ISSN 2347 - 3983
Volume 9. No.10, October 2021
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter079102021.pdf
https://doi.org/10.30534/ijeter/2021/079102021

List Point Marker Path Finding for Artificial Intelligence Movement in 3D Games

Wirawan Istiono[1], Alethea Suryadibrata[2], Alexander Waworuntu[3]
[1]Universitas Multimedia Nusantara, Indonesia, wirawan.istiono@umn.ac.id
[2] Universitas Multimedia Nusantara, Indonesia, alethea@umn.ac.id
[3] Universitas Multimedia Nusantara, Indonesia, alex.wawo@umn.ac.id

Received Date : September 04, 2021 Accepted Date : September 25, 2021 Published Date : October 07, 2021

ABSTRACT

In making a path finding algorithm in a 3D game to determine the direction of the NPC agent towards the destination, the Djiksra algorithm, Depth First Search, Breadth First Search and so on, usually the shortest distance is directly proportional to the travel duration to the target point. In this study, a test will be made using a list marker point such as the Djiksra algorithm to get the shortest distance and fastest time to reach the destination, in making this algorithm the C# language is used and the Unity software is used. After experimenting with various list points in different places in two directions, it was found that the distance traveled is always directly proportional to duration. So the selection of the fastest or shortest path can be done with this list point marker algorithm.

Key words: List point marker, Artificial Intelligence, Non playable character, Path Finding.

1. INTRODUCTION

To create a realistic 3D game, the implementation of artificial intelligence on non-playable character (NPC) agents in the game becomes an important and crucial issue. One of the common behaviors we know about npc in games is path finding. The path finding issue in a three-dimensional game is something that often creates many problems, where there are many artificial intelligence path finding algorithms such as the A-Star algorithm, Breadth First Search, Depth First Search and etc [1]. The most important point in path finding is how to get to the destination as fast as possible through the shortest route. But sometimes in the search for the shortest route, it is not necessarily the fastest route, and vice versa, where the fastest route is not necessarily the shortest route. So it is need to make sure in advance in path finding, what do you want to achieve, is it the fastest route or the shortest route to reach the destination location [2], [3].

In this path finding algorithm research, a list of point markers will be used which is useful for the characters can move according to a predetermined path. The purpose of this study is to measure the speed of the object to reach the destination. With the condition that the object must reach the target destination as fast as possible and also have to do facing tracking towards to the next predetermined list point, this is done for the algorithm that has been created can be directly applied to real project games [4]. In this study, this algorithm will be applied to the NPC patrol project, and then the NPC will be able to pursue the target, when the target enters the scope of the NPC sensor, and the NPC will pursue the target up to a certain distance, where when the target is too far from the target. NPC, then the NPC will return to the last list point the target is located.

2. LITERATURE STUDY

2.1 Non Playable Character (NPC)

An NPC is any character in the game that is not controlled by the player. The term NPC comes from traditional tabletop role-playing games where it applies to characters controlled by the gamemaster or referee rather than by other players. In video games, this agent NPC usually means a computer-controlled (not by player) character who has a predefined set of behaviors that could potentially affect gameplay, but is not necessarily an actual product of artificial intelligence [5], [6]. The term non-player character is also used in video games to describe an entity that is not under the direct control of a player. The term carries the connotation that the character is not hostile to the player; Hostile characters are referred to as enemies, mobs, or creeps. In video games, NPC is also sometimes used to mean "unplayable character" or "non-player class" [7], [8].

NPC behavior in computer games is usually written and automatic, triggered by certain actions or dialogues with the player character. In certain multiplayer games such as Neverwinter Nights and Vampire games, or the Masquerade series games, a player acting as Game Master can have both player and non-player characters control their actions to continue the storyline. More complex games [9], [10], such as the aforementioned Neverwinter Nights games, allow players to customize the behavior of NPCs by modifying their default

scripts or creating entirely new ones. In some online games, such as MMORPGs, NPCs may be completely scriptless, and are essentially regular character avatars controlled by game company employees. These "non-players" are often distinguished from player characters by the appearance of an avatar or other visual designation, and often serve as in-game support for new players. In other cases, these "live" NPCs are virtual actors, playing ordinary characters who drive a continuous storyline [11], [12].

