Journal of Multiscale Material Informatics
ISSN 3047-5724

Research Article

Implementation of a Finite State Machine for Controlling NPC Customer Behavior in the 2D Game "Cooking Chaos"

Syakira Aulia Tasya[1*], Diny Syarifah Sany[2*]

[1]Universitas Suryakancana, Cianjur, West Java 43216, Indonesia
*Corresponding: syakiraauliaatasya1@email.com[1]; Dsy.sany@gmail.com[2]

Abstract

Non-Player Characters (NPC) are components in games that can be used to create interactions and gameplay dynamics. NPC behavior control is required to enable characters to respond appropriately to conditions occurring during gameplay. This study aims to implement a Finite State Machine (FSM) to control customer NPC behavior in the Unity-based 2D game Cooking Chaos. The implemented FSM consists of four states: Waiting, Angry, Chasing, and Attack. State transitions are influenced by the customer's patience system and interaction conditions with the player. When the patience time expires, the customer changes behavior from waiting to becoming angry, chasing, and potentially attacking the player. The implementation was evaluated using Black Box Testing, FSM Transition Testing, and Multi-Customer Testing. The Black Box Testing results showed that all 12 functional scenarios were valid. FSM Transition Testing on seven scenarios, each tested 10 times, resulted in 70 successful trials out of 70. Multi-Customer Testing demonstrated that the order, patience, and FSM mechanisms operated independently with up to three active customers simultaneously. The results indicate that the implemented FSM consistently controls customer NPC behavior transitions under the defined test scenarios and integrates the service mechanism with direct gameplay consequences for the player.

Keywords: 2D game, Finite state machine, NPC customer, Software testing, Unity.

Received: 21 July 2026 / **Revised:** 16 August 2026 / **Accepted:** 17 August 2026 / **Published:** 3 September 2026

1. INTRODUCTION

Games are interactive media that allow players to interact directly with the system and various objects within the game environment. In game development, Non-Player Characters (NPCs) are characters controlled by the system to create dynamic interactions and support gameplay. NPC behavior must be designed to respond appropriately to game conditions, thereby making the gaming experience more engaging and challenging.

One approach used to manage NPC behavior is the Finite State Machine (FSM). FSM is a state-based behavior control method that allows a character to exist in a specific state and transition to another when transition conditions are met. This state-based approach can model changes in object behavior within a game using a predefined set of states and transition rules [7]. In game AI development, this approach enables the construction of character behaviors based on game conditions [8].

Finite State Machines have been applied in various studies to control character behavior in games. Mandita and Jati [1] implemented FSMs for NPCs in the 3D game Virus Hunter. Syu'aibi et al. [2] applied FSMs in the 2D game Military Defence to control NPC behavior based on game conditions. FSMs have also been applied to NPCs in leadership simulation games, using state machine mechanisms to determine character behavior [3]. More recent research demonstrates the implementation of FSMs in retro arcade fighting games to manage enemy character behavior [4]. Other studies show that FSMs can map NPC behavior in role-playing games, allowing characters to alter their behavior based on predefined conditions [16].

However, these studies generally focus on controlling enemy NPCs that patrol, chase, or attack the player. Research regarding the application of FSMs to customer NPCs in cooking games remains relatively limited. Customer NPCs differ from enemy NPCs due to characteristics such as queuing mechanisms, food ordering processes, patience levels, player-provided service, and behavioral changes when service is not delivered within a specified time limit. These characteristics give the behavior control of customer NPCs a level of complexity distinct from FSM implementations in other game genres.

In this study, an FSM is applied to customer NPCs in the 2D game Cooking Chaos. Customer NPCs begin in a "Waiting" state, awaiting service from the player, who acts as the chef. A patience mechanism

serves as the trigger for behavioral changes when the player fails to provide service within the allotted time. These behavioral changes are implemented through four primary states, Waiting, Angry, Chasing, and Attack, allowing the NPCs to transition automatically based on game conditions.

The study's primary contribution is the application of an FSM to customer NPCs that goes beyond state transitions based solely on player proximity; it integrates order fulfillment, customer patience levels, emotional shifts to anger, player pursuit, and consequences (punishment) for failing to serve customers on time. This approach yields more dynamic NPC behavior, thereby enhancing game challenge and interactivity compared to FSM implementations focused exclusively on enemy behavior.

