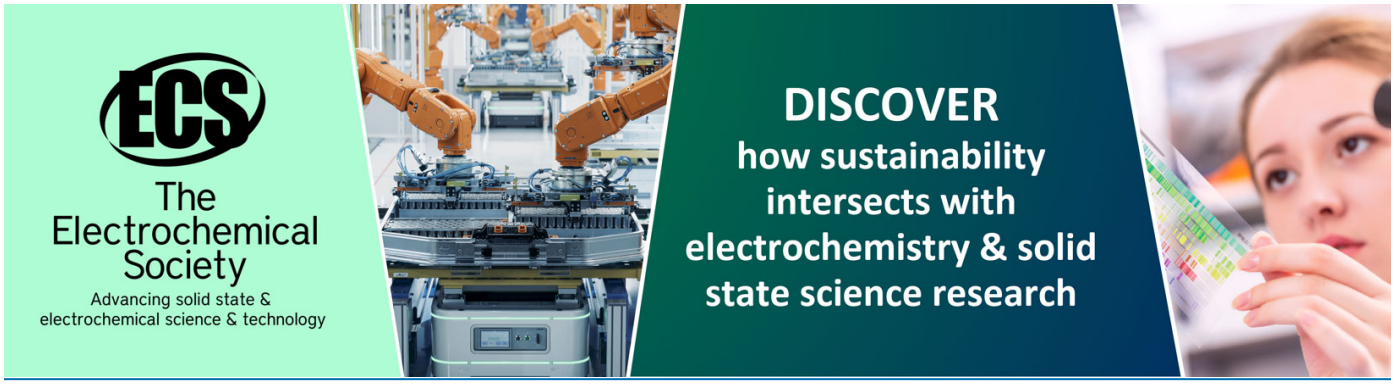
Application of A-Star Algorithm on Pathfinding Game



Journal of Physics: Conference Series **1898** (2021) 012047 doi:10.1088/1742-6596/1898/1/012047

Application of A-Star Algorithm on Pathfinding Game

Ade Candra[1*], Mohammad Andri Budiman[1], Rahmat Irfan Pohan[1]

[1]Faculty of Computer Science and Information Technology, Universitas Sumatera Utara

*ade_candra@usu.ac.id

Abstract. Pathfinding is the method for determining the shortest path. Pathfinding is widely used in various fields of Computer Science. One of them is game development. This research will design and analyze the pathfinding using the A-Star algorithm and implement it on a tree planting game. The Artificial Intelligence in the game will find its own shortest path from the start node to the goal node. The heuristic of A-Star is the Manhattan Distance that allows movement in four different directions. The result shows that the number of visited nodes will increase if the grid has many obstacle nodes, and the average time of the A-Star pathfinding process is 0.0732 seconds.

1. Introduction

Pathfinding has been used in various computer science fields [1,2,3]. In the game development, pathfinding can be used to move objects from their initial place to their destination in the shortest route. As video games develop, pathfinding is also increasingly popular in various games, such as tile-based video games. One of the most popular pathfinding algorithms is the A-Star algorithm.

The basic concept of A-Star is to check the most unexplored locations repeatedly. This algorithm is complete if the location is the destination. If it is not the destination, A-Star will record the neighboring location, and other locations will be explored. The A-Star algorithm's basic terminology is starting point, nodes, open list, closed list, price (cost), and obstacles. A-Star's advantage is using heuristics function as optimization by giving a value to each node that guides A-Star to get the desired solution [4].

A-Star is Dijkstra's development, an algorithm that aims to process efficient path planning among several points (nodes) by using heuristics function. Peter Hart, Nils Nilsson, and Bertram Raphael from Stanford Research Institute introduced this algorithm in 1968 [5]. A-Star uses the best-first search and finds a path with the lowest cost of the initial node given to one destination node. A-Star crosses the graph to build a partial path tree. The leaves of this tree (called a set or open edges) are stored in a priority queue that commands leaf nodes with a cost function, which combines heuristic estimation of costs to reach the destination and the distance traveled from the initial node. A study about the effect of Heuristic values on A-Star‘s performance showed that heuristic value affects searching the nearest route [6]. In other research, A-Star performance can be optimized by improving the heuristic function[7].

