Research Article

The Fabrique: A Pathfinding Algorithm in a Mobile Game Developed Using Construct 3*

[1,*]Ruby Kusumawardhani, [2]Diah Arifah Prastiningtiyas, [3]Chaulina Alfianti Oktavia

[1,2,3] Department of Informatics, Sekolah Tinggi Informatika & Komputer Indonesia, Indonesia

[*] Corresponding Author: hi.rubywardhani@gmail.com

Abstract: The rapid growth of the digital game industry, particularly on mobile platforms, has driven the development of algorithms to enhance gameplay quality and player experience. The A* algorithm is a widely used pathfinding method for controlling the movement of non-playable characters (NPCs) in games. This study aims to evaluate the implementation of the A* algorithm in The Fabrique, a mobile game developed using Construct 3, a 2D game development engine. Testing was conducted across various path and obstacle scenarios. The results indicate that the A* algorithm delivers fast computation time and optimal pathfinding for short-distance navigation. In the medium to high obstacle scenarios, the algorithm maintained good performance with only minimal increases in processing time. The implementation of the A* algorithm in The Fabrique proved effective, contributing to a more dynamic and interactive gameplay experience. With an average user satisfaction rate of 81.94%, the algorithm demonstrates not only technical efficiency but also strong user acceptance.

Keywords: A-Star Algorithm, Construct 3, Pathfinding, NPC (Non-Playable Character), Game Development.

Citation: Kusumawardhani, R., Prastiningtiyas, D. A., & Oktavia, C. A. (2025). The Fabrique: A Pathfinding Algorithm in a Mobile Game Developed Using Construct 3. *Iota*, 5(2). https://doi.org/10.31763/iota.v5i2.915

Academic Editor: Adi, P.D.P

Received: April 06, 2025

Accepted: April 17, 2025

Published: May 12, 2025

1. Introduction

The development of the digital game industry has been ongoing since the 1950s and continues to evolve today [1]. With advancements in mobile device technology, mobile-based games have become increasingly popular, encouraging developers to create engaging and interactive gaming experiences [2]. Algorithms play a critical role in game development, serving as the foundation for various functions and mechanisms that ensure smooth gameplay and enhance player experience [3].

Pathfinding methods are among the algorithms commonly used in game development to manage the movement of non-playable characters (NPCs) [4]. The primary objective of these methods is to determine the most efficient route to a target, thereby contributing to more engaging and dynamic gameplay [5]. The A* algorithm is one of the most widely used pathfinding algorithms, combining heuristic approaches and graph search techniques to identify the shortest path [6].

The A* algorithm is selected for its strength in balancing pathfinding accuracy and computational efficiency, particularly for mobile platforms with limited resources [7]. Compared to other algorithms such as Dijkstra or Breadth-First Search (BFS), A* offers superior search efficiency while still maintaining optimal path accuracy [8][9]. Other artificial intelligence algorithms, such as Naive Bayes, have also been applied in decision-making systems within information technology, highlighting the importance of selecting algorithms that align with system requirements [10]. Furthermore, A* supports dynamic NPC movement in complex and variable environments, making it well-suited for games featuring diverse levels and obstacles [11].

Construct 3 is a 2D game development engine that enables rapid and user-friendly game creation without requiring extensive programming knowledge [12]. The platform offers a range of features, including a visual editor, animation support, vector graphics, easy asset management, and compatibility with mobile-based game development [13].

In the first quarter of 2023, the gaming industry demonstrated significant growth, with global players spending approximately USD 1.64 billion and downloading nearly 1.2 billion games each week [14]. The mobile gaming sector continues to thrive due to its high accessibility, presenting developers with new opportunities to reach wider audiences and drive innovation [15]. Moreover, other intelligent computing methods, such as the Fuzzy Sugeno algorithm, have been used in hardware selection decision-making, illustrating the flexibility of such approaches in various contexts—including potential applications in game logic [16].

This study focuses on the development of an RPG (Role-Playing Game) genre game that enables freedom of movement and exploration [17]. The game, titled The Fabrique, follows the story of the main character, Brow Lee, who explores the capital city after completing his education. In the game, the player must guide the character to a destination while avoiding proximity or contact with NPCs, utilizing the A* pathfinding algorithm [2].

The objective of this study is to evaluate the A* pathfinding method in the development of The Fabrique [5]. The focus is on the implementation of the algorithm for path detection toward a target, emphasizing optimal decision-making in dynamic environments by incorporating various criteria [18]. The study aims to strengthen the foundational research on game development involving dynamic and interactive NPC movement [1]. The novelty of this study lies in the application of the A* algorithm in a mobile-based RPG game developed using Construct 3—an engine that has not been widely explored in the context of pathfinding [12][13]. Previous studies have predominantly utilized Unity or desktop platforms [5], thus this research contributes to the expansion of heuristic pathfinding implementation in mobile gaming environments with constrained resources [11].

2. Literature Review

This section reviews the theoretical and empirical foundations underlying the development of *The Fabrique*, particularly the implementation of the A* pathfinding algorithm to support Non-Playable Character (NPC) behavior in a Construct 3-based game. The review focuses on three main aspects: pathfinding algorithms in-game navigation systems, game development technology using Construct 3, and the challenges and advantages of applying artificial intelligence (AI) in mobile-based games. The purpose of this discussion is to contextualize the significance of integrating intelligent algorithms with visual interfaces in modern game development, to enhance efficiency, engagement, and overall user experience.

