Toward the Believability of Non-Player Characters (NPC) Movement in Video Games

Rawia Mohamed[1,*], Waleed Al Adrousy[1], Samir Elmougy[1]
[1]Department of Computer Science, Faculty of Computers and Information, Mansoura University, Mansoura 35516, Egypt
Emails: peri_mohamed@yahoo.com ; waleed_m_m@mans.edu.eg ; mougy@mans.edu.eg
Correspondence: peri_mohamed@yahoo.com

Abstract

In video games, artificial intelligence is the effort of going beyond scripted interactions, however complex into the arena of truly interactive systems. To make a game world appear more real, these video games must be responsive, adaptive, and intelligent. For example, in real time strategy games, if there is an enemy seeking/hunting the player, it will be moving in paths, turning around and even maybe jumping in order to find the player. In this case, if the enemy acts/moves more real like human, it will be a benefit for making the game more attractive and exciting. This paper aims to develop a fast, intelligent, and realistic pathfinding approach that makes a user feel that he/she is playing with a human being instead of a machine. To achieve this, this paper presents a Heap Heuristic A* Algorithm as an enhancement of A* algorithm, in which the Chebyshev distance is used to control the smoothness of the resulted path and heapsort algorithm to sort the nodes easily without a lot of memory consumption. Compared to the pervious improved A* algorithms, the proposed algorithm produces a smoother path while consuming less memory to get a final result of human like movement. The experiments results showed that the proposed algorithm reduced the computing time by 66.6% using a grid size of 200*200 compared with A*MOD algorithm. Also, they showed that the proposed work takes almost 91ms to find the path compared to 363 ms and 116 ms when Native A* and A*MOD algorithms are used, respectively, Furthermore, the proposed algorithm performance remains stable in the case of increasing the number of visited nodes, despite the changing order of obstacles.

Keywords: NPC; artificial agent; path finding; games theory

1. Introduction

The increasing popularity and marketing chances of computer games have made them a hot topic for investigation regarding solving many problems, including pathfinding and movements of non-player characters (NPCs). In video games, NPC could be a computer-controlled character, which, as a rule, is an adversary of a human player [1]. Computer games are up-to-date applications that gradually could be improved in a better way for the entertainment and non-entertainment fields, such as education and healthcare. The algorithms of graph search look at a graph to search for general discovery or explicit information. Their goal is to visit as much of the graph as possible without guaranteeing the explored paths optimality. On the other hand, Game graph algorithms use the graph representation of the game to search the possible moves and evaluate their outcomes, helping computers agents to play the game at a high level. There are three important problems that need to be discussed with relevance to computer games: 1) NPC movement, 2) NPC choice making, and 3) NPC learning. A believable NPC carries on in a way that makes it is not obvious from human players [2]. The word “believable” here means that when playing against a computer, you believe that the NPC movement mimics human movement while searching for the target. NPC movement is one of the most significant parts of computer games and Artificial Intelligence (AI), in general. Some of the important aspects of a believable

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

NPC movement are: Improving the game play, making game playing more challenging, maximizing the game popularity among audience, and ensuring that players do not feel bored during playing; NPC shouldn't always beat the human player. NPC movement optimization depends on: Improvement of the pathfinding algorithm, application of the optimization on a suitable environment, which can handle the whole game and the player's needs and ensure that the improved pathfinding algorithm is suitable for making the NPC move as human as possible during searching for the target.

This work focuses on the improvement of the NPC movement by developing a reasonably fast, intelligent, and realistic pathfinding approach that would make the user feel that he/she is playing with a human being with a reasonable level of thinking. In this work, A* algorithm, which is one of the most popular search algorithms having many advanced features, is adopted to be more efficient and faster in finding the path between two points (i.e., the **start** point from which the agent starts its path and the goal or target point, which the object should reach as the ending point of its path) [3]. It is used to solve traditional shortest-distance problems, and its heuristic function is easier for modifying tasks [4]. It is used for the optimality concept, with the goal of finding the lowest cost path between the start and end nodes. In addition, the algorithm is employed to evaluate the costs between nodes during movement. Heapsort is one of the leading sorting strategies that provide linear worst-case running time. It is created using a heap data structure from the array at that point utilizing the heap to sort the array. The use of such an algorithm to sort the nodes inside the A* algorithm is a good choice because it is a fast way of sorting the nodes needed to move from the starting node to the target node. This algorithm also considers minimal memory space (in-place) by maintaining an initial list of items to be sorted. It does not require more memory space to operate [5]. In [6], the authors changed the way of calculating heuristic function and nodes storage methodology. In [7], the authors improved the A* algorithm by sorting the nodes using Heapsort algorithm. In this paper, a hybrid path planning technique, called Heap_Heuristic_A* algorithm, is proposed and developed using both Heapsort algorithm and Chebyshev heuristic function, to get smooth final path.

The remainder of this paper is structured as: Section 2 discusses some related works. Section 3 explains the proposed approach. Section 4 discusses the experiments and results. Section 5 presents the conclusions and future works.

2. Related Work

A* algorithm is an ideal heuristic search algorithm that is a combination of the pathfinding thought of Dijkstra's and the greedy best-first search algorithms [3]. A reachable shortest path could be obtained in an efficient and accurate manner by using an evaluation function. Geographic Information Systems (GIS) and game routing use this algorithm [8]. In [7], A* algorithm was optimized using Heap-sort algorithm in which the result showed 50%-80% faster than standard A* algorithm when applicable to NPC alone in dungeon-crawling game. In [9], Zhao and Liu optimized the A* algorithm to be closer to the human pathfinding performance based on AI and space vector operations. First, this algorithm explores in the start–target node direction. Next, it moves toward the target node. When it encounters a dead end, the algorithm should go back to return to its recent node position and continue searching using the original A* algorithm. If the node is in the horizontal or vertical direction position with the target, it reduces the calculations of the neighborhood node heuristic by 75% (i.e., 75% of the neighborhood calculations. However, when it encounters a dead end (i.e., a node cannot complete its path to the target node), the algorithm should go back to return to its recent node position and continue searching using the original A* algorithm. In [10], Hell et al. discussed procedural generation, in which there are smaller file sizes, less memory usage, and decreased work for developers and designers. They used a modified version of Dijkstra's algorithm depending on the idea that if a node is included in the tree cycle or "visited," it is flagged at that point, and will not be set within the tree vertices once more. Meng and Zhang [8] improved an algorithm using vectors to calculate the direction of the current node neighbors. The results showed that their work reduces 50% of the neighborhood calculations Smołka et al. [6] made two edits on A* algorithm to make the road more natural. They worked on two points to reduce the number of visited nodes. The first point was to change the heuristic function from being calculated using the Manhattan distance to being calculated using the Chebyshev distance. The second edit comprised the post-processing of nodes. The post-processing idea was difficult as it took more time from them in the running time. In [11], Guo and Luo discussed how to improve the heuristic function distance and how to conduct a comparison of the four ways of calculating this distance between the current and target nodes. The authors also mentioned which of the four distances is appropriate for each case of the algorithm upon application. Meanwhile, Harsadi et al. [12] proposed a method that reduces the computation time of dynamic pathfinding through using the flocking behavior to avoid obstacles and applied an Artificial Potential Field by an NPC follower to move toward the target by dividing it into some areas or radius. In [13], Candra et al. introduced a video game using A* algorithm with the Manhattan distance that allows movement in four distinctive directions. The results indicated that the number of visited

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

nodes expands in case the grid faces numerous obstacle nodes. The average time of the A* pathfinding preparation is 0.0732s.

