Sir Syed University of Engineering & Technology (SSUET)
Software Engineering Department

Course Name: Data Structures & Algorithm (SE-203L)

Semester: 3rd
Batch: 2024S
Section:

PROJECT REPORT

Project Title: ***MEMORY GAME*** *(TILES MATCHING)*

Submitted To:
Ms. Maham Imran

Submitted By:
M. Ahmed Raza

TABLE OF CONTENTS

1. INTRODUCTION OF THE PROJECT

This project is a **Java-based memory matching game** developed for academic purposes as part of our **Data Structures and Algorithms (DSA)** course. The goal of the project was not just to build a functional game but also to apply and demonstrate the use of key programming and DSA concepts in a practical, visual application.

Context:

In today's learning environments, engaging students with interactive tools has become essential. Games are a great way to make learning fun and reinforce technical skills. This memory game helps players improve their concentration and memory skills, while also allowing us, as developers, to apply key DSA concepts in a fun and educational project.

The project was created to meet the following needs:

- To reinforce object-oriented programming and GUI skills using Java.
- To implement core data structure concepts in a real-world project.
- To promote logic development through game mechanics.
- To demonstrate teamwork in software development.

Project Description:

The game involves flipping tiles to find and match pairs of images. It has two levels:

- **Level 1 (4x4)**: A simpler grid for easy memory recall.
- **Level 2 (6x6)**: A more challenging grid for advanced memory skills.

Before each level starts, the player gets a short preview (5 seconds for 4x4, 8 seconds for 6x6) where all tiles are revealed to help with memorization.

When a player clicks on two tiles:

- If the images match, the tiles remain visible.
- If the images don't match, the tiles flip back.

Each matched pair is recorded in a **queue** to maintain the order of matches. After each level, a summary of matched image pairs is shown using this queue.

Objectives:

- Build a memory game using **Java and Swing** for GUI.
- Create **two playable levels** with increasing difficulty.
- Use **data structure concepts** like:
  - **ArrayList**: to store and shuffle image tiles.
  - **2D Arrays**: to place tiles in the grid.
  - **Queue**: to record the order in which image pairs were matched.
- Implement **tile flipping logic** and **preview display** before each stage.
- Show a **match history summary** using a queue after each level.
- Ensure smooth transition between the two levels.

2. ALGORITHM / PSEUDOCODE OF EACH OPERATION

1. Tile Shuffling and Placement

```
ALGORITHM ShuffleAndPlaceTiles(gridSize):
    numPairs ← (gridSize × gridSize) / 2
    imageList ← empty list

    FOR i ← 1 TO numPairs:
        image ← load image "img" + i
        ADD image TO imageList TWICE  // two copies for matching

    SHUFFLE imageList

    index ← 0
    FOR row ← 0 TO gridSize - 1:
        FOR col ← 0 TO gridSize - 1:
            tile ← new Tile(imageList[index])
            place tile at tiles[row][col]
            index ← index + 1
```

2. Tile Click Handling

```
ALGORITHM handleTileClick(tile):
    IF tile is already flipped OR matched OR second tile is selected:
        RETURN

    flip tile

    IF first selected tile is NULL:
        selectedTile1 ← tile
    ELSE:
        selectedTile2 ← tile
        CALL checkForMatch()
```

3. Check for Match Between Two Tiles

```
ALGORITHM checkForMatch():
    IF selectedTile1.image == selectedTile2.image:
        mark both tiles as matched
        ADD image name TO matchHistoryQueue
        selectedTile1 ← NULL
        selectedTile2 ← NULL

        IF all tiles matched:
            CALL showMatchHistory()
    ELSE:
        START timer to flip back both tiles after delay
```

4. Show Preview Before Level Starts

```
ALGORITHM showPreview(seconds):
    FOR EACH tile IN grid:
        flip tile to show image

    WAIT seconds

    FOR EACH tile IN grid:
        flip tile back to hide image
```

5. Show Match History Summary

```
ALGORITHM showMatchHistory():
    message ← "You matched the following tiles:\n"
    FOR EACH match IN matchHistoryQueue:
        message ← message + match + "\n"

    DISPLAY message as popup
```

1. PLAN OF WORK

Day 1-5: Setup & Planning
Meet as a team to decide who's doing what.

Set up the project in IntelliJ/NetBeans with folders for:

GameBoard.java

Tile.java

ImageManager.java

MemoryGame.java

Make sure all tile images are in the resources/ folder.

Goal: Get the whole structure in place so we can start coding right away.

Days 6–12: Core Game Features
Tile + Game Board
Build the Tile class with flip and match logic.

Create the GameBoard with a grid of shuffled tiles.

Add the timer to flip tiles back if they don't match.

Implement the match-checking and preview logic.

Image Handling
Add ImageManager to load images and warn if any are missing.

Goal: Have a working Level 1 game (4x4 grid) by the end of Day 3.

Day 13-18: Leveling Up + Match History
Add Level 2 (6x6 grid) with longer preview time.

Track match history and show it after each level using a pop-up.

Make the game automatically go from Level 1 to Level 2.

Goal: Complete both levels and game flow.

Day 19-22: Testing & Polish
Play through the full game to fix bugs.

Make sure all tiles work, images load, and dialogs show properly.

Clean up the code and make sure it’s easy to read.

Goal: Have a fully working, clean version of the game ready to demo or submit.

