2048 Merge

This Unity package provides a complete foundation for developing merge-based games, such as 2048 Merge, Fruit Merge, and Ball Merge, where objects of the same category collide and combine into higher-level items. It offers an easy-to-use system for game developers to quickly prototype, reskin, and customize their merge game without complex coding.

Why Choose This Package?

✅ **Pre-Built Merge Mechanics** – No need to code from scratch! The package includes a ready-to-use system where similar objects automatically merge upon collision, generating a higher-tier item.

✅ **Easy to Reskin & Customize** – Swap out sprites, models, and UI elements effortlessly to create **your unique merge game** in minutes.

✅ **Optimized for Mobile** – Designed specifically for **Android & iOS**, ensuring smooth gameplay, automatic screen scaling, and touch-friendly controls.

✅ **Simple & Modular Codebase** – Well-structured scripts make it **easy to modify**, allowing developers to tweak game rules, add new item types, and expand game mechanics.

✅ **Dynamic Gameplay Area Adjustment** – The package includes **automated screen scaling** using colliders, ensuring proper gameplay boundaries regardless of screen size.

✅ **Prefab-Based System** – All merge items, UI elements, and effects are stored as **reusable prefabs**, making level design and content updates faster and more efficient.

✅ **Quick Build & Deployment** – The package is designed to help developers **quickly create, test, and publish their games** with minimal setup.

Whether you're building a **2048-inspired puzzle**, a **fruit-merging game**, or a **physics-based ball merge challenge**, this package provides **everything you need to start development immediately**.

Technical Specifications

- **Unity Version:** Compatible with Unity **2022.3.56f1**
- **Render Pipeline Support:** Built-in
- **Physics Engine:** Uses Unity's default **2D Physics Engine** (supports custom colliders).
- **Dependencies:** Requires **TextMeshPro** (for UI).

Project Contents

1. **Audio**

- Contains all audio files used in the game.
- Examples: Background music, sound effects, voice-overs.

2. **Document**

- Stores documentation related to the project.
- Examples: Design documents, technical specifications, guides.

3. **Fonts**

- Contains font assets used in the game UI.

4. **Graphics**

- Contains graphical assets such as textures, sprites, and materials.
- **2D**: Stores 2D sprites, icons, and textures.
- **Material**: Contains materials and shaders for rendering.

5. **Physics**

- Contains physics-related materials used in the game.
- Examples: Physics materials for collision and friction settings.

6. **Prefabs**

- Stores reusable game objects and UI elements.
- Examples:
  - Game item prefabs
  - UI prefabs
  - Particle effect prefabs

7. **Resource**

- Contains scriptable objects that store settings for various features.
- Examples:
  - Sound settings
  - Monetization settings

8. **Scenes**

- Stores Unity scene files.
- Examples:
  - Splash screen scene
  - Mae Game logic and mechanics.
  - Examples: In game scenes

9. **Scripts**

- Contains all the scripts related to th
  - Player movement scripts
  - UI behavior scripts
  - Game mechanics scripts

Gameplay Area Adjustment Based on Screen Size

Overview

This document describes the functionality of adjusting the gameplay area dynamically based on the mobile screen size using colliders. The colliders are positioned on all four sides of the screen and automatically adjust when the screen size changes. This ensures that the play area scales accordingly while maintaining proper boundaries.

