Finite State Automata and Machines Mobile (FSAM): A Visualization Centric Finite State Machine Simulator accepting Diagrammatic Input for Mobile Devices

Von Vincent O. Vista and Jaime M. Samaniego

Abstract— **Formal languages and automata theory are fundamental pillars of computer science. Educators recognised early on that automata theory is difficult to teach due to its theoretical nature, prompting the development and application of pedagogical tools for teaching and learning. There are numerous PC and web-based simulators, but the mobile platform is lacking, and all of the available simulators have either: only basic features; being Descriptive Language Based or Visualization Centric and accepting Structured Input in paradigm; or having a poor UI and UX when evaluated in accordance with UI guidelines and UX principles. This resulted in the development of Finite State Automata and Machines mobile (FSAM), which offers diagramming and simulation capabilities for finite state automata, pushdown automata, and turing machines. Moreover, it offers step and bulk simulations, file I/O with cross-compatibility with JFLAP, NFA conversion, and DFA minimization. There are no public C# libraries for automata building and simulation at the moment, and FSAM was built from the ground up. User evaluation was conducted using the Usefulness, Satisfaction, and Ease of use (USE) questionnaire with respondents who have used automata simulators before. Using a 7-point likert scale and calculating the mean scores, the results show that Usefulness received 6.40, Ease of Use received 6.36, Ease of Learning received 6.48, and Satisfaction received 6.53. All of the scales received a mean score of 6 (agree) to 7 (strongly agree), indicating that respondents overall agree-strongly agree with FSAM's usefulness, ease of use, ease of learning, and satisfaction. Furthermore, the system is built and designed to be modular, allowing FSAM to be expanded and extended to other finite state machine applications and functions.**

Index Terms— **automata, finite state automata, pushdown automata, turing machines, automata theory, simulator, mobile development, finite state machines**

I. Introduction

The study of formal languages and automata theory is important since it is one of the core foundations of the field of computer science. This is enforced by the fact that there is at least one course dedicated to automata theory in each and every computer science curriculum [1]. Educators realized early on that automata theory is hard to teach. They thought that educational tools would aid in the teaching and learning of automata theory, and as technology advanced, automata simulators were developed to aid in the teaching of the course [1]. Currently, there are plenty of simulators developed to work on personal computers and the web, but when it comes to mobile, it is lacking with only a few simulators available. The simulators available currently in the major mobile application markets such as Google Play often only have very basic features, descriptive language-based or visualization centric accepting structured input in paradigm, or with poor UI and UX. There are currently 6.648 billion smartphone users, or 83.32% of the world's population in 2022 [2]. Today's students use their mobile devices for a variety of purposes, including academic work, and some use their phones exclusively without a personal computer. Therefore, having a mobile application for creating and simulating finite state automata will allow students to study and explore formal languages and automata theory whenever and wherever they choose.

Presented to the Faculty of the Institute of Computer Science, University of the Philippines Los Baños in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science

A. Background of the Study

Automata Theory is the study of abstract machines called automata (singular: automaton) and computation issues they can solve [3]. Automata are abstract models of machines that can perform computations by going through a number of states or configurations. A transition function figures out the next state based on a small part of the current state at each state of computation. Consequently, the algorithm accepts the input once it reaches an acceptable configuration. Automata theory is applied and used in many fields including but not limited to programming language and compiler design, database and software engineering, networks, security, computational biology, natural language processing, and artificial intelligence [4]. There are four major families of automaton. In order of least powerful and simplest to most powerful and complex, we have finite-state machine, pushdown automata, linear-bounded automata, and Turing machines. The power of an automata class is classified by the class of formal languages they can recognize, often showed with the Chomsky hierarchy [5].

Educators understand that theoretical computer science courses are notoriously difficult to teach for a variety of reasons and have a reputation for being challenging [1]. This is why they devised a method to simplify the process with the aid of pedagogical tools for teaching, which take the form of simulators [1]. The development of automata simulators dates back to the early 1960s and has evolved to use various approaches over the years, with the two major classifications being language-based automata simulators and visualization centric automata simulators [1]. Visualization centric automata simulator further divides into two subclasses, namely visualization centric automata simulators accepting structured input and visualization centric automata simulators accepting diagrammatic input. Additionally, classifying automata simulators only became necessary as many more simulators have been developed. Majority of these simulators has been developed for personal computers, such as Automata Editor, JFAST, PetC, and the most popular being JFLAP. Some have also been developed for the web such as Automaton Simulator and FSM Simulator. These simulators have varying levels of features, quality and design. A platform that is not often considered for development of automata simulators is the mobile platform.

The ubiquity of mobile phones is especially attractive to develop apps for. According to Bank my Cell's October 2022 Mobile User Statistics, there are currently 6.648 Billion smartphone users, or 83.32% of the world's population in 2022 [2]. Mobile phones are also always carried around or near their users, as they are often treated as extensions of ourselves in the modern era. This implies that developing for mobile will give the highest reach and accessibility for applications. Students also use their mobile devices not just for social media and communication, but also for doing academics, with some using their phones exclusively for those who do not have access to a personal computer or a laptop. Some educators also use their mobile devices for teaching especially during online classes.

B. Statement of the Problem

Numerous automata simulators have been developed for the PC platform, and some have been developed for the web platform. However, there is a deficiency in the mobile market for automata simulators. After analyzing and filtering irrelevant search results in Google Play and Apple Store, two of the largest markets for mobile applications, the number of automata simulators is less than ten. All of which are either: (1) few in functionality and features for building and simulating automata [5]; (2) classified under descriptive language-based [6] or visualization centric accepting structured input [7] [8], so it is impossible to alter the generated diagram; or (3) poor in user-interface design and user experience when evaluated based on the laws of UX and principles of UI design [9] [10].

C. Objectives

This research aims to develop a visualization centric finite state machine simulator for mobile that individuals learning automata theory can utilize anytime and anywhere. It specifically aims to:

1) Develop a mobile application for diagramming and simulating finite state machines.
2) Create an import and export system for the created finite state machines.
3) Implement other pertinent FSM functions, such as NFA conversion and DFA minimization.
4) Assess the usefulness, ease of use, ease of learning, and satisfaction of its users.

