# Graph Theory Implementation in the Puzzle Game KAMI 2

Reyhan Emyr Arrosyid - 13519167
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13519167@std.stei.itb.ac.id*

***Abstract*—Entertainment has always been a part of human life. Entertainment can come in many different forms. One of those forms is video games. Video games themselves are a massive field and can be divided into many genres. One of the genres is puzzle games. Puzzle games often require thinking to solve challenges and missions given by the game. As game developers try to find new and unique ways to make a puzzle game more challenging, sometimes they integrate mathematical concepts such as graphs into their games. An example of such games that will be discussed in this paper is KAMI 2. This paper will cover the implementation of graph theory in the game KAMI 2 as well as a simple program that simulates the game using graphs.**

***Keywords*—Graph Theory, KAMI 2, Puzzle game.**

## I. INTRODUCTION

Whether it is watching television, listening to music, or playing games, entertainment has always been a part of human history. One of the most popular forms of entertainment in this day and age is video games. With the advent of digital media, video games has become even more widely spread. Video games can be classified into multiple different genres. One of such genres is puzzle. Puzzle games are games in which the primary conflict is not so much between the player-character and other characters, but rather the figuring out of a solution, which often involves solving enigmas, navigation, learning how to use different tools, and the manipulating or reconfiguring of objects. Most often there is a visual or sonic element to the puzzles as well, or at least some verbal description of them [6]. As a way to make puzzle games fresh and more challenging, game developers sometimes integrate mathematical concepts into their games whether deliberately or not. In turn, the player can then use the same concepts to better understand or even solve the puzzle contained in such games. One out of the many puzzle games is KAMI 2.

Fig. 1. KAMI 2 [9]

KAMI 2 is a puzzle game developed by State of Play for Android and IOS. It is a sequel to the first game KAMI. In short, KAMI 2 can be described as a puzzle game with a grid consisting of triangles of different colors. The goal of the game is to color the grid to a single color. A more detailed explanation of the game will be discussed in the later section.

## II. GRAPH THEORY

A graph is a structure that consists of nodes or vertices and edges. A graph is usually used to represent a network of connected objects. Mathematically, a graph is denoted by the pair (V, E) where V is a nonempty set of vertices and E is a set of edges that connect two vertices. As an example, consider the following graph.

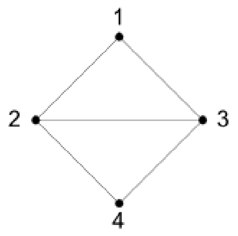


Fig. 2. Example of a graph [1]

Let the graph above be G. The formal notation of G is

$$G = (V, E)$$
$$V = \{1, 2, 3, 4\}$$
$$E = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4)\}$$

### *A. Types of Graph*

Graphs can be classified into multiple different types. Some of the more common ones are as follows:

1. Null Graph

A null graph is a type of graph that has no edges. Null graph can also be called an empty graph. A null graph is denoted by $N_n$ where n is the number of vertices in the graph.

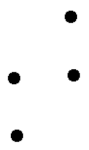


Fig. 3. Null graph [2]

2. Undirected Graph

An undirected graph is a graph in which edges are bidirectional, in other words, every edge in the graph does not have any direction associated to it. Another way of defining it is the edge of an undirected graph forms an unordered pair, so the edge (a, b) is equal to (b, a)

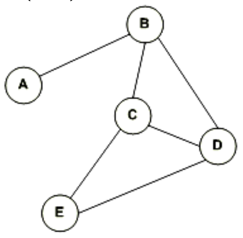


Fig. 4. Undirected graph [2]

3. Directed Graph

In contrast to an undirected graph, edges in a directed graph have a direction associated with them. The edges in a directed graph are represented by an arrow.

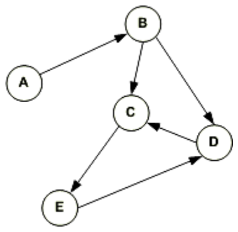


Fig. 5. Directed graph [2]

4. Simple Graph

A simple graph is a graph with no loops and at most one edge connecting two vertices.

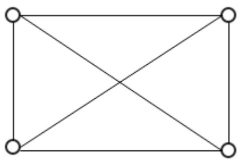


Fig. 6. Simple graph [2]

5. Multigraph

A multigraph is a graph that contains multiple edges connecting the same two vertices.