In early and less advanced RPGs, NPCs only had monologues. The code directs the appearance of a dialog box, floating text, cutscene, or other means of displaying the NPC's speech or reaction to the player [13]. These kinds of NPC speeches are often designed to give an instant impression of the speaker's character, providing character sketches, but they can also advance the story or illuminate the world around the PC. Similar to this is the most common form of storytelling, non-branching dialogue, in which the NPC's speech is displayed the same as above, but the player character or avatar responds or initiates the speech with the NPC. In addition to the objectives listed above, it allows the development of the player's character [14].

2.2 Path Finding

Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points. This is a more practical variant of solving a maze. This field of research is heavily based on Dijkstra's algorithm for finding the shortest path in a weighted graph. Pathfinding is closely related to the shortest path problem, in graph theory, which examines how to identify the path that best meets several criteria (shortest, cheapest, fastest, etc.) between two points in a large network [15], [16].

In essence, the pathfinding method searches for a graph by starting at one vertex and exploring adjacent vertices until the destination vertex is reached, generally with the intention of finding the cheapest route. While graph search methods such as breadth-first search will find a route if given enough time, other methods, "cruising" the graph, will tend to reach their destination more quickly [17]. An analogy would be someone walking across a room; rather than checking every possible route beforehand, the person will generally walk towards the destination and only deviate from the path to avoid obstacles, and make the deviations as small as possible. The two main problems of pathfinding are, the first to find the path between two nodes in the graph; and the second is the shortest path problem, to find the optimal shortest path. Basic algorithms such as breadth-first and depth-first search solve the first problem by exhausting all possibilities; starting from a given node, they repeat all potential paths until they reach the destination node. This algorithm runs in linear time, where V is the number of vertices, and E is the number of edges between vertices [11].
Quadtrees can be used for hierarchical path searches. The idea was first described by the video game industry, which requires planning on a large map with a low amount of CPU time. The concept of using abstractions and heuristics is older and was first mentioned under the name ABSTRIPS (Abstraction-Based STRIPS) which is used to efficiently search the state space of logic games. A similar technique is the navigation mesh (navmesh), which is used for geometric planning in games and multimodal transport planning used in the problem of a salesman's journey with more than one transport vehicle [2], [12]. A map is separated into groups. At high-level layers, paths between clusters are planned. After the plan is found, the second path is planned in a cluster at a lower level This means that planning is carried out in two stages, namely guided local searches in the original space. The advantage is, the number of nodes is smaller and the algorithm works very well. The disadvantage is that hierarchical pathplanners are difficult to implement [10], [15].

3. RESEARCH METHODOLOGY

The first step in testing the application is to make a simple map by including one object in the form of a 3D capsule as an NPC agent and also several cube objects for walls that are shaped like the letter T. Next, a list of point markers is placed as a path marker that can be traversed by NPC agents. All mappings can be seen in Figure1.

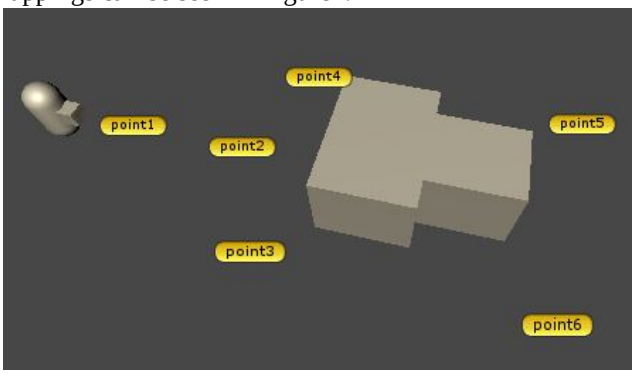

Figure1: Simple map for agent, wall and list point marker

After creating a map for the NPC agent movement experiment, the next step is to prepare a list of connected marker paths and determine the starting point and destination of the point marker where the NPC agent is, in this case, the NPC agent will start from point1 and then the target will go to the target. point6.

```
[SerializeField] Transform[] way1;
[SerializeField] Transform[] way2;

void Start() {
  start_point = GameObject.Find("point1").transform.position;
  finish_point = GameObject.Find("point6").transform.position;
  this.transform.position = start_point;
}
```

