Modelling Candy Crush Saga using Formal Language Automata

Gopa Pulastya
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22116@bl.students.amrita.edu

Harshitha Reddy
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22108@bl.students.amrita.edu

Rishi Anirudh K
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22139@bl.students.amrita.edu

C. Lakshmi Sindhu
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie22126@bl.students.amrita.edu

Niharika Panda
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
p_niharika@blr.amrita.edu

Abstract— **Recently, there has been an increasing number of advances in the field of formal methods for game analysis, with a particular focus on the modelling and optimization of all aspects of game dynamics and player experience. This paper revisits formal modelling of the popular match-three-puzzle game Candy Crush Saga within a Non-Deterministic Finite Automaton framework. By studying these key game dynamics such as state transitions for candies, special candy formations, and the activation of boosters, we represent the complex decision-making processes and randomness intrinsic to gameplay. The NFA model captures different aspects and transitions from success conditions to failure conditions across different levels, giving us better insights into game strategies and structures. With tools like JFLAP, the process of modelling becomes visual, formalized, and quite possibly opens AI-driven solutions with the automation of playing challenges. A comparison will be made using the Pushdown Automata framework where NFA models are benchmarked against the Pushdown Automata with the intention of finding how differently both capture a game's state transitions to bring insight to be shown in comparison.**

Keywords – Non-Deterministic Finite Automaton (NFA), Pushdown Automaton (PDA), AI-driven solutions, JFLAP, Candy Crush Saga, Transitions

I. INTRODUCTION

The complexity of modern video games has made modelling and analysis of their mechanics with formal methods increasingly interesting. Candy Crush Saga, being one of the most widely played puzzle-based games, presents a particularly interesting case for formal modelling through automata theory. The game revolves around matching candies of similar colors to achieve specific goals, create special candies from defined combinations, and apply boosters to navigate challenging levels. These mechanics require elaborate state-based interactions and decision-making processes by players, so that Candy Crush is an especially strong candidate for automata-based modelling and analysis.

This report aims to discuss the modeling of Candy Crush Saga using two different approaches: NFA and PDA. NFA can then represent non-deterministic elements such as making special candies, using boosters and state condition distinct from each level. Moving over to PDA which is equipped with a stack memory we analyze how playing memory-dependent actions like tracking progress will be done in game modes involving complex state transitions and the conditions of backtrack can be modeled to demonstrate distinction and advantages this approach provides.

The report details a comparative analysis between NFA and PDA representations of Candy Crush through implementation and simulation within JFLAP. This provides an illustration of how formal models can help reveal aspects of the complexity of gameplay while potentially informing strategies for AI-driven solutions, automated gameplay, or optimization of game mechanics. These models serve both as a means of pedagogical representation in understanding game mechanics from an automata perspective and have more advanced applications for further computational analysis.

By implementation and simulation using JFLAP, this report is a comparative comparison between the NFA and PDA representations of Candy Crush. It shows formal models may be used in various aspects of gameplay to perhaps guide strategies in AI-driven solutions, automated gameplay, and optimizing game mechanics. Such a model offers a didactic tool for understanding gameplay from an automata perspective while being a base for even further advanced computational analysis.

