Mesh Solver by BFS and A* Technique.

By: AnshVaid & Ayush Gupta

Abstract

There are more than one ways to reach from the initial state to the goal state. But when it comes for the computers to solve this situation then which path should it choose to reach to its goal, then here comes the role of Artificial Intelligence(AI), i.e. to create some sort of program so that a machine could behave like humans or we can say that the decision taken by the machine in the similar manner as the human beings use their intelligence and take decision.

The project is the implementation of Breadth First Search (BFS) and A* search technique for giving a human like behaviour to the machine with the help of some program and logics only to show the simulation that could be used in the vehicles such as buses, bicycles and even nowadays in cabs to detect the shortest path from your current position to the destination and avoiding the hurdles as well, in the same way as the human beings use their intelligence to avoid hurdles on the path during walking or driving.

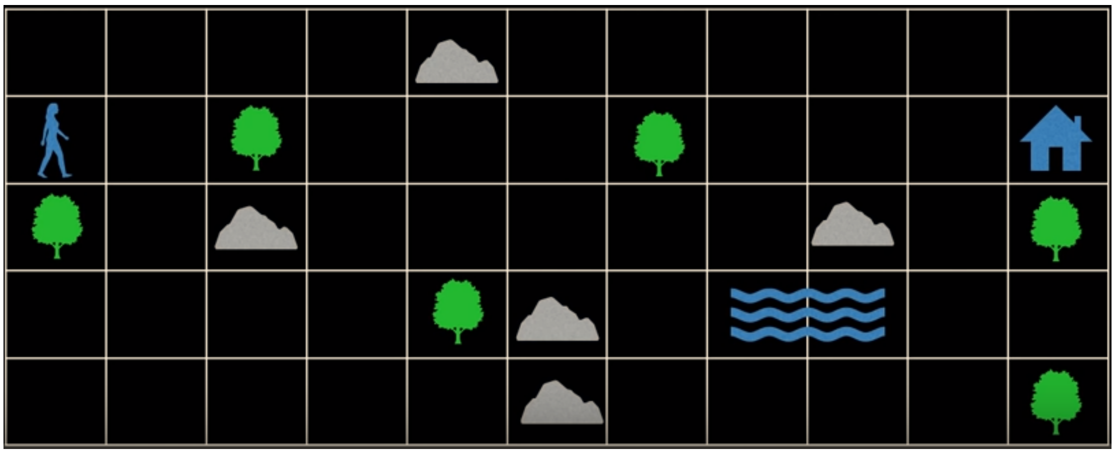


Figure 1: Above picture showing a human being whose goal state is to reach home by avoiding trees and other hurdles in its path.

Introduction

There are two types of searches: Informed Search and Uninformed Search. Breadth First Search (BFS) is the example of Uninformed Search and A* search technique is the example of Informed Search technique. The aim of the project is only to use Breadth First Search (BFS) and A* search technique in order to get the shortest path from initial state to the goal state and to differentiate between these to search techniques depending upon the time to search and return back the shortest path. Well it is very clear cut thing that BFS technique will take more time as compare to A* because BFS as the name suggests Breadth First Search, searches for all the possible paths from the initial state to the goal state and sometimes it is also not sure that the path will be found or not, on the other hand A* searches the best path from the beginning itself if the help of cost and heuristics unlike Breadth First Search Technique.

Motivation

One of the most famous problems in AI field is *Dungeon Problem Statement* in which the aim is to reach to the goal state from the starting position by avoiding the hurdles and also in the shortest path.

Figure 2: A game based on Dungeon Problem.

The question is all about *"Is the escape possible?"*, if the answer is *"yes"* then another question arises *"How long will it take?"*. So, the project is based on this problem and it is solved with the help of certain logics and GUI in the program.

