ರೈ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ

Rai Technology University

(Estd. under Karnataka Act. No. 40 of 2013)

PROJECT REPORT

MEMORY GAME PROJECT

SUBMITTED BY:

SWATHI.R

RTU24101IT011

Program Name: B.TECH IT AIML

Semester: 3rd Sem

Session: 2025-2026

Submitted To:

Faculty Name: Mr. SATYAM

Department of Computer Science & Engineering College of Engineering

Rai Technology University

ರೈ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ

Rai Technology University

(Estd. under Karnataka Act. No. 40 of 2013)

College of Engineering & Application

CERTIFICATE

This is to certify that SWATHI.R , RTU24101IT011 a Students of Rai Technology University, Bangalore, has successfully completed the Project work on "MEMORY GAME" in partial fulfilment of the requirements for the 3rd semester B. Tech program at the College of Engineering & Application, Rai Technology University ,Bangalore during the academic year 25-26

Signature of the Subject Faculty **Signature of the HOD**

ABSTRACT

The Memory Game project is a Python-based desktop application developed using Tkinter. It is designed as an interactive puzzle game to improve concentration and memory skills. Players flip cards to reveal emojis and try to match pairs. The system records the number of turns taken and displays a congratulatory message when all pairs are matched. The project demonstrates the use of GUI design, event handling, and basic game logic implementation in Python.

This the Information of my Mini Project..

TABLE OF CONTENTS

INTRODUCTION

In today’s digital era, memory-enhancing games are widely used as both entertainment and educational tools. The Memory Game is designed to improve short-term memory and focus while providing fun gameplay. This project uses Python's Tkinter library to create a simple yet visually engaging application. The game involves flipping cards, matching pairs of emojis, and keeping track of turns taken. It is lightweight, user-friendly, and suitable for all age groups.

OBJECTIVE

The main objectives of the Memory Game project are:

1. To design and develop an interactive memory game using Python.

2. To demonstrate the use of Tkinter for GUI-based applications.

3. To implement basic game logic such as card shuffling, matching pairs, and turn counting.

4. To provide an engaging and user-friendly experience.

5. To create a scalable foundation for future enhancements such as difficulty levels

FULL CODE

```
import tkinter as tk

import random

from functools import partial

from tkinter import messagebox


# Emoji pairs

EMOJIS = ["🍓", "🍉", "🍇", "🍋", "🍍", "🥝"]


# Define colors for each emoji

EMOJI_COLORS = {

    "🍓": "red",

    "🍉": "green",

    "🍇": "purple",

    "🍋": "gold",

    "🍍": "orange",

    "🥝": "brown"

}


class MemoryGame:

    def __init__(self, root):

        self.root = root

        self.root.title("🥝 Memory Game")
```

```
        self.root.resizable(False, False)


        self.turns = 0
        self.buttons = []
        self.cards = []
        self.choice_one = None
        self.choice_two = None
        self.disabled = False


        self.setup_ui()
        self.shuffle_cards()


    def setup_ui(self):
        # Smaller canvas for small screen
        self.canvas = tk.Canvas(self.root, width=400, height=500,
highlightthickness=0)
        self.canvas.pack(fill="both", expand=True)


        # Draw vertical gradient
        for i in range(500):
            r = 11
            g = 12
            b = 42 + i // 10
            color = f"#{r:02x}{g:02x}{b:02x}"
```

```
            self.canvas.create_line(0, i, 400, i, fill=color)


        # Title
        self.title_label = tk.Label(self.canvas, text="✨ Memory
Game ✨",
                                    font=("Segoe UI Emoji", 20, "bold"),
                                    bg=None, fg="#FFD700")
        self.title_label.place(relx=0.5, y=20, anchor="center")


        # New Game button
        self.new_game_btn = tk.Button(self.canvas, text="🔄
New Game",
                                      command=self.shuffle_cards,
                                      bg="#1C1F4A", fg="#FFD700",
                                      font=("Arial", 10, "bold"),
                                      relief="raised",
activebackground="#2C2F6C")
        self.new_game_btn.place(relx=0.5, y=60,
anchor="center")


        # Grid frame
        self.grid_frame = tk.Frame(self.canvas, bg=None)
        self.grid_frame.place(relx=0.5, rely=0.55,
anchor="center")
```

