美妆叠叠乐游戏软件源程序

```
using UnityEngine;
namespace MakeupPuzzle.Core
{
	public static class GameLogger
	{
		private const bool ENABLE_LOG = false;
		public static void Log(string message)
		{
			if (ENABLE_LOG)
			{
				UnityEngine.Debug.Log(message);
			}
		}
		public static void Log(string tag, string message)
		{
			if (ENABLE_LOG)
			{
				UnityEngine.Debug.Log($"[{tag}] {message}");
			}
		}
		public static void LogColor(string message, string color)
		{
			if (ENABLE_LOG)
			{
				UnityEngine.Debug.Log($"<color={color}>{message}</color>");
			}
		}
		public static void LogWarning(string message)
		{
			UnityEngine.Debug.LogWarning(message);
		}
		public static void LogWarning(string tag, string message)
		{
			UnityEngine.Debug.LogWarning($"[{tag}] {message}");
		}
		public static void LogError(string message)
		{
			UnityEngine.Debug.LogError(message);
		}
		public static void LogError(string tag, string message)
		{
			UnityEngine.Debug.LogError($"[{tag}] {message}");
		}
		public static void LogPerformance(string message)
		{
			if (ENABLE_LOG)
			{
				UnityEngine.Debug.Log($"<color=cyan>[Performance] {message}</color>");
			}
		}
		public static void LogPerformance(string tag, float timeMs)
		{
			if (ENABLE_LOG)
			{
				UnityEngine.Debug.Log($"<color=cyan>[Performance] {tag}: {timeMs:F2}ms</co
lor>");
			}
		}
	}
}
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
namespace MakeupPuzzle.Core
```

```
{
    public class PersistentSingleton<T> : MonoBehaviour where T : MonoBehaviour
    {
        private static T instance;
        private static object lockObject = new object();
        private static bool applicationIsQuitting = false;
        public static T Instance
        {
            get
            {
                if (applicationIsQuitting)
                {
                    return null;
                }
                lock (lockObject)
                {
                    if (instance == null)
                    {
                        instance = FindObjectOfType<T>();
                        if (instance == null)
                        {
                            GameObject singletonObject = new GameObject();
                            instance = singletonObject.AddComponent<T>();
                            singletonObject.name = typeof(T).ToString() + " (Singleton)";
                        }
                    }
                    return instance;
                }
            }
        }
        protected virtual void Awake()
        {
            if (instance == null)
            {
                instance = this as T;
                DontDestroyOnLoad(gameObject);
            }
            else if (instance != this)
            {
                Destroy(gameObject);
            }
        }
        protected virtual void OnApplicationQuit()
        {
            applicationIsQuitting = true;
        }
        protected virtual void OnDestroy()
        {
            if (instance == this)
            {
                instance = null;
            }
        }
    }
}
using UnityEngine;
using UnityEngine.SceneManagement;
namespace MakeupPuzzle.Core
{
    public class Singleton<T> : MonoBehaviour where T : MonoBehaviour
    {
        private static T instance;
        private static bool applicationIsQuitting = false;
        public static T Instance
        {
            get
            {
                if (applicationIsQuitting)
                {
                    GameLogger.LogWarning($"[Singleton] Instance of {typeof(T)} already destr
oyed. Application is quitting.");
                    return null;
                }
                if (instance == null)
                {
                    instance = FindObjectOfType<T>();
                    if (instance == null)
                    {
                        GameLogger.LogWarning($"[Singleton] No instance of {typeof(T)} foun
d in scene. " +
                                                                           "Please add a GameObject with this component to the
scene.");
                    }
                }
                return instance;
            }
        }
        protected virtual void Awake()
        {
            if (instance != null && instance != this)
            {
                GameLogger.LogWarning($"[Singleton] Duplicate instance of {typeof(T)} found o
n {gameObject.name}. Destroying duplicate.");
                Destroy(gameObject);
                return;
            }
            instance = this as T;
            GameLogger.Log($"[Singleton] {typeof(T)} instance registered: {gameObject.name}
");
        }
        protected virtual void OnDestroy()
        {
            if (instance == this)
            {
                instance = null;
                GameLogger.Log($"[Singleton] {typeof(T)} instance destroyed and reference cle
ared");
            }
        }
        protected virtual void OnApplicationQuit()
        {
            applicationIsQuitting = true;
        }
    }
}
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "Brand_", menuName = "MakeupPuzzle/Brand Config")]
    public class BrandConfigSO : ScriptableObject
    {
        [Header("品牌标识")]
        [Tooltip("品牌ID（与文件夹名称一致，如：DuShiMeiYing）")]
        public string brandID = "DuShiMeiYing";
        [Header("显示信息")]
        [Tooltip("品牌显示名称（如：都市魅影）")]
        public string brandDisplayName = "都市魅影";
        [Tooltip("品牌图标（用于按钮）")]
        public Sprite brandIcon;
        [Header("可选配置")]
        [Tooltip("品牌描述（预留）")]
        [TextArea(2, 4)]
        public string brandDescription = "";
        [Tooltip("排序权重（数值越小越靠前）")]
        public int sortOrder = 0;
        private void OnValidate()
        {
            if (string.IsNullOrEmpty(brandID))
            {
                GameLogger.LogWarning($"[BrandConfigSO] {name} 的 brandID 为空！");
            }
            if (string.IsNullOrEmpty(brandDisplayName))
            {
                GameLogger.LogWarning($"[BrandConfigSO] {name} 的 brandDisplayName
为空！");
            }
            if (brandIcon == null)
            {
                GameLogger.LogWarning($"[BrandConfigSO] {name} 的 brandIcon 未设置！");
            }
        }
    }
}
using UnityEngine;
using System.Collections.Generic;

namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "Box_", menuName = "MakeupPuzzle/Collection Box")]
    public class CollectionBoxSO : ScriptableObject
    {
        [Header("礼盒标识")]
        [Tooltip("礼盒ID（格式：品牌ID_Box编号，如：DuShiMeiYing_Box1）")]
        public string boxID = "DuShiMeiYing_Box1";
        [Tooltip("礼盒名称（如：都市魅影套装1）")]
        public string boxName = "都市魅影套装1";
        [Header("礼盒外观")]
        [Tooltip("礼盒图片（未解锁时显示，解锁后正常显示）")]
        public Sprite boxSprite;
        [Header("化妆品列表")]
        [Tooltip("该礼盒包含的化妆品列表")]
        public List<CosmeticItemSO> cosmeticItems = new List<CosmeticItemSO>();
        [Header("UI布局配置")]
        [Tooltip("化妆品图标的挂载点位置（相对于礼盒中心的偏移量）")]
        public List<Vector2> mountPoints = new List<Vector2>();
        [Header("完成奖励")]
        [Tooltip("集齐礼盒后奖励的道具类型")]
        public PowerUpType rewardPowerUpType = PowerUpType.ClearSlot;
        [Tooltip("奖励道具数量")]
        public int rewardPowerUpCount = 1;
        [Header("可选配置")]
        [Tooltip("礼盒描述（预留）")]
        [TextArea(2, 4)]
        public string boxDescription = "";
        public int GetCosmeticCount()
        {
            return cosmeticItems.Count;
        }
        public List<string> GetCosmeticIDs()
        {
            List<string> ids = new List<string>();
            foreach (var cosmetic in cosmeticItems)
            {
                if (cosmetic != null)
                {
                    ids.Add(cosmetic.cosmeticID);
                }
            }
            return ids;
        }
        private void OnValidate()
        {
            if (string.IsNullOrEmpty(boxID))
            {
                GameLogger.LogWarning($"[CollectionBoxSO] {name} 的 boxID 为空！");
            }
            if (boxSprite == null)
            {
                GameLogger.LogWarning($"[CollectionBoxSO] {name} 的 boxSprite 未设置！
");
            }
            if (cosmeticItems.Count == 0)
            {
                GameLogger.LogWarning($"[CollectionBoxSO] {name} 的化妆品列表为空！");
            }
            for (int i = 0; i < cosmeticItems.Count; i++)
            {
                if (cosmeticItems[i] == null)
                {
                    GameLogger.LogWarning($"[CollectionBoxSO] {name} 的化妆品列表中第 {
i} 项为null！");
                }
            }
            if (mountPoints.Count < cosmeticItems.Count)
            {
                int needCount = cosmeticItems.Count - mountPoints.Count;
                for (int i = 0; i < needCount; i++)
                {
                    mountPoints.Add(Vector2.zero);
                }
                GameLogger.Log($"[CollectionBoxSO] {name} 自动补齐了 {needCount} 个挂载
点（默认位置为0,0）");
            }
        }
    }
}
using System;
using System.Collections.Generic;
using UnityEngine;
namespace MakeupPuzzle.Core
```

```
{
    [Serializable]
    public class CollectionProgressData
    {
        [Header("解锁数据")]
        public List<string> unlockedBoxIDs = new List<string>();
         public List<string> unlockedCosmeticIDs = new List<string>();
        [Header("奖励发放记录")]
        public List<string> rewardedBoxIDs = new List<string>();
          public CollectionProgressData()
        {
            unlockedBoxIDs = new List<string>();
            unlockedCosmeticIDs = new List<string>();
            rewardedBoxIDs = new List<string>();
        public bool UnlockBox(string boxID)
        {
            if (string.IsNullOrEmpty(boxID))
            {
                GameLogger.LogWarning("[CollectionProgressData] 礼盒ID为空");
                return false;
            }
            if (unlockedBoxIDs.Contains(boxID))
            {
                GameLogger.LogWarning($"[CollectionProgressData] 礼盒 {boxID} 已经解锁过
了");
                return false;
            }
            unlockedBoxIDs.Add(boxID);
            GameLogger.Log($"<color=green>[CollectionProgressData] 解锁礼盒: {boxID}</colo
r>");
            return true;
        }
        public bool IsBoxUnlocked(string boxID)
        {
            return unlockedBoxIDs.Contains(boxID);
        }
        public int GetUnlockedBoxCount()
        {
            return unlockedBoxIDs.Count;
        }
        public bool UnlockCosmetic(string cosmeticID)
        {
            if (string.IsNullOrEmpty(cosmeticID))
            {
                GameLogger.LogWarning("[CollectionProgressData] 化妆品ID为空");
                return false;
            }
            if (unlockedCosmeticIDs.Contains(cosmeticID))
            {
                GameLogger.LogWarning($"[CollectionProgressData] 化妆品 {cosmeticID} 已
经解锁过了");
                return false;
            }
            unlockedCosmeticIDs.Add(cosmeticID);
            GameLogger.Log($"<color=green>[CollectionProgressData] 解锁化妆品: {cosmeticI
D}</color>");
            return true;
        }
        public bool IsCosmeticUnlocked(string cosmeticID)
        {
            return unlockedCosmeticIDs.Contains(cosmeticID);
        }
        public int GetUnlockedCosmeticCount()
        {
            return unlockedCosmeticIDs.Count;
        }
        public bool MarkBoxRewarded(string boxID)
        {
            if (string.IsNullOrEmpty(boxID))
            {
                GameLogger.LogWarning("[CollectionProgressData] 礼盒ID为空");
                return false;
            }
            if (rewardedBoxIDs.Contains(boxID))
            {
                GameLogger.LogWarning($"[CollectionProgressData] 礼盒 {boxID} 的奖励已经
发放过了");
                return false;
            }
            rewardedBoxIDs.Add(boxID);
            GameLogger.Log($"<color=green>[CollectionProgressData] 记录礼盒奖励已发放: {b
oxID}</color>");
            return true;
        }
        public bool IsBoxRewarded(string boxID)
        {
            return rewardedBoxIDs.Contains(boxID);
        }
        public void UnlockCosmetics(List<string> cosmeticIDs)
        {
            foreach (var id in cosmeticIDs)
            {
                UnlockCosmetic(id);
            }
        }
        public void ClearAll()
        {
            unlockedBoxIDs.Clear();
            unlockedCosmeticIDs.Clear();
            GameLogger.LogWarning("[CollectionProgressData] 所有收藏进度已清空");
        }
    }
}
using UnityEngine;
using System;
namespace MakeupPuzzle.Core
{
    [Serializable]
    public class CosmeticInstanceData
    {
        public string cosmeticID;          // 引用的化妆品ID
        public bool isLock;                 // 是否为奖杯
        public int layer;                  // 堆叠层级（数值越大越上层）
        public Vector2 position;            // 世界坐标位置
        public float rotation;             // 旋转角度
        public string instanceID;
        [Tooltip("预计算的可点击状态，在关卡编辑器保存时自动计算")]
        public bool precomputedClickable = true;   // 默认true（兼容旧数据）
        public CosmeticInstanceData()
        {
            instanceID = System.Guid.NewGuid().ToString();
        }
        public CosmeticInstanceData(string cosmeticID, Vector3 position, int layer, bool isLock = f
alse)
        {
            this.cosmeticID = cosmeticID;
            this.position = position;
            this.layer = layer;
            this.isLock = isLock;
            this.instanceID = System.Guid.NewGuid().ToString();
        }
    }
}
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "CosmeticItem", menuName = "MakeupPuzzle/Cosmetic Item
")]
    public class CosmeticItemSO : ScriptableObject
    {
        [Header("基础信息")]
        public string cosmeticID;          // 唯一标识符（如: lipstick_dior_999)
        public string itemName;              // 显示名称（如: 迪奥烈艳蓝金唇膏)
        public Sprite icon;                  // 图标
        [Header("分类信息")]
        public string type;                  // 类型（口红、眼影等）
        public string brand;                 // 品牌（迪奥）
        public string colorSeries;          // 色系（红色系）
        [Header("游戏属性")]
        public int rarity;                  // 稀有度（1-5）
        [Header("描述")]
        [TextArea(3, 5)]
        public string description;          // 描述文本（传奇红唇，高订色泽...）
        [Header("预制件引用")]
        public GameObject prefab;            // 关联的GameObject预制
    }
}
using UnityEngine;
using System.Collections.Generic;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "LevelData", menuName = "MakeupPuzzle/Level Data")]
    public class LevelDataSO : ScriptableObject
    {
        [Header("关卡信息")]
        public string levelID;
        public string levelName;
        public int difficulty = 1;                          // 1-5难度等级
        [Header("🏆 奖杯关卡")]
        [Tooltip("勾选后，通关时会显示奖励面板")]
        public bool isTrophyLevel = false;
        public LevelType levelType = LevelType.Normal;
        [Header("堆叠区数据")]
        public List<CosmeticInstanceData> cosmeticInstances = new List<CosmeticInstanceDa
```

```
ta>();
        [Header("订单数据")]
        public List<OrderBagData> orderBags = new List<OrderBagData>();
        [Header("生成参数")]
        public int totalCosmeticsCount;
        public int cosmeticTypeCount;
        public int stackingLayers;
        [Header("算法参数")]
        public int dependencyDepth = 0;                    // 依赖深度 (0-3) - 预留
