Comparison Between A* And Obstacle Tracing Pathfinding In Gridless Isometric Game

Lailatul Husniah
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
husniah@umm.ac.id

Rizky Mahendra
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
rizky.ade.mahendra@gmail.com

Ali Sofyan Kholimi
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
kholimi@umm.ac.id

Eko Cahyono
Informatics Department
Universitas Muhammadiyah Malang
Malang, Indonesia
ekobudi@umm.ac.id

***Abstract*— The pathfinding algorithms have commonly used in video games. City 2.5 is an isometric grid-less game which already implements pathfinding algorithms. However, current pathfinding algorithm unable to produce optimal route when it comes to custom shape or concave collider. This research uses A* and a method to choose the start and end node to produce an optimal route. The virtual grid node is generated to make A* works on the grid-less environment. The test results show that A* be able to produce the shortest route in concave or custom obstacles scenarios, but not on the obstacle-less scenarios and tight gap obstacles scenarios.**

Keywords—game, pathfinding, A*, grid-less, and isometric

I. INTRODUCTION

Video games genres and styles have evolved in the past decades. The current technologies make it easier for a developer to implement or combine one genre to another. One of the genres that we studied is 2D simulation games with isometric perspective. In the past time, most the 2D isometric games rely on the grid as its base. Almost every game-objects shown on that game has bounded to that grid. Grid usage also makes it easier to implement pathfinding and mapping. Common A* algorithm could have implemented very well.

Unity3D is a game engine to make a high-quality 3D and 2D games [1]. Although it could produce a 2D game, all game objects occupy a 3D space. Thus a default 2D game would not become a grid based game. However, it still possible to develop a grid-based 2D game by generating the grid itself. Based on such occasion, to solve pathfinding problems, we can either generate a virtual 2D grid to implement basic A* pathfinding or create a custom pathfinding method to adapt grid-less environment.

"City 2.5" is a grid-less 2D isometric city-building simulation game. Since we are part of the developer who develops this game, we can access and modify every part of this game. City 2.5 use a custom pathfinding method called Obstacle Tracing (OT) [2]. However, this method is not optimal since it cannot always guarantee the shortest path. The tendency of NPC (Non-Player Character) to encircling the obstacle could make the path even longer. NPC is also relying on obstacle collider convex shape which is means when NPC found a concave collider the path taken would be longer.

This previous study [3] use Fuzzy Mamdani Logic to solve navigation problems that made better navigation on the robot hexapod fire extinguisher. Other studies [4] use A* to solve pathfinding problems in the 2.5D isometric game. The game is grid-based, which preferable for A* algorithm. The primary goal of that study is to investigate and determine the optimal pathfinding strategy based on several measures such as steps and time have taken to reach the goal using A* algorithm [5].

A* itself is favorable as a pathfinding method as it is simple to implement, is very efficient, and has lots of scope for optimization [6,7]. A* able to generate the shortest path because it uses a heuristic function to estimate the distance of any point to target point [8]. A* implementation on Unity3D has been done by [9], and so we need to adapt the implementation method for this game.

A* is an algorithm which measures the heuristic distance between a given point, while the pathfinding itself rely on the search space on how the A* graph represented in the game. In this study [10,11] there are several ways to represent the search space. Previous research [12] use A* to solve pathfinding method for a grid-based graph. In this research, we will use the common A* search space representation, a rectangular grid. The advantage of using the grid is it is easier to generate automatically [13] and easy to implement map representation [14].

Some studies about comparing pathfinding methods have been done by [11,12,13,14,15]. Unfortunately, none of those paper discusses pathfinding on grid-less environment explicitly. What makes this paper different from previous studies is the case study, grid-less environment, and the method Obstacle Tracing, which is rarely used in other paper and research The goal of this paper is to solve NPC pathfinding problems on a grid-less 2D isometric game. Therefore, A* is implemented and compare it with current Obstacle Tracing method. This paper also focuses on how to implement A* properly without altering the unique rules and game design itself. Thus we will limit not to include the computational complexity of the two algorithms, but compare how the path has taken and the speed of computation.

