International Journal of Engineering & Technology, 7 (4) (2018) 3918-3923

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet. v7i4.15084
Research paper

Implementation of a* algorithm within navigation mesh in an artificial intelligence based video games

Temmy Subrando [1] *, Fauzan Akbar Prasetyatama [1], Devi Fitrianah [2]

[1] Student, Department of Computer Science, Mercu Buana University
[2] Associate Professor, Department of Computer Science, Mercu Buana University
**Corresponding author E-mail: 41514010137@student.mercubuana.ac.id*

Abstract

Pathfinding is one of the key components for Artificial Intelligence in video games, It addresses the problem to find the shortest path between starting point and destination point, before A* became a staple algorithm for pathfindings, many search algorithms were used including Djikstra, Depth, and Breadth-First searches, Pathfindings are known to be resource-intensive processes especially in a large world but it mostly depends on what algorithm the game is using and what method it is implemented, using A* combined with Navigation Mesh, a popular abstract data structure in video games, this paper reviews the overall system resources used during the pathfinding pro-cess with a destination point that is always moving in a real-time so that it will gives realistic results since, in video games, a Player is constantly moves around the world, the result that we have gathered, we have concluded that A* flawlessly works with Navigation Mesh, and the performance impacts of both A* and Navigation Mesh combined is non-existent.

Keywords: *A* Algorithm; Artificial Intelligence; Navigation Mesh; Pathfinding; Video Games.*

1. Introduction

Pathfinding has been the most discussed problem not only in a Modern Games but also in Robotics, Geographic Information System and Statistics [1] to find the nearest path from the starting point to the destination point [2], a lot of search algorithms were utilized such as Djikstra's or Depth-First Search long before A* came, Today, A* Algorithm is the most widely used algorithm for pathfinding [3], although A* can works just fine using Grids-Search, it would suffer performances hit due to its high memory usage especially in a larger world, in today's Video Games, Navigation Mesh is a popular Data Structure for Pathfinding Algorithm with all its advantages over grids such as using much fewer system resources in 3D and larger worlds [4].
In Modern Games, Pathfinding is implemented into Artificial Intelligence that are mostly in a form of Non-Playable Character or NPC because it depends on pathfinding to help them navigate [5], Most Game's AIs navigates the game world with the intents to search for a Player's location, chasing, disrupting and challenging them, To be able to do this Navigation Mesh is implemented into the game as a Data Structure for A* to use instead of grids. In the game with a maze-like map in which pathfinding really shines, Pathfinding capability is needed to be implemented to AI for it to be able to pathfinds itself to the Player.
For that sole reason, A* algorithm is chosen to be implemented into the games that we were developing. The main purpose of AI in our games is to pursue the Player, and thus, if AI detects Player, the Algorithm will calculate the nodes, if any of the nodes leading to the player with the least cost is detected, a path is then generated and then AI will utilize that path. In this paper, The game is developed exclusively to analyze how A* pathfinding works in an Unreal Engine 4's Navigation Mesh, and to prove whether its performances impact to the system is acceptable by using a scenario where multiple AIs will run toward the Player.

2. Related study

2.1. A* algorithm

While in some AI games, the pathfinding component is only a tiny fraction of problems to support the game development, the A* algorithm main problem itself is, how could it help the AI to solve a more difficult problems? in their research, Barnouti et al tested the algorithm using images that represent strategy game's maps and mazes, the images are converted into three main colors in grid, so that the colored grids represent which colors is the actual path, and which one is the obstacles and etc, then the starting and destination points were decided, after repeated tests of 100 different images, more than 85% of the images used in the test was able to find the shortest path between the points as shown in Fig. 1, the performance impacts of pathfinding using A* was also pretty negligible [6].

