**Lab1-System Analysis and Design**

**-Vaishnavi Kothamirkar**

**Game Process Description**

The multi-player memory game involves players taking turns to find matching pairs of symbols on a game board. The game progresses through several steps, from initialization to completion, with the objective of finding all matching pairs. The player with the most pairs at the end of the game is declared the winner.

**Step 1: Initialization**

Process: The game begins with a randomized distribution of symbol pairs across a grid on the game board. Each symbol is hidden behind a tile.

Data Elements: Symbol pairs, game board grid.

**Step 2: Player Turn**

Process: Players take turns to reveal two tiles at a time. Each turn involves the following sub-steps:

Player Selection: A player chooses two tiles to reveal.

Reveal Tiles: The chosen tiles are revealed to show their symbols.

Check for Match: The game checks if the revealed symbols match.

Data Elements: Player's choice, revealed symbols.

**Step 3: Match/No Match**

Match:

Process: If the symbols match, the tiles remain open, and the player scores a point.

Output: Updated board with open tiles, updated player score.

No Match:

Process: If the symbols do not match, a short delay occurs allowing players to memorize the symbols, and then the tiles are covered again.

Output: Board returns to previous state with tiles covered.

**Step 4: Switch Turn**

Process: After a player's turn, whether a match was made or not, the turn switches to the next player.

Data Elements: Current player identifier.

**Step 5: Check Game Status**

Process: After each turn, the game checks if all pairs have been matched.

Data Elements: Number of matched pairs, total pairs.

**Step 6: Game End**

Process: The game ends when all pairs are matched.

Output: Final game board, final scores of each player.

**Step 7: Declare Winner**

Process: The player with the most pairs matched is declared the winner.

Output: Winner announcement.

**Interactions in Detail**

**Player Input:**

Input: Players input their choice of two tiles.

Interaction: This input triggers the game logic to reveal the tiles.

**Game Logic Processing:**

Process: The game logic processes the player's choice, reveals tiles, checks for matches, updates scores, and determines if the game continues or ends.

Interaction: Each step in the process provides feedback to the player and sets up the next step or turn.

**Output to Player:**

Output: Visual representation of the game board, scores, messages indicating match/no match, and turn switch notifications.

Interaction: Players receive this output to make informed decisions for their next move.

**Turn Management:**

Process: Management of player turns and switching between players after each turn.

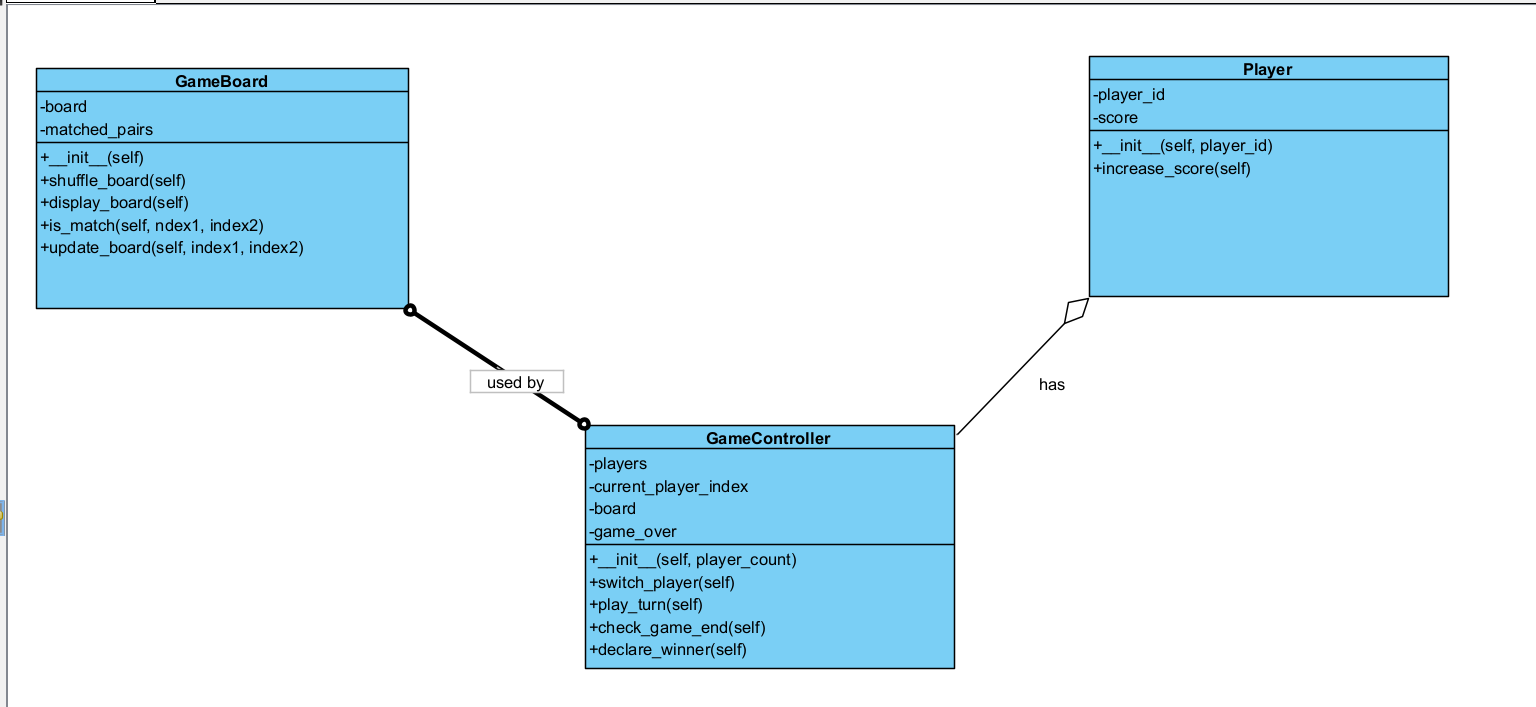
Interaction: Ensures fair play and equal opportunities for each player.