In this research, the authors develop a 2D game with a tile-based concept using a grid map on the Android platform. The game has several objects such as enemies, barriers, barriers, trees, and land. A-Star algorithm is used to determine the closest path the enemy can go to the tree. Furthermore, the player must try to protect the tree from the enemy by touching the enemy or using a barrier to keep the tree alive. The game will end when the enemy manages to cut down trees. This research will show the path that the enemy will pass to get to the tree and the time needed to take the path.

2. Method

2.1. Flowchart

The system flowchart of the game is shown in figure 1.

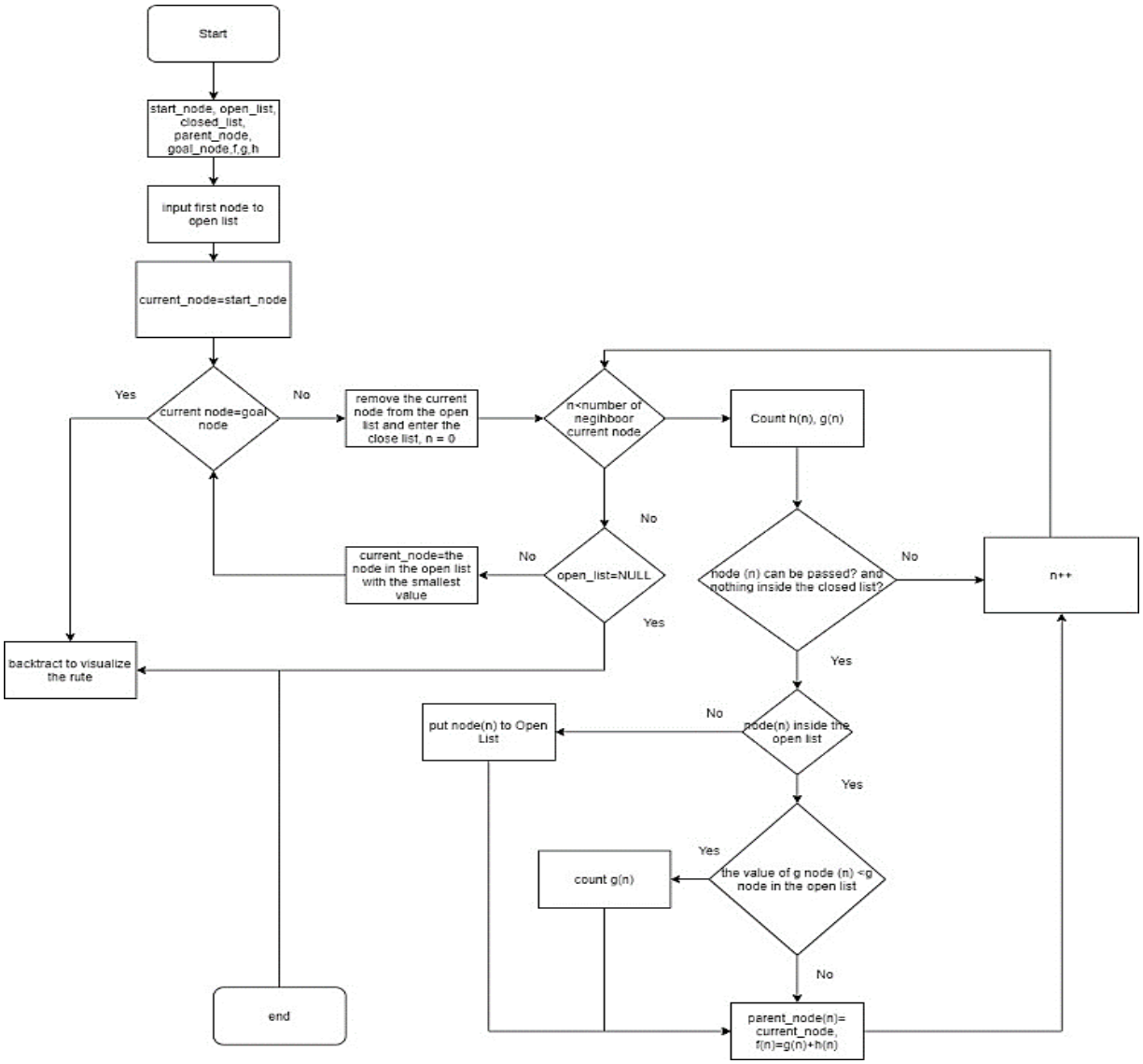


Figure 1. System flowchart

2.2. Pseudocode

Pseudocode is a description of an algorithm in simple structure of computer language to help the user grasp the system. The pseudocode of A-Star algorithm can be write as:

```
function A*(start, goal)
    open_list = set containing start
    closed_list = empty set
    start.g = 0
```

```
start.f = start.g + heuristic(start, goal)
while current is not goal
    current = open_list element with lowest f cost
    remove current from open_list
    add current to closed_list
    for each neighbor of current
        if neighbor not in closed_list
                neighbor.f = current.g + heuristic(neighbor, goal)
                if neighbor is not in open_list
                    add neighbor to open_list
                else
                    openneighbor = neighbor in open_list
                    if neighbor.g < openneighbor.g
                            openneighbor.g = neighbor.g