II. RELATED WORKS

De Souza et al. proposed a teaching-learning methodology to improve the understanding of formal languages and automata theory by combining theoretical explanations with practical activities using tools like JFLAP. They demonstrated improved comprehension and problem-solving skills compared to traditional methods [1]. Cachat studied infinite two-player games on pushdown graphs, focusing on reachability and Büchi games. The study developed algorithms to compute winning regions and strategies using automata-theoretic approaches, with applications in model checking and program synthesis [2]. Carayol et al., made a study on higher-order pushdown games, analyzing winning regions, and strategy synthesis for more complex games modeled with pushdown automata. They contributed to the game theory by extending the concept of strategy synthesis to higher order systems [3]. Baicoianu et al. studied the use of automata in game design, which is how finite automata optimized game mechanics, player interactions, and decision-making systems and provided a structured approach toward modeling game states and transitions [4]. Lee et al. proposed a DFA-based technique to recognize basketball events and illustrated high accuracy in event detection through real-world data with implications for sports analysis and broadcasting technologies [5]. Vayadande et al. emphasized the practical implementation and testing of a DFA machine for recognizing regular languages. They discussed its design, performance evaluation, and optimization for real-time applications [6]. Jamil et al. described an infinite runner game modelled using automata theory, highlighting how automata simplified game mechanics and enhanced gameplay logic through structured state transitions [7]. Li et al. designed the game of Ludo, with finite automata in the description of state transitions and gameplay. The authors illustrated that the benefits of automata lie in the development of efficient and reliable game logic [8]. Panda et al. discussed a research analysis of various designs in scanner and parsers regarding performance, accuracy, and efficiency. The researchers presented comparison between the old-fashioned approaches and the newer methods used in health and engineering; they further exhibited how such methods could optimize computational workflow [9]. Prajana et al. proposed a method in recognizing patterns in which the researcher used finite automata to detect the occurrence of strings in text files. The approach used deterministic as well as nondeterministic finite automata for conducting efficient search and matching in patterns. The research worked out the applicability of finite automata for reduction of complex search operations that can be utilized for application in text processing and analyzing data [10]. Mathew et al. presented an automated framework with the conversion of regular expression into NFA. Their framework ensured the process to be precise in conversion. The study demonstrated the practical applications of the framework in compiler design and formal language processing and generated utility in automating the complex task [11]. Sharma et al. proposed a compiler design that is robust enough to recognize and process multiple programming languages. Techniques for lexical analysis and syntax parsing were used in the process toward multilingual support, with ample focus on the adaptability and efficiency of the compiler. Practical applications in programming language development and cross-language compatibility were emphasized [12]. Reddy et al. developed an audio encryption algorithm that used a combination of cellular automata and AES. It had made audio data more secure by using cellular automata complexity and robustness of AES. It was shown that the above approach could be used effectively to protect audio transmissions against access by unauthorized parties [13]. Pranavi et al. presented the implementation of a system search service based on a custom lexical search algorithm. The study focused on improving the efficiency and accuracy of searching by tailoring the lexical analysis process to specific use cases. The proposed method showed improved performance in retrieving relevant data from large datasets [14].

III. Methodology

A. Understanding Game Mechanics:

First comes deconstructing the core mechanics of Candy Crush Saga. Its core is based on the grid where the player should match up three or more candies of the same color to get them off the board. Other mechanics include the following:

- Special Candies:
  These are the striped, wrapped, and color bomb candies formed through certain combinations of candies.
- Boosters:
  The game provides in-game items called boosters that give players an edge by clearing rows or creating a specific type of candy.
- Objective-Based Levels:
  The levels have objectives, like scoring a certain number of points or clearing specific tiles and each level has a move limit.
- Randomness:
  New candies come after moves, and this brings some randomness to the game, hence increasing its complexity.
- Failure and Success:
  The player wins when they reach the level objectives or loses when they exhaust the number of moves without achieving level objectives.

B. Methodology for Modeling with NFA:

1. Identifying Key States:

The states of the game Candy Crush can be grouped into discrete states that help in representing key stages of play. We have discovered the following key states:

- Normal State (Initial Move):
  The player picks and matches the candies.
- Special Candy Creation:
  Transitions take place when the player produces a special candy based on particular match patterns- striped, wrapped, or color bomb.
- Booster Usage:
  The player chooses to use a booster for support while gaming.
- Random Candy Drop:
  It brings new candies randomly each time the candies are removed.
- Level Success/Failure:
  The player completes its goal or runs out of its moves.

Every movement triggers a state transition that's contingent on the actions from the player or on results from random candy drops as soon as the moves come into play.

2. Designing the NFA:

The NFA is constructed based on the identified states and transitions between them. An NFA can make more than one transition from any state for a given input. This property makes it perfect to model the non-deterministic properties of Candy Crush, like the random generation of candies or the multiple moves that could be made from a single game state.

Key elements in the design of the NFA are:

- States:

For example, game scenarios such as matching candies, special candies, usage of boosters, and completion of levels with success or failure.

- Transitions:

Defined based on user inputs or game events, for example the creation of a set of candies, or the usage of a booster.

- Alphabet:

Refers to the set of moves to be played in the game like selecting moves, generating special candies, and using boosters.

- Initial State:

The state from which a game begins.

- Final States:

They reflect the completion of a level. They can be successful, where the player has achieved all objectives or run out of moves.

3. JLAP implementation:

Using JFLAP, we could create the NFA to display states and transitions graphically. JFLAP is a tool in simulation and visualization of automata. The steps involved for the JFLAP implementation are as follows:

- Creating States:

All game states, such as normal moves, special candy creation, and booster usage, can be created as nodes through q0, q1, q2, etc.

- Transition Mapping:

The transitions of states are represented using arrows. It is labeled with the input, which can be action by the player or may be an outcome such as generating random candies.