2.1 Pathfinding

Pathfinding refers to the process of determining the shortest path between two points in a defined space. In the context of games, this algorithm is utilized to enable NPCs to navigate the game environment in a logical and goal-oriented manner. The A* (A-Star) algorithm is among the most widely adopted pathfinding techniques due to its efficiency in identifying optimal paths, based on the evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ denotes the actual cost from the start node and $h(n)$ represents the estimated cost to the goal node [18][19].

2.1.1 Comparison of Pathfinding Algorithms

The choice of pathfinding algorithm significantly impacts the performance of NPC navigation, especially on mobile platforms with limited computational resources. Commonly used algorithms include Breadth-First Search (BFS), Dijkstra's algorithm, and A*. The A* algorithm is particularly noted for its efficiency, as it combines the exhaustive nature of Dijkstra's algorithm with heuristic guidance that accelerates the search process [8].

Compared to Dijkstra's algorithm, A* offers superior performance in real-time game scenarios, as it selectively evaluates promising nodes based on the $f(n)$ function, rather than exhaustively examining all possibilities. A study by Wu et al. (2020) demonstrated that A* achieved up to 30% greater time efficiency than Dijkstra's algorithm in grid-based NPC navigation simulations [18]. Meanwhile, BFS, being an uninformed algorithm, explores all potential paths indiscriminately without regard to distance or cost, making it less efficient in complex game environments [8]. Algorithms lacking structured pathfinding—such as those using random or static movement—are generally unable to respond to dynamic environments, resulting in non-adaptive NPCs and diminished gameplay quality [3]. This adaptability makes A* particularly well-suited for scenarios where responsiveness and efficiency are critical, such as in fast-paced, dynamically changing game worlds.

Table 1. Comparison of Pathfinding Algorithms

Algorithm	Optimal Path	Heuristic	Time Efficiency	Advantages	Limitations
A*	Yes	Yes	High	Fast, accurate, adaptive	Requires well-designed heuristic
Dijkstra	Yes	No	Moderate	Always find the optimal path	Explores all nodes, slower in large maps
BFS	Yes (in unweighted graphs)	No	Low	Simple to implement	Inefficient in large or weighted graphs
No Algorithm	No	No	Very Low	Easiest to implement	Random or unrealistic NPC behavior

Accordingly, the A* algorithm was selected for *The Fabrique* due to its optimal balance between accuracy and efficiency, particularly suited for mobile platforms with limited resources [6].

2.2 Non-Playable Character (NPC)

Non-playable characters (NPCs) refer to game characters controlled by the system rather than by players. These characters often serve narrative roles, vendors, enemies, or quest-givers. NPCs enhance the depth and complexity of the game world by offering diverse interactions to players. Their presence is essential for enabling meaningful exchanges between the player character and other game entities, thereby contributing to a more immersive gameplay experience [1].

2.3 Construct 3

Construct 3 is a browser-based game engine that utilizes a visual programming system through event sheets. It supports the integration of plugins such as Pathfinding Behavior and Line of Sight, allowing developers to implement automatic navigation and player detection without traditional coding [12]. Moreover, Construct 3 enables direct export to mobile platforms and provides an asset library to accelerate the development process. The use of such visual engines is especially beneficial for novice developers aiming to focus on game logic without being burdened by programming complexity.

2.4 *Artificial Intelligence*

The implementation of artificial intelligence, particularly in the form of pathfinding for NPCs, poses several technical challenges, including tilemap configuration, obstacle avoidance, and real-time path computation optimization [6]. Nonetheless, the benefits are considerable, including enhanced gameplay quality, more immersive player interactions, and balanced game difficulty levels. Empirical evidence suggests that games incorporating A*-based AI-driven NPCs experience improved user retention. In the context of *The Fabrique*, such implementation facilitates new challenges and contributes to delivering a more realistic and engaging user experience [20].

3. Conceptual Framework

This section outlines the conceptual framework of *The Fabrique*, a game that incorporates the A* pathfinding algorithm to regulate the movement of non-playable characters (NPCs). The framework is designed to address both technical and operational aspects in the development of an intelligent NPC system capable of navigating the game environment efficiently and responsively. The discussion encompasses the game system architecture, the logic of the A* algorithm, the interaction mechanisms between NPCs, the environment, and the player, as well as visual implementation using Construct 3. This provides a structured approach to understanding how pathfinding logic is applied within a visual game engine and how the system operates in the context of mobile-based gameplay.

3.1 *System Design*

3.1.1 *Global Scheme*

The global scheme describes the overall system structure, including tilemap layout, initial NPC positions, navigation paths, and player interaction points. The game is developed using Construct 3, a platform that supports the integration of visual components such as *Pathfinding Behavior* and *Line of Sight*. Each level is designed with varied obstacles that influence the optimal navigation paths available to NPCs, ensuring that pursuit behaviors can dynamically adapt to player movements.

