Pathfinding Algorithms in Game Development

Pathfinding Algorithms in Game Development

Abdul Rafiq[1], Tuty Asmawaty Abdul Kadir[2], Siti Normaziah Ihsan[3]

[1,2,3] Soft Computing and Intelligent System Research Group (SPINT)
Faculty of Computing, College of Computing and Applied Sciences
Universiti Malaysia Pahang, Kuantan, Pahang, Malaysia

E-mail: tuty@ump.edu.my

Abstract. This review paper provides an overview of a pathfinding algorithm for game development which focuses on the algorithms and their contribution to game development. The algorithms were categorised based on their search performance. The aim of this paper is to investigate and provide insights into pathfinding algorithms for game development in the last 10 years. We summarise all pathfinding algorithms and describe their result in terms of performance (time and memory). The result of this paper is metaheuristic techniques have better performance in terms of time and memory compared to heuristic techniques as a pathfinding algorithm.

1. Introduction

The rising global popularity of video games was the trigger to the increasing research interest in solving many AI issues related to video games such as decision-making, movement, strategy, and pathfinding. Commonly, pathfinding for a non-player character uses up a lot of CPU power and memory. This is a problem that has attracted constant attention from researchers. Pathfinding depends on the game environment where the obstacle may be static or dynamic. Video games mostly consist of three components which are player character, non-player character and others. Player character and non-player character are important roles in a video game. Player character is the main character controlled by a user while non-player character or known as NPC is a game object not controlled by a player in a video game [1]. Pathfinding is used for a non-player character (NPC) to find a path between the origin point to the goal point.

