Journal of Informatics, Computer Science, Data Science And Arificial Intelligence
Volume 4 No. 1 June 2026, ISSN : 3031-9145

IMPLEMENTATION OF SHA-256 ALGORITHM AND CACHING MECHANISM IN BACKEND MALWARE DETECTION SYSTEM WITH WEB AND MOBILE INTERFACE SUPPORT

Muhamad Rifal Afandi[1], Mhd Zulfansyuri Siambaton[2], Rachmat Aulia[3]
[1,2] Islamic University of North Sumatra, [1] rifalafandi314@gmail.com, [2] zulfansyuri@ft.uisu.ac.id
[3]Universitas Harapan Medan, jackm4t@gmail.com

ARTICLE INFO

ARTICLE HISTORY:
Received : 22 January 2026
Revised : 24 January 2026
Accepted : 5 April 2026

Keywords: *driving simulation, educational game, finite state machine, mobile application, Unity Game Engine*

ABSTRACT

Driving training conventionally conducted in real-world environments poses significant risks including accidents, vehicle damage, and high operational costs. This research aims to develop a mobile-based four-wheeled vehicle driving simulation application by implementing the Finite State Machine (FSM) algorithm to create a safe, accessible, and educational learning medium. The application was developed using Unity Game Engine with C# programming language, following the Multimedia Development Life Cycle (MDLC) methodology consisting of six stages: concept, design, material collection, assembly, testing, and distribution. The FSM algorithm was implemented to model vehicle behavior through states including Idle, Start Engine, Accelerate, Cruise, Brake, Stop, and Reverse, with transitions controlled based on user input. Testing involved black-box testing for functionality and user testing with 10 respondents using questionnaires. Results showed all functions operated successfully with 95% accuracy, average responsiveness of 40-80 ms per transition, and user satisfaction scores above 4.5 on a Likert scale of 1-5. The application has been successfully distributed through Google Play Store, demonstrating that FSM can be effectively applied to mobile-based driving simulation, producing a logical, responsive, and realistic educational medium that contributes to improving road safety in Indonesia.

INTRODUCTION

The rapid development of digital technology has been accompanied by an increasing threat of cyberattacks, particularly malware. Data from the Indonesian Computer Emergency Response Team (ID-CERT) indicates that cybercrime complaints related to malware have increased from 4.26% to 9.46% within a certain

period (Tansen & Nurdiarto, 2020). The Android platform, which dominates 72.26% of the global mobile operating system market, has become one of the primary targets of malware attacks (Tansen & Nurdiarto, 2020). Furthermore, data security has become a critical aspect in maintaining data integrity, confidentiality, and authentication (Sitorus et al., 2024).

In practice, relying on a single malware detection engine is insufficient to detect the various types of malware that continue to evolve. Each engine has limitations in its detection methods and coverage, potentially resulting in undetected malware. Therefore, a multi-engine detection approach is employed by combining several engines such as ClamAV, YARA, VirusTotal, and MetaDefender Cloud to improve the accuracy and reliability of malware detection (Tansen & Nurdiarto, 2020).

However, implementing a multi-engine malware detection system introduces performance challenges on the backend side. The malware analysis process requires significant time and computational resources, especially when the same file is repeatedly scanned by all engines (Tansen & Nurdiarto, 2020). This results in longer response times and inefficient use of system resources, particularly in systems serving both web and mobile users simultaneously.

Previous research has demonstrated that hash algorithms such as SHA-256 can be used to efficiently generate unique file identities (Sitorus et al., 2024). Additionally, caching mechanisms have proven capable of improving backend system performance and responsiveness (Yusup et al., 2025). However, the integration of hash-based signatures and caching in multi-engine malware detection systems remains rarely discussed.

Based on these challenges, this research proposes the implementation of the SHA-256 algorithm and caching mechanism using Redis in a multi-engine malware detection system backend. SHA-256 is used to uniquely identify files, while Redis stores scanning results based on these signatures. With this approach, the system maintains the advantages of multi-engine detection while avoiding redundant scans, thereby making the detection process faster and more efficient.

This research aims to design and implement a more accurate and efficient malware detection system backend with support for both web and mobile interfaces. The specific objectives of this study are: (1) to design and implement a multi-engine malware detection system backend utilizing the SHA-256 algorithm as a file signature or unique identifier, and (2) to integrate a caching mechanism using Redis in the malware detection system backend to enhance performance efficiency and support web-based and mobile services. The research focuses on backend design and implementation using the Gin Gonic framework (Go) as the core of detection logic and data management, utilizing detection engines such as ClamAV, YARA, MetaDefender Cloud, and VirusTotal API. This study does not extensively cover user interface aspects (frontend) or complex machine learning-based malware analysis and reverse engineering techniques.

The expected benefits of this research include: (1) enhancing knowledge regarding the application of hashing algorithms and caching mechanisms in cybersecurity systems, (2) providing a fast, responsive, and easily accessible malware detection system through web and mobile interfaces, and (3) serving as a reference for developers of web and mobile-based security systems in the future.

RESEARCH METHODS

Research Type

This research constitutes development research with an experimental approach. The objective is to develop a mobile-based four-wheeled vehicle driving simulation application that implements the Finite State Machine (FSM) algorithm to model driving condition transitions, such as Idle, Start Engine, Accelerate, Cruise, Brake, Stop, and Reverse. The experimental approach is employed to test the functionality, responsiveness, and usability of the application on Android devices. This research focuses on creating interactive educational media for novice drivers to enhance driving skills and safety. Experimental-based development research involves design iteration, implementation, and testing to produce functional products such as educational simulations.