Figure2: Code preparation setting begin and target points

In figure 2 it can be seen that there are two ways or paths to get to the destination. Then the start way point and finish way point are determined, and also will determined the placement of NPC agents in the starting position. Because there are two paths that may be traversed by the NPC agent to the destination, each list point marker will be included in an object array or dictionary array.

```
Dictionary<string, float> listDistanceWay = new Dictionary<string, float>();
listDistanceWay.Add("Way1", calc_distance_destination("way1", way1));
listDistanceWay.Add("Way2", calc_distance_destination("way2", way2));
```

Figure3: Dictionary array to marking and calculate distance

The value that inserted into the dictionary array that has been shown in Figure3 is the street name, namely “way1” and “way2” and also the calculation of the distance from the starting point to the destination point to get the shortest distance, this calculation is not based on the number of points, but based on the distance from each point, the function to perform the calculation that has been shown in Figure3 is calc_distance_desitination.

```
float calc_distance_destination(string wayname, Transform[] pathchoosen) {
    float distFromStartToFinish = 0f;
    for (int a = 0; a < pathchoosen.Length-1; a++) {
      Vector3 posCurrent = pathchoosen[a].transform.position;
      Vector3 posTarget = pathchoosen[a + 1].transform.position;
      distFromStartToFinish += Vector3.Distance(posCurrent, posTarget);
    }
    return distFromStartToFinish;
}
```

Figure4: Function for calculating distance for every points marker

After getting the results of calculating the distance between the two paths, the next step is sorting the dictionary array to get the shortest list, but because the dictionary function does not have a sorting facility, then each item in the dictionary list needs to be inserted into the Array List first, so the sorting can be done. based on the value of the items in the dictionary array, how to insert the dictionary array into the Array List, can be seen in Figure5.

```
List<KeyValuePair<string, float>> myList = new
List<KeyValuePair<string, float>>(listDistanceWay);
myList.Sort(
  delegate (KeyValuePair<string, float> firstPair,
  KeyValuePair<string, float> nextPair) {
    return firstPair.Value.CompareTo(nextPair.Value);
  }
);
```

Figure5: Dictionary Array in Array List for sorting based on Value

Next step is to insert the results of sorting the first serial number, namely the items that have the shortest distance to the destination into a variable called "choosen_way", and then in the update loop function, the NPC agent will move from the starting point to the destination point. The NPC agent movement update function can be seen in Figure6.

```
void Update() {
   if (chosen_way == 1) {
        npc_move(way1);
   } else {
      npc_move(way2);
   }
}
```

Figure6: Update function for moving Agen NPC

For the movement of the NPC agent from the initial point to the destination point, it is done with the moveTowards function, where in that function has been calculated and calculated the movement of the distance from the starting point to the destination point. And then after the NPC agent moves, a distance check will be carried out using the Vector3.Distance function, which serves to measure the distance between ObjectA and ObjectB, if the two objects are closer, the return value will decrease, and when the return value has reached a certain distance , then the transfer target point will be changed, so that the NPC agent can move to the next point. And after the NPC agent has moved from one point to another, it will check whether the point is the last point, if not, then the target point will move to the next point, but if the target point is a finish point, it will display the calculation of the travel time from the start point to the finish point. The movement and time display when the NPC agent reaches the finish can be seen in Figure7.

```
void npc_move(Transform[] pathchoosen) {
    Vector3 curPos = this.transform.position;
    Vector3 targetPos = pathchoosen[point_no].transform.position;
    this.transform.position = Vector3.MoveTowards(curPos, targetPos, npc_speed * Time.deltaTime);

    Vector3 posWOy = this.transform.position;
    posWOy.y = pathchoosen[point_no].transform.position.y;
    if (Vector3.Distance(posWOy, targetPos) < 0.1f) {
            if (point_no < pathchoosen.Length - 1) {
              point_no++;
        } else if (point_no == pathchoosen.Length-1) {
              if(!Finished) {
            float finishTime = Time.time - StartTime;
             Debug.Log("Finish Time: " + finishTime);
             Finished = true;
        }
    }
}
```

Figure7: Agen NPC moving code

The next step is to make the NPC agent face forward to the next target point, it is necessary to rotate the NPC agent object. The rotation of the NPC agent is carried out using the

LookRotation function from Quaternion, this code will be entered along with the movement of the NPC agent, namely in the npc_move function, and in Figure 8 is an example of rotation using the LookRotation function.

```
Vector3 posTarget = targetPos - curPos;
this.transform.rotation = Quaternion.LookRotation(posTarget);
```

Figure8: Rotation agen NPC with LookRotation

4. RESULT

After experimenting with the movement of the NPC agent on the mapping in Figure1, where the starting point, namely point1 goes to the target point, namely point6, with the application of two paths, where the first path to be traversed by the NPC agent is point1, point2, point3 and point6 then for the second path, namely point1, point2, point4, point5, point6, with the speed of the NPC agent running is 2px/frame where the frame rate per second is between 120 to 130 fps. It found the result the number of paths and path distances and speeds as shown in Table1.