D. Significance of the Study

The developed application will address the glaring issues and problems of the automata simulators currently available on the market, namely their lack of features, descriptive language-based or visualization centric accepting structured input in paradigm, and poor UI design and UX. Students will be able to study and investigate automata theory at any time and place, as the mobile platform provides the greatest accessibility and reach. Those who do not have access to a personal computer will be able to explore finite automata through their mobile devices. In addition, there are currently no visualization centric automata simulators accepting diagrammatic input available for iOS. Since FSAM will be developed using the Unity Engine which can build to both Android and iOS systems, It has the potential to be the first of its kind for iOS devices, as there is currently only one finite state automata simulator in the Apple Store, and it is descriptive language based in paradigm. There are also no currently available pure finite state machine libraries for C# that fits the needs and features that FSAM is implementing. It is pure in this sense if it is designed and suited for traditional automata theory functions and not for other systems such as event management and kinematics. While few FSM libraries exist, they are frequently so simple that they only implement FSA and lack non-determinism support. Because of this, the systems and algorithms of FSAM must be designed from the ground up, taking into account both the base system and the graphical user interface. Plans to open-source the project may provide other developers with a feature-rich finite state machine library, as well as avenues and opportunities to expand FSAM into other FSM fields and areas.

II. Related Literature

A. Types of Automata Simulators

Chakraborty and colleagues [1] wrote "Fifty Years of Automata Simulation: A Review" wherein they reviewed the state and trends of automata simulation, but more importantly, made classifications on automata simulators based on their design paradigms and other properties. Aside from that, they categorized the currently available automata simulators to their classifications. Figure 1 shows the classification tree of automata simulators. We can see that simulators are divided into two major classifications which are language based and visualization centric, which further divides into sub-classifications. These will be important in analyzing the currently available automata simulators, not just in PCs but in all platforms.

The language-based approach is the older of the two primary ways to simulating automata and it uses a symbolic language to define an automaton and simulates the run of an automaton by either compilation or interpretation of the program. It is further divided into sub-classifications.

The second primary type of simulators are visualization centric automata simulators. These simulators use high-quality graphics, interface, and animation to show automaton behavior. Inferring from its title alone, these simulators are visual in nature. They are divided based how automata are specified, either in structured forms or diagrammatic form.

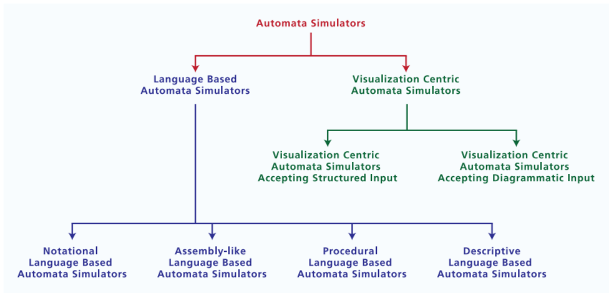

Automata Simulators
Language Based Automata Simulators
Visualization Centric Automata Simulators
Visualization Centric Automata Simulators Accepting Structured Input
Visualization Centric Automata Simulators Accepting Diagrammatic Input
Notational Language Based Automata Simulators
Assembly-like Language Based Automata Simulators
Procedural Language Based Automata Simulators
Descriptive Language Based Automata Simulators

Fig. 1. Classification of Automata Simulators

In general, older automata simulators employ mostly language-based simulators, but as time goes on and technology advanced, visualization centric automata simulators became more prevalent due to their user-friendliness and visual nature. Although to this day, simulators categorized in both classifications are still being used in one way or another.

B. UI and UX

UI and UX are indispensable to the development of modern software, and their significance is growing. In a market saturated with applications, having a good UI and UX is a simple way to stand out. The difficulty of aesthetics and visual design stems from the fact that everyone has their own opinion and preference. Nonetheless, numerous designers and professionals in these fields have developed principles and rules that serve as guidelines for developing good UI and UX. Yablonski's ”Laws of UX” [10] is a collection of twenty-one best practices that designers can use to create a positive user experience (UX). Nielsen's Usability Heuristics [9] comprise ten guidelines for enhancing the usability of UI design. We use these laws and principles as a basis for critiquing and evaluating existing simulators for automata. Furthermore, we can utilize these as guides to the UI design and UX of FSAM. Numerous publications and studies cite various principles and varying numbers of them, but when compared, they contain the same principle, are derived from other principles, or are simply stated differently.

C. PC Simulators

The PC platform have the majority of automata simulators that are currently available, this is because it is the oldest platform compared to web and mobile. It is worth noting that the previous examples of simulators stated in the previous subsection are all on the PC platform.

Fransson [11] compared nine automata simulators for the PC platform including JFLAP which we established is the de-facto standard of automata simulators. The other eight are Automata Editor, Auto Ed by Max, Automaton Sim, JFAST, PetC, Tuatara, Turing Machine Simulator and Visual Automata Simulator. Aside from the paradigm that a simulator uses, we can also classify a simulator by the types of machines it supports like Finite State Automata, Pushdown Automata, and Turing Machine. Fransson evaluated the simulators based on functionality, tools, user friendliness, layout/design, compatibility, and documentation. After which the grades are added and this determines the ranking of the simulator. It is also worth noting that all of these results are based on the subjective reasoning of Fransson, but each rating that they did didn't fall short on explanations and evaluations. The results and evaluation still provide a good insight on the usability and features of a particular simulator. Unsurprisingly, JFLAP came out on top with the highest total score.

D. Web Simulators

Web has more simulators than mobile but significantly less. By web simulators we refer to simulators that can run in a web browser and not websites hosting downloads to automata simulators, which is often the case. The web simulators being evaluated are the first results when we search for “finite automata simulators online” in Google. By paradigm, Automaton Simulator [12] and DFA – Simulator [13] are visualization centric accepting diagrammatic input; Finite automaton simulator [14] is visualization centric accepting structured input; and, FSM Simulator [15] and Automaton Simulator [16] are descriptive languaged based. These were evaluated by examining their features and quality.

The simulators vary in terms of layout and design quality, versatility, and offered features. In the area of UI design, it is important to note that the majority of sites were dated. Some glaring issues include transition minimalism issues, poor UX when creating transitions (contrary to Neilsen's flexibility-efficiency rule), limited states, and paradigm constraints (for language-based)

E. Mobile Simulators

The mobile platform has the fewest available simulators. The mobile simulators being evaluated are the first results when we search for “finite state automata” in Google Play, which is one of the largest markets for mobile applications. Furthermore, the results have been filtered to those that are true automata simulators, that is fitting into one of the classifications of the automata specified in the first subsection. There are many applications that also offer modules and quizzes regarding automata theory but have no construction or simulation features, those are also excluded.

