MPLEMENTATION OF MULTI AGENT SYSTEM PATHFINDING ALGORITHM USING LIFELONG PLANNING A * ON THE LABIRIN GAME NPC

Danang Setyo Widodo[1],Galih Hermawan[2]

[1,2]Universitas Komputer Indonesia
Jl. Dipati Ukur No. 102-114 Bandung
E-mail : danang.wiodo65@gmail.com[1],galih.hermawan@email.unikom.ac.id[2]

ABSTRACT

Labirynth is a puzzle in the form of complex road branching and has many dead ends. The aim of the game is that players must find a way out of an entrance to one or more exits. Pathfinfing is artificial intelligence that uses the fastest path search algorithm. The application that can be done with pathfinding includes searching the path in a game and searching the path on a map. One algorithm that can be used is Lifelong Planning A *, an incremental version of A * that can adapt to changes in the graph without recalculating the entire graph, by updating the g-value (distance from the start) from the previous search during the current search to fix it when needed. Lifelong Planning Algorithm A * aims to determine the accuracy of the success rates of many agents in the pathfinding algorithm in determining the steps of the labyrinthine Agent. The characteristics of the game that will be built have the use of obstruction features to block labyrinthine agents from chasing players, and several labyrinthine agents that move simultaneously (multi agent) .. In the testing process of 30 repetitions that have been prepared by the npc Agent able to find 25 paths on 8 Npc agents and have an average accuracy of 91.7 percent. And has a speed value of 166,999 ms in the length of execution time with 15x15 orders and 8 Agents. After testing the system with the Black Box method and calculating the level of accuracy, it can be concluded that the program is functioning correctly.

Kata Kunci: Lifelong Planning a*, multi Agent, pathfinding, labirynth, Game.

1. INTRODUCTION

Maze game is a game looking for paths where the labyrinthine path gets a lot of predetermined obstacles to reach the destination. At this time there are still many players who are confused in finding a way out, how to get a bonus value and can raise it to the next step. So in achieving the goal needed a solution, one solution that can be used is the backtracking algorithm. Backtracking algorithm is an algorithm that can be used to create a labyrinthine application where the algorithm works is to find the right solution to determine the right path to achieve the intended goal [2]. But in its implementation this algorithm only uses the Single Agent System. So the results are unknown if there are several agents tested.

There are several studies on the application of AI in Pathfinding NPC in labyrinth games such as those conducted by Ilham Ramadhan in 2017, said the Jump Point Search algorithm can be applied to optimize the search for the fastest NPC dynamically [1]. The problem of the previous research is that there has been no testing of 2 or more NPCs.

The second is the research conducted by Sri Anggraini Surianto about the Implementation of Iterative Deepening A * (IDA *) Algorithms with Stochastic Node Caching (SNC) for Enemy Pathfinding in Maze Games, Position of players, enemies and number of walls / barriers also affect the number of node expansion. The farther the distance between the player and the enemy, the more nodes will be expanded. The more number of walls / barriers in the map, the more nodes that are expanded. So that it causes a decrease in process speed if it is carried out on 2 or more Agents [11].

Pathfinding is one of the most basic problems of artificial intelligence (AI) in the game. Poor pathfinding can make the characters in the game look brainless. Effective handling of pathfinding problems can make the game more fun and provide a player with a deep playing experience [3].

One of the pathfinding algorithms that can be used in finding and recognizing this pathway is the Lifelong Planning A * (LPA *) algorithm. The LPA * algorithm is a development of the A * algorithm developed by Hang Ma and Jiaoyang Li in 2017. These algorithms use hundreds of agents and assignments [2].

1.1 GOALS AND PURPOSE

Based on the existing problems, the purpose of this study is to implement the Lifelong Planning A * algorithm on the labyrinthine NPC simultaneously in the search for the fastest path. Then the objective to be achieved from this study is to determine the accuracy of the success rates of many agents in the pathfinding algorithm in determining the steps of the labyrinthine NPC by using Lifelong Planning A *.

1.2 Research Method

Research methods are a way to achieve a goal in a study. For this study the authors used descriptive research methods. Descriptive research method is a study that aims to describe a phenomenon that occurs today by using scientific procedures to answer the problem actually.