978-1-5386-8402-3/18/$31.00 ©2018 IEEE

II. RESEARCH METHODOLOGY

A. Study on Current Method

City 2.5 is a city building simulation game developed using the Unity3D engine. It is relying heavily on Unity3D component for everything to works, including its pathfinding method which is Obstacle Tracing. After some studies on this game, we can state the flow of its pathfinding method on Fig. 1 and the predicted result on Fig. 2. At the flowchart in Fig. 1 we can see this method using Raycast [16] to detect the obstacle which blocking its way and move toward it. If the obstacle is not the destination, then it needs to be encircled. Although this method able to find the route, however, if the obstacles have a concave or custom shape, the path taken might be not optimal (not shortest).

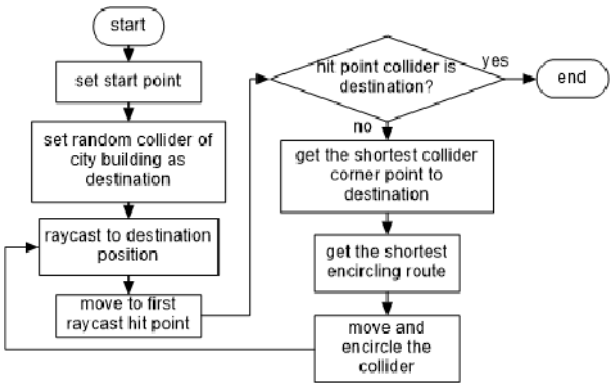

Fig. 1. Obstacle Tracing method flowchart

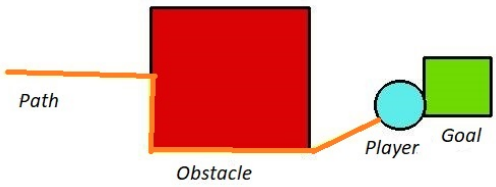

Fig. 2. Pathfinding result using Obstacle Tracing

B. Design for A* method

This game requires a dynamic map which able to change when the world itself is changing. To achieve that, a method to generate and validate A* map has created as seen in Fig. 3. There are three parameters to generate a grid map which is size X and size Y as how much nodes generated on X and Y axis and space between which determine how far the gap between nodes.

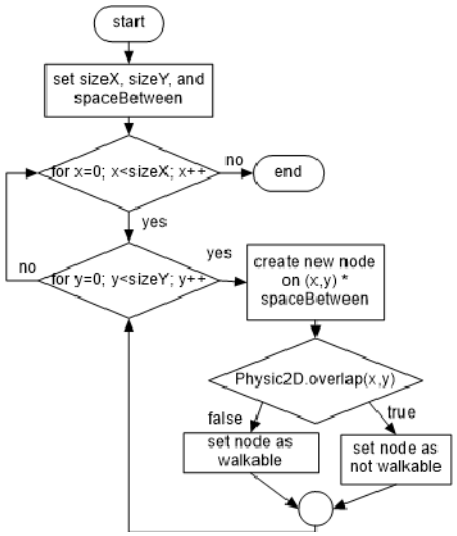

Fig. 3. Map generating flowchart

To validate if the node is either walkable or not, it can achieve by calling Unity method Physic2D.OverlapPoint [17]. This method will return true if a point in the game world is inside a 2D collider. Since the obstacles in this game are using 2D collider, so it is possible to determine if the node was walkable when Physic2D.OverlapPoint return false, and vice versa.

The conventional A* path-computing could be applied after the map has generated as shown in Fig. 4. The next problems that need to be solved are how to determine the start and end point for each NPCs. In this game, NPC does not have any collider, which means that it could occupy any node as long as it is walkable. So we could do reverse computing to determine the closest node as the start point as shown in Fig. 5.