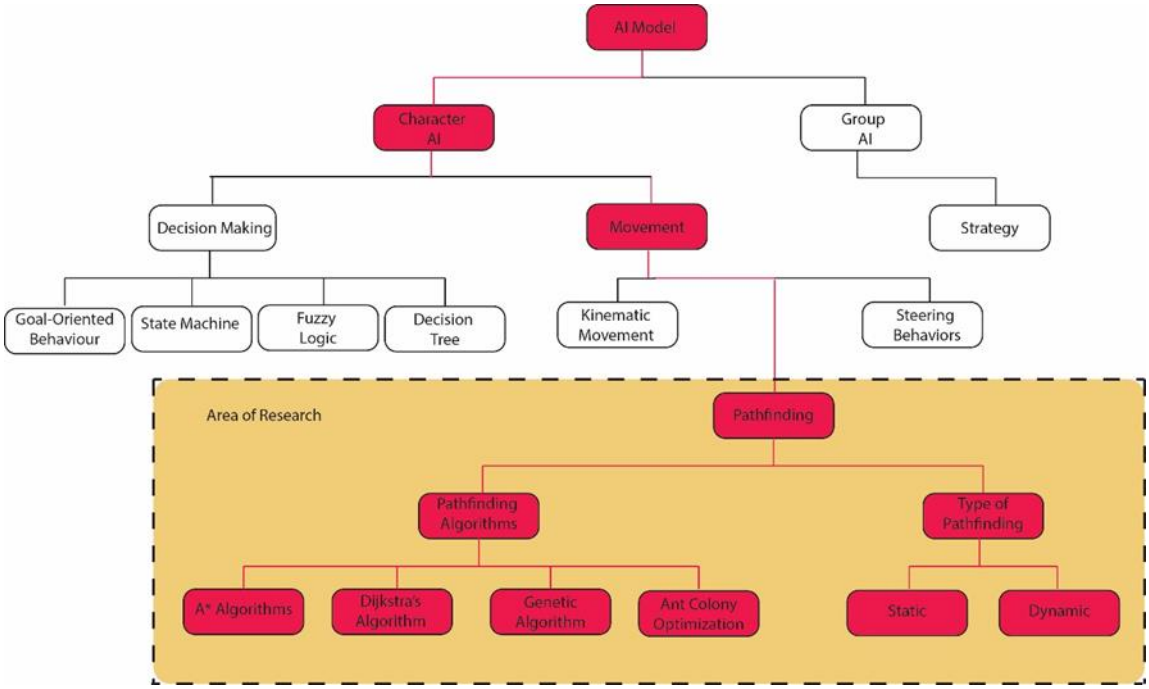

Figure 1. Area of research

Figure 1 illustrates an AI model for a video game. The AI model splits into three sections: Movement, Decision Making and Strategy. Decision-making refers to the character deciding what to do next. Movement refers to the character being able to move anywhere. Strategy refers to group strategy required to coordinate a team. The Movement and Decision-making sections contain algorithms that work for player characters or non-player characters. Pathfinding is part of movement besides kinematic movement and steering behaviours. The techniques or methods used to determine the path in the video game are called the pathfinding algorithms. Common pathfinding algorithms were developed to solve pathfinding problems such as Dijkstra [2], A* algorithm [3], genetic algorithms [4], and ant colony optimisation [5]. There are two types of pathfinding for a non-player character in a video game. The first is static pathfinding where the target node as the player does not move. The second is dynamic or real-time pathfinding where the target node as the player moves freely and randomly.

2. Pathfinding in Games

Pathfinding is a plotting node to find the shortest or minimum path between two points, which is from source to destination by a computer application. Pathfinding is a major component of many important applications in the fields of video games, robotics [5], crowd simulation [6], and GPS [7]. This paper focuses on pathfinding algorithms in video game development. Pathfinding algorithms are used for the agent or non-player character to find a path between the origin point to the goal point. Pathfinding is one of the requirements to create a realistic non-player character in a video game. A video game can be fun and entertaining especially when the non-player character is realistic enough. However, the main problem for a video game is the need for an optimal pathfinding for non-player characters. Garham has supported this by stating that the common problem in a video game is to find suitable pathfinding for agent movement. Pathfinding is implemented in any condition such as static, dynamic and real-time environments. The techniques or methods used to determine the path in a video game are called pathfinding algorithms. Pathfinding algorithms were developed to solve pathfinding problems such as Dijkstra, A* algorithm, genetic algorithms, and ant colony optimisation. Pathfinding algorithms are used to solve the shortest path problem and the optimal path. Usually, A* and Dijkstra's algorithms are used as a solution method to find the shortest path. Ant colony optimisation (ACO) and genetic algorithm are used as a search technique to find the minimum cost path in a graph.

2.1 Heuristic techniques

Heuristic is used to solve the problem in a faster and efficient way by optimal solution, accuracy, and precision [6]. Heuristic algorithms aim to find a good solution to a specific problem like pathfinding in a reasonable amount of computation time but with no guarantee of efficiency. 'Heuristic' means to find in Greek. Based on previous studies, many algorithms were developed to solve pathfinding problems such as the A* algorithm, Dijkstra's algorithm, and Depth First Search. The A* algorithm and Dijkstra's algorithm are the most popular techniques. All these techniques can be categorised as heuristics.

2.1.1 A algorithms*

The A* algorithm is one of the popular techniques used for pathfinding due to its accuracy and performance. It is used to find the shortest path between two nodes. The A* algorithm has been applied in several video game genres such as real-time strategy games [9], role-playing games [10], racing games [11] and turn-based strategy games. The A* algorithm was introduced by Hart, Nilsson, and Raphael in 1967 [12] to solve many problems, pathfinding in a video game being one of them. In the Non-Player (NPC) context, pathfinding is used to guide between two node points in order to capture the player character.

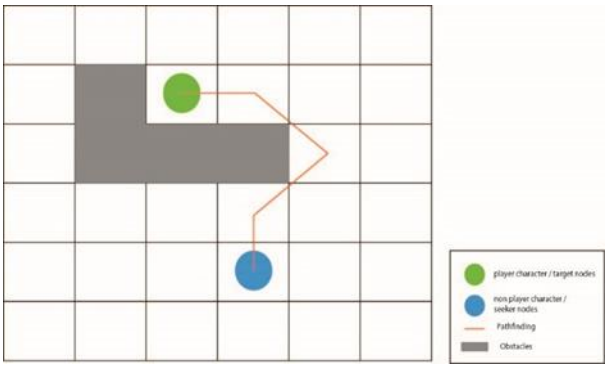

Figure 2. A* algorithm

The A* algorithm always tries to carry pathfinding by exploring the minimum value or lowest path to give the best minimum solution. The A* algorithm implements a heuristic function to evaluate that lowest path. The heuristic function allows the algorithm to quickly and accurately estimate the path. The advantage of the A* algorithm is it is very easy to understand the flow and logic. Due to its simplicity, the A* algorithm has always been chosen by programmers to solve pathfinding problems. This is because the A* algorithm finds the minimum solution by finding the shortest path. A* uses a heuristic function $f_{(n)}$ to determine the node. The value of the function $f_{(n)}$ is:

$$f_{(n)} = g_{(n)} + h_{(n)} \qquad (1)$$

$g_{(n)}$ is the cost that is required to reach a target node from the starting node. $g_{(n)}$ will calculate the cost so far to reach the target node. $h_{(n)}$stands for heuristic value, where estimate form node to target node. If the grid has obstacles, $f_{(n)}$ will estimate and pick the lowest cost to give a good result. The A* algorithm becomes Dijkstra's algorithm when $h_{(n)}$ is zero which is guaranteed to find the shortest path.

2.1.2 Dijkstra's algorithm

Dijkstra's was introduced in 1970 by Holland as one of the best path algorithms [13]. Dijkstra's algorithm is a classic graph search algorithm which can find the shortest path between two points on the graph. It has been applied in many areas such as network routing [14], public transportation [15], and logistics [16]. Dijkstra's algorithm was the only choice for a long time in pathfinding until the A* algorithm became a popular method for pathfinding in video games [17]. We will discuss how Dijkstra's algorithm works as a pathfinding algorithm.

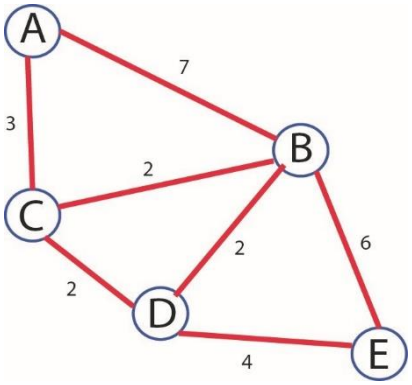

Figure 3. Dijkstra's algorithm

Figure 3 shows a graph with five nodes (A, B, C, D, E) and seven paths. There are many possible paths that will allow us to reach the E node from the A node. Dijkstra's algorithm will help us find the shortest path between the two nodes. From the starting node which is the A node, it visits the path with the smallest value known as the distance. Once it has moved to the smallest node, it will check each of its neighbouring nodes. For each neighbour node, they will calculate the distance from the starting node. If the distance or current path is less than another path, it will update the shortest path.

Table 1. The path node

Node	The path from a node	Shortest path
A	0	-
B	7 & 5	5
C	3	3
D	5,9,11	5
E	9,14,15 & 13	9

Table 1 shows the shortest path from the A node to each node. There are two pathways from A to B and the shortest path is 5. The only path from A to C has a value of 3. We want the shortest path from the A to E nodes. After calculating all the possible paths, we finally find the shortest path by visiting all the related nodes. The shortest distance from the A to E nodes is 9. Dijkstra's algorithm is a simple searching algorithm and easy to understand. It will explore each node to find the shortest path. The advantage of Dijkstra's algorithm is the coding is very easy to implement. Although Dijkstra's algorithm is easy to understand, it is not the best method to solve pathfinding problems.

2.2 Meta-Heuristic techniques

Metaheuristics are general high-level strategies that combine lower-level techniques for exploration and exploitation of the search space. Metaheuristic is a higher level heuristic. Generally, it performs better than heuristic. Metaheuristics can reduce search time and look good enough to solve complex pathfinding for a video game. Based on the study, metaheuristic algorithms such as genetic algorithm and ant colony optimisation were used to solve the pathfinding problem in a video game. Metaheuristics are based on some natural phenomenon. The most successful metaheuristic algorithms have been inspired by natural systems. For example, ant colony optimisation (ACO) and the bee algorithm were developed based on the behaviour of animals.