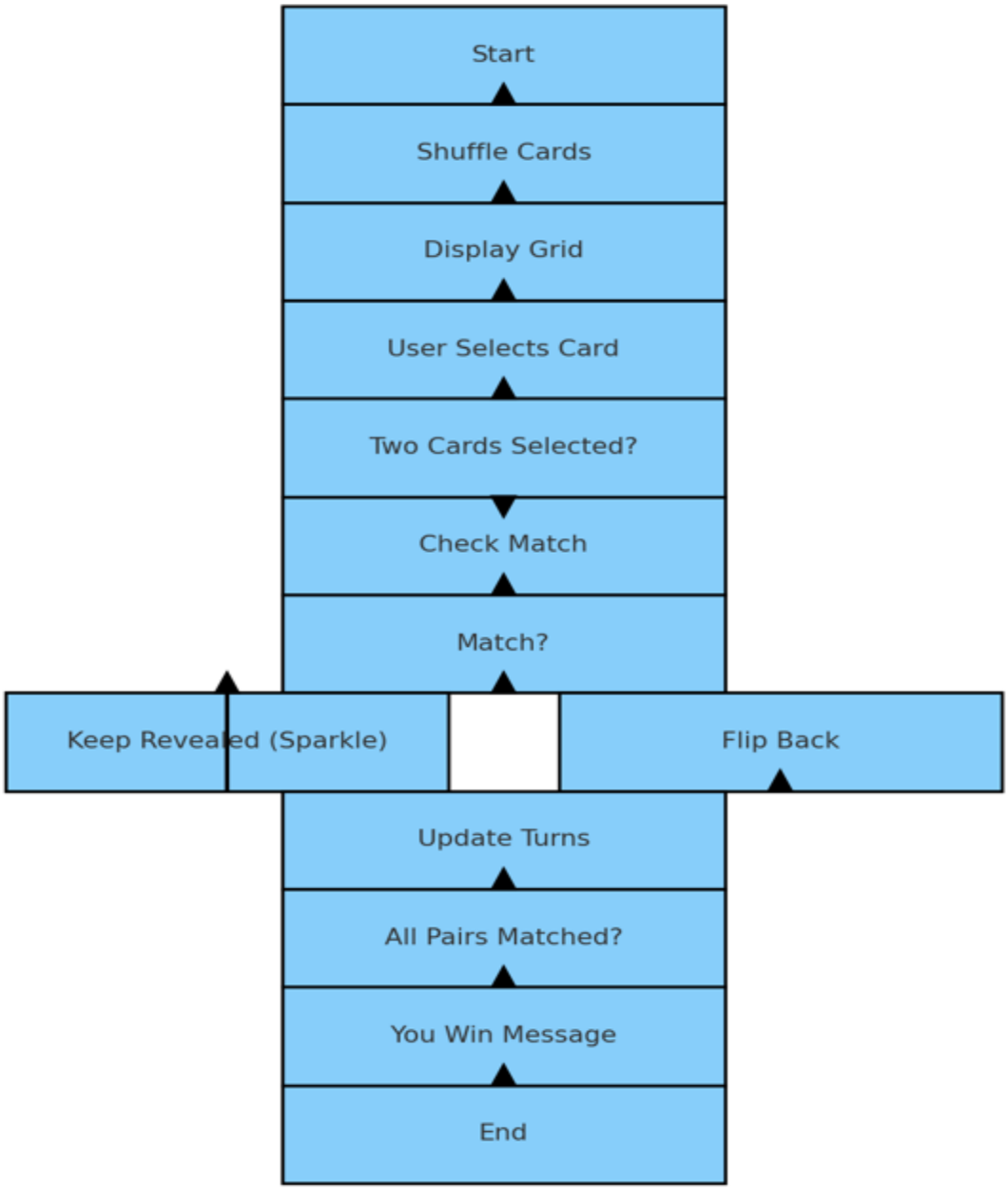
```
        # Turns label
        self.turns_label = tk.Label(self.canvas, text="Turns: 0",
                                    font=("Arial", 12, "bold"),
                                    bg=None, fg="#87CEEB")
        self.turns_label.place(relx=0.5, y=480, anchor="center")

    def shuffle_cards(self):
        self.cards = [{"emoji": e, "matched": False} for e in
EMOJIS * 2]
        random.shuffle(self.cards)
        self.turns = 0
        self.choice_one = None
        self.choice_two = None
        self.disabled = False
        self.update_turns()
        self.render_grid()

    def render_grid(self):
        for widget in self.grid_frame.winfo_children():
            widget.destroy()
        self.buttons = []

        for index, card in enumerate(self.cards):
```

```
            btn = tk.Button(self.grid_frame, text="❓", width=6,
height=3,
                            command=partial(self.handle_choice, index),
                            font=("Segoe UI Emoji", 18, "bold"),
relief="raised",
                            bg="#1C1F4A", fg="white",
activebackground="#2C2F6C")
            row = index // 4
            col = index % 4
            btn.grid(row=row, column=col, padx=3, pady=3)
            self.buttons.append(btn)

    def handle_choice(self, index):
        if self.disabled:
            return

        card = self.cards[index]
        btn = self.buttons[index]

        if card["matched"] or btn["text"] != "❓":
            return

        btn.config(text=card["emoji"],
fg=EMOJI_COLORS.get(card["emoji"], "white"))
```

```
        if self.choice_one is None:
            self.choice_one = index
        elif self.choice_two is None:
            self.choice_two = index
            self.disabled = True
            self.root.after(800, self.check_match)

    def sparkle_effect(self, btn, emoji, color):
        btn.config(text=f"✨{emoji}✨", fg="gold")
        self.root.after(600, lambda: btn.config(text=emoji,
fg=color))

    def check_match(self):
        c1 = self.cards[self.choice_one]
        c2 = self.cards[self.choice_two]

        if c1["emoji"] == c2["emoji"]:
            c1["matched"] = True
            c2["matched"] = True
            self.sparkle_effect(self.buttons[self.choice_one],
c1["emoji"], EMOJI_COLORS[c1["emoji"]])
            self.sparkle_effect(self.buttons[self.choice_two],
c2["emoji"], EMOJI_COLORS[c2["emoji"]])
        else:
```