Breadth First Search approach to Solve the Problem

By: Ansh Vaid

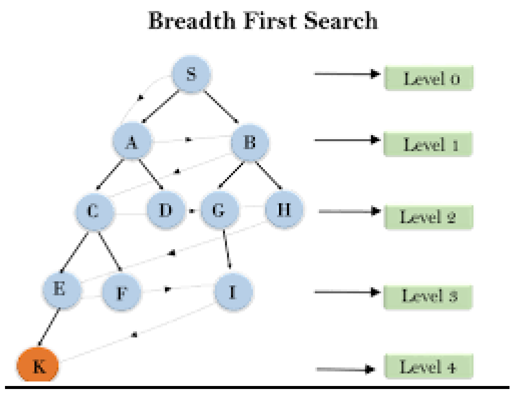


Figure 3: Level order search and goal state is K.

Breadth First Search also known as BFS is an uninformed searching technique which is a level order search, tries each and every node at least once to get the desired output. In programming, it can be implemented using *Queue* which works on First In First Out (FIFO) logic, i.e. a node of a tree that comes in queue first will be processed first.

The mesh could be implemented in the form of a graph easily and with the help of direction vectors.

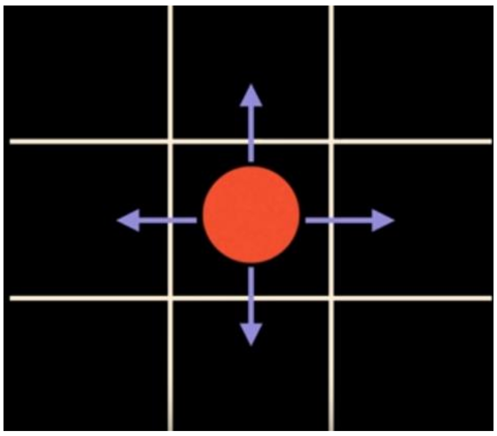
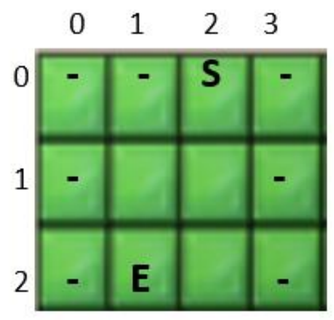


Figure 4: Red spot is current position. Move this spot Left, Right, Upward, Downward direction.

It can be done by assuming the above 9 cells as 2D array. Let current position be (0,0), now to go in left direction, subtract 1 from *x* position, i.e. (-1,0); to go in right direction, add 1 from *x* axis, i.e. (1,0); now to go in upward direction, add 1 from *y* position, i.e. (0,1); now to go in downward direction, subtract 1 from *y* position, i.e. (0,-1).

In python lists are used for implementing 2D array. The size of the mesh will define the number of coordinates that will be used in the program. If there are 5 columns in the mesh and 7 rows in the mesh, i.e. the length of the list is 7 and the length of each element in 7 rows is 5.

Implementation



Suppose we have the above mesh, it can be created with the help of List in python. So, if the above mesh has to be created then it can be done in following way:

```
Map=[“__S_”,
     “_  _”,
     “_E _”]
```

In above “Map” is a list in which “_” represents the obstacle or wall, “ “ represents the free path, “S” represents the Starting point and “E” represents the Exit point.

Now main focus is to program in such a way that we have the list of all the coordinates of obstacle or wall in one list, coordinate of Starting and Ending point in different variables and coordinates of all the Spaces i.e. path, in one list and also to maintain a dictionary to backtrack from exit to starting point.

In my program I have used the following:

- Walls: For storing the coordinates of obstacles, i.e. “_”.

  Therefore coordinates: (0,0) , (1,0) , (3,0) , (0,1) , (3,1) , (0,2) , (3,2) w.r.t above example will be stored in walls list.

- Path: For storing the coordinate of spaces and exit point, i.e. “ “ and “E”.

  Therefore coordinates: (1,1), (2,1) , (1,2) , (2,2) w.r.t above examples will be stored in the path list.

- Visited: For storing or keeping the track on the coordinates that are processed one by one from the path list to reach exit point coordinate.

- Q(will be used for queue): The search will start from S(start) coordinate. Now There is a need to find the neighbouring coordinates whether it is an obstacle or a path, which can help us to reach to path. So:

- Add 1 to “x” coordinate of S(start) to get the coordinate of node present on the right of S(start). Therefore, the coordinate changes to (3,0). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” then append it to the path list, if this coordinate is “_” then append it to the walls list and add it to the queue.