In [14], a literature review is presented for A* pathfinding algorithm including some of its improved versions while comparing A* algorithm with some other search algorithms. Also, it discusses the reason that the original A* algorithm cannot keep up with some current pathfinding demands. However, it introduces a modification on A* algorithm and its heuristic function to make it faster and more accurate. In [15], an optimal approach is presented in which the features are combined from SRC that is used as oracle for providing optimal move from the starting point to the target point. Also, JPS is used to tell how many steps the move can be repeated once the direction is available. The results of the conducted experiments on some grid maps showed strong improvement when using their combined approach. In [16], a pre-processing algorithm is introduced to analyze the geometry of obstacles in the grid map and saves information about the blocked areas in a memory-efficient balanced binary search tree data structure. They used a search algorithm to access the binary search tree during the actual pathfinding for identifying the blocked areas in that map which avoided exploring them. The authors showed that the search time is reduced by 30%, on average. In [17], the authors identified a system that obtained a vehicle's steering system characteristics by applying the determining the preview gain value in the pure pursuit way; the vehicle's steering angle responses were attained by testing EPS actuator. In [18], the authors identified Glowworm Swarm Optimization (GSO), in which they showed that this optimization helps in overcoming different optimization problems and improved GSO performance using hybridization and modification methods.

Some of the previous algorithms did not change the time complexity of the original A* algorithm. As aforementioned, when encountering a dead end, they must go back to their recent node positions and continue search using A* algorithm. Hence, it does not optimize the inability of A* algorithm very well. In addition, improvements must be made in the heuristic function to make the route appear more real to the user. From reviewing the literature, we found that [6] used heuristic function and post processing of nodes to improve A* algorithm, while [7] used Heap sort to improve the whole algorithm performance; this motivated us to make a hybrid path planning technique using both Heapsort with Chebyshev heuristic function.

3. The Proposed Method

The main aim of this paper is to improve the pathfinding A* algorithm through enhancing the smoothness of the final path while improving memory capacity by choosing a good sorting algorithm, Fig. 1 presents the proposed algorithm development life cycle. Unity is a cross-platform game engine that aims to "democratize" game development by making it open to more developers and to enable users to build games and simulation experiences in 2D and 3D. Moreover, it provides an essential scripting API in C# programming language for the Unity editor, as well as the drag and drop functionality [19]. Thus, Unity has been used as the framework, in which the engine/NPC can be developed. Some steps must be taken to reach the development goal of our engine/NPC.

A. Pathfinding A algorithm*

A* algorithm starts by creating two lists, namely, OPEN and CLOSED, which are the basics in this algorithm. OPEN is used for the nodes to be evaluated using the cost function, $f(n) = g(n) + h(n)$, where "n" denotes the node inside the map; $g(n)$, the total distance from the start node to the current one "n," and $h(n)$, the heuristic function used to calculate the distance from the current position "n" to the desired or target location. CLOSED is used for the nodes calculated by the cost function and get out of OPEN. Fig. 2 presents the A* flowchart.

B. Creating the 2D Environment for Pathfinding

This subsection elaborates on the creation of the pathfinding algorithm environment.

1. Creating the grid

Unity3d is used as the platform, in which the practical work was built and tested. In this platform, a 2D environment was created to be tested on any game or any educational project. This platform was created using C# and Unity libraries. Starting from the creation of the grid ground as a surface plane, some cubes would be added to represent the obstacles and create a layer for the unwalkable objects to be assigned to these cubes to inform the object that it cannot walk through them.

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

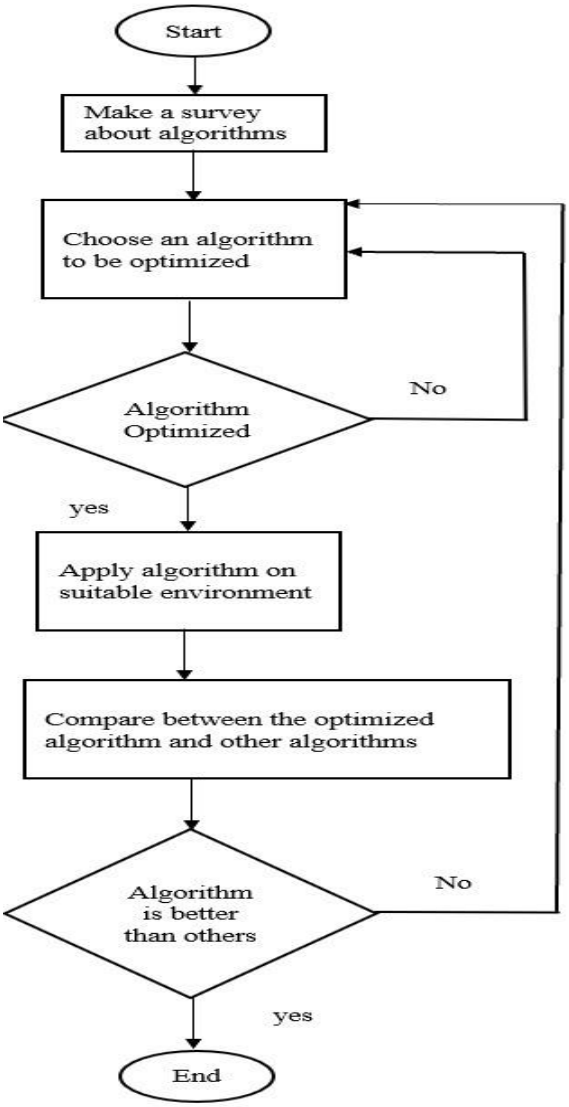

Figure 1: The proposed algorithm development life cycle

1. Adding the A Game Object*

An empty object, called A*, was added and made to take the coordinates of (0,0,0) as the whole grid will be centered around this object. This game object is where the pathfinding and grid scripts would be attached to and where all the algorithm modifications would be applied. The grid class should have the function by which the grid would be created and sized, and the path drawn. The grid should be layered in the x and y axes because the y axis represents the Z axis in a 3D space. In the grid class, the node diameter is used to calculate the grid size as follows:

$$\text{gridSize}(x, y) = \frac{\text{gridWorldSize}(x,y)}{\text{nodeDiameter}} \qquad (1)$$

