INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

A-star Optimization with Heap-sort Algorithm on NPC Character

Owen Riady Chandra[1], **Wirawan Istiono**[1]*

1 Universitas Mutltimedia Nusantara, Scientia Boulevard Curug Sangereng, Tangerang, Banten, 15810, Indonesia

OPEN ACCESS

Received: 25-04-2022

Accepted: 09-08-2022

Published: 07-09-2022

Citation: Chandra OR, Istiono W (2022) A-star Optimization with Heap-sort Algorithm on NPC Character. Indian Journal of Science and Technology 15(35): 1722-1731. https://doi.org/10.17485/IJST/v15i35.857

* **Corresponding author**.

wirawan.istiono@umn.ac.id

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: To find out whether the A-star algorithm that is optimized with the heap-sort algorithm can reach the target destination faster than the native A-star when applied on NPC to find the path. **Methods**: Comparisons are made by implementing an optimized A-star algorithm with heap-sort and native A-star on an NPC in a dungeon-crawling game genre by applying five different scenarios, where each scenario has a different start position of the NPC, and the target goal is set. The comparison is made by measuring the time required by both algorithms to reach the target goal in milliseconds with Unity engine software and C# language. **Findings**: From the results of experiments and calculation, it was found that the A-star algorithm optimized with the heap-sort algorithm was 50%-80% faster than using only the A-star algorithm in cases that was applied alone on NPC in dungeon-crawling game. **Novelty**: The novelty in this research is the optimization of the A-star algorithm with heap-sort, and then that algorithm is compared with the native A-star algorithm that is applied to NPCs in the dungeon-crawling game genre**.**

Keywords: AStar; Heap Sort; NonPlayable Character; Optimization Algorithm; Pathfinding

1 Introduction

Pathfinding has the goal of finding the shortest path to its destination. There are various algorithms that can be implemented to find a path; one of them is the A-Star algorithm [1,2]. This algorithm is one of the most famous pathfinding algorithms due to its flexibility and ability to be used on various surfaces to find the shortest path by selecting the path with the lowest value [3]. This algorithm is chosen because the A-Star algorithm has been optimized from Dijkstra's algorithm, which finds the path to many destinations, into finding the path to one destination only [4,5]. The A-Star algorithm prioritizes the path that is closest to the destination. There is only one destination for enemy NPC in this research, which is the player's character. The A-star algorithm has a fast computation time, and this algorithm works by determining in advance two points such as points A and B, where point A is the origin point while point B is the destination point [6,7]. Usually, between points A and B, there is a barrier that will determine the number of turns and the shortest or longest path from A to B.

In the pathfinding process, the A-Star algorithm has one weakness, it needs to check every single existing node and calculate the heuristic value. This process takes a lot of time before the enemy NPC starts to move toward its destination [8]. The longer the path that must be passed, the longer it will take to complete the process. To solve this problem, optimization of the algorithm can be done to shorten the calculation process [9]. Research to shorten pathfinding time by A-star algorithm has been conducted by C. Liu, et al. In his research, C.Liu, et al. implemented Improved A-star considering water current, traffic separation, and berthing for vessel path planning, the result is more precise towards and faster its destination [10,11]. By modifying the algorithm's calculation, the time it took to find the path has been successfully reduced by 45-64% compared to the standard A-star algorithm.

In this research, to shorten pathfinding time, the A-Star algorithm is optimized with the heap sort algorithm. The heap sort algorithm works by sorting available data by the lowest value, but it only compares data that has been placed in the child node and parent node [12]. This algorithm is chosen because it fits how the A-Star algorithm works to find the path with the lowest value [13]. This process predicts will helps the A-Star algorithm to calculate faster. Based on the background explanation, it can be concluded that this research aim is to find out if the A-star algorithm that is optimized with the heap-sort algorithm can reach the target destination faster when applied to NPC to find the path in the dungeon-crawling games.

2 Methodology

The first step in this research is to do a literature study, where in this step, the data and information about the A-star algorithm, heap-sort algorithm, and dungeon-crawler games are collected for game design and development. And the second step is analysis and design, where in this step, an analysis of what is needed in the development of this dungeon game is carried out. The required assets will be sorted to determine which one is the most suitable. In addition, the author will also study how the A-star algorithm works in the movement of NPC in the game. After that, the design of the structure and mechanics in the game will be carried out using the Game Design Document.

