EMITTER International Journal of Engineering Technology
Vol. 4, No. 1, June 2016
ISSN: 2443-1168

Review of A* (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game

Moh. Zikky

Game Technology Study Program, Multimedia Creative Department
Politeknik Elektronika Negeri Surabaya, Jl. Raya ITS Sukolilo Surabaya
Email: zikky@pens.ac.id

Abstract

Shortest pathfinding problem has become a populer issue in Game's Artificial Intelligent (AI). This paper discussed the effective way to optimize the shortest pathfinding problem, namely Navigation Mesh (NavMesh). This method is very interesting because it has a large area of implementation, especially in games world. In this paper, NavMesh was implemented by using A* (A star) algorithm and examined in Unity 3D game engine. A* was an effective algorithm in shortest pathfinding problem because its optimization was made with effective tracing using segmentation line. Pac-Man game was chosen as the example of the shortest pathfinding by using NavMesh in Unity 3D. A* algorithm was implemented on the enemies of Pac-Man (three ghosts), which path was designed by using NavMesh concept. Thus, the movement of ghosts in catching Pac-Man was the result of this review of the effectiveness of this concept. In further research, this method could be implemented on several optimization programmes, such as Geographic Information System (GIS), robotics, and statistics.

Keywords: NavMesh, Pac Man, shortest pathfinding, A*

1. INTRODUCTION

Pac-Man has become a legendary video game over 1980s. The most interesting case in Pac-man is the intelligent behavior of three ghosts (Pac-Man's enemies). They walk and catch Pac-Man wherever he goes. Three ghosts are the agents or NPC on which Artificial Intelligence (AI) algorithm is embedded. Previously, most of the algorithm used on AI is Dijkstra [1].

Shortest pathfinding is a determining process of several objects' movement to another position without collision [2]. Actually, this method is not only used in AI game, but also used in other fields, such as Geographic Information System (GIS), robotics, and statistics, among others. Several theories have been delivered to solve the pathfinding problem, such as Finite State Machine (FSM), Graph, Dijkstra, and A*. FSM is the simplest algorithm which can be implemented in this pathfinding problem but its connection is

large and the movement is easy to be predicted. Other algorithm which can be implemented in pathfinding problem is Graph. Graph makes several waypoints depending on the weight of graph value. Other than Graph there is Dijkstra algorithm. It uses greedy principle. Greedy principle on Dijkstra algorithm shows that on each step, we choose the side which has minimum weight and put it into a set of solutions [2].

Today's industrial development requires optimal shortest pathfinding algorithm in solving the problem because their resources have developed very complex. To solve the problems in pathfinding algorithm, game industries have made several research and removed several parts which are not used. Finally, they find the solution namely Navigation Mesh (NavMesh). But, NavMesh can only select a reasonable path for each moving object. Navigation mesh is a technique to represent a game world using polygons. Due to its simplicity and high efficiency in representing the 3D environment, navigation mesh has become a mainstream choice for 3D games [2]. For its implementation, this paper examined NavMesh concept in Unity 3D game engine.

2. RELATED WORKS

To support the basic theory about ghost navigation (NPC in Pac-man), several concepts (research) about basic navigation mesh (NavMesh) and its features on unity 3D are discussed as follows.

2.1 Navigation Mesh

Navigation mesh has become a popular concept which is used in shortest pathfinding problem of 3D games because 3D environment mostly uses polygon structure. In Navigation Mesh, the properties of polygon object or terrain can guarantee a free-walk for a game character as long as the character which stays in the same polygon is a set of complex polygon [2].