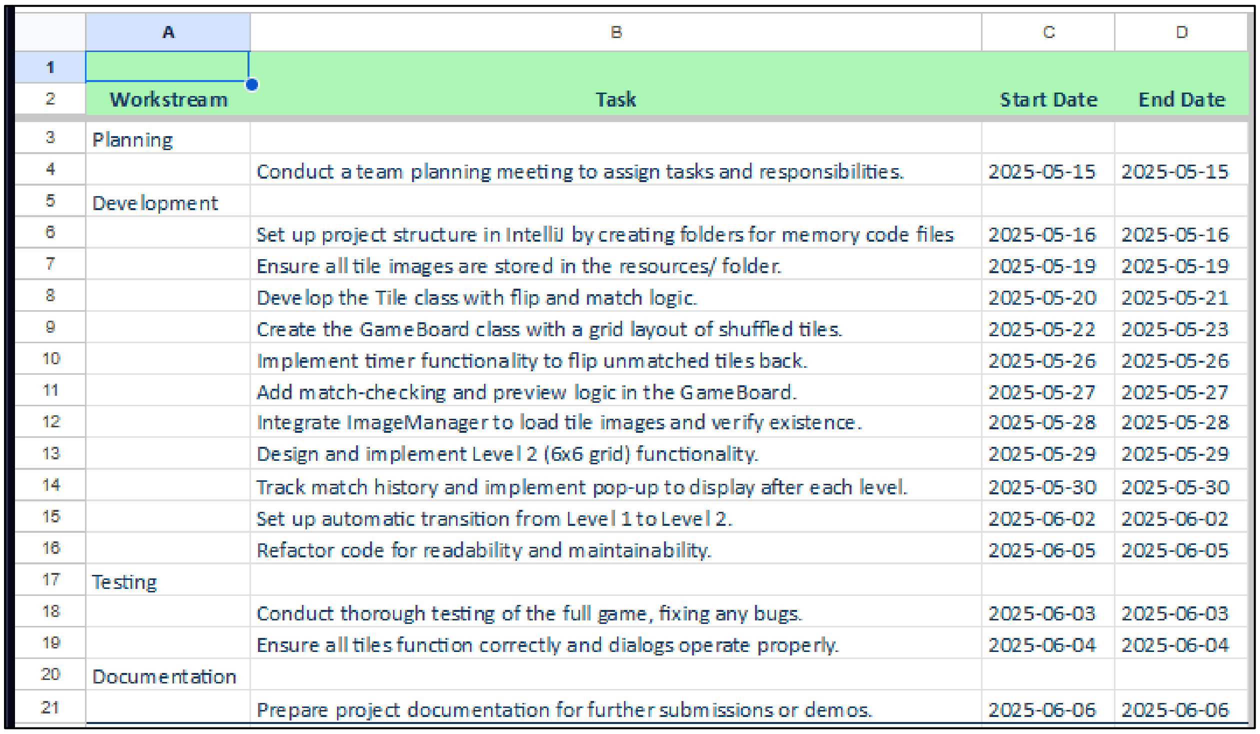
Tools
Java (with Swing for UI)

IntelliJ IDEA or Netbeans

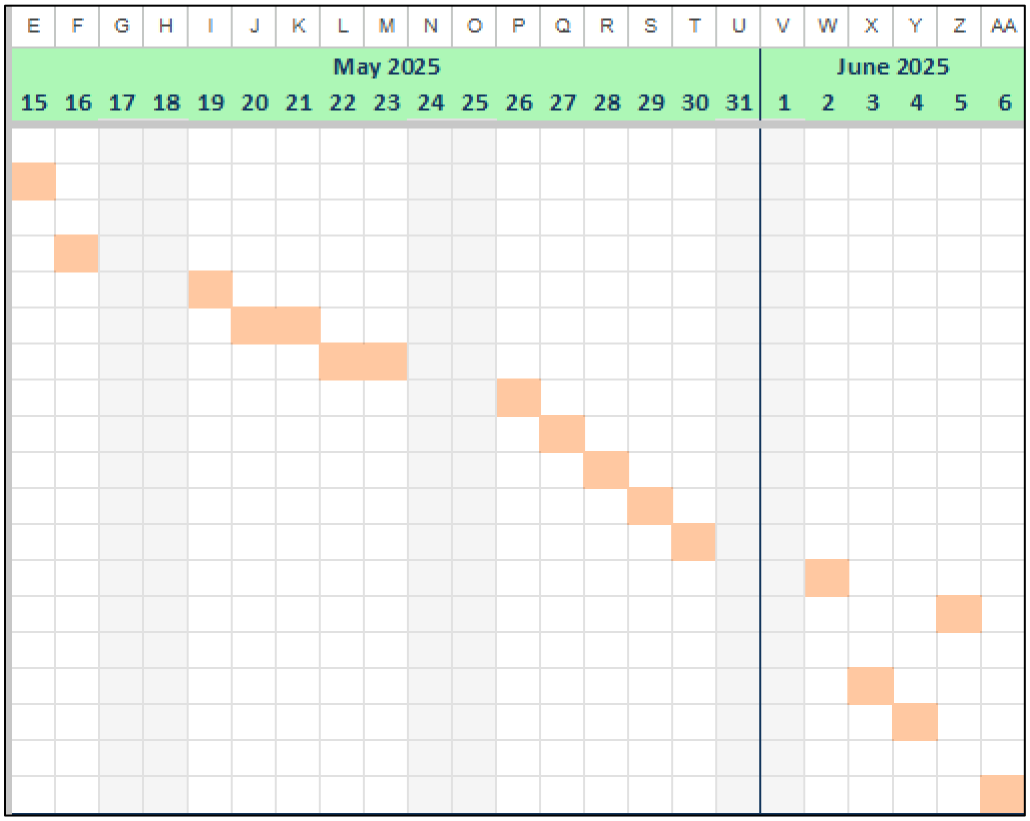
Git (optional, but helpful for backups)

