Journal of Positive School Psychology
2022, Vol. 6, No. 3, 8434–8441 

# Comparing the efficiency of Pathfinding Algorithms for NPCs in platform games

Umar Affandi Shahrin Iskandar[1], Norizan Mat Diah[2]*, Marina Ismail1[3], Azizi Abdullah[4]

*[1,2,3]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia*
*[4]Faculty of Information Science and Technology, The National University of Malaysia, 43600 Bangi, Malaysia*
*Email: [1]english1119@gmail.com, [2]norizan@fskm.uitm.edu.my, [3]marina@fskm.uitm.edu.my, [4]azizia@ukm.edu.my*

**Abstract**

Pathfinding has been a significant video game research area for decades. It is usually utilised as the core of any Artificial Intelligence moves in computer games. This research aims to identify a better suited and more efficient pathfinding algorithm for the platformer video game genre. This study compared two algorithms: the A* and Dijkstra algorithms. Both algorithms were implemented in a platform game environment and tested with several different obstacles for non-player characters (NPCs). The parameters measured were processing time, the length of the path taken, and the number of blocks/nodes played in the computational process. To evaluate the algorithms' performance, the travel time taken, the computed nodes, and the distance travelled by the NPC to reach its destination were analysed for each algorithm. The findings indicate that both algorithms are suitable for specific conditions in a platformer environment; Dijkstra's performed accurately and managed to find the shortest path when the route to the objective required less vertical movement, while A* performed more efficiently when the NPC was required to reach an objective that required more vertical movement. The results also suggest that A* performed better than Dijkstra's algorithm, as it has a heuristics function that increased its flexibility.

**Key-words**: A*, Dijkstra, Non-Player Character (NPC), Pathfinding, platform game.

## I. INTRODUCTION

Computer games, or more commonly referred to as video games, have constantly been evolving ever since their emergence in the 1950s, as observed by Chikani (2015). In recent years, these games have been expanding rapidly to become the multi-billion-dollar industry that it is today, while encompassing more than just entertainment for kids. The gaming industry is now also a medium used to tell complete and engaging stories, with exceptional graphics that takes players into fantasy worlds where almost anything can happen. Besides, with the advent of e-sports and online streaming, gaming has also become a viable career option for those intending to take their hobbies and skills further. Perez et al. (2019) stated that Artificial Intelligence or AI is an essential component of a video game. The AI defines how a computer opponent behaves in a video game, where their behaviours could range from simple patterns that repeatedly loop in to even beating a championship-holding player. Of the essential aspects of video game AI is pathfinding.

Pathfinding has been a significant research area in video games for decades. Many programmers have attempted to find the best way to represent realistic movements for their creations. Such research has led to the creation of many different pathfinding algorithms just for video games in general. Depending on the genre and

type of video game that has been developed, certain pathfinding algorithms may serve certain games better than others. The appropriate selection of algorithms that can best serve as the building block of a specific genre or type of video game is also essential so as not to waste development resources needlessly. However, despite existing research within the field of pathfinding (Ostrowski, 2015), there seems to be barely any study on the efficiency of specific pathfinding algorithms in a specific video game genre.

Non-player characters (NPCs), which may include animals, monsters, humans, or vehicles, are all required to move in a goal-oriented manner, and as such, the programming must be able to identify a good path for the NPCs to reach their goal from their origin. These movements must be identified while simultaneously avoiding obstacles and using the most efficient path there is. Therefore, it is essential to build and apply the most appropriate pathfinding algorithm to serve different types of video games according to what the developer needs. One of the video games that has made extensive use of AI is the platformer video game.

## II. BACKGROUND OF STUDY

Computer games and AI have long been associated with each other. Even before the computer sciences acknowledged AI as a valid field of study, early researchers of computer sciences had written programs that could play games to determine whether computers could handle and solve tasks that generally would require intelligence. Most early research on game-playing AI was focused on classic board games, such as Checkers and Chess, as stated by (Yannakakis & Togelius, 2018). Yannakakis and Togelius (2018) also observed that in the last two decades, a research community has emerged around applying AI to games other than board games, in particular computer games. In this case, AI is used in most computer games to induce difficulty and challenges depending on the type and genre of the game or when it is impossible to provide another human opponent for the player.

