Application of Graph Theory in Navigation in Open-World Games

Nathaniel Evan Gunawan - 13516055
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516055@std.stei.itb.ac.id

***Abstract*—Games have become an inseparable part of our lives. Among these games are open-world games that allow the player to freely explore a given world and approach the game as the player wishes. Navigation has become a necessity in many, if not all open-world games. In this paper we look at how graph theory and A* algorithm may be implemented in these games, and how point-to-point navigation in these games may work.**

***Keywords*— A* algorithm, Dijkstra's algorithm, BFS algorithm, graph theory**

I. History of Games

The rapid advancement of technology during the past several centuries brought about the dawn of digital computers in the 20th century, which back in its early days in the 1930s used to be large, bulky, heavy and much slower, the earliest of which taking up spaces up to the size of a room, and operating at 5 to 10Hz. Since then, it is only a matter of time until computers start taking over a lot of aspects of our daily lives, one of them being entertainment. At this day and age, we find ourselves surrounded by computers to the point where we probably cannot live several consecutive days - let alone an entire year - without having to interact with a computer.

The development of computers also brought about the birth of video games, a form of digital entertainment that has become widely popular and penetrated our society very deeply today, with a lot of so-called "hardcore/competitive gamers" dedicating their lives only to playing video games and being extremely good at it. There have been countless debates on whether video games are more detrimental than beneficial to our society, and it's up to every individual to choose which side he or she is on, but the fact that it has become quite the hot debate topic today only proves that video games have become a major part of the lives of millions or even billions of people, and there is no denying it.

The first video game was invented in 1947 [1], but it was only in the 1970s that video games started to become commercially available to the masses, starting with "Computer Space" [2], a coin-operated video game which utilised a black-and-white screen for its display, and was built from much simpler electronic components compared to today's computing systems. Fast forward to today's era, and it only becomes evident how much video games have evolved; they have become tremendously more complex, with advancements in graphics and game engine far beyond what gamers in the 1970s could have possibly imagined. What started out as simple moving lines on an oscilloscope in the 1950s would evolve into rasters and vectors in late 1970s and early 1980s, and today we have gone far beyond that; we are already seeing tens of thousands of polygons in our current generation video games with photorealistic graphical effects. There simply is no telling what is about to come a decade or two from now.

II. About Open-World Games

This rapid growth of video games has also introduced new genres to the table, including open world games. Open world games are games in which players are given open, virtual worlds for them to roam freely, enabling them to approach the game's objectives as the players wish, as opposed to games with a fixed, linear gameplay. The concept of free-roaming exploration was introduced by the 1976 text adventure game "Colossal Cave Adventure" [3], but it wasn't until the 1980s that the open-world concept finally took on a definite concept [3]. Open-world games like "The Legend of Zelda" and "Ultima" also happened to be received very positively by the masses and critics alike. Since then, numerous open-world games have been spawned, several notable examples being "Super Mario 64", the "Grand Theft Auto" series, "The Elder Scrolls" series, "World of Warcraft", "Minecraft", and the "Assassin's Creed" series. The open-world "genre" has only become increasingly popular since then, mostly favoured by players who prefer extensive explorations and countless possibilities in their games.

Open-world is, for the most part, not usually a separate genre. There exist various kinds of games within the realm of open-world itself. Some of these games can be considered as open-world racing games (the "Need for Speed" franchise is a notable example of this) due to being essentially a racing game but featuring an open world. Some others can be classified as open-world first person shooter (FPS) games, open-world role-playing games (RPG), or open-world adventure games. Whatever the game is, if it features an open world to roam about and explore freely, it counts as an open-world game.