PNG images in a resources/ folder

4. PROJECT SCHEDULING



	A	B	C	D
1				
2	Workstream	Task	Start Date	End Date
3	Planning			
4		Conduct a team planning meeting to assign tasks and responsibilities.	2025-05-15	2025-05-15
5	Development			
6		Set up project structure in IntelliJ by creating folders for memory code files	2025-05-16	2025-05-16
7		Ensure all tile images are stored in the resources/ folder.	2025-05-19	2025-05-19
8		Develop the Tile class with flip and match logic.	2025-05-20	2025-05-21
9		Create the GameBoard class with a grid layout of shuffled tiles.	2025-05-22	2025-05-23
10		Implement timer functionality to flip unmatched tiles back.	2025-05-26	2025-05-26
11		Add match-checking and preview logic in the GameBoard.	2025-05-27	2025-05-27
12		Integrate ImageManager to load tile images and verify existence.	2025-05-28	2025-05-28
13		Design and implement Level 2 (6x6 grid) functionality.	2025-05-29	2025-05-29
14		Track match history and implement pop-up to display after each level.	2025-05-30	2025-05-30
15		Set up automatic transition from Level 1 to Level 2.	2025-06-02	2025-06-02
16		Refactor code for readability and maintainability.	2025-06-05	2025-06-05
17	Testing			
18		Conduct thorough testing of the full game, fixing any bugs.	2025-06-03	2025-06-03
19		Ensure all tiles function correctly and dialogs operate properly.	2025-06-04	2025-06-04
20	Documentation			
21		Prepare project documentation for further submissions or demos.	2025-06-06	2025-06-06



E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
May 2025																	June 2025					
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6
🟧																						
	🟧																					
				🟧																		
					🟧	🟧																
							🟧	🟧														
											🟧											
												🟧										
													🟧									
														🟧								
															🟧							
																		🟧				
																					🟧	
																			🟧			
																				🟧		
																						🟧

5. DATA STRUCTURES USED IN PROJECT

In this memory tile-matching game, we used the following data structures to manage the game logic, gameplay flow, and user interactions:

1. Array (2D Array):

- **Used in:** `Tile[][] tiles`
- **Purpose:** To store the grid of tiles in both 4x4 and 6x6 levels.
- **Reason:** A 2D array makes it easy to loop through rows and columns to display or check the state of each tile.

2. ArrayList (Dynamic Array):

- **Used in:** `List<ImageIcon> images`
- **Purpose:** To temporarily store and shuffle image icons before assigning them to tiles.
- **Reason:** ArrayList allows dynamic resizing and easy shuffling using `Collections.shuffle()`.

3. Queue (LinkedList as Queue):

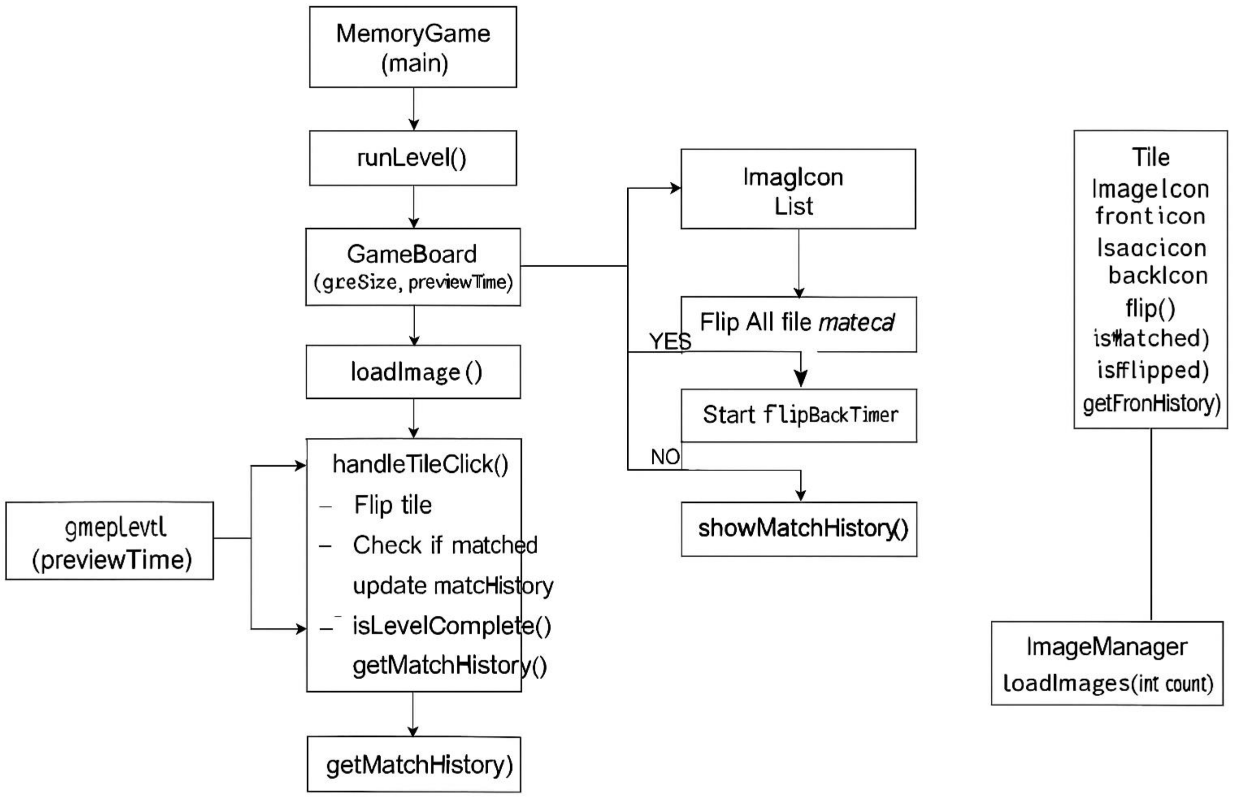
- **Used in:** `Queue<String> matchHistory = new LinkedList<>();`
- **Purpose:** To record the order in which the user matched image pairs.
- **Reason:** Queue helps maintain a First-In-First-Out (FIFO) order to display the match history in the correct sequence.