Functionality

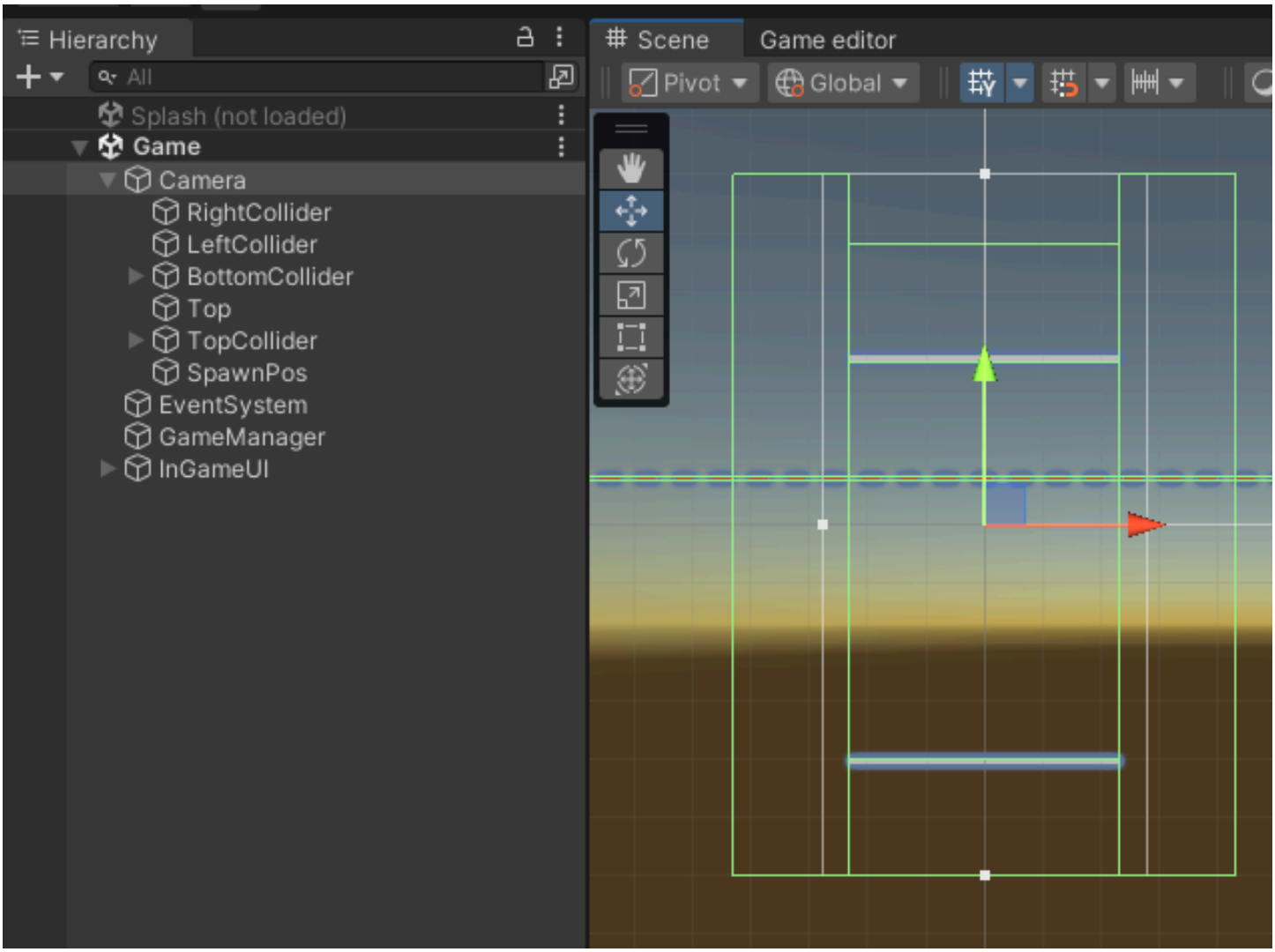
1. **Dynamic Adjustment:**
   - The colliders automatically reposition themselves based on the mobile screen size.
   - If the screen size increases, the play area expands accordingly.
   - If a fixed play area is required, it can be set manually.
2. **Script Implementation:**
   - The script responsible for this functionality is CameraCollider.cs, which is attached to the camera.
   - The collider positions are calculated based on the camera's **orthographic size**.
   - This ensures that the colliders always align with the edges of the visible screen.

Key Components

- **CameraCollider.cs** (Attached to Camera):
  - Calculates the correct position for the colliders.
  - Updates the colliders when the screen size changes.
- **Colliders on Four Sides:**
  - Positioned on the **Top, Bottom, Left, and Right** edges of the screen.
  - Prevents objects from moving beyond the screen boundaries.

Configuration

- If you want the play area to dynamically adjust, the script will handle it automatically.
- If a fixed play area is preferred, set manual values for the collider positions and disable automatic adjustment.



