IMPLEMENTATION OF JUMP POINT SEARCH AND FINITE STATE MACHINE ALGORITHM AS THE SHORTEST PATHWAY DETERMINANT AND BEHAVIOR IN ARCADE GAME

Fajar Ramadan

Informaticts Engineering Course
Faculty of Engineering and Computer Science of the University of Computer of Indonesia
Jl. Dipati Ukur 112-114 Bandung
E-mail : alphaenciel@gmail.com

ABSTRACT

Game Arcade to be built is a game chase while fighting. A similar game is the "Tom & Jerry in House Trap" game available on PS One that takes the background of the story as in the "Tom & Jerry" cartoon. So the enemy NPC requires an algorithmic method that can help in finding the path for movement and also to determine the action decisions that need to be done.

The algorithm to be used as the shortest path determinant is the Jump Point Search algorithm. This algorithm is the result of the development of the A * algorithm by adding the jump points function when doing path-finding. While the method to be used as a determinant of behavior is the Finite State Machine method. Finite State Machines (FSM) is a control system design method that describes the behavior or working principle of a system by using three things, namely: state, event and action.

The result of the research shows that the algorithm with the simulation program is known that the Jump Point Search algorithm has good performance, it can be seen from the relatively small memory usage that is only 33.1% of the total memory usage. Also from the results of testing methods Finite State Machine can be seen that the enemy NPC already has a behavior that can change according to existing conditions.

Keywords: artificial intelligence, arcade games, pathfinding algorithms, behavior determinants

1. INTRODUCTION

Arcade Game is one type of game that relies on the dexterity of the player in holding the control [1]. Games of this type usually have a short level, easy control, and have a simple gameplay. Arcade Games to be built is a game chase while fighting. In game fights and chases would require a good hurdle to make the game interesting. In addition to the complicated romp form, NPC (Non Playable Character) enemies must be able to react according to the circumstances or conditions that occur in the game.

A similar game is the "Tom & Jerry in House Trap" game that is on PS One that takes the background of the story as in the "Tom & Jerry" cartoon [2]. Where cat and mouse characters chase each other in the house and can attack each other until the health point of his opponent runs out. Game "Tom & Jerry in House Trap" is a blend of types of game labyrinth and fighting games. In the game the movement of NPC in the pursuit of the player is still less precise, because the NPC will only pursue the player if the player is visible (in the range of vision NPC) and if the player is not visible then the NPC will move randomly to another room. In addition, NPC behavior is still monotonous, sometimes when the NPC already has a weapon does not directly attack the player in front of him but instead spin to look for other weapons or look for traps.

The enemy NPC must be able to recognize the arena well so that it can search for weapons quickly, set traps, run away from players when not possessing weapons, and chase players when possessing weapons. So that enemy NPC requires an algorithm method that can help in finding the path for movement and also to determine the action decisions that need to be done.

Jump Point Search is a path search algorithm to a target where the algorithm moves very quickly on uniform-cost grid maps by jumping over many places it deems necessary to skip [3]. This algorithm is the result of the development of the A * algorithm by adding the jump points function when doing path-finding. Jump Point Search can be described in two simple pruning rules that are applied recursively during search, specific rules for straight steps and for diagonal steps. An important key in both cases is to trim the near neighbor sets around the node by trying to prove that there is an optimum path from the current parent node to each neighbor. Based on previous research [3] it is known that the Jump Point Search algorithm applied to games with labyrinth types such as Pacman is able to produce the average number of visited nodes best compared to A *, IDA * and PEA * algorithms. With the average visited node for the Jump Point Search 11.7 algorithm, the algorithm IDA * 15.57, algorithm A * 27.8, and PEA

algorithm * 28.63. Jump Point Search algorithm is a simple but effective algorithm, optimal without using much memory, and very fast performance.

Finite State Machines (FSM) is a control system design method that describes the behavior or working principle of a system by using three things, namely: state, event and action [4]. The system may switch or transition to another state if it obtains certain inputs or events, whether from external devices or components in the system itself (eg timer interruptions). The transition of this state is generally accompanied by action by the system in response to input. The action can be either a simple action or involving a relatively complex set of processes. Finite State Machine can be used on the NPC as a decision maker so that the NPC can work with the maximum. Based on previous research [4] it is known that the Finite State Machine method applied to games with Role-Playing Game (RPG) The Relationship simulation can produce different NPC responses to match the conditions occurring in the game and the more responses or behaviors Which is applied to the NPC then the more interesting the game because it can produce a more interesting storyline.