2.2. Navigation mesh

Navigation Mesh is a set of 2D convex polygon meshes that enables the agent to traverse the area within the game, in other words, the agent in the game can freely walk in this area without getting obstructed by any barriers in the environment as long as there is navigation mesh in the area, before Navigation Mesh, Grid-Search space was widely used for pathfinding, in grid-search, the pathfinding is a resource-intensive process especially in large worlds and grid-search unable to represent a part of the world that cannot be aligned by grids, this became a liability because 3D World isn't just rectangular-shaped, and then Navigation Mesh came along in mid 1980s in a robotics field and rose in popularity when it was first

Copyright © 2018 Temmy Subrando et. al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

implemented for video game AI in 2000, up to this day Navigation Mesh is still more popular and widely used for pathfinding especially in large 3D World, because the world mostly constructed using polygon structure [1] and Navigation Mesh able to represent 3D world more accurately with fewer polygons than grids, while also able to expand to non grid-aligned areas in which grids cannot [4], In Unreal Engine 4, Navigation Mesh has its own tools, the tools analyze the geometry level and its surrounding area based on the user's choice and generate the mesh accordingly [7].

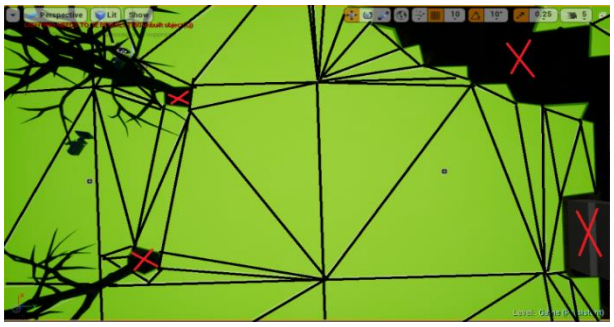


Fig. 1: Triangulation Navigation Mesh in Unreal Engine 4.18.3 in Top-Down Perspective.

According to Xiao Cui and Hao Shi, Triangulation Navigation Mesh as depicted in Fig. 1, is a Navigation Mesh in which polygons are replaced with triangles, by maximizing the minimum angle of a triangle, this type of Navigation Mesh is guaranteed to give an optimal path that will not cross any triangle more than once [2] the figure shows Triangulation Navigation Mesh system in Unreal Engine 4, the "X" marks represent the Obstacle around Navigation Mesh, among which are objects, trees or steep terrains in which the Navigation Meshes Agent cannot traverse through, the Straight Lines represent the Triangulation in Navigation Mesh and enable A* Algorithm to generates paths to the nearest point using the edge of triangles as its nodes then trace it in a straight line to the path leading to the destination point, after that the new parent node repeat the same process all over until it reaches a target point [2].

2.3. Artificial intelligence

Artificial Intelligence in video games covers the behaviors, decision-making processes of an NPC, navigating the game world, reacting to Player's decisions, and defeating Player, these elements of Artificial Intelligence can be separated into two different types, Navigating the game world and Defeating an opponent can be considered as Game AI, while AI that makes decisions and reacts to Player's decisions is a Context AI [8], in Dota 2, the first task of its main AI called The Creeps are navigating in their corresponding lanes until The Creeps meet the opposing faction's units in their way, in that case, they will completely stop what they are doing to attack and even try to chase the enemy units far from the their intended lane only to stop until the said units die or vanish into the fog of war, this is the scenario of where the Game AI is implemented and more details can be seen in Fig. 2.



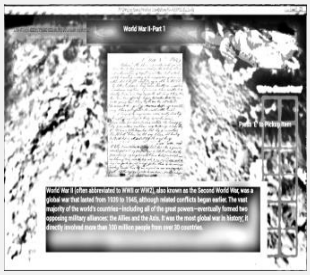
Fig. 2: The Creeps Navigating in Lanes and Attack Each Other [9].