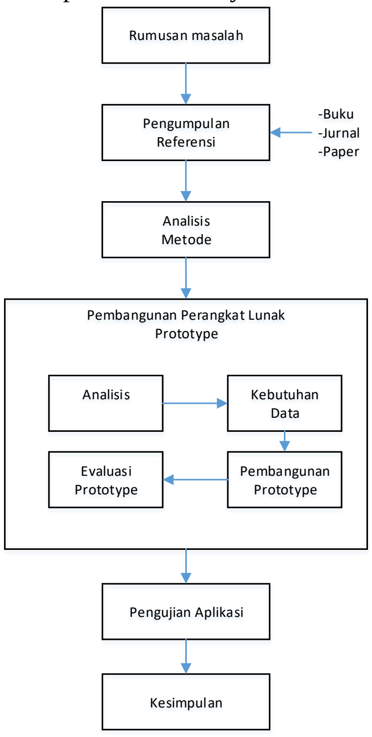


Figure 1. Research Method

In the flow of research methods carried out are the stages that will be carried out :

1. Reference Collection

At this stage collect journals and papers related to this research and search for books to explore the material about pathfinding algorithms, A * algorithms, and Lifelong Planning A * algorithms.

2. Method Analyst

then do the analysis. Analyzing the Lifelong Planning Algorithm A * method.

3. Software Development

The model of the software development process used is the Prototype model. The steps in the Prototype model are.

a. Analisyst

Analyze the things needed to identify Pathfinding steps. In this stage a method analysis is carried out and also the collection of information and data needs by collecting journals, papers, and information relating to the research that will be conducted.

b. Data needs

The data collection stage is in the form of the number of wall barriers, the number of npc, and the destination point.

c. Prototype Development

the implementation phase of the analysis process and system data requirements that have been carried out.

d. Prototoype Evaluatioan

The testing phase of the prototype application that was built.

4. Aplication Testing

Application Testing is testing the methods that have been implemented into the program.

5. Research Result

The results of the study are the final stage in this study. The tests performed produce multi agent NPC steps and good accuracy or not.

2. THE CONTENT OF RESEARCH

2.1 Game labirynth

In general, mazes are made for entertainment purposes. In real life, labyrinths can be found in small road arrangements or alleys in residential areas. It is very difficult if someone who is foreign to the area to find a way. Labirin is a puzzle in the form of complex road branching and has many dead ends. The aim of the game is that players must find a way out of an entrance to one or more exits. It can also be the condition of the player winning, that is when he reaches a point or destination in the labyrinth.

In everyday life there are several labyrinth applications that can be found especially in the game industry, because the labyrinth challenges players to find a way out from a predetermined point. Besides that in some countries a maze of human size is created and challenges people to enter it, as one of the attractions to attract tourists [7].

The following is a picture of one of the maze games:

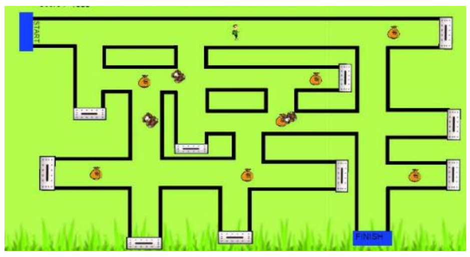


Figure 2. Game labirynth

2.2 Pathfinding

Pathfinding is one application that is handled by artificial intelligence using the search algorithm [3].

The application that can be done with pathfinding is to search the road on a map and search for routes or paths in a game. The pathfinding algorithm that is used must be able to recognize the path and all map elements that cannot be skipped. A good pathfinding algorithm can be used to detect a number of obstacles that exist on the road and find a path to avoid them, so that the path taken is shorter than it should be if it doesn't use the pathfinding algorithm.

See the illustration in Figure 3

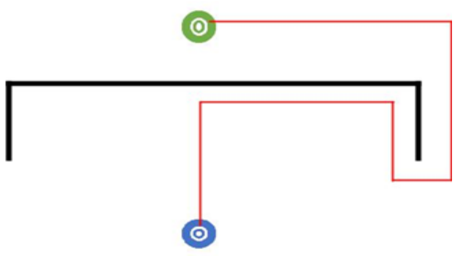


Figure 3. Routing without pathfinding

In Figure 3. from the beginning that is green towards the end, without the pathfinding algorithm, npc will only check for obstacles or obstacles from the surrounding environment. The unit will move forward to reach the destination, only after approaching the obstacle, then walking to the right on the way.