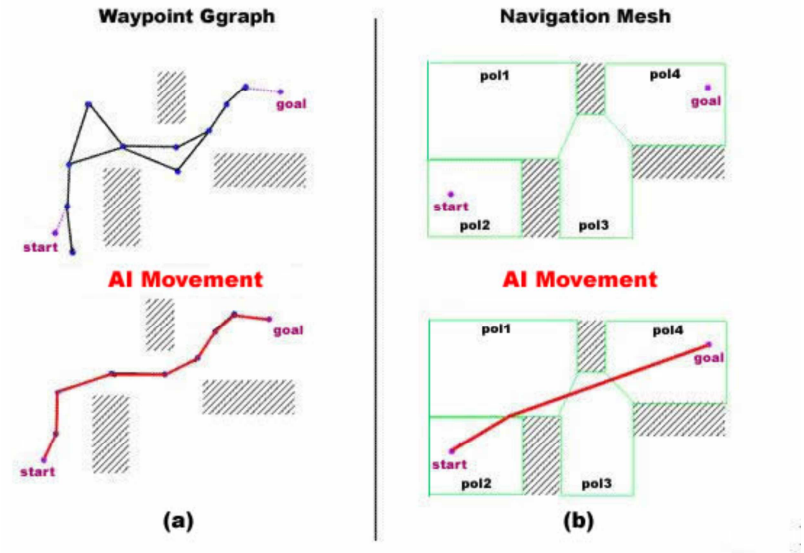


Figure 2.1. Pathfinding with waypoint graph and navigation mesh [3]

As shown in Figure 2.1, there are differences between shortest pathfinding problems which use waypoint Graph and Navigation Mesh. Figure 2.1 (a) shows how graph algorithm takes a path in movement from start to goal. Graph chooses the nearest point based on the weight value which is established. There is no optimization after the process. But, if we compare it with NavMesh concept in Figure 2.1(b), the character moves from the start point to the destination/goal in a straight line. In this case, the start point is not in the same polygon as that of the goal point. The character needs to determine to which polygon it will then go. It repeats this step until both the character and the goal are located in the same polygon. Finally, the character can move to the destination in a straight line [2].

Pathfinding process in navigation mesh can be implemented with several algorithms, but the most effective and popular today's algorithm which is implemented for shortest pathfinding problem is A* (A Star) algorithm. A* is a generic search algorithm that can be used to find solutions for many problems, and pathfinding is one of them. For pathfinding, A* algorithm repeatedly examines the most promising unexplored location it has seen. When the location explored is the goal, the algorithm is finished. Otherwise, it has noted all location around for further exploration. A* is probably the most popular pathfinding algorithm in today's AI (Artificial Intelligence) game(s) [3].

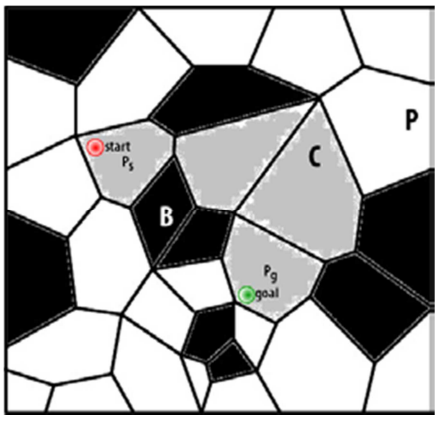


Figure 2.2. Example of NavMesh construction [3]

Figure 2.2 is the example of NavMesh implementation using P symbol as walkable area and B as blocked area/obstacle. In this mesh navigation, polygons with C symbols are chosen for navigation way of start point (Ps polygon) towards the goal point (Pg polygon); while in the navigation mapping, NavMesh needs optimum algorithm to reach the goal. First, A* algorithm chooses the shortest path by making connections among walkable polygons. Therefore, A* is able to detect the walkable polygons from the start point to the goal. Generally, A* chooses one point in every walkable polygon and then connects them until the start and the goal points make one connected line.

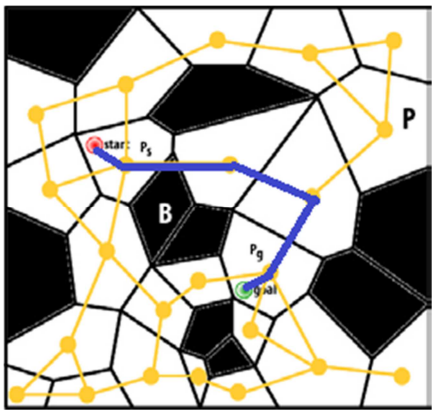
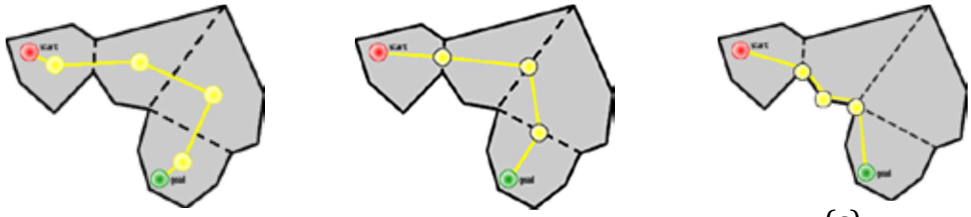