Example of Context AI can be seen in a game called The Elder Scrolls V: Skyrim, in one of the quest where the Player character is asked to mediate the two conflicting factions and gives decision to which area or land they can control, if the Player's decisions are too one-sided on one faction, the losing faction's member will be upset and dialogues to a Player character when they have a conversation later in the game may not be so friendly and will shows some level of disappointments of Player's decisions depending on how unjust the Player's decisions are, on the other hand, the winning faction's members will address the Player with some gratitude and may even give the Player a gift [10].

3. Material collections

3.1. Storyboard

Table 1: Game's Storyboard

Game's Title : The Private Investigator's Chronicles	
Scene 1	Scene 2
	
Description Title : Main Menu Action : Click "Play Game" to start the game, "Options" to adjust resolution and "Quit" to close the game	Description Title : Exploring The Woods to Find a Hidden Basement Action : Use "W,A,S,D" and "Spacebar" to control the character, escape from enemy by hiding behind objects or run through the woods.
Scene 3	Scene 4
	
Description Title : Finding Letters in a Hidden Basement Action : Press 'E' to examine the Letter and press 'E' again to add the Letter to inventory	Description Title : Game's Ending Action : After getting out of Hidden Basement and then going back to his car

3.2. Materials

In video game that we developed where the game is used as a platform to implements A* algorithm within the Navigation Meshes, some materials were required to support the development, these materials ranged from a set of model, audio, texture, image, and animations and came from a license-free or editorial-only sources, for more details, we categorized all materials used in the game into their types below.

i) Model

For models, it can be characters, foliages, rocks, doors, cars, barrels, fences, weapons and other static mesh models, the models used came from Mixamo's Character Pack [11] Assets, Yaroslav's Buick gsx 455 [12], Soul Cave Assets [13], Unreal Engine 4's Standard Assets, Container Packs, Anafeyka's Knife and Blade [14], Open World Demo Collection [15], for our Player character we used a standard Unreal Engine 4's Mannequin Model, Mixamo's Vampire is used for AI's character and Adam as Main Menu model [11], and AI's weapon came from Anafeyka's [14], gameworld3d Container Pack's barrels, crates, fences [16] and michael-mihalyfi's door are used for environmental objects, as well as rocks, trees, and bushes from Soul Cave. the models used are in FBX and OBJ extension.

ii) Audio

Less than a dozen audio files were added in the game, audio files were used in the map and characters, to gives the Player the feels of exploring the forest, we added ambiance sounds of bird chirpings, crickets, winds as background audios that always play wherever the Player is at and a waterflows sound effect that plays only when Player is in the close proximity of the river, we also added a footstep and stomp sound effects to Player's character movements, these audios are in .wav format and came from Unreal Engine 4's Standard Assets and Soul Cave Assets [13].

iii) Image

Image is a type of material consisting of graphic files that are used in almost all elements of the game, these image formats that we used in our game are in TGA and JPG extension, while TGA is used as a textures to paint the terrains, character models, and any other meshes, the JPG is used on Main Menu Screen, in a buttons, game's title text, Key and the Letter item itself, as can be seen in Fig. 6, TGA texture file came from Unreal Engine 4's Standard Assets and Soul Cave's [13] , The JPG files are Main Menu buttons which are custom made and a maisonboheme's letter.

iv) Animation

Similar to models, animation files are of FBX extension, There are a few animations used in this game, from walking, running, jumping, attacking, and dying animations, these animations came from Unreal Engine 4's Standard Assets, Animation Starter Pack [17], and from Mixamo's Character Pack [11], all the animations that came from Mixamo's can only be used on the models that also came with it as they both share the same skeletal meshes, and vice versa, the Unreal Engine 4's and Animation Starter Pack's animations are created exclusively to be used for Mannequin's model.