Therefore, in this research Jump Point Search algorithm will be used to be implemented on enemy NPC as AI which serves to determine the path of movement and Finite State Machine as decision maker.

2. RESEARCH CONTENTS

Here is an analysis of Jump Point Search algorithm implementation and Finite State Machine method in arcade game.

2.1. Arcade Game

Arcade Game is a computer game genre that relies on the dexterity of its players in holding control. Characteristics of arcade games is to have a short level, easy control, and the level of difficulty that increases quickly. This type of game is designed to provoke adrenaline players and does not require a good storyline. Games like this require a relatively short learning time. The nature of arcade games is real-time [1].

Arcade Game has many types of games, one of which is a game of chase. In this case a similar game is a Tom & Jerry House Trap game on PS One [2].



Gambar 2.1 Game Tom & Jerry House Trap

Game Tom & Jerry House This trap is a game chase while fighting that uses the background of the house as arena. In this game the player must take weapons first to be able to attack the enemy or can also install a trap that is scattered around the arena. The game will end if one of the players runs out of health points.

2.2. Jump Point Search Algorithm

Path-finding in a uniform-cost grid environment is a problem that is often encountered in the field of robotics and video games. There are several algorithms devoted to path-finding, for example in the HPA * algorithm. But this algorithm can not move toward the destination if the path that has been obtained is closed by a dynamic obstacle and the start and goal are on a different cluster [3]. Jump Point Search algorithm or abbreviated JPS algorithm is one of the existing path-finding algorithms. The algorithm was published in 2011 by D. Harabor and A. Grastien [5]. This algorithm is one of the development of A * algorithm that aims to reduce memory usage and make path search faster.

Notations and terminology used in the JPS algorithm are as follows

1. p (x) is the start or the initial node
2. z is the goal or destination node
3. neighbors (x) are nodes around node x
4. y can be achieved from x by passing through several nodes in a given direction, y can be expressed as:

$$y = x + k\vec{d} \quad (1)$$

With:

X = active node

K = number of step units from x to y

d = intended direction

Y = successful candidate node x

In JPS there is a new strategy that can accelerate the search path by selectively expanding to certain nodes on the grid map called a jump point.

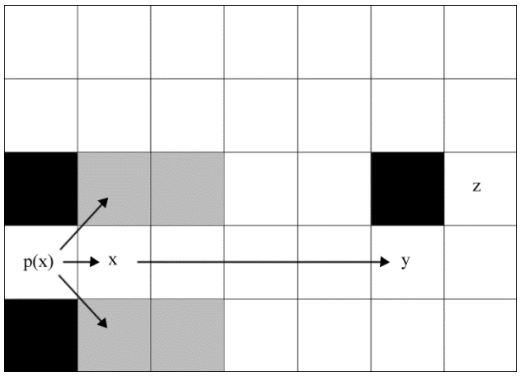


Figure 2.2 Jumping straight on the JPS

In Figure 2.2 it can be seen that node x expands with parent p (x). The destination direction of the expansion node x is straight to right or horizontal to the right. Unlike the A * algorithm that includes all the nodes that exist from x to y, then on JPS this is not done. The JPS will immediately do a search to the right until it finds a node like y. If a node such as y is found, mark the node as a successor of x and assign g (y) value to the node.

$$g(y) = g(x) + dist(x,y) \qquad (2)$$

With:
G (x) = cost of node x
Dist (x, y) = distance from node x to y
G (y) = cost from initial node to y

$$dist(x,y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (3)$$

With:
$dist(x,y)$ = distance from node x to y
x_2 = the position of the successor node x
x_1 = the position of the initial x node
y_2 = the position of successor node y
y_1 = the position of the initial node y