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

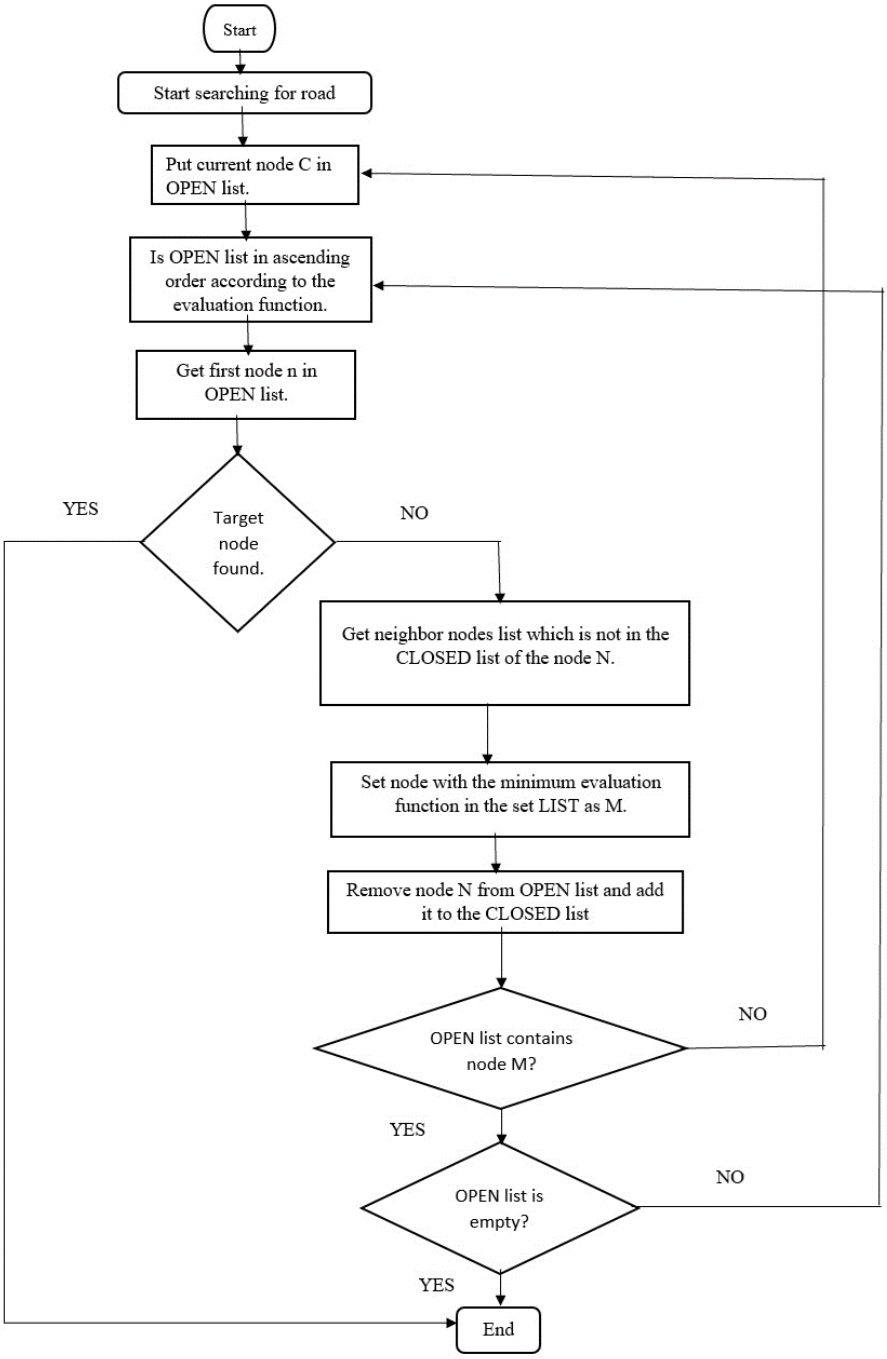

Figure 2: Flowchart of A* algorithm

Equation (1) presents how many nodes should be fit in the world size in both the x and y axes. The grid should be created in both the x and y axes and checked to ensure that the path does not collide with any unwalkable mask. In the grid class, one ensures that the player knows which node it stands on by making the node seen from the world point on which the player stands on, not on the grid point. In other words, the world position is put into a percentage for both the x and y coordinates of how far along the grid it is. It is 0% on the left, 0.5% in the middle, and 1% on the right. This ends the grid class with getting the node from the player position to ensure that the path would be drawn right after the application of the algorithm.

C. Towards the believability of the A algorithm*

In our work, two steps to improve the A* algorithm are considered. The first step is based on using the modified heuristic function from A*MOD algorithm [6], where the heuristic function uses Chebyshev distance instead of Manhattan distance. Manhattan distance is the sum of the absolute wheelbases (distance) of points *x* and *y* in the coordinate system, where (x_n, x_{final}) and (y_n, y_{final}) are the two points in the two dimensions. The

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

distance between them using Manhattan is derived as follows:

$$|x_n - x_{final}| + |y_n - y_{final}| \qquad (2)\ [11]$$

Meanwhile, the Chebyshev distance is the measure in the vector space where the distance between *x* and *y* is the maximum of the absolute value of each coordinate value difference [9]. (x_n, x_{final}) and (y_n, y_{final}) are the two points in the two dimensions. The distance between them using Chebyshev is obtained as follows:

$$h(n) = \max(|x_n - x_{final}|, |y_n - y_{final}|) \qquad (3)\ [11]$$

The change from the Manhattan distance to the Chebyshev distance forces the path to go along the diagonals more regularly and look more common to the player. In the second step, a Heapsort algorithm is applied [7].

1. *A* Algorithm*

In this algorithm, if g_{cost} and h_{cost} are the distances from a given vertex to the start vertex, from a given vertex to the stop vertex (i.e., heuristic function), respectively, then f_{cost}, the evaluation function, is defined as:

$$f_{cost} = g_{cost} + h_{cost} \qquad (4)$$

The slowest part of this algorithm is that each iteration must pass through the entire Open List to try to find the node with the lowest f_{cost}. It has the characteristics: 1) completeness: ensuring obtaining a solution if such one exists, 2) Optimality: finding the optimal path with the lowest cost, 3) Efficiently: exploring fewer nodes by utilizing heuristics. Also, it helps the agent navigate the environment efficiently while avoiding obstacles. In game development, it is used to create intelligent enemies that can follow the player.

2. *Heapsort*

Heap is a better choice for organizing grid nodes in a more efficient manner than that in the original A*. Heap is called max heap if the parent nodes are greater than their child nodes, otherwise it is called min-heap. Fig. 3 presents an overview of the min- and max-heap.