4. Timer (Swing Timer):

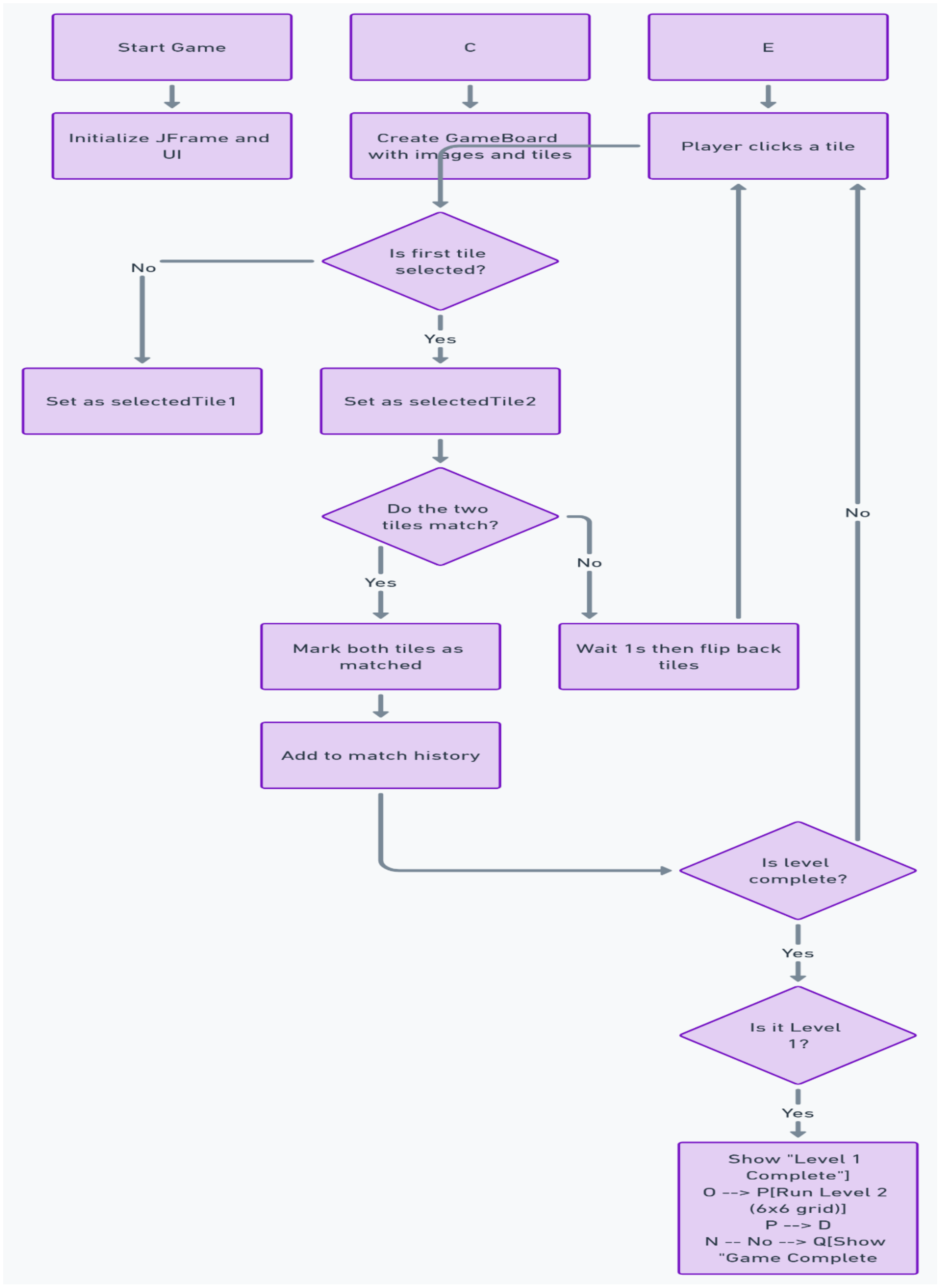
- **Used in:** `javax.swing.Timer`
- **Purpose:** To add delay when flipping tiles back if they do not match.
- **Reason:** It creates a brief pause so the user can see the mismatched pair before they flip back.