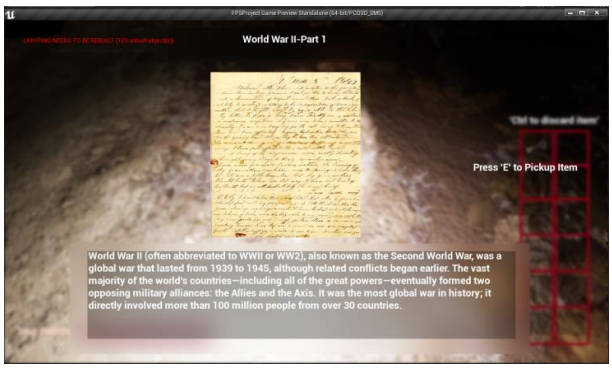


Fig. 3: Example of Material Used in the Game.

4. Methodology

4.1. Game development method

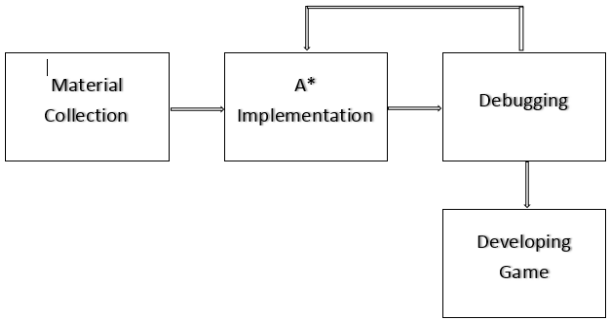


Fig. 4: Block Diagram of Research Method.

i) Material Collections

The Materials used in this work is taken from many sources, including Unreal Engine 4 Marketplace and CGTrader, most of the materials used are either free-license or for editorial purposes only, other than materials collecting, during this phase, we also created a concept, designed the map, created a storyboard for the game and so on.

ii) A* Implementation

Implementing the A* to the AI isn't just about using set of scripting tools but we also used certain components to help the AI agent find its way to the Player or roaming around the map, first is by using Navigation Mesh and baked it to cover the entire map, adding collision to the AI agent, putting massive invisible walls around the uneven terrains to give a maze-like feels so that the algorithm doesn't simply pathfinds in a straight line to the destination point, and the second is by utilizing Unreal Engine 4's AI Perception component to help the AI agent detects the hostile (in this case, the Player) and makes decisions based on it, we are expecting to create AI that able to pathfinds around the map to reach the destination (either its supposed roaming locations or Player's locations), able to detect Player, losing the sight of Player, running to the last known location of a Player and so on.

iii) Debugging

After implemented the A* pathfinding scripts to the AI agent or after doing minor changes to either scripts or the agent, we always tested the AI to see whether it could find its way to the Player or roaming around the map as expected and to ensure that it always chose the shortest possible route, if we discovered that the AI isn't behaving the way it should be such as getting stuck in the obstacle instead of avoiding it, the process went back and loops between A* Implementation phase and Debugging phase until the AI is working the way we see fit.

iv) Developing Game

After the entire processes are finished, in this final phase, we focused on finishing the game by putting key mechanics, such as implementing combat system, puzzle system, designing and expanding the map and adding main menu screen including graphic options, painting the terrain, adding foliages, texturing all the meshes, adding sound effects by utilizing some if not all materials we've been collecting into game.

4.2. A* (a star) algorithm

A* is an algorithm used to find the shortest path, it is an improvement from best-first search algorithm that modified its heuristic function to gives the best result by combining the heuristic function [h(n)] and distance cost [g(n)], for A* to find the shortest path, it needs a set of lists, Open and Closed, Open List contains nodes that still have potential to be the shortest path and their heuristics have been calculated, while Closed List is a list that contain the explored nodes that had been selected as the shortest nodes to the path and already closed for further explorations.