        public float clusterDensity = 0.5f;            // 聚簇密度 (0.0-1.0) - 预
        public int randomSeed = 0;                         // 随机种子（0表示使用时间种子）
        [Header("验证信息")]
        public bool isValidated;
        public string validationResult;
    }
    [System.Serializable]
    public enum LevelType
    {
        Normal,                // 普通关（每个订单包内化妆品ID相同）
        BrandPerBag,           // 每个包品牌关（每个包内品牌相同，不同包可以不同品牌）
        BrandGlobal,           // 全关卡品牌关（整个关卡所有包都是同一个品牌）
        ColorPerBag,           // 每个包颜色关（每个包内颜色相同，不同包可以不同颜色）
        ColorGlobal,           // 全关卡颜色关（整个关卡所有包都是同一个颜色）
        TypePerBag,            // 每个包类型关（每个包内类型相同，不同包可以不同类型）
        TypeGlobal             // 全关卡类型关（整个关卡所有包都是同一个类型）
    }
}
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "OrderBagConfig", menuName = "MakeupPuzzle/Order Bag C
onfig")]
    public class OrderBagConfigSO : ScriptableObject
    {
        public string configName;          // 配置名称 (如: OrderBagX2Config)
        public int slotCount;              // 槽位数量
        public GameObject bagPrefab;    // 订单包UI预制件 (OrderBagX2Item等)
    }
}
using System;
using System.Collections.Generic;
namespace MakeupPuzzle.Core
{
    [Serializable]
    public class OrderBagData
    {
        public string configID;                        // 引用的订单包配置ID
        public List<string> requiredItems;             // 需要的化妆品ID列表
        public int orderIndex;                          // 订单顺序索引
        public int difficulty;                         // 难度评分（用于排序）
        public int tempSlotCount = 4;                   // 临时槽数（默认4）
        public OrderBagType bagType = OrderBagType.Normal;   // 订单包类型
        public string brandName = "";                   // 品牌名称
        public string colorSeries = "";                    // 色系名称
        public string typeName = "";                        // 类型名称
        public bool isPerfectOrder = false;              // 是否为完美订单
        public string currentSkinID = "normal";        // 当前使用的皮肤ID
    }
    [Serializable]
    public enum OrderBagType
    {
        Normal,                  // 普通包
        BrandPerBag,         // 每个包品牌包
        BrandGlobal,          // 全关卡品牌包
        ColorPerBag,          // 每个包颜色包
        ColorGlobal,          // 全关卡颜色包
        TypePerBag,            // 每个包类型包
        TypeGlobal             // 全关卡类型包
    }
}
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "OrderBagSkin", menuName = "MakeupPuzzle/Order Bag Skin
")]
    public class OrderBagSkinSO : ScriptableObject
    {
        [Header("皮肤标识")]
        [Tooltip("皮肤唯一ID，如: normal, perfect, gold, diamond")]
        public string skinID = "normal";
        [Tooltip("皮肤显示名称，如: 普通，完美，黄金，钻石")]
        public string skinName = "普通皮肤";
        [Header("视觉资源")]
        [Tooltip("背景图 Sprite")]
        public Sprite backgroundSprite;
        [Tooltip("边框图 Sprite（可选）")]
        public Sprite frameSprite;
        [Tooltip("整体色调（可选，默认白色）")]
        public Color tintColor = Color.white;
        [Header("特效（可选）")]
        [Tooltip("完成订单时的粒子特效")]
        public ParticleSystem completionParticle;
        [Tooltip("完成订单时的音效")]
        public AudioClip completionSound;
        [Header("描述信息")]
        [TextArea(3, 5)]
        [Tooltip("皮肤描述，如: 普通订单包的默认外观")]
        public string description;
    }
}
using System;
using System.Collections.Generic;
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [Serializable]
    public class PlayerGameData
    {
        [Header("玩家身份信息")]
        public string playerID = "";                    // 玩家唯一ID（微信OpenID/抖
        public string playerName = "";                   // 玩家昵称
        public long lastSaveTime = 0;                    // 最后保存时间戳（Unix时间）
        [Header("游戏进度")]
        public int currentLevel = 1;                     // 当前关卡
        public int maxUnlockedLevel = 1;                 // 最大解锁关卡
        [Header("货币系统")]
        public int makeupCoins = 0;                      // 美妆币数量
        [Header("收藏馆数据")]
        public CollectionProgressData collectionProgress = new CollectionProgressData();
        [Header("道具库存")]
        public PowerUpInventoryData powerUpInventory = new PowerUpInventoryData();
        [Header("排行榜数据")]
        public int totalScore = 0;                       // 总分（用于排名，预留）
        public int totalStars = 0;                       // 总星数（预留）
        public PlayerGameData()
        {
            collectionProgress = new CollectionProgressData();
            powerUpInventory = new PowerUpInventoryData();
            UpdateSaveTime();
        }
        public void UpdateSaveTime()
        {
            lastSaveTime = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
        }
        public string ToJson()
        {
            UpdateSaveTime();
            return JsonUtility.ToJson(this, true); // true = 格式化输出（便于调试）
        }
        public static PlayerGameData FromJson(string json)
        {
            if (string.IsNullOrEmpty(json))
            {
                GameLogger.LogWarning("[PlayerGameData] JSON为空，返回默认数据");
                return new PlayerGameData();
            }
            try
            {
                return JsonUtility.FromJson<PlayerGameData>(json);
            }
            catch (Exception e)
            {
                GameLogger.LogError($"[PlayerGameData] JSON解析失败: {e.Message}");
                return new PlayerGameData();
            }
        }
        public PlayerGameData Clone()
        {
            string json = ToJson();
            return FromJson(json);
        }
        public void ResetToDefault()
        {
            currentLevel = 1;
            maxUnlockedLevel = 1;
            makeupCoins = 0;
            collectionProgress = new CollectionProgressData();
            powerUpInventory = new PowerUpInventoryData();
            totalScore = 0;
            totalStars = 0;
            UpdateSaveTime();
            GameLogger.Log("[PlayerGameData] 数据已重置为默认值");
        }
    }
}
using System;
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [Serializable]
    public class PowerUpInventoryData
    {
        public int addSlotCount = 0;          // 加槽位道具数量
        public int clearSlotCount = 0;       // 清空槽位道具数量
        public int removeOrderCount = 0;    // 消除订单道具数量
        public bool HasAddSlot() => addSlotCount > 0;
        public bool HasClearSlot() => clearSlotCount > 0;
        public bool HasRemoveOrder() => removeOrderCount > 0;
        public bool ConsumeAddSlot()
        {
            if (addSlotCount > 0)
            {
                addSlotCount--;
                return true;
            }
            return false;
        }
        public bool ConsumeClearSlot()
        {
            if (clearSlotCount > 0)
            {
                clearSlotCount--;
                return true;
            }
            return false;
        }
        public bool ConsumeRemoveOrder()
        {
            if (removeOrderCount > 0)
            {
                removeOrderCount--;
                return true;
            }
            return false;
        }
        public void AddPowerUp(PowerUpType type, int amount = 1)
        {
            switch (type)
            {
                case PowerUpType.AddSlot:
                    addSlotCount += amount;
                    break;
                case PowerUpType.ClearSlot:
                    clearSlotCount += amount;
                    break;
                case PowerUpType.RemoveOrder:
                    removeOrderCount += amount;
                    break;
            }
        }
        public void Save()
        {
            PlayerPrefs.SetInt("PowerUp_AddSlot", addSlotCount);
            PlayerPrefs.SetInt("PowerUp_ClearSlot", clearSlotCount);
            PlayerPrefs.SetInt("PowerUp_RemoveOrder", removeOrderCount);
            PlayerPrefs.Save();
        }
        public void Load()
        {
            addSlotCount = PlayerPrefs.GetInt("PowerUp_AddSlot", 0);
            clearSlotCount = PlayerPrefs.GetInt("PowerUp_ClearSlot", 0);
            removeOrderCount = PlayerPrefs.GetInt("PowerUp_RemoveOrder", 0);
        }
    }
    public enum PowerUpType
    {
        AddSlot,        // 加槽位
        ClearSlot,      // 清空槽位
        RemoveOrder     // 消除订单
    }
}
using System;
namespace MakeupPuzzle.Core
{
    [Serializable]
    public class PowerUpUsageData
    {
        public int addSlotUsedCount = 0;        // 加槽位使用次数
        public int clearSlotUsedCount = 0;      // 清空槽位使用次数
        public int removeOrderUsedCount = 0;    // 消除订单使用次数
        public const int MAX_USES_PER_LEVEL = 2; // 每关最大使用次数
        public bool CanUseAddSlot() => addSlotUsedCount < MAX_USES_PER_LEVEL;
        public bool CanUseClearSlot() => clearSlotUsedCount < MAX_USES_PER_LEVEL;
        public bool CanUseRemoveOrder() => removeOrderUsedCount < MAX_USES_PER_LE
VEL;
        public void UseAddSlot() => addSlotUsedCount++;
        public void UseClearSlot() => clearSlotUsedCount++;
        public void UseRemoveOrder() => removeOrderUsedCount++;
        public void ResetForNewLevel()
        {
            addSlotUsedCount = 0;
            clearSlotUsedCount = 0;
            removeOrderUsedCount = 0;
        }
    }
}
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "TrophyRewardConfig", menuName = "MakeupPuzzle/Trophy
Reward Config")]
    public class TrophyRewardConfigSO : ScriptableObject
    {
        [Header("关卡信息")]
        [Tooltip("关卡编号（如5、10、15）")]
        public int levelNumber;
        [Header("奖励内容")]
        [Tooltip("奖励标题文本（如：开启红毯女王套装）")]
        [TextArea(2, 4)]
        public string rewardTitle = "恭喜解锁新套装！";
        [Tooltip("奖励图标（显示在AwardImage上）")]
        public Sprite rewardIcon;
        [Header("🆕 收藏馆奖励")]
        [Tooltip("通关后解锁的礼盒ID（格式：DuShiMeiYing_Box1）")]
        public string unlockedBoxID = "";
        [Header("可选配置")]
        [Tooltip("对应的化妆品ID（预留，用于后续收藏馆系统）")]
        public string cosmeticID;
        private void OnValidate()
        {
            if (levelNumber <= 0)
            {
                GameLogger.LogWarning($"[TrophyRewardConfigSO] {name} 的 levelNumber
无效（<=0）！");
            }
            if (rewardIcon == null)
            {
                GameLogger.LogWarning($"[TrophyRewardConfigSO] {name} 的 rewardIcon
未设置！");
            }
            if (!string.IsNullOrEmpty(unlockedBoxID))
            {
                if (!unlockedBoxID.Contains("_"))
                {
                    GameLogger.LogWarning($"[TrophyRewardConfigSO] {name} 的 unlocked
BoxID 格式可能不正确（建议格式：品牌ID_Box编号）");
                }
            }
        }
    }
}
using UnityEngine;
namespace MakeupPuzzle.Core
{
    [CreateAssetMenu(fileName = "TutorialConfig", menuName = "MakeupPuzzle/Tutorial Config
", order = 100)]
    public class TutorialConfigSO : ScriptableObject
    {
        [Header("基础配置")]
        [Tooltip("对应的关卡编号")]
        public int levelIndex;
        [Tooltip("引导步骤ID(用于日志和调试)")]
        public string stepID;
        [Header("显示配置")]
        [Tooltip("父节点路径(例如: Canvas/UIManager)")]
        public string parentPath;
        [Tooltip("本地坐标")]
        public Vector3 localPosition;
        [Tooltip("本地缩放")]
        public Vector3 localScale = Vector3.one;
        [Header("预制体路径")]
        [Tooltip("新手引导预制体资源路径(Resources下的相对路径)")]
        public string prefabPath = "Effects/XinShouYinDao";
        [Header("🆕 挖孔遮罩配置")]
        [Tooltip("是否使用挖孔遮罩")]
        public bool useHoleMask = false;
        [Tooltip("挖孔目标节点路径（例如: Canvas/PropPanel/AddSlotButton）")]
        public string holeTargetPath;
        [Tooltip("挖孔边距（扩大挖孔范围，单位：像素）")]
        public Vector2 holePadding = new Vector2(20f, 20f);
        [Tooltip("圆角半径（单位：像素）")]
        public float cornerRadius = 20f;
    }
}
using UnityEngine;
using MakeupPuzzle.Core;

namespace MakeupPuzzle.EditorLevel
{
    public class CosmeticEditorItem : MonoBehaviour
    {
        private CosmeticInstanceData instanceData;
        private CosmeticItemSO cosmeticSO;
        private bool isClickable = true;
        private SpriteRenderer spriteRenderer;
        private Color originalColor;
        private bool isDragging = false;
        private Vector3 dragOffset;
        private Camera mainCamera;
        private void Awake()
        {
            spriteRenderer = GetComponent<SpriteRenderer>();
            if (spriteRenderer != null)
            {
                originalColor = spriteRenderer.color;
            }
            mainCamera = Camera.main;
        }
        public void SetData(CosmeticInstanceData data, CosmeticItemSO so)
        {
            instanceData = data;
            cosmeticSO = so;
            gameObject.name = $"{so.itemName}_Layer{data.layer}";
        }
        public void SetClickable(bool clickable)
        {
            isClickable = clickable;
            UpdateVisual();
        }
        private void UpdateVisual()
        {
            if (spriteRenderer != null)
            {
                if (isClickable)
                {
                    spriteRenderer.color = originalColor;
                }
                else
                {
                    spriteRenderer.color = originalColor * 0.6f;
                }
            }
        }
        public CosmeticInstanceData GetInstanceData()
        {
            return instanceData;
        }
        public void UpdatePositionToData()
        {
            if (instanceData != null)
            {
                instanceData.position = transform.position;
                instanceData.rotation = transform.rotation.eulerAngles.z;
            }
        }
        public CosmeticItemSO GetCosmeticSO()
        {
            return cosmeticSO;
        }
        public bool IsClickable()
        {
            return isClickable;
        }
        #region 拖动编辑功能
        private void OnMouseDown()
        {
            if (mainCamera == null) return;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(Input.mousePosition);
            mouseWorldPos.z = 0;
            dragOffset = transform.position - mouseWorldPos;
            isDragging = true;
            if (spriteRenderer != null)
            {
                spriteRenderer.color = Color.yellow;
            }
            Debug.Log($"开始拖动: {cosmeticSO.itemName}, 层级: {instanceData.layer}");
        }
        private void OnMouseDrag()
        {
            if (!isDragging || mainCamera == null) return;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(Input.mousePosition);
            mouseWorldPos.z = 0;
            transform.position = mouseWorldPos + dragOffset;
        }
        private void OnMouseUp()
        {
            if (!isDragging) return;
            isDragging = false;
            UpdatePositionToData();
            if (spriteRenderer != null)
            {
                spriteRenderer.color = originalColor;
            }
            Debug.Log($"拖动结束: {cosmeticSO?.itemName}, 新位置: ({transform.position.x:F
2}, {transform.position.y:F2}), 旋转: {instanceData.rotation:F1}°");
        }
        #endregion
    }
}
using UnityEngine;
using System.Collections.Generic;
using System.Linq;
using MakeupPuzzle.Core;
```

```
namespace MakeupPuzzle.EditorLevel
{
    public class CosmeticGenerator : MonoBehaviour
    {
        [Header("生成参数")]
        [SerializeField] private float randomOffset = 2f;   // 随机偏移量（保留
        [Header("分层位置策略")]