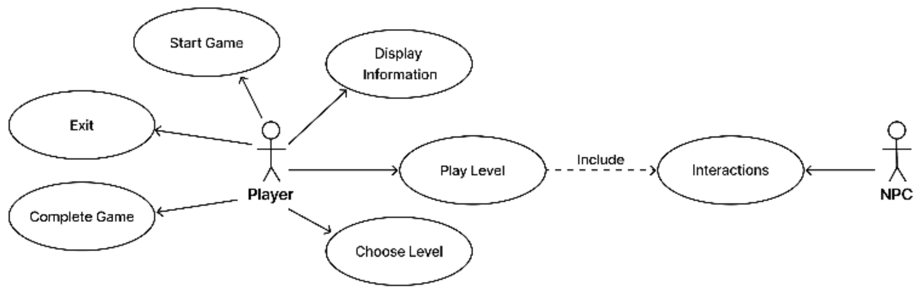


Figure 1. Use Case Diagram

3.2 *System Architecture*

The system architecture includes the configuration of the *event sheet*, *instance variables*, and condition-based logic systems. These components govern the transition of NPC states from passive to active (chase mode). The evaluation of the function f(n) = g(n) + h(n) is executed via the *Pathfinding Behavior*, which automatically assesses the most promising nodes until the target is reached. The system also utilizes *tilemap obstacles* as parameters to define navigable and non-navigable areas for the NPCs.

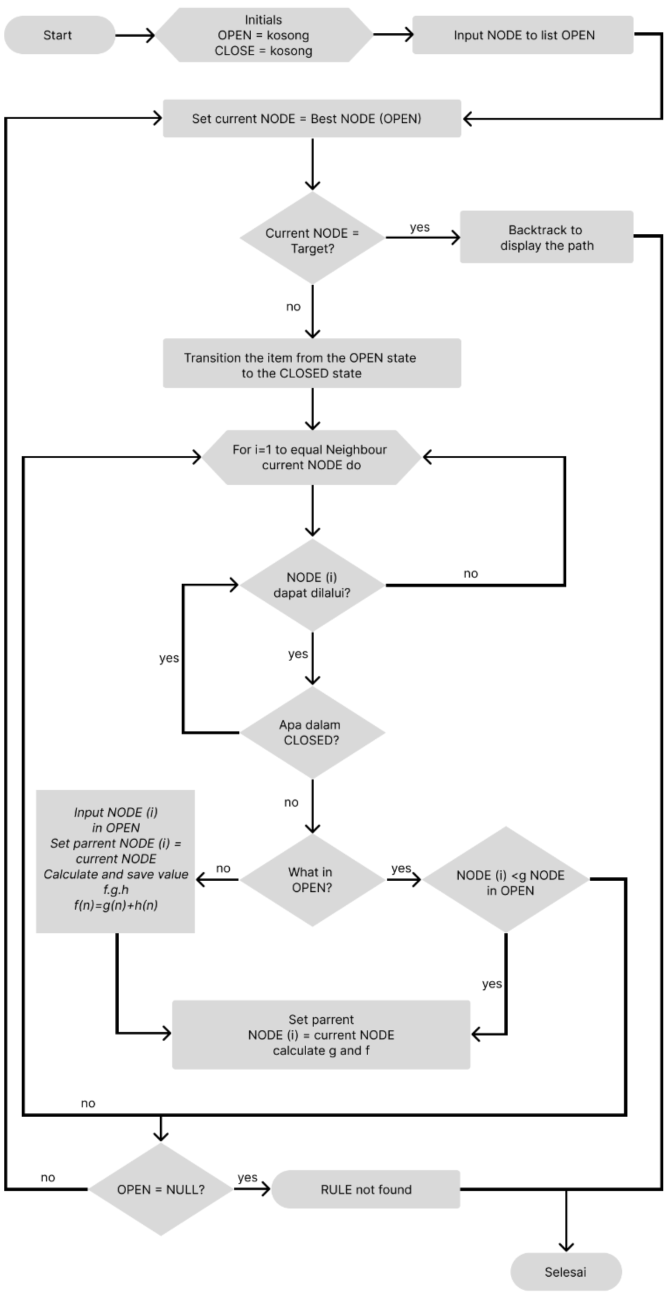

Start
Initials
OPEN = kosong
CLOSE = kosong
Input NODE to list OPEN
Set current NODE = Best NODE (OPEN)
Current NODE = Target?
yes
Backtrack to display the path
no
Transition the item from the OPEN state to the CLOSED state
For i=1 to equal Neighbour current NODE do
NODE (i) dapat dilalui?
no
yes
yes
Apa dalam CLOSED?
no
Input NODE (i) in OPEN Set parrent NODE (i) = current NODE Calculate and save value f.g.h f(n)=g(n)+h(n)
no
What in OPEN?
yes
NODE (i) <g NODE in OPEN
yes
Set parrent NODE (i) = current NODE calculate g and f
no
no
OPEN = NULL?
yes
RULE not found
Selesai

Figure 2. A* Algorithm Flowchart

3.3 *Intelligent Interaction Model*

The NPC remains in a random movement pattern when the player is not detected by the visual system (*Line of Sight*). However, once the player enters the detection radius without being obstructed by environmental obstacles, the NPC automatically calculates the shortest path to the player's position using the A* algorithm. The behavioral state transitions are regulated by the *event sheet,* which evaluates object distance parameters and visual status conditions. This model is designed to enable rapid and adaptive NPC responses to changes in player positioning.