Cooking Chaos is a Unity-based 2D game in which the player takes on the role of a chef responsible for fulfilling customer orders. Each customer receives a random food order and has a 15-second patience window while waiting for service. Successfully serving the correct food earns the player extra money, whereas failing to provide service before the time limit expires causes the customer NPC to undergo a behavioral change dictated by the designed FSM mechanism.

Given this context, the study aims to implement an FSM to control customer NPC behavior in the Unity-based 2D game Cooking Chaos and to evaluate the alignment of state transitions with the system design. Evaluation is conducted using Black Box Testing, FSM Transition Testing, and Multi-Customer Testing to ensure that each NPC consistently executes behaviors appropriate to the game conditions.

2. METHODS

2.1. Research Stages

This research focuses on the application of Finite State Machine as a mechanism for controlling NPC customer behavior in the 2D Cooking Chaos game, which was developed using Unity. Research stages include problem identification, literature study, gameplay analysis, FSM design, FSM implementation, testing, and results analysis.

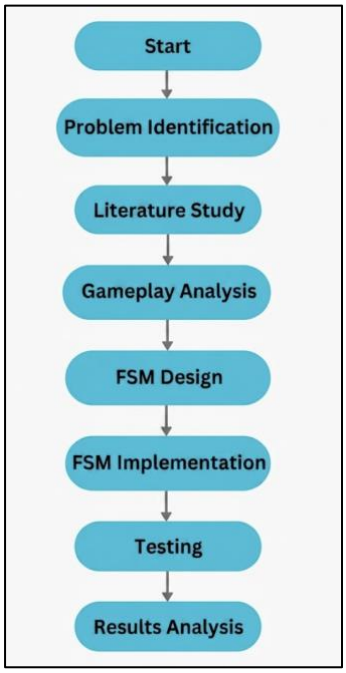

Figure 1. Research stages.

The problem identification stage is carried out to identify the need for controlling NPC customer behavior. Literature studies are used to study the FSM concept and previous research. Gameplay analysis identifies the mechanisms of customer appearance, order selection, patience, food service, player pursuit, attacks, and conditions for the end of the customer cycle. Next, four main states were designed, namely Waiting, Angry, Chasing, and Attack, then implemented using C# in Unity.

At the implementation stage, each state in the FSM is given parameters (Table1) that determine changes in NPC behavior. State Waiting has a patience time of 15 seconds. When the patience value reaches zero, the NPC moves to the Angry state for a few moments before entering the Chasing state. In the Chasing state, the NPC moves towards the player's position using a predetermined speed. If the NPC's distance to the player has reached the attack Distance value, the NPC enters the Attack state and reduces the player's Health Point (HP) by one point for each attack. These parameters are used to ensure that the transition process between states runs consistently throughout the game.

Table 1. Finite state machine implementation parameters for the customer NPC.

Parameter	Value	Description
Wait Time	15 s	The duration the customer is in the Waiting state before turning Angry
Angry Duration	3 s	The duration the customer is in the Angry state before moving to Chasing
Chase Duration	5 s	The duration of the customer chasing the player in the Chasing state
Chase Speed	4 m/s	Customer movement speed when chasing players
Attack Distance	0,5 unit	The distance between the customer and the player that triggers the move to the Attack state

A `wait time` value of 15 seconds serves as the customer's patience limit. When this timer reaches zero, the NPC transitions from the Waiting state to the Angry state. The Angry state lasts for 3 seconds before the NPC enters the Chasing state. In the Chasing state, the NPC moves toward the player's position at a `chaseSpeed` of 4 for a duration of 5 seconds. If the distance between the NPC and the player falls below 0.5 units, the NPC transitions to the Attack state and deals damage to the player. If the chase duration expires before the NPC reaches attack range, the NPC leaves the game arena. The testing phase consists of Black Box Testing, FSM Transition Testing, and Multi-Customer Testing. The final stage is analysis of the results to evaluate the suitability of the implementation to the design (Fig. 1).

