# 2D TILE MATCH PUZZLE GAME

## Game Description:

### Gameplay:

This is a **2D puzzle Game** Created in unity. The main Objective of this game is to **free the tiles** from the 6x6 grid, by **pushing** the tiles from the grid to below tray. When **two identical tiles** are matched in the tray, they are paired and **removed** from the **tray**. The player must **clear all tiles** in the grid to complete a level.

Since the tray has limited **capacity of 5**, player must carefully choose which tiles to collect to avoid filling the tray without forming matches. Also, player can select only the free tiles from the grid. A tile is considered free if it is located in the top row or there is no tile above it in the same column. Once a pair is matched and removed, the grid automatically moves the tiles to fill up the removed positions.

The player attains a **higher score** when all the **levels are completed** without restarting.in contrast, if the player is unable to complete a level, they can decide to start again on the same level. As a result, any points they may have accumulated will be **set back to zero**. They will then be required to play the level again in order to gain points.

## Implementation details:

### Input system:

The game uses Unity's **Old Input Manager**, specifically **Unity's Event System with IPointerClickHandler**, because the tiles are implemented as **UI elements**. This input system works seamlessly with UI-based interactions and provides a **stable and efficient solution for mobile platforms such as Android**.

### Grid structure:

The **game grid** is represented using a **2D array**, which stores the position and state of each tile. Each tile contains a **tileid** for matching purposes and **row** and **column** for position in the grid.

There are totally **nine types of tiles** in the grid and the tiles are created from **prefabs** that are referenced through a **Scriptableobjec**t in order to have control over tile types and tile graphics. By using Scriptableobjects and prefab variants, it is very easy to add a new type by simply adding a new prefab variant and changing the corresponding sprite and ID without having to alter anything in the game logic. This provides flexibility and scalability.

### Level structure:

Each level uses a **predefined level asset** created as **scriptableobjects**. when a new level starts, the current grid is **cleared and recreated** on the basis of the information found in the predefined level asset.

The **level scriptableobject** holds information about number of **rows**, number of **columns** and an **array of tileids** that specify a layout for a grid in a level. To implement a new level with new type of grid like 8x8, the developer only needs to create a new variant of the Scriptableobject with just the values of the rows, columns, and Tile ID array changed. This model makes it easy to create different grids with a variety of configurations.

### Tile Selection and matching mechanism:

A tile is considered free if it is not blocked by another tile on top of it. This check is performed using the tile's **row and column position** within the grid. Once a free tile is selected, removed from the grid's 2D array and added to the ray for calculating pairs.

when a tile is removed from the grid, the game updates the column that had the **tile deleted** instead of recomputing the whole grid. All tiles located below the removed tile **shift upward** to fill the empty space. During this process,2D array that holds the logical grid data is updated based on the new row position.

**Tile Matching logic** is handled within the tray system. A **Dictionary<int, List<Tile>>** is used, where the key represents the **Tile ID** and the value stores all tiles of that type currently present in the tray. Each time a tile is added:

- The dictionary is updated with the tile's ID.
- If **two identical Tile IDs** are detected, those tiles are **paired and removed** from the tray.
- The matched tiles are destroyed, and the **player's score is increased** accordingly.

If the tray reaches its maximum capacity of five tiles **without forming any valid match**, the game transitions to a **loss state**, ending the level.

## Challenges Faced:

### Grid Layout Reordering:

The tile layout was initially managed by **Unity's Gridlayoutgroup component**. However, the Gridlayoutgroup **automatically rearranged** all remaining tiles when tiles were destroyed after matching. As a result, the intended gameplay logic and visual consistency were **broken**.

So, the **code** was used to **manually control** the grid layout. A logical 2D grid structure was used to manage tile positions, and visual **placement** was handled by directly updating each **tile's Recttransform**. By this, whenever a tile was removed, only the affected column was updated, guaranteeing that unrelated tiles stayed in place. This method preserved steady and predictable gameplay behavior, allowed for exact control over tile movement.

### Tray matching Logic:

The initial idea for handling tray matching was to use **set** to detect matching tile because it does **not allow duplicates**. But this approach has several drawbacks such as it does not allow to track identifying the presence of two identical tiles since it does not track frequency. It was challenging to deal with situations like controlled pairing or **several identical tiles**. Since, Sets do not maintain order, which is **crucial** for tray animations and **UI positioning**, **tile removal logic became unclear**.

Because of these disadvantages, I decided to use **Dictionary<int,list<tile>>** which is a **HashMap technique**. the key represents the **Tile ID** and the value stores **all tiles** of that type currently present in the tray. This allows for effective tile lookup and **grouping** based on the ID, which in results for **Easy detection** and removal of **matching pairs.**

## Additional Features:

### Level Transition Animation:

To improve user experience, a level transition animation is used when the player completes a level and moves to the next level. **Level Manager** triggers the **transition animation** before loading the next level, ensuring that level changes feel **smooth and polished**.

### Audio Feedback:

Basic audio feedback is included to enhance player interaction. A short **sound effect plays** when a tile is **added to the tray**, and also when a pair is **matched and removed**.

## GITHUB Link:

https://github.com/SUBRAMANIYAN01/2DPuzzleGame