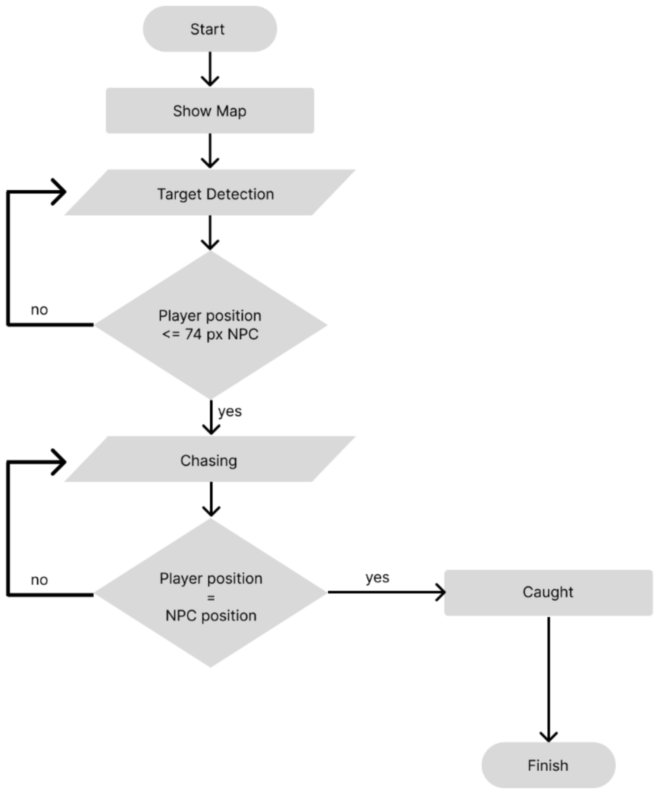


Figure 3. NPC Flowchart

3.4 Realtime Monitoring

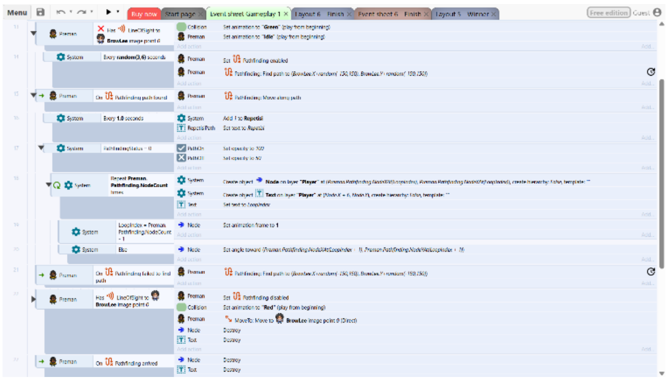
Throughout gameplay, the system records data such as NPC travel time to the player, the number of nodes traversed, and the effectiveness of selected paths across different levels. Data collection is conducted manually via direct observation and debugging logs from Construct 3. This information is utilized to assess the effectiveness of the A* algorithm's integration of level design complexity and navigation demands.

4. Technical Specification

The technical specifications for *The Fabrique* game are developed using Construct 3, with a primary focus on the application of the A* pathfinding algorithm, the structure of game assets, and the performance of NPC logic during gameplay. These specifications aim to explain how the technical design supports the dynamic behavior of NPC characters, creating an immersive and adaptive gameplay experience. Additionally, the event-driven system implemented allows NPCs to respond to various game conditions in real time. This approach not only enhances interaction flexibility but also strengthens the overall interactive narrative of the game.

4.1 Event Sheet Implementation

The movement logic of NPCs is controlled through the *event sheet* in Construct 3. In the initial state, NPCs move randomly when the player is not detected. Once the player enters the detection area defined by the *Line of Sight* feature, the A* pathfinding algorithm is activated to calculate the shortest path to the player's location.



```
Menu   Buy now | Start page | Event sheet Gameplay 1 | Layout 6 - Finish | Event sheet 6 - Finish | Layout 5 - Winner     Free edition  Guest

Preman    Has LineOfSight to               Collision   Set animation to "Green" (play from beginning)
          BrowLee image point 0            Preman      Set animation to "Idle" (play from beginning)
  System    Every random(3,6) seconds      Preman      Set Pathfinding enabled
                                           Preman      Pathfinding: Find path to (BrowLee.X+random(-150,150), BrowLee.Y+random(-150,150))
Preman    On Pathfinding path found        Preman      Pathfinding: Move along path
  System    Every 1.0 seconds              System      Add 1 to Repetisi
                                           RepetisiPath Set text to Repetisi
  System    PathfindingStatus = 0          PathOn      Set opacity to 100
                                           PathOff     Set opacity to 50
    System  Repeat Preman.                 System      Create object Node on layer "Player" at (Preman.Pathfinding.NodeXAt(loopindex), Preman.Pathfinding.NodeYAt(loopindex)), create hierarchy: false, template: ""
            Pathfinding.NodeCount          System      Create object Text on layer "Player" at (Node.X + 6, Node.Y), create hierarchy: false, template: ""
            times                          Text        Set text to LoopIndex
      System  LoopIndex = Preman.          Node        Set animation frame to 1
              Pathfinding.NodeCount
              - 1
      System  Else                         Node        Set angle toward (Preman.Pathfinding.NodeXAt(LoopIndex + 1), Preman.Pathfinding.NodeYAt(LoopIndex + 1))
Preman    On Pathfinding failed to find    Preman      Pathfinding: Find path to (BrowLee.X+random(-150,150), BrowLee.Y+random(-150,150))
          path
Preman    Has LineOfSight to               Preman      Set Pathfinding disabled
          BrowLee image point 0            Collision   Set animation to "Red" (play from beginning)
                                           Preman      MoveTo: Move to BrowLee image point 0 (Direct)
                                           Node        Destroy
                                           Text        Destroy
Preman    On Pathfinding arrived           Node        Destroy
                                           Text        Destroy
```

Figure 4. NPC Event Sheet

The A* algorithm uses the formula as the equation 1.

$$f(n) = g(n) + h(n) \tag{1}$$

g(n) = actual distance traveled from the starting point
h(n) = estimated distance to the target
f(n) = total path cost

The system automatically updates the path when the player's position changes, ensuring that NPCs always follow the most efficient route. This process is visualized through a flowchart of the A* algorithm, which illustrates the node selection steps and the NPC movement direction in real time.