A* works by searching the neighboring nodes and calculating the node's [f(n)] scores, starting from the starting node, A* will look for any other nodes adjacent to its starting point and adding them to the Open List including the starting node, if one of the found nodes have the lowest [f(n)] score, the said node would become the parent nodes for A* to continue its search and the previous parent nodes

(in this case, the starting node) will be added to Closed List, and the processes repeat until A* find its way to the destination node, A* uses the following formula to decides the shortest path: f(n) = g(n) + h(n).
Using the formula, A* will then decides which node would be the shortest path to the destinations by using variable of [f(n)], if the [f(n)] score of the node is the lowest than the alternative nodes, A* will continue its search from node with the least [f(n)] cost. For a visual representation sake, [g(n)] is treated as fuel used to reach from one path to another, where the [h(n)] is an estimation of distances that still have to be taken to reach the destinations and [f(n)] is a fuel cost per distance traveled, by incrementing these two scores together the [f(n)] score is then obtained.

4.3. Artificial intelligence

Behaviors of AI Agent and its Pathfinding processes in the game is presented with the flowchart below.

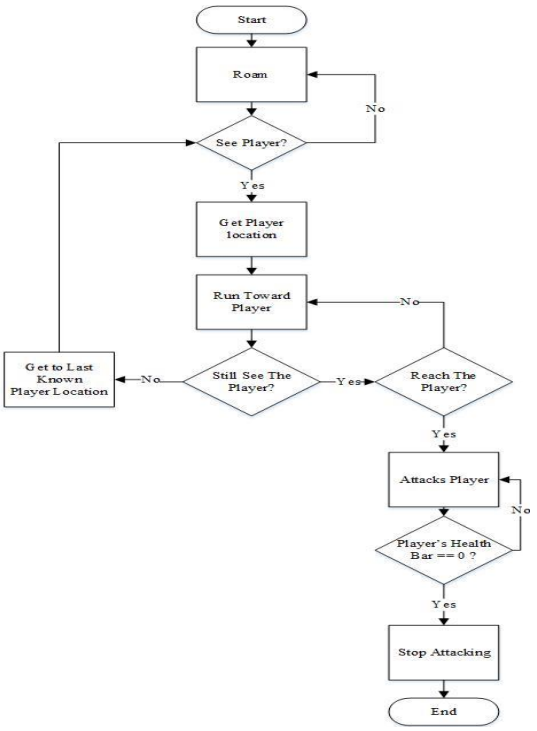


Fig. 5: Ai's Flowchart.

4.3.1. Roam

A* will choose random node in a map as a destination point, finding the shortest route to the said point by calculating the surrounding nodes where the Agent currently standing, after shortest path has been found, Agent will walk toward the destination, until It sees Player in which the process will go straight to "See Player?" otherwise continue roaming and repeating the process by finding a new node as destination point after Agent reaches its current destination.

4.3.2. See player

Agents will only detect target inside the radius of its 45° cone-shaped vision. if they don't see Player the process will loop back to "Roam", otherwise the process continues to "Get Player Location".

4.3.3. Get player location

If The Player get caught inside of Agents' visions, Player's current location will be recorded in a real time and will be set as a destination point and shortest route will be determined by calculating every node that leads to the Player's current whereabout by using the formula shown in A* (A Star) Algorithm section.

4.3.4. Run toward player

After nodes have been calculated and the shortest route has been determined, Agents will then run toward destination point which is Player's current location.

4.3.5. Still see the player

If Agents still hold the sight of Player, They will continue running toward Player until it "Reaches the Player", otherwise the process goes to "Get to Last Known Location".

4.3.6. Get to last known player location

If Agents lose sights of Player, Player's Last Known Location is then saved and become their current destination points to which they will run toward to, whether or not the Player still lingers in the area, the process will always loop back to "See Player?" decision.

4.3.7. Reach the player

Agents will continue running toward Player until the Agent is able to outrun the Player, If they do, they will go straight to "Attacks Player".

4.3.8. Attacks player

Agents will initiate combats and attack Player until Player's current health equals to zero.

4.3.9. Player's health bar == 0

If Player's health bar equals to zero, the Player will die and the Agent will "Stop Attacking", otherwise continue "Attacks Player" until the process is fulfilled.

4.3.10. Stop attacking

Agents will stop attacking if they successfully kill the Player and game over screen is then shown.