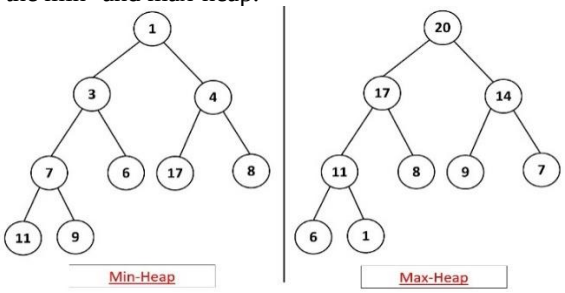

Figure 3: Min- and max-heap

Heapsort is an array of elements with their indices sorted from left to right. If we assign a heap with some indices and sample of f_{cost}, each parent node must be less than both of its child nodes (Fig. 3). A new node is added to the heap by placing it at the end of the heap. We then check if it is lower than its parent node to determine whether it is in the correct position of the heap. It will be swapped with its parent if it is lower than its parent node. The same comparison is repeated until it arrives in its right place, that is, it should be less than both of its child nodes. The number of comparisons is small because the algorithm does not need to check many nodes to find the correct position in the heap. By contrast, if a node must be removed from the heap with the lowest f_{cost} and its position must be filled up, the node at the end of the heap is taken and placed in the removed node place. This node is then compared with its child nodes. If it is greater than any of them, it is swapped with the node of the lowest f_{cost} and so on until it becomes less than its child. This is a fast method of node sorting. As soon as we swap the parent node with one of its child nodes, there will be no need to compare it with any node in the other half of the tree. Equations (5, 6, 7) can be used to find a heap parent node or a child node (Fig. 4). To find the parent node of 11, subtract 1 to give 10 and divide it by 2 to give 5. To find the parent node of 12, subtract 1 to give 11 and divide it by 2 to give 5.5 (i.e., 5 is an integer). Now, to find the children of number 4, multiply it by 2 to give 8 and then add 1 to give 9 for the left child and add 2 to give 10 for the right child.

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

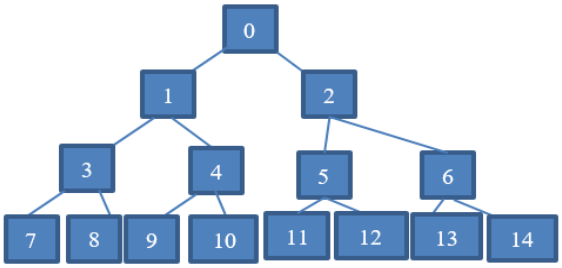

Figure 4: Sorted min-heap tree

The following equations are used to find the parent node of a node *n* [20].

$$\text{Parent node index} = (n - 1) / 2. \quad (5)$$

The index of the left child node, (*n*), is obtained as:

$$\text{Left child node index} = 2n + 1. \quad (6)$$

The index of the right child node, (*n*), is obtained as:

$$\text{Right child node index} = 2n + 1. \quad (7)$$

In [7], the authors improved the native A* algorithm using the Heapsort algorithm, which made the results of the improvement 50%-80% faster than the native A-star algorithm. We followed their steps by adding the Heap-sort algorithm to make our improved algorithm work faster besides changing the way of calculating the heuristic function.

3. Heuristic Function

Additional editing performs slight changes inside the heuristic function penalty. The path goes along the diagonals more regularly when the metric is changed. Furthermore, it becomes more common to a human spectator but tragically produces some zigzags. Here, a vertex penalty is applied, and the penalty is a multiplier included within the computer priorities of corner vertices (the priority decides the processing vertices order within the queue); the multiplier $\sqrt{2}$ is used in [6].

D. The Proposed Adapted Optimized Heap_Heuristic_A Algorithm*

Fig. 5 presents the main architecture of the proposed approach. The proposed optimized Heap_Heuristic_A* algorithm, presented in Alg. 1, was evaluated using a sample project on Unity 2018.2.5f1 (64 bits) [19]. The results indicated the goal we aimed to reach herein starting from the smoothness until the memory-consuming time during processing.

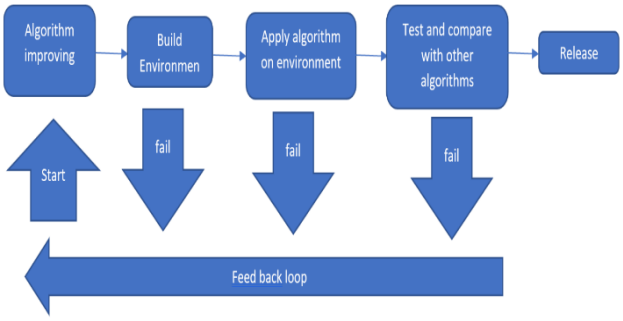

Figure 5: Main architecture of the proposed approach

Algorithm 1: The Proposed adapted Heap_Heuristic_A* algorithm

Algorithm 1: *Heap_Heuristic_A* (Start, Target)*

Input:
G: Grid Map, S: Start node, T: Target node

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

***Output**:*
Ps: The shortest path from the Start to the Target,
Ol: Open List (nodes to be evaluated in Heap),
Cl: Closed List (evaluated nodes)
Procedure:
Ol: = {Start}
Retrace := {}
While Ol.length >0
Ol = HeapSort (Ol)
Current: = Ol [0]
Cl.Add(Current)
Retrace.Add(Current)

If Current = T
Ps = Retrace
Return Ps
End If

For each neighbor of Current
If !Cl. IsTraversable (neighbor)
then Continue
End If

If neighbor.IsCloserThan(Current) or ! Ol.Contains(neighbor)
Ol.Add(neighbor)
If neighbor.IsDiagonalNeighborTo (Current)
M := to √2
Else
M := 1
End If
*G-value := (distance from Start to neighbor) * M*
Neighbor. Parent := Current

If !.Ol.Contains(neighbor)
Ol.Add(neighbor)
End If
End If
End For
End While

E. The Proposed Adapted Optimized Heap_Heuristic_A Algorithm in details*

The proposed algorithm starts by creating OPEN and CLOSED lists, as the basic lists in A* algorithm. OPEN list is used for the nodes to be evaluated using the cost function: $f(n) = g(n) + h(n)$, where "n" is the node inside the map. The role of A*algorithm is to select nodes according to a value '**f**'. At each step, it picks the node/cell that has the lowest '**f**', and process that node/cell. $g(n)$ is the total distance from the start node to the current one "n", $h(n)$ is the heuristic function, used to calculate distance from the current position "n" to the desired or target location which is a smart guess. There are a lot of ways to calculate the heuristic function. One of these ways is Chebyshev distance which is used in the proposed algorithm. CLOSED is used for the nodes that had been calculated by the cost function '*f*' and outside OPEN list. Then, the algorithm checks if there are any nodes inside OPEN list to make it as the current node and thus uses Heapsort algorithm to sort the nodes according to the current node in OPEN list. Next, if it finds that the current node became the target, it returns the whole path as a result and ends the program. Otherwise, it checks whether the neighbours of the current node cannot lead to the target or not, and if it is in the Closed list or not. If this condition is satisfied, then the algorithm skips to the next node and calls it's a new current node and thus it checks whether the new current node has the shortest distance of its neighbours and is not in the Open list or not. If this is true, then the new neighbour will be added to the Open list to be the current node. Next, the algorithm calculates the heuristic function using Chebyshev distance with applying that if a neighbour is in the diagonal path of the

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

current neighbour or not to calculate 'h' value multiplied by $\sqrt{2}$ to make the route looks smoother. All the conditions are iterated until the algorithm arrives the target. At last, it will return the path as a result and end the program.