AI research involves areas of Machine Learning, high-level decision-making based on random inputs, and eventually, accurate intelligence equal to those of a human, as detailed in a survey by Müller et al. (2016). Game AI often only consists of some if-then conditions or heuristics that are just enough to give a player a good challenge or experience during gameplay. One of the more prominent uses of AI in video games is for pathfinding or NPCs or objects in video games.

Pathfinding is one of the essential aspects of computer game development. According to Krishnaswamy et al. (2009), one of the crucial elements of game AI is pathfinding, i.e., enabling an agent to calculate a path from a starting point to a target around any number of obstacles. Developers must choose the correct algorithm as the groundwork for starting a project.

Over the years of computer game development, agent movement has been singled out as one of the biggest challenges in designing realistic AI in computer games. Designing AI for many games is about the movement of agents or bots, also known as non-playable characters (NPC), around a virtual world. Designing and developing a complex system for decision-making is worthless if an NPC cannot overcome a set of obstacles by implementing those decisions.

The most common issue of pathfinding in a video game is to cleverly avoid obstacles and seek out the most efficient path over different terrain (Cui & Shi, 2011). This means that the criteria to choose the most suitable algorithms include that it can lead the NPC on the shortest path with the least computing time. The shortest path can mean how fast the NPC navigates through the nodes and how many moves it needs to make to get the shortest path.

Krishnaswamy et al. (2009) compared the efficiency of Pathfinding Algorithms in Game Development. The study conducted a test between Djisktra A* and D* in a top-down simulated environment, with an agent trying to

find a designated goal. In their experiments, they used factors such as node visitations, average nodes in a path, the average length of a path, average path execution time, and finally, average path execution time in main application loops to determine the efficiency of the three algorithms. Another similar research was done by Permana et al. (2018), who researched the comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS, in a Maze Runner Game. The study compared the efficiency of the algorithms in a 2-D top-down maze runner game. In their experiments, they ran three different algorithms: A*, Dijkstra, and Breadth-First Search, considering the computed blocks, the time taken to reach the goal, and the distance travelled by each algorithm.

The next study was Barnouti's (2016) Pathfinding in Strategy Games and Maze Solving using the A* Search Algorithm. The study discussed the implementation of the A* algorithm in a strategy game and briefly compared it to other algorithms like Depth-First Search, Breadth-First Search, and Best First Search algorithms.

Sazaki et al. (2017) used a pathfinding algorithm for an NPC to race against the player in a car racing game. The pathfinding method used by the NPCs in this game was the A* algorithm, used to find the shortest path on the track, and combined with the Dynamic Pathfinding Algorithm to avoid static or dynamic obstacles in its path. The study showed that both methods could be implemented in car racing games, with the track conditions being blocked by static obstacles. While moving on the track with dynamic obstacles, the combination of both methods passed through the course under certain conditions only.

The final study conducted the Simulation and Comparison of Efficiency in Pathfinding algorithms in games (Noori et al., 2015). The research compared all the existing pathfinding algorithms in a maze runner-like simulation using a 2-D map. Some of the algorithms used in the research included A*, Breadth-first Search, Depth-First Search, and Dijkstra's algorithm.

The current research project compares A * and Dijkstra pathfinding algorithms to determine their efficiency as a groundwork algorithm for a platformer game. Although previous research has compared the algorithms' efficiency, none has focused on this computer game genre. The study will employ the same basic efficiency tests as per previous research to measure the algorithms tested, with the variables being the processing time, the length of the path taken, and the number of blocks/nodes played in the computational process.

## III. IMPLEMENTATION OF THE PATHFINDING ALGORITHMS

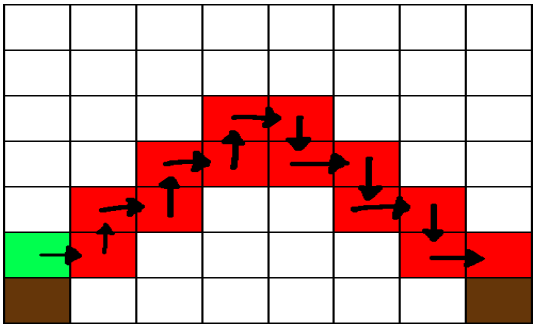
The research requires creating a 2-D platformer game prototype to prepare a suitable environment to test both algorithms. Both Dijkstra and A* algorithm will be developed and implemented to suit the designed platformer game. Both algorithms will be executed on the created simulations or levels to test and record the results for both algorithms.