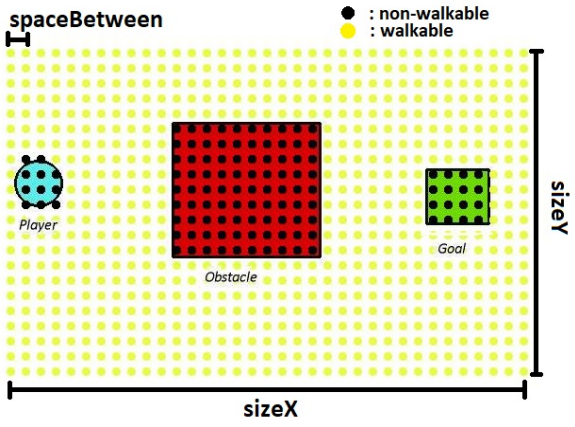

Fig. 4. Map generation result

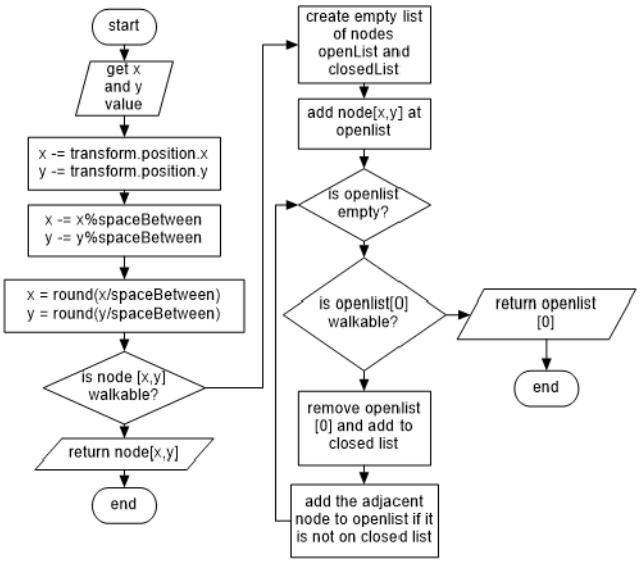

Fig. 5. Find nearest node flowchart

Method on Fig.5 could be used to determine endpoint, but another problem would have occurred like in Fig. 6. The node returned would be always same regardless the NPC start position which would cause less-optimal shortest path.

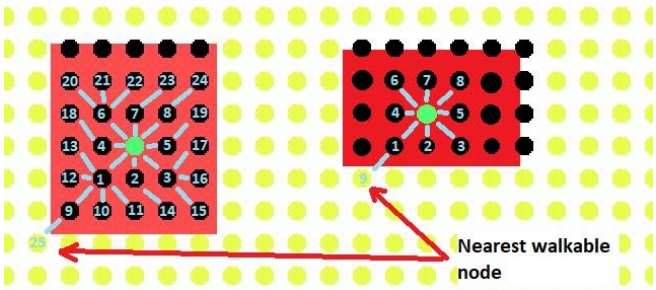

Fig. 6. Current search algorithm always returns the left-bottom side node

To prevent that behavior, it necessary to know which node that closest with destination and start point. It could be achieved by do a Raycast from current position to destination and search the closest node from the hit point, resulting in a method on Fig. 7.

