Crewmate Gameplay Strategy Optimization Using Graph Modelling for "The Skeld" Map in *Among Us*

Ryandito Diandaru 13519157
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13519157@std.stei.itb.ac.id

***Abstract*—*Among Us* is a murder mystery game developed by InnerSloth, it is a game where players roleplay as spacemen and there are two different objectives for each role. This paper is constrained only to be covering the Crewmate role on "The Skeld" map. The objective of a crewmate role is to vote out the hidden impostor(s) among themselves or to finish all the tasks assigned to each player collectively. In order to complete the tasks assigned, players have to walk around the map to go to task areas. To optimize the gameplay strategy of a crewmate, a model in the form of weighted graph is used. The weights of the model graph are determined through the distance it takes to get from a task map to another and a heatmap of player death locations provided by the developers of the game. For each set of tasks assigned, a new complete weighted graph is formed based on the aforementioned model weighted graph. The statistically best route then can be found by finding the Hamiltonian shortest path from said new complete weighted graph.**

***Keywords*— InnerSloth, Dijkstra, Hamiltonian, Weighted, Graph.**

I. Introduction

Among Us is a Unity-built murder mystery/social deduction video game developed by InnerSloth released on June 15th 2018, available on iOS, Microsoft Windows, and Android platforms. A game of *Among Us* lets players roleplay as spacemen (with a minimum and maximum of players of 4 and 10 respectively) in which there are 1 to 3 impostor(s) working together on a spaceship. Depending on the role given by the algorithm of the game, each player has different objectives to complete to win the game. Essentially, the roles assigned divides the players into 2 teams, which are the Crewmates and the Impostors. As an impostor, the objective is to kill crewmates until there are more than or equal number of impostors to the number of crewmates without giving away their status as an Impostor, impostors can also trigger emergency alarms by sabotaging parts of the spaceship as a way to distract crewmates from finishing their objectives, some sabotaging schemes set by the Impostors has to be resolved by the players before a certain amount of time given to resolve said sabotage, otherwise the Impostors win the game. On the other side, to win the game as a crewmate is to vote out all the impostors or to work as a team and finish each and every single task given to every crewmate. Players can call an emergency meeting if a dead body is found and a discussion is open to vote out the suspected Impostor. However, each player can press an emergency meeting button in the middle of the spawning area if they want to forcefully open a discussion, this also leads to a voting of who should be ejected (vote out as impostor). When a player dies, they can still act upon their roles (crewmates can still finish their remaining tasks and impostors can still sabotage the spaceship) but they will not be able to vote and report a dead body, they also cannot talk to the players that are still surviving.

The game mechanics of *Among Us* entails a formulation of a strategy, be it playing as an impostor or a crewmate. One of the obvious strategies of playing for both as a crewmate or as an impostor is to stay alive for as long as possible throughout the game in order to be still be able to vote out impostors/murder crewmates.

The Developers of *Among Us* provided statistics of the game which is accessible through their website. With the data provided, weighted-graph modelling for task completion for the crewmate role is possible. This paper will formulate a weighted graph to model an itinerary for a crewmate in order to complete the tasks given while maintaining the alive status for as long as possible.

II. Literature Review

A. Weighted Graphs

Weighted graphs are a kind of graph, defined by the equation

$$G = (V, E) \tag{1}$$

In which V denotes vertices and E denotes edges. The difference between a non-weighted graph and a weighted graph is that each edge is assigned a certain numerical value, called a weight. The weights assigned may represent various things, they are often referred to as the "cost" of moving through said edge. Weighted graph may or may not be directed. In applications, weighted graph may represent the length of a route, how much it costs to move from one place to another, etc. [1]