Figure 2.3. Navmesh graph and the shortest path

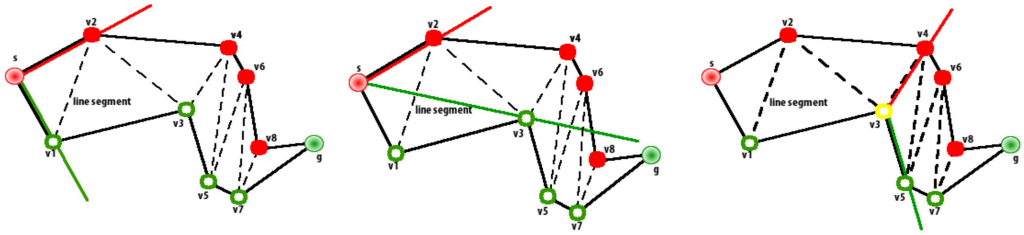
Figure 2.3 shows that the yellow graph is the path point connection which can be passed by available NavMesh; while the blue line is the shortest path which is determined by A* algorithm. In this process, A* shortest pathfinding problem has not been optimized because every walkable polygon point still has weaknesses, for example the polygon with a wide area. There are several solutions of shortest pathfinding which are delivered by A* algorithm, it uses centroid point, edge midpoint, or obstacle point as shown in Figure 2.4.



(a) (b) (c)

Figure 2.4. Several standard pathfinding using A*; (a) pathfinding with connecting polygon centroid, (b) pathfinding with connecting edge midpoint, and (c) pathfinding with connecting obstacle corner [3]

To optimize Figure 2.4 problem, Xiao Cui and Hao Shi [1] made the solution with Triangulation concept. This concept is also used in unity 3D optimization. In triangulation optimization, the polygons are replaced with triangles.



(a) (b) (c)

Figure 2.5. Pathfinding steps (a → c) using tringulation Optimation [3]

The triangulation optimization as shown in Figure 2.5 is made by using the minimum angle of a triangle which must be maximized. In this picture, the optimization is made by using effective tracing with segmentation line. S point symbol is the start and g point symbol is the goal. Pathfinding passes the start point to the goal point. In this case, firstly, s goes through v1 and v2 respectively. Then, on the next step, v1 goes through v3 (v3 is recognized as the shortest path which is close to the goal point). After that, Figure 2.5 (b) shows that tracing line is updated from s to v3 straightly because v3 has already established a point in the first post segmentation, then the path line from v1 to v3 is removed, and v3 must decide the next point that is traced as the start point in the previous segmentation (See figure 2.5 (c)). Finally, it continues until it reaches the goal point. The optimum path does not cross any triangle more than once.

2.2 Navigation System in Unity 3D

Unity is a cross-platform game engine which is developed and founded by Unity Technologies in 2005. Unity has released several versions, one of which is the newest version, namely 5.2.2 which has new feature called NavMesh path library.