By paradigm, Automata [6] is descriptive language-based; Finite Automata [8] and Automata Simulator [7] are visualization centric accepting structured input; and, Finite State Machine FREE [17], AutomataTaker [18], and CMSimulator [19], [20] are visualization centric accepting diagrammatic input. Some of the user interface design flaws included: the use of default Android JDK graphics; consistency issues; design flaws that conflict with system visibility, recognition, and flexibility; and outdated design systems. In terms of UX, several issues were also identified, ranging from: limitation of paradigm; poor user-friendliness and intuitiveness for diagram creation; limited transition and state characters. For features, the range on the capability of the application is wider, with some having support for Turing Machines, and Context-Free Grammar. Although some simulators only have support for diagramming and not for simulation. CMSimulator, created by Chuda and colleagues [19], [20], is arguably the best automata simulator among the reviewed simulators. However, discoverability is hindered by its name. Its UI is simple and employs widgets similar to Material UI (MUI). Concerning its UX, however, the following issues were identified: redundancy issues with certain features (contrary to Hick's Law) and an inefficient simulation string input system.

F. Evaluation Models

To determine the effectivity and usability of an application, various models and methods have been developed.

One of these models is the Technology Acceptance Model originally developed by Fred Davis. According to TAM, the success of new technology adoption is built on positive attitudes toward two factors: perceived usefulness and perceived ease of use [21]. Many scholars have subsequently developed and modified this model, developing more intricate variants, but the original model is still utilized and widely recognized [21]. The System Usability Scale (SUS) is an effective and popular evaluation instrument that measures the usability of a system, such as software, websites, mobile applications, or hardware [22]. Similar to TAM, modifications and extensions to the original SUS have been produced.

The Usefulness, Satisfaction, and Ease of Use (USE) Questionnaire is a 30-item questionnaire scored on a 7-point Likert scale with four scales: usefulness, ease of use, ease of learning, and satisfaction [23]. Diverse types of surveys were employed to measure user opinions toward a variety of consumer goods. Following each study, factor analysis revealed that customers evaluated items primarily based on three dimensions: usefulness, satisfaction, and ease of use. which became the basis of the USE Questionnaire [23].

In addition to the aforementioned instruments, we also have Questionnaire for User Interface Satisfaction, Perceived Usefulness and Ease of Use, Nielsen's Heuristic Evaluation, and Computer System Usability Questionnaire [24].

III. Materials and Methods

The study aims to develop a finite automata simulator for mobile using the design paradigm of visualization centric automata simulator accepting diagrammatic input to address the lack of automata simulators for the mobile platform and gaps identified from existing mobile automata simulators in the market. This chapter discusses the tools that is used to develop the application, implementation of the features, and evaluation of the developed application.

A. Development Tools

The mobile application is developed using the Unity Engine, which is a cross-platform 3D/2D game engine and integrated development environment for developers [25]. C# is used as the main programming language, which is a modern, object-oriented, and type-safe programming language developed by Microsoft [26] and is integrated with Unity. The IDE for writing code is Visual Studio 2022 and primarily tested on Android 11. Unity is used to test the adaptability of the user interface for various device screen sizes and performance of the application using the adaptive performance tools with changing number of cores, CPU level, GPU level, and thermal level. The application is developed on a Windows 11 64-bit machine, with Intel Core i77500U processor, and 16 GB of RAM. The app has a minimum API level of Android 5.1 ”Lollipop”. Java and Intellij IDEA IDE was used for developing and conducting unit tests with the JFLAP java archive.

B. Main Menu

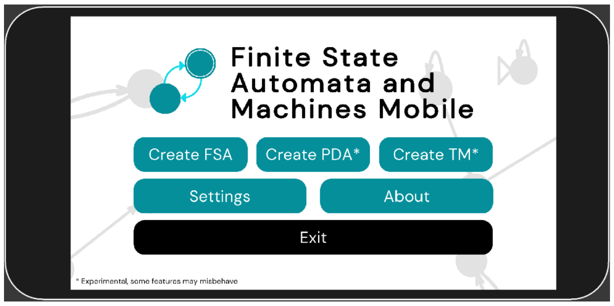

Finite State Automata and Machines Mobile
Create FSA
Create PDA*
Create TM*
Settings
About
Exit
* Experimental, some features may misbehave

Fig. 2. FSAM Main Menu

The current version of the main menu includes the create, settings, and about options. A user can create either FSA, PDA or TMs. In its current state, settings is a placeholder for future application options. The about section provides information about the application and developer contact information.

C. Automata Configuration

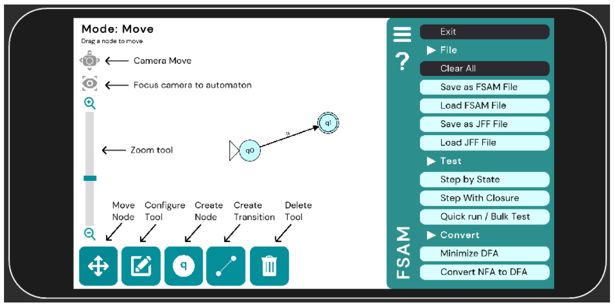


Fig. 3. FSAM Automata Configuration Scene (with tooltip toggled)

The primary window for configuring automata consists of four major sections: status texts (upper left), camera controls (left), main controls (lower left), and sidebar (right). The only differences between screens of the configuration scene are the controls in the lower left corner and the content of the sidebar.

D. Camera Controls and Status Text

There are three controls for the camera. The zoom function allows the user to zoom in and out of the canvas. The second tool is the camera move tool, which allows the user to drag the screen to move the canvas. The camera focus tool allows the user to focus the automaton on the screen by centering the automaton on the canvas, adjusting the zoom to fit the automaton on the screen, and adjusting whether the sidebar is open or not.

The status texts provide feedback and information to the user regarding the currently selected or running tools or functions. It also displays instructions on how to use the application's tools and functions.

E. Automata Configuration Home Screen

The home screen, which is also the initial screen, contains the automata configuration controls that has controls for moving and editing states, and creating and deleting states and transitions.

1) Move state: The move state tool lets the user move the states around the canvas.

2) Configure state Tool: Set state as start state, set state as final state, and rename a state are among the configuration options available for states. A menu containing configuration options appears when a state is tapped and is anchored to the upper left of the tap location on the state.

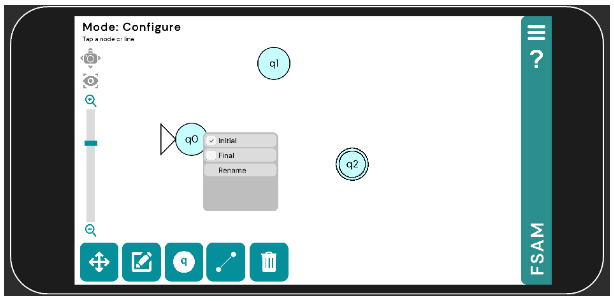


Fig. 4. Configure state Menu

3) Create State Tool: This tool permits the creation of states. It is named by default using a "q" character appended with the state ID integer upon creation. The state ID is determined by an accumulator beginning at index O. When a.fsam or.jff file is loaded, the maximum ID of the save file determines the accumulator. Through the state configuration tool, the user can later alter the names of the states.