4. Experiments and Results

A. Improving A Algorithm using the Vector Space inside a Simple Dynamic Environment*

One of the first experiments involved the application of a small sample of the original A* algorithm [21]. using the Unity3d program and editing it into the optimized A* algorithm created by Meng and Jianjun [8] The next step was to improve the results of [8] by adding a dynamic changing environment with both static and dynamic obstacles. These obstacles are represented by the big yellow cubes in Fig. 6. The enemies are represented by the small yellow cubes. The enemies move the obstacles distributed in the landscape. The top-right green cube is the target position. The down-left red cube is the start position. The black line between them is the path resulting from the application of the improved algorithm. The result of this experiment did not sufficiently satisfy the desired goal as it produced an agent that can find its path from the starting point to the destination. In other words, although it made the game feel good, there was neither a satisfaction of any excitement nor the feeling of mimicking the reality and the believability that we are aiming to reach.

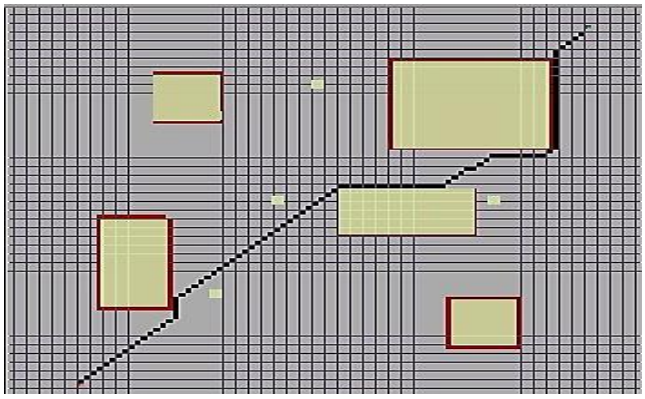

Figure 6: Overview of the first experiment

B. Improving A* Algorithm using only Heapsort Algorithm inside a Dynamic Environment

The second experiment involved the application of a heap optimization on a dynamic environment, where the user can create some buildings as the obstacles that would stop the agent when moving from the starting point to its target. Some issues were found in the infrastructure of the dynamic environment. An example of these concerns is that it did not match the research goal. Fig. 7 presents an overview of this experiment.

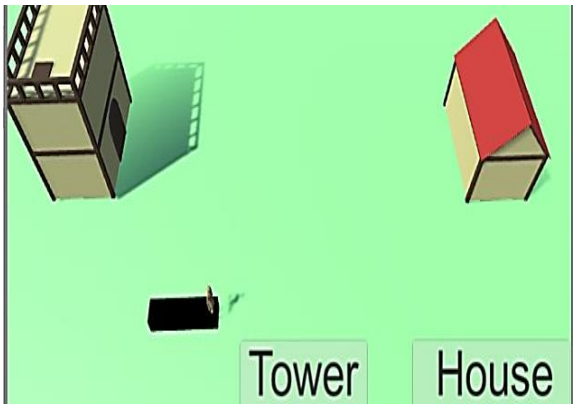

Figure 7: An overview of the second experiment

C. Applying the Proposed Heap_Heuristic_A* Algorithm

Finally, we were able to achieve our desired goal by making a simple project that made the game agent mimic human behavior without taking much space or time. This was obtained by applying two improvements to the A* algorithm, as discussed in Section 3.4. The subsequent sections discuss the comparisons between the proposed Heap_Heuristic_A* algorithm and the improvements done by other researchers.

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

1. Comparison between Native A* Algorithm, A* with Heapsort Algorithm, A*MOD Algorithm, and the Proposed Adapted Heap_Heuristic_A* Algorithm through Path Shape

Fig. 8–11 shows different comparisons between Native A* Algorithm [3], A* with Heapsort Algorithm [7], A*MOD Algorithm [6], and the proposed Heap_Heuristic_A* algorithm. These algorithms were tested using Unity3d in the same environment. The same number of obstacles is represented by the yellow and red cubes with different order. The green ball represents the starting position. The red ball represents the target position. The black line denotes the resulting path reached from the starting point to the target position. The results of testing some grid sizes are shown below. In Fig. 8 the four algorithms were applied on the same environment with some obstacles with a grid size of 50 × 50.

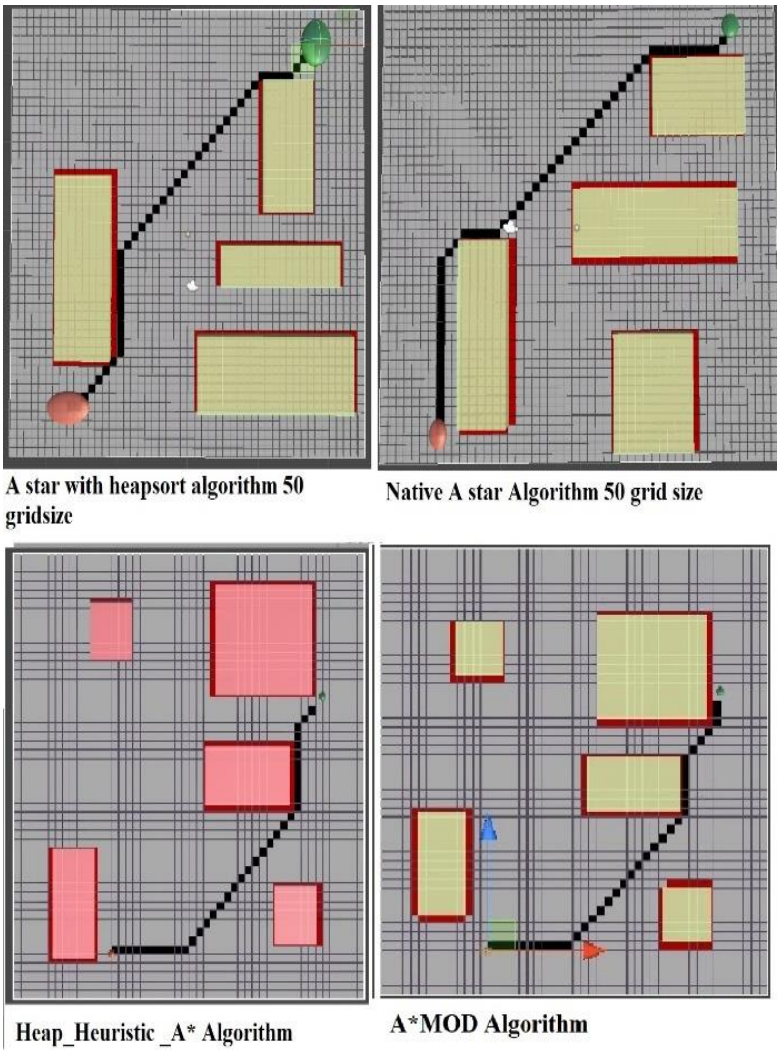

Figure 8: Comparison between the four algorithms using 50 × 50 grid size

In Fig. 9, the four algorithms were applied on the same environment with some obstacles with a grid size of 100 × 100. In Fig. 10, the four algorithms were applied on the same environment with some obstacles with a grid size of 150 × 150, the four paths demonstrate that the path in the Heap_Heuristic_A* algorithm is becoming more natural than the other paths.