- Subtract 1 from “x” coordinate of S(start) to get the coordinate of node present on the left of S(start). Therefore, the coordinate changes to (1,0). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” then append it to the path list, if this coordinate is “_” then append it to the walls list and add it to the queue.

- Add 1 to “y” coordinate of S(start) to get the coordinate of node present below the S(start). Therefore, the coordinate changes to (2,1). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” then append it to the path list, if this coordinate is “_” then append it to the walls list and add it to the queue.

- Subtract 1 from “y” coordinate of S(start) to get the coordinate of node present above the S(start). Therefore, the coordinate changes to (2,-1). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” then append it to the path list, if this coordinate is “_” then append it to the walls list and add it to the queue.

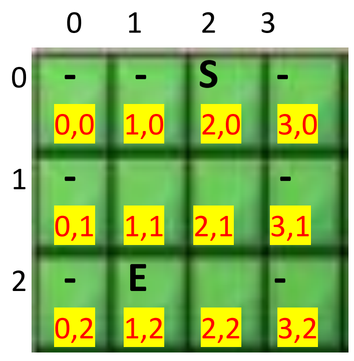


Figure 4: Showing all the coordinates of mesh

Walls=[(0,0) , (1,0) , (3,0) , (0,1) , (3,1) , (0,2) , (3,2)]
Path =[(1,1), (2,1) , (1,2) , (2,2)]
Queue=Answer will vary but it might be correct
Visited=Depend upon the Queue popping.

Once all the coordinates are processed and added it respective lists and queues then start dequeuing from queue, and simultaneously add to the visited list as well to avoid any ambiguities such as the same coordinate processed again. If you want to backtrack also from the exit point then use dictionaries in which initial coordinates (x,y) will act as key and the other neighbouring path coordinate(present left, right, above ,below the initial node) will act as value of the key, and at the time of backtracking find the value of key from the value and gradually you will reach the start point from exit point.

Dictionary= {(2,0) : (2,1) ,(2,1) : (1,1) ,(2,1) : (2,2) ,(1,1) : (1,2) , (2,2) : (1,2)}

Now to backtrack the values of above dictionary will be used to get their respective key values and gradually the program will able to backtrack from (1,2) to (2,0).

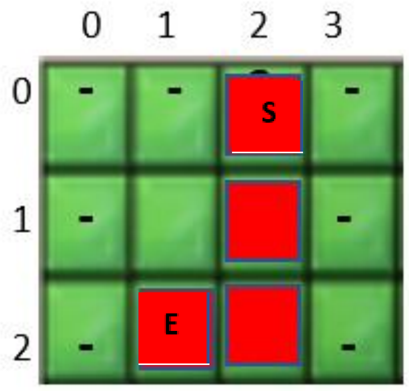


Figure 5: Result after backtrack.

About GUI

After implementing this program the program will not be so much user friendly in terms of graphics, so for GUI use Turtle Library or Kivy(Extension of python GUI).
In this program the code has the implementation of Turtle Library. It is not necessary that the programmer should know about turtle library or have a command on it because the code uses only 4-5 keywords from turtle library again and again, can be simply learnt from Google also.

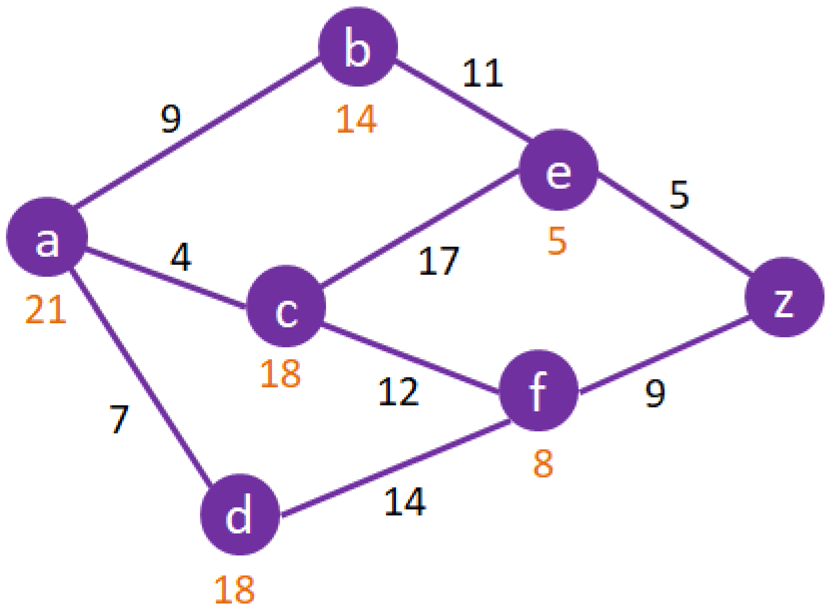
1.Turtle()	5. Goto()
2.Penup()	6. Stamp()
3.Pendown()	7. Shape()
4.Color()	8. Hideturtle()