2.2. Overview of the Cooking Chaos Game

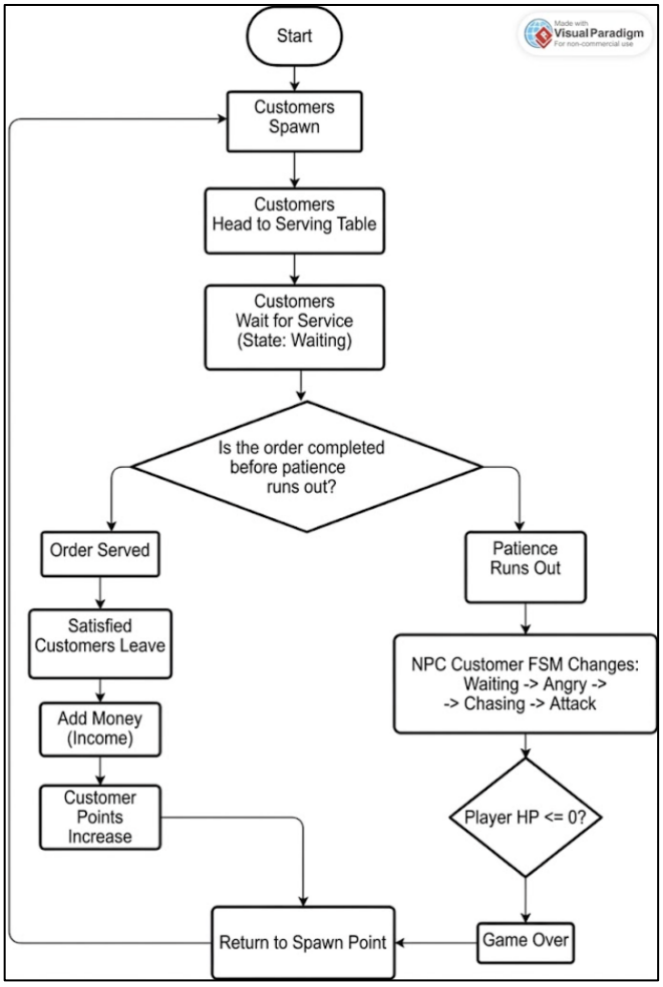

Figure 2. Cooking chaos gameplay flow.

Cooking Chaos is a 2D game with a food service theme set in a beach area (Fig. 2). Customers appear automatically via CustomerSpawner and are directed to the available CustomerPoint (Serving Table). The game provides a maximum of three CustomerPoints, so that a maximum of three customers can occupy the service area simultaneously. The limitation on the number of customers is part of the gameplay design to maintain the level of difficulty and ensure that the service area can still be managed by players. This limitation is a game design decision and not a limitation of the Finite State Machine method of handling a large number of NPCs simultaneously.

Once active, customers receive random orders via GenerateOrder() from three types of food, namely Shrimp, Coffee, and Coconut Ice. While waiting for service, the customer's patience value will continue to decrease. If the food provided matches the current Order, the player gets an additional 10,000, and the customer leaves the game arena. On the other hand, if the food provided does not match the order, the customer remains in the service area and waits until the patience time expires.

When the patience value reaches zero, the FSM changes the customer's behavior from the Waiting state to Angry. After the duration of the Angry state is over, the customer moves to the Chasing state and chases the player. If the distance between the customer and the player has met the attackDistance value, the customer moves to the Attack state and reduces the player's HP by one point through an attack mechanism in the form of a kick. If the player's HP reaches zero, the game enters the Game Over condition.

2.3. Finite State Machine Design

The FSM in this research has four main states (Fig. 3 and Table 2), namely Waiting, Angry, Chasing, and Attack. The initial state is Waiting. When currentTime reaches zero, the state changes to Angry. After the Angry duration is over, the state changes to Chasing. When the customer's distance to the player meets attackDistance, the state changes to Attack. If the chase duration runs out before the customer reaches the player, the customer leaves the game arena. Exit is a terminal condition and not an explicit state in the FSM enum.