4.2 Game Assets

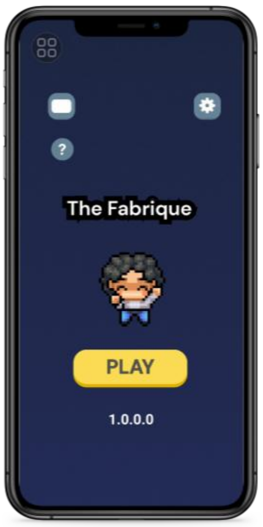


Figure 5. Main Menu Interface

The structure of game assets includes the main menu interface, the gameplay screen, and conditions when interactions between NPCs and players occur. The initial game interface is simple and intuitive, allowing players to easily navigate to the level selection screen. This layout ensures accessibility for users of all experience levels, enhancing the overall user onboarding process.

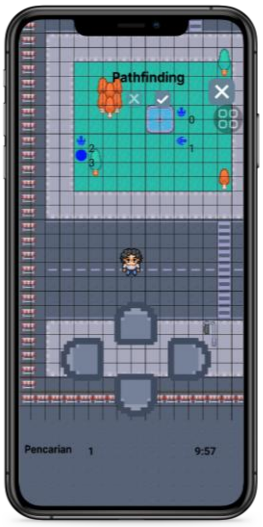


Figure 6. Gameplay Interface

This screen shows the gameplay situation when the player is not yet detected by the NPC. In this condition, the NPC moves randomly according to the logic defined in the event sheet. This stage is important for testing the difference in NPC behavior when the A* pathfinding algorithm is not activated, compared to when the NPC begins to chase the player. Such baseline behavior provides a clear contrast to demonstrate the effectiveness of intelligent pathfinding when detection occurs.

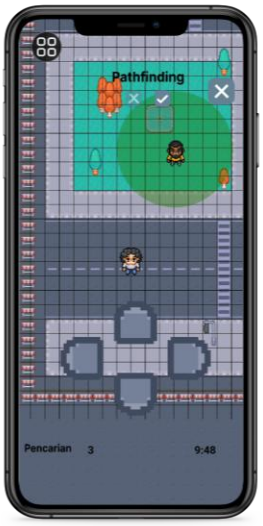


Figure 7. NPC Search Interface

Once the player enters the line of sight radius, the A* pathfinding algorithm is activated, and the NPC begins actively chasing the player. This display shows the dynamic interaction supported by the intelligent navigation system based on the algorithm, which is optimally implemented on the mobile platform via Construct 3. The transition highlights the algorithm's capability to adapt in real time to player movements within complex environments.

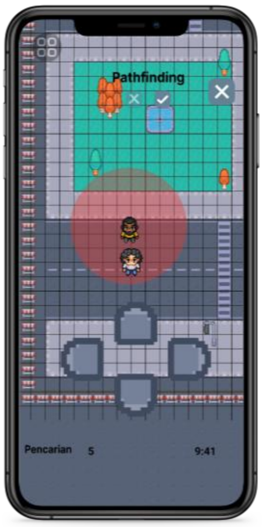


Figure 8. NPC Detection Interface

Once the player enters the line of sight radius, the A* pathfinding algorithm is activated, and the NPC begins actively chasing the player. This display shows the dynamic interaction supported by the intelligent navigation system based on the algorithm, which is optimally implemented on the mobile platform via Construct 3. This implementation exemplifies the smooth integration of AI navigation with mobile game mechanics for immersive gameplay.

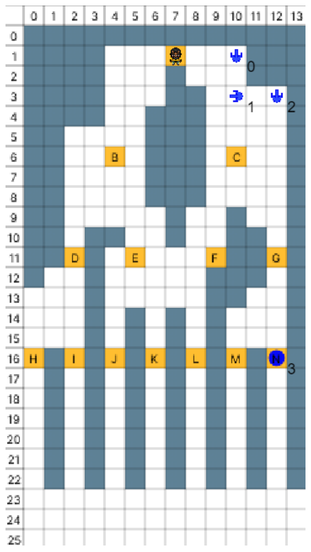


Figure 9. Map Simulation

The game map consists of a 13×25 grid, used as a simulation for NPC movement relative to the player's position. Each grid unit represents a navigation unit in the A* algorithm, enabling systematic testing of the chase and avoidance logic. This structured layout also facilitates debugging and performance optimization during development.

4.3 Testing

4.3.1 Functional Testing

Functional testing is conducted to verify that each component of *The Fabrique* game operates as intended. Each component is tested with two possible outcomes: **success** or **failure**.

Table 2. Functional Testing

No	Test Component	Testing	Notes
1	Splash Screen	Displays t*he Fabrique* logo splash screen	Successful
2	Main menu	Displays menu buttons, information, and play button	Successful
3	Information Menu	Displays game instructions	Successful
4	Settings Menu	Displays mute and unmute audio buttons	Successful
5	Main Character	Displays Brow Lee's character	Successful
6	NPC Character	Displays Preman and Senior characters	Successful
7	Level 1	Displays level 1 map leading to the dormitory	Successful
8	Level 2	Displays level 2 map leading to the workplace	Successful
9	Level 3	Displays level 3 map leading to the boss's room	Successful
10	Level 4	Displays level 4 map leading back to the dormitory	Successful
11	Level 5	Displays level 5 map leading to the police station	Successful
12	Win Layout	Displays win layout screen	Successful
13	Lose Layout	Displays lose layout screen	Successful

4.4 NPC Behavior Testing

NPC behavior testing aims to evaluate the detection capabilities and response of NPCs to the player's presence. The initial detection system has a radius of 74 pixels, forming a full circle (360 degrees). The NPC will chase the player if the player's position is within the detection radius (≤ 74 px) and return to patrolling mode if the player's position is outside this radius (> 74 px). If the player's position coincides with the NPC's position (x = x_NPC, y = y_NPC), the status is considered "caught."