In the third step in this research, implementation is carried out, where in this step, the game will be developed by implementing the design that has been made and also implementing an optimized A-star algorithm with heap-sort on NPC character. The game will be built on the Unity Engine using the C# programming language. And after the implementation of the algorithm in the dungeon-crawler games, the next step is the testing step, where in this step will make a comparison between the optimized A-star algorithms with heap-sort with native A-star to get which one algorithm is faster in pathfinding cases. And after the result is obtained, the next step will do documentation and calculation to get the percentage result.

The dungeon-crawler games that will be made to test the optimized A-Star algorithm will be named "Escape the Dungeon," where the gameplay is the same as regular dungeon games, where the players will explore dungeons, face monsters, and collect keys to open the exit. This game also will implement formal elements too to support and see performance optimized A-star algorithm in real games dungeon-crawlers. The formal element that will implement in this research is player elements to interact with the environment and enemy NPC and objective elements, where the player must survive and collect all the keys scattered in the dungeon map to open the exit [14]. And also, there are procedure elements, such as; to win the game, the player must be able to collect all the keys to open the exit. The player must defend their life, so they do not run out. If both conditions are met, then the player will be declared the winner. And also have the elements of a rule, such as the player is unable to attack the enemy, and the game has resource elements too, such as Life for the player and a key to opening the door and finish the game.

The conflict in this game is that player must explore dungeons to collect keys, as well as survive so the player is not killed by wandering enemies. And in boundary elements, the player can only explore dungeon maps that have been determined by the game from the start. And the outcome elements, the player will complete the game when all the keys have been collected, open the exit from the dungeon, and exit the dungeon. The flow starts when the player enters the game to the main menu. There are several options in the main menu, which are play, how to play, credits, and exit. If the player selects play, then the game module will be run, which oversees starting the game. If the player selects how to play, a page explaining the controls of the game will be displayed. When the player finishes viewing how to play, the player can select exit and return to the main menu. If the player selects credits, a thank you page will be displayed for contributions from various parties. The player can select exit when they have finished and return to the main menu. The last option is the exit, which will take the player out of the game and complete the gameplay.

The flowchart of gameplay can be seen in Figure 1. The flow begins with the display of the main gameplay screen. After that, a life check will be carried out; if life is less than 1, then the game will end. If life is still more than 1, then the game continues. The player can do several things in the game; the first is move. If the player wants to move, they can press the W, A, S, and D keys or the arrow keys on the keyboard. If the player selects the pause button, the pause menu will open. Players can select a resume to continue the game or the main menu to return to the main menu and end the game. If the player hits the enemy NPC, life will be reduced. If the player has collected all the keys, the door will automatically open, and the player must go through it to win.

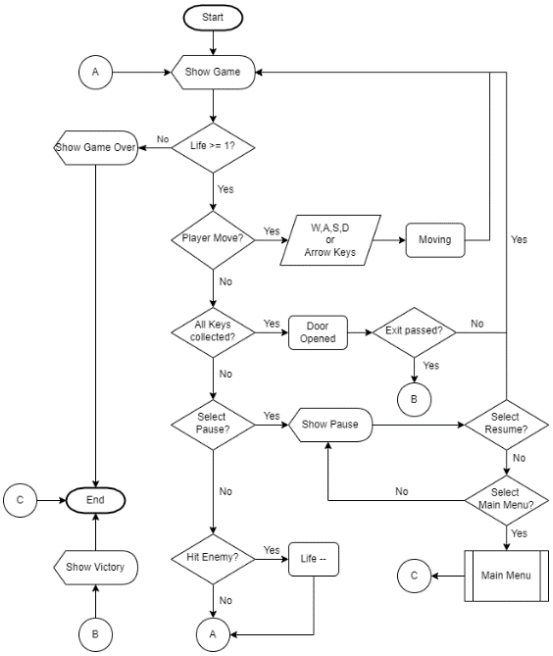

Fig 1. Gameplay Flowchart

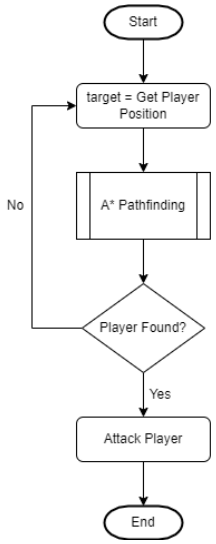

Fig 2. Flowchart NPC in enemy character

The flowchart of the enemy can be seen in Figure 2. Enemy NPC will move towards the position of the player. The movement of the enemy NPC will follow the process of the A-star algorithm module to determine the shortest path that can be taken to get to the player's position. If it reaches the player's position, the enemy NPC will attack.

