### Report on Python Game Collection

#### Submitted by: Harshit Chauhan(23csu363)&Saksham Parmar(23csu367)

### ****1. Introduction****

This report presents the design, features, and functionality of the **Python Game Collection**. The project, developed using Python's Tkinter module, provides a selection of interactive mini-games. It is designed to demonstrate the application of GUI programming concepts while offering an engaging experience for users.

The program includes six games:

1. Tic-Tac-Toe
2. Word Scramble
3. Math Quiz
4. Odd or Even
5. Ludo Dice Roll
6. Trivia Quiz

Each game is implemented with a unique window and tailored user interface, allowing users to interact intuitively.

### ****2. Objectives****

The primary objectives of this project are:

* To create a collection of mini-games with a graphical user interface (GUI).
* To provide a user-friendly design with clear navigation between games.
* To implement modular, reusable, and expandable code.
* To validate user inputs effectively and ensure smooth functionality.

### ****3. Features and Functionality****

#### **3.1 Main Menu**

* **Description:**
  + Displays a list of available games as buttons.
  + Each button navigates to the respective game window.
* **Interface:**
  + Designed with a consistent theme and visually appealing layout.

python

Copy code

root = tk.Tk()

root.title("Python Game Collection")

root.configure(bg="#dcedc8")

for game in ["Tic-Tac-Toe", "Word Scramble", "Math Quiz", "Odd or Even", "Trivia Quiz", "Ludo Dice Roll"]:

tk.Button(root, text=game, command=lambda game=game: start\_game(game), font=("Arial", 14), bg="#4caf50", fg="white", width=25).pack(pady=10)

root.mainloop()

#### **3.2 Games Included**

1. **Tic-Tac-Toe:**
   * A two-player game with a 3x3 grid.
   * Alternates turns between 'X' and 'O'.
   * Detects win conditions and ties.

python

Copy code

if game\_name == "Tic-Tac-Toe":

board = [' '] \* 9

current\_player = 'X'

def button\_click(i):

nonlocal current\_player

if board[i] == ' ':

board[i] = current\_player

buttons[i].config(text=current\_player)

if check\_win():

result\_label.config(text=f"Player {current\_player} wins!")

elif ' ' not in board:

result\_label.config(text="It's a tie!")

current\_player = 'O' if current\_player == 'X' else 'X'

def check\_win():

win\_conditions = [(0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6)]

return any(board[i] == board[j] == board[k] != ' ' for i, j, k in win\_conditions)

game\_window = tk.Toplevel(root, bg="#f1f8e9")

buttons = [tk.Button(game\_window, text=' ', width=10, height=3, command=lambda i=i: button\_click(i), font=("Arial", 12), bg="#b2dfdb") for i in range(9)]

for i, button in enumerate(buttons):

button.grid(row=i // 3, column=i % 3)

result\_label = tk.Label(game\_window, font=("Arial", 14), bg="#f1f8e9", fg="#333")

result\_label.grid(row=3, column=0, columnspan=3)

1. **Word Scramble:**
   * Presents a scrambled word for the player to unscramble.
   * Provides feedback for correct and incorrect answers.

python

Copy code

elif game\_name == "Word Scramble":

words = ["python", "programming", "developer", "algorithm"]

word = random.choice(words)

scrambled\_word = "".join(random.sample(word, len(word)))

def check\_answer():

if entry.get().lower() == word:

result\_label.config(text="Correct!")

else:

result\_label.config(text=f"Wrong! The word was {word}.")

game\_window = tk.Toplevel(root, bg="#ffe0b2")

tk.Label(game\_window, text=f"Unscramble the word: {scrambled\_word}", font=("Arial", 14), bg="#ffe0b2").pack(pady=10)

entry = tk.Entry(game\_window, font=("Arial", 12))

entry.pack(pady=10)

tk.Button(game\_window, text="Submit", font=("Arial", 12), command=check\_answer, bg="#fb8c00", fg="white").pack(pady=10)

result\_label = tk.Label(game\_window, font=("Arial", 14), bg="#ffe0b2")

result\_label.pack(pady=10)

### ****4. Design Approach****

The program is structured around a single main function, start\_game(game\_name), which handles the logic for all games. This modular approach allows for:

* Easy addition of new games.
* Reuse of common design elements.
* Simplified debugging and maintenance.

python

Copy code

def start\_game(game\_name):

if game\_name == "Tic-Tac-Toe":

# Logic for Tic-Tac-Toe

elif game\_name == "Word Scramble":

# Logic for Word Scramble

elif game\_name == "Math Quiz":

# Logic for Math Quiz

# Additional games implemented similarly

### ****5. Future Enhancements****

1. **New Games:**
   * Adding the Snake game placeholder.
   * Introducing multiplayer functionality.
2. **Leaderboards:**
   * Track scores and display rankings for competitive play.
3. **Settings:**
   * Allow players to adjust difficulty levels or customize gameplay options.
4. **Mobile Compatibility:**
   * Adapt the application for mobile devices using frameworks like Kivy.

### ****6. Conclusion****

The **Python Game Collection** showcases the versatility of Tkinter in creating a variety of engaging mini-games. Its modular structure, interactive design, and input validation make it a valuable educational tool. Future expansions and enhancements can transform this project into a more comprehensive gaming platform.

**End of Report**