                            openneighbor.parent = neighbor.parent
        if open_list is empty
                return false
         return backtrack_path(goal)
```

3. Result and Discussion

Game Form is a place for the game process and the player's interaction with the game objects. In this game form, the player can place the tree objects and the protective objects. The pathfinding process of enemy objects also occurs in the game form to route the tree object.

For example, figure 2 has a 16x7 game map grid (Column x Row) in a matrix consist of game object nodes. Each object has an identity, namely land = "" (empty), tree = "T", and stone = "S". There are three tree objects in the matrix, and the enemy cannot step over a stone.

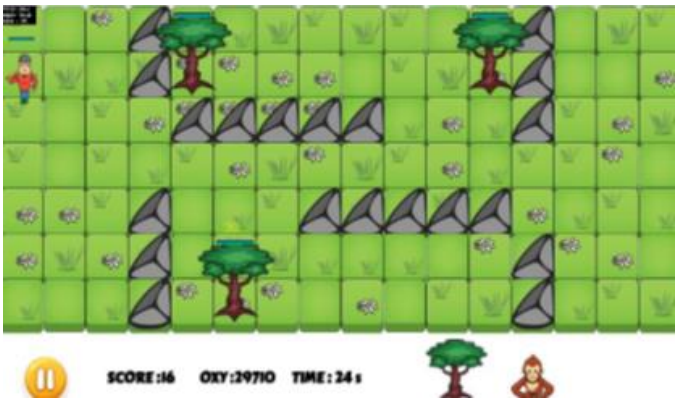


Figure 2. Game grid

Each object's position is defined as (column, row):
1. Enemy E = (1,2)
2. Tree T1 = (5,2)
3. Tree T2 = (6,7)
4. Tree T3 = (12.2)

The following will explain the pathfinding process with the A-Star algorithm.

3.1. Searching

This process aims to find a way for a targeted tree by the enemy. Figure 3, figure 4, and figure 5 are the step-by-step process of searching for tree nodes, in this example, tree node T1. The grid description is as below:

Red: unwalkable node
Yellow: current node
Gray: open node
Orange: closed node
Green color: path

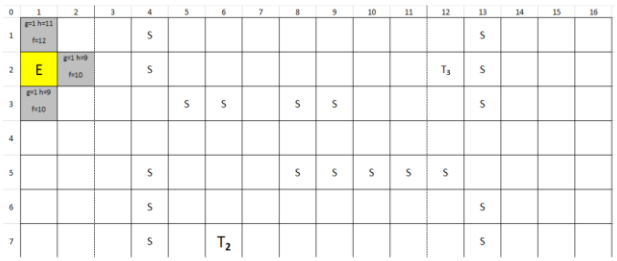


Figure 3. First searching process

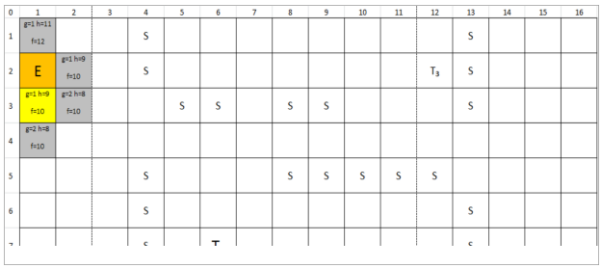


Figure 4. Second searching process

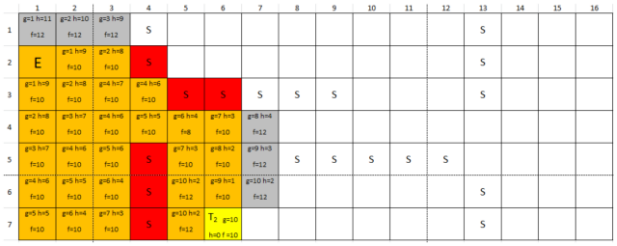


Figure 5. Final searching process

The searching process will finish if the current node is the same as the goal node (destination node).