2.2.1 Genetic Algorithm

The genetic algorithm is one of the popular techniques used for searching in computer science. It was used to find a heuristic solution for optimisation and search problems in a large area of space. The genetic algorithm was introduced by John Holland [7]. It has been applied in many areas such as biomimetic invention [8], automotive design [9], engineering design [9] and robotics [10]. The genetic algorithm uses biological methods such as mutation and inheritance. GA converts the decision variables of the search problem into strings. The strings act as candidate solutions and the search problem is referred to as chromosomes. The alphabets act as genes and the values of genes are called alleles. For example, a salesman has a travelling problem and the route represents chromosomes while the city is a gene. In video games, the genetic algorithm has been applied as a pathfinding algorithm. The genetic algorithm and best-first search were used to optimise the search for the path between two points [11]. The result shows that GA has a better performance on a map with obstacles compared to best-first search. Other than that, the experiment to optimise pathfinding using a techniques genetic algorithm A* algorithm. The result shows that RTP-GA has a better performance in term of searching time on maps with obstacles compared to the A*algorithm [4]. Another result shows GAMMAs find paths quicker than the A* algorithm in terms of time [10].

2.2.2 Ant Colony Optimisation

Ant Colony Optimisation (ACO) is a technique inspired by the behaviour of ants when navigating a path from their nest to food sources. It was first described in 1997 by Gambardella Dorigo [12]. The Ant Colony Optimisation (ACO) technique is based on the ants' ability to find the shortest path between their nest and food sources. Firstly, each of the ants starts moving randomly. Each member of the ant colony tries to find a food source. After finding food, they will communicate with each other via pheromone trails to navigate the shortest path. Pheromones are a type of chemical substance released into the environment by ants. The connection line will be created as pathfinding between the starting point (nest) to the target point (food).

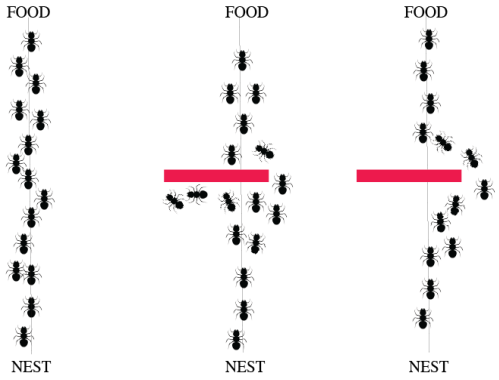

Figure 4. Ant Colony Optimisation

Figure 4 shows that the main idea of Ant Colony Optimisation (ACO) is mimicking real ant behaviour. In the first picture, each ant moves in a line from its nest to the food. In the middle picture, an obstacle is placed in the line between their nests to the food. Each ant moves randomly left and right to avoid the obstacle. The ants go around the obstacle and choose to turn right because it will reach the food quicker, whereas by turning left it will take a longer time and path. Pheromones are a chemical substance released into the environment by ants. Finally, the pheromones show a shorter and faster path around the obstacle.

The main objective of Ant Colony Optimisation (ACO) is as a search technique to find a minimum cost path in a graph. Ant Colony Optimisation (ACO) is used in many applications in the fields of video games [13], robotics [14], submarine [15] and so on. In the field of robotics, this paper proposes Ant Colony Optimisation (ACO) to improve path planning for robots. Ant Colony Optimisation (ACO) is applied in pathfinding for a robot to find an optimal path from a source point to a goal point while avoiding any obstacles in the configuration space. Other than that, Ant Colony Optimisation (ACO) is used for vehicles such as submarines. This paper is a study of a modified Ant Colony Optimisation (ACO) technique to improve pathfinding for a submarine in the ocean [15]. The Ant Colony Optimisation (ACO) technique has an advantage over the A* algorithm. The ant colony can adapt to changes in real time. According to V. Selvi, the ant colony optimisation (ACO) technique can be used in a dynamic application because it is able to adapt to changes such as new distances [16]. Due to its advantage, many researchers have applied and improved the ant colony technique to solve a certain complex environment.

3. Findings

Table 2 shows the summary of pathfinding algorithms used for game development. The papers are selected from those published between 2007 and 2018. Only 10 papers are related to the study. The summary is categorised into 5 sections; Year, Authors, Pathfinding Algorithms, Time, and Memory.