Moreover, to test the sensitivity of the system, the detection radius is increased by 20 pixels at each subsequent test level.

Table 3. NPC Behavior Testing

Experiment	State	Condition	Action	Notes
1	Patrolling	Player (x,y) ≤ **NPC (20,20)**	Chasing	Successful
	Chasing	Player (x,y) > **NPC (20,20)**	Patrolling	Successful
	Chasing	Player (x,y) = **NPC (20,20)**	Caught	Successful
2	Patrolling	Player (x,y) ≤ **NPC (40,40)**	Chasing	Successful
	Chasing	Player (x,y) > **NPC (40,40)**	Patrolling	Successful
	Chasing	Player (x,y) = **NPC (40,40)**	Caught	Successful
3	Patrolling	Player (x,y) ≤ **NPC (60,60)**	Chasing	Successful
	Chasing	Player (x,y) > **NPC (60,60)**	Patrolling	Successful
	Chasing	Player (x,y) = **NPC (60,60)**	Caught	Successful
4	Patrolling	Player (x,y) ≤ **NPC (80,80)**	Chasing	Successful

Experiment	State	Condition	Action	Notes
	Chasing	Player (x,y) > **NPC (80,80)**	Patrolling	Successful
	Chasing	Player (x,y) = **NPC (80,80)**	Caught	Successful
5	Patrolling	Player (x,y) ≤ **NPC (100,100)**	Chasing	Successful
	Chasing	Player (x,y) > **NPC (100,100)**	Patrolling	Successful
	Chasing	Player (x,y) = **NPC (100,100)**	Caught	Successful

4.5 A Pathfinding Testing*

This testing aims to evaluate the effectiveness and efficiency of the A* algorithm in various map scenarios, including conditions with short paths, and long paths, as well as maps with simple to complex obstacles. The parameters tested include the number of nodes explored (open list), the number of paths found, and computation time.

Table 4. A* Pathfinding Testing

Level	Type of Obstacle	Trial	Start (x,y)	Goal (x,y)	Open list	Path	Time
1	Short Path	3	(418,571)	(91,740)	3	2	8 s
		1	(418,571)	(91,740)	3	2	15 s
		2	(418,571)	(91,740)	5	4	26 s
2	Long Path	1	(626,243)	(91,740)	2	3	18 s
		2	(626,243)	(91,740)	2	4	21 s
		3	(626,243)	(91,740)	2	7	34 s
3	Simple Obstacles	3	(412,200)	(123,710)	6	3	15 s
		1	(412,200)	(123,710)	3	3	18 s
		2	(412,200)	(123,710)	7	6	30 s
4	Moderate Obstacles	2	(297,73)	(91,740)	9	3	14 s
		3	(297,73)	(91,740)	6	4	24 s
		1	(297,73)	(91,740)	5	5	25 s
5	Multi-Agent	2	(116,306), (582, 308)	(91,740)	2	2	8 s
		3	(116,306), (582, 308)	(91,740)	2	2	9 s
		1	(116,306), (582, 308)	(91,740)	2	2	10 s

The results demonstrate that the A* algorithm is capable of generating optimal paths with an average computation time ranging from 5 to 9 seconds per search, at a maximum movement speed of 60 pixels/second.

4.6 *User Experience Evaluation*

A total of 30 participants took part in the user experience (UX) testing by responding to 12 statements related to game usability and functionality. Respondents rated each statement using the following scale:

1 = Strongly Disagree (SD)

2 = Disagree (D)

3 = Agree (A)

4 = Strongly Agree (SA)

Table 5. Questionnaire Statements

No	Statement
1	The main menu page is easy to understand
2	The level pages are distinguishable.
3	The information provided is easy to comprehend
4	The gangster NPC can follow the player accurately.
5	The senior NPC can follow the player accurately.
6	Level 1 is playable without issues.
7	Level 2 is playable without issues.
8	Level 3 is playable without issues.
9	Level 4 is playable without issues.
10	Level 5 is playable without issues.
11	The player's movement speed is appropriate.
12	The game is extremely difficult to play.

Moreover, the Score Calculation Formula is shown in Equation 2.

$$Y = \left(\frac{\sum N.R}{Ideal\ Scores}\right) \text{ x } 100\ \% \tag{2}$$

Where:
N = Rating scale (1–4)
R = Number of respondents selecting a particular scale value
Σ = Summation
Skor Ideal = Total number of respondents × highest score

Table 6. Respondents' Questionnaire Results

No	Criterion	*N*	*R*	*N.R*	Σ(*N.R*)	Ideal Score	*Y*
1	SD	1	0	0	97	120	80,83%
	D	2	1	2			
	A	3	21	63			
	SA	4	8	32			
2	SD	1	0	0	99	120	82,5%
	D	2	1	2			
	A	3	19	57			
	SA	4	10	40			
3	SD	1	0	0	99	120	82,5%
	D	2	2	4			
	A	3	17	51			
	SA	4	11	44			
4	SD	1	0	0	99	120	82,5%
	D	2	2	4			
	A	3	17	51			
	SA	4	11	44			
5	SD	1	0	0	98	120	81,67%
	D	2	2	4			
	A	3	22	66			
	SA	4	7	28			
6	SD	1	0	0	100	120	83,33%
	D	2	2	4			
	A	3	16	48			
	SA	4	12	48			
7	SD	1	0	0	101	120	84,17%
	D	2	0	0			
	A	3	19	57			
	SA	4	11	44			

