JTAM (Jurnal Teori dan Aplikasi Matematika)
http://journal.ummat.ac.id/index.php/jtam
p-ISSN 2597-7512 | e-ISSN 2614-1175
Vol. 7, No. 2, April 2023, pp. 522-532

Formation of Non-Perfect Maze Using Prim's Algorithm

Mahyus Ihsan[1], Fahrul Razi[2], Ikhsan Maulidi[3*], Vina Apriliani[4], Zahnur[5]
[1,2,3]Department of Mathematics, Universitas Syiah Kuala, Banda Aceh, Indonesia
[1,2,3]Multimedia Research Group, Universitas Syiah Kuala, Banda Aceh, Indonesia
[3]Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa, Jepang
[4]Department of Mathematics Education, UIN Ar-Raniry, Banda Aceh, Indonesia
[5]Department of Informatics, Universitas Syiah Kuala, Banda Aceh, Indonesia
ikhsanmaulidi@usk.ac.id[1]

Article History:
Received : 28-12-2022
Revised : 06-03-2023
Accepted : 06-04-2023
Online : 08-04-2023

Keywords:
maze;
non-perfect maze;
Prim's algorithm;
Grid graph.

ABSTRACT

Maze is a place that has many paths with tortuous paths that are misleading and full of dead ends and can be viewed as a grid graph. A non-perfect maze is a maze that has a cycle. This research produces an algorithm that can form a non-perfect maze with a size of $m \times n$ which has two types of bias. The first bias is the composition of the percentage of horizontal and vertical partitions. The second bias is the percentage of the number of cycles. The algorithm created in this study was generated by modifying Prim's algorithm and the use of Fisher-Yates algorithm which is used in random selection in Prim's algorithm. The non-perfect maze algorithm begins with the calculation of the parameter values of the two types of bias and continues with forming a perfect maze and ends with forming a non-perfect maze. The algorithm that has been designed can form a non-perfect maze with a complexity of $O(|E|^2)$, where E is the set of edges of an $m \times n$ grid graph. Flash-based application development is also carried out in order to implement algorithms to obtain a non-perfect maze. The non-perfect maze is produced in a two-dimensional visual form in the form of an image along with its corresponding grid graph. The application is capable of displaying up to the first 20 solutions of the biased maze.

https://doi.org/10.31764/jtam.v7i2.12772

A. INTRODUCTION

An algorithm is a well-defined computational procedure that requires a value or a set of values as its input and produces a value or a set of values as its output (Cormen et al., 2022). The algorithm itself has developed very far recently along with the emergence of new problems with a higher level of complexity. Many fields use algorithms as a tool in solving a problem, including the field of insect research which uses algorithms for automatic classification of their research data (Miller, 2019). Another area that uses algorithms as a tool in solving problems is mathematics.

One example of a problem that can be taken is the maze formation process. A maze is a place that has many paths with tortuous paths that are misleading and full of dead ends. In the field of psychology, a maze is used to create learning theories by studying albino rats in studying the maze (Baddeley, 2012). The maze can also be used for map development in

games, for example, Hendrawan produces Android-based maze games using the Growing Tree algorithm (Hendrawan, 2018).

A maze is said to be perfect if there is only one route connecting two arbitrary positions in the maze. On the other hand, a maze is said to be non-perfect if there is more than one route connecting two arbitrary positions in the maze (Fitzgerald et al., 1985). A Perfect Maze has a correspondence with the spanning tree of the grid graph. By adding at least one edge to the spanning tree, it will form a cycle. This cycle forms a non-perfect maze. The structure of a maze is closely related to the spanning tree of the graph. Therefore, the algorithm that forms a spanning tree or a minimum spanning tree can be used in the formation of a generator algorithm in a maze.

One of the algorithms used in spanning tree formation is Prim's algorithm. Prim's algorithm works iteratively by growing a tree on each iteration. Each stage of the prim algorithm will select a new vertex outside the vertex that has been selected where this new vertex is an adjacency of the selected vertex so that each stage of the tree will increase by one vertex. The iteration will stop until all connected vertices form a minimum spanning tree. In maze formation, various algorithms have been successfully developed from research results, one of the algorithms that are easy to apply to maze formation is a search algorithm using the Depth First Search (DFS) (Dubey & Sarita, 2016).