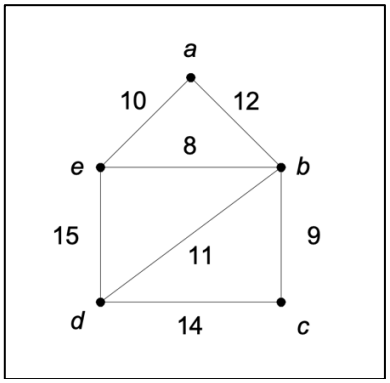

Fig. 2.1. Example of a weighted graph. Image retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

B. Complete Graphs

A complete graph is a simple graph in which every vertex has a corresponding edge that leads to every other vertex on the same graph. A complete graph with n numbers of vertices denoted by K_n has a number of edges denoted by E in the following equation [2]

$$E = \frac{n(n-1)}{2} \qquad (2)$$

A visual representation of a complete graph is as follows:

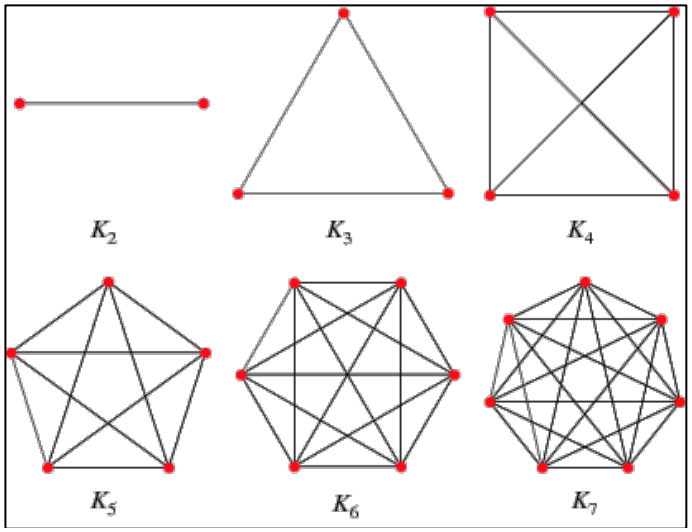

Fig. 2.2. Visual representation of a complete graph. Image retrieved from https://mathworld.wolfram.com/CompleteGraph.html

C. Dijkstra's Algorithm

Dijkstra's Algorithm is an algorithm that finds the shortest path from a weighted graph. The algorithm is known to be the best at present in determining the shortest path from 2 vertices in a weighted graph. In the process, a label is called. Said label consists of 2 different parts. The first part is a letter representing a symbol in front of a point that shows where it is, and the second part, a numerical value which represents weights an edge of 2 vertices. All labels start out as a temporary label in the beginning, and in every algorithm cycle, they make one for a permanent label. Therefore, the shortest/cheapest path can be generated at most by n-1.[3] Below is an example of a pseudocode for Dijkstra's Shortest Path Algorithm.[4]

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

Fig. 2.3. Pseudocode for Dijkstra's Shortest Path Algorithm. Image retrieved from https://www.programiz.com/dsa/dijkstra-algorithm

D. Hamiltonian circuit and path

A Hamiltonian circuit is a path in a graph that goes through every vertex in the graph and goes back to the starting vertex, making the starting vertex got passed twice. Whereas a Hamiltonian path is a path in a graph that goes through every vertex in the graph but does not return to the starting vertex, in consequence, the path passes through every vertex exactly only once. Both Hamiltonian circuit and Hamiltonian path does not need to go through every edge in the graph. A graph is called a Hamiltonian graph if it has a Hamiltonian circuit, and it is called a semi-Hamiltonian graph if it has a Hamiltonian path. Every complete graph is a Hamiltonian graph.[5]