Conclusion

In this way by using Breadth First Search we were able to search for the exit point from the starting point, though the search takes time and as the number of grids increases the searching time will increase because the program has to update the values in the respective lists.

A* Search approach to Solve the Problem

By: Ayush Gupta

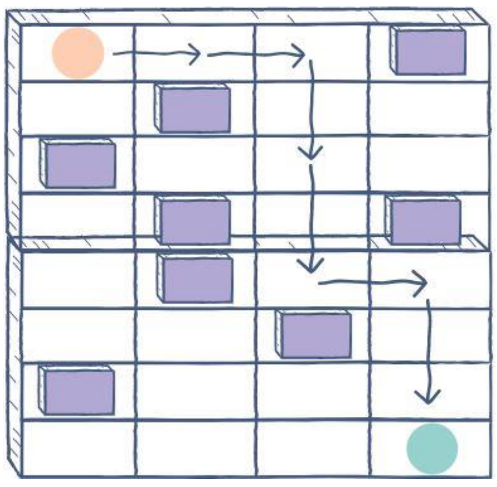


A * algorithm is a searching algorithm that searches for the shortest path between the *initial and the final state.* It is used in various applications, such as *maps*.

In *maps* the A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state)

Imagine a square grid which possesses many obstacles, scattered randomly. The initial and the final cell is provided. The aim is to reach the final cell in the shortest amount of time.

Here A* Search Algorithm comes to the rescue:



Explanation

A* algorithm has 3 parameters:

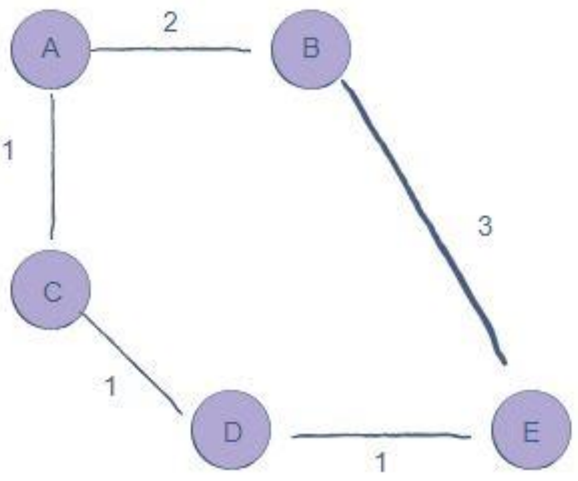
- **g :** the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
- **h :** also known as the *heuristic value,* it is the **estimated** cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We **must** make sure that there is **never** an over estimation of the cost.
- **f :** it is the sum of g and h. So, **f = g + h**

The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the *smallest f-valued cell* and moves to that cell. This process continues until the algorithm reaches its goal cell.

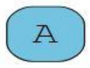
Example

A* algorithm is very useful in graph traversals as well. In the following slides, you will see how the algorithm moves to reach its goal state.

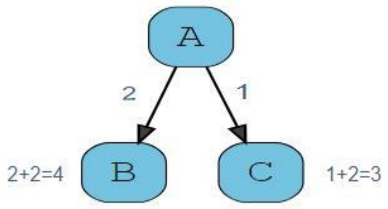
Suppose you have the following graph and you apply A* algorithm on it. The initial node is **A** and the goal node is **E**.