Other maze formation studies have also been carried out using the Prim's algorithm which produces a Perfect Maze (Hutama et al., 2014), where this research became the inspiration to develop research on the formation of the non-perfect maze using Kruskal algorithm (Ihsan et al., 2021). Inspired by these two studies, this research studied non-perfect maze objects with a modified Prim's algorithm. The research resulted in a maze with a changeable ratio of the vertical aisle and horizontal aisle. Other research on mazes can be seen in (Jonasson & Westerlind, 2016), (Hoetama et al., 2018), (Ullah et al., 2022), and (Roy et al., 2022).

In this research, an algorithm for maze formation was developed by modifying Prim's algorithm. The algorithm produced in this study will form a non-perfect maze that depends on the composition of the percentage of horizontal and vertical biases and the percentage of cycles, where the biases in the formation of the maze are useful for producing varied maze shapes. The bias itself is a tendency of dominance between vertical partitions and horizontal partitions. In addition to partition bias, other biases will also be added, namely the percentage bias of the number of cycles contained in the maze. In addition, the complexity of the algorithm that has been made and the visualization of the algorithm in the form of an application are obtained. The benefit of this non-perfect maze is that it becomes a reference for game creators to create variations of the game arena such as the maze game (Nugroho et al., 2017).

B. METHODS

The method of work carried out in this research has several stages as follows:

1. Algorithm Development and Complexity Analysis

The algorithm used in this study is Prim's algorithm, which is the algorithm used to find the minimum spanning tree (Foulds, 2012). This algorithm starts by calculating the parameters input that will be used to determine the appearance of the maze that will be generated later, followed by making a spanning tree obtained using the Prim algorithm and the use of the Fisher-Yates algorithm (Subaeki & Ardiansyah, 2017) in the randomization process and ends by forming maze non-perfect.

For example, given a grid graph of size $m \times n$ and it is known that a value of pc represents the percentage of the cycle base. A cycle base is the set of all cycles in a graph where each cycle contained in a base cycle set cannot be formed by a combination of other cycles in the cycle base set (Galbiati, 2003; Kavitha et al., 2009). Furthermore, it can be determined the value of the parameter k that corresponds to the equation

$$k = \lceil pc(mn - m - n + 1) \rceil. \quad (1)$$

Let js represent the number of partitions contained in a non-perfect Maze which can be determined by the equation

$$js = mn - m - n + 1 - k. \quad (2)$$

Suppose that psv and psh represent the vertical partition percentage and the horizontal partition percentage, where $psv = 1 - psh$. Suppose jsh represents the number of horizontal partitions, jsv represents the number of vertical partitions, jeh represents the number of horizontal edges and jev represents the number of vertical edges. Calculation of these parameters can be determined by the equation

$$jsh = [psh.js], \quad (3)$$

$$jsv = js - jsh, \quad (4)$$

$$jev = (m-1)n - jsh, \quad (5)$$

$$jeh = (n-1)m - jsv. \quad (6)$$

An explanation of the equation determining this parameter can be seen in (Ihsan et al., 2021). The schema of the formation of the algorithm is given in Figure 1.