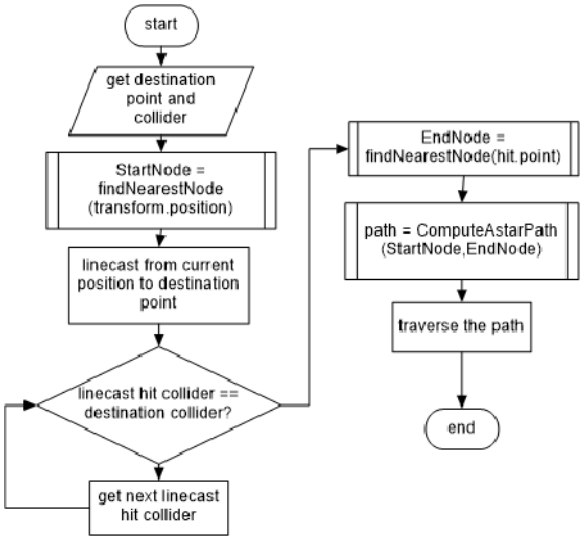

Fig. 7. Determining start and end node

III. RESULT AND ANALYSIS

The goal of this paper is to compare both methods. A* pathfinding comparison has been made by [15] using node count and travel length as a parameter. There are ten different scenarios to test. Both NPC with different algorithm placed in the same spot and need to find the path to destination target. A * map made using the SpaceBetween parameter with a value range of 0.1, 0.2, and 0.3.

A* map made on three space between parameters value which is 0.1, 0.2, and 0.3. The test parameters are:

Node: Node count need to reach the goal

Actual: Actual distance from start to goal (obstacle ignored)

Travel: Travel distance from start to goal

Error: Difference from target goal point, to an actual goal point

Difference: Algorithm precision score. Where Difference = Actual (Travel + Error). The bigger Difference score is considered better NPC

A. *Scenario 1: No Obstacles*

Table I has shown that OT is better at a Difference score, because of OT NPC able to walk straight to the goal as shown on Fig. 8. While A* has shorter travel distance, yet the error is too big because A* movement is dependent on nodes.

Fig. 8. NPC go straight to the destination

TABLE I. SCENARIO 1 RESULT

Obs. = 0	No Obstacle				
	Node	*Actual*	*Travel*	*Error*	*Difference*
OT	2	2.747102	2.386049	0.361054	**-1.1E-06**
A* 0.1	21	2.747102	2.487411	0.440114	-0.180423
A* 0.2	10	2.747102	**2.340456**	0.572451	-0.165805
A* 0.3	7	2.747102	2.381878	0.534509	-0.169285

B. *Scenario 2: Natural placement*

In Fig. 9 obstacles placed naturally as the game behave. Table II shows that A* able to perform better than OT in

term of the node and travel distance on 0.3 set up. However, the error still significant compared to OT.

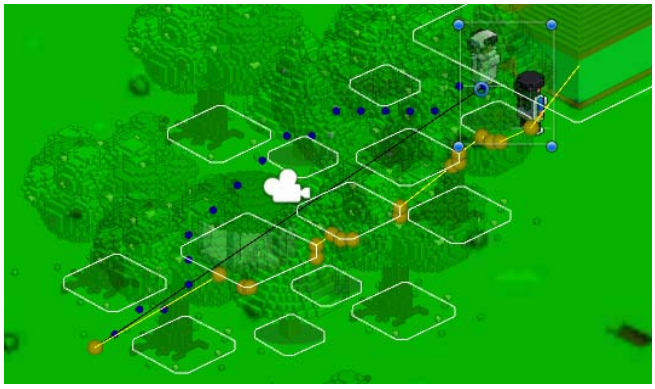

Fig. 9. Scenario 2: both NPCs took a different path

TABLE II. SCENARIO 2 RESULT

Obs. = 13	Normal Placement				
	Node	*Actual*	*Travel*	*Error*	*Difference*
OT	24	6.817501	6.777603	0.662324	**-0.622426**
A* 0.1	57	6.817501	6.790489	0.801852	-0.77484
A* 0.2	28	6.817501	6.546429	1.024489	-0.753417
A* 0.3	18	6.817501	**6.382557**	1.189037	-0.754093

C. *Scenario 3: Tight gap obstacles*

In Fig 10, it is clear that A* unable to make a route through a narrow gap, resulting OT has better scores as shown in Table III. It is possible to generate the shortest path by making node gap smaller, but the nodes count would increase significantly, and more nodes mean more calculating process.