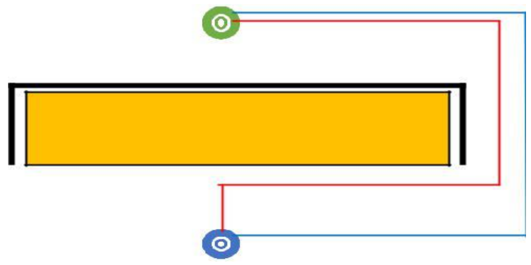


Figure 4. Determining routes with pathfinding

Next, determining the path with the pathfinding algorithm in Figure 4. will plan the way forward to find shorter paths and avoid obstacles or obstacles symbolized by the light blue line to reach the destination, without walking into an obstacle or obstacle. therefore the function of the pathfinding algorithm is important to solve some problems in determining paths.

2.3 Algorithm *Lifelong Plannning A** (LPA*)

Lifelong Planning A * is an incremental version of A *, which can adapt to changes in the graph without recalculating the entire graph, by updating the g-value (distance from the start) from the previous search during the current search to fix it when needed. Lifelong Planning A * uses heuristics, which are the lower bound for the path cost of the node given to the destination. The first search is the same as version A * which breaks the connection between nodes with the same f value and supports a smaller g value. Using the calculation algorithm as follows.

$$g*(s) = \begin{cases} 0 & if\ s = {}^{s}start, \\ \min_{s' \in \mathrm{pred}(s)}\left(g(s') + c(s',s)\right) & otherwise. \end{cases}$$

Dimana :

- S = node
- predecessors from s = $pred\ (s) \in s$
- Cost travel from point s to Point s'=$0 < c(s,s') \leq \infty$
- Start Point = ${}^{s}start \in S$
- Start distance= Length from closest **S**$_{start}$ to **S**
- g(s) = Estimation start distance g*(s)

The following is an algorithm from *LPA** :

```
    procedure Kalkulasikey(s)
    return [min($g(s)$, $rhs(s)$) + $h(s)$; min($g(s)$, $rhs(s)$)];
    procedure inisialisasi()
    $U = \emptyset$;
    for all $s \in S$ $rhs(s) = g(s) = \infty$;
    $rhs(s_{start}) = 0$;
    U.tambah($s_{start}$, [$h(s_{start})$; 0]);
    procedure UpdateSimpul(u)
    if ($u \neq s_{start}$) $rhs(u) = \min_{s' \in pred(u)}(g(s') + c(s', u))$;
    if ($u \in U$ ) U.hapus($u$);
    if ($g(u) \neq rhs(u)$) U.tambah($u$, Kalkulasikey($u$));
    procedure Hitungjalurterpendek ()
    while (U.TopKey() $\dot{<}$ Kalkulasikey($s_{goal}$) OR $rhs(s_{goal}) =$
    $u$=U.Pop();
    if ($g(u) > rhs(u)$)
    $g(u)=rhs(u)$;
    for all $s \in succ(u)$ UpdateSimpul($s$);
        else
    $g(u)=\infty$;
        for all $s \in succ(u) \cup \{u\}$ UpdateSimpul($s$);
    procedure Main()
    Inisialisasi();
    Hitungjalurterpendek();
        Tunggu cost berubah;
        for semua titik ($u$, $v$) dengan cost titik yang berubah
        Update cost titik $c(u,v)$;
        UpdateSimpul($v$);
```

Figure 5. Algorithm *Lifelong Planning* A*

2.4. System Analyst

The system that was built was the Pathfinding NPC system in the labyrinth game. This system calculates the steps of several NPCs from the initial location to the destination. The following is a Figure of the system to be built:

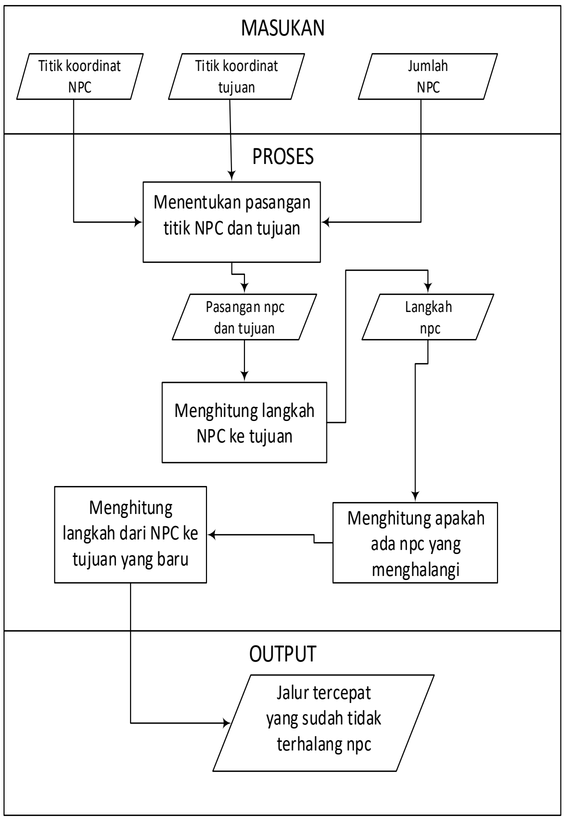

MASUKAN
Titik koordinat NPC
Titik koordinat tujuan
Jumlah NPC
PROSES
Menentukan pasangan titik NPC dan tujuan
Pasangan npc dan tujuan
Langkah npc
Menghitung langkah NPC ke tujuan
Menghitung langkah dari NPC ke tujuan yang baru
Menghitung apakah ada npc yang menghalangi
OUTPUT
Jalur tercepat yang sudah tidak terhalang npc

Figure 6. General system figure

2.5 Input Analyst

Input analysis explains the input data that will be needed in the application of the LPA * algorithm. Input analysis is needed in the LPA algorithm * namely the Agent's initial position, Agent's final position, obstacle position and search depth. Before calculating using the LPA algorithm * the actual cost will be calculated on the path itself. The map used in this simulation with a pixel scale measuring 400 pixels wide and 600 pixels high. One layer is stored in a grid (a small grid) that is dynamic where the user can determine the number of grids. Each grid has a size of 20 pixels wide and 20 pixels high, sample folders with a width of 20 pixels and a height of 20 pixels can be seen in Figure 7. The following:

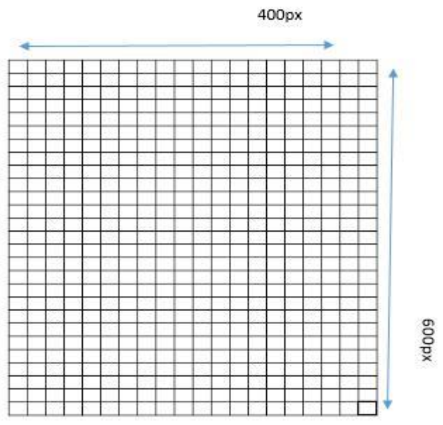


Figure 7. Screen size description

To simplify the calculation, there are several values that will be simplified. First the node value that was 20x20 pixels was changed to 1x1, meaning that node distance 1 to another node is 1. So each node has the same cost, namely 1 horizontally and vertically. Movements can only be diagonal and vertical.

2.6 Sample Case

To better understand the Lifelong Planning A * algorithm, it can be seen from the example as in Figure 8. It is assumed that in the simulation there are 2 Agents (blue and red) who have their own goals, besides that there is also a barrier wall.

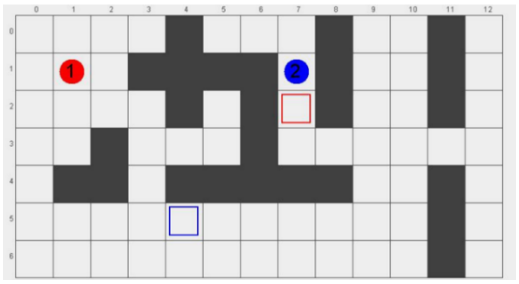


Figure 8. Initial conditions for Search LPA *

The first process is to calculate the heuristics from the Agent to the goal, find the shortest path from the initial node, node storage uses three dimensions of space-time (x, y, time). After performing the initial calculation, and the initial path has been found, each agent moves to the destination of the closest node, such as Figure 3.4 below:

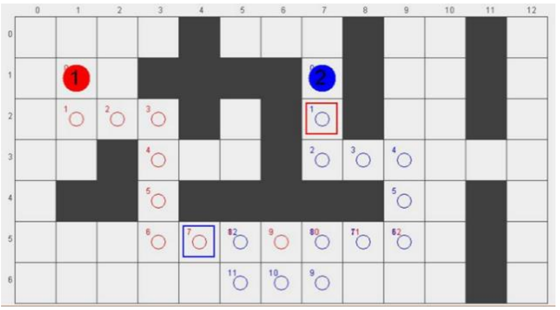
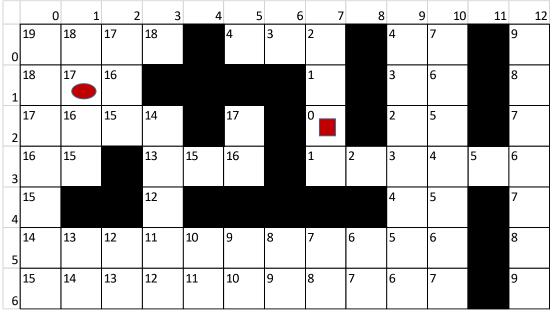