        [SerializeField] private float centerClusterRadius = 2f;   // 底层聚
        [SerializeField] private float topLayerSpreadFactor = 1.0f; // 顶
        public List<CosmeticInstanceData> GenerateWithOrderBagTempSlots(
            List<OrderBagData> orderBags,
            float areaMinX,
            float areaMaxX,
            float areaMinY,
            float areaMaxY)
        {
            Debug.Log($"=== 开始生成化妆品（订单包级临时槽数算法）===");
            Debug.Log($"订单包数量: {orderBags.Count}");
            Debug.Log($"生成区域: X({areaMinX} ~ {areaMaxX}), Y({areaMinY} ~ {areaMaxY})");
            List<CosmeticInstanceData> instances = new List<CosmeticInstanceData>();
            HashSet<int> usedLayers = new HashSet<int>();
            OrderBagData trophyBag = null;
            int trophyBagIndex = -1;
            for (int i = 0; i < orderBags.Count; i++)
            {
                if (orderBags[i].configID == "OrderBagX1Config")
                {
                    trophyBag = orderBags[i];
                    trophyBagIndex = i;
                    break;
                }
            }
            if (trophyBag != null)
            {
                Debug.Log($"<color=yellow>检测到奖杯包，索引: {trophyBagIndex}</color>");
                foreach (var cosmeticID in trophyBag.requiredItems)
                {
                    var instance = CreateInstance(cosmeticID, 0, instances.Count, true); // isLo
ck=true
                    instances.Add(instance);
                    usedLayers.Add(0);
                    Debug.Log($"  - 🏆 奖杯化妆品 {cosmeticID}: 层级 0 (固定)");
                }
            }
            for (int bagIndex = orderBags.Count - 1; bagIndex >= 0; bagIndex--)
            {
                var bag = orderBags[bagIndex];
                if (bag.configID == "OrderBagX1Config")
                {
                    Debug.Log($"<color=yellow>跳过奖杯包（已处理）</color>");
                    continue;
                }
                int currentBagTempSlots = bag.tempSlotCount;
                int currentBagItemCount = bag.requiredItems.Count;
                int accumulatedCount = CalculateAccumulatedCount(orderBags, bagIndex + 1);
                int maxLayer = currentBagItemCount + currentBagTempSlots + accumulatedCo
unt;
                Debug.Log($"订单包 {bagIndex} ({bag.configID}): 需要 {currentBagItemCount}
个物品，临时槽数 {currentBagTempSlots}，层级范围 1~{maxLayer}");
                foreach (var cosmeticID in bag.requiredItems)
                {
                    int layer = GetRandomUnusedLayer(1, maxLayer, usedLayers);
                    if (layer == -1)
                    {
                        Debug.LogError($"无法分配层级！已用层级: {string.Join(",", usedLayer
s)}");
                        return null;
                    }
                    var instance = CreateInstance(cosmeticID, layer, instances.Count, false);
                    instances.Add(instance);
                    usedLayers.Add(layer);
                    Debug.Log($"   - {cosmeticID}: 层级 {layer}");
                }
            }
            int totalMaxLayer = usedLayers.Max();
            AssignRandomPositionsAndRotations(instances, totalMaxLayer, areaMinX, areaMax
X, areaMinY, areaMaxY);
            Debug.Log($"<color=green>✓ 成功生成 {instances.Count} 个化妆品，使用层级: {st
ring.Join(",", usedLayers.OrderBy(x => x))}</color>");
            return instances;
        }
        private int CalculateAccumulatedCount(List<OrderBagData> orderBags, int fromIndex, int
trophyBagIndex = -1)
        {
            int count = 0;
            for (int i = fromIndex; i < orderBags.Count; i++)
            {
                if (i == trophyBagIndex)
                    continue;
                count += orderBags[i].requiredItems.Count;
            }
            return count;
        }
        private int GetRandomUnusedLayer(int minLayer, int maxLayer, HashSet<int> usedLayer
s)
        {
            List<int> availableLayers = new List<int>();
            for (int layer = minLayer; layer <= maxLayer; layer++)
            {
                if (!usedLayers.Contains(layer))
                {
                    availableLayers.Add(layer);
                }
            }
            if (availableLayers.Count == 0)
            {
                return -1; // 没有可用层级
            }
            return availableLayers[Random.Range(0, availableLayers.Count)];
        }
        private CosmeticInstanceData CreateInstance(string cosmeticID, int layer, int instanceInd
ex, bool isLock)
        {
            return new CosmeticInstanceData
            {
                cosmeticID = cosmeticID,
                layer = layer,
                position = Vector3.zero, // 稍后赋值
                rotation = 0f, // 稍后赋值
                isLock = isLock,
                instanceID = System.Guid.NewGuid().ToString()
            };
        }
        private void AssignRandomPositionsAndRotations(
            List<CosmeticInstanceData> instances,
            int maxLayer,
            float areaMinX,
            float areaMaxX,
            float areaMinY,
            float areaMaxY)
        {
            float centerX = (areaMinX + areaMaxX) / 2f;
            float centerY = (areaMinY + areaMaxY) / 2f;
            float areaRadiusX = (areaMaxX - areaMinX) / 2f;
            float areaRadiusY = (areaMaxY - areaMinY) / 2f;
            foreach (var instance in instances)
            {
                float layerNormalized = (float)instance.layer / maxLayer;
                float spreadRadiusX = Mathf.Lerp(centerClusterRadius, areaRadiusX * topLayerSpreadFactor, layerNormalized);
                float spreadRadiusY = Mathf.Lerp(centerClusterRadius, areaRadiusY * topLayerSpreadFactor, layerNormalized);
                float angle = Random.Range(0f, 360f) * Mathf.Deg2Rad;
                float distance = Random.Range(0f, 1f); // 0~1 之间
                distance = Mathf.Sqrt(distance); // 开方使分布更均匀
                float offsetX = Mathf.Cos(angle) * distance * spreadRadiusX;
                float offsetY = Mathf.Sin(angle) * distance * spreadRadiusY;
                float finalX = centerX + offsetX;
                float finalY = centerY + offsetY;
                finalX = Mathf.Clamp(finalX, areaMinX, areaMaxX);
                finalY = Mathf.Clamp(finalY, areaMinY, areaMaxY);
                instance.position = new Vector3(finalX, finalY, 0);
                instance.rotation = Random.Range(0f, 360f);
                Debug.Log($"Layer {instance.layer}: Position=({finalX:F2}, {finalY:F2}), Rotation={instance.rotation:F1}°, SpreadRadius=({spreadRadiusX:F2}, {spreadRadiusY:F2})");
            }
        }
        public CosmeticInstanceData AddTrophyCosmetic(
            string trophyCosmeticID,
            List<CosmeticInstanceData> existingInstances,
            float centerX,
            float centerY)
        {
            int maxLayer = 0;
            foreach (var instance in existingInstances)
            {
                if (instance.layer > maxLayer)
                {
                    maxLayer = instance.layer;
                }
            }
            var trophyInstance = new CosmeticInstanceData
            {
                cosmeticID = trophyCosmeticID,
                layer = 0,
                position = new Vector3(centerX, centerY, 0),
                rotation = 0f, // 奖杯不旋转
                isLock = true,
                instanceID = System.Guid.NewGuid().ToString()
            };
            foreach (var instance in existingInstances)
            {
                instance.layer++;
            }
            Debug.Log($"<color=green>✓ 奖杯化妆品已添加到层级0，其他化妆品层级+1</color>");
            return trophyInstance;
        }
    }
}
using UnityEngine;
using TMPro;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class LevelEditorManager : MonoBehaviour
    {
        [Header("当前关卡")]
        [SerializeField] private LevelDataSO currentLevelData;
        [Header("子系统引用")]
        [SerializeField] private PrefabLibraryUI prefabLibrary;
        [SerializeField] private StackingAreaUI stackingArea;
        [SerializeField] private OrderAreaUI orderArea;
        [SerializeField] private LevelSettingsUI levelSettings;
        [Header("核心组件")]
        [SerializeField] public CosmeticGenerator cosmeticGenerator;
        [SerializeField] public OrderGenerator orderGenerator;
        [SerializeField] private LevelValidator levelValidator;
        [Header("资源缓存")]
        private CosmeticItemSO[] allCosmeticItems;
        private OrderBagConfigSO[] allOrderBagConfigs;
        private static LevelEditorManager instance;
        public static LevelEditorManager Instance => instance;
        private void Awake()
        {
            instance = this;
        }
        private void Start()
        {
            InitializeEditor();
        }
        private void InitializeEditor()
        {
            LoadAllResources();
            if (prefabLibrary != null)
                prefabLibrary.Initialize(this);
            if (stackingArea != null)
                stackingArea.Initialize(this);
            if (orderArea != null)
                orderArea.Initialize(this);
            if (levelSettings != null)
                levelSettings.Initialize(this);
            var algorithmPanel = FindObjectOfType<AlgorithmParaPanel>();
            if (algorithmPanel != null)
                algorithmPanel.Initialize(this);
            if (currentLevelData == null)
                CreateNewLevel();
        }
        private void LoadAllResources()
        {
            allCosmeticItems = Resources.LoadAll<CosmeticItemSO>("Cosmetics");
            if (allCosmeticItems.Length == 0)
            {
                Debug.LogError("没有找到任何CosmeticItemSO资源！请在Resources/Cosmetic
s/文件夹下创建化妆品数据。");
            }
            else
            {
                Debug.Log($"成功加载 {allCosmeticItems.Length} 个化妆品数据");
            }
            allOrderBagConfigs = Resources.LoadAll<OrderBagConfigSO>("OrderBags");
            if (allOrderBagConfigs.Length == 0)
            {
                Debug.LogError("没有找到任何OrderBagConfigSO资源！请在Resources/Order
Bags/文件夹下创建订单包配置。");
            }
            else
            {
                Debug.Log($"成功加载 {allOrderBagConfigs.Length} 个订单包配置");
            }
        }
        public void CreateNewLevel()
        {
            currentLevelData = ScriptableObject.CreateInstance<LevelDataSO>();
            currentLevelData.levelID = System.Guid.NewGuid().ToString();
            currentLevelData.levelName = "New Level";
            stackingArea.ClearAll();
            orderArea.ClearAll();
            Debug.Log("创建新关卡: " + currentLevelData.levelID);
        }
public void SaveLevel()
{
    if (currentLevelData == null)
    {
        Debug.LogError("没有要保存的关卡数据!");
        return;
    }
    Debug.Log("<color=cyan>========== 开始保存关卡（含预计算） ==========</color>");
    var cosmeticObjects = stackingArea.GetAllCosmeticObjects();   // 获取场
    var cosmeticInstances = stackingArea.GetAllInstances();         // 获取实例
    bool precomputeSuccess = LevelPrecomputeHelper.PrecomputeClickableStates(
        cosmeticObjects,
        cosmeticInstances
    );
    if (!precomputeSuccess)
    {
        Debug.LogError("<color=red>预计算失败！关卡保存已取消。</color>");
        #if UNITY_EDITOR
        UnityEditor.EditorUtility.DisplayDialog(
            "保存失败",
            "预计算遮挡关系失败！\n\n可能原因：\n1. 化妆品缺少CosmeticItem组件\n2. 化妆品
缺少Collider2D组件\n\n请检查后重试。",
            "确定");
        #endif
        return;
    }
    currentLevelData.cosmeticInstances = cosmeticInstances;
    currentLevelData.orderBags = orderArea.GetAllOrderBags();
    string fileName = currentLevelData.levelName.Replace(" ", "_");
    string path = $"Assets/Resources/LevelData/{fileName}.asset";
    if (!Directory.Exists("Assets/Resources/LevelData"))
    {
        Directory.CreateDirectory("Assets/Resources/LevelData");
    }
#if UNITY_EDITOR
    LevelDataSO existingAsset = UnityEditor.AssetDatabase.LoadAssetAtPath<LevelDataSO>(p
ath);
    if (existingAsset != null)
    {
        UnityEditor.EditorUtility.CopySerialized(currentLevelData, existingAsset);
        UnityEditor.EditorUtility.SetDirty(existingAsset);
        Debug.Log($"<color=green>✓ 关卡已更新（含预计算）: {path}</color>");
    }
    else
    {
        UnityEditor.AssetDatabase.CreateAsset(currentLevelData, path);
        Debug.Log($"<color=green>✓ 关卡已创建（含预计算）: {path}</color>");
    }
    UnityEditor.AssetDatabase.SaveAssets();
    UnityEditor.AssetDatabase.Refresh();
    Debug.Log($"<color=green>✓ 关卡保存成功！化妆品数: {cosmeticInstances.Count}, 订单包
数: {currentLevelData.orderBags.Count}</color>");
Debug.Log("<color=cyan>==========================================</color>");
#else
    Debug.LogError("保存关卡功能仅在编辑器模式下可用！");
#endif
}
        public void LoadLevel(string levelPath)
        {
            #if UNITY_EDITOR
            currentLevelData = UnityEditor.AssetDatabase.LoadAssetAtPath<LevelDataSO>(lev
elPath);
            #endif

            if (currentLevelData != null)
            {
stackingArea.LoadInstances(currentLevelData.cosmeticInstances);
                orderArea.LoadOrderBags(currentLevelData.orderBags);
                Debug.Log($"关卡已加载: {currentLevelData.levelName}");
            }
        }
        public void ValidateLevel()
        {
            if (levelValidator != null)
            {
                var instances = stackingArea.GetAllInstances();
                var orderBags = orderArea.GetAllOrderBags();
                bool isValid = levelValidator.Validate(instances, orderBags);
                currentLevelData.isValidated = isValid;
                currentLevelData.validationResult = levelValidator.GetValidationReport();
                levelSettings.ShowValidationResult(isValid, currentLevelData.validationResult);
            }
        }
        public LevelDataSO CurrentLevel => currentLevelData;
        public CosmeticItemSO[] GetAllCosmeticItems() => allCosmeticItems;
        public OrderBagConfigSO[] GetAllOrderBagConfigs() => allOrderBagConfigs;
        public CosmeticItemSO GetCosmeticItemByID(string id)
        {
            return allCosmeticItems?.FirstOrDefault(item => item.cosmeticID == id);
        }
        public OrderBagConfigSO GetOrderBagConfig(string configName)
        {
            return allOrderBagConfigs?.FirstOrDefault(config => config.configName == configNa
me);
        }
    }
}
using UnityEngine;
using System.Collections.Generic;
using MakeupPuzzle.Core;
#if UNITY_EDITOR
using UnityEditor;
#endif
namespace MakeupPuzzle.EditorLevel
{
    public static class LevelPrecomputeHelper
    {
        public static bool PrecomputeClickableStates(
            List<GameObject> cosmeticObjects,
            List<CosmeticInstanceData> instanceDataList)
        {
            if (cosmeticObjects == null || instanceDataList == null)
            {
                Debug.LogError("[PrecomputeHelper] 输入参数为空！");
                return false;
            }
            if (cosmeticObjects.Count != instanceDataList.Count)
            {
                Debug.LogError($"[PrecomputeHelper] 物体数量({cosmeticObjects.Count})与数
据数量({instanceDataList.Count})不匹配！");
                return false;
            }
            if (cosmeticObjects.Count == 0)
            {
                Debug.LogWarning("[PrecomputeHelper] 没有化妆品需要预计算");