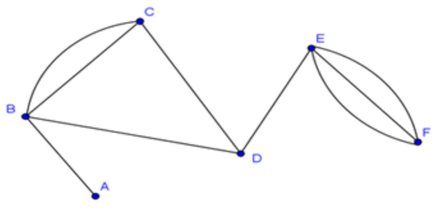


Fig. 7. Multigraph [2]

6. Planar Graph

A planar graph is a graph that can be drawn without any two edges intersecting one another except at a vertex. The graph in Fig. 6 is planar because it can be drawn with no edges intersecting one another.

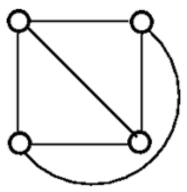


Fig. 8. Planar graph of the graph in Fig. 6 [2]

### *B. Graph Terminology*

In graph theory, there are several commonly used terminologies. Some of the terminologies are as follows:

1. Adjacent

Two vertices are said to be adjacent if they are directly connected by an edge. In Fig. 4, vertex B is adjacent to vertex A, C, and D.

2. Incident

Suppose in a graph exist an edge e = ($v_1$, $v_2$), then it can be said that the edge e is incident with the vertex $v_1$ and $v_2$.

3. Isolated Vertex

An isolated vertex is a vertex that does not have an edge that is incident with it.

4. Degree

The degree of a vertex is a number that represents the number of edges that are incident to the vertex. The vertex B in Fig. 4 has a degree of 3.

5. Path

A path is a sequence of alternating vertex and edge in the form $v_0$, $e_1$, $v_1$, …, $v_{n-1}$, $e_n$, $v_n$ such that every two vertices in the sequence are connected by the edge between the two vertices in the sequence. For example, $v_0$ and $v_1$ are connected by $e_1$.

6. Cycle or Circuit

A cycle or a circuit is a path that starts and ends at the same vertex.

7. Connected Graph

Two vertices are said to be connected if there exists a path between them. The term connected can also be used in the graph itself. A graph is called a connected graph if every pair of vertices is connected.

8. Subgraph

Let G = (V, E) be a graph. It can be said that the graph $G_1$ = ($V_1$, $E_1$) is a subgraph of G if $V_1 \subseteq V$ and $E_1 \subseteq E$.

### *C. Graph Representation*

Graphs have many applications in modeling real-life problems. Computers are often needed in solving problems with graphs. As it is difficult to store a graph in a computer in its original form, a representation of a graph is needed to store the graph. The three most used representations of graphs will be explained in the following section.

1. Adjacency Matrix

In this representation method, a square matrix of size n x n is constructed with n being the number of vertices in the graph. Then let $M_{ij}$ be the element of the matrix in row i and column j. $M_{ij}$ is then set to 1 if vertex i and vertex j is adjacent and 0 if they are not adjacent. The graph in Fig. 2 can be represented with the adjacency matrix below.

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

Fig. 9. Adjacency matrix of Fig. 2 [1]

2. Incidence Matrix

Another way to represent a graph using a matrix is with an incidence matrix. An incidence matrix is a matrix with size m x n with m being the number of vertices in a graph and n being the number of edges in the same graph. Each element of the matrix $M_{ij}$ is then set to 1 if the vertex i is incident with the edge j and 0 if it is not. The incidence matrix of the graph in Fig. 2 is the matrix below.

$$\begin{array}{c|ccccc} & 12 & 13 & 23 & 24 & 34 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 \\ 4 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Fig. 10. Incidence matrix of Fig. 2

3. Adjacency List

The third way of representing a graph is by using an adjacency list. In this representation, for each vertex in the graph, a list of its neighbors is maintained. It means, every vertex of the graph contains a list of its adjacent vertices [3]. The graph in Fig. 2 can be represented with the adjacency list below.

Table 1. Adjacency list of Fig. 2 [1]

| Vertex | Adjacent Vertex |
|---|---|
| 1 | 2, 3 |
| 2 | 1, 3, 4 |
| 3 | 1, 2, 4 |
| 4 | 2, 3 |

### *D. Graph Traversal*

Graphs traversal is a process of systematically visiting every vertex on a given graph. There are many algorithms to traverse a graph. Two such algorithms are Breadth First Search (BFS) and Depth First Search (DFS).