---

6. BLOCK DIAGRAM



7. SYSTEM FLOW DIAGRAM

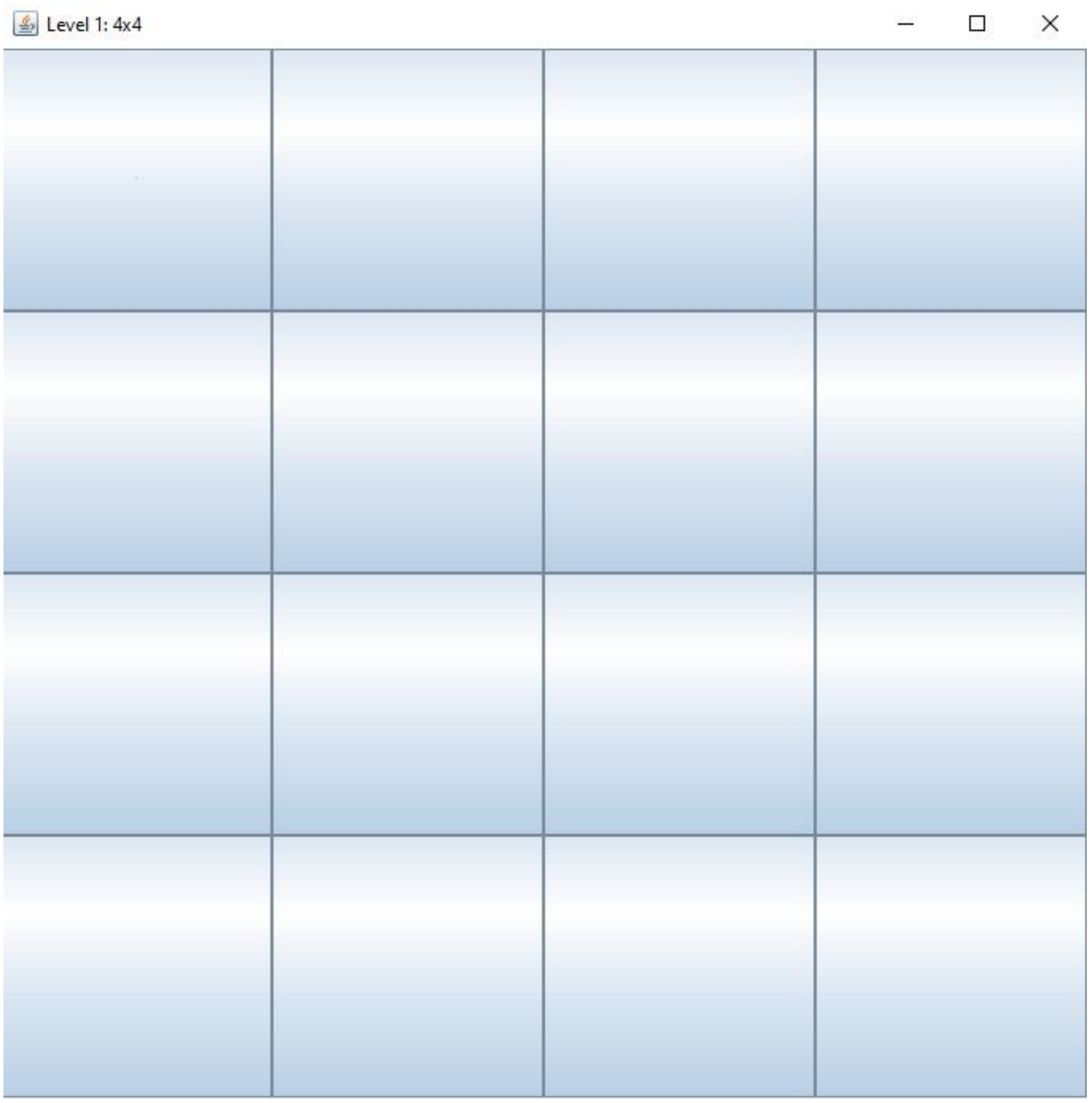


SAMPLE SYSTEM FLOW DIAGRAM

8. USER GUIDE

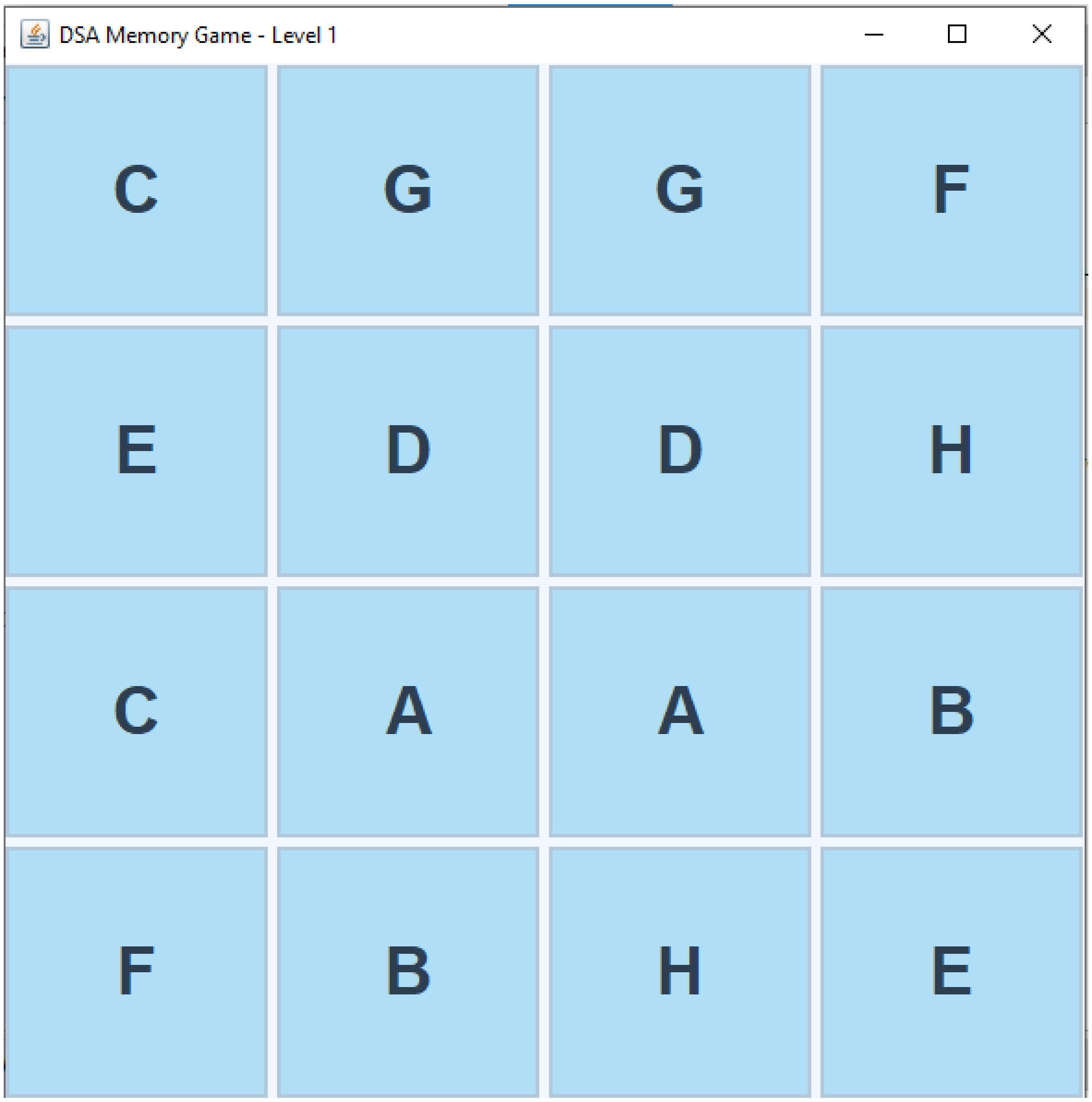
Step 1: Launch the Game