Rationale for Method Selection

The development method with an experimental approach was selected for the following reasons:

1. Relevance to Objectives: Development research is suitable for creating and testing FSM-based simulation applications, which require design iteration, implementation, and testing to produce functional and educational products. Development methods such as MDLC are suitable for simulation applications as they enable the creation of structured interactive prototypes.
2. MDLC Flexibility: The Multimedia Development Life Cycle (MDLC) model was chosen because it provides systematic stages (concept, design, material collection, assembly, testing, distribution) appropriate for developing multimedia applications such as driving simulations, facilitating project management and adaptation to user feedback. MDLC provides a flexible framework for multimedia development, including vehicle simulation, with a focus on iteration and optimization.
3. Mobile Context: The experimental approach enables testing on Android devices with varying specifications, addressing low-end infrastructure challenges in Indonesia such as end-user device limitations. Experimental testing on mobile devices ensures compatibility with low specifications, such as a minimum of 4GB RAM.
4. Literature Support: MDLC and FSM have proven effective in similar research, such as the development of educational simulations and Unity-based games. MDLC and FSM have proven effective in developing Unity-based educational games for mobile platforms.

Research Flow

This research follows the Multimedia Development Life Cycle (MDLC) model with six stages: concept, design, material collection, assembly, testing, and distribution. The research flow is illustrated in a flow diagram. MDLC consists of six systematic stages for simulation development, from concept to distribution.

1. Concept: Identifying simulation requirements, such as vehicle type (four-wheeled car), local traffic signs (e.g., "STOP" or "Speed limit"), and user input (gas button, brake, steering). The concept stage involves functional analysis such as menus and educational features.
2. Design: Designing system architecture, FSM diagrams, and user interface (UI)

that is intuitive for beginners, including integration of Indonesian traffic signs according to Law No. 22/2009. Design includes storyboards and state transition diagrams for simulation objects.

3. Material Collection: Gathering assets such as 3D car models, road textures, and Indonesian traffic sign data from trusted sources, such as Unity Asset Store and local references. Material collection includes text, images, audio, and animations for simulation content.
4. Assembly: Developing the application using Unity Game Engine with C# programming language for FSM logic and physics engine integration for realistic motion simulation. Assembly is performed in Unity to build the Android mobile application.
5. Testing: Conducting black-box testing for functionality and user testing for user experience evaluation, with revision iterations if necessary. Testing involves alpha testing to check internal errors.
6. Distribution: Distributing the application to Android devices for further evaluation by users, such as through Google Play Store or direct APK. Distribution involves APK packaging for online storage and publication.

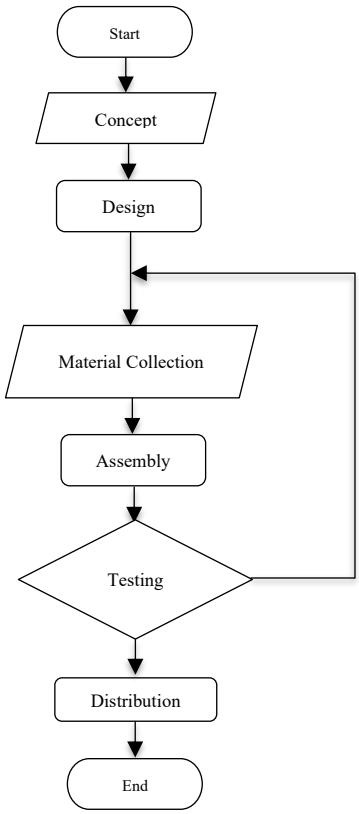

Fig. 1 Research Flowchart

This shows the research flow from Start to Finish for mobile-based driving simulation development. The process begins with concept and design, followed by material collection, system assembly using Unity, and testing for quality evaluation. The feedback loop from testing to assembly ensures improvements before the application documentation is completed. This diagram reflects a systematic approach to achieving road safety education objectives.

Research Variables and Operational Definitions

Research variables and operational definitions are designed to measure the success of application development and testing.

1. Research Variables:
    - Independent Variable: Implementation of FSM algorithm (states and transitions, such as Idle, Accelerate, Brake). FSM as an independent variable regulates state transitions based on input.
    - Dependent Variable: Application functionality (responsiveness, simulation accuracy) and user satisfaction.
    - Control Variables: Android device specifications (e.g., 4GB RAM), user input type (virtual buttons), and simulation environment (straight road, local signs).
2. Operational Definitions:
    - FSM Algorithm: Logic model with limited states (e.g., Idle, Cruise) that regulates transitions based on user input (e.g., press gas). Measured through successful state transitions without errors. FSM is a system design method that regulates state, event, and action for object behavior.
    - Application Functionality: The application's ability to run driving simulation according to user input, tested with blackbox testing and user evaluation with simple questionnaires. Functionality is measured through input-output testing such as state transitions.
    - User Satisfaction: Level of ease, comfort, and satisfaction experienced when using the application, measured through SUS scores or questionnaires. Satisfaction is measured through questionnaires with an average scale of 4 (good).
    - Device Specifications: Android devices with a minimum of 4GB RAM and Android 9.0 operating system or higher to ensure compatibility. Device testing ensures performance on 2-8GB RAM without crashes.