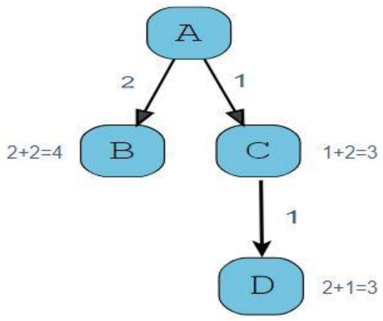
At every step, the f-value is being re-calculated by adding together the g and h values. The minimum f-value node is selected to reach the goal state. Notice how node B is *never* visited



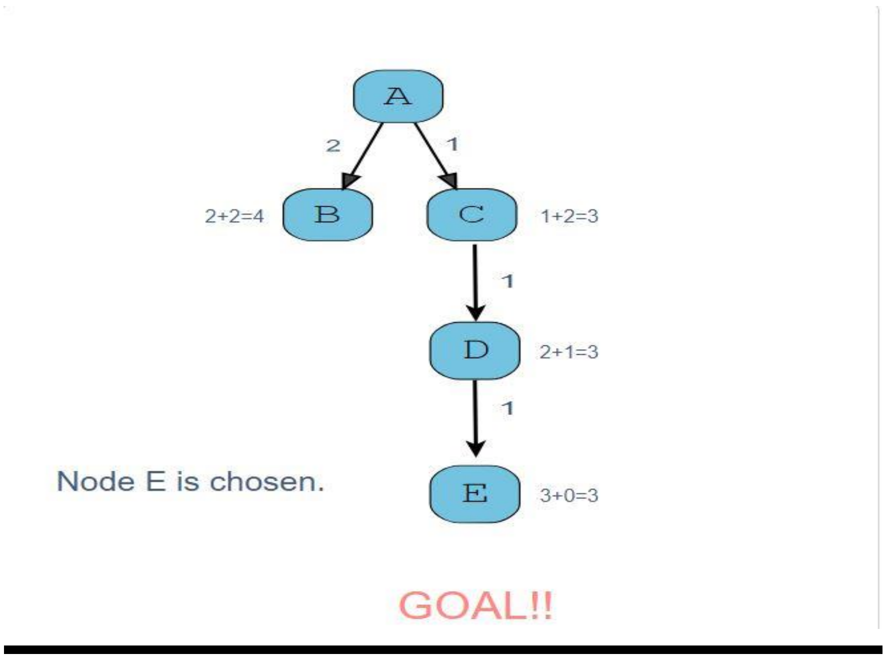
Root node A.



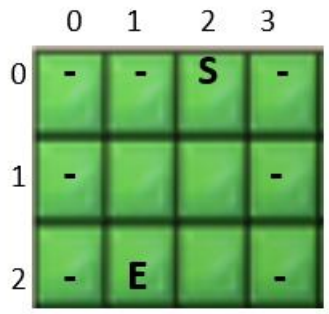
Node C is chosen.



Node D is chosen.



Implementation



Suppose we have the above mesh, it can be created with the help of List in python. So, if the above mesh has to be created then it can be done in following way:

```
Map=["++S+",
     "++  +",
     "+E  +"]
```

In above "Map" is a list in which "+" represents the obstacle or wall, " " represents the free path, "S" represents the Starting point and "E" represents the Exit point.

Now main focus is to program in such a way that we have the list of all the coordinates of obstacle or wall in one list, coordinate of Starting and Ending point in different variables and coordinates of all the Spaces i.e. path, in one list and also to maintain a list to backtrack from exit to starting point.

In my program I have used the following:

- Walls: For storing the coordinates of obstacles, i.e. "_".

  Therefore coordinates: (0,0) , (1,0) , (3,0) , (0,1) , (3,1) , (0,2) , (3,2) w.r.t above example will be stored in walls list.

- Path: For storing the coordinate of optimal path i.e. "1"

  Therefore coordinates: (0,2), (2,1) , (1,2) w.r.t above examples will be stored in the path list.

- Closed List: For storing or keeping the track on the coordinates that are processed one by one from the path list to reach exit point coordinate.

- Open List: For storing or keeping the track on the coordinates that are not processed one by one from the path list to reach exit point coordinate.