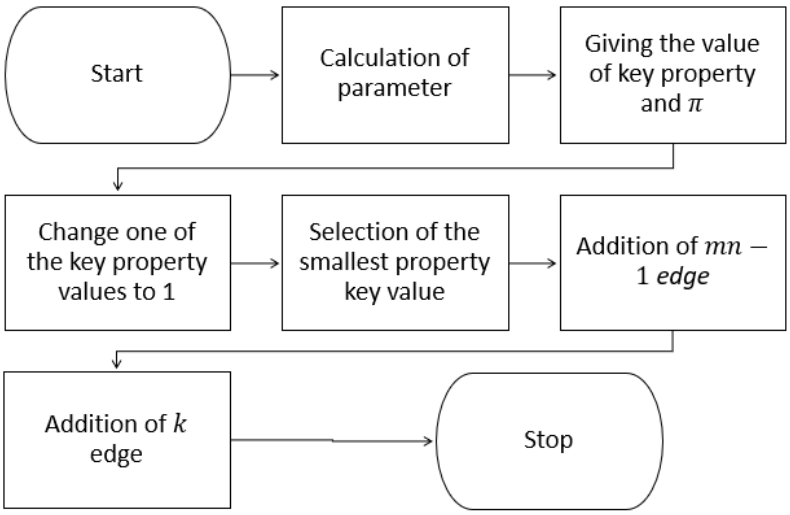


Figure 1. Schematic of Spanning Tree Formation Algorithm

After the algorithm is formed, it is then seen how the complexity of the algorithm Chatterjee & Kiao (2021) and its comparison with the algorithm that has been formulated previously in the research of (Ihsan et al., 2021).

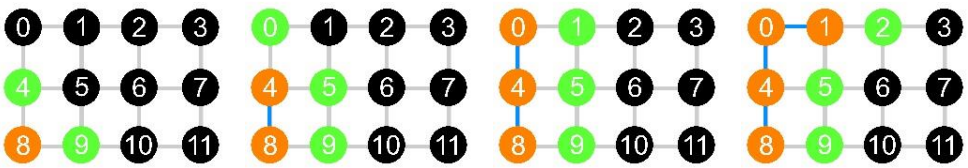
2. Program Design, Coding, and Testing

After obtaining a non-perfect maze formation algorithm, then the algorithm is implemented into a program or application. The application for forming a non-perfect maze refers to Prim's modified algorithm. This stage consists of application design, coding and program testing. The design carried out at this stage includes the design of storyboards and user interfaces. The design of the storyboard is carried out in order to explain how it works and a general overview of the resulting application visualization. User interface design is the design of interaction objects in applications that must pay attention to visual aspects such as colour, layout, and size of objects, as well as the suitability of icon images on buttons and their functions.

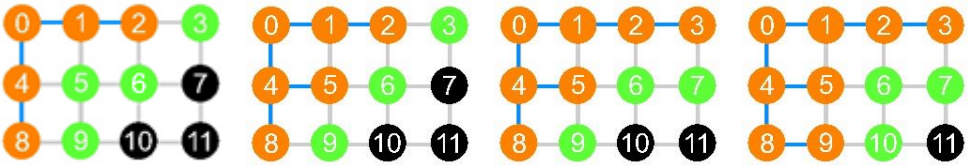
Program coding is done with the help of Adobe Animate CC 2017 software (Brooks, 2016). The programming language used is Action Script 3.0 (Rosenzweig, 2013). This stage is the realization of the storyboard and user interface design into the developed program. In this process, coding is carried out starting from the navigation button which is useful for directing the user to the desired page and followed by coding from the algorithm that has been made. Coding is also done to make some features that will be used in the application. This stage is also carried out to ensure the program produces the correct output. The program will be tested to be able to accept valid input to match the existing parameter types. If at this stage the program appears to have experienced an error, it will return to the coding stage and make improvements. Some references in making storyboards and user interfaces can be seen in (Hart, 2008), (Kung, 2013), and (Groner, 2016).

C. RESULT AND DISCUSSION

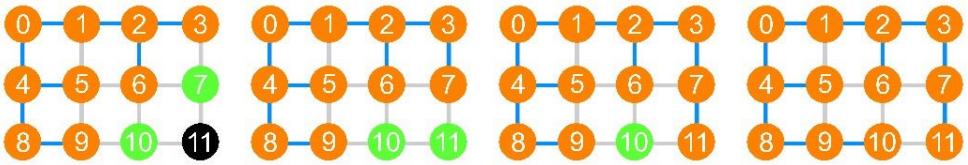
Making a non-perfect maze is obtained by first forming a perfect maze and then removing some of the remaining partitions that are still there according to the desired number of cycles. This aims to ensure that the maze is connected, where graph G which was originally a spanning tree will load cycles according to the addition of edges to graph G. In the spanning tree formation process, edges are selected to be added to the tree where the addition of edges at this stage can not be cause cycles to appear. This is prevented by removing the selected vertex from the vertex set. An edge is added if it is an incident between the selected vertex and the previously selected vertex. The vertex selected at this stage is the adjacency vertex of the previously selected vertex. An illustration of adding edges can be seen in Figure 2 where the orange vertex is the selected vertex and the green vertex is the vertex adjacency of the selected vertex, as shown in Figure 2.