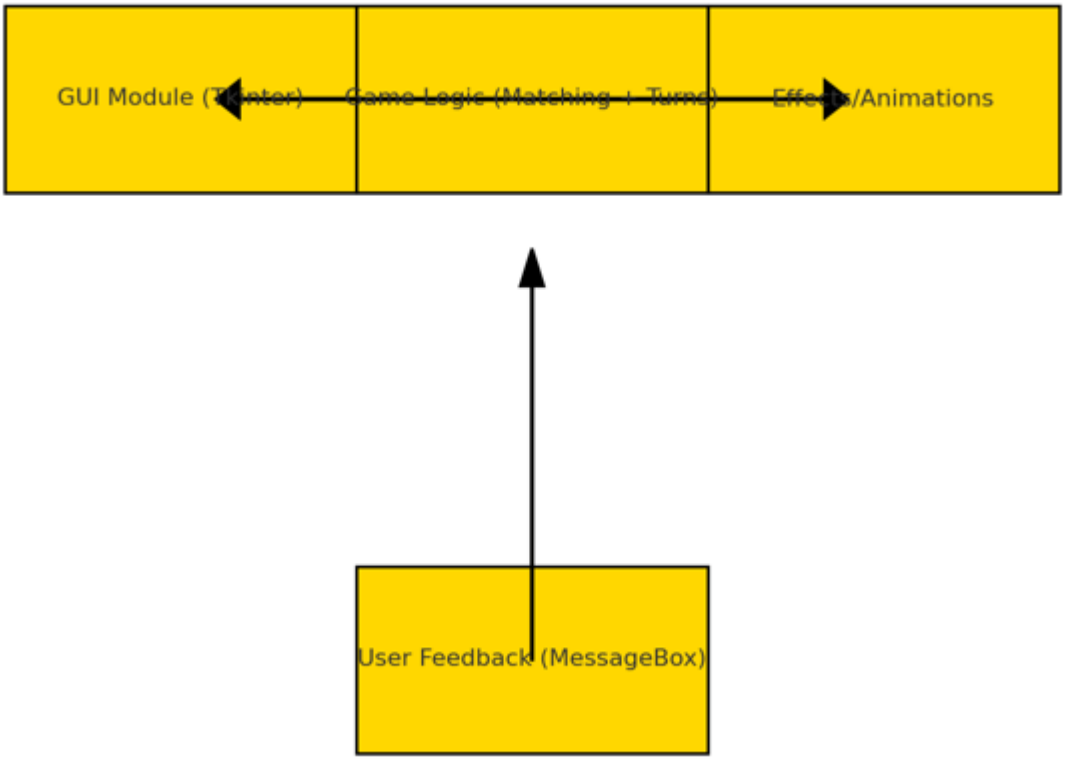
```
            self.buttons[self.choice_one].config(text="❓",
fg="white")
            self.buttons[self.choice_two].config(text="❓",
fg="white")

        self.choice_one = None
        self.choice_two = None
        self.disabled = False
        self.turns += 1
        self.update_turns()

        if all(card["matched"] for card in self.cards):
            messagebox.showinfo("🎯 You Win! 🌟",
                                f"🎉 Congratulations! 🎉\nYou matched all
pairs in {self.turns} turns!\n✨ Excellent memory under the
stars! ✨")

    def update_turns(self):
        self.turns_label.config(text=f"Turns: {self.turns}")

if __name__ == "__main__":
    root = tk.Tk()
    game = MemoryGame(root)
    root.mainloop()
```

FLOW OF CODE



THIS SHOWS HOW THE CODE RUNS AND USE OF CODE

BLOCK DIAGRAM



DESIGN AND IMPLEMENTATION

The Memory Game is designed with simplicity and interactivity in mind. The GUI is built using Tkinter, with buttons representing the hidden cards. A grid layout organizes the cards, while labels and buttons provide game controls.