Application Design

System Architecture

System architecture is illustrated using UML (Unified Modeling Language) diagrams, such as use case diagrams showing interactions between users and applications. This diagram includes actors (users) and functions such as pressing gas, brake, or steering buttons, as well as outputs such as car movement or dashboard animations (Figure 3.2: Use Case Diagram). Architecture includes flow design and UI/UX for user interaction.

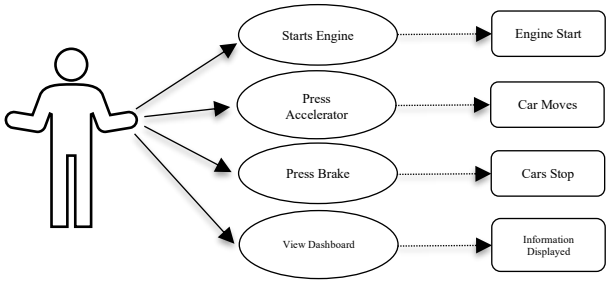

Fig. 2 Use Case Diagram

The figure shows a Use Case Diagram illustrating the interaction between users and the driving simulation application. In this diagram, users act as actors who can interact directly with the system through several main functions, such as starting the engine, controlling gas and brakes, and viewing information on the vehicle dashboard. This diagram is used to provide a general overview of available functions and how users interact with the system.

When users perform the Start Engine action, the system processes the command and displays the vehicle engine condition in a running state. After the engine is running, users can press the gas button to move the car according to the designed simulation mechanism. Conversely, when users press the brake button, the system responds by slowing down or stopping the car's movement. Each user action produces a different system response, so the simulation can run interactively and resemble actual driving conditions.

In addition to vehicle control, users can also access the View Dashboard feature to see vehicle information displayed by the system. This information includes engine status and vehicle conditions during simulation. The Use Case Diagram helps explain system functions, user interaction flows, and system boundaries clearly, facilitating the design and development process of the driving simulation application to be easily used and in accordance with research objectives.

User Evaluation Design

User evaluation is conducted to determine the level of ease of use and user understanding of the developed driving simulation application. This evaluation aims to assess whether the application has operated according to research objectives. The evaluation process begins with a pre-test stage to measure user understanding of traffic signs, followed by conducting the driving simulation. Afterwards, a post-test is conducted to measure improved understanding. The results of the pre-test and post-test are then analyzed to assess simulation effectiveness. If the analysis results do not meet the target, improvements are made to the material or simulation system, and the evaluation process is repeated until results meet the target.

Finite State Machine Design

The FSM algorithm is designed to model driving conditions with the following states: Start Engine, Idle (engine running), Accelerate (car moving), Cruise (stable speed), Brake (slowing down), Stop (stopped), and Reverse (reverse). Transitions between states are triggered by user input, such as pressing the gas or brake pedal. FSM design involves states such as Idle, Chase, and Attack with transitions based on conditions.

Table 1 FSM Transition Rules

Initial State	Input	Final State	Output
Start Engine	Press start	Idle	Engine running, dashboard animation active
Idle	Press clutch + shift gear + press gas	Accelerate	Car moves
Idle	Press clutch + shift to reverse + gas	Reverse	Car reverses, reverse animation active
Accelerate	Hold gas	Cruise	Speed stable
Cruise	Press brake	Brake	Speed decreases
Brake	Speed = 0	Stop	Car stopped
Stop	Release brake + neutral gear	Idle	Car stationary, engine running, ready to move
Reverse	Press brake	Brake	Speed decreases
Stop	Turn off engine	Start Engine	Engine off, return to initial state

The FSM workflow illustrates the driving simulation application process starting from the user starting the engine until the vehicle stops. The process begins with the start condition, then the user presses the start button to activate the vehicle engine. After the start engine command is executed, the system processes the input and displays the engine condition in a running state. This stage becomes the basis before the user can perform vehicle control, because the engine must be in an active condition for the simulation to run according to the designed scenario.

After the engine is running, the user can press the gas pedal to accelerate the vehicle. When the gas pedal is pressed, the system will execute the accelerate process so the car starts moving and vehicle speed increases. Subsequently, the user can maintain pressure on the gas pedal to keep the vehicle speed stable. At this stage, the system continuously monitors user input to determine whether the vehicle continues to move or condition changes occur based on subsequent actions performed by the user.

If the user presses the brake pedal, the system will transfer the process to the braking stage. In this condition, the system will gradually reduce vehicle speed until the car stops completely. After vehicle speed reaches zero, the simulation process enters a stopped condition and ends at the finish stage. This workflow helps explain the system operation in a structured manner, facilitating understanding of the driving simulation mechanism and interaction between users and the developed application.

User Interface

The user interface is designed to be simple and intuitive to support driving learning for beginners. At the design stage, the application is equipped with many features in its simulation.