In Fig. 11, the four algorithms demonstrate that the movement in the Heap_Heuristic_A* algorithm is getting more natural than it is in the other grid sizes as it makes the user feel like it is a human who searches here and there for a place to go through, not just a machine going in a straight line to find the path.

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

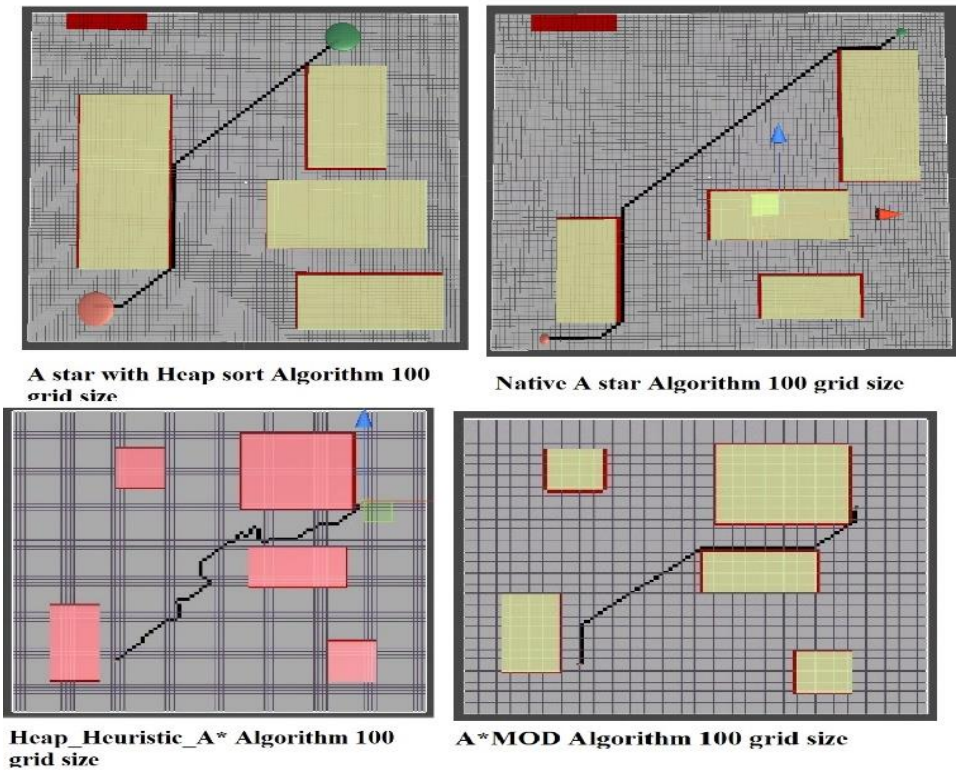

Figure 9: Comparison between the four algorithms using 100 × 100 grid size.

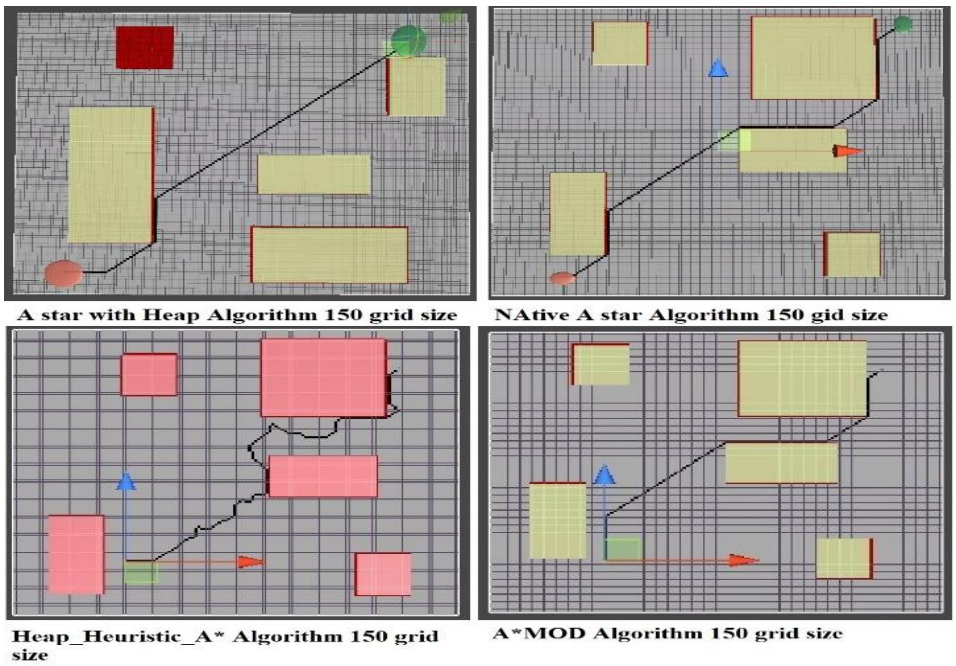

Figure 10: Comparison between the four algorithms using 150 × 150 grid size.

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

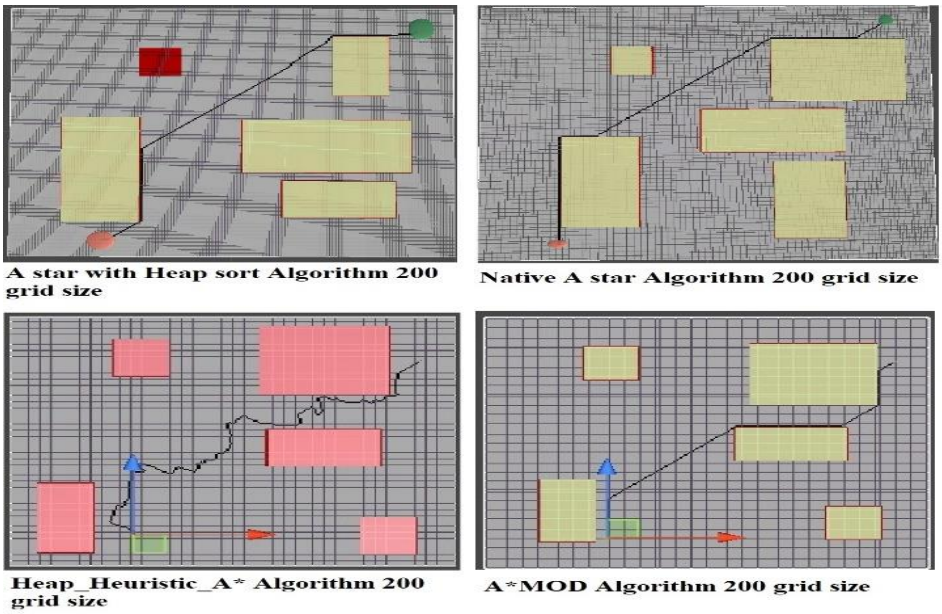

Figure 11: Comparison between the four algorithms using 200 × 200 grid size.