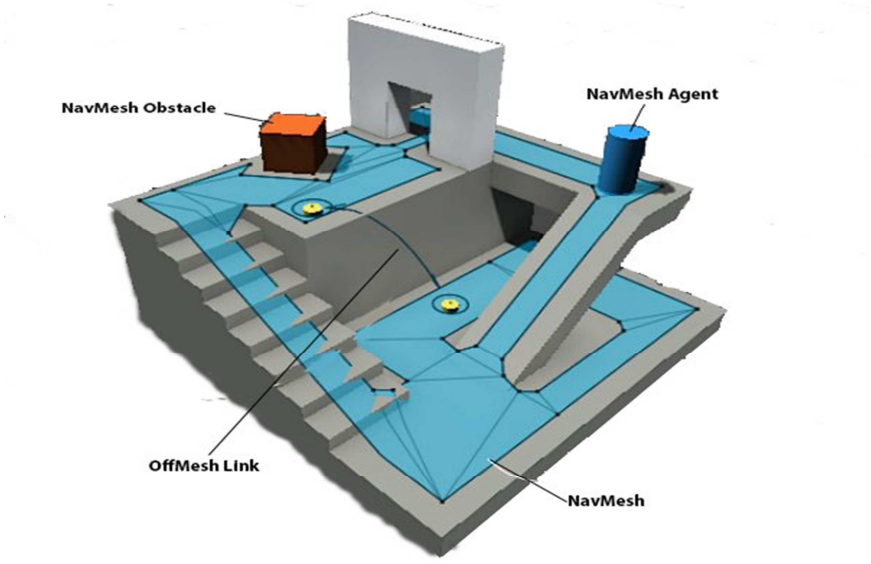


Figure 2.6 Navigation Mesh System Components [4]

As shown in Figure 2.6, there are some components which implement NavMesh system in Unity, such as: NavMesh, NavMesh Obstacle, Navmesh Agent, and OffMeshLink. NavMesh is a polygon structure which describes the walkable surfaces of the game world and allows us to find path from one walkable location to another one in the game world; NavMesh Agent is a component which can move towards the goal on NavMesh surface, so the agent can avoid the obstacle; NavMesh Obstacle is an object which should be

avoided by the agent's movement; and OffMeshLink is a connection point which allows us to incorporate navigation shortcuts which cannot be represented when using a walkable surface.

3. SYSTEM DESIGN

In the system design process, we designed the area of PacMan Game, then determined the obstacle, and created the actors including player, agent/NPC, and bonus actors.

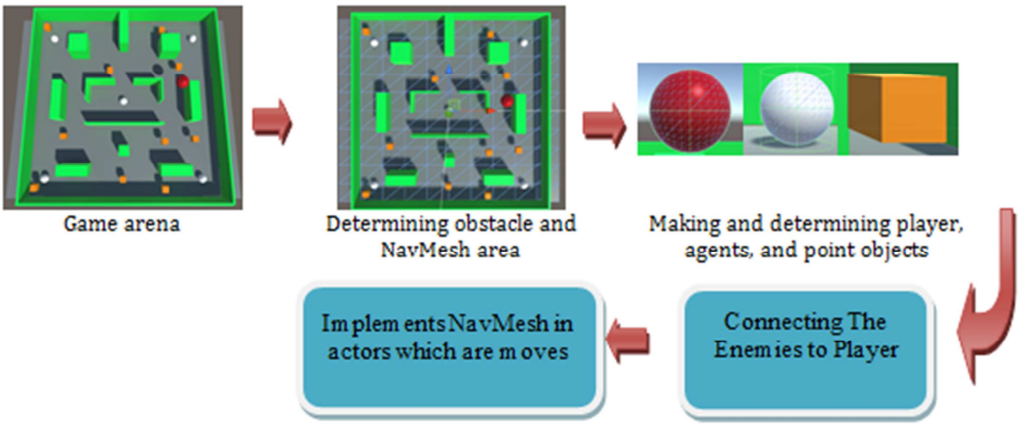


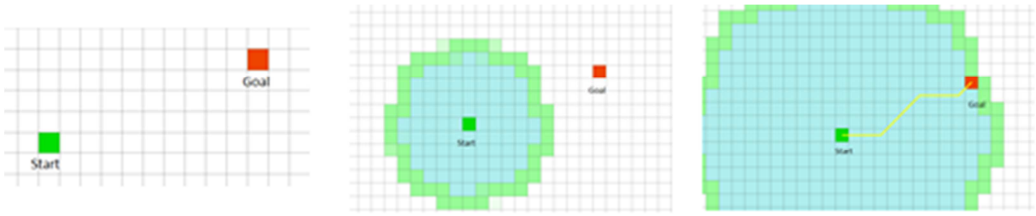
Figure 3.1. Flow diagram of NavMesh implementation on Pac Man Game using Unity 3D

As shown in Figure 3.1, there were five steps to examine the movement of agent on navigation mesh. Game arena was made in unity using a terrain. We created the obstacle by using object as the wall. Then, we determined the path of NavMesh surfaces and the outside area which were unwalkable by the actors, and then, we created and determined the actor (red color) as the pac-man, the white ball as the ghosts (pac-man's enemies), and the orange box as the bonus point. After that, we put the ghosts randomly in the arena and we connected to the actor with NavMesh A* as a pathfinding agent. If one of the ghosts collided with the actor, the health point of the actor would be decreasing. But if the health point became 0 (zero), the game was over.

