**INT-404**

**Artificial Intelligence**

A\* ALGORITHM TO FIND OUT OPTIMAL PATH FROM UNIMALL TO LOVELY BAKE STUDIO

**by*RAJKAMAL***

**End Term Report**

**s.no** **Name Reg no Roll no**

**1 RAJKAMAL 11803811 44**

**2 RAHUL MISHRA 11803805 43**

**3 MUNTAZIR ALAM 11803809 45**

**Section: K18HV**



**Department of Intelligent Systems School of Computer Science Engineering Lovely Professional University, Jalandhar April-2020**

## STUDENT DECLARATION

This is to declare that this report has been written by us. No part of the report is copied from other sources. All information included from sources have been duly acknowledged. We aver that if any part of the report is found to be copied, we shall take full responsibility for it.

# Name of student1:- Name of Student3:-

RAJKAMAL RAHUL MISHRA

Reg. No: 11803811 Reg. No: 11803805

Roll NO: 44 Roll No: 43

# Name of Student2:-

# MUNTAZIR ALAM

# Reg no:-11803809

# Roll.no:-4

**TITLE PAGE NO**

**1. Background and objectives of project assigned ................ ………………………………..1**

**1.1 ........................................................................................…… …………………………1**

**1.1.1 .................................................................................... …………………………………3**

**1.1.2 ....................................................................................................................... 5**

**1.2 ......................................................................................................................... 7**

**1.2.1 ....................................................................................................................... 7**

**1.2.2 ....................................................................................................................... 9**

**1.2.3 ...................................................................................................................... 10**

**2 Description of Project ... ...................................................................................... 14**

**2.1 .......................................................................................................................... 17**

**2.2 .......................................................................................................................... 18**

**BONAFIDE CERTIFICATE**

Certified that this project report “\A\* ALGORITHM TO FIND OUT OPTIMAL PATH FROM UNIMALL TO LOVELY BAKE STUDIO” is the bonafide work of “RAJKAMAL,RAHUL MISHRA,MUNTAZIR ALAM” who carried out the project the project work under my supervision.

## Mr. Dipen Saini Assistant Professor

**Dept: Computer Science & Engineering LPU**

**Phagwara PUNJAB**

1. **Background and Objective of the Project:**

## Introduction:

* We present [A\*](https://en.wikipedia.org/wiki/A*_search_algorithm) is the most popular choice for pathfinding, because it’s fairly flexible and can be used in a wide range of contexts.
* It is an Artificial Intelligence algorithm used to find shortest possible path from start to end states.
* It could be applied to character path finding, puzzle solving and much more. It really has countless number of application.
* **Peter Hart, Nils Nilsson** and **Bertram Raphael** of Stanford Research Institute.first published the algorithm in 1968.
* The A\* algorithm uses *both* the actual distance from the start and the estimated distance to the goal.

## Objective:

This project is useful for investors to invest in stock market based on the various factors. The project target is to create a program that analyses A\* ALGORITHM TO FIND OUT OPTIMAL PATH FROM UNIMALL TO LOVELY BAKE STUDIO.

The main feature of this project is to generate an approximate forecasting output and create a general idea of future values based on the previous data by generating a pattern. The scope of this project does not exceed more than a generalized suggestion tool.

## Description of the Project:

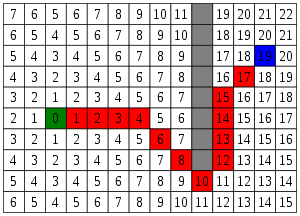
# https://miro.medium.com/max/263/1*HppvOLfDxXqQRFn0Cv2dHQ.gif F = G + H

One important aspect of A\* is f = g + h. The f, g, and h variables are in our Node class and get calculated every time we create a new node. Quickly I’ll go over what these variables mean.

* F is the total cost of the node.
* G is the distance between the current node and the start node.
* H is the heuristic — estimated distance from the current node to the end node.

Let’s take a look at a quick graphic to help illustrate this.

https://miro.medium.com/max/60/1*iSt-urlSaXDABqhXX6xveQ.png?q=20



wow such numbers, very color

Awesome! Let’s say node(0) is our starting position and node(19) is our end position. Let’s also say that our current node is at the the red square node(4).

## G

*G is the distance between the current node and the start node.*

If we count back we can see that node(4) is 4 spaces away from our start node.

We can also say that G is 1 more than our parent node (node(3)). So in this case for node(4), currentNode.g = 4.

## H

*H is the heuristic — estimated distance from the current node to the end node.*

So let’s calculate the distance. If we take a look we’ll see that if we go over 7 spaces and up 3 spaces, we’ve reached our end node (node(19)).