No	Criterion	*N*	*R*	*N.R*	Σ(*N.R*)	Ideal Score	*Y*
8	SD	1	0	0	100	120	83,33%
	D	2	1	2			
	A	3	18	54			
	SA	4	11	44			
9	SD	1	0	0	97	120	80,83%
	D	2	1	2			
	A	3	21	63			
	SA	4	8	32			
10	SD	1	0	0	100	120	83,33%
	D	2	1	2			
	A	3	18	54			
	SA	4	11	44			
11	SD	1	0	0	95	120	79,17%
	D	2	5	10			
	A	3	15	45			
	SA	4	10	40			
12	SD	4	13	52	95	120	79,17%
	D	3	10	30			
	A	2	6	12			
	SA	1	1	1			

Based on the results of the questionnaire distributed to 30 respondents consisting of 12 statements, the percentage for each statement was obtained. Subsequently, the overall average was calculated by summing the percentage of each statement and dividing it by the total number of statements. The calculation result showed an average of 81.94%. The following is the graph of the A* Pathfinding testing using five types of obstacles with different levels of difficulty.

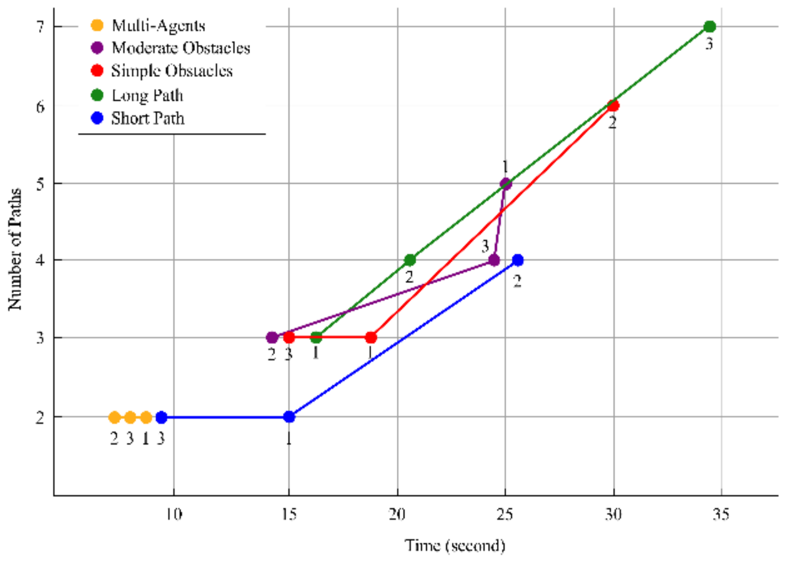


Figure 10. A Pathfinding Testing Graph*

In the condition of a short path without obstacles, such as at the blue point, the A* Pathfinding algorithm showed low computation time, ranging from 8–15 seconds, with a total of 2 paths in Level 1 and 3, and 4 in Level 2. Meanwhile, under longer path conditions (green point), the computation time increased to 34 seconds with 7 paths in Level 3, 4 in Level 2, and 3 in Level 1.

In conditions with simple obstacles (redpoint), computation time ranged from 14–30 seconds with a total of 3 paths in Level 1 and 3, and 6 in Level 2. Meanwhile, in medium-obstacle conditions (purple point), this algorithm recorded a time between 14–25 seconds with 5 paths in Level 1, 3 in Level 2, and 4 in Level 3. In multi-agent path conditions, computation time was relatively low, i.e., 8–10 seconds, with the same number of paths, 2, at all levels. Although there was an increase in computation time in more complex paths, the performance of the A* Pathfinding algorithm remained optimal and within reasonable limits.

5. Conclusions

Based on the results of testing and analysis, the implementation of the A* Pathfinding algorithm in the game The Fabrique showed good performance in various scenarios. In short paths without obstacles, the computation time ranged from 8–15 seconds with 2–4 paths, while in long paths, the computation time could reach up to 34 seconds with 3–7 paths. In simple obstacles, the algorithm showed a computation time of 14–30 seconds with 3–6 paths, and in medium obstacles, the computation time was 14–25 seconds with 3–5 paths. In multi-agent scenarios, computation time was stable at around 8–10 seconds with a consistent number of 2 paths. Overall, the A* Pathfinding algorithm has been tested and has demonstrated good performance in various conditions and level scenarios. With an average user satisfaction of 81.94%, the algorithm is not only technically efficient but also well-received by users.

As a follow-up to this research, several developments can be carried out in the future. One of them is to apply the Pathfinding algorithm to more complex game scenarios, such as environments with non-grid maps or dynamically designed levels. Further research can also explore the integration of the A* algorithm with other approaches, such as machine learning, to produce NPC behavior that is more adaptive to changing game situations. In addition, the developed navigation system can be tested in multiplayer game modes or in difficulty-level designs that adjust to the player's abilities. With these developments, the pathfinding system is expected to provide a more realistic and challenging gaming experience.