After both algorithms have been implemented in the developed platformer game, testing and simulation will be done to compare the results. Based on previous research in the related field, that the best way to evaluate the efficiency of the algorithms is to analyse the algorithm's process time, length of path taken, and the number of blocks/nodes that are played in the computational process after simulation and testing (Bintoro et al., 2018).

The algorithms need to be modified to work with a platformer-type videogame by implementing the gravitational restrictions associated with platformer games. Then, the algorithm is used to create an NPC that attempts to reach a destination most efficiently as determined by the respective algorithms. Branicki (2015) stated that it is essential to decide on the paths that can be taken by the bot or NPC and what form the paths themselves will be when adapting the algorithms. In a platformer, the ability to jump is a key part of the game and will need to be considered. The rule definitions of the NPC or the character movement must also be precise. For example,

an NPC can jump at a height of 3 cells. Its movement during a jump should be accounted for, and once it has reached the maximum height, it should still move to the side. Figure 1 shows an example of a character's maximum movement path that follows the rules defined above.

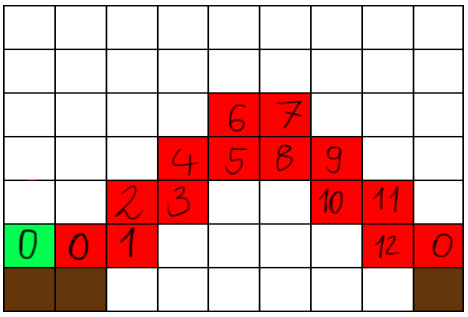
**Figure 1– Character movement path in a jumping arc**



During this movement, each cell that is in the NPC's movement path will have to keep track of the data on jump height to ensure that the character is not going any higher than allowed to eventually fall. In this issue, each cell needs to be assigned jump values and each cell value should be increased by increments of 1 for the jump duration.

As can be seen from Figure 2 below, the cell with the number 6 is the highest point during the jump, and as the character starts to fall, the cells involved during the fall duration continues the increments, as the jump is still considered ongoing. The value 0 denotes when the character is grounded. Referring to Branicki (2015), for a standard A* or Dijkstra algorithm, nodes that are already visited are usually never processed again. However, in a platformer, these nodes need to be processed because the jump values in the nodes must also be considered, instead of just the x- and y-axis coordinates.

**Figure 2– Cell values during jumping movement**




Both the A* and Djikstra algorithms were implemented in the same way. Note that A* is a modification of the Dijkstra's algorithm that uses a heuristic to determine which vertices to search and can be changed between both algorithms through adding and removing the heuristics functions from the algorithms. The heuristic is the estimated cost (or distance) from the node that is being searched, to the target node. The lower the heuristics, the more likely one will explore the next node's edges instead of some node that is further away. Hence, reaching the target from the source will be much faster.

### *3.1 Experimentation and Testing*

In the study, both the A* algorithm and Djikstra's algorithm were implemented in the prototype platformer game as a testing environment for the algorithms' efficiency. The NPC will move to a directed objective, and node lines will appear, showing the calculated nodes done by the algorithms and the movement paths that the NPC will take to reach its destination. Krishnaswamy et al. (2009) compared the efficiency of both Dijkstra's and A* in a top-down simulated environment, instead of a platformer, with three different sets of environments as the testing ground.

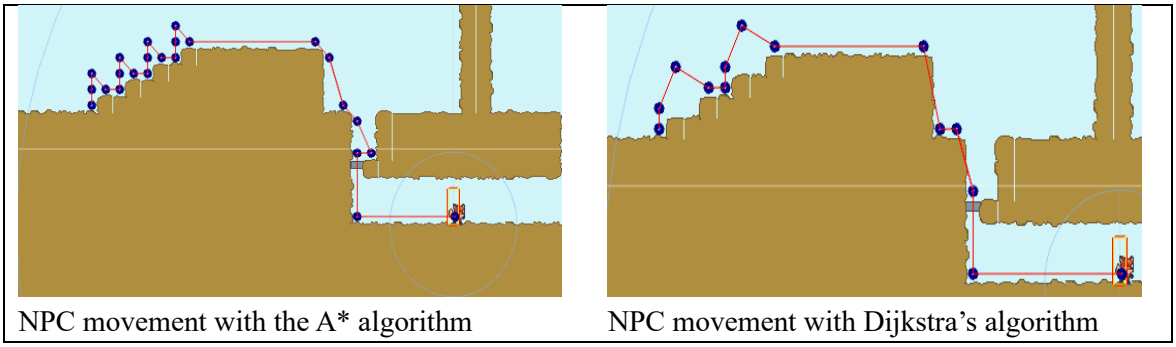
Both the A * and Djikstra's algorithms were tested with the same obstacles, with three variations, and the results for each variation were recorded. All the variations were tested ten times each with each algorithm to ensure that the readings taken were accurate and not anomalous.