4) Create Transition Tool: This tool allows the creation of transitions between states. Transitions are created by dragging from a state which will become the start state and to another state which will become an end state.

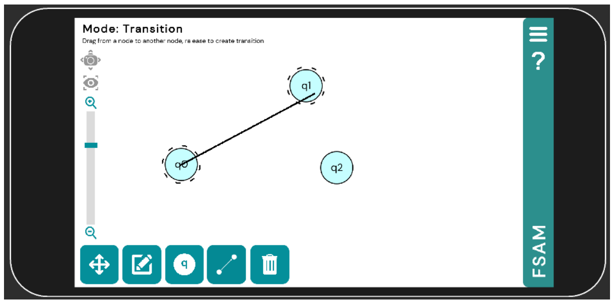

Mode: Transition
Drag from a node to another node, release to create transition
q1
q0
q2
?
FSAM

Fig. 5. Creating a transition between two states

It will display a modal input window for the transition character. Empty strings are not permitted, and the user can add an epsilon transition by clicking the **epsilon** character in the modal window. Any previously utilized transition characters are added in the scrollable part of this window for easier access and creation of transitions. Indicators with the use of broken circles to show the selected states are put into place for the users to know which states are currently selected. After dragging from the start state to the end state, a transition modal window will appear wherein the user may input the transition character

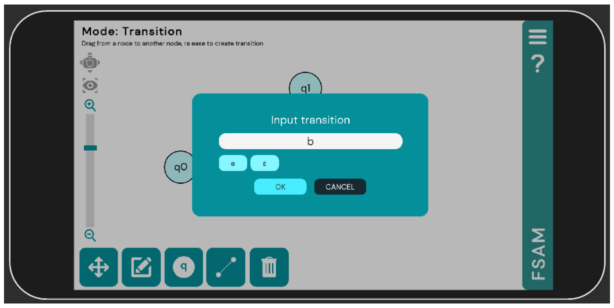

Mode: Transition
Drag from a node to another node, release to create transition
q1
q0
Input transition
b
a
ε
OK
CANCEL
?
FSAM

Fig. 6. Create Transition Modal Window

5) Delete Tool: This tool allows the user to delete state and transitions. If a state is tapped, it deletes the states, as well as any in-going and out-going transitions from that state. If a transition character is tapped, adjustments are made to the other transitions affected or in relation with the transition automatically.

F. File Saving and Loading System

FSAM has two options for its file I/O, **FSAM** and **JFF** files. A flag variable determines whether changes were made to the automaton after it was created from an empty/cleared canvas or after it was loaded from an FSAM or JFF file. If the save button for either format is pressed and the flag is set to true, a modal warning will appear asking if the user wants to proceed with the operation because the current configuration has not yet been saved.

FSAM files are one-of-a-kind save and load files created specifically for the application. FSAM files are saved in JSON format and contain a serialized version of the automaton configuration that is being saved. Files with the extension are also deserialized using JSON for loading of FSAM files. For serialization and deserialization, the **Newtonsoft Json library** is used. FSAM files are more accurate than JFF files in their positioning in the canvas since the state positions that are being saved with the file are the same as the coordinate system of Unity.

JFF files are based on the JFLAP save file and are designed to be compatible with JFLAP. JFF files are saved in XML format and contain a serialized version of the automaton configuration saved by mimicking the JFLAP save file. For loading JFF files, files with the extension are deserialized using XmlDocument Xdoc. The application converts the coordinate system of Unity to the coordinate system of the Java UI of JFLAP for FSAM to create JFF files. For loading JFF files, the process of position conversion is reversed. Because of the difference in the position system of Unity and Java, JFF files are less accurate in positioning than FSAM files.

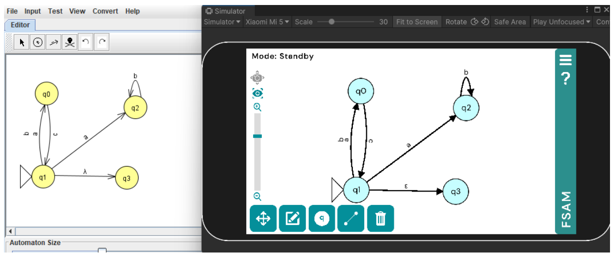


Fig. 7. A JFF file opened with both FSAM and JFLAP

G. Automata Simulation

In FSAM, there are three operations that allow for the simulation of automata: step by state, step with closure, and bulk testing. These operations mainly differ in their initialization and stop conditions but only call upon one function for the simulation logic.

1) Step Simulations: Step by State and Step with Closure: The step simulations allow the user to track the simulation run states at each step and iteration.

a) Controls and Interface: First, the user enters the test string in the input field located in the upper right corner of the screen (empty test strings are accepted), and then presses the start run button. If a simulation is already in progress, the run button changes to a stop button, allowing the user to halt the simulation. In addition, the input field is disabled until the current simulation has completed execution. The user may progress the simulation by tapping the step button.

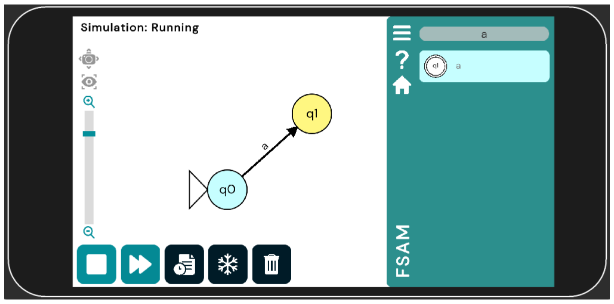


Fig. 8. Step Simulation Interface when a simulation is running

The user can select a simulation run state by tapping on the state in the right side, and it allows the viewing of the trace history for that run state, freezing/thawing of a state, and removal of a state.

b) Freezing, Thawing, and Removing Run States: A frozen run state will not step to the next state with the next step button pressed until the state is thawed with the same button that froze it. In the run states scroll view, a frozen state will be highlighted in purple. Run states in the accepted (green) or rejected (red) categories cannot not be frozen.

A run state can be removed by tapping on it and then pressing the remove button, which removes the selected run state from the simulation execution.

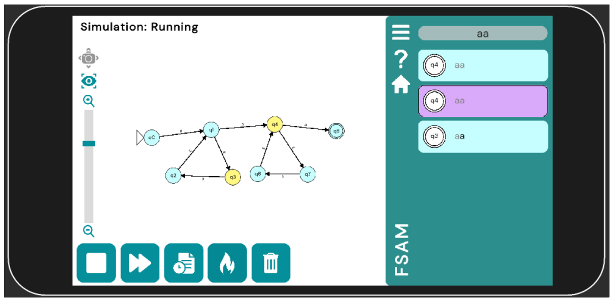


Fig. 9. A frozen run state along with a selected run state

c) Trace History: By first choosing a state and then pressing the trace history button. The trace of the previous states of the selected run state, as well as the transition characters that allowed the run state to go through that state, can be viewed by the user.