                return true;
            }
            Debug.Log($"<color=cyan>========== 开始预计算 {cosmeticObjects.Count} 个化
妆品的遮挡状态 ==========</color>");

            #if UNITY_EDITOR
            EditorUtility.DisplayProgressBar("预计算遮挡关系", "正在准备数据...", 0f);
            #endif
            try
            {
                Physics2D.SyncTransforms();
                Debug.Log("[PrecomputeHelper] 物理系统已同步");
                List<ItemData> itemDataList = new List<ItemData>();
                for (int i = 0; i < cosmeticObjects.Count; i++)
                {
                    GameObject obj = cosmeticObjects[i];
                    CosmeticInstanceData instanceData = instanceDataList[i];
                    if (obj == null || instanceData == null)
                    {
                        Debug.LogWarning($"[PrecomputeHelper] 索引{i}的物体或数据为空，
跳过");
                        continue;
                    }
                    Collider2D collider = obj.GetComponent<Collider2D>();
                    if (collider == null)
                    {
                        Debug.LogWarning($"[PrecomputeHelper] {obj.name} 没有Collider2D
组件，默认可点击");
                        instanceData.precomputedClickable = true;
                        continue;
                    }
                    if (!collider.enabled)
                    {
                        Debug.LogWarning($"[PrecomputeHelper] {obj.name} 的Collider2D未
启用！");
                    }
                    SpriteRenderer sr = obj.GetComponent<SpriteRenderer>();
                    int sortingOrder = sr != null ? sr.sortingOrder : instanceData.layer;
                    ItemData itemData = new ItemData
                    {
                        gameObject = obj,
                        collider = collider,
                        sortingOrder = sortingOrder,
                        instanceData = instanceData,
                        position = obj.transform.position
                    };
                    itemDataList.Add(itemData);
                    Debug.Log($"[PrecomputeHelper] 物品 {i}: {obj.name}, SortingOrder={sorti
ngOrder}, Pos=({obj.transform.position.x:F2}, {obj.transform.position.y:F2}), Collider={collider.GetT
ype().Name}");
                }
                Debug.Log($"<color=green>[PrecomputeHelper] 成功构建 {itemDataList.Count}
个物品数据</color>");
                int clickableCount = 0;
                int blockedCount = 0;
                for (int i = 0; i < itemDataList.Count; i++)
                {
                    ItemData currentItem = itemDataList[i];
                    #if UNITY_EDITOR
                    float progress = (float)(i + 1) / itemDataList.Count;
                    EditorUtility.DisplayProgressBar(
                        "预计算遮挡关系",
                        $"正在分析 {i + 1}/{itemDataList.Count}: {currentItem.gameObject.nam
e}",
                        progress);
                    #endif
                    bool isBlocked = IsBlockedByOthers(currentItem, itemDataList);
                    currentItem.instanceData.precomputedClickable = !isBlocked;
                    if (!isBlocked)
                    {
                        clickableCount++;
                        Debug.Log($"<color=green>[PrecomputeHelper] ✓ {currentItem.game
Object.name} (Layer {currentItem.sortingOrder}): 可点击</color>");
                    }
                    else
                    {
                        blockedCount++;
                        Debug.Log($"<color=yellow>[PrecomputeHelper] ✗ {currentItem.game
Object.name} (Layer {currentItem.sortingOrder}): 被遮挡</color>");
                    }
                }
                Debug.Log($"<color=cyan>========== 预计算完成 ==========</color>");
                Debug.Log($"<color=green>✓ 可点击: {clickableCount} 个</color>");
                Debug.Log($"<color=yellow>✗ 被遮挡: {blockedCount} 个</color>");
                if (blockedCount == 0 && clickableCount > 10)
                {
                    Debug.LogWarning($"<color=orange>⚠ 警告：所有 {clickableCount} 个物
品都是可点击状态，这可能不正常！</color>");
                    Debug.LogWarning($"<color=orange>请检查：</color>");
                    Debug.LogWarning($"<color=orange>1. 物品是否正确设置了不同的 Sortin
gOrder？</color>");
                    Debug.LogWarning($"<color=orange>2. 物品之间是否有重叠？</color>");
                    Debug.LogWarning($"<color=orange>3. Collider2D 是否正确设置？</colo
r>");
                }
Debug.Log($"<color=cyan>================================</color>");
                return true;
            }
            catch (System.Exception e)
            {
                Debug.LogError($"[PrecomputeHelper] 预计算失败: {e.Message}\n{e.StackTrac
e}");
                return false;
            }
            finally
            {
                #if UNITY_EDITOR
                EditorUtility.ClearProgressBar();
                #endif
            }
        }
        private class ItemData
        {
            public GameObject gameObject;
            public Collider2D collider;
            public int sortingOrder;
            public CosmeticInstanceData instanceData;
            public Vector3 position;
        }
        private static bool IsBlockedByOthers(ItemData item, List<ItemData> allItems)
        {
            if (item == null || item.collider == null) return false;
            int higherItemsCount = 0;
            int testedCount = 0;
            foreach (var otherItem in allItems)
            {
                if (otherItem == item) continue;
                if (otherItem.sortingOrder <= item.sortingOrder) continue;
                higherItemsCount++;
                if (otherItem.collider == null) continue;
                testedCount++;
                bool isTouching = item.collider.IsTouching(otherItem.collider);
                if (isTouching)
                {
                    float distance = Vector3.Distance(item.position, otherItem.position);
                    Debug.Log($"<color=red>[遮挡检测] {item.gameObject.name} (Layer {item.s
ortingOrder}) 被 {otherItem.gameObject.name} (Layer {otherItem.sortingOrder}) 遮挡！距离={dist
ance:F2}</color>");
                    return true;
                }
            }
            Debug.Log($"[遮挡检测] {item.gameObject.name} (Layer {item.sortingOrder}): 检测
了 {testedCount}/{higherItemsCount} 个上层物品，未被遮挡");
            return false;
        }
        public static bool HasPrecomputedData(List<CosmeticInstanceData> instances)
        {
            if (instances == null || instances.Count == 0)
                return false;
            int trueCount = 0;
            int falseCount = 0;
            foreach (var instance in instances)
            {
                if (instance.precomputedClickable)
                    trueCount++;
                else
                    falseCount++;
            }
            if (falseCount == 0 && instances.Count > 5)
            {
                Debug.LogWarning($"<color=orange>[PrecomputeHelper] 所有 {instances.Cou
nt} 个物品都是可点击状态,可能未真正预计算,使用运行时检测</color>");
                return false;
            }
            Debug.Log($"<color=green>[PrecomputeHelper] ✓ 发现预计算数据: 可点击={trueC
ount}, 被遮挡={falseCount}</color>");
            return true;
        }
    }
}
using UnityEngine;
using System.Collections.Generic;
using System.Linq;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class LevelValidator : MonoBehaviour
    {
        private const int MAX_TEMP_SLOTS = 5;
        private const int MAX_ACTIVE_ORDERS = 2;
        private List<string> tempSlots;
        private List<string> validationLog;
        public bool Validate(List<CosmeticInstanceData> instances, List<OrderBagData> orderB
ags)
        {
            tempSlots = new List<string>();
            validationLog = new List<string>();
            var workingInstances = instances.Select(i => CloneInstance(i)).ToList();
            var workingOrders = orderBags.Select(o => CloneOrder(o)).ToList();
            int currentOrderIndex = 0;
            List<OrderBagData> activeOrders = new List<OrderBagData>();
            for (int i = 0; i < MAX_ACTIVE_ORDERS && i < workingOrders.Count; i++)
            {
                activeOrders.Add(workingOrders[currentOrderIndex++]);
            }
            while (workingInstances.Count > 0)
            {
                var topItem = GetTopAccessibleItem(workingInstances);
                if (topItem == null)
                {
                    validationLog.Add("ERROR: No accessible items but level not complete!");
                    return false;
                }
                if (topItem.isLock)
                {
                    if (workingInstances.Count == 1)
                    {
                        validationLog.Add("SUCCESS: Trophy collected, level complete!");
                        return true;
                    }
                    else
                    {
                        validationLog.Add("ERROR: Trophy is accessible but other items remai
n!");
                        return false;
                    }
                }
                bool matched = false;
                foreach (var order in activeOrders)
                {
                    if (order.requiredItems.Contains(topItem.cosmeticID))
                    {
                        order.requiredItems.Remove(topItem.cosmeticID);
                        workingInstances.Remove(topItem);
                        matched = true;
                        validationLog.Add($"Matched {topItem.cosmeticID} to order");
                        if (order.requiredItems.Count == 0)
                        {
                            activeOrders.Remove(order);
                            validationLog.Add($"Order completed");
                            if (currentOrderIndex < workingOrders.Count)
                            {
activeOrders.Add(workingOrders[currentOrderIndex++]);
                            }
                            MatchFromTempSlots(activeOrders);
                        }
                        break;
                    }
                }
                if (!matched)
                {
                    if (tempSlots.Count >= MAX_TEMP_SLOTS)
                    {
                        validationLog.Add($"ERROR: Temp slots full! Cannot place {topItem.co
smeticID}");
                        return false;
                    }
                    tempSlots.Add(topItem.cosmeticID);
                    workingInstances.Remove(topItem);
                    validationLog.Add($"Placed {topItem.cosmeticID} in temp slot ({tempSlots.C
ount}/{MAX_TEMP_SLOTS})");
                }
            }
            return true;
        }
        private CosmeticInstanceData GetTopAccessibleItem(List<CosmeticInstanceData> instan
ces)
        {
            var sorted = instances.OrderByDescending(i => i.layer).ToList();
            foreach (var item in sorted)
            {
                if (IsAccessible(item, instances))
                {
                    return item;
                }
            }
            return null;
        }
        private bool IsAccessible(CosmeticInstanceData item, List<CosmeticInstanceData> allIte
ms)
        {
            foreach (var other in allItems)
            {
                if (other == item) continue;
                if (other.layer <= item.layer) continue;
                float distance = Vector2.Distance(item.position, other.position);
                if (distance < 50f) // 假设50单位内算遮挡
                {
                    return false;
                }
            }
            return true;
        }
        private void MatchFromTempSlots(List<OrderBagData> activeOrders)
        {
            List<string> matched = new List<string>();
            foreach (var order in activeOrders)
            {
                foreach (var tempItem in tempSlots)
                {
                    if (order.requiredItems.Contains(tempItem))
                    {
                        order.requiredItems.Remove(tempItem);
                        matched.Add(tempItem);
                        validationLog.Add($"Matched {tempItem} from temp slot to order");
                    }
                }
            }
            foreach (var item in matched)
            {
                tempSlots.Remove(item);
            }
        }
        private CosmeticInstanceData CloneInstance(CosmeticInstanceData original)
        {
            var clone = new CosmeticInstanceData();
            clone.cosmeticID = original.cosmeticID;
            clone.isLock = original.isLock;
            clone.layer = original.layer;
            clone.position = original.position;
            clone.rotation = original.rotation;
            return clone;
        }
        private OrderBagData CloneOrder(OrderBagData original)
        {
            var clone = new OrderBagData();
            clone.configID = original.configID;
            clone.requiredItems = new List<string>(original.requiredItems);
            clone.orderIndex = original.orderIndex;
            clone.difficulty = original.difficulty;
            return clone;
        }
        public string GetValidationReport()
        {
            return string.Join("\n", validationLog);
        }
    }
```

```
}
using UnityEngine;
using System.Collections.Generic;
using System.Linq;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class OrderGenerator : MonoBehaviour
    {
        public List<OrderBagData> GenerateOrderBags(
            int x2Count,
            int x3Count,
            int x4Count,
            int x5Count,
            int typeCount,
            LevelType levelType)
        {
            Debug.Log($"=== 开始生成订单包 ===");
            Debug.Log($"配置: X2={x2Count}, X3={x3Count}, X4={x4Count}, X5={x5Count}");
            Debug.Log($"化妆品类型数: {typeCount}, 关卡类型: {levelType}");
            var bagCombination = CalculateBagCombination(x2Count, x3Count, x4Count, x5Cou
nt);
            if (bagCombination == null || bagCombination.Count == 0)
            {
                Debug.LogError("订单包配置无效！");
                return null;
            }
            var allCosmetics = GetAllCosmetics();
            if (allCosmetics == null || allCosmetics.Length == 0)
            {
                Debug.LogError("没有可用的化妆品！");
                return null;
            }
            List<OrderBagData> orderBags = null;
            switch (levelType)
            {
                case LevelType.Normal:
                    orderBags = GenerateNormalLevelBags(bagCombination, allCosmetics, typ
eCount);
                    break;
                case LevelType.BrandPerBag:
                    orderBags = GenerateBrandPerBagLevelBags(bagCombination, allCosmeti
cs, typeCount);
                    break;
                case LevelType.BrandGlobal:
                    orderBags = GenerateBrandGlobalLevelBags(bagCombination, allCosmetic
s);
                    break;
                case LevelType.ColorPerBag:
                    orderBags = GenerateColorPerBagLevelBags(bagCombination, allCosmetic
s, typeCount);
                    break;
                case LevelType.ColorGlobal:
                    orderBags = GenerateColorGlobalLevelBags(bagCombination, allCosmetic
s);
                    break;
                case LevelType.TypePerBag:
                    orderBags = GenerateTypePerBagLevelBags(bagCombination, allCosmetic
s, typeCount);
                    break;
                case LevelType.TypeGlobal:
                    orderBags = GenerateTypeGlobalLevelBags(bagCombination, allCosmetic
s);
                    break;
            }
            if (orderBags == null || orderBags.Count == 0)
            {
                Debug.LogError("订单包生成失败！");
                return null;
            }
            orderBags = ShuffleAndReindexOrderBags(orderBags);
            Debug.Log($"<color=green>✓ 成功生成 {orderBags.Count} 个订单包（已随机打乱
顺序）</color>");
            PrintOrderBagsInfo(orderBags);
            return orderBags;
        }
        private List<OrderBagData> ShuffleAndReindexOrderBags(List<OrderBagData> orderBa
gs)
        {
            Debug.Log($"<color=yellow>开始随机打乱 {orderBags.Count} 个订单包...</color>");
            string beforeOrder = string.Join(", ", orderBags.Select(b => b.configID));
            Debug.Log($"打乱前顺序: [{beforeOrder}]");
            for (int i = orderBags.Count - 1; i > 0; i--)
            {
                int j = Random.Range(0, i + 1);
                var temp = orderBags[i];
                orderBags[i] = orderBags[j];
                orderBags[j] = temp;
            }
            for (int i = 0; i < orderBags.Count; i++)
            {
                orderBags[i].orderIndex = i;
            }
            string afterOrder = string.Join(", ", orderBags.Select(b => b.configID));
            Debug.Log($"打乱后顺序: [{afterOrder}]");
            Debug.Log($"<color=green>✓ 订单包已随机打乱并重新编号（0 ~ {orderBags.Count
- 1}）</color>");
            return orderBags;
        }
        private Dictionary<OrderBagConfigSO, int> CalculateBagCombination(
            int x2Count, int x3Count, int x4Count, int x5Count)
        {
            var combination = new Dictionary<OrderBagConfigSO, int>();
            var allConfigs = Resources.LoadAll<OrderBagConfigSO>("OrderBags");
            if (allConfigs.Length == 0)
            {
                Debug.LogError("没有找到OrderBagConfigSO配置！");
                return null;
            }
            var x2Config = System.Array.Find(allConfigs, c => c.slotCount == 2);
            var x3Config = System.Array.Find(allConfigs, c => c.slotCount == 3);
            var x4Config = System.Array.Find(allConfigs, c => c.slotCount == 4);
            var x5Config = System.Array.Find(allConfigs, c => c.slotCount == 5);
            if (x2Count > 0 && x2Config != null) combination[x2Config] = x2Count;
            if (x3Count > 0 && x3Config != null) combination[x3Config] = x3Count;
            if (x4Count > 0 && x4Config != null) combination[x4Config] = x4Count;
            if (x5Count > 0 && x5Config != null) combination[x5Config] = x5Count;
            return combination;
        }
        private CosmeticItemSO[] GetAllCosmetics()
        {
            return Resources.LoadAll<CosmeticItemSO>("Cosmetics");
```

```
        }
        private List<OrderBagData> GenerateNormalLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics,
            int typeCount)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var selectedCosmetics = SelectRandomCosmetics(allCosmetics, typeCount);
            int cosmeticIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    var selectedCosmetic = selectedCosmetics[cosmeticIndex % selectedCosm
etics.Count];
                    cosmeticIndex++;
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.Normal;
                    bag.tempSlotCount = 4;
                    for (int slot = 0; slot < config.slotCount; slot++)
                    {
                        bag.requiredItems.Add(selectedCosmetic.cosmeticID);
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<OrderBagData> GenerateBrandPerBagLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics,
            int typeCount)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var brandGroups = allCosmetics
                .Where(c => !string.IsNullOrEmpty(c.brand))
                .GroupBy(c => c.brand)
                .ToList();
            if (brandGroups.Count == 0)
            {
                Debug.LogError("没有找到带品牌信息的化妆品！");
                return GenerateNormalLevelBags(bagCombination, allCosmetics, typeCount);
            }
            var selectedBrands = brandGroups.OrderBy(x => Random.value).Take(typeCount).T
oList();
            int brandIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    var selectedBrand = selectedBrands[brandIndex % selectedBrands.Count];
                    brandIndex++;
                    var brandCosmetics = selectedBrand.ToList();
                    var selectedItems = SelectCosmeticsWithRepeat(brandCosmetics, config.sl
otCount);
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.BrandPerBag;
                    bag.brandName = selectedBrand.Key;
                    bag.tempSlotCount = 4;
                    foreach (var cosmetic in selectedItems)
                    {
                        bag.requiredItems.Add(cosmetic.cosmeticID);
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<OrderBagData> GenerateBrandGlobalLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var brandGroups = allCosmetics
                .Where(c => !string.IsNullOrEmpty(c.brand))
                .GroupBy(c => c.brand)
                .ToList();
            if (brandGroups.Count == 0)
            {
                Debug.LogError("没有找到带品牌信息的化妆品！");
                return null;
            }
            var selectedBrand = brandGroups[Random.Range(0, brandGroups.Count)];
            var brandCosmetics = selectedBrand.ToList();
            Debug.Log($"全关卡品牌关：选择品牌 [{selectedBrand.Key}]，共 {brandCosmetics.
Count} 个化妆品");
            int cosmeticIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.BrandGlobal;
                    bag.brandName = selectedBrand.Key;
                    bag.tempSlotCount = 4;
                    for (int slot = 0; slot < config.slotCount; slot++)
                    {
                        var cosmetic = brandCosmetics[cosmeticIndex % brandCosmetics.Cou
nt];

                        bag.requiredItems.Add(cosmetic.cosmeticID);
                        cosmeticIndex++;
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<OrderBagData> GenerateColorPerBagLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics,
            int typeCount)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var colorGroups = allCosmetics
                .Where(c => !string.IsNullOrEmpty(c.colorSeries))
                .GroupBy(c => c.colorSeries)
                .ToList();
            if (colorGroups.Count == 0)
            {
                Debug.LogError("没有找到带颜色信息的化妆品！");
                return GenerateNormalLevelBags(bagCombination, allCosmetics, typeCount);
            }
            var selectedColors = colorGroups.OrderBy(x => Random.value).Take(typeCount).To
List();
            int colorIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    var selectedColor = selectedColors[colorIndex % selectedColors.Count];
                    colorIndex++;
                    var colorCosmetics = selectedColor.ToList();
                    var selectedItems = SelectCosmeticsWithRepeat(colorCosmetics, config.slo
tCount);
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.ColorPerBag;
                    bag.colorSeries = selectedColor.Key;
                    bag.tempSlotCount = 4;
                    foreach (var cosmetic in selectedItems)
                    {
                        bag.requiredItems.Add(cosmetic.cosmeticID);
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<OrderBagData> GenerateColorGlobalLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var colorGroups = allCosmetics
                .Where(c => !string.IsNullOrEmpty(c.colorSeries))
                .GroupBy(c => c.colorSeries)
                .ToList();
            if (colorGroups.Count == 0)
            {
                Debug.LogError("没有找到带颜色信息的化妆品！");
                return null;
            }
            var selectedColor = colorGroups[Random.Range(0, colorGroups.Count)];
            var colorCosmetics = selectedColor.ToList();
            Debug.Log($"全关卡颜色关：选择颜色 [{selectedColor.Key}]，共 {colorCosmetics.C
ount} 个化妆品");
            int cosmeticIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.ColorGlobal;
                    bag.colorSeries = selectedColor.Key;
                    bag.tempSlotCount = 4;
                    for (int slot = 0; slot < config.slotCount; slot++)
                    {
                        var cosmetic = colorCosmetics[cosmeticIndex % colorCosmetics.Coun
t];
                        bag.requiredItems.Add(cosmetic.cosmeticID);
                        cosmeticIndex++;
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<OrderBagData> GenerateTypePerBagLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics,
            int typeCount)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var typeGroups = allCosmetics
                .Where(c => !string.IsNullOrEmpty(c.type))
                .GroupBy(c => c.type)
                .ToList();
            if (typeGroups.Count == 0)
            {
                Debug.LogError("没有找到带类型信息的化妆品！");
                return GenerateNormalLevelBags(bagCombination, allCosmetics, typeCount);
            }
            var selectedTypes = typeGroups.OrderBy(x => Random.value).Take(typeCount).ToLi
st();
            int typeIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    var selectedType = selectedTypes[typeIndex % selectedTypes.Count];
                    typeIndex++;
                    var typeCosmetics = selectedType.ToList();
                    var selectedItems = SelectCosmeticsWithRepeat(typeCosmetics, config.slot
Count);
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.TypePerBag;
                    bag.typeName = selectedType.Key;
                    bag.tempSlotCount = 4;
                    foreach (var cosmetic in selectedItems)
                    {
                        bag.requiredItems.Add(cosmetic.cosmeticID);
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<OrderBagData> GenerateTypeGlobalLevelBags(
            Dictionary<OrderBagConfigSO, int> bagCombination,
            CosmeticItemSO[] allCosmetics)
        {
            List<OrderBagData> orderBags = new List<OrderBagData>();
            int orderIndex = 0;
            var typeGroups = allCosmetics
                .Where(c => !string.IsNullOrEmpty(c.type))
                .GroupBy(c => c.type)
                .ToList();
            if (typeGroups.Count == 0)
            {
                Debug.LogError("没有找到带类型信息的化妆品！");
                return null;
            }
            var selectedType = typeGroups[Random.Range(0, typeGroups.Count)];
            var typeCosmetics = selectedType.ToList();
            Debug.Log($"全关卡类型关：选择类型 [{selectedType.Key}]，共 {typeCosmetics.C
ount} 个化妆品");
            int cosmeticIndex = 0;
            foreach (var kvp in bagCombination)
            {
                var config = kvp.Key;
                int bagCount = kvp.Value;
                for (int i = 0; i < bagCount; i++)
                {
                    OrderBagData bag = new OrderBagData();
                    bag.configID = config.configName;
                    bag.orderIndex = orderIndex++;
                    bag.requiredItems = new List<string>();
                    bag.bagType = OrderBagType.TypeGlobal;
                    bag.typeName = selectedType.Key;
                    bag.tempSlotCount = 4;
                    for (int slot = 0; slot < config.slotCount; slot++)
                    {
                        var cosmetic = typeCosmetics[cosmeticIndex % typeCosmetics.Count];
                        bag.requiredItems.Add(cosmetic.cosmeticID);
                        cosmeticIndex++;
                    }
                    orderBags.Add(bag);
                }
            }
            return orderBags;
        }
        private List<CosmeticItemSO> SelectRandomCosmetics(CosmeticItemSO[] allCosmetics,
int count)
        {
            if (count <= allCosmetics.Length)
            {
                return allCosmetics.OrderBy(x => Random.value).Take(count).ToList();
            }
            else
            {
                var result = allCosmetics.ToList();
                int needMore = count - allCosmetics.Length;
                for (int i = 0; i < needMore; i++)
                {
                    result.Add(allCosmetics[Random.Range(0, allCosmetics.Length)]);
                }
                return result.OrderBy(x => Random.value).ToList();
            }
        }
        private List<CosmeticItemSO> SelectCosmeticsWithRepeat(List<CosmeticItemSO> cos
metics, int count)
        {
            List<CosmeticItemSO> result = new List<CosmeticItemSO>();
            if (cosmetics.Count == 0)
            {
                Debug.LogError("化妆品列表为空！");
                return result;
            }
            var shuffled = cosmetics.OrderBy(x => Random.value).ToList();
            for (int i = 0; i < count; i++)
            {
                result.Add(shuffled[i % shuffled.Count]);
            }
            return result;
        }
        private void PrintOrderBagsInfo(List<OrderBagData> orderBags)
        {
            Debug.Log("========== 订单包详细信息（随机打乱后）==========");
            foreach (var bag in orderBags)
            {
                string typeInfo = "";
                switch (bag.bagType)
                {
                    case OrderBagType.Normal:
                        typeInfo = "[普通包]";
                        break;
                    case OrderBagType.BrandPerBag:
                        typeInfo = $"[每个包品牌: {bag.brandName}]";
                        break;
                    case OrderBagType.BrandGlobal:
                        typeInfo = $"[全关卡品牌: {bag.brandName}]";
                        break;
                    case OrderBagType.ColorPerBag:
                        typeInfo = $"[每个包颜色: {bag.colorSeries}]";
                        break;
                    case OrderBagType.ColorGlobal:
                        typeInfo = $"[全关卡颜色: {bag.colorSeries}]";
                        break;
                    case OrderBagType.TypePerBag:
                        typeInfo = $"[每个包类型: {bag.typeName}]";
                        break;
                    case OrderBagType.TypeGlobal:
                        typeInfo = $"[全关卡类型: {bag.typeName}]";
                        break;
                }
                string itemsInfo = string.Join(", ", bag.requiredItems);
                Debug.Log($"订单包 {bag.orderIndex}: {bag.configID} {typeInfo} - [{itemsInfo}]");
            }
            Debug.Log("================================================");
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class AlgorithmParaPanel : MonoBehaviour
    {
        [Header("UI引用")]
        [SerializeField] private Slider deepSlider;                    // 依赖深度
        [SerializeField] private TMP_Text deepValueText;               // 依赖深度
        [SerializeField] private Slider densitySlider;                 // 聚簇密度
        [SerializeField] private TMP_Text densityValueText;            // 聚簇密
        [SerializeField] private TMP_InputField seedInput;             // 随机种子