The flowchart of the A-Star algorithm with heap sort optimization can be seen in Figure 3. The flow begins with the creation of two lists, an open list, and a closed list. The open list will be filled with nodes that will be visited, while the closed list will be filled with nodes that have been visited. The initial position of the NPC will initially be included in the open list. Next, a loop will be carried out to check whether the open list is not zero, which means it is empty. If the open list is zero, then the process will end. If the open list is not zero, then the process will continue following the process in the heap sort module. The value returned by the process in the heap sort module will be moved to the closed list.

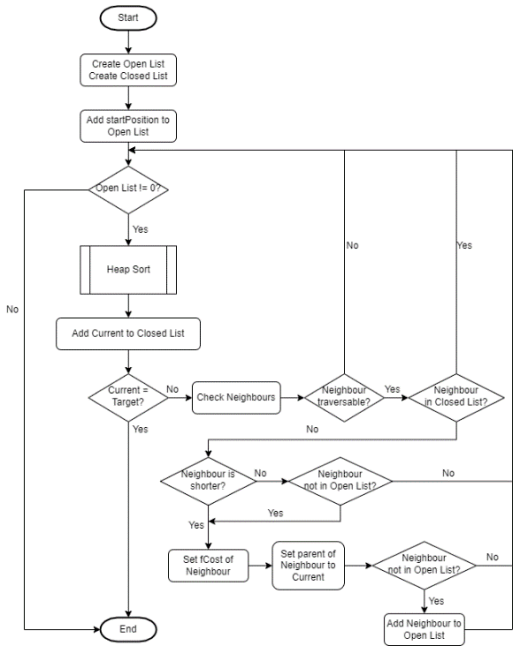

Fig 3. A-star Algorithm with Heap Sort Optimisation Flowchart

Next, check whether current is the intended target position. If yes, then the process will end. If not, it will be checked on the neighbor or neighboring nodes of the current. If the neighbor cannot be passed and is in the closed list, then the process will return to check the value of the open list. If the neighbor can be passed and is not in the closed list, then the process will continue. Neighbors that have lower fCost, which means the path is shorter, will be added to the open list and the current node will be the parent of the neighbor. There is a final check again to make sure whether the neighbor is in the open list or not. If the neighbor is on the open list, the process will return to check the value of the open list. If the neighbor is not on the open list, then the neighbor will be included in the open list. The process will again continue to check the value of the open list.

The flowchart of heap sort can be seen in Figure 4. The flow starts by moving the value from heap length into i. Then it will check whether the value of i is greater than root or not. Otherwise, the process will end. If yes, then the value of i will be moved into maxIndex. Next, check whether i%2 == 0. If not, a parentIndex search will be performed by calculating (i-1)/2. If yes, then the value of i will be subtracted by 1. Then next heap[i] will be compared with heap[maxIndex]. If heap[i] is greater than heap[maxIndex], then the maxIndex value will be i. Otherwise, the process will continue to look for parentIndex by counting (i-1)/2. Then check whether heap[maxIndex] is smaller than heap[parentIndex] or not. If yes, then the value of the two will be swapped and the value of i is reduced by 1, then the process enters a loop checking whether the value of i is greater than the root or not. Otherwise, the process will go straight back to the loop.

The algorithm implemented in this research is the A-star algorithm as the basis for pathfinding from NPCs to reach the target destination. The Snippets of optimization code A-star algorithm for NPC can be seen in Figure 5 to Figure 8.

The code snippet for the declaration of the A-star algorithm node with Heap-sort Optimization can be seen in Figure 5. The node shown in Figure 5 is a class that is used to store data that is useful for creating a grid on the map. The ”Walkable” variable is used to determine whether a node in the map can be traversed or not, and the ”worldPos” variable is to store the position of nodes in the world or map. The ”GridX” variable is used to store the horizontal position of objects in the grid, while ”gridY” is used to store the vertical position of objects in the grid. The ”fCost” method is used for storing data and for calculations to determine the shortest path to the destination.