(a) (b) (c) (d)



(e) (f) (g) (h)



(i) (j) (k) (l)

Figure 2. Adding an Edge when Forming a Spanning Tree

In Prim's algorithm, the order of edge selection is done by sorting the edges based on the smallest weight of the edge that is incident with its vertex parent and then reviewing whether the edge is valid or not to be added. Selection using this method is not possible because the weight for all edges in graph G is 1 so that selection based on the order of edge weights is not possible. In the designed algorithm, the order of edge selection is done randomly using the Fisher-Yates algorithm. The purpose of selecting one edge randomly at each stage is to obtain one solution from other possible solutions from the existing candidate edges.

From Figure 2 above, the tracing of vertices starts from state 8. The vertices that are adjacency to state 8 are vertex 4 and 9. In this illustration, the selected state is state 4 (Figure 2b). If the state chosen is state 9, then the solution formed is said to be different. This allows for other solutions to be formed. Based on the illustration in Figure 2, the sequence of states that form a non-perfect maze is 8-4-0-1-2-5-3-9-6-7-11-10.

The algorithm starts by calculating the parameters, namely the value of $k, js, jsh, jsv, jeh, jev$ using equation (1) to equation (6). Furthermore, the addition of edges is done to ensure that there is an edge that connects the rows and columns. The addition of edges is done after the algorithm selects the initial vertex parent at random and selects the vertex that has an adjacency with the vertex parent as the next vertex parent at random which contains the edge as an incident from the two selected vertex. Then adds an edge that connects the vertex that are mutually adjacency, with the maximum number of additional edges as much as $mn - 1$ to form a spanning tree. Edge addition is done by checking whether the added edge meets the jev and jeh values or not by using the CHECK-EDGE procedure. If the addition of an edge meets the conditions, then the edge will be added to set A. After the set $A = mn - 1$, then k edges will be added. Edges that can be added are taken through the set $Q = E - A$ where the set Q represents the set of edges that are not in set A. The selection of this edge is done based on the results of randomization using the Fisher-Yates algorithm on the set of vertex that are checked again by CHECK-EDGE. When the set $A = mn - 1 + k$ then the algorithm will be completed. This makes the resulting graph G is a connected subgraph of the $m \times n$ grid graph which has $mn - 1 + k$ edges and k cycles.

Based on Figure 1, the formation of the spanning tree is the first part of the algorithm, where at this stage the parameters are calculated, adding edges between rows and columns, randomizing edges, and adding $mn-1$ edges. For edge randomization, the Fisher-Yates algorithm is used. The addition of an edge must meet two criteria, namely the edge that is added is an edge that has an incident with the vertex parent and its vertex adjacency which was chosen at random and the added edge does not violate the predetermined jev and jeh. For the criteria for adding k edges that must be met, only the addition of edges does not violate jev and jeh. The overall algorithm can be seen in Table 1.

Table 1. Non-Perfect Maze Algorithm

Algorithm Non-Perfect Maze Algorithm

procedure NON-PERFECT $MAZE(G(m,n,w), psv, psh, pc)$
    $k = \lceil pc(mn - m - n + 1) \rceil$
    $js = mn - n - k + 1$
    $jsh = [psh \cdot js]$
    $jsv = js - jsh$
    $jev = (m-1)n - jsh$
    $jeh = (n-1)m - jsv$
        for setiap *vertex* $v \epsilon G.V$ **do**
        MAKE-SET(V)
        $V.key = 100$
        $V.\pi = NIL$
    SHUFFLE-SET(V)
    $V(0).key = 1$
    $VTemp = V$
    while $VTemp \neq 0$ **do**
        $u =$ EXTRACT-RANDOM-MIN(Q)
        f**or** setiap *vertex* $v \epsilon V.adj[u]$
            if $v \epsilon Vtemp$ dan $w(u,v) < v.key$
                $v.\pi = u$
                $v.key = w(u,v)$
        ADD-RANDOM-EDGE(E_T, v])
    while $E_T < E_T + k$ **do**
        for setiap $e(u,v) \epsilon G.E$
            ADD-EDGE($E_T, e(u,v)$)
procedure ADD-EDGE($E_T, e(u,v)$)
    for setiap $e_T \epsilon E_T$ **do**
        if $e(u,v) \neq e_T$
            CHECK-EDGE($e(u,v)$)
            Tambahkan $e(u,v)$ ke E_T
procedure CHECK-EDGE($e(u,v)$)
    if rasio $v - u = 1$
        $e(u,v)$ adalah *edge* vertikal
    else
        $e(u,v)$ adalah *edge* horizontal