        [Header("按钮")]
        [SerializeField] private Button confirmButton;                 // 应用按钮
        [SerializeField] private Button resetButton;                   // 重置按钮
        [SerializeField] private Button cancelButton;                  // 取消按钮
        private LevelEditorManager manager;
        private const int DEFAULT_DEPENDENCY_DEPTH = 0;
        private const float DEFAULT_CLUSTER_DENSITY = 0.5f;
        private void Awake()
        {
            if (confirmButton != null)
                confirmButton.onClick.AddListener(OnConfirm);
            if (resetButton != null)
                resetButton.onClick.AddListener(OnReset);
            if (cancelButton != null)
                cancelButton.onClick.AddListener(OnCancel);
            if (deepSlider != null)
            {
                deepSlider.minValue = 0;
                deepSlider.maxValue = 3;
                deepSlider.wholeNumbers = true;
                deepSlider.onValueChanged.AddListener(OnDeepSliderChanged);
            }
            if (densitySlider != null)
            {
                densitySlider.minValue = 0f;
                densitySlider.maxValue = 1f;
densitySlider.onValueChanged.AddListener(OnDensitySliderChanged);
            }
            gameObject.SetActive(false);
        }
        public void Initialize(LevelEditorManager manager)
        {
            this.manager = manager;
        }
        public void Open()
        {
            gameObject.SetActive(true);
            LoadCurrentParameters();
        }
        private void LoadCurrentParameters()
        {
            if (manager == null || manager.CurrentLevel == null)
            {
                SetDefaultValues();
                return;
            }
            var levelData = manager.CurrentLevel;
            if (deepSlider != null)
            {
                deepSlider.value = levelData.dependencyDepth;
            }
            if (densitySlider != null)
            {
                densitySlider.value = levelData.clusterDensity;
            }
            if (seedInput != null)
            {
                seedInput.text = levelData.randomSeed.ToString();
            }
            UpdateValueTexts();
        }
        private void SetDefaultValues()
        {
            if (deepSlider != null)
                deepSlider.value = DEFAULT_DEPENDENCY_DEPTH;
            if (densitySlider != null)
                densitySlider.value = DEFAULT_CLUSTER_DENSITY;
            if (seedInput != null)
                seedInput.text = "0";
            UpdateValueTexts();
        }
        private void OnDeepSliderChanged(float value)
        {
            if (deepValueText != null)
            {
                deepValueText.text = Mathf.RoundToInt(value).ToString();
            }
        }
        private void OnDensitySliderChanged(float value)
        {
            if (densityValueText != null)
            {
                densityValueText.text = value.ToString("F2");
            }
        }
        private void UpdateValueTexts()
        {
            if (deepSlider != null && deepValueText != null)
            {
                deepValueText.text = Mathf.RoundToInt(deepSlider.value).ToString();
            }
            if (densitySlider != null && densityValueText != null)
            {
                densityValueText.text = densitySlider.value.ToString("F2");
            }
        }
        private void OnConfirm()
        {
            if (manager == null || manager.CurrentLevel == null)
            {
                Debug.LogError("没有可用的关卡数据！");
                return;
            }
            var levelData = manager.CurrentLevel;
            if (deepSlider != null)
            {
                levelData.dependencyDepth = Mathf.RoundToInt(deepSlider.value);
            }
            if (densitySlider != null)
            {
                levelData.clusterDensity = densitySlider.value;
            }
            if (seedInput != null && int.TryParse(seedInput.text, out int seed))
            {
                levelData.randomSeed = seed;
            }
            else
            {
                levelData.randomSeed = 0; // 默认使用时间种子
            }
            Debug.Log($"<color=green>✓ 算法参数已保存！</color>");
            Debug.Log($"依赖深度: {levelData.dependencyDepth}");
            Debug.Log($"聚簇密度: {levelData.clusterDensity}");
            Debug.Log($"随机种子: {levelData.randomSeed}");
            Close();
        }
        private void OnReset()
        {
            SetDefaultValues();
            Debug.Log("算法参数已重置为默认值");
        }
        private void OnCancel()
        {
            Close();
        }
        private void Close()
        {
            gameObject.SetActive(false);
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using TMPro;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class CosmeticLibraryItem : MonoBehaviour, IBeginDragHandler, IDragHandler, IEnd
DragHandler
    {
        [Header("UI组件")]
        [SerializeField] private Image icon;
        [SerializeField] private TMP_Text nameText;
        private CosmeticItemSO cosmeticSO;   // 完整的化妆品SO
        private StackingAreaUI stackingArea;
        private Camera mainCamera;
        private bool isDragging = false;
        public CosmeticItemSO CosmeticData => cosmeticSO;
        private void Awake()
        {
            mainCamera = Camera.main;
        }
        public void Initialize(CosmeticItemSO cosmetic, StackingAreaUI area)
        {
            SetData(cosmetic, area);
        }
        public void SetData(CosmeticItemSO cosmetic, StackingAreaUI area)
        {
            cosmeticSO = cosmetic;
            stackingArea = area;

            if (cosmetic != null)
            {
                if (icon != null && cosmetic.icon != null)
                {
                    icon.sprite = cosmetic.icon;
                    icon.color = Color.white;
                }
                if (nameText != null)
                {
                    nameText.text = cosmetic.itemName;
                }
                gameObject.name = $"LibraryItem_{cosmetic.itemName}";
            }
        }
        public void OnBeginDrag(PointerEventData eventData)
        {
            if (cosmeticSO == null || stackingArea == null) return;
            isDragging = true;
            Debug.Log($"开始从库中拖动: {cosmeticSO.itemName}");
        }
        public void OnDrag(PointerEventData eventData)
        {
            if (!isDragging || mainCamera == null || cosmeticSO == null) return;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(eventData.position);
            mouseWorldPos.z = 0;
            if (stackingArea != null)
            {
                stackingArea.ShowDragPreview(cosmeticSO, mouseWorldPos);
            }
        }
        public void OnEndDrag(PointerEventData eventData)
        {
            if (!isDragging) return;
            isDragging = false;
            if (stackingArea != null)
            {
                stackingArea.HideDragPreview();
            }
            if (mainCamera != null && cosmeticSO != null)
            {
                Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(eventData.positio
n);

                mouseWorldPos.z = 0;
                if (stackingArea != null && IsInStackingArea(mouseWorldPos))
                {
                    stackingArea.CreateCosmeticFromLibrary(cosmeticSO, mouseWorldPos);
                    Debug.Log($"✓ 在堆叠区创建化妆品: {cosmeticSO.itemName}");
                }
                else
                {
                    Debug.Log("拖放位置不在堆叠区内");
                }
            }
        }
        private bool IsInStackingArea(Vector3 worldPos)
        {
            return worldPos.x >= -8f && worldPos.x <= 8f &&
                   worldPos.y >= -4f && worldPos.y <= 4f;
        }
        public CosmeticItemSO GetCosmeticSO()
        {
            return cosmeticSO;
        }
    }
}
using UnityEngine;
using UnityEngine.EventSystems;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class DraggableItem : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragH
andler
    {
        private CosmeticInstanceData instanceData;
        private CosmeticItemSO cosmeticSO;
        private StackingAreaUI stackingArea;
        private bool isDragging = false;
        private Vector3 dragOffset;
        private Camera mainCamera;
        private SpriteRenderer spriteRenderer;
        private Color originalColor;
        private void Awake()
        {
            mainCamera = Camera.main;
            spriteRenderer = GetComponent<SpriteRenderer>();
            if (spriteRenderer != null)
            {
                originalColor = spriteRenderer.color;
            }
        }
        public void Initialize(CosmeticInstanceData data, CosmeticItemSO so, StackingAreaUI ar
ea)
        {
            instanceData = data;
            cosmeticSO = so;
            stackingArea = area;
        }
        public void OnBeginDrag(PointerEventData eventData)
        {
            if (mainCamera == null) return;
            isDragging = true;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(eventData.position);
            mouseWorldPos.z = 0;
            dragOffset = transform.position - mouseWorldPos;
            if (spriteRenderer != null)
            {
                spriteRenderer.color = Color.yellow;
            }
            if (stackingArea != null)
            {
                stackingArea.SetItemDragging(true);
            }
            Debug.Log($"开始拖动: {cosmeticSO?.itemName ?? "Unknown"}");
        }
        public void OnDrag(PointerEventData eventData)
        {
            if (!isDragging || mainCamera == null) return;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(eventData.position);
            mouseWorldPos.z = 0;
            transform.position = mouseWorldPos + dragOffset;
        }
        public void OnEndDrag(PointerEventData eventData)
        {
            if (!isDragging) return;
            isDragging = false;
            if (instanceData != null)
            {
                instanceData.position = transform.position;
                if (stackingArea != null)
                {
                    stackingArea.UpdateInstanceData(instanceData);
                    stackingArea.SetItemDragging(false);
                }
            }
            if (spriteRenderer != null)
            {
                spriteRenderer.color = originalColor;
            }
            Debug.Log($"拖动结束: {cosmeticSO?.itemName ?? "Unknown"}, 新位置: ({transfor
m.position.x:F2}, {transform.position.y:F2})");
        }
        public CosmeticInstanceData GetInstanceData()
        {
            return instanceData;
        }
        public void UpdatePositionToData()
        {
            if (instanceData != null)
            {
                instanceData.position = transform.position;
            }
        }
        public void SetClickable(bool clickable)
        {
            if (spriteRenderer != null)
            {
                if (clickable)
                {
                    spriteRenderer.color = originalColor;
                }
                else
                {
                    spriteRenderer.color = originalColor * 0.6f;
                }
            }
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.IO;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class LevelSettingsUI : MonoBehaviour
    {
        [Header("UI按钮")]
        [SerializeField] private Button newLevelButton;
        [SerializeField] private Button saveLevelButton;
        [SerializeField] private Button loadLevelButton;
        [SerializeField] private Button playtestButton;
        [Header("文件操作UI")]
        [SerializeField] private TMP_InputField fileNameInput;
        [Header("关卡类型")]
        [SerializeField] private TMP_Dropdown levelTypeDropdown;   // 🆕 关
        private LevelEditorManager manager;
        public void Initialize(LevelEditorManager manager)
        {
            this.manager = manager;
            SetupButtonListeners();
            SetupLevelTypeDropdown();
            if (fileNameInput != null)
            {
                fileNameInput.text = "NewLevel_" + System.DateTime.Now.ToString("yyyyMMdd
_HHmmss");
            }
        }
        private void SetupButtonListeners()
        {
            if (newLevelButton != null)
                newLevelButton.onClick.AddListener(OnNewLevel);
            if (saveLevelButton != null)
                saveLevelButton.onClick.AddListener(OnSaveLevel);
            if (loadLevelButton != null)
                loadLevelButton.onClick.AddListener(OnLoadLevel);
            if (playtestButton != null)
                playtestButton.onClick.AddListener(OnPlaytest);
        }
        private void SetupLevelTypeDropdown()
        {
            if (levelTypeDropdown == null)
            {
                Debug.LogWarning("LevelTypeDropdown 未赋值！");
                return;
            }
            levelTypeDropdown.ClearOptions();
            var options = new System.Collections.Generic.List<string>
            {
                "普通关",
                "品牌关",
                "颜色关",
                "类型关"
            };
            levelTypeDropdown.AddOptions(options);
            levelTypeDropdown.value = 0;
levelTypeDropdown.onValueChanged.AddListener(OnLevelTypeChanged);
        }
        private void OnLevelTypeChanged(int index)
        {
            if (manager.CurrentLevel == null) return;
            LevelType newType = (LevelType)index;
            manager.CurrentLevel.levelType = newType;
            Debug.Log($"关卡类型已改变为: {newType}");
        }
        private void OnNewLevel()
        {
            manager.CreateNewLevel();
            if (fileNameInput != null)
            {
                fileNameInput.text = "NewLevel_" + System.DateTime.Now.ToString("yyyyMMdd
_HHmmss");
            }
            if (levelTypeDropdown != null)
            {
                levelTypeDropdown.value = 0;
            }
            Debug.Log("创建新关卡");
        }
        private void OnSaveLevel()
        {
            if (fileNameInput != null && !string.IsNullOrEmpty(fileNameInput.text))
            {
                string fileName = fileNameInput.text;
                foreach (char c in Path.GetInvalidFileNameChars())
                {
                    fileName = fileName.Replace(c, '_');
                }
                if (manager.CurrentLevel != null)
                {
                    manager.CurrentLevel.levelName = fileName;
                }
                manager.SaveLevel();
                PrintLevelStatistics();
                Debug.Log($"关卡保存为: {fileName}");
            }
            else
            {
                Debug.LogError("请输入文件名！");
            }
        }
        private void OnLoadLevel()
        {
            #if UNITY_EDITOR
            string path = UnityEditor.EditorUtility.OpenFilePanel(
                "选择关卡文件",
                "Assets/Resources/LevelData",
                "asset"
            );
            if (!string.IsNullOrEmpty(path))
            {
                if (path.StartsWith(Application.dataPath))
                {
                    path = "Assets" + path.Substring(Application.dataPath.Length);