Theorem 2.1 *Every complete graph is a Hamiltonian graph.*

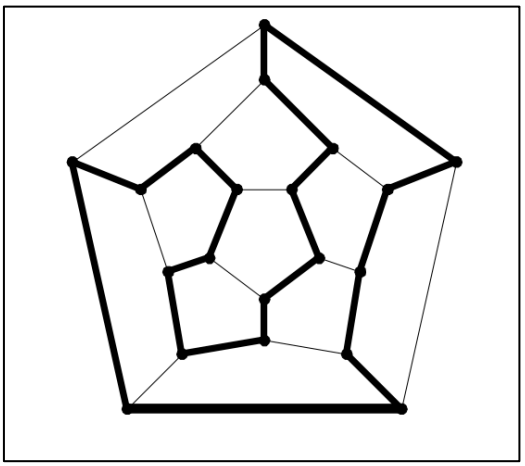

Fig. 2.4. Example of a Hamiltonian graph. Image retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian3.pdf

III. Graph Model Construction

A. Constraints

The version of *Among Us* in which this paper is based on is v2020.11.17o, this version has 3 different maps for players to choose from, and the maps are as follows: "The Skeld", "Mira HQ", and "Polus". This paper will be constrained to only discuss the properties of the map "The Skeld".

Tasks that can be assigned to the players are divided into 3 categories, which are short tasks, long tasks, common tasks, and the categories does not include tasks that was caused by a sabotage by an impostor. The details of the tasks possible to be assigned to the players in "The Skeld" is as follows:

TABLE 1
TASKS ASSIGNED IN "THE SKELD"

Task Category	Task Name
Short Task	Align Engine Output
Short Task	Calibrate Distributor
Short Task	Clean O_2 Filter
Short Task	Clear Asteroids
Short Task	Prime Shields
Short Task	Stabilize steering
Short Task	Unlock Manifolds
Long Task	Upload data
Long Task	Submit Scan
Long Task	Start Reactor
Long Task	Inspect sample
Long Task	Fuel engines
Common Task	Swipe Card
Common Task	Fix wiring

* Multiple areas of the map assigning the same common task is possible, there also exists a task that do not end at the same place. Emergency tasks are excluded from the list.

This paper also will only discuss the properties of the role of a Crewmate, only in which the assignment of the aforementioned tasks is possible. The reason for this decision is that deaths in "The Skeld" that is not on the spawning area can only be caused by the murder of a crewmate by an impostor. As for the tasks, however, impostors can also complete but only the emergency tasks generated from an impostor sabotage. The act of completing self-caused tasks by impostor players if often done to steer off suspicion of themselves.

B. Modelling Bases

The model built is based on "The Skeld" map, the layout of said map is as follows:

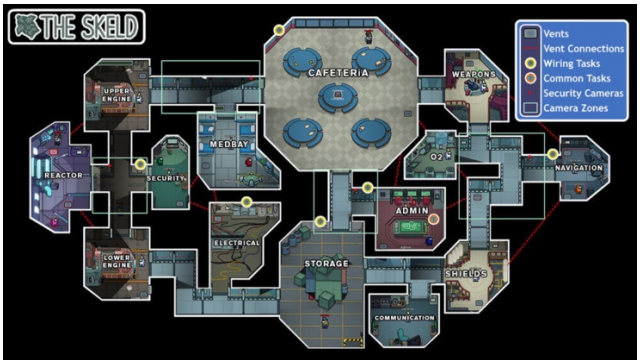

Fig. 3.2.1. "The Skeld" map layout by InnerSloth, with task locations labelled. Image retrieved from earlygame.com

As mentioned before, a few common tasks can take place in more than one place, meaning that the number of vertices of the graph built can exceed the number of the tasks names available for assigning.

To determine the weight of each edge of the graph, death rates of a task location is extracted from the heatmap that shows how many players has ever died in that particular area. The heatmap retrieved from the developer is as follows:

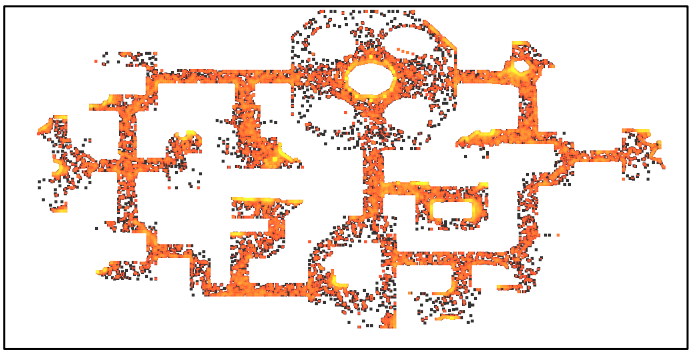

Fig. 3.2.2. "The Skeld" heatmap showing player death locations. Image by InnerSloth (@InnerSlothDevs).