Fig. 10. Scenario 3: OT NPC can go through the narrow gap

TABLE III. SCENARIO 3 RESULT

Obs. = 4	Tight Gap Obstacles				
	Node	*Actual*	*Travel*	*Error*	*Difference*
OT	17	5.845109	**5.730087**	0.355027	**-0.240005**
A* 0.1	73	5.845109	8.620098	0.396763	-3.171752
A* 0.2	36	5.845109	8.638044	0.53361	-3.326545
A* 0.3	25	5.845109	8.580089	0.580986	-3.315966

D. *Scenario 4 : Straight row (horizontal) obstacles*

Table IV has shown that A* is better on every configuration because OT weakness was exposed. OT unable to pre-compute the path, that makes its NPC need to encircle every single obstacle in the path as shown in Fig. 11.

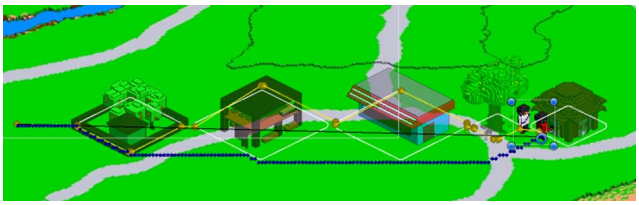

Fig. 11. Scenario 4: A* NPC has the most efficient path

TABLE IV. SCENARIO 4 RESULT

Obs. = 4	Straight Row Obstacle				
	Node	*Actual*	*Travel*	*Error*	*Difference*
OT	27	15.12298	16.01473	0.828928	-1.720678
A* 0.1	145	15.12298	15.15446	0.70184	-0.73332
A* 0.2	73	15.12298	15.29319	0.70184	-0.87205
A* 0.3	48	15.12298	**14.98526**	0.81357	**-0.67585**

E. *Scenario 5: Wide Obstacle*

When the obstacle is modified like on Fig.12, it influences the pathfinding result. Table V shows that A* is better on travel distance from every configuration. While OT needs a little bit longer path because it needs to go to the nearest Raycast hit point first.

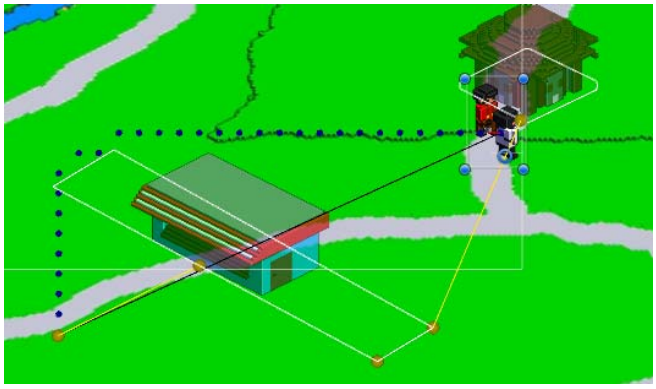

Fig. 12. Scenario 5: NPCs took a different path

TABLE V. SCENARIO 5 RESULT

Obs. = 1	Wide Obstacles				
	Node	*Actual*	*Travel*	*Error*	*Difference*
OT	6	7.84419	9.709723	0.28821	-2.153743
A* 0.1	91	7.84419	9.507587	0.410222	-2.073619
A* 0.2	44	7.84419	9.553211	0.410222	-2.119243
A* 0.3	31	7.84419	**9.274145**	0.62037	**-2.050325**

F. *Scenario 6: River crossing*

The more complex and irregular collider shape like a river in Fig. 13, the longer path need to be traversed by OT

algorithm. However, it does not a matter for A* algorithm since it does not depend on obstacle shape. The result on Table VI, A* is better than OT