Table1: Result Table Distance and duration

Path	Point list	Number of Point	Distance	Duration
1	1,2,3,6	4	14.38	7.09 sec
2	1,2,4,5,6	5	20.73	10.19 sec

As can be seen in Table1, the farther the distance traveled and the greater the number of points that must be passed, it will have an impact on the distance and length of time for NPC agents to reach their destination. But what if the point list is changed to be as shown in Figure9, by changing the number of points to be the same.

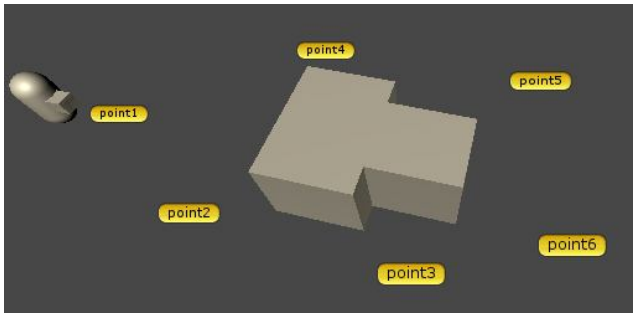

Figure9: Simple map for agent with difference distance list point

As can be seen in Figure9, the mapping made places the start point and target point still the same, namely point1 for start point and point6 for target point, but the difference is the distance between points to the target point and also the number of points between path1 and path2 is the same . The results of the experiment as can be seen in Table2

Table2: Result Table Distance and duration with difference point distance and number of point

Path	Point list	Number of Point	Distance	Duration
1	1,2,3,6	4	15.28	7.59 sec
2	1,4,5,6	4	19.56	9.6 sec

As can we seen in Table 2, it can be concluded that the mileage is always matching with duration, even though the list points are fewer, but if the distance is large, the time needed to get to the destination can be longer. But what if there are more number of points but shorter distances, will the duration of the trip stay longer, in figure10 is an example of mapping where the number of points is more but the distance is closer than the one with fewer number of points but more distance much as shown in Figure10.

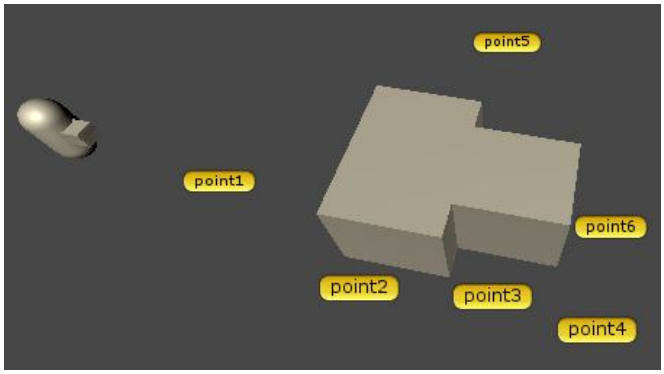

Figure10: Simple map for agent with difference distance list point

In Figure10, the direction of path1 is from point1, point2, point3, point4 and point6, while for path2 it is point1, point5 and point6, and the starting point and target point are still the same, namely point1 for start point and point6 for target point, if done testing with the mapping results above, it will be found the results as shown in Table 3.

Table3: Result Table Distance and duration with difference mapping

Path	Point list	Number of Point	Distance	Duration
1	1,2,3,4,6	5	12.68	6.24 sec
2	1,5,6	3	14.91	7.35 sec

As can be seen in Table3, a small number of points does not always make the distance shorter and the duration of the trip faster.

5. CONCLUSION

From the three experimental results, which are shown in Table1, Table2 and Table3, it can be concluded that the distance traveled is always directly proportional to duration. So the selection of the fastest or shortest path can be done with

this list point marker algorithm. So by using this algorithm, selecting the fastest or shortest path at the start of the game or the beginning of the movement of the NPC agent's action will produce the same result.

