

Annexure-6

Copyright Submission Form (Computer Software)

Title of the Software/Application: *MEOMORY PUZZLE GAME*

Name of authors: Dr Bhaveshkumar Choithram Dharmani

Name of Student/Scholar: Clement Bobby

Registration Number: 12412797

Program Name: B. Tech. in ECE

Name/ UID of Supervisor: Dr Bhaveshkumar Choithram Dharmani (23409)

School: School of Electronoics and Electrical Engineering

Name and UID of the all co-authors/Co-Supervisor:

Affiliation details (with Pin code) of co-authors/Co- Supervisor:

Language of the Work: Python

Summary, Uniqueness and Utility of work:

This Memory Puzzle game is designed to help improve cognitive skills, specifically focusing on memory retention and attention. In the game, players are challenged to match pairs of cards by flipping them over one at a time. The goal is to find all matching pairs in the shortest time possible. Using Python and the Pygame library, the game offers an engaging and interactive experience. It also features scoring, time tracking, and the option to reset the puzzle, giving players the opportunity to improve with each attempt.

The game is not only fun, but it also serves as a useful educational tool for students looking to enhance their memory and concentration skills. Its simple yet visually stimulating interface encourages users to play repeatedly, reinforcing memory improvement. This project also demonstrates how Python can be used for creating games, offering valuable insights for anyone interested in game development or interactive learning experiences.

Flow Diagram:

1. Start Game
 - Initialize game (shuffle cards, set timer, score, etc.)

2. Display Cards
 - Show the cards face-down.
3. Player Interaction
 - Player selects two cards to flip.
 - Check if the two cards match.
 - If they match: Keep the cards face-up, update score.
 - If they don't match: Flip the cards back face-down.
4. Game Progression
 - Check if all pairs are matched.
 - If all pairs are matched: Show success message and end game.
 - If not: Continue until all pairs are matched.
5. End Game
 - Display the player's score and time taken.
 - Offer option to restart the game.

The flow diagram would help anyone understand the sequence of steps in the game and how they are connected. It's a great way to visually plan and communicate how the game works.

Would you like help in creating a flow diagram for your project?

4o mini

You said:

make a flow diagram for memory puzzle game

ChatGPT said:

ChatGPT

It seems that the flow diagram couldn't be generated at the moment. However, I can describe how to create one for your Memory Puzzle game:

1. Start Game
 - Action: Initialize the game (shuffle cards, set timer, set score, etc.)
 - Next step: Display Cards
2. Display Cards
 - Action: Show all cards face-down.
 - Next step: Player Interaction
3. Player Interaction
 - Action: Player selects two cards to flip.
 - Next step: Check if Cards Match
4. Check if Cards Match

- Decision:
 - If the cards match: Keep them face-up and update the score.
 - If they don't match: Flip them back face-down.
 - Next step: Game Progression
5. Game Progression
- Action: Check if all pairs are matched.
 - Decision:
 - If all pairs are matched: Show success message and end the game.
 - If not: Continue playing.
 - Next step: End Game or continue to Player Interaction
6. End Game
- Action: Show score and time taken.
 - Option: Offer the player the option to restart the game

Source Code/Object Code:

```
import pygame
import random
import time
import os

pygame.init()

# Screen dimensions
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Memory Puzzle Game (CLM)")

# Colors
WHITE = (255, 255, 255)
```

```

BLACK = (0, 0, 0)
GRAY = (200, 200, 200)
BLUE = (50, 150, 255)
RED = (255, 50, 50)
GREEN = (50, 255, 50)
TURN_COLORS = {1: (200, 100, 100), 2: (100, 200, 100)}

# Grid settings (modifiable for levels)
ROWS, COLS = 5, 5
CARD_WIDTH, CARD_HEIGHT = WIDTH // COLS, HEIGHT // ROWS

# Fonts
font = pygame.font.SysFont("comicsansms", 30)
big_font = pygame.font.SysFont("comicsansms", 40)

# Use solid colors for cards
cards = [pygame.Surface((CARD_WIDTH, CARD_HEIGHT)) for _ in range((ROWS * COLS) // 2)]
for i, surface in enumerate(cards):
    surface.fill((random.randint(50, 255), random.randint(50, 255), random.randint(50, 255))) #
Random colors
cards = cards * 2
random.shuffle(cards)
cards = [cards[i:i + COLS] for i in range(0, len(cards), COLS)]

# Game variables
time_limit = 30 # 30 seconds per turn
hints_left = {1: 3, 2: 3} # 3 hints per player
start_time = time.time()
revealed = [[False] * COLS for _ in range(ROWS)]
player1_score = 0
player2_score = 0