1. Algorithm Complexity Analysis

The complexity of the algorithm is obtained from the algorithm that has been compiled above. The time complexity of lines 2-7 is $O(6)$, followed by initializing vertex and assigning attribute values that require $O(1)$ time which is done as much as $|V|$ so the time complexity on lines 8-11 is $O(|V|)$. SHUFFLE-SET randomization has a time complexity of $O(|V|)$ and EXTRACT-RANDOM-MIN has a complexity of $O(\log|V|)$ which is done as much as $|V|$. The time complexity for ADD-RANDOM-EDGE is $O(\log|E|)$ which is done as much as $|V|$ and lines 22-24 have a time complexity of $O(|E|^2)$ So the total time complexity required is $O(6 + |V| + |V| + |V|\log|V| + |V|\log|E| + |E|^2) = O(6 + 2|V| + |V|\log|V| + |V|\log|E| + |E|^2)$. Due to $6,2|V|$, $|V|\log|V|$, and $|V|\log|E|$ being smaller than $|E|^2$, then the complexity of the algorithm is $O(|E|^2)$. The complexity of the non-perfect maze formation algorithm with Prim's algorithm, when compared with the results of previous studies using the Kruskal algorithm (Ihsan et al., 2021), is still not good enough. This is because in previous studies the complexity of the algorithm obtained was $O(E \log V)$.

2. Algorithm Coding and Non-Perfect Maze Application Design

The coding begins by initializing the vertex and edges contained in the $m \times n$ grid graph. At this stage, labeling of mn vertex will be carried out according to the (i, j) position of the vertex. All labeled vertex will be accommodated in array V which is then assigned the value of the property key and π for each labeled vertex. The categorization of $2mn - m - n$ edges is also carried out by taking into account the type of the edge. The next step is to calculate some of the maze parameters. The input obtained from the algorithm is m, n, psv, psh, and pc sequentially stating the number of rows, number of columns, percentage of vertical partition, percentage of horizontal partition, and percentage of cycle base. The parameters calculated in the algorithm are k, js, jsh, jsv, jev, and jeh which sequentially state the number of additions from the edges after the formation of the spanning tree, the number of partitions, the number of vertical partitions, and the number of horizontal partitions, as shown in Figure 3.

```
k = Math.ceil(persentase_cycle * (Graf.m * Graf.n - Graf.m - Graf.n + 1));
js = Graf.m * Graf.n - Graf.m - Graf.n - k + 1;
jsh = Math.round(psh * js);
jsv = js - jsh;
jeh = new Number((Graf.n - 1) * Graf.m - jsv);
jev = new Number((Graf.m - 1) * Graf.n - jsh);
```

Figure 3. Parameter Coding

The formation of a non-perfect maze begins with forming a perfect maze first, where the initial vertex selection (Va) is carried out randomly using the Fisher-Yates algorithm on the set of Vtemp vertex and continues with the search for its vertex adjacency. After that, it is checked whether the value of the property key of the vertex adjacency is smaller than the weight of the edge that connects Va to its adjacency, if it is true then changes are made to the property value of the vertex adjacency. Next is the selection of the smallest property value from the set of VTemp vertex. After the vertex with the smallest property value is obtained,