As seen on Fig. 2, every orange block shows that a player has ever died in that area, notice that some areas are significantly brighter than some other areas, this means that more players playing as a crewmate has ever got killed by an impostor on that particular location. An exception is to be made on the spawning locations, that is to minimize an error since players can "die" on the spawning area by logging out of the game once it started.

With Fig. 3.2.1 and Fig. 3.2.2 combined, a map that shows task locations and the death rate on the particular area can be generated, the figure is the following:

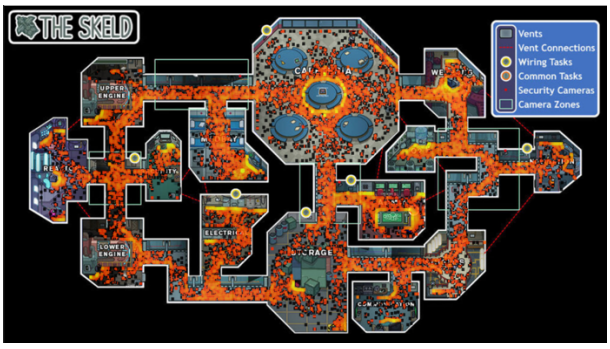

Fig. 3.2.3. Combined image of the heatmap of player deaths and "The Skeld" map layout.

Inside the game itself, the players will be given an abstraction of the map which locates the tasks so that the said players are aware of where to navigate.

Fig. 3.2.4. Task map abstraction, yellow alert symbol indicates a task to be finished. (InnerSloth)

C. *Establishing the task itinerary model*

The weight of the graph is determined from the data extracted from the heatmap as seen in Fig. 3.2.2, the weight of the graph represents the risk to get from one point to another. The weighting calculation is based on the assumption that the more people get killed in that particular area, the riskier the area gets. Notice that places with bright colors tend to be places of doing tasks, especially those of long tasks.

Hexadecimal values for each task location are to be extracted and ranked based on their brightness on the scale of 1-10 with 1 as the place where it is the least likely for a player to be murdered by an impostor and 10 being the most likely and designate them to the corresponding categories based on the rank of each vertex. The tool used on this method is a color-sorting tool on elektrobild.org/tools/sort-colors, in which hexadecimal values is put and the algorithm will sort the colors based on said hexadecimal values. The unordered list of hexadecimal values extracted from the heatmap are as follows: #ff4e29, #ffd333, #f4ff33, #ff9d33, #f5ff33, #ff8033, #ffeb33, #ff7027, #ffe033, #ffcf33, #ffbc33, #fefe33, #ffb633, #f3ff33, #fff733, #f2ff2a, #fffc33, #ff5933, #ff8333, #ffe633, #ffda33, #ffe733, #ff9633, #f3ff33, and the result is as seen on the figure below

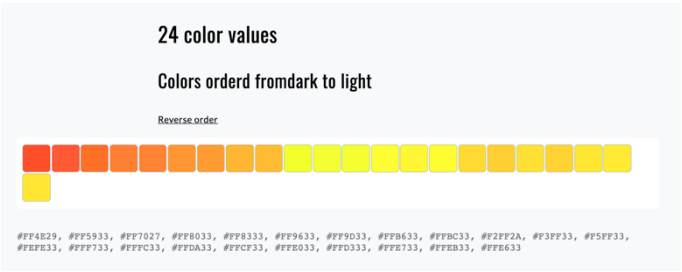

Fig. 3.3.1. Color sorting from dark to light output from elektrobild.org/tools/sort-colors

An error is seen on the picture, while colors ranging from #f2ff2a to #fffc33 is significantly brighter than the rest of the values, they are not on the bright end of the list. A correction and categorization procedure yields a properly categorized hexadecimal values, as seen in the table below.