Acknowledgments: This research was conducted by Informatics Study Program students as part of the Final Project, with the support of the Sekolah Tinggi Informatika dan Komputer Indonesia and under the guidance of Ms. Diah Arifah Prastiningtiyas, S.Kom., M.T., and Ms. Chaulina Alfianti Oktavia, S.Kom., M.T. This research is part of an academic activity fully supported by the institution in completing the undergraduate degree.

Author contributions: The authors are responsible for building Conceptualization, Methodology, analysis, investigation, data curation, writing—original draft preparation, writing—review and editing, visualization, supervision of project administration, funding acquisition, and have read and agreed to the published version of the manuscript.

Funding: The study was conducted without any financial support from external sources.

Availability of data and Materials: All data are available from the authors.

Conflicts of Interest: The authors declare no conflict of interest.

Additional Information: No Additional Information from the authors.

References

[1] Kay, M., & Powley, E. J. (2018, August). The effect of visualising NPC pathfinding on player exploration. In Proceedings of the 13th international conference on the foundations of digital games (pp. 1-6).

[2] Octavian, F., & Hermawan, L. (2023). Penerapan Algoritma Pathfinding A* dalam Game Dual Legacy berbasis Android. Jurnal Buana Informatika, 14(01), 20-29.

[3] Mutaqin, G., Fadilah, J. N., & Nugroho, F. (2021). Implementasi Metode Path Finding dengan Penerapan Algoritma A-Star untuk Mencari Jalur Terpendek pada Game "Jumrah Launch Story".

[4] Junanto, E., Osmond, A. B., & Ansori, A. S. R. (2020). Membuat Pergerakan Non-Player Character (Npc) Menggunakan Metode A Star. eProceedings of Engineering, 7(1).

[5] Agung, E. (2022). Implementasi metode pathfinding dengan algoritma a* pada game rogue-like menggunakan unity. Indonesian Journal on Computing (Indo-JC), 7(3), 81-94.

[6] Sujaka, T. T., Abd Latif, K., Hadi, S., Hasbullah, H., & Hammad, R. (2022). A* Pathfinding Applications in Two-Dimensional AI Video Games. INAJEEE (Indonesian Journal of Electrical and Electronics Engineering), 5(1), 25-29.

[7] Pratama, J. Y. (2024). Analisis Perbandingan Algoritma Dijkstra dan A-Star dalam Menentukan Rute Terpendek. JIMU: Jurnal Ilmiah Multidisipliner, 2(03), 668-682.

[8] Aditiya, V., & Herdiana, B. (2021). Analisis Perbandingan Algoritma Breadth First Search (BFS) dan Algoritma A. Telekontran: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan, 9(2), 139-153.

[9] Miyombo, M. E., Liu, Y. K., Mulenga, C. M., Siamulonga, A., Kabanda, M. C., Shaba, P., ... & Ayodeji, A. (2024). Optimal path planning in a real-world radioactive environment: A comparative study of A-star and Dijkstra algorithms. Nuclear Engineering and Design, 420, 113039.

[10] Wicaksono, F. Y. A., Anita, A., & Oktavia, C. A. (2019). Sistem Penunjang Keputusan Penentuan Dosen Pembimbing Menggunakan Algoritma Naive Bayes Studi Kasus STIKI Malang. J-INTECH (Journal of Information and Technology), 7(02), 109-114.

[11] Al-qerem, A., Ali, A. M., & Izneid, B. A. (2024). Investigating NPC Path Finding Behaviors with Navigation Mesh and Grid Map Techniques. Artificial Intelligence and Economic Sustainability in the Era of Industrial Revolution 5.0, 675-688.

[12] Permatasari, S., Asikin, M., & Dewi, N. R. (2022). MaTriG: Game edukasi matematika dengan construct 3. The Indonesian Journal of Computer Science, 11(1).

[13] Pratama, R. R., & Surahman, A. (2021). Perancangan Aplikasi Game Fighting 2 Dimensi Dengan Tema Karakter Nusantara Berbasis Android Menggunakan Construct 2. J. Inform. dan Rekayasa Perangkat Lunak, 1(2), 234-244.

[14] Newzoo. (2023). The global games market in 2023: A new era of growth. Newzoo. Retrieved from https://newzoo.com/resources/blog/last-looks-the-global-games-market-in-2023.

[15] SensorTower. (2023). Mobile game market growth and trends in Q1 2023. SensorTower. Retrieved from https://sensortower.com/blog/2023-game-market

[16] Diah Arifah, P., & Daniel Rudiaman, S. (2019). Implementasi Fuzzy Sugeno Dalam Pemilihan Processor.

[17] Wahyuni, S. N., & Andiyoko, C. (2018). Pembuatan Game Berbasis Pembelajaran Menggunakan RPG Maker MV. Jurnal Mantik Penusa, 2(2).

[18] Zhang, Cheng & Ao, Lei & Yang, Junsheng & Xie, Wenchuan. (2020). An Improved A* Algorithm Applying to Path Planning of Games. Journal of Physics: Conference Series. 1631. 012068. 10.1088/1742-6596/1631/1/012068.

[19] Pua, G. E. (2020). Komparasi Algoritma A Star, Dijkstra dan BFS Untuk Path Finding NPC Dalam Game/Geraldin Everdin Pua/57160405/Pembimbing: Richard Vinc N. Santoso.

[20] Pramono, S., & Verawati, I. (2024). Optimasi Performa Game Dugeon Escape Menggunakan Algoritma A-Star. Journal Automation Computer Information System, 4(2), 136-145.