4. RESULTS

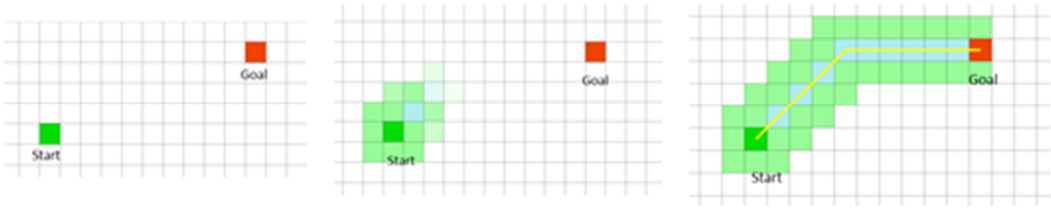
A. Dijkstra and A* concept in shortest pathfinding

To ensure the effectivity of A* implementation, the comparison between Dijkstra and A* algorithm for searching the pathfinding was shown in the Figure 4.1. Figure 4.1 showed how dijkstra algorithm worked to find the goal. In this picture, the searching process was reached with radial search segmentation. The start agent caught the goal by detecting paths arround it until the goal agent entered the inside of the radial search area. Thus, the agent took the path towards the goal. This method was different from A* algorithm as shown in Figure 4.2.



(a) Start (b) Search process (c) Find the goal

Figure 4.1. Dijkstra pathfinding processing



(a) Start (b) Search process (c) Find the goal

Figure 4.2. A* pathfinding processing

Figure 4.2 showed how the start agent searched and caught the goal. The agent just tried in one radial layer around it, then pointed the best direction to the nearest point of the goal until it caught the goal. So, if it was compared to Dijkstra pathfinding concept, A* had more efficient way and its pathfinding's time was shorter than Dijkstra pathfinding.

B. Dijkstra and A* concept in shortest pathfinding with wall obstacle

Here, it was examined how Dijkstra algorithm and A* Algorithm worked in shortest pathfinding problem with the obstacle. Figure 4.3 showed the differences between Dijkstra and A* method in the shortest pathfinding. Figure 4.3 showed how the agent caught the goal by using Dijkstra algorithm, while A* algorithm caught the goal by using wall obstacle. The colors of pattern which existed around the start agent showed how the agent detected the goal. Figure 4.3 (a) showed that the discovery area was larger than what was shown on Figure 4.3 (b) which was very simple and effective.

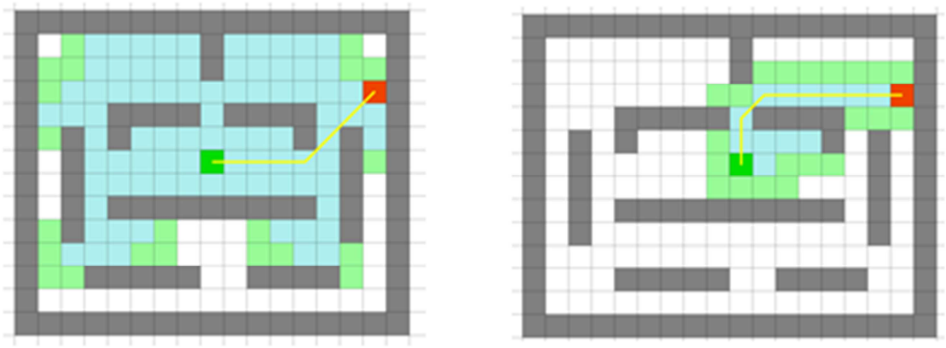


Figure 4.3. Shortes pathfinding concept with obstacle

C. A* implementation in Pacman Game using Unity 3D

By placing several ghost agents randomly, we could detect the behavior resulted by navigation mesh step by using A* algorithm implemented to the character. To know how this method ran effectively, we captured several experiments as follows:

1. *Placing the actor in the center between two obstacles/walls.*
   The result was that the enemy ran towards the player straightly. Detailed result was shown in Figure 4.4.