Let’s apply the Pythagorean Theorem! a² + b² = c². After we’ve applied this, we’ll see that currentNode.h = 7² + 3². Or currentNode.h = 58.

But don’t we have to apply the square root to 58? Nope! We can skip that calculation on every node and still get the same output. Clever!

With a heuristic, we need to make sure that we can actually calculate it. It’s also very important that the heuristic is always an underestimation of the total path, as an overestimation will lead to A\* searching for through nodes that may not be the ‘best’ in terms of f value.

## F

*F is the total cost of the node.*

So let’s add up h and g to get the total cost of our node. currentNode.f = currentNode.g + currentNode.h. Or currentNode.f = 4+ 58. Or currentNode.f = 62.

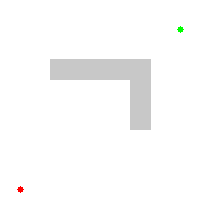
# Time to use f = g + h

Alright, so that was a lot of work. Now with all that work, what am I going to use this f value for?

With this new f value, we can look at all our nodes and say, “Hey, is this the best node I can pick to move forward with right now?”. Rather than running through every node, we can pick the ones that have the best chance of getting us to our goal.

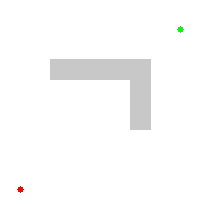
Here’s a graphic to illustrate. On top, we have Dijkstra’s Algorithm, which searches without this f value, and below we have A\* which does use this f value.

https://miro.medium.com/freeze/max/60/1*2jRCHqAbTCY7W7oG5ntMOQ.gif?q=20



Dijkstra’s Algorithm

https://miro.medium.com/freeze/max/60/1*HppvOLfDxXqQRFn0Cv2dHQ.gif?q=20



A\* Algorithm (Wikipedia)

**Dijkstra’s Algorithm**

So taking a look at Dijkstra’s algorithm, we see that it just keeps searching. It has no idea which node is ‘best’, so it calculates paths for them all.

**A\* Algorithm**

With A\*,we see that once we get past the obstacle, the algorithm prioritizes the node with the lowest f and the ‘best’ chance of reaching the end.

# A\* Method Steps — from Patrick Lester

I’ve pasted the steps for A\* from Patrick Lester’s article that you can check out [here](http://www.policyalmanac.org/games/aStarTutorial.htm). The same website is also listed below in resources. This is an insanely good explanation, and is why I decided to go with it rather than writing it again.

1. Add the starting square (or node) to the open list.

2. Repeat the following:

A) Look for the lowest F cost square on the open list. We refer to this as the current square.

B). Switch it to the closed list.

C) For each of the 8 squares adjacent to this current square …

* If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
* If it isn’t on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
* If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If you are keeping your open list sorted by F score, you may need to resort the list to account for the change.

D) Stop when you:

* Add the target square to the closed list, in which case the path has been found, or
* Fail to find the target square, and the open list is empty. In this case, there is no path.
* 3. Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path.