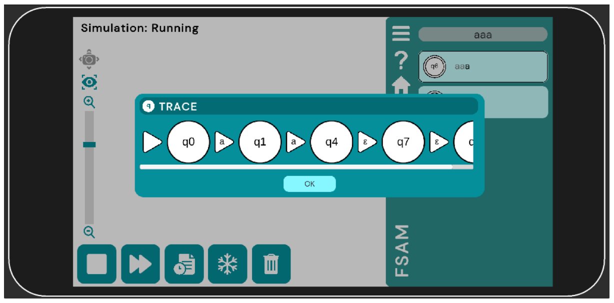


Fig. 10. Trace History Sample

2) Bulk Testing: Bulk testing enables the simulation of multiple test strings; however, in this mode, we are unable to view the individual run states and only know whether the string was accepted or rejected.

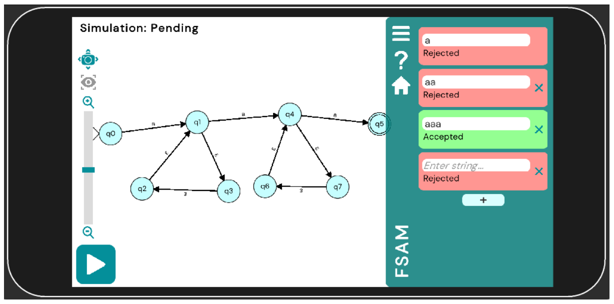


Fig. 11. Bulk Testing Interface

a) Controls and Interface: In bulk testing, we enter the test string into one of the application's input fields on the sidebar. By pressing the plus button on the side of the input fields, we can add more test strings and remove test strings. After pressing the start simulation button, the simulation results will appear at the bottom of the bulk testing input fields, along with a color indicator indicating whether it is accepted or rejected.

H. DFA Minimization

Similar to the file saving and loading system, a modal warning is presented to the user warning that major changes are going to happen with the automaton configuration. Then, the application checks if the automaton is a DFA. It is important to note that determinism in this case refers to a state not having multiple transitions for a transition character since the minimization algorithm itself would fill the missing transition characters in the automaton. In general, the **equivalence method** [27] is used to minimize the automaton. The old automaton configuration, equivalences, partitions, and new automaton configuration are also logged for it to be shown to the user at the end of the minimization execution.

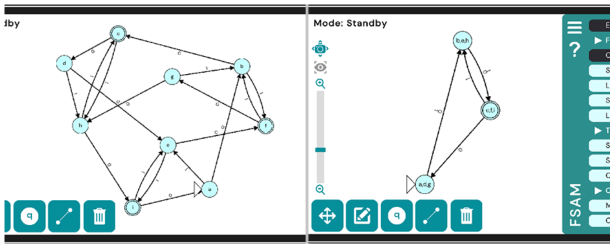


Fig. 12. NFA before and after conversion

I. NFA Conversion to DFA

Similar to the file saving and loading system, a modal warning is presented to the user warning that major changes are going to happen with the automaton configuration. Then, the application checks if the automaton is a NFA or epsilon-NFA. In general, the **subset construction method** [28] is used to convert the automaton. The old automaton configuration and new automaton configuration are also logged for it to be shown to the user at the end of the conversion execution.

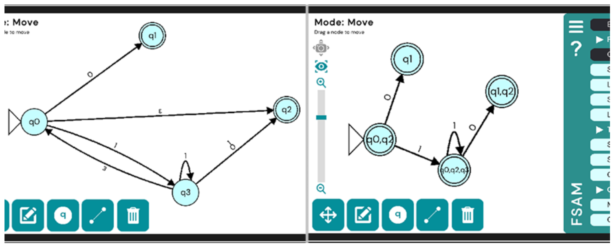


Fig. 13. NFA before and after conversion

J. Finite State Machines Supported

1) Finite State Automata: FSAM's Finite State Automata capabilities includes the diagramming, simulations, saving and loading, and NFA to DFA conversion features.

2) Pushdown Automata: FSAM's Pushdown Automata capabilities include diagramming, simulations, and saving and loading. However, step by closure is disabled for simulations. The transition creation system implemented for PDA consists of three fields, namely read, push, and pop fields, and permits multicharacter input. Step simulations for PDA include a stack and string view. PDA also includes a toggle for accept by final state or accept by empty stack option.

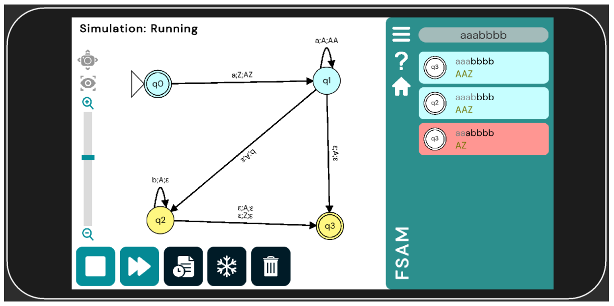


Fig. 14. Pushdown automata in FSAM

3) Turing Machines: FSAM's Turing Machines capabilities include diagramming, simulations, and saving and loading. However, step by closure is not inherently applicable for TMs. The transition creation system implemented for TMs consists of three fields, namely read, write, and direction fields. Step simulations for TMs include the view for the infinite tape and read and write head.

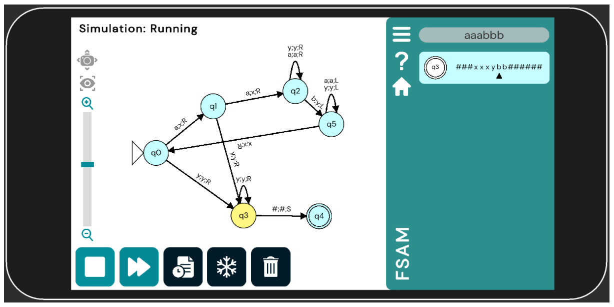


Fig. 15. Turing Machine in FSAM

K. Unit Testing and Validation of Correct Outputs

FSAM's simulation and conversion features undergo automated and programmatic unit testing to verify simulation output and other application features. In summary, FSAM generates a random automaton and runs it through the function being tested, creating a custom log file. The results are inputted to a Java validator, the same automaton is used with the equivalent JFLAP function executed, and the log file is compared. If results match, the test passes; otherwise, it fails. Each unit test is recorded in two log files: a log file specific to the function that contains the accumulated number of tests, successes, and failures; and a log file for all tests that contains the automata configuration, inputs, and outputs.