- Open the project in **NETBEANS**.
- Run the program using the `MemoryGame.java` class.
- A game window appears with a grid of tiles facing down.



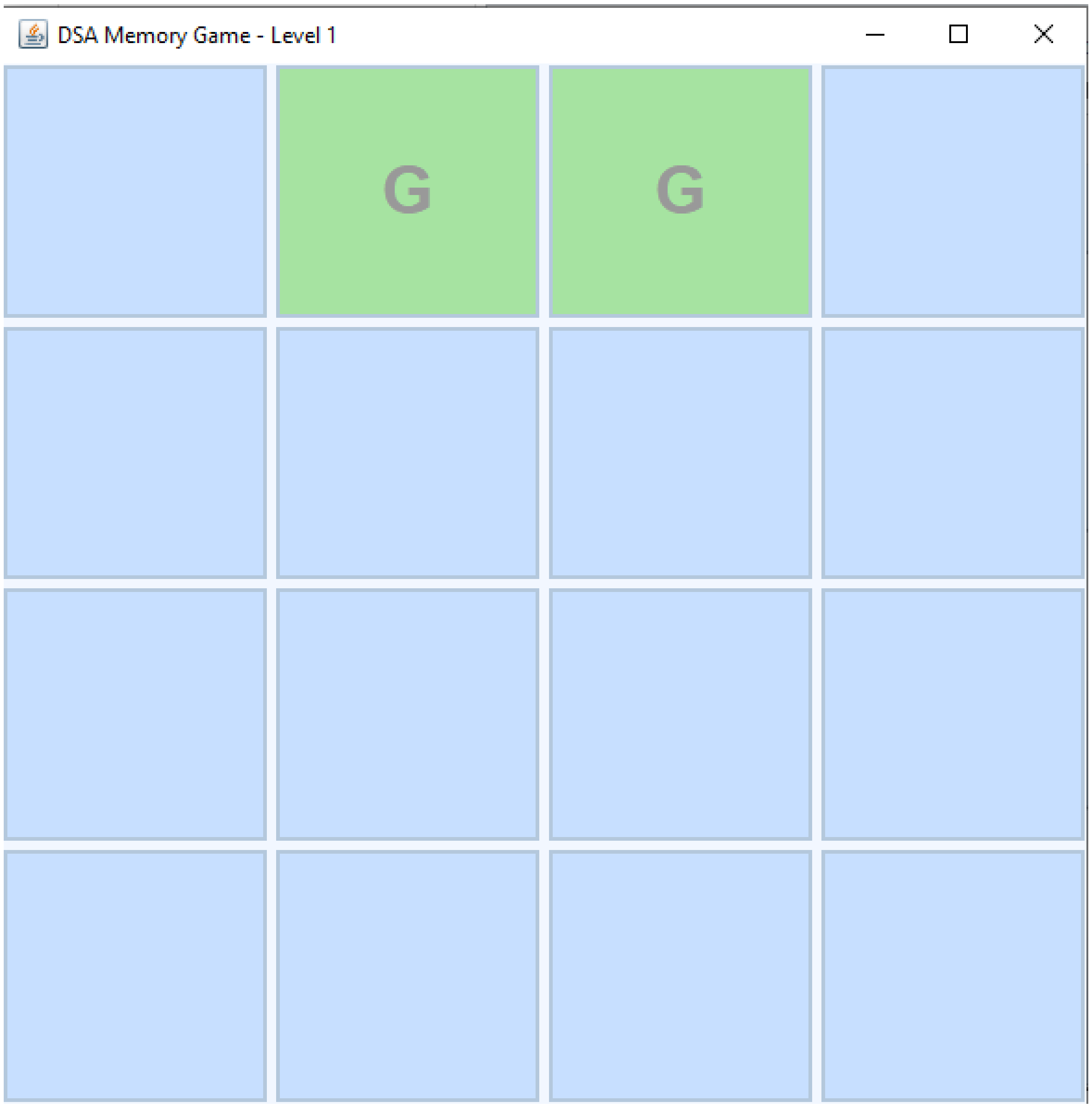
Step 2: Preview Time (Memorization Phase)