Figure 9. Initial condition with the path to be passed

The following is the heuristic of Agent 1 to goal 1 and Agent 2 to goal 2 using manhattan distance



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19	18	17	18	⬛	4	3	2	⬛	4	7	⬛	9
1	18	17 🔴	16	⬛	⬛	⬛	⬛	1	⬛	3	6	⬛	8
2	17	16	15	14	⬛	17	⬛	0 🟥	⬛	2	5	⬛	7
3	16	15	⬛	13	15	16	⬛	1	2	3	4	5	6
4	15	⬛	⬛	12	⬛	⬛	⬛	⬛	⬛	4	5	⬛	7
5	14	13	12	11	10	9	8	7	6	5	6	⬛	8
6	15	14	13	12	11	10	9	8	7	6	7	⬛	9

Figure 10. heuristic *Agent* 1

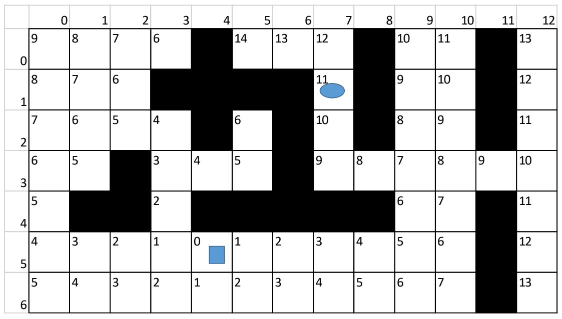


Figure 11. heuristic *Agent* 2

In this iteration the LPA * algorithm adds to the currentWindow variable to + 1 currentWindow and currentTime becomes the currentTime +1, after that the next step is to check again, whether there are units that must be recalculated or not, the unit to be checked is taken randomly based on currentWindow.

2.7 Testing

System testing from this research is conducted in order to find out how the application that has been made can be run accordingly or not. The testing uses three techniques, namely functional testing, performance testing and accuracy testing.

2.7.1 Functional Testing

Functional testing is done to check that the functionality that has been made has been running or not, the following is a scenario of functional testing.

Table 1. Test results and cases (normal data)

Actio n	**What will be expected**	**Result**
Enter starting point	Displays the starting point on the labyrinth	Be accepted
Enter the destination point	Displays the destination point on the labyrinth	Be accepted
Enter barrier data	Displays a barrier on the labyrinth	Be accepted
Press the step button	Displays the first route of each NPC on the labyrinth	Be accepted
Press the Animation button	Showing each NPC moves from the starting point	Be accepted
Press the Stop button	Display each NPC in the last position	Be accepted
Pressing the reset button	Displays folders with no obstacles and NPCs	Be accepted
Enter data row values Example: 13	Can be filled with numbers to enter row value data	Be accepted

Based on the testing of each functional using the black box method that has been made, it is concluded that the system created has worked in accordance with the expected functional.

2.7.2 Performance Testing

The performance to be tested on the application that has been made is the travel time from the initial position to the final position of each unit and the number of vertices examined in each cycle.

The performance tested on the application that has been made is the node examined and the time taken resulting from the implementation of a minimum order of 5 x 5 without a barrier with the number of agents from 2 to 8.

Table 2. Node expansion test results

Ordo \ Total	2	4	6	8
5X5	96 ms	357 ms	771 ms	892ms
10X10	309 ms	870 ms	1074 ms	1798 ms
15X15	622 ms	1154	1793 ms	2486 ms

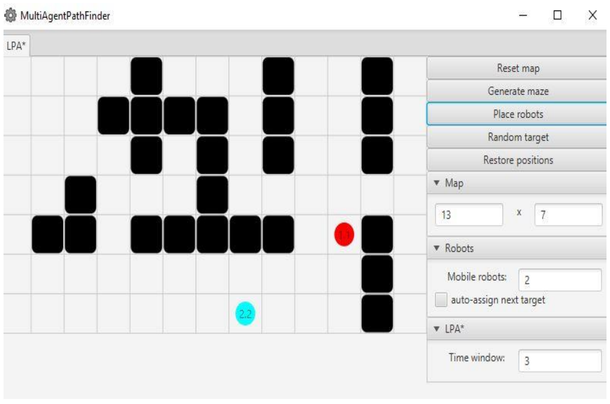
The results of testing the execution time on orders 5x5, 10x10 and 15x15 without a barrier with the number of agents from 2 - 8 can be seen in Table 3.