3.2. Backtracking

After the goal node is obtained, the next process is backtracking. The backtracking process aims to map the route that will be traversed by calling the parent node, starting from calling the parent of goal node to the parent of start node. The backtracking process stops when the parent node is the initial node. The results of the backtracking process can be seen in figure 6.

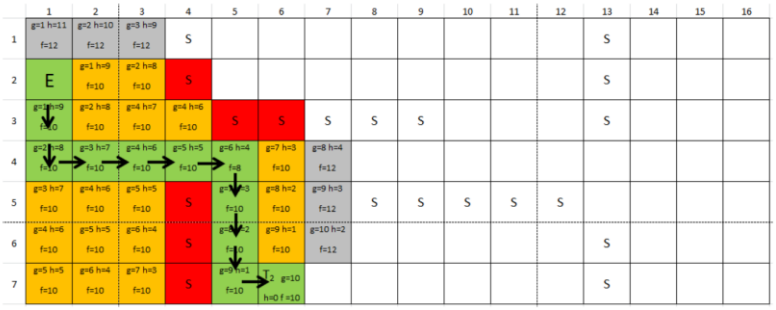


Figure 6. Backtracking

3.3. Nearest Tree Route

To find out which tree will be headed first by the enemy, the enemy must find each tree's route. In the above way, the result of finding the route from the enemy to each tree is:

a. E to T1 has 18 steps
b. E to T2 has 10 steps
c. E to T3 has 15 steps
d. E to T3 has15 steps

Table 1. Heuristic Manhattan Distance

Ex	Ey	Tx	Ty	dx=[Tx-Ex]	dy=[Ty-Ey]	h=dx+dy	Route length
1	2	5	2	4	0	4	18
1	2	6	7	5	5	10	10
1	2	12	2	11	0	11	15

Based on table 1, the tree with the shortest route is the T2 tree, with an estimated heuristic of 10.

3.4. Testing the Pathfinding A-Star

As shown in table 2, the test is carried out to determine the effect of many barriers on many visited nodes, the effect of many visited nodes on the route, and the effect of many visited nodes and route length on time with different start and goal nodes.

Table 2. Number of nodes visited against many roadblocks

No	Start Node (column, row)	Goal Node (column, row)	Number of barriers	Number of visited nodes	Route's length	Time (second)
1	1,2	6,7	20	40	10	0.004
2	1,2	12,2	20	49	15	0.007
3	1,2	2,5	20	69	18	0.009
4	1,1	1,7	45	112	67	0.015
5	1,9	16,16	60	144	84	0.016

4. Conclusion

In this study, the number of visited nodes is directly proportional to the route's length and the pathfinding processing time. Furthermore, the number of barriers also directly proportional to the nodes that are

visited. Finally, the more objects that are obstacles (unwalkable nodes), the longer the processing time needed to find a route from the enemy object node to the tree object node.

References

[1] Mocholi JA, Jaen J, Catala A and Navarro E 2010 *Expert Systems with Applications* **37** 7
[2] Brondani JR, Silva LAL, Zacarias E, de Freitas P 2019 *Expert Systems with Applications* **138**
[3] Mathew GE 2015 *Procedia Computer Science* vol 47 (Amsterdam: Elsevier) pp 262-271
[4] Munir R and Lidya L 1998 *Algoritma dan Pemrogaman* (Bandung: Informatika)
[5] Delling D, Sanders P, Schultes D and Wagner D 2009 *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation* (Berlin: Springer)
[6] Skiena S. S 2008 *The Algorithm Design Manual* (New York: Department of Computer Science State University).
[7] Cui X and Hao S 2011 A*-based pathfinding in modern computer games *International Journal of Computer Science and Network Security* (2011)11.1. pp.125-130