However, this leaves open the possibility that certain configurations are not tested and produce incorrect results, regardless of the number of tests conducted. Because of this, an app-accessible bug and problem tracker has been created, which redirects to a github repository for tracking FSAM bugs, where users can submit encountered issues for the developer to resolve in a future patch or update.

L. User Evaluation

The application is tested by students who are currently taking and taken the CMSC 141: Formal Languages and

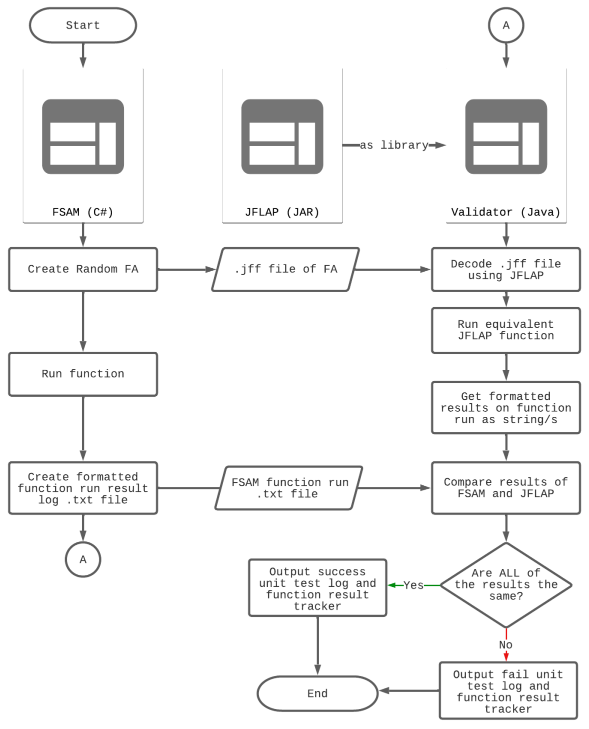

Start
A
FSAM (C#)
JFLAP (JAR)
as library
Validator (Java)
Create Random FA
.jff file of FA
Decode .jff file using JFLAP
Run equivalent JFLAP function
Run function
Get formatted results on function run as string/s
Create formatted function run result log .txt file
FSAM function run .txt file
Compare results of FSAM and JFLAP
A
Are ALL of the results the same?
Yes
Output success unit test log and function result tracker
No
Output fail unit test log and function result tracker
End

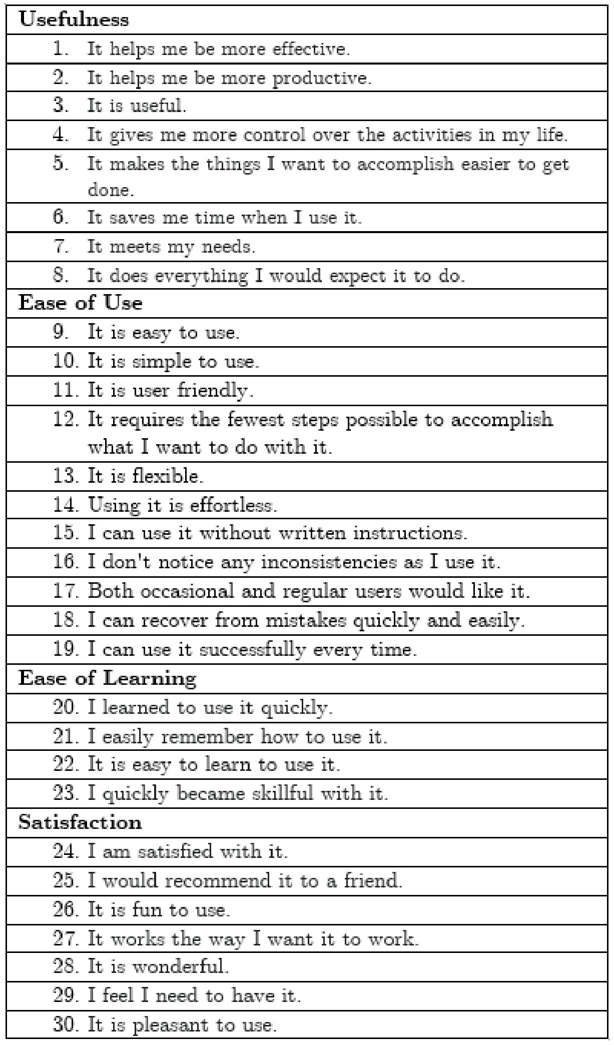
Fig. 16. Unit Testing and Validation Workflow

Automata Theory course offered by the Institute of Computer Science of the College of Arts and Sciences in University of the Philippines – Los Baños. The course is being taught with the aid of JFLAP, meaning that respondents of the study already have a background in using automata simulators, particularly in the design paradigm of visualization centric automata simulators accepting diagrammatic input. Android devices is used for testing, and respondents must have phones running the Android operating system. The APK of the application is provided to the respondents.

Users complete the Usefulness, Satisfaction, and Ease of Use (USE) Questionnaire after evaluating the functionality of the application. The USE Questionnaire consists of four scales: Usefulness, Ease of use, Ease of Learning, and Satisfaction. Each statement is evaluated with a 7-point Likert-type scale ranging from “strongly disagree (with a score of 1) to “strongly agree” (with a score of 7). The usefulness questions cover items 1 to 8, the ease of use questions cover items 9 to 19, the ease of learning questions cover items 20 to 23, and the satisfaction questions cover items 24 to 30. An additional section is also included at the end of the form for providing optional qualitative feedback.

It is also important to note that the users only evaluated the FSAM version that contains only FSA. PDA and TMs have been added to FSAM past the user evaluation and unit testing phases. However, since the tools and features of PDA and TMs are identical to those of FSA with a few modifications, the evaluation is still applicable.

The average of each scale of the questionnaire is computed and it determines the assessment of the users when it comes to the usefulness, ease of use, ease of learning, and satisfaction of the application.



Usefulness
1. It helps me be more effective.
2. It helps me be more productive.
3. It is useful.
4. It gives me more control over the activities in my life.
5. It makes the things I want to accomplish easier to get done.
6. It saves me time when I use it.
7. It meets my needs.
8. It does everything I would expect it to do.
Ease of Use
9. It is easy to use.
10. It is simple to use.
11. It is user friendly.
12. It requires the fewest steps possible to accomplish what I want to do with it.
13. It is flexible.
14. Using it is effortless.
15. I can use it without written instructions.
16. I don't notice any inconsistencies as I use it.
17. Both occasional and regular users would like it.
18. I can recover from mistakes quickly and easily.
19. I can use it successfully every time.
Ease of Learning
20. I learned to use it quickly.
21. I easily remember how to use it.
22. It is easy to learn to use it.
23. I quickly became skillful with it.
Satisfaction
24. I am satisfied with it.
25. I would recommend it to a friend.
26. It is fun to use.
27. It works the way I want it to work.
28. It is wonderful.
29. I feel I need to have it.
30. It is pleasant to use.