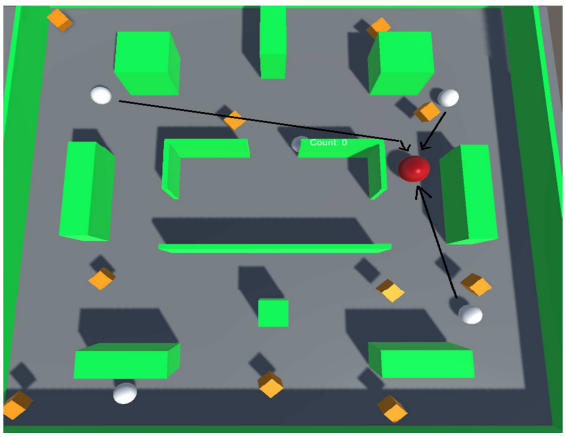


Figure 4.4. The movement of the enemy, it walked straightly towards the Actor (Red ball)

2. *Placing the actor near the border of the arena and letting one of the enemies stayed beside another wall.*
   The result was that the enemy walked to the actor through the obstacle edge, meaning that it took shorter path rather than enemy's way to the actor.

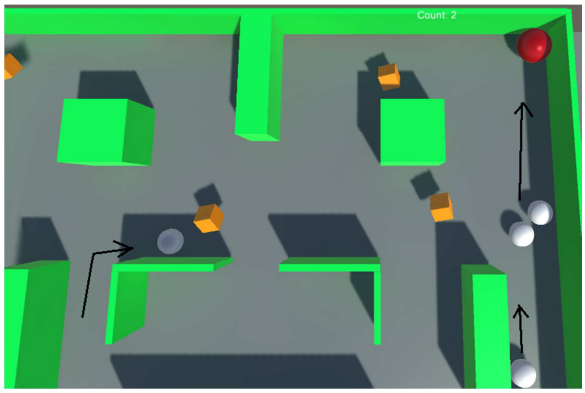


Figure 4.4. The movement of enemy, it walked through the obstacle edge towards the Actor (Red ball)

6. CONCLUSION

It is concluded that Navigation Mesh (NavMesh) and A* algorithm had become the best solution in solving the shortest pathfinding problem for today's industrial need.

The comparison between dijkstra and A* shortest pathfinding problem was different in the concept of pathfinding's area. Dijkstra detected the goal by radial detection mode, while A* used one radial's layer around it, and then pointed the best direction to the nearest point of the goal. It repeated until it caught the goal.

As comparison on Figure 4.1, 4.2, and 4.3 experiment, navigation mesh with A* algorithm was the more effective pathfingding than Djikstra algorithm. The data of comparison was shown in Table 1.

Table 1. NavMesh with A* algoritm and Djikstra Algoritm comparation

Experiment	Name of Algoritm	Steps to catch the goal
Figure 4.1	**A* algoritm**	**43 steps**
Figure 4.2	Djikstra	348 steps
Figure 4.3	**A* algoritm**	**33 steps**
	Djikstra	111 steps

The implementation of navigation mesh (NavMesh) process in 3D polygon surfaces was the key of effective mapping search. By using this method, the agent did not need to explore the unwalkable area. Therefore, updating path with straight line and minimizing the corner point in connecting start towards the goal using A* algorithm had become the best way and the core of effectiveness in A* optimization.

REFERENCES

[1] Mahardiansyah Kartika, **Dijkstra's Algorithm Application on the Pac-Man Game**, *Makalah IF2091 Struktur Diskrit Program Studi Teknik Informatika STEI ITB Bandung*, 2010-2011.

[2] Xiao Cui and Hao Shi, **An Overview of Pathfinding in Navigation Mesh**, *IJCSNS International Journal of Computer Science and Network Security*, 48 VOL.12 No.12, 2012.

[3] Xiao Cui and Hao Shi, **A*-based Pathfinding in Modern Computer Games**, *IJCSNS International Journal of Computer Science and Network Security*, 48 VOL.11 No.12, 2011.

[4] Official Website of 3D Game Engine **http://unity3d.com,** accesed on October 2015.

[5] Shortest Pathfinding, **http://qiao.github.io/PathFinding.js/visual/**, accesed on November 2015.