The code snippet from creating a Grid can be seen in Figure 6, where the “grid” is a class that is useful for building grid tiles on a map. The size of the grid tiles can be adjusted according to the map to be built. In addition, the Grid class also functions to find neighbor nodes and change the position of nodes in the world into grid coordinates that are useful for finding the positions of enemy NPCs and players. Both of this information will be used by the Pathfinding class.

The code snippet of the heap sort can be seen in Figure 7, where the Heap sort code serves to optimize the work of pathfinding, which initially has to check every node in openSet to find the node with the lowest fCost. Heap sort works by comparing the value of the child node with the parent node. If the value of the child node is less than the parent node, swapping will be performed.

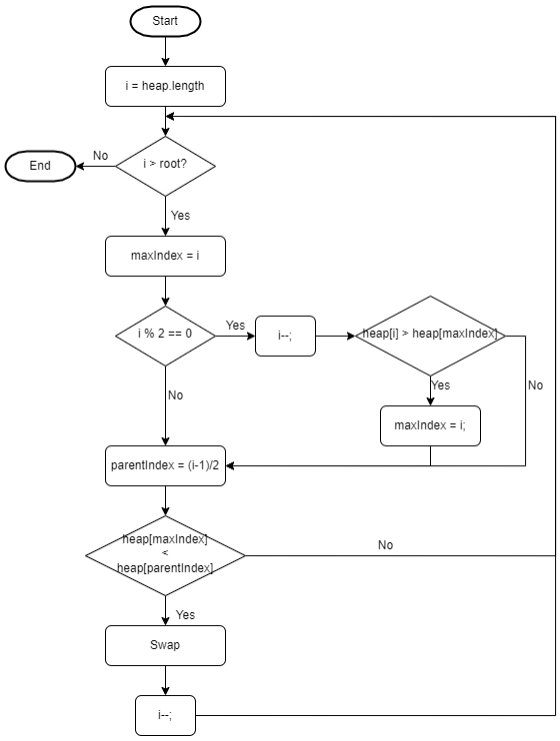

Fig 4. Heap Sort Flowchart

```
public class Node : IHeapItem<Node>
{
    public bool walkable; //walkable nodes
    public Vector3 worldPos; //node position in world
    public Node parent;

    public int gCost;
    public int hCost;
    int heapIndex;

    public int gridX; //node in grid
    public int gridY; //node in grid

    public Node(bool _walkable, Vector3 _worldPos, int _gridX, int _gridY) {
        walkable = _walkable;
        worldPos = _worldPos;
        gridX = _gridX;
        gridY = _gridY;
    }

    public int fCost{
        get{
            return gCost + hCost;
        }
    }
```

Fig 5. Code Snippets of Declaration Node A-star Algorithm with Heap Sort Optimization

```
void CreateGrid(){
    grid = new Node[gridSizeX,gridSizeY];
    Vector3 worldBottomLeft = transform.position - Vector3.right * gridWorldSize.x/2
    - Vector3.up * gridWorldSize.y/2;

    for(int x = 0; x < gridSizeX; x++){
        for(int y = 0; y < gridSizeY; y++){
            Vector3 worldPoint = worldBottomLeft + Vector3.right * (x*nodeDiameter+nodeRadius)
            + Vector3.up * (y*nodeDiameter+nodeRadius);
            bool walkable = !(Physics2D.OverlapCircle(worldPoint, nodeRadius, unwalkableMask));
            grid[x,y] = new Node(walkable, worldPoint, x, y);
        }
    }
}
```

Fig 6. Snippets of Code CreateGrid in A-star Algorithm with Heap Sort Optimization

```
void SortDown(T item){
    while(true){
        int childIndexLeft = item.HeapIndex * 2 + 1;
        int childIndexRight = item.HeapIndex * 2 + 2;
        int swapIndex = 0;

        if(childIndexLeft < currentItemCount){
            swapIndex = childIndexLeft;

            if(childIndexRight < currentItemCount){
                if(items[childIndexLeft].CompareTo(items[childIndexRight]) < 0){
                    swapIndex = childIndexRight;
                }
            }
            if(item.CompareTo(items[swapIndex]) < 0){
                Swap(item, items[swapIndex]);
            }else{
                return;
            }
        }else{
            return;
        }
    }
}
```