The calculation of the heuristic value at each node in the JPS is also required to calculate the distance of the node with the destination node. Calculation of heuristic value for nth node in jump point can use Manhattan Distance calculation which the formula is as follows:

h(y) =(abs|currentX–targetX|+abs|currentY–targetY|) (4)

With :
h(y) = the minimum cost estimate of the node y to the destination node
currentX = position of the initial X node
currentY = position of the initial Y node
targetX = position X destination node
targetY = position Y destination n when y can be achieved by taking k

Having found the cost value of a node pursued from the beginning to y in the form of g (y) and the heuristic value from y to the destination, an equation that can determine the true value of the node in the jump point, therefore the equation in A * is f(x) = g (x) + h (x) is still used in the JPS. The equation in the JPS is as follows:

$$f(y) = g(y) + h(y) \qquad (5)$$

With:
f(y) = the true value of node y
g(y) = cost from initial node to y
h(y) = the minimum cost estimate of the node y to the destination node

If a search for a specified direction finds a barrier, then it is assumed the search for that direction does not yield anything. When expanding from x, it can be seen that there is an evaluation of other nodes that are gray. In the process of jumping is also required a rule that can recognize whether a nodes need to be seen or can be ignored. This rule is called pruning.

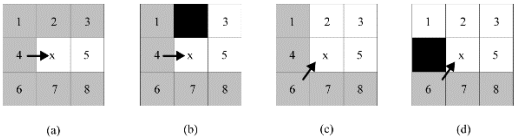


Figure 2.3 Pruning rules on JPS algorithm

Pruning has the meaning of pruning which in this case means unnecessary nodes will be ignored. The nodes can be ignored by looking at the direction used in the expansion. In pruning there are two terms used, namely natural neighbors and forced neighbors. Natural neighbors are nodes that can not be ignored, while forced neighbors are nodes that are forced not to be ignored.

There are several circumstances to be observed, depending on how the expansion from p (x) to x is done, whether perpendicularly whether it is horizontal and vertical or diagonal.

Straight Pruning: Crop all nodes n within neighbors (x), in addition to designated nodes using horizontally or vertically expansion direction. It can be seen in Figure 2.3 (a) that the gray nodes are neglected nodes because they can not be reached using the expansion direction used ie horizontally to the right. In Figure 2.3 (a) only the node with the number 5 is taken, because it can be achieved with a usable expansion direction.

Diagonal Pruning: Prune all nodes n within neighbors (x), in addition to designable nodes using diagonally used expansion directions. It can be seen in Figure 2.3 (c) that the gray nodes are neglected nodes because they can not be achieved using the expansion direction used ie the diagonal to the right, whereas the nodes with the numbers 2 and 5 are not gray because they are considered to be Can not be achieved from p (x). Therefore nodes that are not neglected in Figure 2.3 (c) are only nodes with numbers 2, 3 and 5 only.

The resulting nodes of the pruning, whether straight or diagonal, are called natural neighbors while the gray nodes in Figure 2.3 are called non-natural neighbors. When neighbors (x) have nodes that are an obstacle, then the result of pruning will result in forced neighbors. Forced neighbors can be recognized if n is not a natural neighbors of x passable

and x has inaccessible neighbors nodes that are close to n.

In Figure 2.3 (b) it can be seen that the node with number 3 is forced neighbors in straight pruning because there are non-viable nodes close to node number 3, while in figure 2.6 (d) n = 1 is a forced neighbor on pruning diagonal.

The JPS algorithm has similarities to the A * algorithm, where the nodes to be used in the path determination are stored in memory. But the difference is that the JPS algorithm does not store all the nodes that are the path to the destination. In the previous A * algorithm explanation has been explained about how to store nodes in the A * algorithm in the form of open and closed list. In JPS the storage nodes on the closed list are omitted. This algorithm will only store some nodes that are inserted into the open list.

Open list in JPS is a successor node or node y of node x which is called a jump point stored into memory. A node y can be entered into an open list and called as a jump point if it meets the following conditions:

1. The y node is the goal node.
2. The node y has at least one forced neighbors.
3. On a diagonal search, the search horizontally or vertically has a node that has condition 1 or condition 2.

For more details on condition 3 can be seen in Figure 2.4, where x expands diagonally, in the expansion found node y is entered into the open list due to the horizontal expansion to the right on the node y there is a goal.