Within the span of 40 years, it is only natural that open-world video games start to introduce more complex mechanics and vaster open worlds to explore, with an unprecedented level of

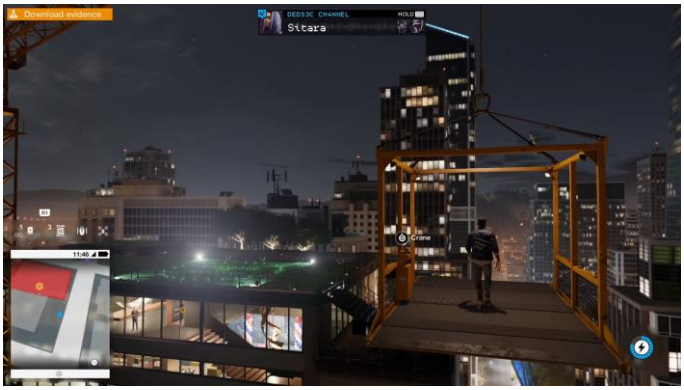


Figure 1: A screenshot of "Watch Dogs 2", an open-world action-adventure video game made by Ubisoft. Source: http://totalgamingnetwork.com/attachment.php?attachmentid=6131 (accessed on Sunday, 3 December 2017)

attention to detail in the design of the open worlds, sometimes spanning up to several multiple cities, with objectives and other game items scattered around that massive map. This calls for a working real-time navigation system inside the game, that allows the players to pinpoint a specific location in the open world, and have the game provide these players with directions that enable them to get to anywhere in the gigantic map, accomplish numerous objectives and/or simply explore the open world to its farthest corners. In other words, a navigation system is only necessary if the developers of the open-world game want the players to play the game how it is intended to be played.

This navigation system is where graph theory and the A* algorithm comes into play. The application of graph theory in open-world games allows the massive open worlds featured in those games to be charted as a weighted graph, and we apply the A* (pronounced as "A star") algorithm to find the shortest path between two points on the map, which is basically the essence of a navigation system. We would also elaborate why the A* algorithm is preferable compared to three other algorithms, namely Dijkstra's algorithm, the breadth-first-search (BFS) algorithm, and the greedy version of BFS.

III. Graph Theory

A. Graphs and Their Types

A graph is a discrete structure consisting of several objects, some pairs of which are connected directly to each other. These objects are called ***vertices***, and they are connected to each other or to itself by means of ***edges***. To be precise, the mathematical definition of a graph is as follows:

A graph is an ordered pair (V, E) consisting of a nonzero set of ***vertices*** (represented by V) and a set of ***edges*** that connect the ***vertices*** (represented by E). [4] As part of the definition of a graph in discrete mathematics, the set of ***vertices*** (V) must not be empty, whereas E may be empty. In such a case where a graph doesn't have any edges, the graph is said to be an ***empty graph***.

There are several different types of graphs, depending on whether: 1) there are any multiple edges that connect the same pair of vertices, 2) the edges have directions, 3) there are any edges that connect a vertex to itself, and 4) whether a certain number (***weight***) is assigned to each of the edges.

A graph which do not have multiple edges connecting the same pair of vertices, whose edges are not directed and all connect different pair of vertices, is called a ***simple graph***. A graph that may have multiple edges connecting the same pair of vertices is called a ***multigraph***. A ***pseudograph*** is a graph that may have not only multiple edges connecting the same pair of vertices, but also loops, i.e. edges that connect a vertex to itself. Note that all the aforementioned types of graphs are ***undirected graphs***, meaning that the edges do not have directions. Graphs whose edges have directions are called ***directed graphs***.

A directed graph which do not have loops, nor multiple edges connecting the same pair of vertices is called a ***simple directed graph***, whereas a ***multiple directed graph*** is a directed graph that may have loops, and edges connecting the same vertices. A ***mixed graph*** is a graph that may have both directed edges and undirected edges, in addition to loops and multiple edges connecting the same pair of vertices. A graph whose edges are assigned with a certain number that represents the ***weight*** of the edge, is called a ***weighted graph***.