Fig 7. Snippets of Code Heap Sort that using for Optimization A-star algorithm

The results of the process carried out will then be returned to the Pathfinding class to continue with the next process.

```
IEnumerator FindPath(Vector3 startPos, Vector3 targetPos){
    Stopwatch sw = new Stopwatch();
    sw.Start();

    Vector3[] waypoints = new Vector3[0];
    bool pathSuccess = false; //check if pathfinding is success

    Node startNode = grid.NodeFromWorldPoint(startPos);
    Node targetNode = grid.NodeFromWorldPoint(targetPos);

    if(startNode.walkable && targetNode.walkable){
        HeapOptimize<Node> openSet = new HeapOptimize<Node>(grid.MaxSize); //heap
        HashSet<Node> closedSet = new HashSet<Node>();
        openSet.Add(startNode);

        while(openSet.Count > 0){
            Node currentNode = openSet.RemoveFirst();
            closedSet.Add(currentNode);

            if(currentNode == targetNode){
                sw.Stop();
                print ("Path found : " + sw.ElapsedMilliseconds*1000 + " µs");

                pathSuccess = true;
                break;
            }
            foreach(Node neighbour in grid.GetNeighbours(currentNode)){
                if(!neighbour.walkable || closedSet.Contains(neighbour)){
                    continue;
                }
                int newMovementCostToNeighbour = currentNode.gCost + GetDistance(currentNode, neighbour);
                if(newMovementCostToNeighbour < neighbour.gCost || !openSet.Contains(neighbour)){
                    neighbour.gCost = newMovementCostToNeighbour;
                    neighbour.hCost = GetDistance(neighbour, targetNode);
                    neighbour.parent = currentNode;

                    if(!openSet.Contains(neighbour))
                        openSet.Add(neighbour);
                }
            }
        }
    }
```

Fig 8. Potongan Kode Pathfinding Algoritma A-star dengan Optimisasi Heap Sort

The code snippet of Pathfinding can be seen in Figure 8, where the Pathfinding method is a class that contains the main function to find the shortest path using the A-star algorithm as the basis. Two main nodes are needed for the pathfinding function to run, the first mode is ”startNode” as the starting position and the other node is ”targetNode” as the end position. The process of running the pathfinding function starts by checking whether startNode and targetNode can be passed or not. After that, a list will be made for node, openSet, and closedSet.The function will continue to run in the loop as long as openSet is greater than zero. Inside the loop, the node in the openSet with the smallest fCost will be checked by the heap sort module and inserted into the closedSet. If ”currentNode” is targetNode, the process will terminate. If not, it will continue to the next checking process.

The next process is checking the ”neighbor” of the currentNode. If the neighbor is not walkable or included in the closedSet, then the check will be continued to the next neighbor, then a new cost calculation will be carried out, which checks whether the new cost is less than the currentNode cost or whether the neighbor is not in the openSet. If one of the conditions is met, then

the data in the neighbor will be replaced with the new one. The last step is to check if the neighbor is not in openSet. Otherwise, the neighbor will be included in the openSet. After that, the process will loop again until the targetNode is found.

3 Results And Discussion

In research conducted by Hong, et al[15], An improved A-star algorithm for long-distance off-road path planning tasks was developed to identify viable paths between start and destination based on a terrain data map generated using a digital elevation model. In this research, optimizing the algorithm is carried out in two phases, data structure and retrieval strategy. The hybrid data structure of minimum heaps and 2D arrays greatly reduces the time complexity of the algorithm. In the second phase, the optimal search strategy was designed does not check whether the goal is reached at the first stage in the search for the globally optimal path, thereby increasing execution efficiency, but by using this method, it takes a lot of time to be able to start the movement until it can find the target goal, this weakness will be more evident when the longer path that needs to be traversed then the more time it will need. Research to shorten the search time with the A-star algorithm has also been carried out by S. Erke, et al[6], by implementing weighted processing by adding new variables to the smallest distance calculation process so that the calculation results can be more precisely targeted towards the target. By modifying the calculation of the A-star algorithm, the result shows the performance of modifying the algorithm is better, stable, and more suitable in the actual application compared to state-of-the-art techniques.