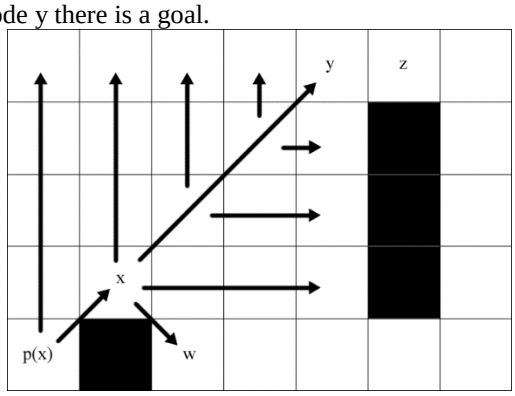


Figure 2.4 Jumping diagonally on the JPS

In Figure 2.4 it can be seen that x expands diagonally. At the beginning the expansion is done vertically or horizontally to node x where w is forced neighbors because it can not be achieved by expansion p (x). On each vertical or horizontal expansion each node visited will apply the pruning rule. If the vertical or horizontal expansion is not found nodes that can be entered into the open list or encounter a barrier, then continue the expansion diagonally. It can be seen in the figure that expansion stops at node y due to horizontal expansion found node z or destination node.

2.3. Finite State Machine Method

Finite State Machines (FSM) is a control system design method that describes the behavior or working principle of a system by using three things, namely: state, event and action [4]. The system may switch or transition to another state if it obtains certain inputs or events, whether from external devices or components in the system itself (eg timer interruptions). The transition of this state is generally accompanied by action by the system in response to input. The action can be either a simple action or involving a relatively complex set of processes.

Figure 2.3 shows the FSM with two states and two inputs and four different outputs as shown in the figure, when the system starts up, the system will transition to State0, in which case the system will generate Action1 if Event0 occurs, whereas Event1 Then Action2 will be executed then the system then transition to State1 state and so on.

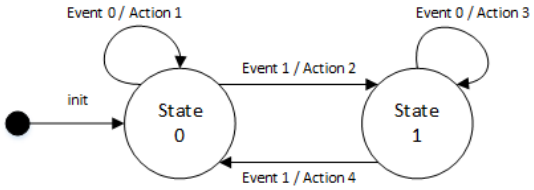


Figure 2.5 Simple state diagram

Formally FSM is expressed by 5 tuples or M = (Q, Σ, δ, S, F), where:

Q = the set of states / positions
Σ = set of input / input / alphabet symbols
Δ = transition function
S = initial state / initial state, S Q
F = set of final states, F Q

FSM consists of two types, non-output and FSM does not output. The non-output FSM is used for language recognition in the computer, with input entered to be obtained whether the input is known by the computer language or not. One of the uses of FSM not output is the compiler program, which is a program to check whether the user's command is correct or false. As for FSM output used to design the machine or system.

3. Testing

Here is a test on Jump Point Search algorithm implementation and Finite State Machine method in arcade game.

3.1. Jump Point Search Algorithm Testing

This test is performed to measure the performance of memory usage in Jump Point Search algorithm implementation in arcade game. Testing is done by using simulation application where the calculation used heuristic Manhattan Distance, Euclidean Distance, and Chebyshev Distance. The following test results of memory performance in Jump Point Search algorithm implementation.

Table 3.1 JPS memory performance test algorithm

Calculation	Average app memory (kb)	Average memory of JPS algorithm (kb)	Percentage (%)
Manhattan Distance	329	102	31
Euclidean Distance	329	109	33.1
Chebyshev Distance	329	116	35.3
Average memory performance (%)			33.1

From the tests that have been done can be seen that the appropriate Jump Point Search algorithm used to find the shortest path to the target. This can be proven from the number of nodes in the open list having nodes forming a path to the target. In calculating the heuristic value in table 3.1 it is known that the memory usage of the three heuristic calculations used is relatively small and the least is the calculation of Manhattan Distance. Based on Table 3.1 it is known that memory usage for Jump Point Search algorithm is only 33.1% of total memory usage overall. So the performance of memory usage on the application of Jump Point Search algorithm can be said good.