TABLE 2
CATEGORIZATION OF HEXADECIMAL VALUES OF DIFFERENT SHADES OF COLORS BASED ON THE "THE SKELD" DEATH LOCATION HEATMAP

Hex Category	Hex Value
2	#ff4e29
3	#ff5933
4	#ff8333
4	#ff8033
4	#ff7027
5	#ff9633
5	#ffb633
5	#ffbc33
5	#ff9d33
6	#ffda33
6	#ffd333
6	#ffcf33
7	#ffe733
7	#ffe633
7	#ffe033
7	#ffeb33
8	#fff733
9	#fffc33
9	#fefe33
10	#f3ff33
10	#f2ff2a
10	#f3ff33
10	#f5ff33
10	#f4ff33

After the hexadecimal values has been properly assigned to the correct categories, an abstraction of the task areas with a color-coded graph can be generated, along with numerical values which represents the risk of being on that specific area. At this point, the edges of the graph are still representing the nearest task locations. Edges on the following graph is based on whether 2 neighboring vertices takes place in the same room, or whether they are merely close to each other.

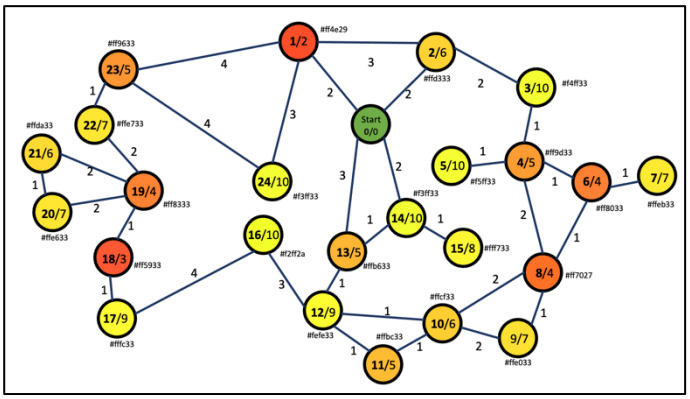

Fig. 3.3.2. Color-coded graph model generated, values on the vertices determined from data extracted from the heatmap.

The two values on the vertices of the graph above on the format X/Y represent the index and the category it falls on respectively.

The numerical values ranging from 1-4 on the edges shown in the graph in the figure above represents the distance between task locations. It is wise to put the distance into consideration because based on the heatmap, a case where a crewmate gets killed by an impostor while moving from one place to another is

also likely.

After generating the model, a Hamiltonian shortest path is to be calculated to find the most statistically effective task completion itinerary.

IV. Task Itinerary Calculation

Each time a game starts, each crewmate will get a set of missions that need to be completed, each set of mission generated by the game will generate a unique weighted graph based on the weighted graph in Fig. 3.3.2.

All vertices that needed to be a member of the new graph is identified to generate a unique complete weighted graph. This is done by identifying which tasks are assigned to the player and matching them from the map to the corresponding vertex as shown in Fig. 3.3.2, e.g., Stabilize Steering task is assigned, then vertex no. 7 is needed to be in the itinerary.

Let U be the set of all vertices of a unique graph generated for each game as a crewmate and let A be the set of all vertices in the graph shown in Fig. 3.2.2. For each U generated by a game in "The Skeld" map, the sets U and A will always satisfy

$$U \subseteq A \tag{3}$$

Thus, the graph for each set of tasks is a subgraph of the graph in Fig. 3.3.2, and the vertices of the said unique graph is a subset of the set of all vertices of Fig. 3.3.2.