Consider the ***mixed graph*** in Figure 2a, as shown below:

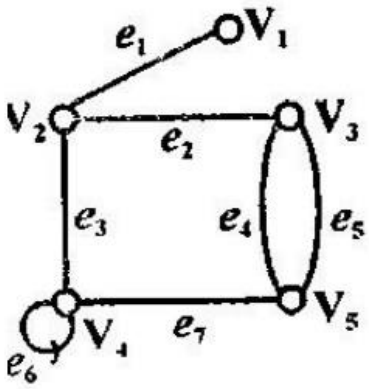


Figure 2a: a mixed graph. Source: http://www.brainkart.com/media/extra/N3jePsE.jpg (accessed on Sunday, 3 December 2017)

e_4 and e_5 are edges which connect the same pair of vertices, namely V_3 and V_5. e_6 is a directed loop, meaning that it is an edge that originates from and points to the same vertex (hence loop) and has a direction shown by the arrow (hence directed). If we take away the direction of e_6, the graph in Figure 2 will become a ***pseudograph***, whereas if we add a direction to the rest of the edges, it will become a ***multiple directed graph***. If we omit e_6 entirely, the graph becomes a ***multigraph***. Omit both e_6 and e_5, and the result is a ***simple graph***. Omitting both e_6 and e_5 and adding a direction to the rest of the edges will result in a ***simple directed graph***. Add a weight to each of the edges, and the result is a ***weighted graph***.

B. Terminologies Related to Graphs

1. ***Adjacency***

Any two vertices that are directly connected to each other by an edge are called ***adjacent***. In Figure 2, V_1 and V_2 are

adjacent to each other, and so are V_2 and V_4, and so on.

2. ***Incidency***

An edge can be called ***incident*** to a pair of vertices if the edge connects a vertex directly to the other vertex. In Figure 2, it can be said that the edge e_2 is ***incident*** to both V_2 and V_3, e_1 is ***incident*** to V_1 and V_2 and so on.

3. ***Degree***

The ***degree*** or ***valency*** of a vertex is the number of edges incident to it. In Figure 2, for example, the vertex V_2 has a degree of 3, while V_1 has a degree of 1.

4. ***Connectivity***

A pair of vertices is said to be ***connected*** if there is at least a pathway that connects the two vertices, directly or indirectly. A graph is considered ***connected*** if every possible pair of vertices in the graph are ***connected***. Taking Figure 2 for example, V_1 and V_2 are directly connected, whilst V_1 and V_3 are indirectly connected, but nevertheless both pair of vertices are ***connected***, and so is every other possible pair of vertices, making the graph in Figure 2 a ***connected*** graph.

5. ***Subgraph***

A graph $G_1 = (V_1, E_1)$ is considered a ***subgraph*** of the graph $G = (V, E)$ if V_1 is a subset of V ($V_1 \subseteq V$) and E_1 is a subset of E ($E_1 \subseteq E$). One of the possible subgraphs of the graph in Figure 2 is illustrated in Figure 2b below:

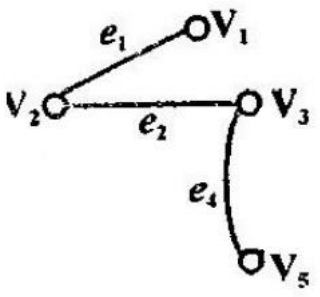


Figure 2b: One of the possible subgraphs of the graph in Figure 2a. Source: http://www.brainkart.com/media/extra/N3jePsE.jpg (edited by the author and accessed on Sunday, 3 December 2017)

C. Application of Graphs

The application of graphs in various aspects of daily life is nothing new; it was first done in the 18th century when Leonhard Euler mathematically solved the notable problem of "Seven Bridges of Königsberg", which resulted in what is regarded as the first paper in the history of graph theory, laying down the very foundations of graph theory in the process. [5] In a nutshell, Euler illustrated the problem of "Seven Bridges of Königsberg" by representing each land mass as vertices, and the bridges as edges, as shown in Figure 3 below:

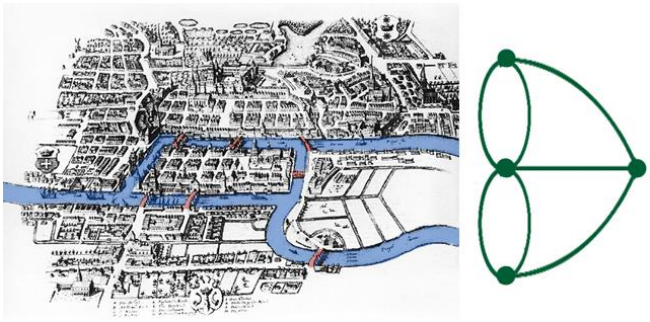


Figure 3: The famous problem of "Seven Bridges of Königsberg" and its graph representation. Source: http://www.mathscareers.org.uk/wp-content/uploads/2015/07/Koenigsberg_bridges.jpg (accessed on Monday, 4 December 2017)

In this paper, we will be applying this very graph theory to a map of a fictitious city in an open-world game. This map will be re-interpreted as a massive, weighted graph with numerous vertices and edges assigned with a weight that represents the distance between two consecutive intersections (which will be represented as vertices). We will also find one search algorithm best suited for the task of navigating from one point to another point on the map.

IV. Search Algorithms

Finding the shortest distance from point to point in large scale is not the easiest problem that can be solved manually by humans. This is exactly the reason why we are using a search algorithm to tackle the problem; we are giving this strenuous task to the computer to solve, then we simply reap the results.

For this purpose, there are 4 search algorithms that we are about to consider: the breadth-first-search (BFS) algorithm, Dijkstra's algorithm, the greedy BFS algorithm, and the A* algorithm. We will then show which algorithm is best suited to tackle this problem.

Let us initialise a starting point and an end. Using BFS, the program will start searching from the starting point by exploring every adjacent vertex from that starting point. From there, it keeps expanding its search area, checking every adjacent vertex of the previous vertices, while keeping track of the vertices that have been visited during the process. Once the destination point has been reached, the program has successfully generated a path to the destination point. Figure 4a below illustrates how BFS accomplishes its goal; the darkened, numbered tiles on the grid collectively represent the search area BFS must explore to reach the designated destination point, the red star represents the starting point, and the purple cross as the destination point.

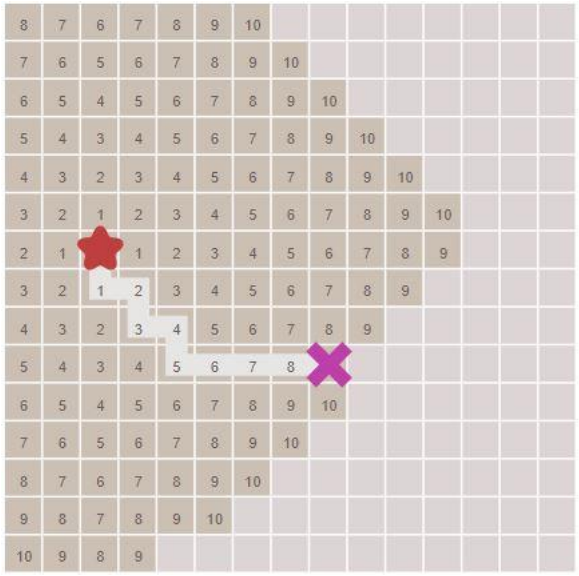


Figure 4a: How BFS works in a non-weighted graph. Source: https://www.redblobgames.com/pathfinding/a-star/introduction.html (accessed on Monday, 4 December 2017)

The BFS algorithm alone isn't quite effective. While it manages to find the shortest path between two points, BFS works blindly without any heuristics to accelerate the process, making it slow and unsuitable for real-time navigation. Moreover, it only works on non-weighted graphs, meaning that it's not suitable to find the shortest path between two points in a map where the distance between any two points on a map may wildly vary.