                }
                manager.LoadLevel(path);

                if (fileNameInput != null)
                {
                    string fileName = Path.GetFileNameWithoutExtension(path);
                    fileNameInput.text = fileName;
                }
                if (levelTypeDropdown != null && manager.CurrentLevel != null)
                {
                    levelTypeDropdown.value = (int)manager.CurrentLevel.levelType;
                }
                PrintLevelStatistics();
                Debug.Log($"关卡加载自: {path}");
            }
            #else
            Debug.LogError("文件选择对话框只在Unity编辑器中可用！");
            #endif
        }
        private void OnPlaytest()
        {
            Debug.Log("========== 开始关卡验证 ==========");
            manager.ValidateLevel();
        }
        public void ShowValidationResult(bool isValid, string report)
        {
            if (isValid)
            {
                Debug.Log("<color=green>✓ 关卡验证通过！关卡可以正常通关。</color>");
            }
            else
            {
                Debug.LogError("<color=red>✗ 关卡验证失败！关卡无法通关。</color>");
            }
            Debug.Log("========== 验证详细报告 ==========");
            Debug.Log(report);
            Debug.Log("========== 验证结束 ==========");
        }
        private void PrintLevelStatistics()
        {
            if (manager.CurrentLevel == null) return;
            int cosmeticCount = manager.CurrentLevel.cosmeticInstances.Count;
            int orderCount = manager.CurrentLevel.orderBags.Count;
            int totalSlots = 0;
            foreach (var order in manager.CurrentLevel.orderBags)
            {
                totalSlots += order.requiredItems.Count;
            }
            Debug.Log("========== 关卡统计信息 ==========");
            Debug.Log($"关卡类型: {manager.CurrentLevel.levelType}");
            Debug.Log($"化妆品总数: {cosmeticCount}");
            Debug.Log($"订单包总数: {orderCount}");
            Debug.Log($"槽位总数: {totalSlots}");
            int trophyCount = 0;
            foreach (var item in manager.CurrentLevel.cosmeticInstances)
            {
                if (item.isLock) trophyCount++;
            }
            int availableItems = cosmeticCount - trophyCount;

            if (availableItems == totalSlots)
            {
                Debug.Log($"<color=green>✓ 物品数量与槽位完美匹配！</color>");
            }
            else if (availableItems > totalSlots)
            {
                Debug.LogWarning($"⚠ 有 {availableItems - totalSlots} 个多余物品（伪钥匙）
");
            }
            else
            {
                Debug.LogError($"✗ 缺少 {totalSlots - availableItems} 个物品！");
            }
            Debug.Log($"奖杯数量: {trophyCount}");
            Debug.Log("================================");
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;
using System.Linq;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class OrderAreaUI : MonoBehaviour
    {
        [Header("输入控件")]
        [SerializeField] private TMP_InputField x2CountInput;              // X
        [SerializeField] private TMP_InputField x3CountInput;              // X
        [SerializeField] private TMP_InputField x4CountInput;              // X
        [SerializeField] private TMP_InputField x5CountInput;              // X
        [SerializeField] private TMP_InputField cosmeticsTypeCountInput;   //
        [Header("按钮")]
        [SerializeField] private Button generateOrderBagBtn;               //
        [SerializeField] private Button clearOrderBagBtn;                  // 清
        [SerializeField] private Button addTrophyBagBtn;                   // 添
        [SerializeField] private Button modifyOrderBtn;                    // 修
        [SerializeField] private Button algorithmSetBtn;                   // 算
        [Header("显示区域")]
        [SerializeField] private Transform orderBagContainer;              //
        [Header("运行时数据")]
        private LevelEditorManager manager;
        private List<OrderBagUI> activeOrderBags = new List<OrderBagUI>();
        private OrderBagUI trophyBag;
        private OrderBagUI selectedOrderBag;                                  // 当
        [Header("面板引用")]
        [SerializeField] private OrderBagInfoPanel orderBagInfoPanel;
        [SerializeField] private AlgorithmParaPanel algorithmParaPanel;
        public void Initialize(LevelEditorManager manager)
        {
            this.manager = manager;
            SetupButtonListeners();
            if (x2CountInput != null) x2CountInput.text = "0";
            if (x3CountInput != null) x3CountInput.text = "4";
            if (x4CountInput != null) x4CountInput.text = "0";
            if (x5CountInput != null) x5CountInput.text = "0";
            if (cosmeticsTypeCountInput != null) cosmeticsTypeCountInput.text = "8";
            if (modifyOrderBtn != null)
                modifyOrderBtn.interactable = false;
        }
        private void SetupButtonListeners()
        {
            if (generateOrderBagBtn != null)
generateOrderBagBtn.onClick.AddListener(OnGenerateOrderBags);
            if (clearOrderBagBtn != null)
                clearOrderBagBtn.onClick.AddListener(OnClearAllOrderBags);
            if (addTrophyBagBtn != null)
                addTrophyBagBtn.onClick.AddListener(OnAddTrophyBag);
            if (modifyOrderBtn != null)
                modifyOrderBtn.onClick.AddListener(OnModifyOrderBag);
            if (algorithmSetBtn != null)
algorithmSetBtn.onClick.AddListener(OnOpenAlgorithmSettings);
        }
        private void OnGenerateOrderBags()
        {
            if (!ValidateInputs(out int x2Count, out int x3Count, out int x4Count,
                out int x5Count, out int typeCount))
            {
                return;
            }
            if (x2Count == 0 && x3Count == 0 && x4Count == 0 && x5Count == 0)
            {
                Debug.LogError("请至少输入一种订单包的数量！");
                return;
            }
            LevelType levelType = LevelType.Normal;
            if (manager.CurrentLevel != null)
            {
                levelType = manager.CurrentLevel.levelType;
            }
            var orderBags = manager.orderGenerator.GenerateOrderBags(
                x2Count,
                x3Count,
                x4Count,
                x5Count,
                typeCount,
                levelType
            );
            if (orderBags == null || orderBags.Count == 0)
            {
                Debug.LogError("生成失败！请检查配置。");
                return;
            }
            ClearAll();
            LoadOrderBags(orderBags);
            Debug.Log($"<color=green>✓ 成功生成 {orderBags.Count} 个订单包！</color>");
            int totalItems = 0;
            foreach (var bag in orderBags)
            {
                totalItems += bag.requiredItems.Count;
            }
            Debug.Log($"总订单包数: {orderBags.Count}, 总化妆品数: {totalItems}");
        }
        private bool ValidateInputs(out int x2Count, out int x3Count, out int x4Count,
            out int x5Count, out int typeCount)
        {
            x2Count = 0;
            x3Count = 0;
            x4Count = 0;
            x5Count = 0;
            typeCount = 0;
            if (x2CountInput != null && !string.IsNullOrEmpty(x2CountInput.text))
            {
                if (!int.TryParse(x2CountInput.text, out x2Count) || x2Count < 0)
                {
                    Debug.LogError("X2数量必须是非负整数！");
                    return false;
                }
            }
            if (x3CountInput != null && !string.IsNullOrEmpty(x3CountInput.text))
            {
                if (!int.TryParse(x3CountInput.text, out x3Count) || x3Count < 0)
                {
                    Debug.LogError("X3数量必须是非负整数！");
                    return false;
                }
            }
            if (x4CountInput != null && !string.IsNullOrEmpty(x4CountInput.text))
            {
                if (!int.TryParse(x4CountInput.text, out x4Count) || x4Count < 0)
                {
                    Debug.LogError("X4数量必须是非负整数！");
                    return false;
                }
            }
            if (x5CountInput != null && !string.IsNullOrEmpty(x5CountInput.text))
            {
                if (!int.TryParse(x5CountInput.text, out x5Count) || x5Count < 0)
                {
                    Debug.LogError("X5数量必须是非负整数！");
                    return false;
                }
            }
            if (cosmeticsTypeCountInput == null || string.IsNullOrEmpty(cosmeticsTypeCountInput.text))
            {
                Debug.LogError("请输入化妆品类型数量！");
                return false;
            }
            if (!int.TryParse(cosmeticsTypeCountInput.text, out typeCount) || typeCount <= 0)
            {
                Debug.LogError("化妆品类型数必须是正整数！");
                return false;
            }
            return true;
        }
        private void OnClearAllOrderBags()
        {
            ClearAll();
            Debug.Log("已清除所有订单包");
        }
        private void OnAddTrophyBag()
        {
            if (activeOrderBags.Count == 0)
            {
                Debug.LogError("请先生成普通订单包！");
                return;
            }
            if (trophyBag != null)
            {
                Debug.LogWarning("已存在奖杯订单包！");
                return;
            }
            var allCosmetics = manager.GetAllCosmeticItems();
            if (allCosmetics.Length == 0)
            {
                Debug.LogError("没有可用的化妆品数据！");
                return;
            }
            var randomCosmetic = allCosmetics[Random.Range(0, allCosmetics.Length)];
            OrderBagData trophyData = new OrderBagData();
            trophyData.configID = "OrderBagX1Config";
            trophyData.requiredItems = new List<string> { randomCosmetic.cosmeticID };
            trophyData.orderIndex = activeOrderBags.Count;
            trophyData.bagType = OrderBagType.Normal;
            trophyData.tempSlotCount = 4;
            var trophyConfig = manager.GetOrderBagConfig("OrderBagX1Config");
            if (trophyConfig == null)
            {
                Debug.LogError("未找到 OrderBagX1Config 配置！");
                return;
            }
            CreateOrderBagUI(trophyData, trophyConfig, true);
            ArrangeOrderBags();
            Debug.Log($"<color=green>✓ 已添加奖杯包：{randomCosmetic.itemName}</color>
");
        }
        private void OnModifyOrderBag()
        {
            if (selectedOrderBag == null)
            {
                Debug.LogWarning("请先选中一个订单包！");
                return;
            }
            if (orderBagInfoPanel != null)
            {
                orderBagInfoPanel.Open(selectedOrderBag.GetData(), selectedOrderBag);   //
✅ 修复
            }
            else
            {
                Debug.LogError("订单包信息面板未赋值！");
            }
        }
        private void OnOpenAlgorithmSettings()
        {
            if (algorithmParaPanel != null)
            {
                algorithmParaPanel.Open();
            }
            else
            {
                Debug.LogError("算法参数面板未赋值！");
            }
        }
        public void LoadOrderBags(List<OrderBagData> orderBags)
        {
            ClearAll();
            var sortedBags = orderBags.OrderBy(bag => bag.orderIndex).ToList();
            Debug.Log($"<color=cyan>═══ LoadOrderBags 开始 ═══</color>");
            Debug.Log($"<color=cyan>原始顺序: {string.Join(", ", orderBags.Select(b => $"{b.con
figID}(idx:{b.orderIndex})"))}");
            Debug.Log($"<color=cyan>排序后: {string.Join(", ", sortedBags.Select(b => $"{b.confi
gID}(idx:{b.orderIndex})"))}");
            foreach (var bagData in sortedBags)
            {
                var config = manager.GetOrderBagConfig(bagData.configID);
                if (config != null)
                {
                    CreateOrderBagUI(bagData, config, false);
                }
            }
            ArrangeOrderBags();
            Debug.Log($"<color=green>✓ 已加载 {activeOrderBags.Count} 个订单包（按orderI
ndex排序）</color>");
        }
        private void CreateOrderBagUI(OrderBagData data, OrderBagConfigSO config, bool isTr
ophy)
        {
            if (config.bagPrefab == null || orderBagContainer == null)
            {
                Debug.LogError($"订单包预制件未设置: {config.configName}");
                return;
            }
            GameObject bagObj = Instantiate(config.bagPrefab, orderBagContainer);
            RectTransform rect = bagObj.GetComponent<RectTransform>();
            if (rect != null)
            {
                rect.sizeDelta = new Vector2(260, rect.sizeDelta.y);
            }
            OrderBagUI bagUI = bagObj.GetComponent<OrderBagUI>();
            if (bagUI == null)
            {
                bagUI = bagObj.AddComponent<OrderBagUI>();
            }
            bagUI.Initialize(data, config, this);
            if (isTrophy)
            {
                trophyBag = bagUI;
                Image bgImage = bagObj.GetComponent<Image>();
                if (bgImage != null)
                {
                    bgImage.color = new Color(1f, 0.85f, 0f, 0.3f);
                }
            }
            else
            {
                activeOrderBags.Add(bagUI);
            }
        }
        public void ArrangeOrderBags()
        {
            foreach (var bag in activeOrderBags)
            {
                bag.gameObject.SetActive(true);
            }

            if (trophyBag != null)
            {
                trophyBag.gameObject.SetActive(true);
            }
        }
        public void SelectOrderBag(OrderBagUI orderBag)
        {
            if (selectedOrderBag != null)
            {
                selectedOrderBag.SetSelected(false);
            }
            selectedOrderBag = orderBag;
            if (selectedOrderBag != null)
            {
                selectedOrderBag.SetSelected(true);
                if (modifyOrderBtn != null)
                    modifyOrderBtn.interactable = true;
            }
            else
            {
                if (modifyOrderBtn != null)
                    modifyOrderBtn.interactable = false;
            }
        }
        public void ClearAll()
        {
            foreach (var bag in activeOrderBags)
            {
                if (bag != null && bag.gameObject != null)
                    Destroy(bag.gameObject);
            }
            activeOrderBags.Clear();
            if (trophyBag != null)
            {
                Destroy(trophyBag.gameObject);
                trophyBag = null;
            }
            selectedOrderBag = null;
            if (modifyOrderBtn != null)
                modifyOrderBtn.interactable = false;
        }
        public List<OrderBagData> GetAllOrderBags()
        {
            List<OrderBagData> allBags = new List<OrderBagData>();
            for (int i = 0; i < activeOrderBags.Count; i++)
            {
                var bagData = activeOrderBags[i].GetData();
                if (bagData != null)
                {
                    bagData.orderIndex = i;   // 更新orderIndex为当前UI顺序
                    allBags.Add(bagData);
                }
            }
            if (trophyBag != null)
            {
                var trophyData = trophyBag.GetData();
                if (trophyData != null)
