Lawrence Technological University

MCS5243 – Theory of Computation

Project Title: - Formal Language Recognition Game

Group 7

Satish Kumar Pyata

000794769

Sai Nikhil Reddy Nadikattu

000793118

Formal Language Recognition Game

Introduction:

The Formal Language Recognition Game is a Python-based educational application aimed at providing users with an interactive learning experience in the field of formal languages and automata theory. By gamifying the process of recognizing formal languages, the project aims to make abstract concepts more accessible and engaging to learners.

Project Goals:

- **Educational:** To teach users about formal languages, their properties, and methods of recognition.
- **Interactive:** To engage users in active learning through gameplay and immediate feedback.
- **Scalable:** To offer multiple difficulty levels to accommodate users with varying levels of knowledge and expertise.
- **Accessible:** To provide a user-friendly interface that makes the learning process intuitive and enjoyable.

Technologies Used:

- **Python 3:** The primary programming language for implementing the game logic and user interface.
- **Tkinter:** Used to create the graphical user interface (GUI) for the game.
- **Random Module:** Utilized for generating random elements such as formal languages and strings.
- **Messagebox Module:** Integrated for displaying messages and feedback within the GUI environment.

Key Features:

- **Dynamic Language Generation:** Generates formal languages of varying complexity based on user-selected difficulty levels.
- **Interactive User Interface:** Provides an intuitive GUI with informative labels, responsive buttons, and clear feedback messages.

- **Questioning Mechanism:** Presents users with questions regarding the membership of specific strings in the generated formal language.
- **Immediate Feedback:** Offers instant feedback on users' responses, guiding them towards correct answers and reinforcing learning.
- **Score Tracking:** Tracks users' performance throughout the game, encouraging progress and improvement.
- **Retry Option:** Allows users to replay the game with a new set of formal languages after completion, promoting continued engagement and learning.

Game Flow:

- **Difficulty Selection:** Users choose a difficulty level (Easy, Medium, or Hard) to determine the complexity of the generated formal language.
- **Question Presentation:** The game presents users with a series of questions, asking whether specific strings belong to the generated formal language.
- **User Interaction:** Users respond to each question by selecting "Yes" or "No" options using interactive buttons.
- **Feedback and Scoring:** Immediate feedback is provided for each response, updating users' scores based on correctness.
- **Game Completion:** After answering a predetermined number of questions (e.g., five), users receive a final score and have the option to retry the game with a new set of formal languages.

Usage:

- Launch the Python script to start the game.
- Choose the desired difficulty level (Easy, Medium, or Hard) to begin.
- Respond to each question presented in the game interface by selecting the appropriate answer ("Yes" or "No").
- Receive instant feedback on the correctness of each response and track your score as you progress.
- Upon completing the game, view your final score and optionally retry with a new set of formal languages.

Source Code:

Python

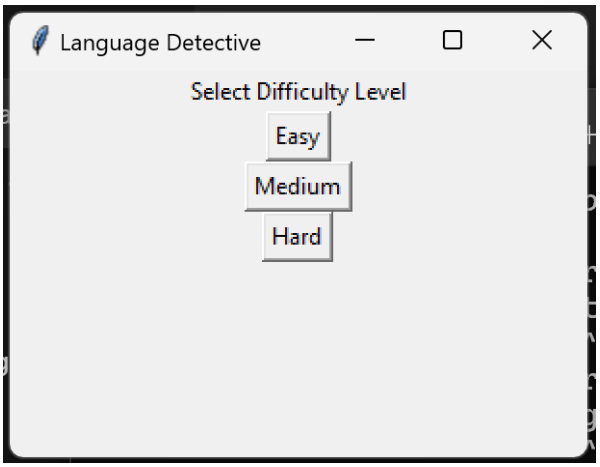
```
import random
import tkinter as tk
from tkinter import messagebox

# Function to generate a formal language based on difficulty level
def generate_formal_language(difficulty):
    if difficulty == "Easy":
        num_strings = random.randint(3, 5)
        alphabet = ['0', '1']
    elif difficulty == "Medium":
        num_strings = random.randint(4, 6)
        alphabet = ['0', '1', '2']
    elif difficulty == "Hard":
        num_strings = random.randint(5, 7)
        alphabet = ['0', '1', '2', '3']
    else:
        raise ValueError("Invalid difficulty level")

    language = [''.join(random.choices(alphabet, k=random.randint(1, 5))) for _
in range(num_strings)]
    return language

def play_game(difficulty):
    language = generate_formal_language(difficulty)
    num_strings = len(language)
    num_questions = 5  # Number of questions to ask
    questions_asked = 0

    def check_string(result):
        nonlocal score
        nonlocal questions_asked
        nonlocal current_string

        if ((current_string in language) and result == "yes") or \
           ((current_string not in language) and result == "no"):
            output_label.config(text="Correct!")
            score += 1
        else:
            output_label.config(text="Incorrect.")

        questions_asked += 1
```

```
        if questions_asked == num_questions:
            yes_button.destroy()
            no_button.destroy()
            try_again_button.pack()
            messagebox.showinfo("Game Over", f"Your score:
{score}/{num_questions}")
        else:
            current_string = generate_question_string(language)
            string_var.set(f"String: {current_string}")

    def try_again():
        window.destroy()
        select_difficulty()

    def generate_question_string(language):
        # Randomly select whether the string should belong to the language or not
        if random.random() < 0.5:
            string = random.choice(language)
        else:
            alphabet = ''.join(set(''.join(language)))
            string = ''.join(random.choices(alphabet, k=random.randint(1, 5)))
        return string

    window = tk.Tk()
    window.title("Language Detective")
    window.geometry("400x200")

    language_str = "Generated Language: " + ", ".join(language)
    language_label = tk.Label(window, text=language_str)
    language_label.pack()

    current_string = generate_question_string(language)
    string_var = tk.StringVar()
    string_var.set(f"String: {current_string}")
    string_label = tk.Label(window, textvariable=string_var)
    string_label.pack()

    question_label = tk.Label(window, text="Does the following string belong to
the language?")
    question_label.pack()

    button_frame = tk.Frame(window)
    button_frame.pack()
```