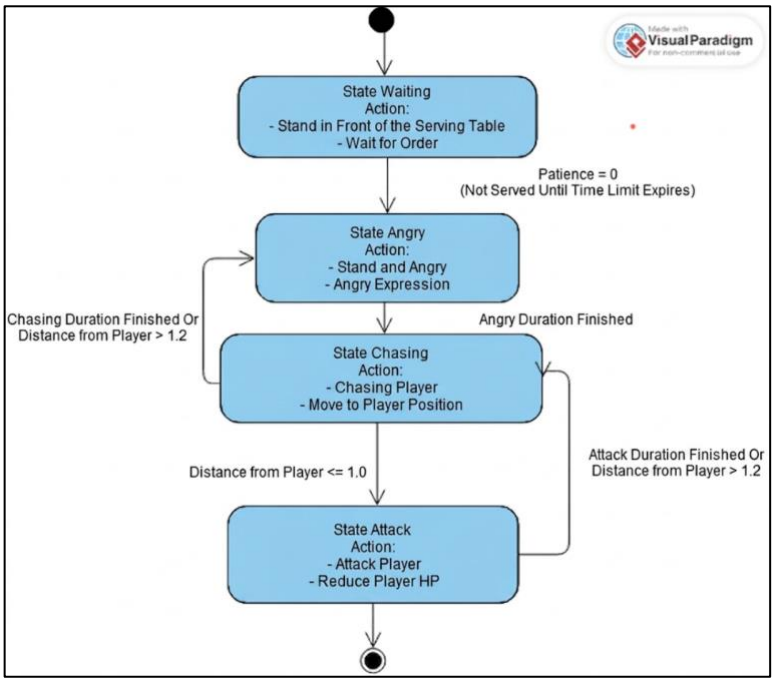

Figure 3. NPC customer FSM diagram.

Table 2. Definition of NPC Customer FSM States

State	Description	Initial Condition	NPC Behavior
Waiting	Customer waiting for service	Customers are in the service process	Patience decreases while waiting
Angry	Customer losing patience	currentTime <= 0	The customer enters an angry state
Chasing	Customer chasing the player	Angry duration is over	Moves according to the player's position
Attack	Customer attacking the player	Distance <= attackDistance	Reduces the player's HP

2.4. Implementation of NPC Customer Behavior

Implementation of NPC behavior involves CustomerSpawner.cs, CustomerMove.cs, and CustomerOrder.cs. CustomerSpawner.cs is responsible for customer emergence and customerPoint management. CustomerMove.cs manages the initial movement to the service point, while CustomerOrder.cs manages orders, patience, and behavior changes based on the FSM. In the Chasing state, the customer moves towards the player's position using Vector2.MoveTowards(). When the customer logs out, FreePoint() is executed before the object is deleted using Destroy(gameObject).

3. RESULTS AND DISCUSSION

3.1. Implementation Results

The results of the research (Figures 4-10) are the implementation of FSM on NPC customers in the Cooking Chaos 2D game. FSM is integrated with spawn, order, patience, food service, player pursuit, HP, and Game Over systems. Failure of a player to provide service before patience runs out triggers a change in NPC behavior and can result in direct consequences for the player.

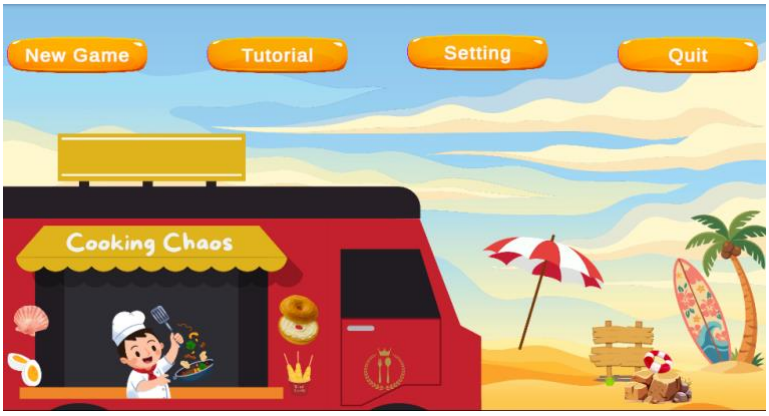

Figure 4. Main menu screen.

Figure 5. New game screen.

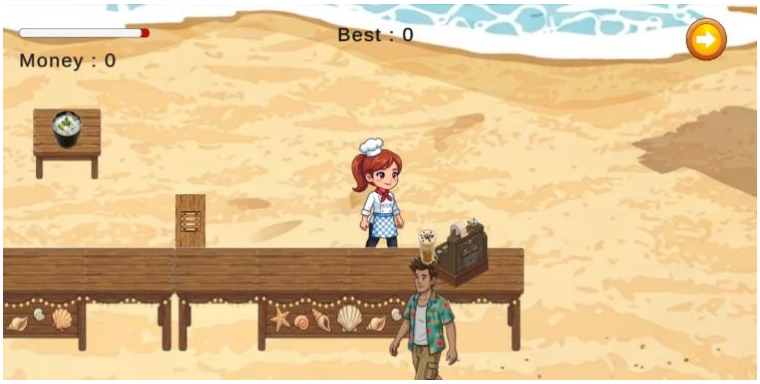

Figure 6. State waiting.