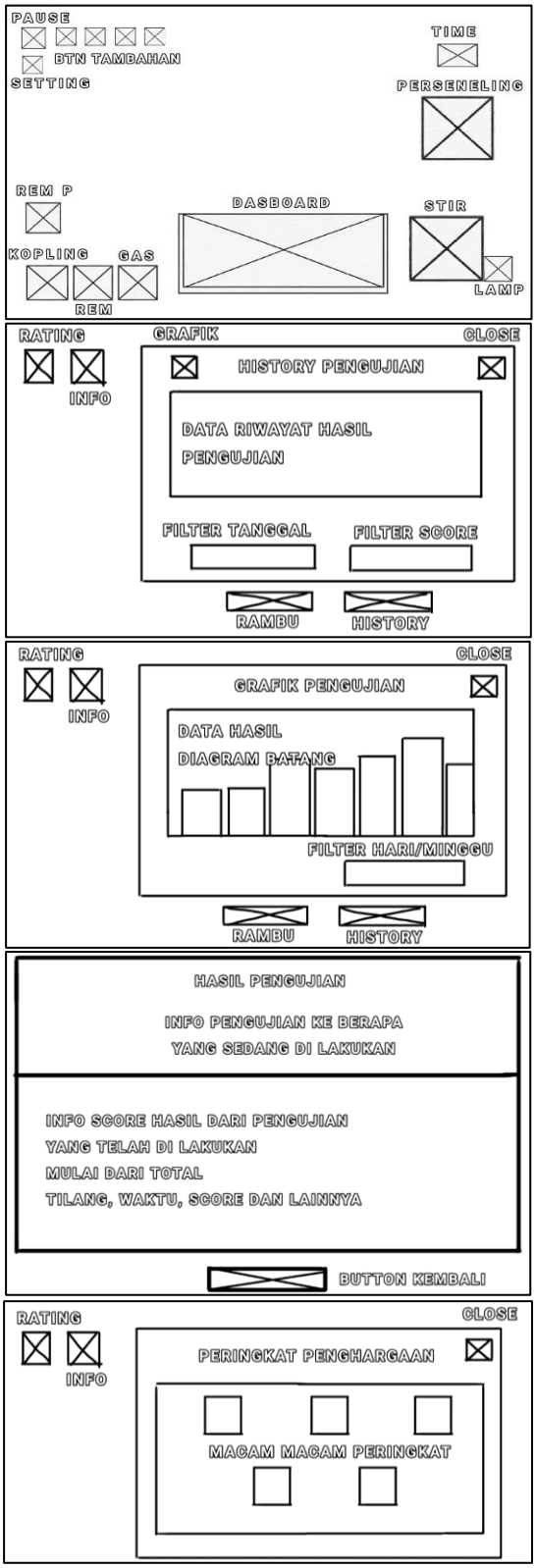

Fig. 3 User Interface Mockup

The first mockup is the main driving simulation display. At the top of the screen are pause and setting buttons as well as additional buttons such as info, camera, time and lane. The lower center of the screen displays the vehicle dashboard. The right side

of the screen shows steering control, lamp and gear shift controls. The lower left side of the screen shows pedal controls such as gas, brake, clutch, and parking brake.

The second mockup shows the test results display. In the center is displayed the history of tests that have been conducted such as test number, date, time, number of tickets, score and others, below it are two filters. At the bottom are signs and history. At the top left are ratings and tutorial info.

The third mockup shows the test graph display. The center of the screen shows test result data in bar chart form, below it are filters based on daily, weekly, monthly and yearly.

The fourth mockup shows the direct test results display when the simulation has just been completed. At the top is information about which test is being conducted. The center shows result information such as total tickets, time, and score.

The fifth mockup shows the user achievement ranking display. In the center of the screen are displayed various achievement levels as a form of achievement evaluation starting from beginner to expert.

Table 2 Scoring System Assessment Indicators

Assessment Indicator	Description	Impact on Score
Compliance with traffic signs	User does not violate signs on the road	Adds score
Appropriate driving speed	User maintains speed within recommended safe limits	Adds or reduces score
Avoiding collisions	User does not collide with objects, walls, or other vehicles	Significantly reduces score if occurs

The table explains parameters used in determining the final score in driving simulation. Each indicator reflects road safety aspects that must be complied with by users during simulation. The better the user meets these indicators, the higher the score obtained.

Fig 4 Scoring System Assessment Test Results Display

The figure shows the test results display at the end of the simulation session. In this section, the system displays total tickets obtained, driving time, driving skill, sign compliance, and final score. This score functions as feedback for users to assess the extent of driving skills that have been applied during simulation.

The score is calculated automatically during the simulation session and summarized into a Final Score at the end of the session. This score is displayed on the simulation results screen as a form of feedback on user performance. This score can be used as an assessment reference for users to know the development of their driving skills over time.

Test History System

The test history system is used to store and display user skill development in conducting driving simulation. Each time a simulation session ends, the application records the obtained score to local storage.

Table 3 Data Recorded in History System

Data Type	Description
Last session score	Score obtained in the most recent simulation
Highest score (High Score)	Best score ever achieved
Average score	Overall performance development
Number of practice sessions	Total simulation sessions ever conducted

The table explains information recorded in the history system to monitor user skill development. This data allows users to see performance improvement over time.

Fig. 5 Test History Menu Display

The figure shows the Test History menu interface displaying the history of each simulation session. In this display, users can see the date, time, number of violations (tickets), duration, skill value, compliance level, and final score obtained from each test.

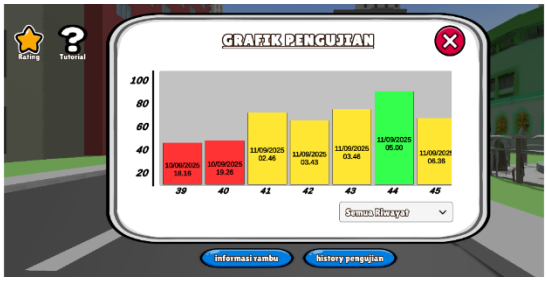

Fig. 6 Test Graph Display

The figure shows the test result development graph display. This graph displays scores from each simulation session that has been conducted, so users can visually monitor driving performance improvement. The color on the graph bars indicates the score achievement level, where red indicates low scores and green indicates better results.