Table 2. Summary of pathfinding algorithms

Year	Authors	Pathfinding Algorithms	Time	Memory
2007	Leigh [17]	Genetic algorithm, A* algorithm	GAMMAs find paths quicker than A* algorithm	-
2011	Machado [4]	Genetic algorithm, A* algorithm	RTP-GA has a better performance in term of searching time on maps with obstacles compared to A*algorithm	-
2012	Santos [11]	Genetic algorithm	PPGA has a better performance on the map with obstacles compared to Best-First Search	-
2012	Recio [18]	Ant Colony Optimization (ACO)	-	-
2016	Hunkeler [5]	Ant Colony Optimization (ACO), Genetic algorithm	Ant Colony Optimization has a better performance compared to genetic algorithm	-
2016	Zikky [2]	Dijkstra's algorithm, A* algorithm	A* algorithm better than Dijkstra's algorithm in solving the shortest pathfinding problem	-
2017	Firmansyah [19]	A* and improved A* algorithm	Improved A* get the fastest path, efficient and short time compared to basic A* algorithm	-
2017	Primanita [20]	Iterative Deepening A* (IDA*), A* algorithm	If the map is without any obstacle IDA* better than A* in terms of time usage but IDA* worse than A* if the map has obstacles.	If the map is without any obstacle, IDA* better than A* in terms of CPU memory
2018	Sabri [3]	A* algorithm, Bee algorithm	A* algorithm faster than Bee in a simple map	Bee better than A* in a complex map
2018	Sazaki [21]	Hybrid A* Algorithms	Hybrid A* get better results compared to A* algorithm on an empty track	Hybrid A* get better results compared to A* algorithm on an empty track

4. Results

In this section we summarise and conclude all the pathfinding algorithms such as the A* algorithm, Dijkstra's algorithm, Genetic algorithm, and Ant Colony Optimisation used for game development. We start with the common pathfinding algorithms used for game development. Then, we summarise and conclude the result based on the performance of time and memory.

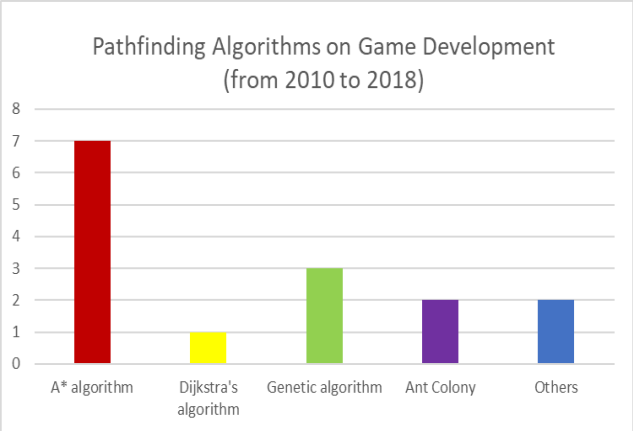

Figure 5. Result of Pathfinding Algorithms

The bar graph illustrates the pathfinding algorithms (Dijkstra's algorithm, A* algorithm, Genetic Algorithm, Ant Colony Optimisation, and Others) used for game development from 2010 until 2018. Overall, The A* algorithm is the most popular technique applied to pathfinding in game development. Both Dijkstra's algorithm and Best First Search (Others) are lesser used techniques applied on pathfinding for game development. Furthermore, metaheuristic techniques such as genetic algorithm and ant colony optimisation have seen a constant increase in usage as pathfinding algorithms from 2010-2019.

In terms of heuristic, the A* algorithm is the most commonly used technique for pathfinding compared to others such as Dijkstra's algorithm and best-first search. The advantage of the A* algorithm is its flow and logic are very easy to understand. Furthermore, the A* algorithm uses a heuristic function which allows the algorithm to quickly and accurately estimate a path. Due to its advantages, the A* algorithm has always been chosen by game programmers as the pathfinding algorithm in game development.

There are many available metaheuristic techniques, but we have found that the genetic algorithm and ant colony optimisation are most commonly used in pathfinding for games. Both algorithms are compared with the A* algorithm. Based on the study, both algorithms perform better than the A* algorithm in terms of time and memory usage. Metaheuristic is a higher level heuristic and generally performs better.

4.1 Time

In this section, we gathered all the pathfinding algorithms based on the result of time. The papers are selected from those published between 2007 and 2018. Only nine papers are related to the study. All the pathfinding algorithms were used for game development. We summarise and conclude the pathfinding algorithms in terms of time performance.

