in duplicate value detection, and the correct cost node was not always

being removed. Due to these problems, we decided to make our own priority queue, which was much simpler.

In conclusion, this project was a unique learning experience for us. We were able to learn and use the Tkinter GUI toolkit, learn the nuances of implementing the A\* algorithm, and overall gain a better understanding of python. The team was also able to practice planning, communication, and using git and github in a group environment.

## References

- [1] Barnouti, N. , Al-Dabbagh, S. and Sahib Naser, M. (2016) Pathfinding in Strategy Games and Maze Solving Using A\* Search Algorithm. Journal of Computer and Communications, 4, 15-25. doi: 10.4236/jcc.2016.411002.
  
- [2] Elder, John. "Tkinter Course - Create Graphic User Interfaces in Python Tutorial." YouTube, YouTube, 19 Nov. 2019, [www.youtube.com/watch?v=YXPyB4XeYLA](https://www.youtube.com/watch?v=YXPyB4XeYLA).
  
- [3] Hutch. Joe. How to Add Scrollbars to a Dynamic GUI in TKinter · Joe Hutchinson. (n.d.). <https://www.joehutch.com/posts/tkinter-dynamic-scroll-area>
  
- [4] Tjiharjadi, Semuil, Marvin Wijaya, and Erwin Setiawan. "Optimization maze robot using A\* and flood fill algorithm." International Journal of Mechanical Engineering and Robotics Research 6.5 (2017): 366-372.