2. Comparison between Native A Algorithm, A* with Heapsort Algorithm, A*MOD Algorithm, and the Proposed Adapted Heap_Heuristic_A* Algorithm (Grid Size and the Number of Visited Nodes)*

The four algorithms were evaluated using a sample project on Unity 2018.2.5f1 (64 bits) [19], applied on intel core i7 with 16gb RAM. Tables 1 and 2 present the comparison results between Native A* Algorithm, A* with Heapsort Algorithm, A*MOD Algorithm, and the Proposed Adapted Heap_Heuristic_A* Algorithm with respect to grid size and number of visited nodes although the order of obstacles was changed it didn't affect the performance of the algorithm.

Table 1: Time consumed for finding a path between two points with varying grid sizes.

Grid size	**Native A***	**A* with heapsort**	**Post-processing (A*MOD)**	**Heap Heuristic A***
50×50	2.0 ms	5.0 ms	3.00 ms	1.00 ms
100×100	22.0 ms	11.0 ms	16.00 ms	8.00 ms
150×150	50.0 ms	42.0 ms	61.00 ms	31.00 ms
200×200	363 ms	58.0 ms	116.00 ms	91.00 ms

In Table 1, the estimated time for finding a path in the Heap_Heuristic_A* algorithm is less than that in all the other three Algorithms, which means that the former is faster and does not take much processing time. The Heap_Heuristic_A* algorithm decreases the computing time by 46.84% compared with A*MOD algorithm in an average case. It reduces the computing time by 66.6%, as long as the grid size is small (similar to the used grid sizes in Table 1).

Table 2: Number of visited nodes in the four algorithms.

Grid size	**Native A***	**A* with Heapsort**	**the post-processing (A*MOD)**	**Heap Heuristic A***
50×50	54	59	39	38
100×100	100	109	77	81
150×150	140	169	135	163
200×200	208	197	187	272

Table 2 demonstrates that the number of visited nodes in the Heap_Heuristic_A* algorithm is more than some

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

algorithms. The increasing number of visited nodes in the proposed algorithm may affect memory capacity; however, from another point of view, this is seen as an advantage as this increase produces some curves in NPC movement which makes the player feel like he/she is playing with a human.

Fig. 12 presents a chart for the time consumption of the Native A* Algorithm, A* with Heapsort Algorithm, A*MOD Algorithm, and the Proposed Adapted Heap_Heuristic_A* Algorithm through different grid sizes with the same obstacles and starting and target node coordinates in every test.

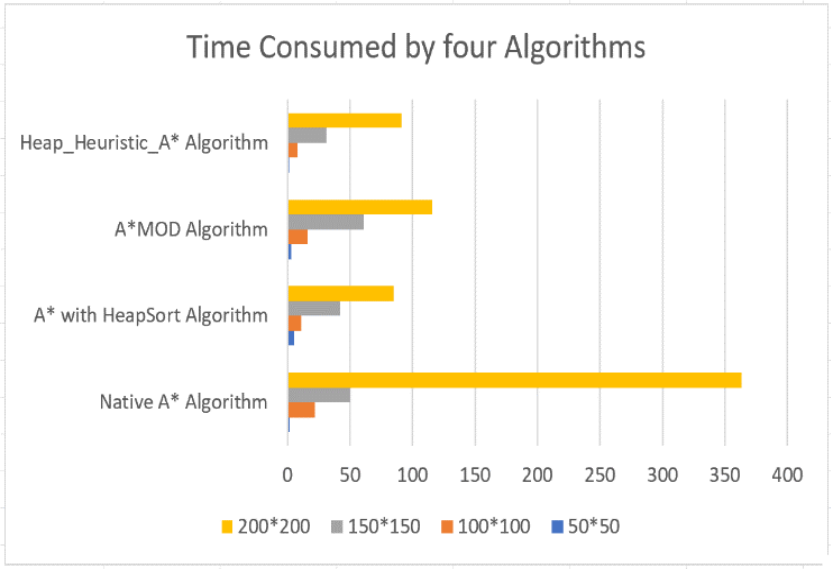

Figure 12: Time consumed by the four algorithms.

Fig. 13 presents a chart depicting the number of visited nodes for seeking a path between two points in Native A* Algorithm, A* with Heapsort Algorithm, A*MOD Algorithm, and the Proposed Adapted Heap_Heuristic_A* Algorithm through different grid sizes with the same obstacles and starting and target node coordinates in every test.

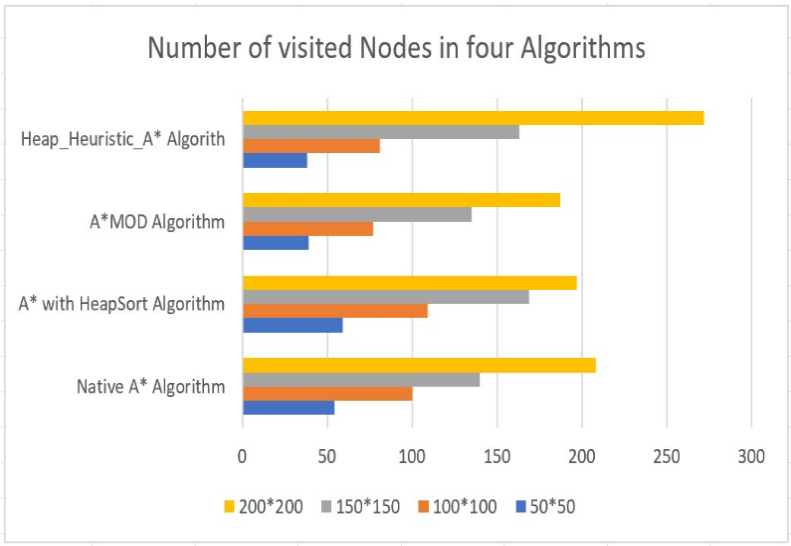

Figure 13: Visited nodes comparisons for the four algorithms.

3. *Comparison of the A*MOD and the proposed adapted Heap_Heuristic_A* algorithms*

The time complexity of the A*MOD algorithm is $O(n^2)$, where n denotes the number of nodes in the Open List. The Heap_Heuristic_A* algorithm is $O(n^2 + n \log n)$; hence, both the Heap_Heuristic_A* and A* MOD algorithms have the same time complexity. The actual running time was minimized (Table 1).