- Representation of non-determinism:

It will allow having many transitions coming from the same state as a representation of the number of outcomes of a given action. For instance, when the same candies are matched, then different configurations of the board can come out.

- Final States:

These are the last state conditions of the game at the end of it as success or failure; by either achieving the set objects or running out of their moves.

a. Simulating Game Play in NFA:

Now, when we develop the NFA in JFLAP, we model various games purely for an assurance that this model goes well. Simulation allows the following capabilities of doing:

• Simulate Move Sequence:

Input sequences like candy matchings are simulated. It is just for the observance of state transitions; otherwise, the NFA properly describes the gameplay of the game.

• Success and Failure Conditions:

One can input sequences that represent different paths of gameplay to test how the NFA transitions to the success or failure state.

• Randomness Handling:

Nondeterminism in the automaton brings multiple possible transitions, just like how candies are randomly produced in the game.

b. Incorporating Game Complexity:

The NFA model can be extended for more complex gameplay scenarios like the following:

• Multiple Levels:

Extra states are added to model different level objectives.

• Advanced Boosters:

Bombs, transitions for their defuse, multi-level obstacles, etc.

• AI-Based Decision Making:

In addition, the model can be further improved by including AI algorithms that predict the best moves for complex levels.

C. Methodology for Modelling with PDA:

1. Converting NFA to Grammar:

Before constructing the PDA, the NFA is converted into a formal grammar. This step yields a set of production rules that reflect the state transitions:

- Grammar Construction: Production rules are drawn to depict the transitions between states, with each rule reflecting one of the many player actions and game outcomes.
- Optimization: The grammar is simplified here to be clear and efficient for the creation of the PDA.

2. Converting Grammar to PDA using JFLAP:

The given formal grammar is converted into a PDA by using JFLAP:

- Creation of State: The states are defined, in-game stages, on every basic move, creation of special candy, and usage of boosters.
- Stack Operations: Implement a stack to keep the record of moves and interactions-sequences that occurred—for instance, memory of candies or previous states.
- Transitions and Stack Manipulation: Transitions involve push/pop operations. This indicates the memory-dependent nature of PDA. For instance, pushing onto the stack the creation of some sort of special candy and popping it once it has been used.
- Initial and Final States: The same definition as in NFA, but stack operations are added to handle the state transitions

3. Extending PDA Complexity:

The PDA is augmented to capture more sophisticated in-game dynamics:

- Memory Tracking: Stack symbols keep track of sequences and dependencies, like boosters used and special candy combinations.
- Multiple Levels: More states and transitions mean more objectives and difficulties for each level.
- AI-Based Decision Making: AI-based elements may be developed to predict the best move.

4. Simulation and Validation in JLAP:

The PDA is used in JFLAP, in order to verify the correctness:

- Moves Simulation with Operations of Stack: The player does an action that triggers moves from one state to the next and changes in the stack.
- Complex State Management: The stack can model complex game conditions.

Resultant Validation: Successful paths illustrate the ending states, while failure conditions are depicted by incorrect paths.

IV. Results and Analysis

Both the NFA and PDA models served as effective representations of the mechanics of Candy Crush Saga, each at a different level of complexity. The NFA captured core gameplay mechanics such as matching candies, creating special candies, usage of boosters, and randomness of candy generation. It was effective for modeling state transitions and basic game interactions. In contrast, the PDA obtained by conversion from NFA to grammar and then to PDA using JFLAP augmented this representation by adding a stack mechanism that allowed the treatment of gameplay scenarios with nested structures such as multi-level objectives or recursive moves.

1. NFA Implementation and Simulation:

- **State Representation**:
These include states for normal gameplay, special candy creation, and usage of boosters, among others. These states were well defined using the JFLAP, with transitions triggered by a player's action or occurrence during the game.
- **Transitions**:
Transitions were implemented for moves that included matching candies, creating special candies, and activating boosters. For instance, the action of matching three candies could transition the NFA into a state of "special candy creation," provided all conditions had been satisfied.
- **Simulation Results**: (During simulation in JFLAP)