5. Result and discussion

5.1. Hardware and software

The Experiment in this paper is done using a computer with AMD FX-6300 CPU 3,5Ghz, with 8GB RAM and NVIDIA GTX 970, The game engine UE4 4.18.3 is utilized to design the layout of the game, scripting, implementing the AI, and also to conduct analysis using the built-in system resource monitoring and debugging tools.

5.2. Result

The concept of A* algorithm is to find the shortest path to the destination point as efficiently as possible. To know how it works, we implement the A * algorithm into AI Agent in the game. We use a testing ground map built exclusively for analyzing in a more detail of how the A * algorithm works. Testing ground is built in the same level as the original map in the game but on a different scale. In this game, we marked the Player in a blue circle and 3 Agents in red, where each Agents is placed in different locations in this maze-like map, as can be seen in Fig. 7.

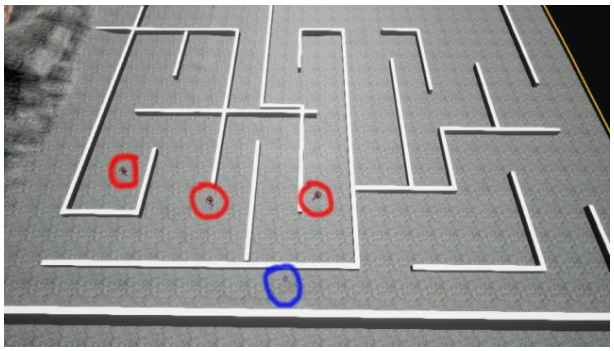


Fig. 6: Agents (Red Circle) and Player (Blue Circle) Locations.

Afterward. there are a few steps taken to see how the A * algorithm works:

1) When the game starts, Agent will roam if It has yet to see the Player. depicted in Fig. 7 is a scenario where Agent has seen the Player, the algorithm will generate the shortest path on a Navigation Mesh that has been calculated leading to the Player and will update the path in a real-time if somehow the Player moves from their current location.

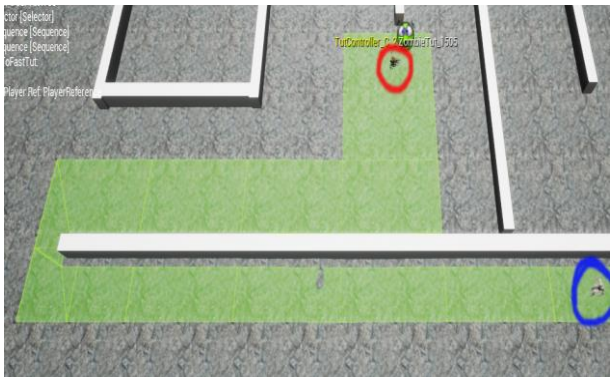


Fig. 7: The Generated Path That Appears in the Game.

2) In Fig.7, it shows a path generated from Navigation Mesh that will lead the Agent to the Player, Agent will run through a more efficient and shortest path as shown in Fig.8, In Fig. 8 in the first figure, we have seen that the Agent chose to run diagonally and to stay closer to the walls or obstacles in the intersections whereas in the next figure, A* generated new paths in a real-time because Player moved slightly from their previous location and an Agent then went straight to the Player by running diagonally, had not Player moved from its location an Agent would have run in a straight line, all with the same reason, that is because it is the shortest path to choose.

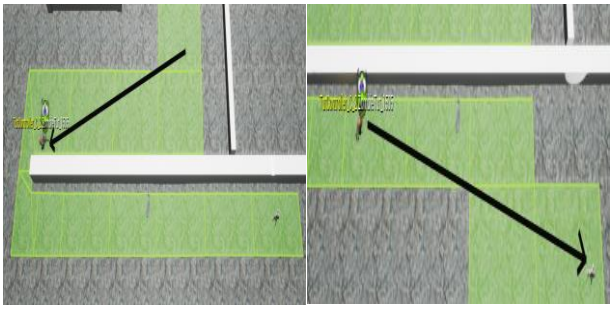


Fig. 8: Agent Finding Its Way to the Player.