4. *Comparison of A* optimization* [8] *and the proposed adapted Heap_Heuristic_A* algorithms*

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

A* optimization algorithm in [8] was built on the concept of the direction factor, whereas our adapted Heap_Heuristic_A* algorithm was created on the combination of the A* and heapsort algorithms. In [8], 50% of the neighborhood calculations were reduced using vectors to calculate the direction of the neighbors of the current node. However, if there are any backtracking, the A* algorithm will have the memory function of remembering only how to calculate the functions using the basic A* algorithm, not the improved one. In contrast, our adapted Heap_Heuristic_A* algorithm prevents this issue as it can continue with the same efficiency, even when there is backtracking and even when the number of nodes is increased.

5. Comparison of the optimized A and the proposed Adapted Heap_Heuristic_A* algorithms*

In [7], the authors optimized A* algorithm to be closer to the human pathfinding performance based on AI and space vector operations. However, when this algorithm encounters a dead end, it should go back to its recent node position and continue searching using the original A* algorithm. By contrast, our adapted Heap_Heuristic_A* algorithm can handle this issue using the heapsort algorithm.

6. Comparison of the modified A Algorithm for Modular Plant Land Transportation and the Proposed Adapted Heap_Heuristic_A* Algorithm*

The authors in [4] optimized A* algorithm through modifying the optimal transportation route search algorithm for special-purpose status, such as plant transportation. This helps in performing swept path analyses. algorithm is more suitable for plant transportation than the navigation algorithm used in general commercial vehicles. In contrast, the proposed Heap_Heuristic_A* algorithm is more suitable for navigation.

6. Conclusion

In this paper, we introduced the Heap_Heuristic_A* algorithm which uses Chebyshev distance and Heap sort algorithm to improve the smoothness and memory capacity of A* algorithm. The Heapsort algorithm is used to sort the nodes when searching for the target to make the sorting step faster, while Chebyshev distance is used to make the resulting route look smoother. Experimentations results showed that Heap_Heuristic_A* algorithm reduced the time consumed to find the final path for example, when the grid size was 200*200, Heap_Heuristic_A* algorithm needed almost 91ms to find the path while Native A* algorithm took 363 ms, and A*MOD algorithm took 116 ms, Furthermore, increasing the number of visited nodes made the path smoother and stable despite the order of obstacles changing. The zigzags that resulted from the heuristic function, which simulated human movement, were satisfied; however, in some cases, they had to be further reduced to make the road smoother. So, we plan to overcome this problem in the future. Also, we plan to replace Chebyshev distance by using other measures and methods such as the diagonal distance heuristic method.

Funding: “This research received no external funding”

Conflicts of Interest: “The authors declare no conflict of interest.”

References

[1] Marek Kopel, and Tomasz Hajas (2018) Implementing AI for non-player characters in 3D video games. Intelligent Information and Database Systems. DOI:10.1007/978-3-319-75417-8_57

[2] Christian Arzate Cruz, and Jorge Adolfo Ramirez Uresti (2018) HRLB^ 2:A reinforcement learning based framework for believable bots. Applied Sciences. https://doi.org/10.3390/app8122453

[3] Mohsen Tavakoli (2019) Performance Evaluation of Competing Data Structures in Pathfinding. Electronic Theses and Dissertations. Performance Evaluation of Competing Data Structures (uwindsor.ca)

[4] Nam Kyu Kang, Ho Joon Son, and Soo-Hong Lee (2018) Modified A-star algorithm for modular plant land transportation. Journal of Mechanical Science and Technology 32.12 : 5563-5571. https://doi.org/10.1007/s12206-018-1102-z

[5] Phuoc Gia Ngo (2020) Game Application Using A* Pathfinding Algorithm to Help Improving Dementia. Game Application Using A* Pathfinding Algorithm To Help Improving Dementia (calstate.edu)

[6] Jakub Smołka et al. (2019), A* pathfinding algorithm modification for a 3D engine. MATEC Web of Conferences. EDP Sciences 252. DOI:10.1051/matecconf/201925203007

[7] 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.

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023

[8] Qiang Meng, and Jianjun Zhang (2019) Optimization and application of artificial intelligence routing algorithm. Cluster Computing. 22, :8747–8755 DOI:10.1007/s10586-018-1963-z

[9] ZhenGuo Zhao, RunTao Liu (2015) A optimization of A* algorithm to make it close to human pathfinding behavior. International Conference on Electrical, Computer Engineering and Electronics. https://doi.org/10.2991/icecee-15.2015.140

[10] Jacob Hell, Michael Clay, and Hala ELAarag (2017) Hierarchical dungeon procedural generation and optimal path finding based on user input. Journal of Computing Sciences in Colleges 33.1:175-183.

[11] Xiaoyan Guo, and Xun Luo (2018) Global Path Search based on A* Algorithm. International Conference on Transportation & Logistics, Information & Communication, Smart City (TLICSC 2018). Atlantis Press. https://doi.org/10.2991/tlicsc-18.2018.59

[12] Paulus Harsadi, Siti Asmiatun, and Astrid Novita Putri (2021) Dynamic Pathfinding for Non-Player Character Follower on Game. Jurnal Teknik Informatika CIT Medicom https://doi.org/10.35335/cit.Vol13.2021.68.pp51-58

[13] Ade Candra, Mohammad Andri Budiman, and Rahmat Irfan Pohan (2021) Application of A-star algorithm on Pathfinding Game. Journal of Physics: conference series. IOP Publishing 1898. 10.1088/1742-6596/1898/1/012047 (harvard.edu)

[14] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan (2021) A systematic literature review of A* pathfinding. Procedia Computer Science 179 : 507-514. https://doi.org/10.1016/j.procs.2021.01.034.

[15] Salvetti, M., Botea, A., Gerevini, A., Harabor, D., & Saetti, A. (2018, June). Two-oracle optimal path planning on grid maps. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 28, pp. 227-231).

[16] Jubair, F., & Hawa, M. (2020). Exploiting Obstacle Geometry to Reduce Search Time in Grid-Based Pathfinding. *Symmetry*, *12*(7), 1186.

[17] Nasseer K. Bachache, Ali Muhssen Abdul-Sadah, Bashar Ahmed Khalaf. (2023). The Steering Actuator System to Improve Driving of Autonomous Vehicles based on Multi-Sensor Data Fusion. Fusion: Practice and Applications, 11 (1), 77-86.

[18] Muhammad A. S. Mohd Shahrom, Nurezayana Zainal, Mohamad F. Ab. Aziz, Salama A. Mostafa. (2023). A Review of Glowworm Swarm Optimization Meta-Heuristic Swarm Intelligence and its Fusion in Various Applications. Fusion: Practice and Applications, 13 (1), 89-102.

[19] Jeff Craighead, Jennifer Burke, and Robin Murphy (2008) Using the unity game engine to develop sarge: a case study. Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008). Using-the-Unity-Game-Engine-to-Develop-SARGE-A-Case-Study.pdf (researchgate.net).

[20] Carlsson, S. (1992). A note on HEAPSORT. *The Computer Journal*, *35*(4), 410-411.

[21] Weddle, C. (2008). Artificial intelligence and computer games. *Florida State University.*

Doi: https://doi.org/10.54216/FPA.140106
Received: September 22, 2023 Revised: November, 02 2023 Accepted: November 11, 2023