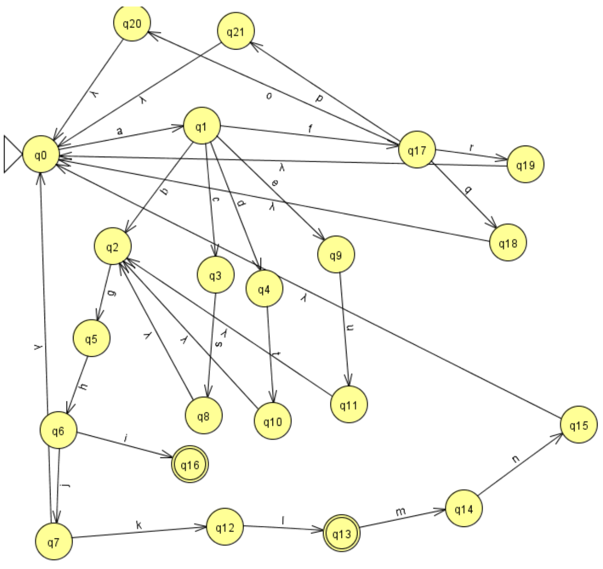
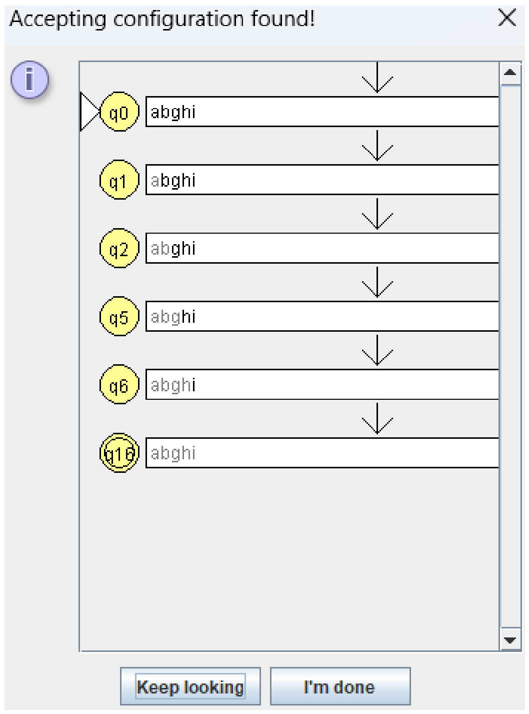


Fig1: Representation of States in the NFA Model

The figure [Fig 1] is an NFA, implemented in JFLAP. The yellow circle shows different states within the automaton. q0 is labeled as the start state and has one arrow coming in to it with no previous state, so it's where the automaton starts its computation. The arrows between the states represent transitions; each one of them has input symbols associated with it, for example, 'a', 'b', 'c', and so on. These symbols determine transitions from one state to another. In some cases, there are more than one transition from a single state on the same input symbol. This demonstrates that the automaton is non-deterministic. For example, for state q1, 'a' will take the automaton to both states q3 and q5. It can choose among the paths. The state q15, denoted with a double circle, is the final or accepting state, meaning that for any string the automaton accepts if it happens to arrive in such a state after having scanned the input string. Transitions represent the rules according to which the automaton processes strings, sometimes walking multiple paths at once, by virtue of its non-determinism. If any path leads from the initial state to an accepting state while consuming the input string, the NFA accepts the string. This flexibility makes NFAs capable of recognizing complex patterns described by their structure.

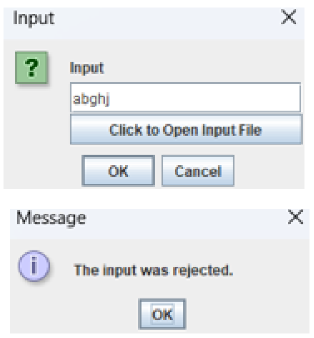


Accepting configuration found!

q0	abghi
q1	abghi
q2	abghi
q5	abghi
q6	abghi
q16	abghi

Fig 2: Representation of Accepting Configuration

The figure [Fig 2] depicts an accepting configuration in an NFA processing the input string "abghi". The automaton moves from states q0, q1, q2, q5, q6, and finally to q16, meaning that the input string is accepted because it has reached an accepting state.



Input

Input: abghj

Message

The input was rejected.

Fig 3: Representation of Rejecting Configuration

The input string "abghj" has been processed through the NFA and rejected [Fig 3]. This implies that no valid path of the initial state to an accepting state could be found by using the input provided to the automaton.

2. PDA Implementation and Simulation:

- **Stack Utilization:**

The PDA, derived by converting the NFA to a grammar and then to a PDA in JFLAP, introduced a stack to handle hierarchical and recursive gameplay mechanics.

- **Advanced Mechanics Representation:**

The PDA simulated more complex gameplay, such as:

Clearing nested objectives, where achieving one goal (like "clear all jellies") unlocked another.