**End Game Conditions and Output:**

Process: Determining when all pairs are matched and declaring the winner based on scores.

Interaction: Provides closure to the game with a clear outcome.

**UML Class Diagram & Its Components:**



**1.Class: GameBoard**

**Attributes:**

board: List[List[str]] - Represents the game board with symbols.

matched\_pairs: List[Tuple[int, int]] - Keeps track of matched pairs' positions.

**Methods:**

\_\_init\_\_(self): Initializes the game board.

shuffle\_board(self): Randomizes the symbols on the board.

display\_board(self): Displays the current state of the board.

reveal\_tile(self, position): Reveals the symbol at the given position.

is\_match(self, pos1, pos2): Checks if two positions have matching symbols.

**2.Class: Player**

**Attributes:**

player\_id: int - Identifier for the player.

score: int - The player's current score.

**Methods:**

\_\_init\_\_(self, player\_id): Constructor for the player.

increase\_score(self): Increases the player's score.

**3.Class: GameController**

**Attributes:**

players: List[Player] - List of players in the game.

current\_player\_index: int - Index of the current player.

board: GameBoard - The game board instance.

game\_over: bool - Flag to check if the game is over.

**Methods:**

\_\_init\_\_(self, player\_count): Initializes the game with the specified number of players.

switch\_player(self): Switches turn to the next player.

play\_turn(self): Manages a single player's turn.

check\_game\_end(self): Checks if the game has ended.

declare\_winner(self): Declares the winner of the game.

**Code:**

**GameBoard.py**

import random

class GameBoard:

def \_\_init\_\_(self):

self.symbols = ['!', '@', '#', '$', '%']

self.board = ['x'] \* 10

self.shuffle\_board()

def shuffle\_board(self):

check = self.symbols \* 2

random.shuffle(check)

self.board = check

def display\_board(self):

print(' '.join(self.board))

def is\_match(self, index1, index2):

return self.board[index1] == self.board[index2]

def update\_board(self, index1, index2):

self.board[index1] = self.board[index1].upper()

self.board[index2] = self.board[index2].upper()

**Player.py**

class Player:

def \_\_init\_\_(self, player\_id):

self.player\_id = player\_id

self.score = 0

def increase\_score(self):

self.score += 1

**GameController.py**

from GameBoard import GameBoard

from Player import Player

class GameController:

def \_\_init\_\_(self, player\_count=2):

self.players = [Player(i) for i in range(player\_count)]

self.current\_player\_index = 0

self.board = GameBoard()

self.game\_over = False

def switch\_player(self):

self.current\_player\_index = (self.current\_player\_index + 1) % len(self.players)

def play\_turn(self):

current\_player = self.players[self.current\_player\_index]

print(f"\nPlayer {current\_player.player\_id + 1}'s turn.")

self.board.display\_board()

index1 = int(input('Enter the index for tile 1: '))

index2 = int(input('Enter the index for tile 2: '))

if self.board.is\_match(index1, index2):

print("It's a match!")

current\_player.increase\_score()

self.board.update\_board(index1, index2)

else:

print('Not a match!')

self.switch\_player()

def check\_game\_end(self):

if all(c.isupper() for c in self.board.board):

self.game\_over = True

self.declare\_winner()

def declare\_winner(self):

highest\_score = max(player.score for player in self.players)

winners = [player.player\_id + 1 for player in self.players if player.score == highest\_score]

if len(winners) == 1:

print(f"\nPlayer {winners[0]} wins!")

else:

print("\nIt's a tie between players: " + ', '.join(map(str, winners)))

**Output:**