Figure 7. State angry.

Figure 8. State chasing.

Figure 9. Pause page.

Figure 10. Tutorial page.

3.2. Black Box Testing

All 12 scenarios obtained a valid status (Table 3), so the functional testing success rate was 12/12 or 100%. These results indicate that the functionality defined in the test scenario works as designed in the test environment used, not that the game is free from all possible errors.

Table 3. Black box testing results.

ID	Feature	Actual Results	Status
BB01	Customer Spawn	Customer successfully appears	Valid
BB02	Customer Point	Customer heads to an available point	Valid
BB03	Random Order	Random order displayed	Valid
BB04	Patience	Patience decreases	Valid
BB05	Serve Correct	Additional money: Rp10,000	Valid
BB06	Wrong Serve	Customer continues waiting	Valid
BB07	Angry	Customer becomes angry	Valid
BB08	Chasing	Customer chases the player	Valid
BB09	Attack	Player's HP decreases	Valid
BB10	Chase Timeout	Customer leaves	Valid
BB11	Free Points	Point becomes available for reuse	Valid
BB12	Game Over	Game Over panel appears	Valid

3.3. FSM Transition Testing

All 70 trials produced states or actions as designed (Table 4). The overall success rate was 70/70 or 100% across the seven transition scenarios tested. The patience system functions as a trigger for Waiting to become Angry, while time and distance conditions determine behavior in the Chasing phase.

Table 4. FSM transition testing results.

ID	Initial State	Condition	Results	Test	Success	Fault	%
FSM01	Waiting	currentTime > 0	Keep Waiting	10	10	0	100%
FSM02	Waiting	currentTime <= 0	Angry	10	10	0	100%
FSM03	Angry	Angry duration ended	Chasing	10	10	0	100%
FSM04	Chasing	Outside attackDistance	Keep Chasing	10	10	0	100%
FSM05	Chasing	Distance <= attackDistance	Attack	10	10	0	100%
FSM06	Chasing	Chase duration expired	Exit	10	10	0	100%
FSM07	Attack	Attack performed	HP -1 and Exit	10	10	0	100%

3.4. Multi-Customer Testing

All four scenarios obtained valid status (Table 5). Each customer instance can have orders, patience, and FSM states that run independently in a configuration of up to three active customers. CustomerPoint management also runs according to design.

Table 5. Multi-Customer testing results.

ID	Condition	Results	Status
MC01	1 customer	The Order and Patience FSMs operate according to the design.	Valid
MC02	2 customers	The Order and Patience systems function independently.	Valid
MC03	3 customers	Each NPC has its own independent FSM state.	Valid
MC04	Maximum 3	`customerPoint` is not duplicated and is reusable.	Valid

3.5. Discussion

The test results show that the Finite State Machine (FSM) implementation is able to control changes in NPC customer behavior according to game conditions. In Black Box Testing, all 12 scenarios obtained valid status, while FSM Transition Testing resulted in 70 successes from 70 trials. These results show that each function and state transition tested can run according to the specified conditions. This consistency occurs because each NPC behavior is represented in a separate state and has clear transition conditions. Thus, changes in NPC behavior do not occur randomly, but follow transition rules that have been designed. A similar approach was also applied in previous research, which used an FSM to map NPC behavior based on certain conditions in the game [16].

In the FSM mechanism, the patience system is the main trigger for changes in NPC behavior. As long as the patience value is still available, the NPC is in the Waiting state and allows players to complete the order. When patience reaches the time limit, the NPC moves to the Angry state, then Chasing, and can then enter the Attack state if the distance condition is met. The structure makes the relationship between player actions and NPC responses more predictable. Players who complete the service can maintain the game state, while failure to serve results in consequences in the form of pursuit and attacks on the player

The consistency of FSM Transition Testing results also shows that separating states based on certain conditions can reduce the possibility of behavior that is not in accordance with the design. Each transition has a specific trigger, namely the patience time for moving from Waiting to Angry, the duration of the Angry state for moving to Chasing, the player's distance for moving from Chasing to Attack, and the chase time limit for ending the Chasing state. With this structure, NPCs can provide consistent responses to the same conditions in each test. The use of FSM to regulate NPC behavior in game environments has also been applied to Unity-based development to produce NPC behavior that is adaptive to game conditions [17].

