Memory Match Game (Unity)

Overview

Memory Match is a 2D card-matching game developed in Unity. Players flip two cards at a time to find matching pairs. The project focuses on clean architecture, controlled input handling, animation synchronization, combo mechanics, and persistent best-time saving.

Gameplay

- Dynamic grid generation based on rows and columns.
- Players can flip only two cards at a time.
- Matching cards disappear
- Non-matching cards flip back after a short delay.
- Move counter tracks player actions.
- Combo system rewards consecutive matches.
- Timer tracks completion time.
- Best time is saved between sessions.

Core Systems

Card System

Each card manages its own flip animation and internal state: isRevealed, isMatched, isFlipping, and canInteract. Cards request permission from GridManager before revealing.

GridManager (Game Logic Brain)

- Generates and shuffles card pairs.
- Registers revealed cards using TryRegisterCard().
- Uses coroutine to synchronize animation and match checking.
- Clears state after each match check.

GameManager (Game State Controller)

- Tracks moves made.
- Tracks matched pairs.

- Manages combo streak logic.
- Controls timer system.
- Saves and loads best completion time.
- Detects win condition and ends the game.

Combo System

The combo system increases the combo counter when consecutive matches occur. If a mismatch happens, the combo resets to zero. This encourages accuracy and rewards memory skill.

Timer & Best Time Saving

- Timer starts at gameplay start.
- Stops when all pairs are matched.
- Compares current time with saved best time.
- Updates persistent best time using PlayerPrefs.
- Adds replay value and performance tracking.

Audio System

Audio is managed using a Singleton pattern. The AudioInstance persists across scenes and prevents duplicate instances. Sound effects include card flip and match sounds.

Visual Effects

Sparkle particle effects trigger on successful matches. Matched cards are deactivated after confirmation to enhance feedback.

Technical Challenges Solved

- Prevented 3-card flipping using centralized registration.
- Synchronized animation and game logic using coroutines.
- Eliminated static counter misuse.
- Implemented persistent best-time saving.
- Designed modular and scalable architecture.

Technologies Used

- Unity (2D)
- C#
- Coroutines
- Unity UI System
- Particle System
- AudioSource
- PlayerPrefs

Learning Outcomes

- Separation of concerns in game architecture.
- State management and rule enforcement.
- Coroutine-based asynchronous logic.
- Persistent data handling.
- Debugging race conditions.
- Designing engaging gameplay systems.

Author

Shashank Malhotra

Unity Developer