Code for the project

* *# This class represents a node*
* **class** **Node**:
* *# Initialize the class*
* **def** \_\_init\_\_(self, position:(), parent:()):
* self.position = position
* self.parent = parent
* self.g = 0 *# Distance to start node*
* self.h = 0 *# Distance to goal node*
* self.f = 0 *# Total cost*
* *# Compare nodes*
* **def** \_\_eq\_\_(self, other):
* **return** self.position == other.position
* *# Sort nodes*
* **def** \_\_lt\_\_(self, other):
* **return** self.f < other.f
* *# Print node*
* **def** \_\_repr\_\_(self):
* **return** ('(**{0}**,**{1}**)'.format(self.position, self.f))
* *# Draw a grid*
* **def** draw\_grid(map, width, height, spacing=2, \*\*kwargs):
* **for** y **in** range(height):
* **for** x **in** range(width):
* print('**%%**-**%d**s' % spacing % draw\_tile(map, (x, y), kwargs), end='')
* print()
* *# Draw a tile*
* **def** draw\_tile(map, position, kwargs):
* *# Get the map value*
* value = map.get(position)
* *# Check if we should print the path*
* **if** 'path' **in** kwargs **and** position **in** kwargs['path']: value = '+'
* *# Check if we should print start point*
* **if** 'start' **in** kwargs **and** position == kwargs['start']: value = '@'
* *# Check if we should print the goal point*
* **if** 'goal' **in** kwargs **and** position == kwargs['goal']: value = '$'
* *# Return a tile value*
* **return** value
* *# A\* search*
* **def** astar\_search(map, start, end):
* *# Create lists for open nodes and closed nodes*
* open = []
* closed = []
* *# Create a start node and an goal node*
* start\_node = Node(start, **None**)
* goal\_node = Node(end, **None**)
* *# Add the start node*
* open.append(start\_node)
* *# Loop until the open list is empty*
* **while** len(open) > 0:
* *# Sort the open list to get the node with the lowest cost first*
* open.sort()
* *# Get the node with the lowest cost*
* current\_node = open.pop(0)
* *# Add the current node to the closed list*
* closed.append(current\_node)
* *# Check if we have reached the goal, return the path*
* **if** current\_node == goal\_node:
* path = []
* **while** current\_node != start\_node:
* path.append(current\_node.position)
* current\_node = current\_node.parent
* *#path.append(start)*
* *# Return reversed path*
* **return** path[::-1]
* *# Unzip the current node position*
* (x, y) = current\_node.position
* *# Get neighbors*
* neighbors = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
* *# Loop neighbors*
* **for** next **in** neighbors:
* *# Get value from map*
* map\_value = map.get(next)
* *# Check if the node is a wall*
* **if**(map\_value == '#'):
* **continue**
* *# Create a neighbor node*
* neighbor = Node(next, current\_node)
* *# Check if the neighbor is in the closed list*
* **if**(neighbor **in** closed):
* **continue**
* *# Generate heuristics (Manhattan distance)*
* neighbor.g = abs(neighbor.position[0] - start\_node.position[0]) + abs(neighbor.position[1] - start\_node.position[1])
* neighbor.h = abs(neighbor.position[0] - goal\_node.position[0]) + abs(neighbor.position[1] - goal\_node.position[1])
* neighbor.f = neighbor.g + neighbor.h
* *# Check if neighbor is in open list and if it has a lower f value*
* **for** node **in** open:
* **if** (neighbor == node **and** neighbor.f > node.f):
* **continue**
* *# Everything is green, add neighbor to open list*
* open.append(neighbor)
* *# Return None, no path is found*
* **return** **None**
* *# The main entry point for this module*
* **def** main():
* *# Get a map (grid)*
* map = {}
* chars = ['c']
* start = **None**
* end = **None**
* width = 0
* height = 0
* *# Open a file*
* fp = open('maze-grid.txt')
* *# Loop until there is no more lines*
* **while** len(chars) > 0:
* *# Get chars in a line*
* chars = [str(i) **for** i **in** fp.readline().strip()]
* *# Calculate the width*
* width = len(chars) **if** width == 0 **else** width
* *# Add chars to map*