Based on two previous studies, namely modification and improvisation of the A-Star algorithm, it was found that modification or improvisation can increase the speed of finding the path but has weaknesses in some cases in each method. In this study, modification of the A-star algorithm was carried out by merging with the Heap-sort algorithm and applied to a dungeon-themed game as a form of pathfinding simulation, with the hope can improving the performance of the A-star algorithm. The heap sort algorithm was chosen as a supporting algorithm for A-Star improvisation because the heap-sort algorithm is similar to the way the A-star algorithm works, which looks for the path with the smallest value, so by combining the A-star algorithm with heap-sort, it is hoped can improve the A-star algorithm performance to make A-Star algorithm can run faster.

The optimization of the A-star algorithm with heap sort is implemented and visualized to NPCs in a game called ”escape the dungeon,” where the game display can be seen in Figure 9a. The game that was created has the main menu screen, wherein the game has a how-to-play and credit screen and gameplay screen to try optimized A-star algorithm with heap sort in actual games.

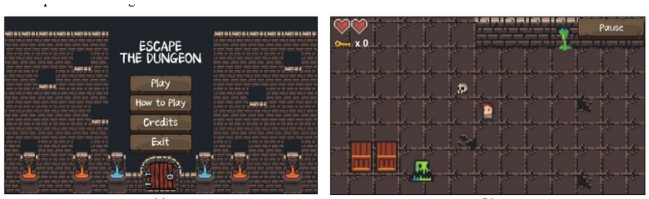

Fig 9. Main Menu screen

When the game start, the gameplay screen can be seen in Figure 9b. In gameplay, the player can move to find keys with the W, A, S, and D keys and the arrow keys on the keyboard. Information on the player's remaining lives and how many keys have been collected can be found in the upper left corner of the screen. For additional gameplay screens, the game also provides a win and lose screen to handle win or lose conditions, where if the player runs out of life due to an enemy attack. The game will automatically end and the player is only given the option to return to the main menu. If the player manages to survive, collect all the keys, and get out of the door then the player will be declared victorious. A victory screen will appear, indicating the game is over and the player can return to the main menu by selecting the button provided.

The testing of the A-Star algorithm follows a scenario. The objects that play a role in this test are enemy NPC and player. An example of the position of enemy NPC and player on the map is shown in Figure 10. The position of the enemy NPC is marked with a black circle and the player's position is marked with a blue circle, while the path to be taken by the enemy NPC is depicted by a grid of yellow tiles. The red grid tiles are the unwalkable part, and the white grid tiles are the walkable sections. The cost used is 10 for horizontal or vertical and 14 for diagonal.

From Figure 10, it can be assumed that the positions of enemy NPCs and players are represented by [X, Y]. The position of the enemy NPC is at [0, 3], and the position of the player is at[5,7]. Then several calculations will be carried out to determine

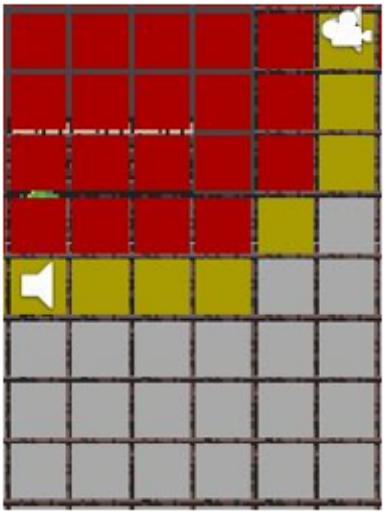

Fig 10. Position Scenario of Enemy NPC and Player

which paths the enemy NPC should take in accordance with the A-star algorithm calculations that have been implemented. These steps are as follows.