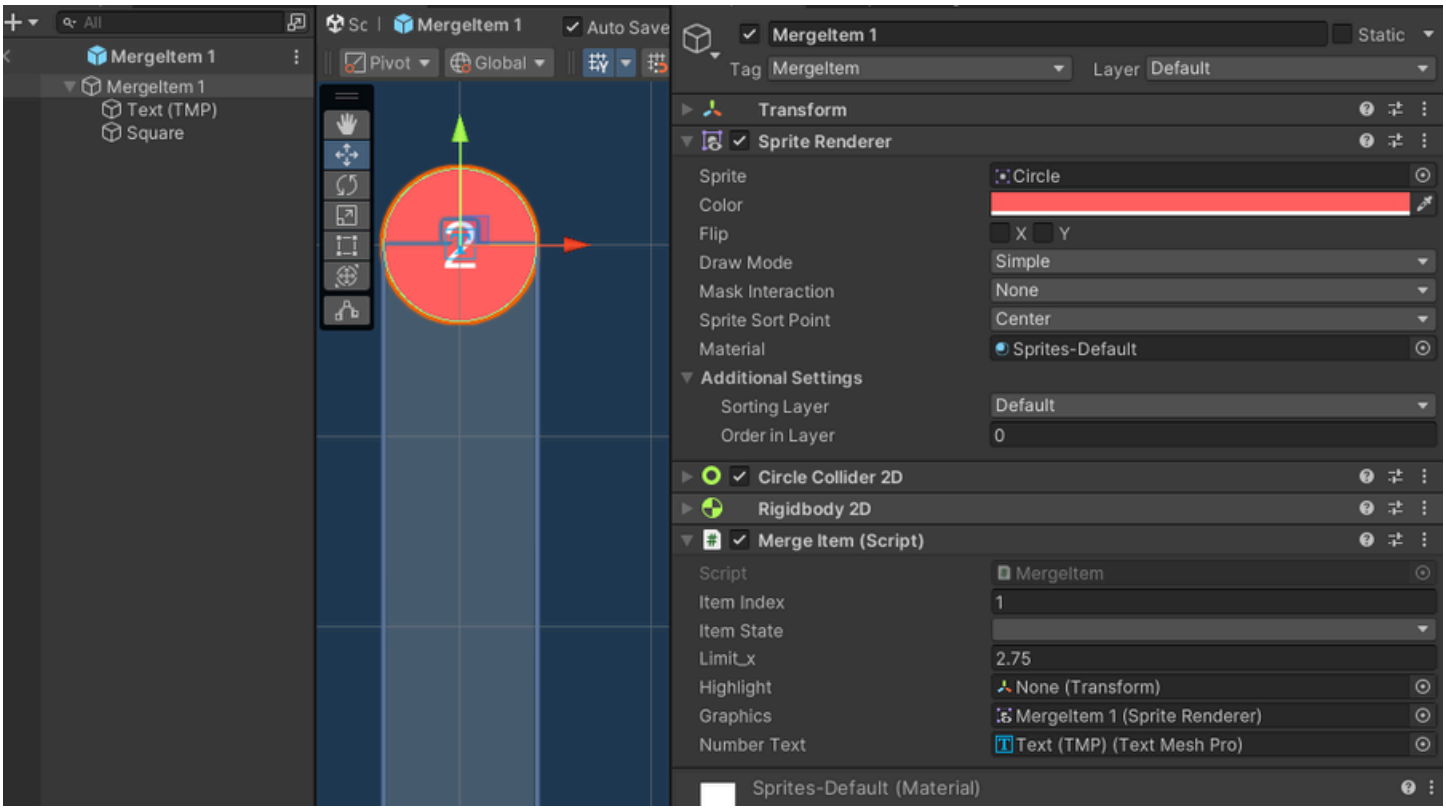
Reskinning a Merge Item - Guide

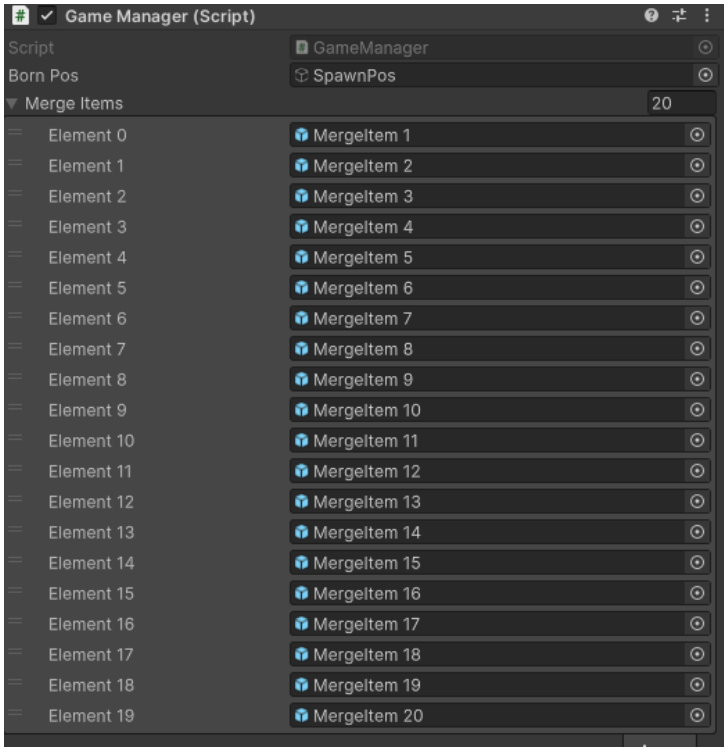
1. **Locate and Select the Prefab:**
   - Navigate to **Assets/Game/Prefab/Merge Item**.
   - Select the prefab you want to modify.
2. **Modify the Visual Appearance:**
   - Change the sprite to your new desired sprite.
   - Adjust the color according to your design.
   - Set the appropriate collider size.

- Adjust the scale to fit the game design.

3. **Configuring the limit_x Field (Position Limit):**
   - The limit_x field defines the horizontal movement boundary of the item.
   - This value is automatically calculated in the OnEnable() method based on the camera's orthographic size using the formula:
   - **limit_x = Camera.main.orthographicSize - (transform.localScale.x * 1.5f);**
   - This ensures that the item remains within the visible screen area.
   - If you want to set a custom position limit, you can **comment out this line in the script** and manually assign limit_x as per your requirement.
4. **Managing Merge Items in GameManager.cs:**
   - All merge items are added in the **GameManager.cs** script.
   - Each item has an itemIndex field that follows an **incremental format**:
     - Merge Item 1 → itemIndex = 1