Design Components:
- Gradient background with title and new game button
- Card grid using Tkinter buttons
- Turns counter
- Pop-up message for winning

Implementation Modules:
1. GUI Module (Tkinter): Handles layout, buttons, labels, and user interaction.
2. Game Logic: Includes card shuffling, matching logic, and turn counting.
3. Effects: Sparkle effect when a match is found.

4. User Feedback: Message box for win notification.

Results and Discussion Results

Results:

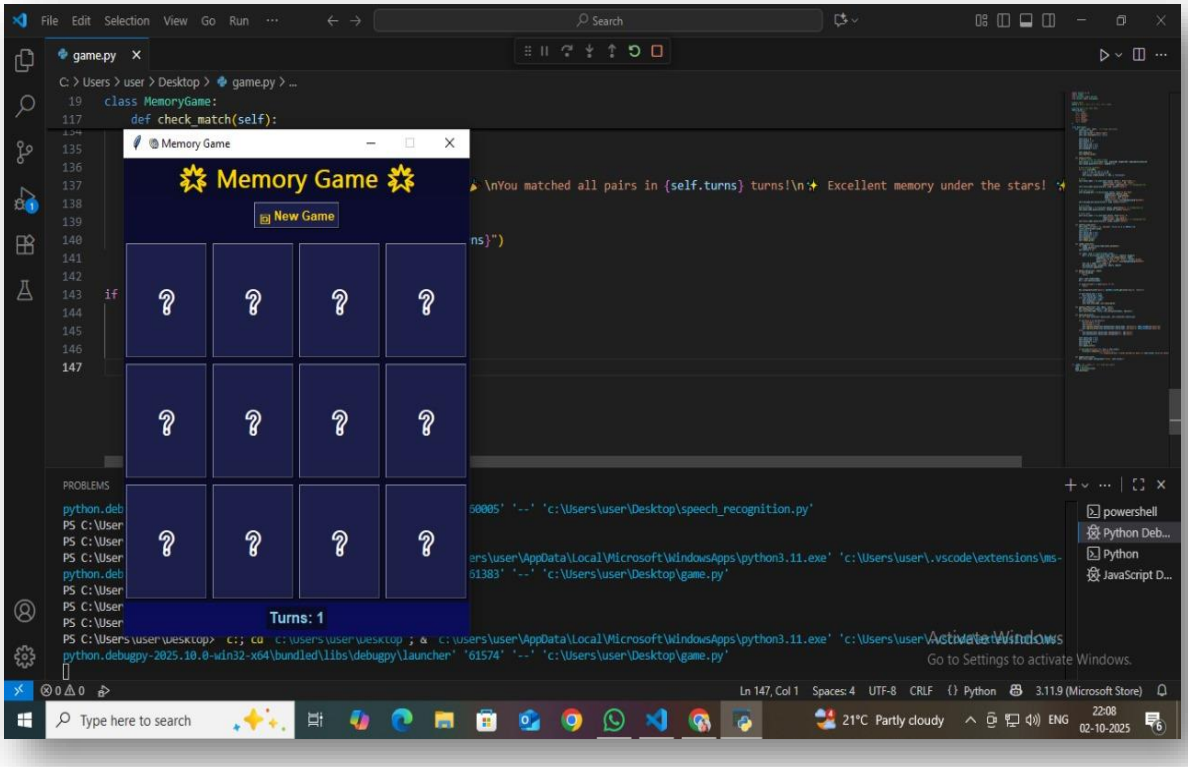
- The game successfully runs with a graphical interface.
- Cards flip to reveal emojis when clicked.
- Matching pairs stay revealed with a sparkle effect.
- The system counts turns accurately.
- A winning message is displayed when all pairs are matched.

Discussion:

- The project demonstrates the use of Python and Tkinter for game development.
- It is suitable for beginners and can be enhanced with difficulty levels and themes.
- The lightweight design ensures smooth performance.

- Future improvements could include score tracking, leaderboard, and sound effects.

SCREENSHOTS OF OUTPUT



Conclusion

The Memory Game developed using Python and Tkinter successfully demonstrates the creation of a fun and interactive desktop game. It helps users improve memory skills while enjoying a simple game. The project meets its objectives and provides a foundation for further enhancements such as scoring systems, advanced graphics, and multiple difficulty levels. It also showcases how Python can be used to build educational and entertaining applications.

THANK YOU