Recursive transitions, where moves looped back to previous states until specific conditions were met.

- **Simulation Results:** (During implementation in JFLP)

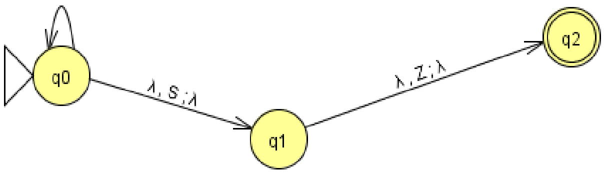
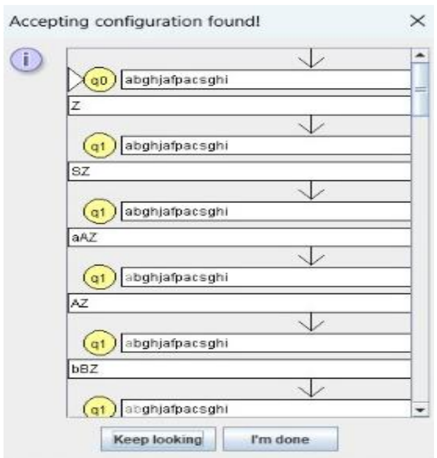


Fig 4: Representation of States and Transitions in PDA

This image [Fig 4] represents a PDA, which extends a NFA by incorporating a stack for context-free language recognition. The automaton has three states: `q0` (initial state), `q1` (intermediate state), and `q2` (final/accepting state). Transitions are labeled in the format `input symbol, stack top: stack operation`. For example, the transition from `q0` to `q1` reads a lambda (λ) without consuming input, checks if the stack top is `S`, and performs no stack modification. Similarly, the transition from `q1` to `q2` checks for `Z` at the stack top with no stack changes. The self-loop at `q0` indicates repetitive stack operations. This PDA recognizes a context-free language by validating input sequences and stack configurations, commonly used for parsing languages or matching patterns like balanced parentheses.

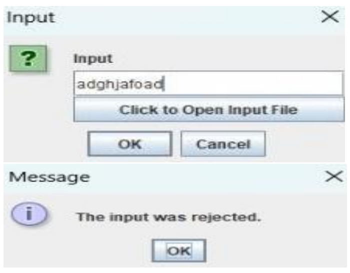


Accepting configuration found!

q0 abghjafpacsghi
Z
q1 abghjafpacsghi
SZ
q1 abghjafpacsghi
aAZ
q1 abghjafpacsghi
AZ
q1 abghjafpacsghi
bBZ
q1 abghjafpacsghi

Fig 5: Representation of Accepting Configuration

The figure [Fig 5] represents an accepting configuration of a PDA. It shows the progression of states (`q0`, `q1`, etc.), stack contents (`Z`, `AZ`, `BZ`), and input string processing (`abghjafpacsfghi`). The automaton reaches a final accepting state with the correct stack and input conditions.



Input

Input

adghjafoad

The input was rejected.

Fig 6: Representation of Rejecting Configuration

This image [Fig 6] shows a rejection configuration for a PDA. The input string `adghjafoad` does not satisfy the automaton's acceptance criteria, leading to a rejection. The PDA's transitions, stack operations, or final state conditions were not met for this input.

3. Comparative Analysis of NFA and PDA:

   a. **Complexity Representation:**

- NFA:
  - Limited to linear, straightforward state transitions (e.g., match candies → create special candy).
  - Could not effectively represent nested objectives or recursive gameplay mechanics.

- PDA:
  - Used stack operations to handle nested objectives and recursive transitions (e.g., pushing objectives onto the stack as sub-goals were introduced).
  - Accurately modeled scenarios like clearing multiple goals within a level.

   b. **Scalability**:

- NFA:
  - As complexity increased, the number of states and transitions grew significantly, making the model harder to manage and simulate.
- PDA:
  - The stack allowed efficient representation of recursive or hierarchical gameplay without requiring additional states, making it more scalable.

   c. **Simulation Accuracy:**

- NFA:
  - Successfully transitioned between states based on player actions for simple levels.

  - Struggled to represent more complex gameplay scenarios involving dependencies between actions or sub-goals.
- PDA:
  - Simulated advanced gameplay accurately by leveraging the stack for memory.

d. Randomness Handling:

- Both Models:
  - Incorporated randomness effectively through nondeterministic transitions, simulating random candy drops.
- PDA Advantages:
  - The PDA better managed random events in nested or recursive scenarios, maintaining game state continuity through its stack.

e. Ease of Use:

- NFA:
  - Easier to design and simulate due to its simpler structure.
  - Ideal for basic gameplay modeling.
- PDA:
  - More complex to implement, requiring careful grammar conversion and stack design.
  - Provided superior modeling for complex levels.