- Before you can click, all the tiles will **flip automatically** and show their images.
- This is the preview phase:
  - **5 seconds** for level 1 (4x4)
  - **8 seconds** for level 2 (6x6)



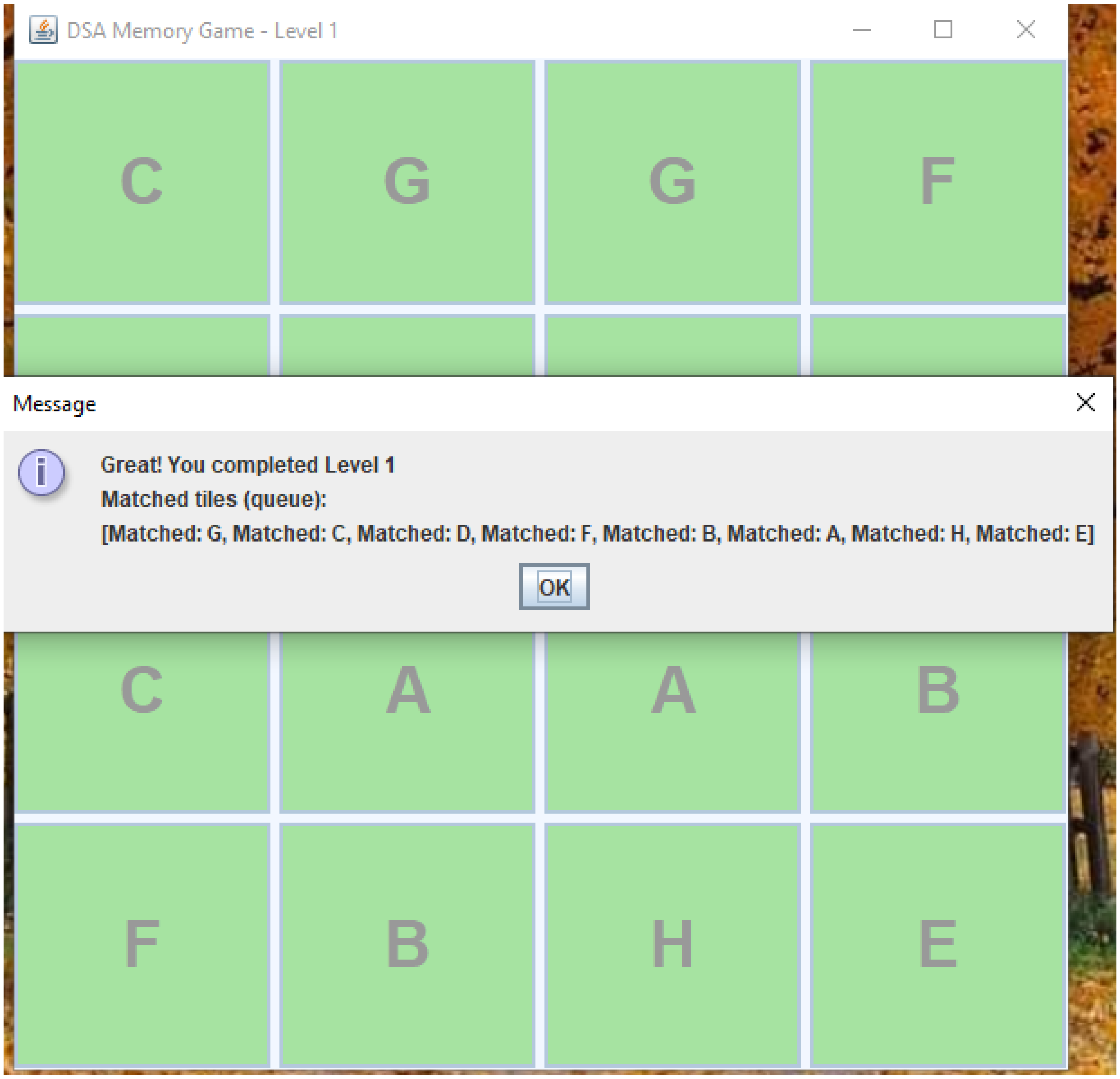
Step 3: Start Playing

- After the preview, tiles flip back.
- **Click on a tile** to flip and reveal the image.
- Click on another tile:
    - If it **matches**, both tiles remain visible.
    - If it **does not match**, both flip back after 1 second.