- Neighbours: The search will start from S(start) coordinate. Now

  There is a need to find the neighbouring coordinates whether it is an obstacle or a path, which can help us to reach to path. So:

  - Add 1 to "x" coordinate of S(start) to get the coordinate of node present on the right of S(start). Therefore, the coordinate changes to (3,0). If there is no such node then leave it otherwise if this coordinate is a " " or "e" or "+" then append it to the neighbour list.

  - Subtract 1 from "x" coordinate of S(start) to get the coordinate of node present on the left of S(start). Therefore, the coordinate changes to (1,0). If there is no such node then leave it otherwise if this coordinate is a " " or "e" or "+" then append it to the neighbour list.

  - Add 1 to "y" coordinate of S(start) to get the coordinate of node present below the S(start). Therefore, the coordinate changes to (2,1). If there is no such node then leave it otherwise if this coordinate is a " " or "e" or "+"then append it to the neighbour list.

  - Subtract 1 from "y" coordinate of S(start) to get the coordinate of node present above the S(start). Therefore, the coordinate changes to (2,-1). If there is no such node then leave it otherwise if this coordinate is a " " or "e" or "+" then append it to the neighbour list.

  - Subtract 1 from "y" coordinate Subtract 1 from "x" coordinate to get the coordinate of node present diagonally above left the S(start). If there is no such node then leave it otherwise if this coordinate is a " " or "e" or "+" then append it to the neighbour list.

- Add 1 to “y” coordinate Subtract 1 from “x” coordinate and of S(start) to get the coordinate of node present diagonally above right the S(start). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” or ”+” then append it to the neighbour list.

- Add 1 to “y” coordinate and Add 1 to “x” coordinate and of S(start) to get the coordinate of node present diagonally below right the S(start). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” or ”+” then append it to the neighbour list.

- Subtract 1 from “y” coordinate and Add 1 to “x” coordinate and of S(start) to get the coordinate of node present diagonally below left the S(start). If there is no such node then leave it otherwise if this coordinate is a “ “ or “e” or ”+” then append it to the neighbour list.

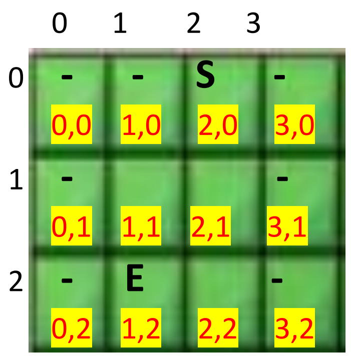


Figure 6: Showing all the coordinates of mesh

Neighbours of S=[(1,0) , (1,1) , (3,0) , (3,1) , (2,1)]
Path =[(2,0) , (1,2) , (2,1)]

Once all the coordinates are processed and added it respective lists then create object for each coordinate of grid and initialize it with f,g,h values h is heuristic which is the distance between the end and the current processing spot , and simultaneously add to the closed list as well to avoid any ambiguities such as the same coordinate processed again. If you want to backtrack also from the exit point then use list path in which initial coordinates (x,y) will act as optimal path corodinates.

About GUI

After implementing this program the program will not be so much user friendly in terms of graphics, so for GUI use Turtle Library (Extension of python GUI).
In this program the code has the implementation of Turtle Library. It is not necessary that the programmer should know about turtle library or have a command on it because the

code uses only 4-5 keywords from turtle library again and again, can be simply learnt from Google also.

1.Turtle()
2.Penup()
3.Pendown()
4.Color()
5. Goto()
6. Stamp()
7. Shape()

Conclusion

In this way by using A Star Search we were able to search for the exit point from the starting point, this search takes minimum time and gives optimal path to reach the end point from the starting point.

BIBLIOGRAPHY

- https://i.ytimg.com/vi/KiCBXu4P-2Y/maxresdefault.jpg for photograph.
- https://www.youtube.com/watch?v=ec0IJsiIuWk for idea.
- https://www.geeksforgeeks.org/deque-in-python/ for deque for dequeue.
- https://www.google.com/imgres for photograph.
- https://www.youtube.com/watch?v=8Hs5gptvBxU for idea.
- https://www.geeksforgeeks.org/turtle-programming-python/ for studying turtle.