3.2. Finite State Machine Method Testing

Finite State Machine method testing is done by adjusting the applied pseudo-code application with the results contained in the arcade game. Testing is done into three parts, namely FSM On Guard, FSM Attacking Players, and FSM Berserk. Here is the test results that have been done.

Table 3.2 FSM On Guard Testing

Cases and Test Results			
Condition	**Expected output**	**Observation result**	**Conclusion**
Enemy's Health Point <= 50	The enemy moves to take potion	The enemy moves to take potion	Succeed
Weapon Distance <Trap Distance	The enemy moves to take weapons	The enemy moves to take weapons	Succeed
Distance Trap <Weapon Distance	The enemy moves to set a trap	The enemy moves to set a trap	Succeed
Enemy's Health Point <= 20	The speed of enemy movement increases	The speed of enemy movement increases	Succeed
The enemy holds weapons	The enemy chasing players	The enemy chasing players	Succeed
Enemy's Health Point = 0	The enemy is dead and the game is over	The enemy is dead and the game is over	Succeed

Table 3.3 FSM Attack Players Testing

Cases and Test Results			
Condition	**Expected output**	**Observation result**	**Conclusion**
The enemy holds weapons	The enemy chasing players	The enemy chasing players	Succeed
Weapon's durability > 0	The enemy attacked the players	The enemy attacked the players	Succeed
Enemy's Health Point <= 20	The enemy moves to take potion	The enemy moves to take potion	Succeed

Table 3.4 FSM Berserk Testing

Cases and Test Results			
Condition	**Expected output**	**Observation result**	**Conclusion**
Enemy's Health Point <= 20	The speed of enemy movement increases	The speed of enemy movement increases	Succeed
The enemy holds weapons and weapon's durability > 0	The enemy attacked the players	The enemy attacked the players	Succeed
Enemy's Health Point < 20	The enemy moves to take potion	The enemy moves to take potion	Succeed

Based on the test results in tables 3.2, 3.3, and 3.4 it can be seen that the Finite State Machine method has been successfully implemented on the

enemy NPC so that enemy NPC can act in accordance with existing conditions.

4. CONCLUSIONS and SUGGESTIONS

4.1. Conclusions

Based on the results of the analysis, implementation and testing, the following conclusions can be drawn:

1. Jump Point Search algorithm can be implemented as the shortest path locator in arcade game. Based on the test using the three heuristic calculations performed it is known that the memory usage in each search path is relatively small. The use of memory for the Jump Point Search algorithm is only 33.1% of total memory usage in arcade games. So it can be said that the performance of memory usage on the application of Jump Point Search algorithm is good.
2. Finite State Machine method can be implemented as a determinant of behaviour in arcade game. And also from the testing results of Finite State Machine method, it can be seen that the enemy NPC behaviour can change according to the condition.

4.2. Suggestions

Suggestions that can be given as input and consideration that is for the application of Path-finding algorithm Jump Point Search can be tried by adding the number of NPC and also can try using the latest variant of path search algorithm shortest. Also the number of NPCs given the behavior can be increased and try to increase the number of behaviors that the NPC has so that it balances the player's ability.

BIBLIOGRAPHY

[1] John Wiley and Sons, 2014.”Gaming Development Fundamentals”. Hoboken, New Jersey, United State of Amerika.

[2] Cartoon Networks, “http://www.cartoonnetworkasia.com/show/tom-and-jerry”, 2017. [Online]. Available: http://www.cartoonnetworkasia.com/show/tom-and-jerry [Accessed 5 April 2017].

[3] Rosa Delima, 2016, “Perbandingan Penerapan Algoritma A*, IDA*, Jump Point Search, dan PEA*”, Jurnal Teknik Informatika dan Sistem Informasi, Vol. 2, No. 3.

[4] Miftah Fauzan Rahadian, 2016, “Penerapan Metode Finite State Machine Pada Game ‘The Relationship’”, Jurnal Informatika Mulawarman, Vol. 11, No. 1.

[5] Harabor, D., and Grastien, Al. 2011.”Online Graph Pruning for Pathfinding on Grid Maps”, In AAAI.