Table 3. Results of execution time

Ordo \ jumlah	2	4	6	8
5X5	4.33ms	8.81ms	15.06ms	55.46ms
10X10	38.68 ms	60.16ms	90.99ms	107.46ms
15X15	60.25 ms	93.12ms	141.77ms	166.99ms

2.7.3 Accuracy Testing

After performance testing, it will be done with accuracy testing using the LPA * and A * algorithm 30 times by using a barrier and the target and the Agent placed randomly. example of the barrier as follows:



13 x 7

Mobile robots: 2

Time window: 3

Figure 12. example of accuracy test

The results of testing the accuracy of Agents 2 - 8 by testing the number that found the path, the amount taken and accuracy can be seen in Table 4 and Table 5.

Table 4 . accuracy test LPA*

Total Agent	*Agent on track*	Node taken	Accuracy(%)
2	30	10	100,0
4	29	11	96,7
6	26	13	86,7
8	25	15	83,3
Average accuracy			91,7

Table 5 . accuracy test A*

Total *Agent*	*Agent on track*	Node taken	Accuracy(%)
2	30	17	100,0
4	25	18	83,3
6	23	19	76,7
8	19	22	63,3
Average accuracy			80,8

3. CONCLUSIONS AND SUGGESTIONS

3.1 Conclusion

Based on the results of analysis, implementation and testing, conclusions can be taken as follows:

1. The Lifelong Planning A * algorithm can be used to pathfinding many npc Agents in a maze game.
2. Based on the accuracy testing carried out using 8 Agents and as many as 30 repetitions it is known that the level of accuracy in the success of the NPC Agent that found its path is 25 Agents with a total of 15 nodes taken. The level of accuracy has a percentage of 91.7%. And it has a velocity value of 166,999 ms in the length of execution time with a 15x15 order and 8 agents so that it can be said that the level of accuracy in the application of the Lifelong Planning A * algorithm is good.

3.2 Suggestion

The advice that can be given for further development, namely for the application of the A * lifelong algorithm, can be tried by adding the number of goals, using the latest variant of the latest shortest path search algorithm, and also increasing the number of behaviors of NPCs thereby reducing failure in determining path searches.

BIIBLIOGRAPHY

[1] Ramadhan, Ilham. 2017. "Implementasi Algoritma Jump Point Search Pada NPC Musuh Untuk Mengejar Pemain Dalam *Game* Labirin (Skripsi)". Bandung: Universitas Komputer Indonesia

[2] Henry, Samuel. 2010. Cerdas dengan *Game*. Jakarta: PT Gramedia Pustaka Utama.

[3] Asmiatun, Siti. 2017. Belajar Membuat *Game* 2D dan 3D Menggunakan Unit. Yogyakarta : Penerbit Budi Utama.

[4] Teresa,Dillon. 2004. Adventure *Games* for Learning and Storytelling. A Futurelab prototype context paper: Adventure Author, FutureLab Report.

[5] Andang, Ismail. 2009. Education *Games* (Menjadi cerdas dan ceria dengan permainan edukatif). Yogyakarta : Pilar Media

[6] Thomas. 2006. Genre and *game* studies : Toward a critical approach to video *game* genres. Simulation & Gaming, Vol. 37, University of Melbourne

[7] Imam Ahmad, W. W. 2017. Penerapan Algoritma A Star (A*) pada *Game* Petualangan Labirin Berbasis Android. Jurnal Ilmu Komputer Dan Informatika.

[8] Kusumadewi, Sri. 2002. Artificial Intelligence . Yogyakarta, Graha Ilmu.

[9] Suyanto. 2007. Artificial Intelligence. Bandung: Informatika.

[10] J.E. Kendall dan K.E. Kendall. 2006. *Systems* Analysis and Design Berbasis Android. Bandung : Informatika.

[11] Surianto, Sri Anggraini. 2014. "Implementasi Algoritma Iterative Deepening A* (IDA*) Dengan Stochastic Node Caching (SNC) Untuk *Pathfinding* Musuh Pada *Game* Labirin (Skripsi)". Bandung: Universitas Komputer Indonesia