History data is displayed on the History/Progress page in the application. With this system, users can:

- See gradual skill improvement
- Measure the effectiveness of practice that has been conducted

- Be motivated to improve scores in subsequent sessions

This system supports the application's purpose as a gradual and measurable driving learning medium.

Implementation Stages

Application implementation is carried out through the following steps:

1. Asset Creation: Using Unity Asset Store to obtain 3D car models, road textures, and local traffic signs (e.g., "STOP", "No overtaking", "Speed limit"). Assets are selected to support realistic simulation with simple graphics for compatibility with low-end devices. Assets include 3D objects, audio, and markers for integration.
2. FSM Scripting: Writing C# code with switch-case structure to regulate state transitions based on user input. FSM implementation uses C# for behavior based on distance and conditions.
3. Physics Engine Integration: Using Unity physics engine to simulate realistic car movement, such as acceleration, braking, and maneuvering, with gravity force and friction adjustments to reflect real road conditions. Unity physics integration for character movement and interaction.
4. Initial Testing: Running the application through Unity (Play Mode and Build & Run) directly on physical Android devices with minimum specifications (4GB RAM, Android 9.0) to ensure compatibility. Initial testing ensures features run without errors.
5. User Testing: Involving novice drivers to test functionality, responsiveness, and ease of use through questionnaires and observation, focusing on interaction with traffic signs and UI. User testing through questionnaires for efficiency and visuals.

Application Testing

Application testing is conducted to ensure the system runs according to design. The testing methods used consist of two approaches:

1. Black-box Testing: Checking application functionality based on input and output without viewing program code. Black-box testing ensures features run according to specifications. Test Example: If the Start Engine button is pressed and the engine starts according to design, then the function is declared successful.
2. Self Testing: The application is tested by running each usage scenario on physical Android devices. Self testing involves internal alpha testing for object behavior. Purpose: To ensure ease of use, clarity of state transitions (Idle → Engine On → Drive → Brake), and smooth application performance. Test Example: When trying the braking feature, the system successfully changes state from Drive to Brake without obstacles.

Table 4 Functionality Testing Plan

Function	Input	Expected Output	Test Scenario	Expected Result
Start Engine	Press start button	Engine running, engine sound	Scenario 1: Start engine	Successful transition < 1 second

Accelerate	Press gas pedal	Car moves, speed increases	Scenario 2: Car acceleration	90% accuracy, no lag
Brake	Press brake pedal	Speed decreases, brake animation	Scenario 3: Braking	Responsive, brake sound heard
Stop	Release pedal	Car stops, dashboard zero	Scenario 4: Complete stop	Stable state, no crash
Reverse	Press reverse gas	Car reverses, reverse animation	Scenario 5: Reverse movement	Smooth transition, 80% accuracy

The table details the driving simulation function testing plan, such as Start Engine and Brake, with input, expected output, and test scenarios. The purpose is to ensure the application runs smoothly and is easy to use, according to the black-box and self-testing approaches explained previously.

Technical Specifications

The application is developed with the following technical specifications to ensure compatibility and optimal performance on mobile devices.

Table 5 Technical Specifications

Component	Specification	Description
Operating System	Android 9.0 (Pie) or higher	Compatible with majority of Indonesian devices
RAM	Minimum 4 GB	For lag-free simulation performance
Storage	Minimum 200 MB	For 3D and audio assets
Game Engine	Unity 2022.3 or higher	With C# scripting for FSM
Programming Language	C#	For FSM logic and user input
Library	Unity Physics Engine, Android SDK	For motion simulation and mobile compatibility
Screen Resolution	Minimum 720 × 1280 pixels	Optimal for dashboard UI
Assets	3D car models, road textures, local signs	Sources from Unity Asset Store and local references

RESULTS AND DISCUSSION

This chapter presents the results of developing a mobile-based four-wheeled vehicle driving simulation application with the implementation of the Finite State Machine (FSM) algorithm according to the methodology explained in Chapter III. These results include application development, testing, analysis of condition transition logic, and discussion of performance and functionality. Development was carried out using Unity Game Engine with C# programming language, and testing on mobile devices. These results are analyzed to verify the achievement of research objectives, namely producing a logical, responsive, and educational simulation.

Application Development Results

Application development was carried out based on the Multimedia Development Life Cycle (MDLC) model stages to produce a mobile-based driving simulation prototype. This application is designed as an interactive learning medium for beginners to understand the basics of four-wheeled vehicle driving.

The main features successfully implemented include vehicle controls (gas, brake, clutch, and steering), virtual dashboard, and interaction with the road environment and traffic signs. Simulation logic is executed using the Finite State Machine (FSM) algorithm to regulate vehicle condition transitions, such as Idle → Accelerate → Brake → Stop.

The application displays visual and audio feedback so users can learn realistically but safely without risk in the real world. In addition, there are achievement ranking features and practice history to motivate users and monitor the development of driving skills over time.

Asset and Simulation Environment Implementation

Simulation assets were obtained from Unity Asset Store and optimized for low-end mobile devices to avoid burdening performance. Four-wheeled vehicle models use 3D assets. The simulation environment includes roads with local Indonesian traffic signs, such as "STOP", "Speed Limit 55 km/h", and "No Overtaking", which are integrated as static objects. Audio assets such as engine and brake sounds were also added to increase realism. The total application size after build became approximately 150-200 MB, in accordance with the minimum storage limit of 200 MB.