ACKNOWLEDGEMENT

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 major contribution to the advancement of technology in Indonesia..

REFERENCES

1. X. Cui and Hao Shi, "Direction Oriented Pathfinding In Video Games," *International Journal of Artificial Intelligence & Applications*, vol. 2, no. 4, pp. 1–11, 2011.
2. M. Mangeruga, A. Casavola, F. Pupo, and F. Bruno, "An underwater pathfinding algorithm for optimised planning of survey dives," *Remote Sensing*, vol. 12, no. 23, pp. 1–17, 2020.
3. N. H. Barnouti, S. S. M. Al-Dabbagh, and M. A. Sahib Naser, "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm," *Journal of Computer and Communications*, vol. 04, no. 11, pp. 15–25, 2016.
4. S. B. Linek, D. Schwarz, G. Hirschberg, M. K. Rust, and D. Albert, "Designing the Non-player Character of an Educational Adventure-game: The Role of Personality, Naturalism, and Color," *Proceedings of the International Technology Education and Development Conference INTED*, no. January, 2007.
5. D. Asha, "Gamification : ' The System Beats Human Huge Critical T hinking ,'" *International Journal of Emerging Technologies in Engineering Research*, vol. 4, no. 8, pp. 72–75, 2016.
6. I. Mabruroh and D. Herumurti, "Adaptive Non Playable Character in RPG Game Using Logarithmic Learning For Generalized Classifier Neural Network (L-GCNN)," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, vol. 4, no. 2, pp. 127–136, 2019.
7. E. V. Soboleva and N. V. Shalaginova, "Simulation of artificial intelligence in a computer game," *Journal of Physics: Conference Series*, vol. 1399, no. 3, 2019.
8. T. Wibowo, "Studi Faktor Pendukung Popularitas Multiplayer Online Battle Arena dengan Pendekatan Kuantitatif," *Ultima InfoSys : Jurnal Ilmu Sistem Informasi*, vol. 12, no. 1, pp. 1–7, 2021.
9. A. El Rhalibi, K. W. Wong, and M. Price, "Artificial intelligence for computer games," *International Journal of Computer Games Technology*, vol. 2009, no. 1, 2009.
10. R. Rismanto, R. Ariyanto, A. Setiawan, and M. Elinggar Zari, "Sugeno Fuzzy for Non-Playable Character Behaviors in a 2D Platformer Game," *International Journal of Engineering & Technology*, vol. 7, no. 4.44, p. 222, 2018.
11. M. B. Nendya, S. Gandang, R. G. Santosa, J. T. Elekto, and F. T. Industri, "Pemetaan Perilaku Non-Playable Character Pada Permainan Berbasis Role Playing Game Menggunakan Metode Finite State Machine," *Journal of Animation & Games Studies*, vol. 1, no. 2, pp. 185–202, 2015.
12. G. Mutaqin, J. N. Fadilah, and F. Nugroho, "Implementasi Metode Path Finding dengan Penerapan Algoritma A-Star untuk Mencari Jalur Terpendek pada Game 'Jumrah Launch Story,'" *Walisongo Journal of Information Technology*, vol. 3, no. 1, pp. 43–48, 2021.
13. Y. T. Mo and S. K. Kim, "Research on the influence of randomness of non-player character interaction behavior on game experience," *International Journal of Engineering Research and Technology*, vol. 12, no. 11, pp. 1986–1991, 2019.
14. H. Warpefelt, *The Non-Player Character: Exploring the believability of NPC presentation and behavior*, no. 16. 2016.
15. A. Suryadibrata, J. C. Young, and R. Luhulima, "Review of Various A* Pathfinding Implementations in Game Autonomous Agent," *IJNMT (International Journal of New Media Technology)*, vol. 6, no. 1, pp. 43–49, 2019.
16. F. Dignum, J. Westra, W. A. Van Doesburg, and M. Harbers, "Games and agents: Designing intelligent gameplay," *International Journal of Computer Games Technology*, vol. 2009, no. 1, 2009.
17. J. B. Clempner, "A shortest-path Lyapunov approach for forward decision processes," *International Journal of Computer Games Technology*, vol. 2009, no. 1, 2009.