1. Breadth First Search (BFS)

Breadth First Search is a graph traversal algorithm that traverses a graph from a source node to all vertex reachable from that source node. This algorithm traverses the graph layer-by-layer, meaning the algorithm will traverse all vertex that is adjacent to the source vertex first, then continues further in the graph until every connected vertex is visited. Consider the following graph.

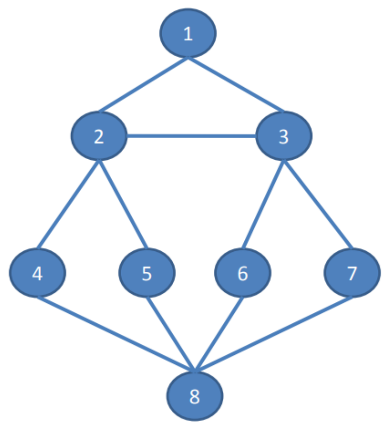


Fig. 11. Example graph [4]

If BFS is used to the graph above with vertex 1 as the starting vertex, the sequence of the visited vertices will be 1, 2, 3, 4, 5, 6, 7, 8.

2. Depth First Search (DFS)

The strategy followed by DFS is, as its name implies, to search "deeper" in the graph whenever possible. DFS explores edges out of the most recently discovered vertex that still has unexplored edges leaving it. Once all the edges have been explored, the search "backtracks" to explore edges leaving the vertex from which was discovered. This process continues until we have discovered all the vertices that are reachable from the original source vertex [5].

If DFS is used to the graph in Fig. 11 with vertex 1 as the starting vertex, the sequence of the visited vertices will be 1, 2, 4, 8, 5, 6, 3, 7

## III. Graph Implementation in KAMI 2

### *A. The Puzzle in KAMI 2*

To understand the puzzle of the game, one must first be familiar with how the game works. In this section, the core gameplay of the game KAMI 2 will be discussed.

First, the player is presented with a selection of levels, upon selecting a level, the player would then be shown a board comprised of colored regions. An example of a level in the game is shown below.

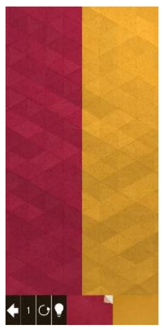


Fig. 12. Example level in KAMI 2

The objective of the game is simple. The player needs to reduce the grid to a single color. The player can do this by selecting a color in the bottom part of the screen then selecting a region to color. If the player colors a region to the same color as an adjacent region, the two regions will merge creating one larger region. Suppose from Fig. 12, the player decides to color the yellow region red, then the whole board will turn to red, thus winning the game. The action of selecting a color and a region is called a move. Each level has a limit on how many moves a player can do. This limit is what creates the challenge of the game.

Fig. 13. Winning state of Fig. 12

## *B. Graph Model*

To model the game using graphs, it must be determined what the vertices and the edges of the graph are representing. A vertex on the graph represents one contiguous region of the same color and the edge represents two adjacent regions. The vertex is then colored by the color of that region. The level in Fig. 12 can be represented by the graph below.

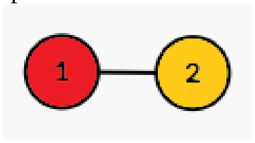


Fig. 14. Graph representation of Fig. 12

It is not too difficult to see the representation since the level only has two regions. Consider the following more complex level

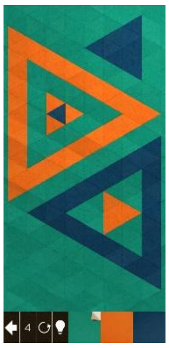


Fig. 15. Another level in KAMI 2

Systematically, the process of transforming a level into a graph is as follows:

1. Label every region in the level
2. Create a colored vertex for each labeled region
3. For each pair of adjacent regions, if the corresponding vertices are not yet connected, connect them with an edge.

The labeled level and the resulting graph from Fig. 15 are the figures below.