* **for** x **in** range(len(chars)):
* map[(x, height)] = chars[x]
* **if**(chars[x] == '@'):
* start = (x, height)
* **elif**(chars[x] == '$'):
* end = (x, height)
* *# Increase the height of the map*
* **if**(len(chars) > 0):
* height += 1
* *# Close the file pointer*
* fp.close()
* *# Find the closest path from start(@) to end($)*
* path = astar\_search(map, start, end)
* print()
* print(path)
* print()
* draw\_grid(map, width, height, spacing=1, path=path, start=start, goal=end)
* print()
* print('Steps to goal: **{0}**'.format(len(path)))
* print()
* *# Tell python to run main method*
* **if** \_\_name\_\_ == "\_\_main\_\_": main()
* [(11, 2), (11, 3), (12, 3), (13, 3), (13, 4), (13, 5), (13, 6), (13, 7), (14, 7), (15, 7), (15, 8), (15, 9), (16, 9), (17, 9), (17, 10), (17, 11), (18, 11), (19, 11), (20, 11), (21, 11), (21, 12), (21, 13), (20, 13), (19, 13), (18, 13), (17, 13), (16, 13), (15, 13), (14, 13), (13, 13), (13, 14), (13, 15), (14, 15), (15, 15), (15, 16), (15, 17), (15, 18), (15, 19), (14, 19), (13, 19), (12, 19), (11, 19), (10, 19), (9, 19), (8, 19), (7, 19), (6, 19), (5, 19), (5, 18), (5, 17), (6, 17), (7, 17), (7, 16), (7, 15), (8, 15), (9, 15), (10, 15), (11, 15), (11, 14), (11, 13), (11, 12), (11, 11), (11, 10), (11, 9), (11, 8), (11, 7), (11, 6), (11, 5), (10, 5), (9, 5), (8, 5), (7, 5), (6, 5), (5, 5), (4, 5), (3, 5), (2, 5), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (1, 11), (2, 11), (3, 11), (3, 10), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9), (9, 10), (9, 11), (9, 12), (9, 13), (8, 13), (7, 13), (7, 12), (7, 11), (6, 11), (5, 11), (5, 12), (5, 13), (4, 13), (3, 13), (2, 13), (1, 13), (1, 14), (1, 15), (2, 15), (3, 15), (3, 16), (3, 17), (2, 17), (1, 17), (1, 18), (1, 19), (1, 20), (1, 21), (1, 22), (1, 23), (2, 23), (3, 23), (4, 23), (5, 23), (5, 22), (5, 21), (6, 21), (7, 21), (8, 21), (9, 21), (10, 21), (11, 21), (12, 21), (13, 21), (14, 21), (15, 21), (15, 22), (15, 23), (14, 23), (13, 23), (12, 23), (11, 23), (11, 24), (11, 25), (12, 25), (13, 25), (13, 26), (13, 27), (12, 27), (11, 27), (11, 28), (11, 29), (11, 30), (11, 31), (10, 31), (9, 31), (9, 32), (9, 33), (9, 34), (9, 35), (9, 36), (9, 37), (8, 37), (7, 37), (7, 36), (7, 35), (7, 34), (7, 33), (6, 33), (5, 33), (5, 32), (5, 31), (5, 30), (5, 29), (6, 29), (7, 29), (8, 29), (9, 29), (9, 28), (9, 27), (9, 26), (9, 25), (9, 24), (9, 23), (8, 23), (7, 23), (7, 24), (7, 25), (7, 26), (7, 27), (6, 27), (5, 27), (4, 27), (3, 27), (2, 27), (1, 27), (1, 28), (1, 29), (1, 30), (1, 31), (1, 32), (1, 33), (2, 33), (3, 33), (3, 34), (3, 35), (4, 35), (5, 35), (5, 36), (5, 37), (5, 38), (5, 39), (6, 39), (7, 39), (8, 39), (9, 39), (10, 39), (11, 39), (11, 38), (11, 37), (12, 37), (13, 37), (13, 38), (13, 39), (14, 39), (15, 39), (16, 39), (17, 39), (18, 39), (19, 39), (19, 38), (19, 37), (18, 37), (17, 37), (16, 37), (15, 37), (15, 36), (15, 35), (14, 35), (13, 35), (13, 34), (13, 33), (14, 33), (15, 33), (15, 32), (15, 31), (15, 30), (15, 29), (16, 29), (17, 29), (17, 30), (17, 31), (17, 32), (17, 33), (17, 34), (17, 35), (18, 35), (19, 35), (19, 34), (19, 33), (19, 32), (19, 31), (20, 31), (21, 31), (21, 32), (21, 33), (22, 33), (23, 33), (23, 34), (23, 35), (22, 35), (21, 35), (21, 36), (21, 37), (21, 38), (21, 39), (22, 39), (23, 39), (24, 39), (25, 39), (25, 38), (25, 37), (25, 36), (25, 35), (25, 34), (25, 33), (25, 32), (25, 31), (24, 31), (23, 31), (23, 30), (23, 29), (22, 29), (21, 29), (21, 28), (21, 27), (21, 26), (21, 25), (22, 25), (23, 25), (24, 25), (25, 25), (26, 25), (27, 25), (27, 26), (27, 27), (27, 28), (27, 29), (28, 29), (29, 29), (29, 30), (29, 31), (29, 32), (29, 33), (29, 34), (29, 35), (29, 36), (29, 37), (29, 38), (29, 39), (30, 39), (31, 39), (31, 38), (31, 37), (32, 37), (33, 37), (33, 38), (33, 39), (34, 39), (35, 39), (36, 39), (37, 39), (38, 39), (39, 39), (40, 39)]