From a gameplay perspective, the implementation of FSM not only functions as a technical mechanism for managing NPCs, but also becomes part of the gameplay loop. The service system is a factor that influences changes in NPC behavior, so players must set service priorities and complete orders within the available time. The change in the NPC from waiting to being angry, then chasing and attacking the player, has direct consequences for the player's failure to manage services. Thus, FSM plays a role in shaping the level of game challenge while increasing interaction between players and NPC customers.

Multi-customer testing results indicate that the ordering, patience, and FSM mechanisms function independently when up to three customers are active simultaneously. Each customer maintains their own state and game conditions, ensuring that a state change in one customer does not trigger a state change in another. Limiting the number of active customers to three is a gameplay design choice intended to maintain appropriate difficulty levels and manageability within the service area; it does not represent the absolute limit of the FSM's capacity to handle multiple NPCs concurrently. Therefore, this figure should be understood as a constraint specific to the game design evaluated in this study.

Compared to other NPC control approaches, the FSM offers the advantage of a simple, easily traceable behavioral structure, as every condition is explicitly represented through states and transitions [7]. This approach is well-suited for the customer NPCs in Cooking Chaos, as the required behaviors, Waiting, Angry, Chasing, and Attack, follow relatively clear stages. When developing game AI, the choice of behavior control method must align with the complexity of the intended behavior [8]. FSMs can become less flexible as the number of states and transition rules increases, as the inter-state relationships grow in number and become more difficult to manage [7].

Behavior Trees offer an alternative for NPCs requiring more complex behaviors, as their structure enables hierarchical and organized decision management [18]. Comparisons between Hierarchical Finite State Machines (HFSMs) and Behavior Trees also reveal distinct characteristics regarding NPC behavior management, based on structural design and decision-making requirements [19]. Consequently, the chosen approach must match the complexity of the behavior to be implemented. In this study, the FSM was selected because the customer NPC behaviors, Waiting, Angry, Chasing, and Attack, follow clearly defined transition paths, allowing for the direct representation of state-transition rules.

Beyond Behavior Trees, pathfinding approaches serve a different function in NPC control. While pathfinding focuses on determining an NPC's movement path from a starting point to a destination, FSMs focus on determining conditions and behavioral changes based on transition rules. Pathfinding algorithms, such as A, can be employed to direct NPC movement toward targets within the game environment [20]. Thus, FSMs and pathfinding are not necessarily mutually exclusive approaches; they can be used concurrently to handle NPC behavior and navigation. In the current implementation of Cooking Chaos, the NPC pursuit mechanism relies on movement toward the player's position and does not yet employ pathfinding for environments with complex obstacles. Therefore, this study focuses on evaluating state transitions and the resulting NPC behaviors based on gameplay conditions. Future research could explore the use of pathfinding, or a combination of FSMs with other navigation methods, to create NPC behaviors that are more adaptive to the game environment.

4. CONCLUSION

Research results demonstrate the successful implementation of an FSM to control NPC customer behavior in the Unity-based game Cooking Chaos 2D. The FSM comprises four states: Waiting, Angry, Chasing, and Attack. A patience system acts as the trigger for transitioning behavior from a non-aggressive to an aggressive state when the player fails to provide service within the allotted time.

Black Box Testing across 12 scenarios confirmed the validity of all scenarios. FSM Transition Testing yielded a 100% success rate (70 out of 70 trials) under the established test conditions. Multi-Customer Testing demonstrated that order handling, patience tracking, and FSM logic operate independently when up to three NPC customers are active simultaneously.

This study has limitations regarding NPC movement, which relies on direct movement toward the player's position rather than a pathfinding algorithm. Future development could consider implementing navigation methods, adding state variations, and incorporating more dynamic decision-making mechanisms.

REFERENCES

[1] F. Mandita and B. K. Jati, "Application of Finite State Machine in the 3D Game 'Virus Hunter'," Jurnal Ilmu Komputer dan Desain Komunikasi Visual, vol. 7, no. 2, pp. 90–101, 2022, doi: 10.55732/jikdiskomvis.v7i2.613.

[2] M. H. Syu'aibi, A. Mahmudi, and K. Auliasari, "Perancangan dan Implementasi Metode FSM (Finite State Machine) pada Game Military Defence 2D Berbasis Android," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 7, no. 4, pp. 2349–2357, 2023, doi: 10.36040/jati.v7i4.7508.