This is where Dijkstra's algorithm comes into play. Dijkstra's algorithm is essentially BFS, but improves on it by taking the distance between two vertices into account, making it suitable for weighted graphs like maps. [6] Every "step" of the resulting path is then taken based on how "far" it is from the starting point. To grasp a better sense of how Dijkstra's algorithm works, let us observe Figure 4b below, which compares the BFS to Dijkstra's algorithm side by side on a sample grid. The red star represents the starting point, the purple cross as the destination point, the khaki-coloured tiles are vertices which have a weight of 1 between them, and the green-coloured tiles being vertices which have a weight of 5 between them.

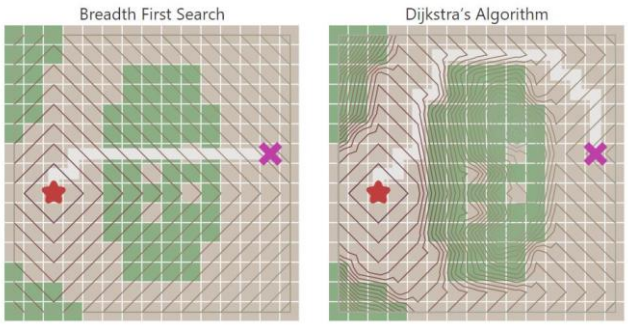


Figure 4b: BFS vs. Dijkstra's in a weighted graph. Source: https://www.redblobgames.com/pathfinding/a-star/introduction.html (accessed on Monday, 4 December 2017)

As evident from Figure 4b above, Dijkstra's algorithm works better to find the shortest path between any two vertices in a weighted graph. However, since Dijkstra's algorithm is essentially BFS for weighted graphs, it is still not fast enough for real-time navigation in open-world games, since Dijkstra's algorithm still must expand its search area to all possible directions in the map, which is redundant if we are only looking to find the shortest distance from one point to another point.

The greedy version of BFS now includes heuristics as part of its algorithm. [6] These heuristics serve as a guide for the algorithm to reach its goal efficiently without blindly exploring the entire map, making it work more efficiently than regular BFS or Dijkstra's algorithm. Imagine a person sitting in the living room of a house, and he or she suddenly smells food being cooked from the kitchen. That smell serves as the person's heuristics, guiding him to where the smell comes from, which in this case is his or her destination point. (The heuristic, in this case, is the Manhattan distance between the two points.)

This greedy version of BFS is still very far from ideal; while it reduces the search area needed to find a path from the starting point to the destination point, it fails to show the shortest path between the two points when we start introducing complexity in the graph by adding barriers. The reason for this is because greedy BFS is only guided by its heuristics; that is, it only tries to get as close as possible to the destination point, not caring how many steps it takes to get there from the starting point. Figure 4c below illustrates how greedy BFS fails to deliver what it is intended to do. In a similar fashion to Figure 4a, the darkened tiles represent the area both algorithms have traversed through, the red star represents the starting point, the purple cross as the destination point, and the blue tiles represent the area that is currently being explored.

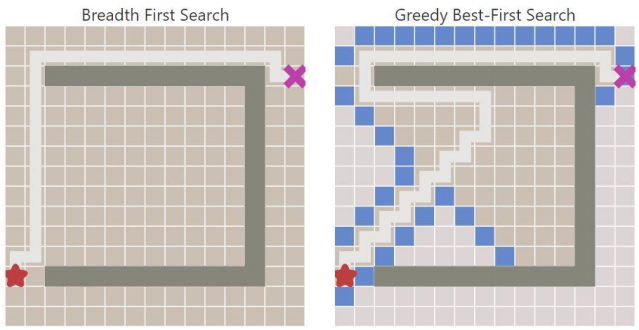


Figure 4c: Greedy BFS works faster, but fails to find the shortest path between the two points. Source: https://www.redblobgames.com/pathfinding/a-star/introduction.html (accessed Monday, 4 December 2017)