* #################################################################################
* #.#...#....@....#...................#...#.........#.......#.............#.......#
* #.#.#.#.###+###.#########.#########.#.#####.#####.#####.#.#.#######.###.#.#####.#
* #...#.....#+++#.#.........#.#.....#.#...#...#...#.......#.#.#.......#.#.#.#...#.#
* #############+#.#.#########.#.###.#.###.#.###.#.#.#######.###.#######.#.#.#.#.#.#
* #+++++++++++#+#...#.#.....#...#...#...#.#.#.#.#...#...#.......#.......#.#.#.#.#.#
* #+#########+#+#####.#.#.#.#.###.#####.#.#.#.#.#####.#.#########.###.###.###.#.#.#
* #+#........+#+++#...#.#.#.#...#.....#.#.#.#...#.#...#.......#.....#.#...#...#...#
* #+#########+#.#+###.#.#.#####.###.#.#.#.#.#.###.#.#########.#####.#.#.###.#####.#
* #+#+++++++#+#.#+++#...#.#.....#.#.#.#...#.#.....#.#.....#.#...#...#.......#...#.#
* #+#+#####+#+#.###+#####.#.#####.#.#.###.#.#######.###.#.#.###.#.###########.#.#.#
* #+++#+++#+#+#...#+++++#.#.......#.#.#...#.....#...#...#.....#.#.#...#...#...#...#
* #####+#+#+#+#########+#.#######.#.###.#######.#.###.#########.###.#.#.#.#.#######
* #+++++#+++#+#+++++++++#.......#.#...#.#.#.....#.#.....#.......#...#.#.#.#.#.....#
* #+#########+#+#########.###.###.###.#.#.#.###.#.#.###.#.#######.###.#.###.#.###.#
* #+++#.#+++++#+++#.....#.#.#...#.#.#.....#...#.#.#...#.#...#...#...#.#.#...#...#.#
* ###+#.#+#####.#+#.#.###.#.###.#.#.#####.###.###.#####.###.#.#.#.###.#.#.#####.#.#
* #+++#+++#.....#+#.#.#...#...#.....#...#.#...#...........#.#.#...#...#.......#.#.#
* #+###+#########+#.#.#.###.#.#####.#.#.###.###.###########.#.#####.#########.###.#
* #+#..+++++++++++#.#.......#.#...#.#.#...#.#...#.#.......#.......#.#...#.....#...#
* #+#.#############.#########.#.#.###.###.#.#.###.#.#####.#.#######.#.#.#.#####.#.#
* #+#.#+++++++++++#.#.#.#.....#.#.....#...#.#.....#...#.#.#.#.#...#.#.#.#.#.....#.#
* #+###+#########+#.#.#.#######.#######.###.#####.###.#.#.#.#.###.#.#.#.#.#####.#.#
* #+++++#+++#+++++#...#.........#.....#...#.....#...#...#.#.....#.#...#.#.#.....#.#
* #.#####+#+#+#######.###########.#######.#.#######.###.#.###.###.#####.#.#.#####.#
* #.....#+#+#+++#...#.#+++++++#.........#.#...#.......#.#.#...#...#.....#.#.#...#.#
* #######+#+###+#.###.#+#####+#.#####.###.#.#.#.#######.#.#####.###.#####.#.###.#.#
* #+++++++#+#+++#.....#+#...#+#...#.#.....#.#.#.#.#.....#...#...#...#.....#...#.#.#
* #+#######+#+#.#####.#+###.#+###.#.#######.#.#.#.#.#######.#.###.#.###.#####.#.#.#
* #+#.#+++++#+#.#+++#.#+++#.#+++#...#.#...#.#...#.#.....#.#...#...#...#.......#...#
* #+#.#+#####+#.#+#+#####+#.###+###.#.#.#.#.#####.#####.#.#####.#####.#########.###
* #+#..+#..+++#.#+#+#+++#+++#.#+#...#...#.#.#...#.....#...#.#...#...#.....#...#.#.#
* #+###+###+#.###+#+#+#+###+#.#+#.#######.#.#.#.#####.###.#.#.###.#.#####.###.#.#.#
* #+++#+++#+#.#+++#+#+#+++#+#.#+#.#.......#...#.........#.#...#...#.#...#...#.#...#
* #.#+###+#+#.#+###+#+###+#+#.#+#.###.###.###########.###.#.###.###.###.###.#.###.#
* #.#+++#+#+#.#+++#+++#+++#+#.#+#.....#...#...#.....#.#...#.....#.....#.#...#...#.#
* #.###+#+#+#####+#####+#.#+#.#+#######.###.#.#####.#.#.#############.#.#.###.#.#.#
* #...#+#+++#+++#+++++#+#.#+#.#+#+++#...#.#.#.......#.#.#...#...#...#...#.#.#.#...#
* ###.#+#####+#+#####+#+###+#.#+#+#+#.###.#.#########.#.#.#.#.#.#.#.#####.#.#.#####
* #...#+++++++#+++++++#+++++..#+++#+++++++$...........#...#...#...#.......#.......#
* #################################################################################
* Steps to goal: 339
* In [ ]:

Conclusion :

A-star (A\*) is a mighty algorithm in Artificial Intelligence with a wide range of usage. However, it is only as good as its heuristic function( which can be highly variable considering the nature of a problem). A\* is the most popular choice for pathfinding because it’s reasonably flexible.

ROLES :

RAJKAMAL:

MUNTAZIR:

RAHUL:

TECHNOLOGY USED:

Python language

AUTOENCODER

Libraries:numpy , pandas , TensorFlow , keras , skylearn