```

```
turn = 1
```

```
first_pick = None
```

```
matched_pairs = 0
```

```
# Leaderboard file
```

```
LEADERBOARD_FILE = "leaderboard.txt"
```

```
def save_score(player1_score, player2_score):
```

```
    """Save scores to the leaderboard."""
```

```
    with open(LEADERBOARD_FILE, "a") as file:
```

```
        file.write(f"Player 1: {player1_score}, Player 2: {player2_score}\n")
```

```
def display_leaderboard():
```

```
    """Display the last 5 scores."""
```

```
    if not os.path.exists(LEADERBOARD_FILE):
```

```
        return
```

```
    with open(LEADERBOARD_FILE, "r") as file:
```

```
        scores = file.readlines()
```

```
    y_offset = 50
```

```
    for score in scores[-5:]:
```

```
        text = font.render(score.strip(), True, BLACK)
```

```
        screen.blit(text, (WIDTH // 4, y_offset))
```

```
        y_offset += 30
```

```
def draw_grid(revealed):
```

```
    """Draws the game grid and reveals cards based on the revealed state."""
```

```
    screen.fill(WHITE)
```

```
    for row in range(ROWS):
```

```

for col in range(COLS):
    x, y = col * CARD_WIDTH, row * CARD_HEIGHT
    if revealed[row][col]:
        screen.blit(cards[row][col], (x, y)) # Display color block
    else:
        pygame.draw.rect(screen, GRAY, (x, y, CARD_WIDTH, CARD_HEIGHT))
        pygame.draw.rect(screen, BLACK, (x, y, CARD_WIDTH, CARD_HEIGHT), 2)

```

```

def display_scores(player1_score, player2_score, turn):
    """Displays the scores and the current player's turn."""
    pygame.draw.rect(screen, TURN_COLORS[turn], (0, HEIGHT - 100, WIDTH, 100)) # Highlight turn
    text1 = font.render(f"Player 1: {player1_score}", True, RED)
    text2 = font.render(f"Player 2: {player2_score}", True, GREEN)
    turn_text = big_font.render(f"Player {turn}'s Turn", True, BLACK)
    screen.blit(text1, (10, HEIGHT - 60))
    screen.blit(text2, (WIDTH - 200, HEIGHT - 60))
    screen.blit(turn_text, (WIDTH // 3, HEIGHT - 90))

```

```

def display_timer():
    """Displays the countdown timer for the current player's turn."""
    elapsed = int(time.time() - start_time)
    remaining = max(0, time_limit - elapsed)
    timer_text = font.render(f"Time: {remaining}s", True, BLACK)
    screen.blit(timer_text, (WIDTH // 2 - 50, HEIGHT - 60))
    return remaining

```

```

def use_hint(player):
    """Reveals all cards briefly."""

```

```

if hints_left[player] > 0:
    hints_left[player] -= 1
    for row in range(ROWS):
        for col in range(COLS):
            revealed[row][col] = True
    draw_grid(revealed)
    pygame.display.update()
    time.sleep(1)
    for row in range(ROWS):
        for col in range(COLS):
            revealed[row][col] = False

```

```

def game_over_screen(player1_score, player2_score):

```

```

    """Displays the winner at the end of the game."""

```

```

    screen.fill(WHITE)

```

```

    if player1_score > player2_score:

```

```

        winner_text = "Player 1 Wins! 🎉"

```

```

    elif player2_score > player1_score:

```

```

        winner_text = "Player 2 Wins! 🎉"

```

```

    else:

```

```

        winner_text = "It's a Tie! 🤝"

```

```

    text = big_font.render(winner_text, True, BLACK)

```

```

    screen.blit(text, (WIDTH // 6, HEIGHT // 3))

```

```

    display_leaderboard()

```

```

    pygame.display.update()

```

```

    time.sleep(3)

```

```

def memory_puzzle_game():

```

```

    """Main game loop for the two-player memory puzzle."""

```

```
global revealed, first_pick, matched_pairs, player1_score, player2_score, turn, start_time
```

```
running = True
```

```
while running:
```

```
    draw_grid(revealed)
```

```
    display_scores(player1_score, player2_score, turn)
```

```
    remaining_time = display_timer()
```

```
    pygame.display.update()
```

```
    if remaining_time == 0:
```

```
        turn = 2 if turn == 1 else 1
```

```
        start_time = time.time()
```

```
    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
```

```
            running = False
```

```
        if event.type == pygame.KEYDOWN:
```

```
            if event.key == pygame.K_h: # Hint
```

```
                use_hint(turn)
```

```
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
```

```
            row, col = event.pos[1] // CARD_HEIGHT, event.pos[0] // CARD_WIDTH
```

```
            if not revealed[row][col]:
```

```
                revealed[row][col] = True
```

```
                if first_pick is None:
```

```
                    first_pick = (row, col)
```

```
                else:
```

```
                    r1, c1 = first_pick
```

```
                    if cards[row][col] == cards[r1][c1]:
```

```
                        matched_pairs += 1
```



```
else:
    time.sleep(0.5)
    revealed[row][col] = False
    revealed[r1][c1] = False
    turn = 2 if turn == 1 else 1
    first_pick = None
```

```
if matched_pairs == (ROWS * COLS) // 2:
    save_score(player1_score, player2_score)
    game_over_screen(player1_score, player2_score)
    running = False
```

```
pygame.quit()
```

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