Finally, the optimal solution is to combine the "accuracy" of Dijkstra's algorithm and the heuristics of greedy BFS, resulting in algorithm that is both accurate and fast. This algorithm is called the A* algorithm. The A* algorithm works by calculating both the distance of a certain grid from the starting point, and its approximate distance to the destination point. [6] If we are to represent the distance of a certain grid from the starting point as g and its approximate distance to the destination point as h, then the weight of each tile will be represented as $f = g + h$. [7] Figure 4d below illustrates how

the A* search algorithm works. The darkened tiles represent the algorithm's search area, while the numbers on the darkened tiles are the value of every tiles' f. (Again, the heuristic, in this case, is the Manhattan distance between the two points.)

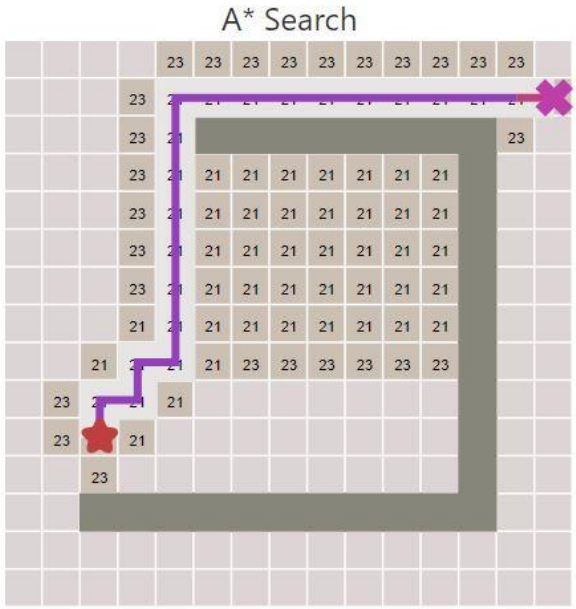


Figure 4d: How A search works. Source: https://www.redblobgames.com/pathfinding/a-star/introduction.html (accessed Monday, 4 December 2017)*

V. Graph Representation of Open-World Map

In this paper, I am going to take the 2009 open-world racing game "Burnout Paradise" as an example of an open-world game which utilises real-time navigation in its gameplay. Being an open-world game, the game obviously features a map of its open world, and this map can be represented as a gigantic graph with many edges and vertices.

Figure 5a: A screenshot of the game "Burnout Paradise". The faint sign reading "Franke Av" indicates that the player is suggested to take a right turn to Franke Avenue at the next intersection. This is the navigation feature of the game.

In this game there are 3 types of events that utilise real-time navigation: "Races", "Burning Routes" and "Marked Man". In "Races", the player races against 7 other CPU players from point to point, the objective being to arrive first at the other end of the race route. "Burning Routes" are like "Time Attack" in other racing games in which the player must get to a certain destination within a certain time limit to win the event. "Marked Man" is a type of event in which the player must arrive at the destination without being taken down more than 3 times by a group of enemies. What all these types of events have in common, is that they all are essentially point-to-point events where the player is free to choose his or her own route from the start point to the destination point. However, the game has a real-time navigation feature, and when the player participates in any of these events, the game will show the player the quickest route which can be taken to help the player reach the destination point of the event as quick as possible.

This navigation feature may be possible thanks to the A* algorithm, which we have shown to be the most effective in finding the shortest route from a point to another point in the map. In fact, it might even be possible that the developers of the game used the very same algorithm for this navigation feature, considering that the algorithm enjoys widespread use today.

We will take a "race" event as an example. This race event is named "Baseball Battle", and the map is as follows:

Figure 3:The start and finish point of the event, along with the relevant part of the map. Red dots represent intersections as vertices/nodes, and the blue lines represent the roads as edges connecting the vertices.