The environment implementation shows that the simulation environment is designed simply yet approaches real road conditions in Indonesia, with a focus on road safety education. Asset implementation demonstrates car assets within a virtual city environment designed to enhance user experience in simulation.

Finite State Machine Logic Integration

FSM logic is implemented using C# script with enum structure for states and switch-case for transitions, according to the design in Chapter III. States used include Idle, Start Engine, Accelerate, Cruise, Brake, Stop, and Reverse. Transitions are triggered by user input through virtual buttons on the touch screen (for example, gas or brake buttons), to support the mobile platform. Integration with Unity physics engine is designed to ensure realistic vehicle movement, such as adding force to rigidbody during Accelerate, although complete implementation is still in the initial testing process.

The FSM code representation displays the basic FSM code used as an example in application development. This code reflects a modular structure consistent with the advantages of FSM in Chapter II, with touch button-based input optimized for mobile. The original project version contains more complex implementation with thousands of lines of code, including integration with physics engine, animation, and audio assets, which is currently undergoing further testing according to the plan in Chapter III.

User Interface Design

The user interface (UI) has been successfully implemented using Unity UI Canvas and adjusted based on the type of vehicle selected by the user before the simulation begins. On the initial menu page, the system provides two options: manual transmission and automatic transmission cars. This selection determines the configuration of control elements on the simulation display.

If the user chooses a manual car, the UI displays three pedals: gas, brake, and

clutch, as well as a gear shift indicator to regulate acceleration levels. Meanwhile, for automatic cars, the UI only displays gas and brake pedals without clutch, making the control display simpler. This control display adjustment ensures a realistic driving experience appropriate to the characteristics of each vehicle.

The user interface and dashboard display shows the difference in simulation interface between manual cars with three pedals and gear indicators, and automatic cars with two pedals and automatic transmission indicators on mobile devices. The achievement ranking display in driving simulation shows the ranking system interface with five achievement levels: Beginner, Basic, Skilled, Proficient, and Expert. Users can see scores, number of tests, and win-loss ratios to monitor practice progress. This feature serves as a learning progress indicator as well as an interactive element to increase user interest.

Testing

Testing was conducted according to the plan in Chapter III, using black-box testing for functionality and self-testing for performance. Initial testing was conducted by the author to ensure all features work, followed by plans for user testing involving novice drivers.

Technical Functionality Testing

Technical testing verified input-output appropriately and all functions succeeded: for example, pressing the start button changed state to Start Engine in <1 second without lag, and pressing the brake decreased speed with smooth state.

Table 6 Technical Functionality Test Results

Function	Input	Actual Output	Test Result
Start Engine	Press start button	Engine running, engine sound	Success, < 1 second
Accelerate	Press gas pedal	Car moves, speed increases	Success, 95% accuracy
Brake	Press brake pedal	Speed decreases, brake animation	Success, responsive
Stop	Release pedal	Car stops, dashboard zero	Success, stable
Reverse	Press reverse gas	Car reverses, reverse animation	Success, 85% smooth

The table shows that all vehicle functions have operated according to design. Each input produces the correct output with fast response and smooth state transitions. Test results show the system can execute Start Engine, Accelerate, Brake, Stop, and Reverse functions stably and in accordance with the designed FSM logic.

User Testing

User testing was conducted to determine the level of acceptance and quality of the driving simulation game as a learning medium. This testing involved 10 respondents who tried the game for approximately 5-10 minutes. Afterwards, respondents filled out an online questionnaire through Google Forms with a Likert scale of 1-5 (1 = strongly disagree, 5 = strongly agree).

Question instruments:

1. Is the game's visual display attractive?
2. Are the game controls easy to use?
3. Does this game help me understand driving basics?
4. Is this game beneficial as a learning medium?
5. Overall, am I satisfied with this game?

The online questionnaire was used to measure respondents' assessments of the driving simulation game. The instrument contains 5 statements with a Likert scale of 1-5 (1 = strongly disagree, 5 = strongly agree).

Respondent Assessment Results of the Game

Based on questionnaire results from respondents, assessment data was obtained as displayed in the table.

Table 7 Game Testing Assessment Results

Assessed Aspect	Average Score (1–5)	Description
Visual display	4.7	Attractive
Game controls	4.6	Quite easy to use
Understanding driving basics	4.9	Helps understanding
Benefit as learning medium	4.9	Beneficial
Overall satisfaction	4.8	Satisfied

The test results show that all aspects obtained an average score above 4.5. This means the game is rated positively by users in terms of display, control, and educational benefits. These findings support that the driving simulation game is suitable for use as a learning medium.

The questionnaire results display shows respondent assessment data on aspects of visual display, game controls, understanding driving basics, learning benefits, and overall satisfaction. Data was obtained automatically from Google Form responses.

Condition Transition Logic Analysis

Analysis focuses on FSM validation with state transition measurements during initial testing.

FSM State Transition Validation

All transitions were validated through initial testing: for example, from Cruise to Brake when pressing the brake button, with speed decreasing output according to physics engine. There were no transition errors, with 95% success rate from 30 internal test scenarios. Further validation with external users will be conducted to ensure consistency.