TABLE I
APPLICATION EVALUATION FORM USING THE USE QUESTIONNAIRE

The evaluation is performed using Google Forms, which also stores the e-mail addresses of the respondents. A respondent may only answer once and must use their UP mail accounts, which further strengthens the criterion for valid respondents in the study. This is also done to guarantee that the results have not been manipulated or fabricated by data tampering or forgery.

A web page for information, downloadables, and testing procedures has also been created to guide respondents and avoid confusion when testing the application. The website can be accessed through https://sites.google.com/view/fsam/home

IV. RESULTS AND DISCUSSIONS

A. Automated Unit Testing Results

The output-related components of the program were subjected to automated unit testing, including string testing for acceptance or rejection, step by state, step with closure, NFA conversion, and DFA minimization. 200 tests were done for

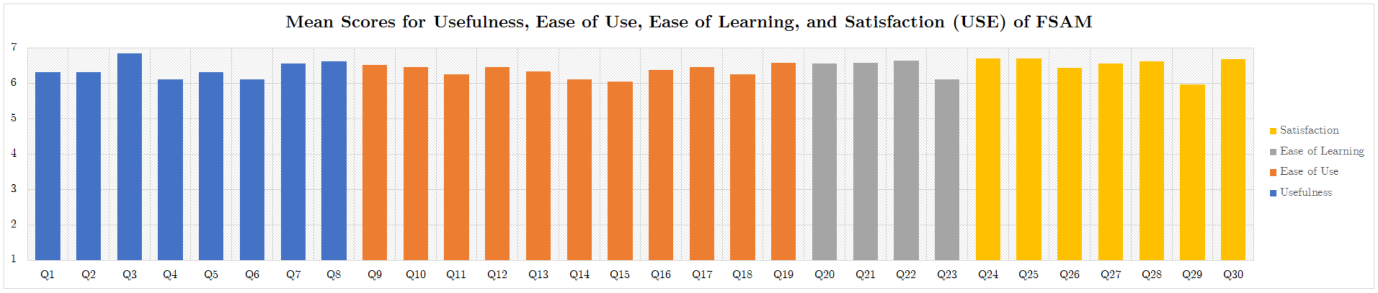


Fig. 17. Mean Scores for Usefulness, Ease of Use, Ease of Lrarning, and Satisfaction (USE) of FSAM

each of the features being tested, totalling to 1000 tests overall, all of which resulted in a 100% success rate. All of the tests that have been executed, along with the complete details on automata configuration, input, and output, are recorded in the run log file.

In addition, if additional unit tests are required or if the code for the functions has been rewritten in a way that may have altered the behavior of the feature, it is simple and quick to execute tests because they are automated.

B. User Evaluation Results

A total of 32 individuals fitting the criteria for valid testers have volunteered for the user evaluation. The interpretation for the 7-point Likert scale is the following: 1 - Strongly disagree; 2 - Disagree; 3 - Somewhat disagree; 4 - Neutral; 5 - Somewhat agree; 6 - Agree and; 7 - Strongly agree.

1) Usefulness: Blue bars in Figure 17 shows the mean scores for each of the questions on the usefulness scale. ”It helps me be more effective” got 6.31, ”It helps me be more productive” got 6.31, ”It is useful” got 6.84, ”It gives me more control over the activities in my life” got 6.13, ”It makes the things I want to accomplish easier to get done” got 6.31, ”It saves me time when I use it” got 6.13, ”It meets my needs” got 6.56, ”It does everything I would expect it to do” got 6.63. All of the questions under usefulness got a mean score between 6 (agree) and 7 (strongly agree).

2) Ease of Use: Orange bars in Figure 17 shows the mean scores for each of the questions on the ease of use scale. ”It is easy to use” got 6.53, ”It is simple to use” got 6.47, ”It is user friendly” got 6.25, ”It is flexible” got 6.47, ”It requires the fewest steps possible to accomplish what I want to do with it” got 6.34, ”Using it is effortless” got 6.13, ”I can use it without written instructions” got 6.06, ”I don’t notice any inconsistencies as I use it” got 6.38, ”Both occasional and regular users would like it” got 6.47, ”I can recover from mistakes quickly and easily” got 6.25, ”I can use it successfully every time” got 6.59. All of the questions under ease of use got a mean score between 6 (agree) and 7 (strongly agree).

3) Ease of Learning: Gray bars in Figure 17 shows the mean scores for each of the questions on the ease of learning scale. ”I learned to use it quickly” got 6.56, ”I easily remember how to use it” got 6.59, ”It is easy to learn to use it” got 6.66, ”I quickly became skillful with it” got 6.13. All of the questions under ease of learning got a mean score between 6 (agree) and 7 (strongly agree).

4) Satisfaction: Yellow bars in Figure 17 shows the mean scores for each of the questions on the usefulness scale. ”I am satisfied with it” got 6.72, ”I would recommend it to a friend” got 6.72, ”It is fun to use” got 6.44, ”It works the way I want it to work” got 6.56, ”It is wonderful” got 6.625, ”I feel I need to have it” got 5.97, ”It is pleasant to use” got 6.69. All of the questions under usefulness got a mean score between 6 (agree) and 7 (strongly agree) except for the prompt ”I feel I need to have it” which got 5.97 but still near the range and falls under the agree side.

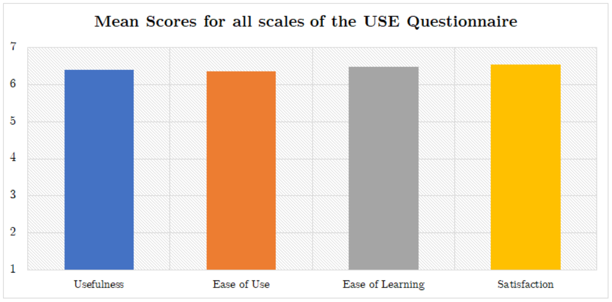


Fig. 18. Mean Scores for all scales of the USE Questionnaire

5) Overall Result: Figure 18 shows the mean scores for all scales of the USE Questionnaire. Usefulness got 6.40, Ease of Use got 6.36, Ease of Learning got 6.48, Satisfaction got 6.53. All of the scales got a mean score between 6 (agree) and 7 (strongly agree), meaning respondents overall agree-strongly agree to the usefulness, ease of use, ease of learning, and satisfaction of FSAM.