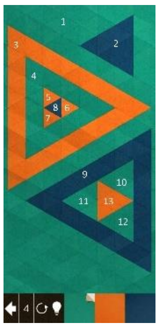


Fig. 16. Labeled level of Fig. 15

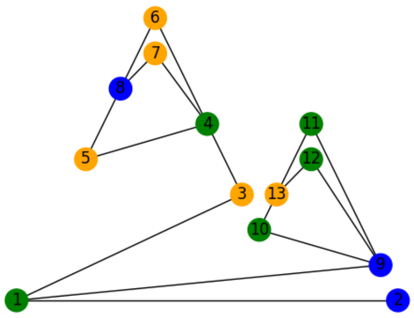


Fig. 17. Graph representation of Fig. 15

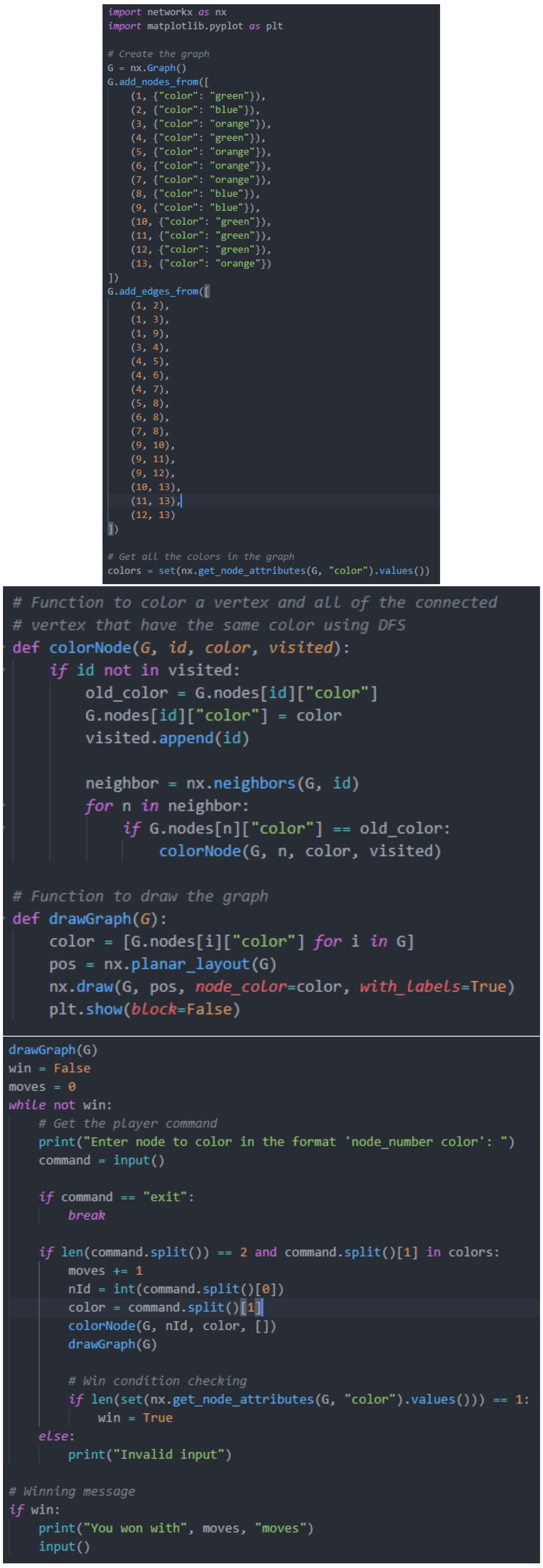
In terms of the graph representation, the objective of the game is to make all vertices of the graph the same color. A move in the game now consists of three steps:

1. Select a vertex *v* and a color.
2. Color the vertex *v* with the selected color.
3. Get a vertex that is adjacent to *v*, if the vertex has the same color as the original color of *v*, repeat from step 2 for that vertex. Repeat for all adjacent vertex of *v*.

Upon further inspection, the third step is similar to traversing a graph and thus could be implemented with a modified graph traversing algorithm such as BFS or DFS. The implementation of the game will be described in the following section.

## IV. PROGRAM IMPLEMENTATION

To better understand the concept of graph theory in the game KAMI 2, a program that simulates the game can be implemented. The program is not a perfect one-to-one representation of the game, rather than implementing a limit on how many moves a player can do, the program counts the number of moves the player has done instead and will be displayed once the player has won. The program is implemented using Python 3 with NetworkX and Matplotlib libraries. Below are the codes of the program implemented with DFS as the traversal algorithm.