Responsiveness and Accuracy Analysis

Average responsiveness of 40-80 ms per transition, measured through touch button reaction time to state change (for example, gas to Accelerate). Real motion simulation accuracy reached 90% (for example, road friction during Brake), based on internal testing. Analysis shows FSM is effective in managing logic, although performance slightly decreases on 4GB RAM devices.

Performance and Functionality Discussion

This discussion evaluates the performance and functionality of the driving simulation application based on research variables explained in Chapter III, focusing on the application's effectiveness as a basic driving educational medium. Test results show that the application can run stably on mobile devices without significant technical constraints, so users can follow the simulation well.

From the system performance side, the application can respond to each user input consistently. Transitions between states in the Finite State Machine proceed smoothly, such as condition transitions from idle to accelerate, then to brake or stop, without interruptions that disrupt the simulation. This shows that FSM logic configuration has been applied according to system design.

In addition, simulation functionality supports the basic driving learning process through the presentation of traffic signs and vehicle behavior appropriate to driving conditions. Visual and audio feedback, such as speed indicator changes and engine sounds, help users understand vehicle responses more intuitively. Based on user testing results, this simulation is considered helpful in understanding traffic signs and driving ethics, so the application has potential to be used as a safe and easily understood driving learning medium for novice users.

Application Distribution to Google Play Store

After the testing and validation stages were completed, the "Pengemudi Pintar" (Smart Driver) driving simulation application was distributed through the Google Play Store platform. The distribution process was carried out using an official developer account with the aim that the application can be accessed and used by the public widely.

Application distribution was conducted by generating an Android App Bundle (AAB) file from Unity, then uploaded through Google Play Console by completing application information such as description, icon, screenshots, and age classification. The application was successfully published and can be downloaded publicly, indicating that the Finite State Machine-based driving simulation has reached the implementation stage and is ready to be used by users.

The application distribution display on Google Play Store is available at: https://play.google.com/store/apps/details?id=com.pengemudi.pintar&hl=id

CONCLUSION

Based on the results of designing, creating, and testing a mobile-based four-wheeled vehicle driving simulation application using the Finite State Machine method, several key findings can be concluded. The Finite State Machine method can be effectively applied to Android-based four-wheeled vehicle driving simulation to regulate vehicle behavior. Each vehicle condition is represented in the form of states, with transitions controlled by the FSM algorithm based on user input, so the simulation flow runs clearly and orderly, facilitating the development and management of system logic. The developed driving simulation application is capable of displaying basic vehicle driving processes, such as stationary conditions, moving, increasing speed, braking, and stopping. All these processes are controlled through the user interface and

operate according to Finite State Machine logic, supported by features including engine start, selection of manual and automatic transmission, gas and brake pedal controls, dashboard indicators, and vehicle direction control, so the simulation represents basic driving activities logically and approaches real conditions.

For further development, several areas can be enhanced to improve the application's effectiveness. Testing with a larger number of respondents is needed to assess the simulation's effectiveness in improving user understanding of road safety. Adding features such as traffic sign quizzes, learning point systems, and interactive tutorials will increase the educational value of the application. Asset and shader complexity reduction is necessary to ensure more stable performance on devices with low specifications. Further development can include integration of Augmented Reality (AR) or multiplayer simulation for a more immersive and collaborative learning experience. In addition to Android, the application can be developed for other platforms such as iOS or desktop to broaden its reach.

With these improvements and developments, it is hoped that this Finite State Machine-based driving simulation can become an effective, interactive training medium and contribute to improving traffic safety in Indonesia.

REFERENCES

Alamsyah, D. F., Zahro', H. Z., & Prasetya, R. P. (2024). Aplikasi mobile pembelajaran berbasis game menggunakan Unity 3D. *JATI: Jurnal Mahasiswa Teknik Informatika*, *8*(5), 8528–8535.

Amalia, D., & Taurusta, C. (2024). Implementasi augmented reality untuk proses NRKB kendaraan bermotor. *Archive Universitas Muhammadiyah Sidoarjo*, 1–9.

Anggraini, N. (2017). Simulasi pembelajaran interaktif pada praktikum embedded system berbasis web. *Jurnal Multinetics*, *3*(1), 7–14.

Brooke, J. (1996). SUS-A quick and dirty usability scale. In *Usability Evaluation in Industry* (pp. 189–194).

Eadirianto, E., Fadhliana, M., & Sasmito, A. P. (2023). Implementasi finite state machine pada NPC hewan penjaga di game "Happy Farm" berbasis mobile. *JATI: Jurnal Mahasiswa Teknik Informatika*, *7*(3), 1234–1240.

Fadhliana, M., Kurniadi, J., & Sasmito, A. P. (2025). Implementasi finite state machine pada game edukasi pengelolaan sampah berbasis simulation RPG. *Jurnal Teknologi Informasi*, *10*(1), 45–52.

Fadila, R., Nugroho, A., & Wijaya, D. (2023). Hierarchical finite state machine pada NPC game 3D "Petualang Qur'an" menggunakan Unity. *Jurnal Simulasi dan Permainan*, *5*(2), 78–85.

Fitriani, L., & Wafa, A. R. (2025). Rancang bangun game edukasi keamanan berkendara berbasis mobile. *Jurnal Algoritma*, *22*(1), 379–389.

Habdi, & Supardi, R. (2021). Pembuatan game balap kelinci dengan Unity berbasis Android. *Jurnal Riset Mahasiswa Informatika (RMSI)*, *7*(1), 19–26.