3) To prove the accuracy of A* algorithm in our AI, depicted in Fig. 9, we moved the Player spawn location and the Agent is still moving close to the walls as it can be seen from the active Navigation Mesh A* it utilizes and only run diagonally after to the destination point after each intersection.

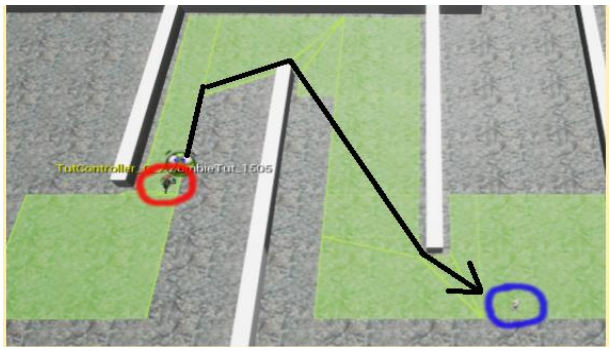


Fig. 9: Player That Has Been Moved.

During this test, we also looking at Frame Rates of the game and its Memory Usages, without activating the Agent's pathfinding capability, in this Idle state the Frame Rates is averaged at 55 fps and uses 2200MB of RAM this includes the UE4's processes, we then enable one of the Agent's pathfinding capability, and the result is still the same, no changes in both Frame Rates and Memory Usage. Afterward, all the 3 Agents' pathfinding were enabled at the same time, and we also took control the Player so that it's always moving with the purpose for the algorithm to always generates new paths to the Player, but as it turns out even with all Agents pathfinding its way to the Player, after around 10 tests, both the Frame Rates and Memory Usage only changes slightly and are insignificant, for the Frame Rates, it averaged at around 54 fps while the lowest of all tests was 52 fps, and 2357MB is the peak memory usage.

6. Conclusion

Pathfinding is a process that determines the movement of an objects from one place to another by choosing the shortest available path to navigate to the destination point without colliding with any obstacles in the way. In this paper, A* is implemented along with Navigation Mesh to find the shortest route from AI's spawn point to its determined roaming point or Player's current position by calculating f(n) score of the node generated by Navmesh's polygons. We built a testing ground map solely for the purpose of this test, the map imitates mazes with walls and objects to act as obstacles for the agent. First, we test the algorithm by determining where the spawn point for AIs and Player are with the pathfinding capabilities of all the three agents enabled at the same time, as soon as Agents detected the Player, A* will calculate every f(n) score of the available alternative routes to the Player, afterward the shortest route has been founded Navmesh will generates the path for Agent to navigate on. in Fig.8 (a) and (b), we can see that the navigating behaviors of the agent is that it always chose to run diagonally to its destination because in some cases, choosing the node diagonally will always yield the result of the least overall f(n) score of 2 nodes combined as opposed to choosing to navigate in each node vertically or horizontally which is how A* works by only choosing the shortest route with the least f(n) score. In the scenario where the position of a Player had been moved as can be seen in Fig. 9, the way it navigates to the player is, the AI still chose to run diagonally with the only exception that it will only run vertically at the start this is because the path generated by Navmesh in this scenario wasn't possible for the Agent to run diagonally. The algorithm also checks for the shortest path in a real-time which means the pathfinding process of A* and path generation of Navigation Mesh will be perpetually running based on the player's current whereabouts.

Acknowledgment

This paper is possible with the help from Allah Subhannahu wat'ala, His permission is what allow us to complete this paper, and with the help from my Associated Professor and Supervisor, Dr. Devi Fitrianah, MTI2 who also financially support this publications, my Academic Supervisor, Anis Cherid, SE, MTI, my co-author Fauzan

Akbar Prasetyatama1, Joel George for his great Unreal Engine 4's tutorials also friends and families who always gave their best advices and much needed supports for me.