C. Qualitative Comments of evaluators

The application’s testers also provided qualitative feedback, which primarily consisted of bug reports, suggestions for new features, and praise for the application. The majority of the reported bugs by respondents have been resolved since the evaluation. Testers praised the application’s simple and intuitive user interface design, features related to displaying more information, and features related to usability.

Implementing multitouch zoom is the most frequently recommended feature. It was attempted to be implemented, but in its current state it is temperamental and buggy, so it may be implemented in a future release. Other suggestions for features, such as cancel for transition modal window, selected function indicators, and color change for improved UX, have also been implemented. The implementation of an undo/redo system will take some time, so it will not be included in the current release. However, it may be included in a future version.

V. Conclusion and Recommendations

Based on the results of the user questionnaires, FSAM has proven to be an excellent application in terms of its usefulness, ease of use, ease of learning, and user satisfaction. It is capable of diagramming and simulating finite state automata; saving and loading FAs with JFLAP cross-compatibility; as well as performing conversion and minimization; all on a mobile platform. Due to the additional features, diagramming, simulation, and file I/O are also supported for PDAs and TMs. Making FSAM a feature-rich automata theory application.

However, based on the qualitative evaluation of the respondents for the application, such as multi-touch controls, an undo and redo system, and additional UX enhancements, there is still considerable room for improvement.

Additionally, the application can be published on the Google Play Store for Android so that it is accessible to all users. The application may also be released for iOS, but it will require separate testing and evaluation to ensure its quality and dependability.

It is also possible to expand and extend the application to other finite state machines (FSMs) or other variations of the FSMs already in FSAM, such as Mealy Machines, Moore Machines, and other automata theory-related features, since FSAM in its current state can serve as a base for a variety of expansions and modifications, as evidenced by the post-evaluation additions of PDA and TMs. Since FSAM is also built on a game engine, it can be extended to create finite state machine and automata theory-based games and interactive learning modules to gamify the subject's learning process, a case in point being the Java-based computer app Kara [29]. Due to the modularity of the FSAM codebase, the expansion and extension procedures are future-proof. There are plans to open-source the application for the benefit of developers who wish to use the library, as well as to improve and expand the current state of FSAM.

References

[1] P. Chakraborty, P. C. Saxena, and C. P. Katti, "Fifty years of automata simulation: A review," *acm inroads*, vol. 2, no. 4, pp. 59–70, 2011.

[2] *How many people have smartphones worldwide*, Oct. 2022. [Online]. Available: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

[3] *Basics of automata theory*. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html.

[4] *Applications of automata theory*. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/apps.html.

[5] N. Chomsky, "Three models for the description of language," *IRE Transactions on information theory*, vol. 2, no. 3, pp. 113–124, 1956.

[6] M. Dahir, *Automata*, 2021. [Online]. Available: https://play.google.com/store/apps/details?id=com.masrik.automation.

[7] P. Chakraborty, *Automata simulator*, 2018. [Online]. Available: https://play.google.com/store/apps/details?id=com.singh.tuhina.automatasimulationcopy.

[8] Scolabs, *Finite automata*, 2015. [Online]. Available: https://play.google.com/store/apps/details?id=project.scolary.mlearning.

[9] F. K. Mazumder and U. K. Das, "Usability guidelines for usable user interface," *International Journal of Research in Engineering and Technology*, vol. 3, no. 9, pp. 79–82, 2014.

[10] J. Yablonski, *Laws of UX: Using psychology to design better products & services*. O'Reilly Media, 2020.

[11] T. Fransson, *Simulators for formal languages, automata and theory of computation with focus on jflap*, 2013.

[12] K. Dickerson, Feb. 2021. [Online]. Available: https://automatonsimulator.com/.

[13] A. Kashyap, Apr. 2020. [Online]. Available: https://bing101.github.io/DFA-simulator/.

[14] C. Burch. [Online]. Available: https://www.cs.cmu.edu/~cburch/survey/dfa/index.html.

[15] I. Zuzak and V. Jankovic, Feb. 2022. [Online]. Available: https://ivanzuzak.info/noam/webapps/fsm_simulator/.

[16] D. Doty. [Online]. Available: https://web.cs.ucdavis.edu/~doty/automata/.

[17] M. Reacher, *Finite state machine free*, 2014. [Online]. Available: https://play.google.com/store/apps/details?id=com.mountainreacher.automata.

[18] Inixel, *Automataker*, 2015. [Online]. Available: https://play.google.com/store/apps/details?id=com.IniXel.AutomaTaker.

[19] D. Chuda, J. Trizna, and P. Kratky, "Android automata simulator," in *International Conference on e-Learning*, vol. 15, 2015, p. 80.

[20] D. Chuda, *Cmsimulator*, 2020. [Online]. Available: https://play.google.com/store/apps/details?id=fiitstu.gulis.cmsimulator.

[21] R. Allen, *Digital marketing models: The technology acceptance model*, Nov. 2020. [Online]. Available: https://www.smartinsights.com/manage-digital-transformation/digital-transformation-strategy/digital-marketing-models-technology-acceptance-model/.

[22] *Technology acceptance model*, Aug. 2021. [Online]. Available: https://edutechwiki.unige.ch/en/Technology_acceptance_model.

[23] A. Lund, "Measuring usability with the use questionnaire," *Usability and User Experience Newsletter of the STC Usability SIG*, vol. 8, Jan. 2001.

[24] *Usability and user experience surveys*. [Online]. Available: https://edutechwiki.unige.ch/en/Usability_and_user_experience_surveys.

[25] A. Sinicki, *What is unity? everything you need to know*, Mar. 2021. [Online]. Available: https://www.androidauthority.com/what-is-unity-1131558/.

[26] *A tour of the c# language*, Sep. 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[27] *Dfa minimization*, Jul. 2021. [Online]. Available: https://www.tutorialspoint.com/automata_theory/dfa_minimization.htm.

[28] R. Dougherty, *Conversion of nfa to dfa (powerset/subset construction example)*, Mar. 2020. [Online]. Available: https://www.youtube.com/watch?v=jMxuL4Xzi_A.

[29] W. Hartmann, J. Nievergelt, and R. Reichert, "Kara, finite state machines, and the case for programming as part of general education," in *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments (Cat. No. 01TH8587)*, IEEE, 2001, pp. 135–141.

Von Vincent O. Vista A senior BS Computer Science student from the University of the Philippines - Los Baños. He is from Pakil, Laguna; he is a member of several volunteer organizations in the province; and he has participated in a number of software development projects, ranging from freelance to contracted work for both privately and publicly funded initiatives.