the edge will be added according to the next selected vertex while paying attention so that the added edge does not violate the predetermined *jev* and *jeh*. If there is a case of a vertex with the smallest property value more than one, a random selection will be made using the Fisher-Yates algorithm. This happens because the weight value is assumed to be equal to 1 for each edge in graph G. Since Prim's algorithm only works on weighted connected graphs, then to overcome the graph equivalent of the maze created where the graph of the maze equivalent is a connected grid graph that is not weighted, then an assumption is made for the weight of each edge on the grid graph to be 1, as shown in Figure 4.

```
if (Math.abs(Va - RAdj2) == rh) {
    PilihAngka1 = Math.min(Va, RAdj2);
    i = new Number(Math.floor(PilihAngka1 / n));
    j = new Number(PilihAngka1 % n);
    TampungEH.push(i * (n - 1) + j);
    KeepEdge_H.push(i * (n - 1) + j)
    Tampung_VAH.push(Va);
    Tampung_VPH.push(RAdj2);
} else {
    PilihAngka1 = Math.min(Va, RAdj2);
    i = new Number(Math.floor(PilihAngka1 / m));
    j = new Number(PilihAngka1 % m);
    TampungEV.push(i * m + j);
    KeepEdge_V.push(i * m + j);
    Tampung_VAV.push(Va);
    Tampung_VPV.push(RAdj2);
}
```

Figure 4. Edge Increment Coding Based on Selected Vertex

The next step is to form a cycle by adding edges to the spanning tree that has been formed. This is done by first checking all edges in the set of edges and checking whether the edges have been selected or not in the loop. If there is an edge that has been selected, then the CekSama attribute will be added 1. Edge will be added if after checking the iteration, the value of the CekSama attribute is 0. The visualization media application that forms a non-perfect maze consists of nine frames, namely the main menu page, developer page, algorithm visualization page, maze list page, parameter limitation page, and the rest are guide pages. The relationship between the nine page frames can be seen in Figure 5 and Figure 6.