#### *3.1.2 Testing Variation 1*

During this variation (Figure 3), it was observed that the NPC with the A* algorithm showed a total of 24 computed movement nodes

compared to Dijkstra's algorithm, which only required 10 nodes in total. The travel time for both algorithms was the same, at 10 seconds, while the maximum distance travelled by A* was a bit longer, at 715, compared to Dijkstra's 686, (refer to Table 1). In this instance, the A* algorithm opted for a smaller jump arc to ascend the obstacles as opposed to Dijkstra, where the NPC used a higher jumping arc to climb the obstacles. While descending, the NPC in A* also moved to an extra node before descending further to reach the objective, while the NPC with Dijkstra's algorithm was able to calculate a more direct path and required 1 less node for itself.

**Figure 3– Movement of NPC during Variation 1 for both algorithms**



NPC movement with the A* algorithm NPC movement with Dijkstra's algorithm

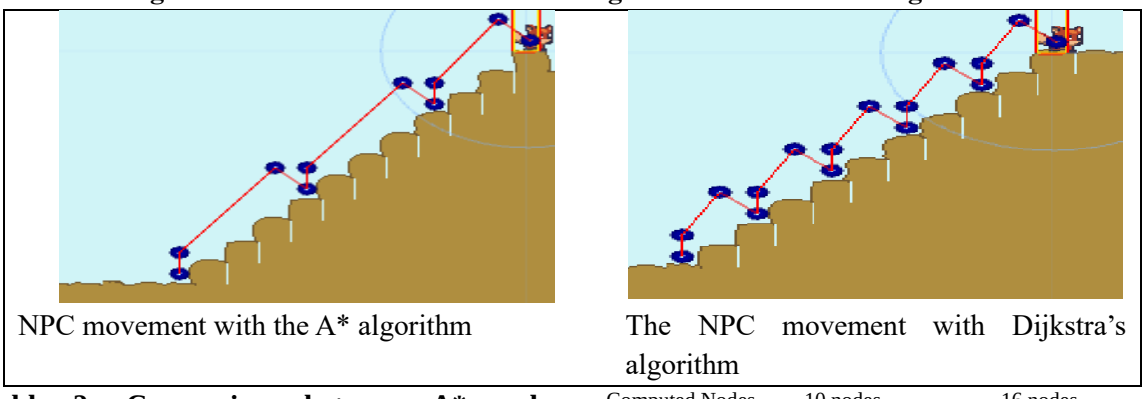
**Table 1– Comparison between A* and Dijkstra for Variation 1**

| Variation 1 | A* | Dijkstra |
|---|---|---|
| Travel time | 11 seconds | 10 seconds |
| Computed Nodes | 24 nodes | 14 nodes |
| Distance travelled (In the Y- and X-axis) | 715 | 686 |

### *3.1.2. Testing Variation 2*

For testing variation 2 (Figure 4), the NPC with A* algorithm was more efficient, as it had fewer computed nodes-10 nodes as compared to the 16 nodes needed by Dijkstra. It also required less travel time at 4 seconds, making it 1 second faster than Dijkstra's 5 seconds, and it had lesser distance travelled due to using fewer jumps overall, with 415 distance while Dijkstra travelled a total of 475 distance. In Table 2, the A* algorithm made higher jumps to ascend the obstacles faster, while Djikstra's algorithm required the NPC to make more jumps, causing it to be slower and with increased total distance travelled due to the Y-axis movement of the jumps.