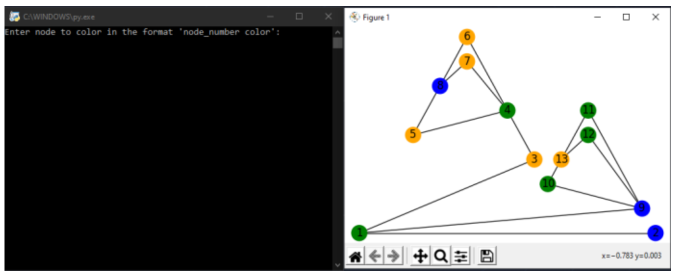
```python
import networkx as nx
import matplotlib.pyplot as plt

# Create the graph
G = nx.Graph()
G.add_nodes_from([
    (1, {"color": "green"}),
    (2, {"color": "blue"}),
    (3, {"color": "orange"}),
    (4, {"color": "green"}),
    (5, {"color": "orange"}),
    (6, {"color": "orange"}),
    (7, {"color": "orange"}),
    (8, {"color": "blue"}),
    (9, {"color": "blue"}),
    (10, {"color": "green"}),
    (11, {"color": "green"}),
    (12, {"color": "green"}),
    (13, {"color": "orange"})
])
G.add_edges_from([
    (1, 2),
    (1, 3),
    (1, 9),
    (3, 4),
    (4, 5),
    (4, 6),
    (4, 7),
    (5, 8),
    (6, 8),
    (7, 8),
    (9, 10),
    (9, 11),
    (9, 12),
    (10, 13),
    (11, 13),
    (12, 13)
])

# Get all the colors in the graph
colors = set(nx.get_node_attributes(G, "color").values())
```

```python
# Function to color a vertex and all of the connected
# vertex that have the same color using DFS
def colorNode(G, id, color, visited):
    if id not in visited:
        old_color = G.nodes[id]["color"]
        G.nodes[id]["color"] = color
        visited.append(id)

        neighbor = nx.neighbors(G, id)
        for n in neighbor:
            if G.nodes[n]["color"] == old_color:
                colorNode(G, n, color, visited)

# Function to draw the graph
def drawGraph(G):
    color = [G.nodes[i]["color"] for i in G]
    pos = nx.planar_layout(G)
    nx.draw(G, pos, node_color=color, with_labels=True)
    plt.show(block=False)
```

```python
drawGraph(G)
win = False
moves = 0
while not win:
    # Get the player command
    print("Enter node to color in the format 'node_number color': ")
    command = input()

    if command == "exit":
        break

    if len(command.split()) == 2 and command.split()[1] in colors:
        moves += 1
        nId = int(command.split()[0])
        color = command.split()[1]
        colorNode(G, nId, color, [])
        drawGraph(G)

        # Win condition checking
        if len(set(nx.get_node_attributes(G, "color").values())) == 1:
            win = True
    else:
        print("Invalid input")

# Winning message
if win:
    print("You won with", moves, "moves")
    input()
```

Fig. 18. Game simulation program

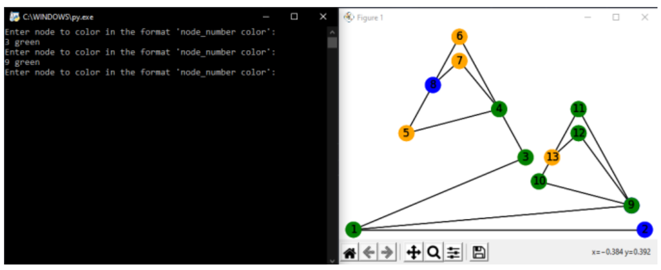
Upon starting the program, the graph representation of a level will be displayed. The player is then able to select a node and color to play the game. Below is the starting display of the program.



```
Enter node to color in the format 'node_number color':
```

Fig. 19. Starting display of the program

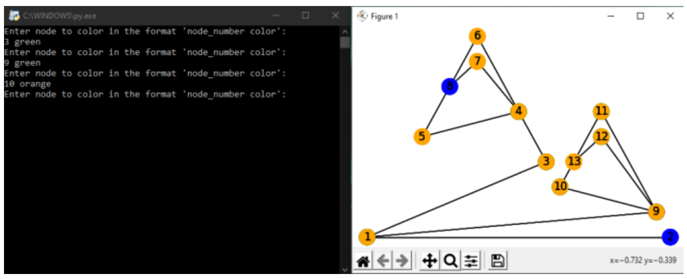
Suppose the player wanted to color nodes 3 and 9 to be green, then the player will need to input '3 green' for the first move and '9 green' for the next move.



```
Enter node to color in the format 'node_number color':
3 green
Enter node to color in the format 'node_number color':
9 green
Enter node to color in the format 'node_number color':
```