Table 3. Result of time

Years	Authors	Result (Time)
2007	[17]	GA better than A*
2011	[4]	GA better than A*
2012	[11]	GA better than First Search
2016	[5]	ACO better than GA
2016	[2]	A* better than Dijkstra
2017	[19]	Improved A* better than A*
2017	[20]	IDA* better than A*
2018	[3]	A* better than Bee (in the simple map)
2018	[21]	Hybrid A* better than A*

Table 3 shows the results of pathfinding algorithms used for game development based on the performance of time. For heuristic techniques, the A* algorithm is faster than Dijkstra for calculating and searching a path [2]. However, a few researchers improved the A* algorithm. In 2017, Firmansyah [19], Primanita [20] and Sazaki [21] improved the A* algorithm and the result was much better than the basic A* algorithm. Overall, for heuristic techniques, the A* algorithm has a good result compared to others, but it can be bettered by the improved A* algorithm.

Metaheuristic techniques are better than heuristic techniques. From the results in Table 4, the Genetic algorithm is faster than the A* algorithm [17], [4]. The Genetic algorithm is also better than another heuristic technique like Best First Search [11]. Metaheuristic is a higher level of heuristic. Generally, it performs better than heuristic. In 2016, the experiment by Hunkeler showed that Ant Colony Optimisation is faster than the Genetic Algorithm [5]. Overall, we can conclude that metaheuristic techniques such as GA, ACO can be a pathfinding algorithm and the results show that they are better than heuristic techniques.

4.2 Memory

In this section we gather all the pathfinding algorithms based on the result of memory. The papers are selected from those published between 2017 and 2018. Only three papers are related to the study. All the pathfinding algorithms were used for game development. We summarise and conclude the pathfinding algorithms in terms of memory usage performance in this section.

Table 4. Result of memory

Year	Author	Result(Memory)
2017	[20]	IDA* better than A* on an empty map
2018	[3]	Bee algorithm better than A* on a complex map
2018	[21]	Hybrid A* gets better results compared to A* algorithm on an empty track

Table 4 shows the results of pathfinding algorithms used for game development based on the performance of memory usage. Three of 10 papers investigated the performance of pathfinding algorithms in terms of memory. For heuristic techniques, the improved A* algorithms like IDA* [10] and Hybrid A* [31] have better results compared to the basic A* algorithm. From the result, we can conclude that improving the algorithm results in reduced memory usage. For a complex map, metaheuristic techniques are very good compared to heuristic techniques. For example, the Bee algorithm shows a better result in a complex map compared to the A* algorithm [2]. Overall, we can

conclude that metaheuristic techniques such as the Bee algorithm can be a pathfinding algorithm and the result shows that it is better than heuristic techniques like the A* algorithm.

5. Summary and Future Trend

This paper provided a brief description of the common pathfinding algorithms (A* algorithm, Dijkstra's algorithm, Genetic algorithm, and Ant Colony) used for game development. In this review paper, we summarised all the pathfinding algorithms and described the result based on their performance in terms of time taken and memory usage. Based on the result, we found that the A* algorithms are the most popular techniques among the other techniques. The improved A* algorithm gets the fastest path and shortest time. However, it takes too much memory to calculate the path in a dynamic environment. Overall, metaheuristic techniques have better performance in terms of time and memory compared to heuristic techniques. In the future, we need good pathfinding algorithms that work for new technology such as Virtual Reality (VR), Augmented Reality (AR), and Hologram.

Acknowledgements

The authors would like to thank Universiti Malaysia Pahang (UMP). This work is supported by Universiti Malaysia Pahang (UMP) and funded by Ministry Education Malaysia under FRGS Grant FRGS/1/2016/ICT01/UMP/02/2 and Master Research Scheme (MRS).