The NFA effectively modeled simple gameplay mechanics, capturing core transitions and states but fell short in representing complex, hierarchical scenarios.
The PDA extended the capabilities of the NFA by leveraging its stack to handle nested objectives, recursive transitions, and advanced gameplay dynamics.
Simulations in JFLAP confirmed that while the NFA sufficed for basic tasks, the PDA was essential for modeling the intricate mechanics of Candy Crush Saga.

V. Conclusion And FutureScope

The modelling of Candy Crush Saga using NFA and PDA provides a structured approach to capturing the game's key dynamics, such as candy matching, special candy creation, booster activation, and achieving level goals. By representing states and transitions, these models encapsulate the core mechanics, randomness, and player driven decisions within the game. While JFLAP simulations effectively illustrate deterministic and non-deterministic behaviors, the scalability of NFA models becomes challenging with increasing game complexity, and some aspects, such as randomness and strategic moves, remain difficult to capture fully. Converting NFA into PDA adds memory capability, enhancing the modelling of sequential game states and conditional gameplay paths. The future work may include probabilistic reasoning, Markov chains, or AI driven strategies to simulate more dynamic gameplay and improve automated level-solving capabilities.

VI. References

[1] De Souza, G.S., Olivete, C., Correia, R.C.M. and Garcia, R.E., 2015, October. Teaching-learning methodology for formal languages and automata theory. In 2015 IEEE Frontiers in Education Conference (FIE) (pp. 1-7). Ieee.

[2] Cachat, T., 2002. Symbolic strategy synthesis for games on pushdown graphs. In Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29 (pp. 704-715). Springer Berlin Heidelberg.

[3] Carayol, A., Hague, M., Meyer, A., Ong, C.H.L. and Serre, O., 2008, June. Winning regions of higher-order pushdown games. In 2008 23rd Annual IEEE Symposium on Logic in Computer Science (pp. 193-204). IEEE.

[4] Baicoianu, A., Dobre, C., Lopataru, M. and Plajer, I.C., 2024. Automata concepts over game design process. Bulletin of the Transilvania University of Brasov. Series III: Mathematics and Computer Science, pp.191-208.

[5] Lee, J., Lee, J., Moon, S., Nam, D. and Yoo, W., 2018, February. Basketball event recognition technique using Deterministic Finite Automata (DFA). In 2018 20th International Conference on Advanced Communication Technology (ICACT) (pp. 675-678). IEEE.

[6] Vayadande, K.B., Sheth, P., Shelke, A., Patil, V., Shevate, S. and Sawakare, C., 2022. Simulation and testing of deterministic finite automata machine. International Journal of Computer Sciences and Engineering, 10(1), pp.13-17.

[7] Jamil, A., Ullah, A. and Rehman, M., 2016. An infinite runner game design using automata theory. International Journal of Computer Science and Software Engineering, 5(7), p.119.

[8] Ali, K.F., Kalyan, V. and Kumar, K.A., 2019, March. Design and implementation of ludo game using automata theory. In 2019 Innovations in Power and Advanced Computing Technologies (i-PACT) (Vol. 1, pp. 1-6). IEEE.

[9] Asmitha, M., and Niharika Panda. "Exploring Different Methods of Scanners and Parsers." 2024 International Conference on Advances in Modern Age Technologies for Health and Engineering Science (AMATHE). IEEE, 2024

[10] Prajana, Mamidi, et al. "Pattern Recognition for Identifying a String Within a Text File Using Finite Automata." 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2024.

[11] Mathew, Joel, et al. "Converting Regular Expressions to Non-Deterministic Finite Automata through an Automated Framework." 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2024.

[12] C. R. A. N. Sharma, B. M. Reddy, D. Swetchana and N. Panda, ”Compiler Design for recognizing different Programming Languages,” 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10724115.

[13] A. S. Reddy, D. N. Achar, M. S. Mol and N. Panda, ”Audio Encryption Using AES and Cellular Automata,” 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10724851.

[14] Pranave, K.C., Challa, V.K.R. and Panda, N., 2024. System Search Service Implementation Based on a Custom Lexical Search. Procedia Computer Science, 235, pp.1548-1557.