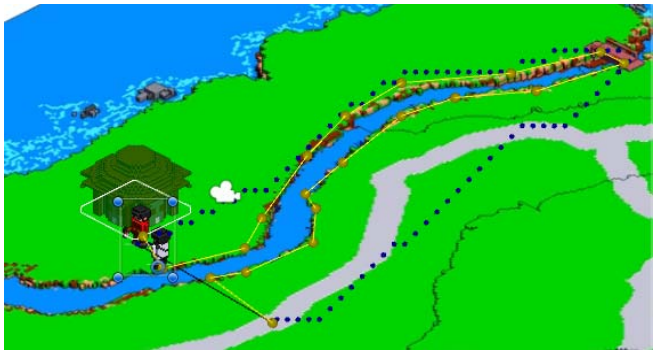

Fig. 13. Scenario 6: OT NPC need to travel alongside the river to reach the goal

TABLE VI. SCENARIO 6 RESULT

Obs. = 1	**River Crossing**				
	Node	***Actual***	***Travel***	***Error***	***Difference***
OT	21	3.158496	21.97575	0.273512	-19.09077
A* 0.1	156	3.158496	**19.12286**	0.318549	**-16.28291**
A* 0.2	80	3.158496	19.46426	0.38832	-16.69408
A* 0.3	54	3.158496	19.69431	0.289024	-16.82484

G. Scenario 7: Non-uniform poly obstacles

One of OT weakness is it need to encircle every obstacle that obstructing the view. In Fig 14, many obstacles have been modified thus make OT a bit longer to encircle. The result in Table VII, A* is the best in every configuration.

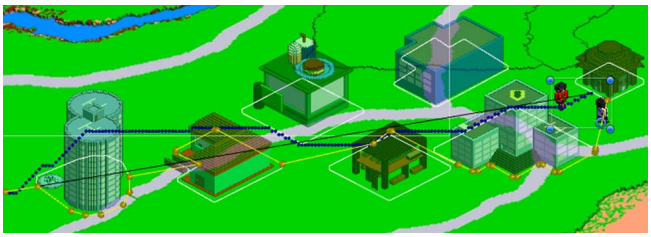

Fig. 14. Scenario 7: OT weakness exposed when the poly is modified

TABLE VII. SCENARIO 7 RESULT

Obs. = 6	**Non-Uniform Poly Obstacles**				
	Node	***Actual***	***Travel***	***Error***	***Difference***
OT	38	18.41779	23.21464	0.295155	-5.092005
A* 0.1	177	18.41779	19.26544	0.557314	**-1.404964**
A* 0.2	88	18.41779	19.20004	0.656201	-1.438451
A* 0.3	59	18.41779	**19.41265**	0.6772	-1.67206

When comparing about runtime speed, unfortunately, A* does not perform better than OT. The result in table VIII show runtime speed for each scenario. In Fig. 15, we can see obvious that OT has the stable runtime because OT only counts the corner/vertices of the collider. However, A* runtime is affected by the traversed node count so the smaller gap between a node of A* will increase node count and computing time.

TABLE VIII. SCRIPT RUNTIME COMPARISON

Scenario	***OT***	***A* 0.1***	***A* 0.2***	***A* 0.3***
1	1.81E-05	0.002579	0.000451	0.000246
2	0.0015984	0.0271087	0.006022	0.004364
3	0.0012391	0.0765834	0.0151963	0.00708
4	0.0014989	0.1833878	0.0280047	0.008471
5	0.0009871	0.0960097	0.0319581	0.006722
6	0.001262	0.8707223	0.1223979	0.078486
7	0.0016031	0.4479461	0.0750437	0.02527