References

[1] M. Zikky, Review of A * (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game, Emit. Int. J. Eng. Technol., vol. 4, no. 1, pp. 141–149, 2016.
[2] X. Cui and H. Shi, An Overview of Pathfinding in Navigation Mesh, IJCSNS International Journal of Computer Science and Network Security, IJCSNS Int. J. Comput. Sci. Netw. Secure. vol. 12, no. 48, 2012.
[3] X. Cui and H. Shi, A*-based Pathfinding in Modern Computer Games, Int. J. Comput. Sci. Netw. Secur. vol. 11, no. 1, pp. 125–130, 2011.
[4] S. Rabin et al., Game AI Pro: Collected Wisdom of Game AI Professionals. 2013.
[5] R. Coleman, Fractal analysis of stealthy pathfinding aesthetics, Int. J. Comput. Games Technol., no. 1, 2009.
[6] N. H. Barnouti and S. S. M. Al-Dabbagh, Pathfinding in Strategy Games and Maze Solving Using A Search Algorithm, J. Comput., pp. 15–25, 2016.
[7] P. L. Newton and J. Feng, Unreal Engine 4 AI Programming Essentials. 2016.
[8] F. Safadi, Artificial Intelligence in Video Games: Towards a Unified Framework, 2015.
[9] Valve, Dota 2. Taken September 27, 2017.
[10] Bethesda Softworks, The Elder Scrolls V: Skyrim, http://elderscrolls.wikia.com/wiki/Season_Unending. Retrieved May 2, 2018.
[11] Mixamo, Mixamo Animation Pack by Mixamo in Characters - UE4 Marketplace. https://www.unrealengine.com/marketplace/mixamo-animation-pack. [Accessed: 23-Jul-2018].
[12] Yaroslav, 3D Buick gsx 455 | CGTrader. https://www.cgtrader.com/free-3d-models/car/sport/buick-gsx-455. [Accessed: 23-Jul-2018].
[13] E. Games, Soul: Cave by Epic Games in Epic Showcase,Environments - UE4 Marketplace. https://www.unrealengine.com/marketplace/soul-cave. [Accessed: 23-Jul-2018].
[14] Anafeyka, [FREE] Knife and Blade for community - Unreal Engine Forums. https://forums.unrealengine.com/community/community-content-tools-and-tutorials/39180-free-knife-and-blade-for-community. [Accessed: 23-Jul-2018].
[15] E. Games, Open World Demo Collection by Epic Games in Epic Showcase, Environments - UE4 Marketplace. https://www.unrealengine.com/marketplace/open-world-demo-collection. [Accessed: 23-Jul-2018].
[16] Gameworld3d, Container Pack with Fence 3D asset | CGTrader. https://www.cgtrader.com/free-3d-models/exterior/industrial/container-pack-with-fence. [Accessed: 23-Jul-2018].
[17] E. Games, Animation Starter Pack by Epic Games in Epic Showcase,Animations - UE4 Marketplace. https://www.unrealengine.com/marketplace/animation-starter-pack. [Accessed: 23-Jul-2018].
[18] A. T. Wibowo and D. Fitrianah, A K-Nearest Algorithm Based Application To Predict Snmptn Acceptance for High School, Int. Res. J. Comput. Sci., vol. 5, no. 01, pp. 9–20, 2018.
[19] E. Yunianto, S. Rudiarto, and D. Fitrianah, Implementation Of Hamming Network Algorithm To Decipher The Characters Of Pigpen Code In Scouting, Int. Res. J. Comput. Sci., vol. 5, no. 01, 2018.
[20] D. Ramayanti, The 3D Game Simulation of 'Anjungan DKI Jakarta' at Taman Mini Indonesia Indah and Betawi Culture based on FPS (First Person Shooter, Int. Res. J. Comput. Sci., vol. 2, no. 12, 2015.