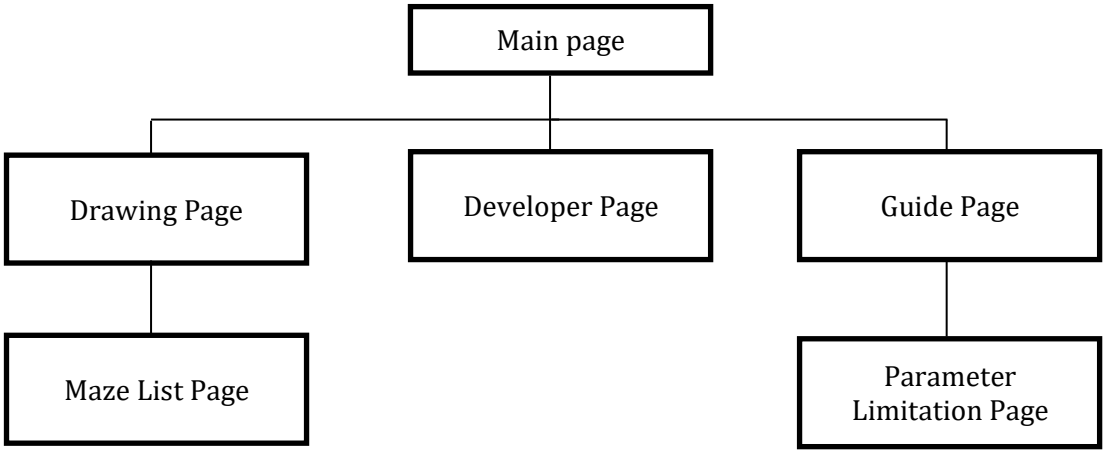


Figure 5. Relationship Diagram between Pages

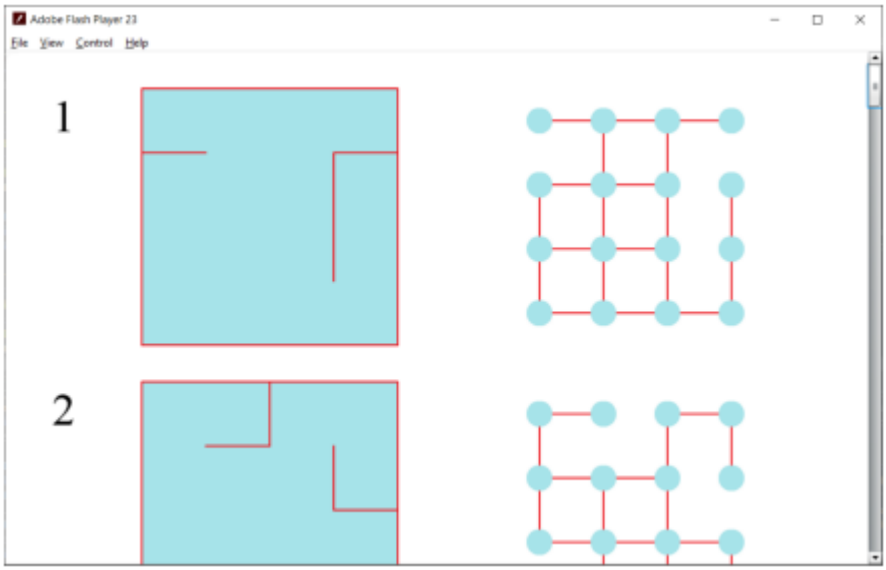


Figure 6. Algorithm Visualization Page User Interface Design

The number of candidate solutions from maze perfect that corresponds to the number of ways to choose $mn-1$ edge from the available $2mn-m-n$ edges is $C(mn-1,2mn-m-n)$. Then check whether the candidate solutions as many as $C(mn-1,2mn-m-n)$ are connected or not. If the solution produces a connected maze, then the resulting maze is a perfect maze. Using the same method, the non-perfect maze solution candidates are $C(mn-1+k,2mn-m-n)$ where k is the number of cycle bases. For example, the number of candidate solutions for a perfect 4×4 maze is $C(15,24)=1307504$, while the number of candidate solutions for a non-perfect maze 4×4 with 3 cycle bases is $C(18,24)=134596$. The number of solutions this will increase as the maze size increases. Taking into account the magnitude of this value, the maze list frame will only display the entire maze solution if the number of solutions is less than 20, while for solutions with a number of more than 20, the maze list page will only display a maximum of 20 other solutions.

The maze that has been generated on the algorithm visualization page can be saved by using the SAVE button. This storage feature is made to make it easier for users to look back at the previously generated maze. All information regarding the maze is stored in this maze repository. The information stored is the value of the parameters $m, n, psv, psh, pc, js, jeh,$

jev, jsh, jsv, and the string text field of the History Box and the string text field parameter. To display the same maze, information about the selected edge is also stored. The maze is saved to a file of type XML using a FileReference object.

D. CONCLUSION AND SUGGESTIONS

This study discusses the creation of a non-perfect maze using Prim's algorithm. An algorithm for biased non-perfect maze formation is designed by modifying Prim's algorithm and the use of the Fisher-Yates algorithm placed in modifying Prim's algorithm. This algorithm has a complexity of $O(|E|^2)$. Visualization applications designed using Adobe Animate software can visualize biased non-perfect maze. Visualization is done by displaying an image of the maze formed along with its corresponding grid graph in two-dimensional form. A maze that has been generated in the application can be saved in the form of a file with XML format and can also display files that have been previously saved.

The suggestions that can be given for this research are the need to increase the number of display solutions from the maze being sought while at the same time developing a non-perfect maze visualization for sizes more than 20×20 with a more diverse maze shape that displays more than 20 solutions. New search methods to generate non-perfect mazes can also be added so that it can be expected that future non-perfect maze searches will be able to go through the stages of finding perfect mazes first. The contribution of this research is that it can become a basis for expanding the dimensions of the study space from 2 dimensions to 3 dimensions.

REFERENCES

Baddeley, A. (2012). Working memory: theories, models, and controversies. *Annual Review of Psychology*, *63*(2012), 1–29. https://doi.org/10.1146/annurev-psych-120710-100422

Brooks, S. (2016). *Tradigital Animate CC: 12 Principles of Animation in Adobe Animate*. CRC Press.

Chatterjee, A., & Kiao, U. (2021). *Time Complexity Analysis*. OpenGenus.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT Press.