1. Calculate the heuristic value (H(x)) using Manhattan distance to obtain the position of the player.
   - Player $^{(5,7)}$ H(x) = 9
2. Calculate F(x) from nodes around [0, 3] towards the position of the player.
   - Node [0, 3]
   - Node [0, 2] F(x) = 80
   - Node $^{(1,2)}$ F(x) = 80
   - Node $^{(1,3)}$ F(x) = 66 (Node selected)
3. Calculate F(x) from nodes around $^{(1,3)}$ towards the position of the player
   - Node $^{(1,3)}$
   - Node $^{(2,2)}$ F(x) = 86
   - Node $^{(2,3)}$ F(x) = 72 (Node selected)
4. Calculate F(x) from nodes around $^{(2,3)}$ towards the position of the player
   - Node $^{(2,3)}$
   - Node $^{(2,3)}$ F(x) = 92
   - Node $^{(3,3)}$ F(x) = 78 (Node selected)
5. Calculate F(x) from nodes around $^{(3,3)}$ towards the position of the player
   - Node $^{(3,3)}$
   - Node $^{(2,4)}$ F(x) = 98
   - Node $^{(3,4)}$ F(x) = 88
   - Node $^{(4,4)}$ F(x) = 78 (Node selected)
6. Calculate F(x) from nodes around $^{(4,4)}$ towards the position of the player
   - Node $^{(4,4)}$
   - Node $^{(3,5)}$ F(x) = 94
   - Node $^{(4,5)}$ F(x) = 84
   - Node $^{(5,5)}$ F(x) = 78 (Node selected)
7. Calculate F(x) from nodes around $^{(5,5)}$ towards the position of the player
   - Node $^{(5,5)}$
   - Node $^{(5,6)}$ F(x) = 78 (Node selected)
8. Calculate F(x) from nodes around $^{(5,6)}$ towards the position of the player
   - Node $^{(5,6)}$
   - Node $^{(5,7)}$ F(x) = 78 (Node selected)

9. Position of player found.

The heap sort optimization implemented on the A-star algorithm was tested in three different scenarios. Testing was executed by counting the time required by the algorithm to find the target, and the time was counted in milliseconds (ms). The map has been divided into a grid of 52 horizontal tiles and 62 vertical tiles. The positions of the enemy NPCs and players in the scenario are represented by [X, Y]. A whole map and obstacle in this game can be seen in Figure 11.

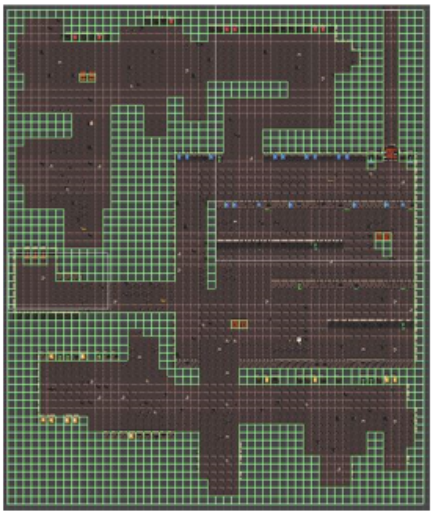

Fig 11. A whole map to test optimized A-star algorithm

The test case was done by creating five scenarios from a different player and NPC position, in the condition the player is not moving from the initialized position. Enemy NPC and player position is shown with [x, y] coordinate, where x and y position is the cell position in the grid map that is shown in Figure 11. Based on the tests carried out with five test scenarios, the results of the time calculation in milliseconds were found for the optimized A-star algorithm with heap sort and A-star native, as shown in Table 1.

Table 1. Scenario 1 of Heap Sort Optimisation Testing

	Case1	Case2	Case3	Case4	Case5
NPC Position	(5,14)	[48, 9] (9)	[36,20]	[30,27]	[26,5] (5)
Player position	[9,33] (9)	[20,54]	[10,47] (10)	[4,41] (4)	[48,50]
Native A-star	4 ms	4 ms	4 ms	5 ms	5 ms
Optimized A-star with Heap-sort	2 ms	1 ms	2 ms	2 ms	1 ms
Percentage	50%	75%	50%	60%	80%

The “percentage” shown in Table 1, is to illustrates the addition of time acceleration in pathfinding. The “percentages” was get from the calculation between optimized A-star with heap sort divided by Native A-star rows. Based on the results shown in Table 1, it can be concluded that optimized A-star with heap-sort is faster 50% to 80% in pathfinding targets than native A-star.