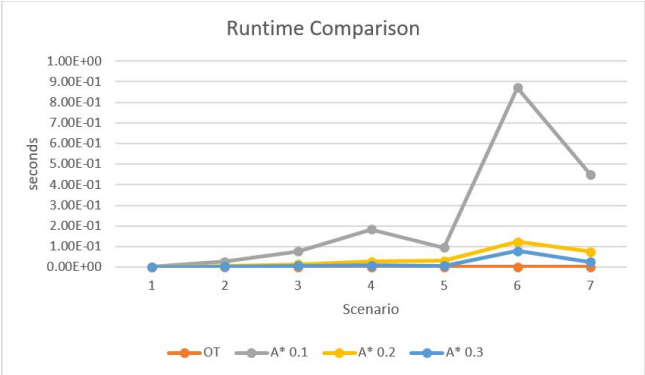

Fig. 15. Runtime comparison graph

IV. CONCLUSION

The A * method can be implemented well in isometric non-grid games with the requirement to create a virtual grid for search space. A* can solve the shortest path regardless of the obstacle shape as long as the generating nodes have not caught in the dead end. The reason A* lose on the term of shortest path compared to Obstacle Tracing is that of jagged movement pattern caused by the grid or when the gap between A* node is larger than the gap between obstacles. However, Obstacle Tracing is better on runtime performance.

V. ACKNOWLEDGMENT

This research supported by Universitas Muhammadiyah Malang.

REFERENCES

[1] Uniy3D Game Engine, [online] Available : https://unity3d.com/ [11March 2018]

[2] M.B. David and S. Glenn, *AI for Game Developers: Creating Intelligent Behavior in Games,* O'Reilly Media, 2014

[3] Hidayati. Q, Rachman. F. Z, & Yanti, N. "Intelligent Control System of Fire-Extinguishing and Obstacle-Avoiding Hexapod Robot," *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, vol. 3, no. 1, pp 1-10. 2017

[4] M. Jason, *Pathfinding Strategy for multiple non-playing characters and agents in a 2.5D Game World,* Athabasca University, Athabasca, Alberta, Canada, 2009

[5] M. Ian and F. John, *Artificial Intelligence for games*, 2nd ed. Morgan Kaufmann Publisher, pp. 215, 2009

[6] B. Ray, S.K. Aung, P. Clifford, and N.S. Thet, *Unity AI Game Programming,* 2nd ed. Packt Publishing, pp. 75, 2015

[7] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006

[8] H. Zhang, S. Minyong, L. Chunfang, "Research and Application of Path-finding Algorithm Based on Unity 3D", *IEEE ICIS 2016,* June 2016

[9] C. Xiao and S. Hao, "A*-based Pathfinding in Modern Computer Games", *IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1,* January 2011

[10] C. Xiao and S. Hao, "Direction Oriented Pathfinding In Video Games", *International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.4,* October 2011

[11] R. Anbuselvi and M.Phil, "Path Finding Solutions For Grid Based Graph", *Advanced Computing: An International Journal (ACIJ), Vol.4, No.2,* March 2013

[12] Y. Bjornsson, M. Enzenberger, R. Holte, J. Schaejfer, and P. Yap, "Comparison of different grid abstractions for pathfinding on maps", In Proceedings of the 18th International Joint Conference on Artificial Intelligence, San Francisco, pp.1511-1512, 2003

[13] S. Yoppi, S. Hadipurnawan, S. Muhammad, "Comparison of A* and Dynamic Pathfinding Algorithm with Dynamic Pathfinding Algorithm for NPC on Car Racing Game", *11th International Conference on Telecommunication Systems Services and Applications (TSSA),* Oktober 2017

[14] N. Azad, and M. Farzad, "Simulation and Comparison of Efficiency in Pathfinding algorithms in Games", *Ciência e Natura, v. 37 Part 2, p. 230−238,* June 2015

[15] R.F. Eka, U.M. Siti, and F. Feri, "Comparative Analysis of A* and Basic Theta* Algorithm In Android-based Pathfinding Games", *6th International Conference on Information and Communication Technology for The Muslim World (ICT4M),* November 2016

[16] Unity Documentation, [online] Available : https://docs.unity3d.com/Manual/Raycasters.html [11March 2018]

[17] Unity Documentation, [online] Available : https://docs.unity3d.com/ScriptReference/Physics2D.OverlapPoint.html [11March 2018]