Step 4: Level Completion

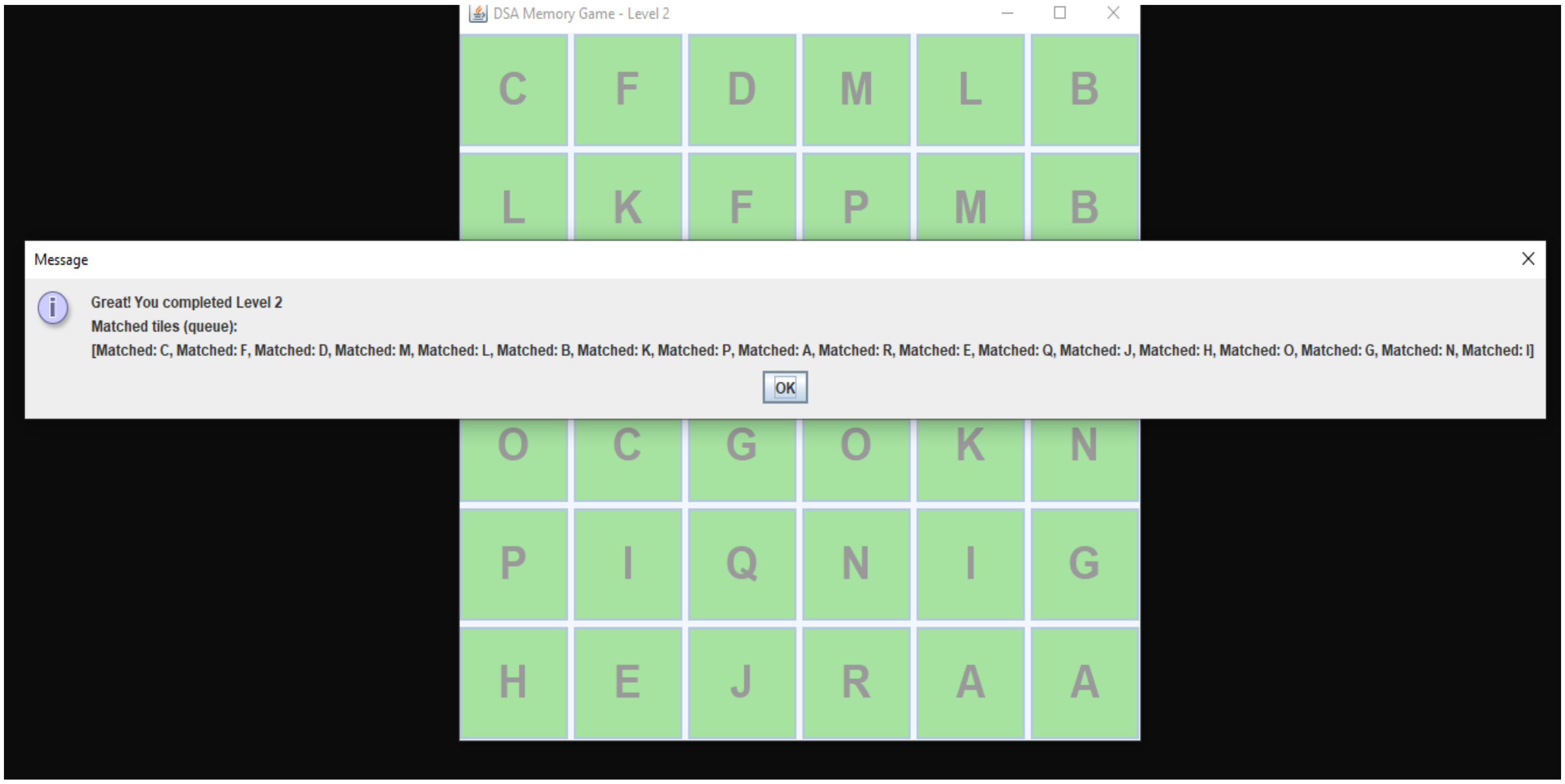
- When all tiles are matched in level 1, a popup appears showing the **match history** (in the order matched).
- Then the game **automatically switches** to the 6x6 grid for level 2.



Great! You completed Level 1
Matched tiles (queue):
[Matched: G, Matched: C, Matched: D, Matched: F, Matched: B, Matched: A, Matched: H, Matched: E]

Step 5: Finish the Game

- After completing the second level (6x6), another **match history popup** appears showing the order of matches in that level.
- After that, the game window closes or you can exit manually.



```
Great! You completed Level 2
Matched tiles (queue):
[Matched: C, Matched: F, Matched: D, Matched: M, Matched: L, Matched: B, Matched: K, Matched: P, Matched: A, Matched: R, Matched: E, Matched: Q, Matched: J, Matched: H, Matched: O, Matched: G, Matched: N, Matched: I]
```

9. CONCLUSION

This project successfully demonstrates how data structures and algorithms can be applied in a real-world game using Java. We developed a memory-based tile matching game with two levels, interactive GUI, animated tile flipping, and a queue to track the match history.

Through this project, we enhanced our understanding of object-oriented programming, graphical user interface design using Swing, and important data structures like arrays, lists, and queues. The game not only challenges the user's memory but also provides a fun and engaging learning experience.

Overall, the project met its objectives of applying DSA concepts practically, developing a multi-stage interactive game, and improving our teamwork and problem-solving skills during implementation.

---

10. FUTURE EXPANSION OF YOUR PROJECT

1. Online Multiplayer Mode

Integrate networking so players from different systems can compete in real-time, with turn-based gameplay and chat functionality.

2. AI Opponent

Introduce an AI-based player with difficulty settings that mimics human memory, providing users a challenging solo experience.

3. Daily Puzzle Challenge

Add a daily puzzle feature where players can complete a time-based challenge and compete on a leaderboard.

4. Performance Analytics

Provide players with statistics such as number of attempts, time per level, match accuracy, and progress over time.

5. Cloud Save and Login System

Implement user accounts and cloud-based saving to allow continuity across devices and progress tracking.