4 Conclusion

The A-star algorithm with heap sort optimization has been successfully implemented in the design and development of dungeon games using the Unity engine and C# programming language. The A-star algorithm with heap sort optimization is implemented in pathfinding of enemy NPC to find the player’s position and then pursue. Based on the calculation of the path search time, the A-star algorithm with heap sort optimization results in an average time of 50% to 80% faster than the native A-star algorithm.

5 Acknowledgment

Thank you to the Universitas Multimedia Nusantara, Indonesia, which has become a place for researchers to develop this journal research. Hopefully, this research can make a significant contribution to the advancement of technology in Indonesia.

References

1) Pardede SL, Athallah FR, Huda YN, Zain FD. A Review of Pathfinding in Game Development. *CEPAT Journal of Computer Engineering: Progress, Application and Technology*. 2022;1(01):47–47. Available from: https://doi.org/10.25124/cepat.v1i01.4863.
2) Wang H, Lou S, Jing J, Wang Y, Liu W, Liu T. The EBS-A* algorithm: An improved A* algorithm for path planning. *PLOS ONE*. 2022;17(2):e0263841–e0263841. Available from: https://doi.org/10.1371/journal.pone.0263841.
3) Istiono W. Effectiveness of Mobile Game-based Education on Algorithm Thinking: Informatic Engineering Case. *International Journal of Emerging Trends in Engineering Research*. 2021;9(3):163–168. Available from: https://doi.org/10.30534/ijeter/2021/02932021.
4) Shi J, Su Y, Bu C, Fan X. A mobile robot path planning algorithm based on improved A*. *Journal of Physics: Conference Series*. 2020;1486(3):032018–032018. Available from: https://doi.org/10.1088/1742-6596/1486/3/032018.
5) Foead D, Ghifari A, Kusuma MB, Hanafiah N, Gunawan E. A Systematic Literature Review of A* Pathfinding. *Procedia Computer Science*. 2021;179(2020):507–514. Available from: https://doi.org/10.1016/j.procs.2021.01.034.
6) Erke S, Bin D, Yiming N, Qi Z, Liang X, Dawei Z. An improved A-Star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems*. 2020;17(5):172988142096226–172988142096226. Available from: https://doi.org/10.1177/1729881420962263.
7) He Z, Liu C, Chu X, Negenborn RR, Wu Q. Dynamic anti-collision A-star algorithm for multi-ship encounter situations. *Applied Ocean Research*. 2022;118:102995–102995. Available from: https://doi.org/10.1016/j.apor.2021.102995.
8) Zhang J, Wu J, Shen X, Li Y. Autonomous land vehicle path planning algorithm based on improved heuristic function of A-Star. *International Journal of Advanced Robotic Systems*. 2021;18(5):172988142110427–172988142110427. Available from: https://doi.org/10.1177/17298814211042730.
9) Istiono W, Suryadibrata A, Waworuntu A, Nusantara UM. List Point Marker Path Finding for Artificial Intelligence Movement in 3D Games. *International Journal of Emerging Trends in Engineering Research*. 2021;9(10):1336–1340. Available from: https://doi.org/10.30534/ijeter/2021/079102021.
10) Liu C, Mao Q, Chu X, Xie S. An Improved A-Star Algorithm Considering Water Current, Traffic Separation and Berthing for Vessel Path Planning. *Applied Sciences*. 2019;9(6):1057–1057. Available from: https://doi.org/10.3390/app9061057.
11) Bayeck RY. Examining Board Gameplay and Learning: A Multidisciplinary Review of Recent Research. *Simulation & Gaming*. 2020;51(4):411–431. Available from: https://doi.org/10.1177/1046878119901286.
12) Chen F, Chen N, Mao H, Hu H. An efficient sorting algorithm - Ultimate Heapsort(UHS). 2019. Available from: https://doi.org/10.48550/arXiv.1902.00257.
13) Abhay G, Abhishek S, Namita G. A variant of Bucket Sort. *10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2019;p. 1–5.
14) Tofterdahl M. Localization Tools in General Purpose Game Engines: A Systematic Mapping Study. *International Journal of Computer Games Technology*. 2021;2021:1–15. Available from: https://doi.org/10.1155/2021/9979657.
15) Hong Z, Sun P, Tong X, Pan H, Zhou R, Zhang Y, et al. Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map. *ISPRS International Journal of Geo-Information*;10(11):785–785. Available from: https://doi.org/10.3390/ijgi10110785.