Fig. 20. Program display after two moves

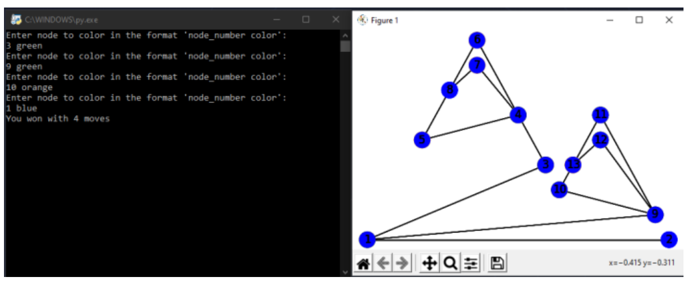
To visualize the coloring algorithm, the input '10 orange' could be entered and all green vertex will turn orange.



```
Enter node to color in the format 'node_number color':
3 green
Enter node to color in the format 'node_number color':
9 green
Enter node to color in the format 'node_number color':
10 orange
Enter node to color in the format 'node_number color':
```

Fig. 21. Program display after the third move

To win the game, the last move will be coloring all the orange vertices blue, this can be done with the input '1 blue'. Below is the final display of the program.



```
Enter node to color in the format 'node_number color':
3 green
Enter node to color in the format 'node_number color':
9 green
Enter node to color in the format 'node_number color':
10 orange
Enter node to color in the format 'node_number color':
1 blue
You won with 4 moves
```

Fig. 22. Winning display

The program that was created is still far from perfect and can still be improved, one such improvement is adding more levels or even implementing automatic graph creation from a level image. Another improvement to the program is to implement an automatic solver that always completes the game with the fewest moves possible.

## V. Conclusion

Graph theory and, by extension, discrete mathematics has many applications in real life. One of the applications is in modeling puzzle video games. Graph theory can be used to better visualize and understand puzzle games. If implemented further, it can even be used to find the solution to the puzzles.

## VI. Appendix

The implemented program mentioned in section IV can be accessed on the author's GitHub page at the link https://github.com/reymyr/KAMI2-Simulation.

## VII. Acknowledgment

First and foremost, the author would like to thank God for the guidance throughout the duration of writing this paper. The author would also like to thank his family and friends for the help in searching for topics for this paper. Lastly, the author would like to thank the lecturers of Discrete Mathematics, Dr. Ir. Rinaldi Munir, M.T., Dra. Harlili S., M.Sc., Fariska Zakhralativa Ruskanda, S.T., M.T., and Dr. Nur Ulfa Maulidevi, ST, M.Sc. for all the knowledge of Discrete Mathematics that made the creation of this paper possible.

## References

[1] Munir. R. *Matematika Diskrit*, Bandung: Penerbit Informatika, 2010, edisi ketiga
[2] Javatpoint. Types of Graphs. https://www.javatpoint.com/graph-theory-types-of-graphs (Accessed 8 December 2020)
[3] Javatpoint. Graph Representations. https://www.javatpoint.com/graph-theory-graph-representations (Accessed 8 December 2020)
[4] Munir. R. Diktat Kuliah Strategi Algoritma ITB (IF2211). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm (Accessed 9 December 2020)
[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2010). *Introduction to Algorithms* (Third ed.). PHI Learning Pvt. Ltd. (Originally MIT Press).
[6] Mark J. P. Wolf. 2001. Genres and the Video Game. *The Medium of the Video Game* (2001), 113–134.
[7] Olston, C. (2020, April 26). Kami2 AI. CMath School. https://cmath-school.com/blog/kami2-ai (Accessed 9 December 2020)
[8] Rozmichelle. Game Theory: An Analysis of KAMI 2. (2020, June 12). http://www.rozmichelle.com/kami2 (Accessed 9 December 2020)
[9] State of Play. KAMI 2. https://www.stateofplaygames.com/kami2 (Accessed 9 December 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.

Jakarta, 9 Desember 2020

[illegible]

Reyhan Emyr Arrosyid 13519167