**Figure 4– Movement of the NPC during Variation 2 for both algorithms**



NPC movement with the A* algorithm The NPC movement with Dijkstra's algorithm

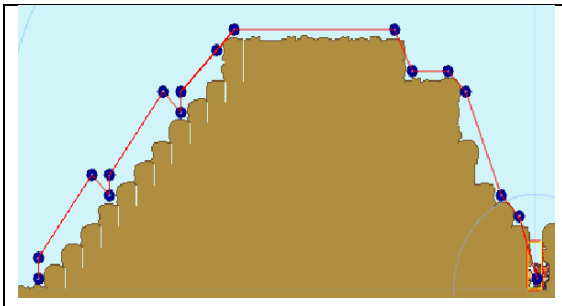
**Table 2– Comparison between A* and Dijkstra for Variation 2**

| Variation 2 | A* | Dijkstra |
|---|---|---|
| Travel time | 4 seconds | 5 seconds |
| Computed Nodes | 10 nodes | 16 nodes |
| Distance travelled (In the Y- and X-axis) | 415 | 475 |

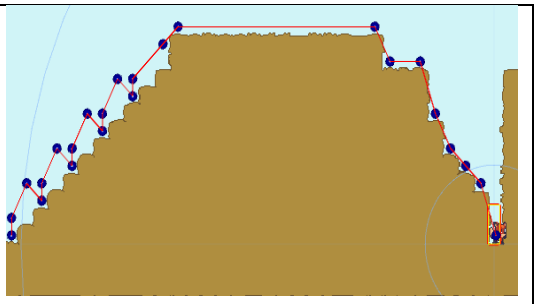
### *3.2.3. Testing Variation 3*

In Variation 3 (Figure 5), NPC with the A* algorithm was more efficient, as it had fewer computed nodes, with 17 nodes compared to the 23 nodes needed by Dijkstra. It also required less travel time, at 11 seconds, making it 1 second faster than Dijkstra's 12 seconds. It also had a lesser distance travelled due to it using fewer jumps overall with 875 distance while Dijkstra travelled a total of 997 distance, as per Table 3. It is observed that A* made higher jumps to ascend the obstacles faster while Djikstra's required the NPC to make more jumps causing it to be slower and with increased total distance travelled due to the Y-axis movement of the jumps. Djikstra's descending of the obstacle was observed to take the more grounded route than the A* algorithm approach, which took the more direct horizontal route available.

**Figure 5– Movement of NPC during Variation 3 for both algorithms**



NPC movement with the A* algorithm



NPC movement with Dijkstra's algorithm

**Table 3– Comparison between A* and Djikstra for Variation 3**

| Test 3 | A* | Dijkstra |
|---|---|---|
| Travel time | 11 seconds | 12 seconds |
| Computed Nodes | 17 nodes | 23 nodes |
| Distance travelled (In the Y- and X-axis) | 875 | 997 |

## IV. RESULTS AND FINDINGS

The results of both algorithms show that the A* algorithm will perform more efficiently once the obstacles that the NPC needs to pass through passed a particular height value, as it will attempt to use the highest possible jumping arc that is available to the NPC to reach the destination faster while requiring less computed nodes. Meanwhile, Dijkstra's algorithm was more efficient when the height to overcome was below a particular value. When this condition is fulfilled, it performs better than the A* algorithm. However, this result does not seem to correlate with previous research, where Dijkstra is always meant to find the shortest path in any situation. Based on the findings of similar studies, a more plausible explanation is that when the distance to the objective becomes farther and higher, Dijkstra's algorithm fails to consider the jump values in the nodes and instead only considers the x- and y-axis coordinates due to possible coding issues. The A* algorithm manages to avoid this problem thanks to its multiple heuristics functions, including custom-made ones that can and will take the above into account when calculating the nodes. As such, it can be said that A* is more suitable for a platformer due to its flexibility, which is due to the heuristics function.

The result of this study indicates that Dijkstra's algorithm is more suitable for closer distance and precise jumping movement, as it will more efficiently help the NPC to reach a closer destination, while the A* algorithm performs better when the NPC needs to get to a farther location and thus requires less precise jumping movements.

The results should help determine which algorithm to use depending on an NPC's travel route in a particular platformer environment. For example, a programmer may decide to use Dijkstra in a more enclosed space, as this means the NPC will not have much room to navigate and thus will need a more precise movement

afforded by Dijkstra's algorithm as opposed to A*, which may cause the NPC to get stuck in the environment. In summary, both algorithms are helpful in a platformer game when the conditions for both algorithms to perform effectively in the platformer environment are met.