References

[1] C. Foudil, D. Noureddine, C. Sanza, and Y. Duthen, "Path Finding and Collision Avoidance in Crowd Simulation," *J. Comput. Inf. Technol.*, vol. 17, no. 3, p. 217, 2009.
[2] M. Zikky, "Review of A* (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game," *Emit. Int. J. Eng. Technol.*, vol. 4, no. 1, pp. 141–149, 2016.
[3] A. N. Sabri, N. H. M. Radzi, and A. A. Samah, "A study on Bee algorithm and A∗ algorithm for pathfinding in games," *ISCAIE 2018 - 2018 IEEE Symp. Comput. Appl. Ind. Electron.*, pp. 224–229, 2018.
[4] A. F. D. V. Machado *et al.*, "Real time pathfinding with genetic algorithm," *Brazilian Symp. Games Digit. Entertain. SBGAMES*, pp. 215–221, 2011.
[5] I. Hunkeler, F. Schar, R. Dornberger, and T. Hanne, "FairGhosts - Ant colony controlled ghosts for Ms. Pac-Man," *2016 IEEE Congr. Evol. Comput. CEC 2016*, pp. 4214–4220, 2016.
[6] L. A. Wolsey, "Heuristic Algorithms," *Integer Program.*, no. January, p. 17, 1998.
[7] J. Car, "An Introduction to Genetic Algorithms.," *Artif. Life*, vol. 3, no. 1, pp. 63–65, 2014.
[8] A. Cenys, D. Gibavicius, N. Goranin, and L. Marozas, "Genetic algorithm based palm recognition method for biometric authentication systems," *Elektron. ir Elektrotechnika*, vol. 19, no. 2, pp. 69–74, 2013.
[9] H. Yu and N. Yu, "Application of Genetic Algorithms To Vehicle Suspension Design," pp. 1–9, 2003.
[10] T. W. Manikas, K. Ashenayi, and R. L. Wainwright, "Genetic algorithms for autonomous robot navigation," *IEEE Instrum. Meas. Mag.*, vol. 10, no. 6, pp. 26–31, 2007.
[11] U. O. Santos, A. F. V Machado, and E. W. G. Clua, "Pathfinding Based on Pattern Detection Using Genetic Algorithms," *XI Brazilian Symp. Games Digit. Entertain.*, pp. 64–72, 2012.
[12] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theor. Comput. Sci.*, vol. 344, no. 2–3, pp. 243–278, 2005.
[13] I. Sabuncu, "Work-In-Progress: Solving Sudoku Puzzles Using Hybrid Ant Colony Optimization Algorithm," *2015 1st Int. Conf. Ind. Networks Intell. Syst.*, pp. 181–184, 2015.
[14] H. Wang, Z. Wang, L. Yu, X. Wang, and C. Liu, "Ant Colony Optimization with Improved

Potential Field Heuristic for Robot Path Planning," *Chinese Control Conf. CCC*, vol. 2018-July, pp. 5317–5321, 2018.
[15] Y. Shan, "Study on submarine path planning based on modified ant colony optimization algorithm," *Proc. 2018 IEEE Int. Conf. Mechatronics Autom. ICMA 2018*, pp. 288–292, 2018.
[16] Vs. DrRUmarani Lecturer and A. professor, "Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques," *Int. J. Comput. Appl.*, vol. 5, no. 4, pp. 975–8887, 2010.
[17] R. Leigh, S. J. Louis, and C. Miles, "Using a genetic algorithm to explore A*-like pathfinding algorithms," *Proc. 2007 IEEE Symp. Comput. Intell. Games, CIG 2007*, pp. 72–79, 2007.
[18] G. Recio, E. Martin, C. Estebanez, and Y. Saez, "AntBot: Ant colonies for video games," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 4, pp. 295–308, 2012.
[19] E. R. Firmansyah, S. U. Masruroh, and F. Fahrianto, "Comparative analysis Of A∗ and basic theta∗ algorithm in android-based pathfmding games," *Proc. - 6th Int. Conf. Inf. Commun. Technol. Muslim World, ICT4M 2016*, pp. 275–280, 2017.
[20] A. Primanita, R. Effendi, and W. Hidayat, "Comparison of A∗ and Iterative Deepening A∗ algorithms for non-player character in Role Playing Game," *ICECOS 2017 - Proceeding 2017 Int. Conf. Electr. Eng. Comput. Sci. Sustain. Cult. Herit. Towar. Smart Environ. Better Futur.*, pp. 202–205, 2017.
[21] Y. Sazaki, A. Primanita, and M. Syahroyni, "Pathfinding car racing game using dynamic pathfinding algorithm and algorithm A∗," *Proc. - ICWT 2017 3rd Int. Conf. Wirel. Telemat. 2017*, vol. 2017-July, pp. 164–169, 2018.