The edges for the set of vertices U is then to be determined, in which each possible edge from 2 different vertices must be calculated. The calculation of the weight for each edge is defined by the equation below

$$W = \left\lfloor \frac{C_1 + C_2}{2} \right\rfloor + P \tag{4}$$

W = Weight calculated
C_1 = Origin vertex category
C_2 = Destination vertex category
P = Shortest distance from vertex C_1 to C_2

Equation (4) above shows W being the weight calculated, C_1 and C_2 being the category of 2 vertices desired to be calculated, and P being the added total of the distance of shortest/cheapest possible path between vertex C_1 and C_2 determined by Dijkstra's algorithm, referring to the graph shown in Fig. 3.2.2. Thorough calculations of all possible edges from the set U entails a complete graph with final edge values. From the complete graph created, a Hamiltonian shortest path is to be determined to be the best possible route to take in order for a crewmate to complete the tasks. Note that for every task finished, a recalculation of the Hamiltonian shortest past may be necessary due to the reset of a task vertex category to zero upon completion and the existence of some tasks that do not end at the same place.

Example 1: Consider the following task assigned to a crewmate.

- Admin: Swipe Card
- Cafeteria: Empty Garbage
- Electrical: Divert Power to Shields
- Weapons: Download Data

With the task locations given in Fig. 4.1 below

Fig. 4.1. Task locations for Example 1.

From the set of tasks above, a complete weighted graph to model the remaining tasks to be done by the player is generated.

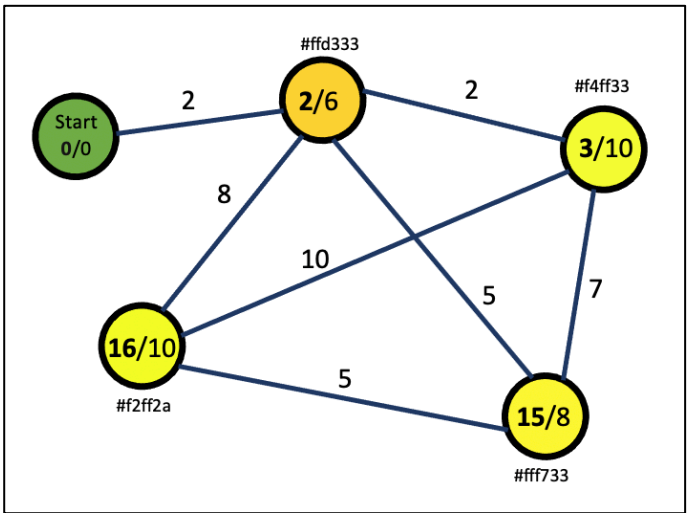

Fig. 4.2. Weighted graph generated for Example 1.

For the sake of time efficiency, the starting point is set to only have one edge. The edge given to the starting vertex is the starting edge for the case in this example. The edges of the graph above is the result of the shortest path calculation, which is the minimum of all possible sum of distance weight from each origin vertex to the destination vertex. Also notice that the set of vertices of the graph in Fig. 4.2 is a subset of the set of vertices of the graph in Fig. 3.2.2, thus the graph generated satisfies (3). A weighted adjacency matrix to represent the actual cost of doing task is then calculated using equation (4).

$$\begin{bmatrix} 0 & 10 & 12 & 16 \\ 10 & 0 & 16 & 20 \\ 12 & 16 & 0 & 14 \\ 16 & 20 & 14 & 0 \end{bmatrix}$$

The ordered weight calculation is as seen on the table below.

TABLE 3
ORDERED WEIGHTED ADJACENCY LIST FOR EXAMPLE 1

Final Weight	Edge
10	(2,3)
12	(2,15)
14	(15,16)
16	(2,16)
16	(3,15)
20	(3,16)

The calculation results yield a Hamiltonian shortest path, as shown in Fig. 4.3 below.