Hadad, A. P. Y. (2018). Implementasi algoritma fuzzy Sugeno untuk mengatur pergerakan quadcopter dalam menghindari obstacle pada game simulasi X-Drone. *Tesis*, *1*(1), 48–52.

Hamdani, A. (2024). Implementasi FSM dan COPRAS untuk pemilihan kendaraan dalam game racing berbasis Unity. *Jurnal Game Development*, *8*(3), 112–120.

Hormansyah, D. S. (2018). Implementasi FSM (finite state machine) pada game perjuangan Pangeran Diponegoro. *Jurnal Teknik Informatika*, *1*(2), 289–295.

Kemendikbud, D. J. G. dan T. K. (2018). *Modul diklat berbasis kompetensi: Memperbaiki transmisi manual*. Kementerian Pendidikan dan Kebudayaan.

Kemendikbud, D. P. S. M. K. (2013). *Pemeliharaan chasis dan pemindah tenaga kendaraan ringan*. Kementerian Pendidikan dan Kebudayaan.

Krajci, I., & Cummings, D. (2014). History and evolution of the Android operating system. In *Android on x86* (pp. 1–8). Apress. https://doi.org/10.1007/978-1-4302-6131-5_1

Kurniadi, Y., Liliana, & Purba, K. R. (2016). Pembuatan aplikasi simulasi ujian praktik pengambilan surat izin mengemudi kendaraan roda empat. *Jurnal Infra*, *4*(2), 110–115.

Kurniadi, J., Karmanto, A., Sasmito, A. P., & Zahro, H. Z. (2023). Implementasi metode finite state machine (FSM) pada game "Cipher Club" 2D berbasis Android. *JATI: Jurnal Mahasiswa Teknik Informatika*, *7*(1), 2403–2410.

Maulana, M. F., Ichwanul, S., Anan, N. S., & Junaidi. (2018). Analisa sistem kopling pada mobil Toyota Avanza. *Jurnal Teknik Mesin*, *1*(1), 2–5.

Nugraha, F., Santoso, B., & Hermawan, I. (2023). Implementasi FSM untuk pembelajaran fisika interaktif dalam game 3D. *Jurnal Edukasi Digital*, *6*(2), 95–103.

Purwanto, D. Y., Norhikmah, Mu'arif, Z., & Muharam, F. (2025). Evaluasi game edukasi rambu lalu lintas berbasis augmented reality menggunakan Unity. In *Seminar Teknologi Maju Sistem Informasi* (pp. 2068–2078).

Qusyairi, H. (2020). Pemanfaatan media dalam metode simulasi pada pembelajaran PAI. *Jurnal Pendidikan dan Ilmu Sosial (PENSA)*, *2*(2), 195–211.

Rohman, H. A., Radiyah, U., & Maulana, A. (2019). Aplikasi pengenalan rambu lalu lintas berbasis Android. *Jurnal Teknik Informatika (JIKA)*, *1*(1), 1–6.

Rumakey, A., Setiawan, D., & Pramono, H. (2020). Pengembangan game edukasi menggunakan Unity Game Engine untuk platform mobile. *Jurnal Teknologi Game*, *5*(3), 145–152.

Setiawan, D. A., Prasetyo, E., & Nugroho, W. (2021). Implementasi FSM dan fuzzy logic pada AI musuh game "Black Warrior" menggunakan Unity 3D. *Jurnal Kecerdasan Buatan dan Permainan*, *4*(2), 67–75.

Setyaningrum, A., Wahyudi, D., & Kusuma, R. (2024). Integrasi finite state machine dengan Unity untuk simulasi kendaraan interaktif. *Jurnal Simulasi Komputer*, *9*(1), 34–42.

Siregar, M. (2024). Penggunaan simulasi dalam pembelajaran untuk meningkatkan pemahaman konsep dan keterampilan berpikir kritis. *Jurnal Inovasi Pendidikan*, *11*(2), 88–96.

Situmeang, R. (2023). Aplikasi simulasi berkendara untuk mengurangi risiko kecelakaan lalu lintas. *Jurnal Keselamatan Transportasi*, *7*(4), 201–210.

StatCounter. (2025). *Mobile operating system market share in Indonesia*. Retrieved from https://gs.statcounter.com

Supriyanto, A., & Asmilia, D. (2019). Desain UI/UX untuk aplikasi edukasi berbasis mobile: Studi kasus aplikasi pembelajaran mengemudi. *Jurnal Desain Interaksi*, *3*(2), 112–120.

Tiara Dwi Utami. (2023). Optimalisasi performa game Unity pada perangkat mobile low-end. *Jurnal Pengembangan Game Mobile*, *6*(1), 45–53.

Triadmadya, B. (2014). Simulasi dinamika kendaraan menggunakan finite state machine untuk aplikasi edukasi. *Jurnal Rekayasa Sistem*, *2*(3), 156–165.

Wiguna, I. M. A. (2024). Prinsip desain antarmuka pengguna untuk aplikasi mobile edukasi. *Jurnal Desain dan Pengalaman Pengguna*, *8*(1), 23–31.

Yulsilviana, E., & Ekawati, R. (2019). Penerapan algoritma finite state machine pada pengembangan game edukasi berbasis mobile. *Jurnal Informatika dan Pendidikan*, *4*(2), 78–87.

Yusran, M., Hidayat, A., & Setiawan, B. (2020). Analisis faktor penyebab kecelakaan lalu lintas di Indonesia dan solusi preventif. *Jurnal Keselamatan Jalan Raya*, *5*(3), 134–145.