Dubey, P., & Sarita, K. (2016). Maze generation and solver. *International Journal of Scientific and Technical Advancements*, *2*(4), 139–142.

Fitzgerald, R. E., Isler, R., Rosenberg, E., Oettinger, R., & Bättig, K. (1985). Maze patrolling by rats with and without food reward. *Animal Learning & Behavior*, *13*(4), 451–462. https://doi.org/10.3758/BF03208022

Foulds, L. R. (2012). *Graph Theory Applications*. Springer Science & Business Media.

Galbiati, G. (2003). On finding cycle bases and fundamental cycle bases with a shortest maximal cycle. *Information Processing Letters*, *88*(4), 155–159. https://doi.org/10.1016/j.ipl.2003.07.003

Groner, L. (2016). *Learning JavaScript Data Structures and Algorithms*. Packt Publishing Ltd.

Hart, J. (2008). *The Art of the Storyboard Second Edition*. Elsevier.

Hendrawan, Y. F. (2018). A Maze Game on Android Using Growing Tree Method. *Journal of Physics: Conference Series*, Vol. 953, No. 1, 012148. https://doi.org/10.1088/1742-6596/953/1/012148/meta

Hoetama, D. O., Putri, F. P., & Winarno, P. M. (2018). Algoritma Fisher-Yates Shuffle dan flood fill sebagai maze generator pada game labirin. *Ultima Computing: Jurnal Sistem Komputer*, *10*(2), 59–64. https://doi.org/10.31937/sk.v10i2.1064

Hutama, D. R., Santosa, R. G., & Karel, J. (2014). Implementasi algoritma Prim sebagai creator jalur permainan maze. *Jurnal Informatika*, *9*(2), 147–155.

Ihsan, M., Suhaimi, D., Ramli, M., Yuni, S. M., & Maulidi, I. (2021). Non-perfect maze generation using Kruskal algorithm. *Jurnal Natural*, *21*(1), 35–45. https://doi.org/10.24815/jn.v21i1.18840

Jonasson, A., & Westerlind, S. (2016). *Genetic algorithms in mazes: a comparative study of the*

performance for solving mazes between genetic algorithms, BFS and DFS.

Kavitha, T., Liebchen, C., Mehlhorn, K., Michail, D., Rizzi, R., Ueckerdt, T., & Zweig, K. A. (2009). Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, *3*(4), 199–243. https://doi.org/10.1016/j.cosrev.2009.08.001

Kung, D. (2013). *Object-oriented Software Engineering*. McGraw-Hill Higher Education.

Miller, T. (2019). Explanation in artificial intelligence: insights from the social sciences. *Artificial Intelligence*, *267*(2019), 1–38. https://doi.org/10.1016/j.artint.2018.07.007

Nugroho, D. A., Harmastuti, H., & Uminingsih, U. (2017). Membangun game edukasi "mathematic maze" berbasis android untuk meningkatkan kemampuan berhitung pada anak sekolah dasar. *Jurnal Statistika Industri Dan Komputasi*, *2*(1), 67–77. https://doi.org/10.34151/statistika.v2i01.1101

Rosenzweig, G. (2013). *ActionScript 3.0 Game Programming University*. Pearson Education India.

Roy, S., Das, S. K., & Kamal, A. H. M. (2022). A multi-path based embedding scheme at perfect maze. *Indian Journal of Computer Science*, *7*(1). https://doi.org/10.17010/ijcs/2022/v7/i1/168954

Subaeki, B., & Ardiansyah, D. (2017). Implementasi algoritma Fisher-Yates Shuffle pada aplikasi multimedia interaktif untuk pembelajaran tenses bahasa inggris. *Infotronik: Jurnal Teknologi Informasi Dan Elektronika*, *2*(1), 67–74. https://doi.org/10.32897/infotronik.2017.2.1.31

Ullah, Z., Chen, X., Gou, S., Xu, Y., & Salam, M. (2022). FNUG: imperfect mazes traversal based on detecting and following the nearest-to-final-goal and unvisited gaps. *IEEE Robotics and Automation Letters*, *7*(2), 5175–5182. https://doi.org/10.1109/LRA.2022.3151393