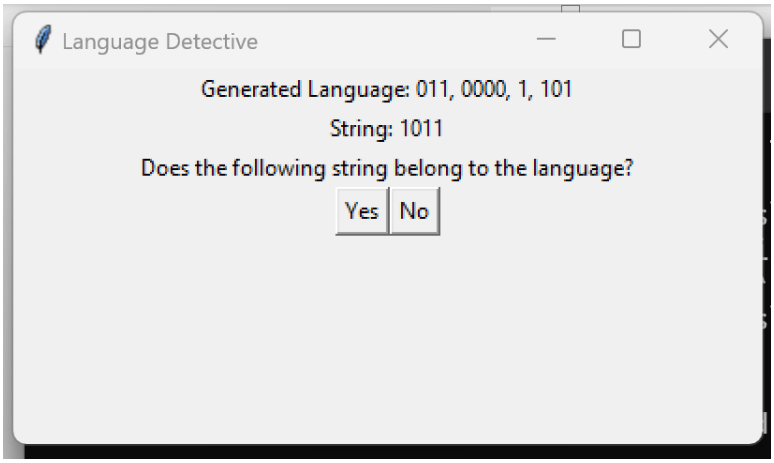
```
    yes_button = tk.Button(button_frame, text="Yes", command=lambda:
check_string("yes"))
    yes_button.pack(side=tk.LEFT)

    no_button = tk.Button(button_frame, text="No", command=lambda:
check_string("no"))
    no_button.pack(side=tk.LEFT)

    output_label = tk.Label(window, text="")
    output_label.pack()

    score = 0

    try_again_button = tk.Button(window, text="Try Again", command=try_again)

    window.mainloop()

def select_difficulty():
    root = tk.Tk()
    root.title("Language Detective")
    root.geometry("300x200")

    def start_game(difficulty):
        root.destroy()
        play_game(difficulty)

    label = tk.Label(root, text="Select Difficulty Level")
    label.pack()

    easy_button = tk.Button(root, text="Easy", command=lambda:
start_game("Easy"))
    easy_button.pack()

    medium_button = tk.Button(root, text="Medium", command=lambda:
start_game("Medium"))
    medium_button.pack()

    hard_button = tk.Button(root, text="Hard", command=lambda:
start_game("Hard"))
    hard_button.pack()

    root.mainloop()

if __name__ == "__main__":
    select_difficulty()
```

Outputs:

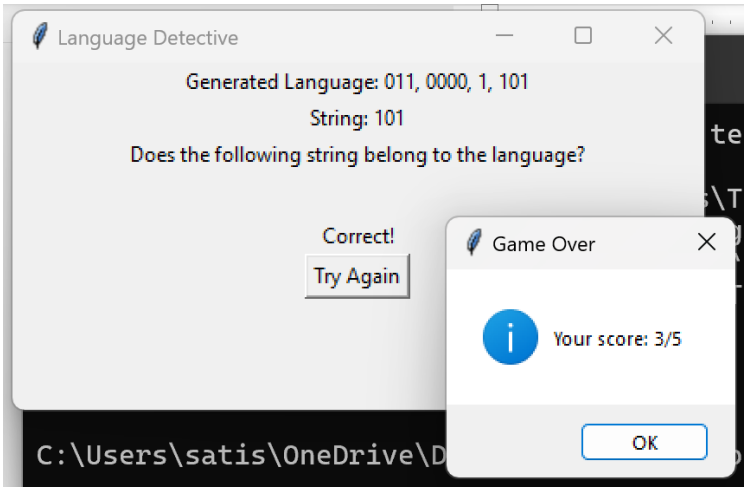
- **Output 1: Difficulty Selection**



- **Output 2: Game Interface**



- **Output 3: Feedback and Score**



Conclusion:

The Formal Language Recognition Game provides an immersive and educational experience for users interested in learning about formal languages and their recognition. Through its intuitive interface, dynamic content generation, and immediate feedback mechanisms, the game effectively engages users and fosters learning in an enjoyable manner.

Future Enhancements:

- **Advanced Difficulty Levels:** Integrate more challenging difficulty levels, incorporating advanced language types such as context-sensitive languages.
- **Tutorial Mode:** Implement interactive tutorials and explanations to enhance user understanding of formal language concepts.
- **Multiplayer Functionality:** Add multiplayer features, allowing users to compete or collaborate in solving language recognition challenges.
- **Localization and Internationalization:** Extend language support and localization efforts to make the game accessible to a broader audience.

References:

- Tkinter documentation: https://docs.python.org/3/library/tkinter.html
- Python random module documentation:

  https://docs.python.org/3/library/random.html
- Formal languages and automata theory textbooks and online resources.