[3] M. B. R. Alvian, S. Bukhori, and M. A. Furqon, "Implementation of Finite State Machine to Determine The Behaviour of Non-Playable Character in Leadership Simulation Game," Journal of Games, Game Art, and Gamification, vol. 9, no. 1, pp. 11–21, 2024, doi: 10.21512/jggag.v9i1.10894.

[4] M. B. Firdaus, A. Z. Waksito, A. Tejawati, M. Taruk, M. K. Anam, and A. Irsyad, "Finite state machine for retro arcade fighting game development," International Journal of Informatics and Communication Technology (IJ-ICT), vol. 14, no. 1, pp. 102–110, 2025, doi: 10.11591/ijict.v14i1.pp102-110.

[5] F. P. Utami, H. Z. Alifa, and M. A. Yaqin, "Implementasi Black Box Testing Pada Game Ular Untuk Mendeteksi Bug," Journal Automation Computer Information System, vol. 4, no. 2, pp. 76–87, 2024, doi: 10.47134/jacis.v4i2.85.

[6] J. Gregory, Game Engine Architecture, 3rd ed. Boca Raton: CRC Press, 2019.

[7] J. Nystrom, Game Programming Patterns. USA: Genever Benning, 2014.

[8] I. Millington dan J. Funge, Artificial Intelligence for Games, 3rd ed. Boca Raton: CRC Press, 2019.

[9] D. Brackeen, Developing Games in Unity. New York: Apress, 2019.

[10] Unity Technologies, "Unity Manual," 2024. [Online]. Available: https://docs.unity3d.com/Manual/. [Accessed: Jul. 20, 2026].

[11] A. Dennis, B. H. Wixom, dan D. Tegarden, Systems Analysis and Design: An Object-Oriented Approach with UML, 6th ed. Hoboken: Wiley, 2021.

[12] R. S. Pressman dan B. Maxim, Software Engineering, 9th ed. New York: McGraw-Hill Education, 2020.

[13] B. Beizer, Software Testing Techniques, 2nd ed. New York: Van Nostrand Reinhold, 1990.

[14] R. Black, Managing the Testing Process, 3rd ed. Indianapolis: Wiley Publishing, 2009.

[15] M. Mustaqbal, R. F. Firdaus, dan H. Rahmadi, "Pengujian Aplikasi Menggunakan Metode Black Box Testing," Jurnal Ilmiah Teknologi Informasi Terapan, vol. 1, no. 3, pp. 31–36, 2015

[16] M. B. Nendya, S. G. Gunanto, and R. G. Santosa, "Pemetaan Perilaku Non-Playable Character Pada Permainan Berbasis Role Playing Game Menggunakan Metode Finite State Machine," Journal of Animation and Games Studies. Sumber ini secara langsung membahas pemodelan perilaku NPC menggunakan FSM.

[17] S. Amalia, M. D. F. Abidullah, F. Marcellino, D. D. Rabani, F. F. Azzahra, and A. Abdiansah, "AI-Driven Traffic Simulation using Unity: Implementing Finite State Machines for Adaptive NPC Behaviour," Journal of Intelligent Computing & Health Informatics, vol. 5, no. 2, 2024. Penelitian ini relevan untuk FSM, NPC, Unity, dan integrasi pathfinding.

[18] P. Alam and H. Haryanto, "Implementasi Behavior Tree dalam Menentukan Perilaku Musuh pada Game Metroidvania 'Cursed World'," TECHNO CREATIVE, vol. 2, no. 2, 2024. Penelitian ini dapat digunakan untuk mendukung pembahasan mengenai Behavior Tree sebagai alternatif ketika perilaku NPC semakin kompleks.

[19] J. Lee and J. Kim, "지능형 NPC의 행동 메커니즘에 따른 계층적 유한 상태 기계와 행동 트리의 효율성 평가," 지능정보논문지, vol. 24, no. 2, pp. 113–118, 2024, doi: 10.7236/JIIBC.2024.24.2.113. Penelitian ini secara langsung membandingkan HFSM dan Behavior Tree pada NPC.

[20] Atthariq and D. A. Putra, "Penentuan Pergerakan Non-Player Character Menggunakan Algoritma A* pada Game Action-Role-Playing Game," Jurnal Infomedia: Teknik Informatika, Multimedia, dan Jaringan, doi: 10.30811/.v2i2.516. Penelitian ini relevan untuk pembahasan pathfinding NPC menggunakan A*.