The red dots represent intersections as vertices/nodes, and the blue lines represent the roads as edges connecting the vertices, thereby effectively creating a graph (or to be more accurate, a subgraph of the entire map of the open world) with a defined starting point and a destination. The graph is not a directed graph, since the player is free to drive along the roads in any direction. It is weighted, though, since there's a distance between any two consecutive nodes and the distance widely varies depending on the location.

The game tracks the position of the player real-time, and provides navigation to the player according to the player's location and the endpoint that the player must reach to complete the event. The navigation guidance is shown when the player is nearing an intersection and the player must make a turn to take the fastest route (as shown in Figure 5a). The resulting fastest route is as shown in Figure 5c:

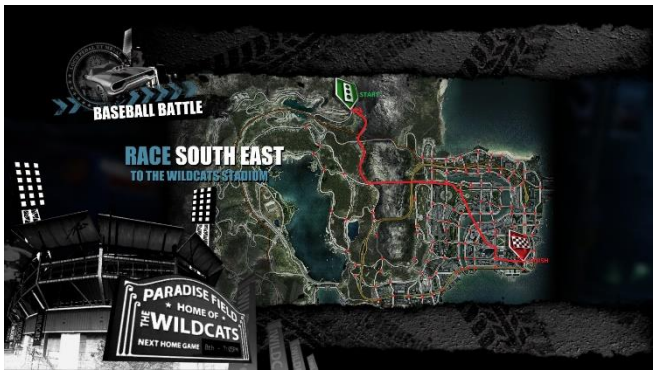


Figure 5c: The fastest route to complete the event, shown by red line. The red line, red dots and red arrow was not captured in-game, but rather drawn by the author as illustration.

VI. Conclusion

Among four of the most common type of search algorithms for finding the shortest route from point to point, namely the BFS algorithm, Dijkstra's algorithm, the greedy BFS algorithm and the A* algorithm, it is evident that the A* algorithm works best for this purpose, and therefore enjoys widespread use today. It can be implemented in various navigation software, including but not limited to open-world video games that feature real-time navigation as an in-game feature.

VII. Acknowledgment

The author would like to thank Dr. Ir. Rinaldi Munir, MT. as the lecturer of the author's Discrete Mathematics class. The author would also like to thank the authors of the resources which the author has referenced to in this paper.

References

[1] U.S. Patent 2455992. [Online] Available: https://www.google.com/patents/US2455992 (link accessed Sunday, 3 December 2017)

[2] Marvin Yagoda (2008). *1972 Nutting Associates Computer Space*. [Online] Available: https://web.archive.org/web/20081228061939/http:/www.marvin3m.com/arcade/cspace.htm (link accessed Sunday, 3 December 2017)

[3] Moss, Richard (March 25, 2017). *Roam free: A history of open-world gaming*. [Online] Available: https://arstechnica.com/gaming/2017/03/youre-now-free-to-move-about-vice-city-a-history-of-open-world-gaming/ (link accessed Sunday, 3 December 2017)

[4] Rosen, Kenneth H. *Discrete mathematics and its applications, 7th ed.* New York: McGraw-Hill, 2012, pp. 641.

[5] University of Kansas, *The Seven Bridges of Königsberg*. [Online] Available: https://www.math.ku.edu/~jmartin/courses/math105-F11/Lectures/chapter5-part1.pdf (link accessed Monday, 4 December 2017)

[6] Red Blob Games, *Introduction to A**. [Online] Available: https://www.redblobgames.com/pathfinding/a-star/introduction.html (link accessed Monday, 4 December 2017)

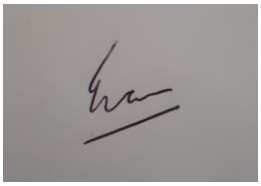
[7] GeeksforGeeks, *A* Search Algorithm*. Available: http://www.geeksforgeeks.org/a-search-algorithm/ (link accessed Monday, 4 December 2017)

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 4 Desember 2017

Ttd (scan atau foto ttd)



Nathaniel Evan Gunawan (13516055)