     - Merge Item 2 → itemIndex = 2
     - Any new item should have an index **one greater than the last item** (i.e., last item index + 1).

Following these steps ensures that the merge items function correctly and maintain consistency in the game.


MergeItem 1
Text (TMP)
Square
Auto Save
Pivot
Global
MergeItem 1
Static
Tag MergeItem
Layer Default
Transform
Sprite Renderer
Sprite Circle
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprites-Default
Additional Settings
Sorting Layer Default
Order in Layer 0
Circle Collider 2D
Rigidbody 2D
Merge Item (Script)
Script MergeItem
Item Index 1
Item State
Limit_x 2.75
Highlight None (Transform)
Graphics MergeItem 1 (Sprite Renderer)
Number Text Text (TMP) (Text Mesh Pro)
Sprites-Default (Material)


Game Manager (Script)
Script GameManager
Born Pos SpawnPos
Merge Items 20
Element 0 MergeItem 1
Element 1 MergeItem 2
Element 2 MergeItem 3
Element 3 MergeItem 4
Element 4 MergeItem 5
Element 5 MergeItem 6
Element 6 MergeItem 7
Element 7 MergeItem 8
Element 8 MergeItem 9
Element 9 MergeItem 10
Element 10 MergeItem 11
Element 11 MergeItem 12
Element 12 MergeItem 13
Element 13 MergeItem 14
Element 14 MergeItem 15
Element 15 MergeItem 16
Element 16 MergeItem 17
Element 17 MergeItem 18
Element 18 MergeItem 19
Element 19 MergeItem 20