Choosing the most suitable pathfinding algorithm for an NPC is important, as such a decision will determine the challenge level an NPC may impose on the player. A less efficient algorithm that cannot track a player character effectively will provide less of a challenge and reduce the amount of fun when playing a platformer game. Meanwhile, the game developer may intentionally use a less efficient pathfinding algorithm to minimise the difficulty for less skilled players to enjoy the game at their own skill level.

## V. CONCLUSION

This study aimed to determine the efficiency of A* and Dijkstra as a pathfinding algorithm for a platformer game environment, where the NPC movement on the Y-axis is restricted. The results of the efficiency of both algorithms indicate that A* is more suitable for long-distance movement in a wide area. At the same time, Dijkstra performed better in an enclosed space with close-distance movements. This finding is essential since choosing a suitable pathfinding algorithm affects the gameplay, whether the NPC imposes more of a challenge on a player character or not. Game developers may also use this finding to balance the level of challenge in a platformer game to provide different challenge levels for players with different skill levels.

Since the study only focused on the ability to jump and its effect on the movement arc of a grounded NPC in the platformer environment, the results of this study will only apply to NPCs that strictly use ground-based movement and not other navigation methods. Hence, further studies can still be conducted to identify how the algorithm would affect the movement efficiency of the NPC when it is given other movement types that are also associated with a platformer game, such as swimming, climbing, or limited fight. It is also recommended that a possible modification to the Djikstra algorithm be made to account for the jump values in the nodes instead of just through the x- and y-axis coordinates when the objective is at a farther distance.

## BIBLIOGRAPHY


1. BARNOUTI, N. H., AL-DABBAGH, S. S. M., NASER, M. A. S. (2016). Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm. Journal of Computer and Communications, 4, 15-25
2. BRANICKI, D., (2015), A* Pathfinding for 2D Grid-Based Platformers, Retrieved from URL: <https://gamedevelopment.tutsplus.com/tutorials/a-pathfinding-for-2d-grid-based-platformers-making-a-bot-follow-the-path--cms-24913> Accessed Feb 16, 2021
3. CHIKHANI, R. (2015). The History of Gaming: An Evolving Community | TechCrunch. Techcrunch. URL: <https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/> Accessed May 3, 2021
4. CUI, X. & SHI, H. (2011). A* - based Pathfinding in Modern Computer Games, IJCSNS Int. J. Comput. Sci. Netw. Secur., 11(1),125-130.
5. KRISHNASWAMY, N. (2009). Comparison of Efficiency in Pathfinding Algorithms in Game Development. The Honors Program Senior Thesis, College of Computing and Digital Media DePaul University, Chicago, Illinois, United States
6. HANDY PERMANA, S. D., YOGHA BINTORO, K. B., ARIFITAMA, B. & SYAHPUTRA, A. (2018). Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game. International Journal of Information System & Technology, 1(2), 1-8.

7. OSTROWSKI, D., POZNIAK-KOSZALKA, I., KOSZALKA, L., & KASPRZAK, A. (2015). Comparative Analysis of the Algorithms for Pathfinding in GPS Systems, Proceedings of The Fourteenth International Conference on Networks (ICN 2015), pp. 102-108
8. PEREZ, D., LIU, J., KHALIFA, A. A., GAINA, R. D., TOGELIUS, J. & LUCAS, S. M. (2019). General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms, in IEEE Transactions on Games, 11(3),195-214.
9. MÜLLER V.C. & BOSTROM N. (2016). Future Progress in Artificial Intelligence: A Survey of Expert Opinion. In: Müller V. (eds) Fundamental Issues of Artificial Intelligence. Synthese Library (Studies in Epistemology, Logic, Methodology, and Philosophy of Science), 376. Springer, Cham, 2016
10. NOORI A. & MORADI F. (2015). Simulation and Comparison of Efficiency in Pathfinding algorithms in Games. Ciência e Natura, 37, 230-238
11. SAZAKI, Y., PRIMANITA, A., M. & SYAHROYNI, M. (2018). Pathfinding car racing game using dynamic pathfinding algorithm and algorithm A∗, Proceedings of The 3rd Int. Conf. Wirel. Telemat (ICWT 2017), pp. 164-169
12. YANNAKAKIS, G.N. & TOGELIUS, J., (2018). Artificial Intelligence and Games, Springer International Publishing, Switzerland