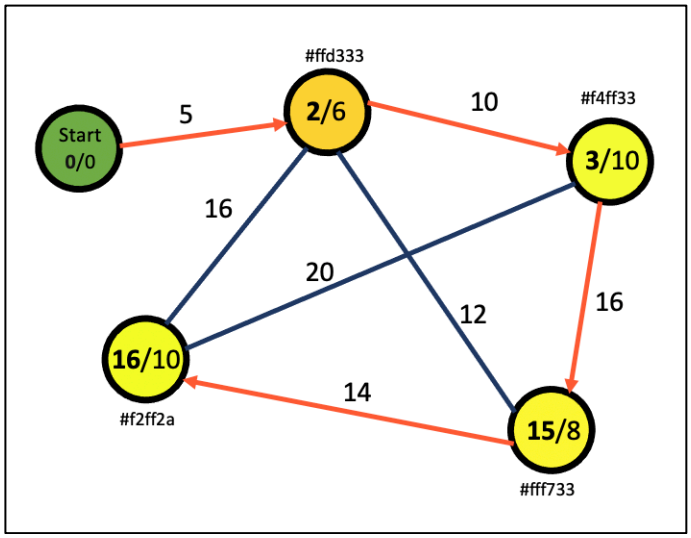

Fig. 4.3. Hamiltonian shortest path for Example 1.

From Fig. 4.3 it is known that statistically the safest path to finishing the set of tasks assigned to the player in Example 1 is to do the tasks through the route as shown on the graph and in the following order:

1. Cafeteria: Empty Garbage
2. Weapons: Download Data
3. Admin: Swipe Card
4. Electrical: Divert Power to Shields

The Hamiltonian shortest path may need recalculation by the time the player finished some of the tasks. Nonetheless, the Hamiltonian shortest path generated is valid for the time being.

V. CONCLUSION

The study of discrete mathematics is applicable to various aspects of human life, that includes even things such as video games. *Among Us* is a survival strategy and also a social deduction game where some branches of the study of Discrete Mathematics is applicable, in which case are graphs and algorithms for finding the shortest/cheapest to get from one point to another. A model in the form of a weighted graph is built to help formulate a strategy for playing a game of *Among Us* as a crewmate in "The Skeld" map. Optimization of crewmate gameplay strategy is done by generating a new complete weighted graph based on the model created and the tasks assigned. That way, the calculation of a Hamiltonian shortest route which shows the best statistical route to take to finish tasks is possible. However, recalculation after each task completion may be needed.

VI. ACKNOWLEDGEMENTS

First and foremost, I want to thank Allah who has given me life and the beautiful gift of human consciousness. Without which, I would never have even been where I am right now, a college student, inheriting years and years' worth of human knowledge development. I want to say thanks to my mother and father, who have supported me from the first day I came into this world, who are there on the ups and downs and the rights and lefts of my life. To all my college professors, especially Dr. Ir. Rinaldi Munir, MT., who had taught me a great portion of the study of Discrete Mathematics, which will surely be useful for a career in computer science, and to all my friends and loved ones, whose acts of love and support, no matter how small, shall not go unnoticed.

REFERENCES

[1] Mcain, "Weighted Graphs Data Structures & Algorithms," 2009.
[2] R. Munir, "Graf(Bag.1)," vol. 1, p. 58, 2020.
[3] Y. Z. Chen, S. F. Shen, T. Chen, and R. Yang, "Path optimization study for vehicles evacuation based on Dijkstra algorithm," *Procedia Eng.*, vol. 71, pp. 159–165, 2014.
[4] Programiz, "Dijkstra's Algorithm." [Online]. Available: https://www.programiz.com/dsa/dijkstra-algorithm. [Accessed: 07-Dec-2020].
[5] R. Munir, "Graf (Bag.3)," p. 48, 2020.

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, 3 Desember 2020

Ryandito Diandaru 13519157