                {
                    trophyData.orderIndex = allBags.Count;   // 奖杯包放在最后
                    allBags.Add(trophyData);
                }
            }
            return allBags;
        }
        public bool HasOrderBags()
        {
            return activeOrderBags.Count > 0 || trophyBag != null;
        }
        public void OnOrderBagReordered()
        {
            SyncActiveOrderBagsWithUIOrder();
            ArrangeOrderBags();
            UpdateAllOrderIndices();
            Debug.Log($"<color=yellow>订单包已重新排序，当前顺序: {string.Join(", ", activeOrd
erBags.Select(b => b.GetData().configID))}</color>");
        }
        private void SyncActiveOrderBagsWithUIOrder()
        {
            if (orderBagContainer == null || activeOrderBags.Count == 0)
```

```
                return;
            activeOrderBags = activeOrderBags
                .OrderBy(bag => bag.transform.GetSiblingIndex())
                .ToList();

            Debug.Log($"<color=cyan>已同步activeOrderBags顺序与UI顺序</color>");
        }
        private void UpdateAllOrderIndices()
        {
            for (int i = 0; i < activeOrderBags.Count; i++)
            {
                var bagData = activeOrderBags[i].GetData();
                if (bagData != null)
                {
                    bagData.orderIndex = i;
                    Debug.Log($"<color=green>更新索引: {bagData.configID} -> orderIndex = {
i}</color>");
                }
            }
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;
using System.Linq;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class OrderBagInfoPanel : MonoBehaviour
    {
        [Header("UI引用")]
        [SerializeField] private TMP_InputField layerCountInput;
        [SerializeField] private ToggleGroup orderTypeGroup;
        [Header("订单包类型Toggles")]
        [SerializeField] private Toggle normalBagToggle;
        [SerializeField] private Toggle brandPerBagToggle;
        [SerializeField] private Toggle colorPerBagToggle;
        [SerializeField] private Toggle typePerBagToggle;
        [Header("🆕 完美订单勾选")]
        [SerializeField] private Toggle isPerfectOrderToggle;   // 🆕 完美订
        [Header("槽位输入框")]
        [SerializeField] private TMP_InputField slot1Input;
        [SerializeField] private TMP_InputField slot2Input;
        [SerializeField] private TMP_InputField slot3Input;
        [SerializeField] private TMP_InputField slot4Input;
        [SerializeField] private TMP_InputField slot5Input;
        [Header("按钮")]
        [SerializeField] private Button confirmButton;
        [SerializeField] private Button cancelButton;
        private OrderBagData currentOrderBag;
        private OrderBagUI currentOrderBagUI;
        private List<TMP_InputField> slotInputs;
        private void Awake()
        {
            slotInputs = new List<TMP_InputField>
            {
                slot1Input, slot2Input, slot3Input, slot4Input, slot5Input
            };
            if (confirmButton != null)
                confirmButton.onClick.AddListener(OnConfirm);
            if (cancelButton != null)
                cancelButton.onClick.AddListener(OnCancel);
            if (isPerfectOrderToggle != null)
            {
isPerfectOrderToggle.onValueChanged.AddListener(OnPerfectOrderToggleChanged);
            }
            Debug.Log("<color=yellow>OrderBagInfoPanel.Awake: 强制初始化皮肤缓存</color>
");
            OrderBagSkinManager.ClearCache();
            OrderBagSkinManager.InitializeSkinCache();
            gameObject.SetActive(false);
        }
        public void Open(OrderBagData orderBag, OrderBagUI orderBagUI)
        {
            currentOrderBag = orderBag;
            currentOrderBagUI = orderBagUI;
            gameObject.SetActive(true);
            LoadOrderBagData();
        }
        private void LoadOrderBagData()
        {
            if (currentOrderBag == null) return;
            if (layerCountInput != null)
            {
                layerCountInput.text = currentOrderBag.tempSlotCount.ToString();
            }
            SetOrderTypeToggle(currentOrderBag.bagType);
            if (isPerfectOrderToggle != null)
            {
isPerfectOrderToggle.onValueChanged.RemoveListener(OnPerfectOrderToggleChanged);
                isPerfectOrderToggle.isOn = currentOrderBag.isPerfectOrder;   isPerfectOrder
Toggle.onValueChanged.AddListener(OnPerfectOrderToggleChanged);
            }
            for (int i = 0; i < slotInputs.Count; i++)
            {
                if (slotInputs[i] != null)
                {
                    if (i < currentOrderBag.requiredItems.Count)
                    {
                        slotInputs[i].text = currentOrderBag.requiredItems[i];
                        slotInputs[i].gameObject.SetActive(true);
                    }
                    else
                    {
                        slotInputs[i].text = "";
                        slotInputs[i].gameObject.SetActive(false);
                    }
                }
            }
            Debug.Log($"加载订单包数据: {currentOrderBag.configID}, 类型: {currentOrderBag.b
agType}, 完美订单: {currentOrderBag.isPerfectOrder}");
        }
        private void OnPerfectOrderToggleChanged(bool isOn)
        {
            if (currentOrderBag == null) return;
            currentOrderBag.isPerfectOrder = isOn;
            if (isOn)
            {
                currentOrderBag.currentSkinID = "perfect";
                Debug.Log("<color=yellow>✓ 勾选完美订单，切换到完美订单皮肤</color>");
            }
            else
            {
                currentOrderBag.currentSkinID = "normal";
                Debug.Log("<color=yellow>✓ 取消完美订单，恢复普通皮肤</color>");
            }
            if (currentOrderBagUI != null)
            {
                currentOrderBagUI.ApplySkin();
                Debug.Log($"<color=green>✓ 皮肤已立即更新：{currentOrderBag.currentSkinI
D}</color>");
            }
        }
        private void SetOrderTypeToggle(OrderBagType bagType)
        {
            if (normalBagToggle != null) normalBagToggle.isOn = false;
            if (brandPerBagToggle != null) brandPerBagToggle.isOn = false;
            if (colorPerBagToggle != null) colorPerBagToggle.isOn = false;
            if (typePerBagToggle != null) typePerBagToggle.isOn = false;
            switch (bagType)
            {
                case OrderBagType.Normal:
                    if (normalBagToggle != null)
                        normalBagToggle.isOn = true;
                    break;
                case OrderBagType.BrandPerBag:
                    if (brandPerBagToggle != null)
                        brandPerBagToggle.isOn = true;
                    break;
                case OrderBagType.ColorPerBag:
                    if (colorPerBagToggle != null)
                        colorPerBagToggle.isOn = true;
                    break;
                case OrderBagType.TypePerBag:
                    if (typePerBagToggle != null)
                        typePerBagToggle.isOn = true;
                    break;
            }
        }
        private void OnConfirm()
        {
            if (currentOrderBag == null)
            {
                Debug.LogError("没有要修改的订单包！");
                return;
            }
            if (!int.TryParse(layerCountInput.text, out int tempSlot) || tempSlot < 0)
            {
                Debug.LogError("临时槽数无效！");
                return;
            }
            currentOrderBag.tempSlotCount = tempSlot;
            OrderBagType newType = GetSelectedOrderType();
            List<string> newItems = new List<string>();
            foreach (var input in slotInputs)
            {
                if (input != null && input.gameObject.activeSelf && !string.IsNullOrEmpty(input.te
xt))
                {
                    newItems.Add(input.text.Trim());
                }
            }
            if (!ValidateCosmeticIDs(newItems, newType, out string errorMsg))
            {
                Debug.LogError(errorMsg);
                return;
            }
            currentOrderBag.requiredItems = newItems;
            currentOrderBag.bagType = newType;
            if (currentOrderBagUI != null)
            {
                currentOrderBagUI.RefreshDisplay();
                currentOrderBagUI.ApplySkin();   // 🆕 确保皮肤也被刷新
            }
            Debug.Log($"<color=green>✓ 订单包修改成功！类型: {newType}, 临时槽数: {temp
Slot}, 完美订单: {currentOrderBag.isPerfectOrder}</color>");

            Close();
        }
        private OrderBagType GetSelectedOrderType()
        {
            if (normalBagToggle != null && normalBagToggle.isOn)
                return OrderBagType.Normal;
            if (brandPerBagToggle != null && brandPerBagToggle.isOn)
                return OrderBagType.BrandPerBag;
            if (colorPerBagToggle != null && colorPerBagToggle.isOn)
                return OrderBagType.ColorPerBag;
            if (typePerBagToggle != null && typePerBagToggle.isOn)
                return OrderBagType.TypePerBag;
            return OrderBagType.Normal;
        }
        private bool ValidateCosmeticIDs(List<string> cosmeticIDs, OrderBagType bagType, out
string errorMsg)
        {
            errorMsg = "";
            if (cosmeticIDs.Count == 0)
            {
                errorMsg = "至少需要一个化妆品ID！";
                return false;
            }
            var allCosmetics = LevelEditorManager.Instance.GetAllCosmeticItems();
            foreach (var id in cosmeticIDs)
            {
                var cosmetic = System.Array.Find(allCosmetics, c => c.cosmeticID == id);
                if (cosmetic == null)
                {
                    errorMsg = $"找不到化妆品ID: {id}";
                    return false;
                }
            }
            return true;
        }
        private void OnCancel()
        {
            Close();
        }
        public void Close()
        {
            gameObject.SetActive(false);
            currentOrderBag = null;
            currentOrderBagUI = null;
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using TMPro;
using System.Collections.Generic;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class OrderBagUI : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler, IPointerClickHandler
    {
        [Header("🆕 皮肤组件")]
        [SerializeField] private Image backgroundImage;  // 背景图组件（需要在
        private OrderBagData data;
        private OrderBagConfigSO config;
        private OrderAreaUI orderArea;
        private LevelEditorManager manager;  // 🆕 添加 manager 引用
        private List<Image> slotImages = new List<Image>();
        private bool isSelected = false;
        private Outline outline;
        private static Dictionary<string, CosmeticItemSO> cosmeticCache = new Dictionary<string, CosmeticItemSO>();
        private static bool isCacheInitialized = false;
        [Header("🆕 槽位颜色配置")]
        [SerializeField] private Color greyedOutColor = new Color(0.5f, 0.5f, 0.5f, 1.0f);  // 置灰颜色
        [SerializeField] private Color normalColor = Color.white;  // 正常颜
        private Canvas canvas;
        private RectTransform rectTransform;
        private CanvasGroup canvasGroup;
        private Vector2 originalPosition;
        private int originalSiblingIndex;
        private Transform originalParent;
        private GameObject insertLinePreview;
        private bool isDragging = false;
        private static void InitializeCosmeticCache()
        {
            if (isCacheInitialized) return;
            var allCosmetics = Resources.LoadAll<CosmeticItemSO>("Cosmetics");
            cosmeticCache.Clear();
            foreach (var cosmetic in allCosmetics)
            {
                if (!string.IsNullOrEmpty(cosmetic.cosmeticID))
                {
                    cosmeticCache[cosmetic.cosmeticID] = cosmetic;
                }
            }
            isCacheInitialized = true;
            Debug.Log($"<color=cyan>[OrderBagUI] 初始化化妆品缓存: {cosmeticCache.Count}个化妆品</color>");
        }
        public void Initialize(OrderBagData bagData, OrderBagConfigSO bagConfig, OrderAreaUI area)
        {
            data = bagData;
            config = bagConfig;
            orderArea = area;
            manager = FindObjectOfType<LevelEditorManager>();
            InitializeCosmeticCache();
            canvas = GetComponentInParent<Canvas>();
            rectTransform = GetComponent<RectTransform>();
            canvasGroup = GetComponent<CanvasGroup>();
            if (canvasGroup == null)
            {
                canvasGroup = gameObject.AddComponent<CanvasGroup>();
            }
            outline = gameObject.GetComponent<Outline>();
            if (outline == null)
            {
                outline = gameObject.AddComponent<Outline>();
            }
            outline.effectColor = Color.yellow;
            outline.effectDistance = new Vector2(3, 3);
            outline.enabled = false;
            if (backgroundImage == null)
            {
                Transform bgTransform = transform.Find("Background");
                if (bgTransform != null)
                {
                    backgroundImage = bgTransform.GetComponent<Image>();
                }
            }
            FindAllSlots();
            ApplySkin();
            RefreshDisplay();
        }
        private void FindAllSlots()
        {
            slotImages.Clear();
            for (int i = 0; i < 10; i++)
            {
                Transform slotTransform = transform.Find($"Slot_{i}");
                if (slotTransform != null)
                {
                    Image slotImage = slotTransform.GetComponent<Image>();
                    if (slotImage != null)
                    {
                        slotImages.Add(slotImage);
                    }
                    else
                    {
                        Debug.LogWarning($"Slot_{i} found but has no Image component!");
                    }
                }
                else
                {
                    break;
                }
            }
            Debug.Log($"<color=cyan>[OrderBagUI] 找到 {slotImages.Count} 个槽位</color>");
        }
        public void RefreshDisplay()
        {
            if (data == null) return;
            gameObject.name = $"OrderBag_{data.configID}_{data.orderIndex}";
            for (int i = 0; i < slotImages.Count; i++)
            {
                if (i < data.requiredItems.Count)
                {
                    slotImages[i].gameObject.SetActive(true);
                    string itemID = data.requiredItems[i];
                    if (cosmeticCache.TryGetValue(itemID, out CosmeticItemSO cosmetic))
                    {
                        if (cosmetic.icon != null)
                        {
                            slotImages[i].sprite = cosmetic.icon;
                            slotImages[i].color = greyedOutColor;   // 初始置灰
                            slotImages[i].enabled = true;

                            Debug.Log($"<color=green>✓ 槽位 {i} 已加载图标: {cosmetic.ite
mName}</color>");
                        }
                        else
                        {
                            Debug.LogWarning($"化妆品 {itemID} 没有设置图标!");
                        }
                    }
                    else
                    {
                        Debug.LogWarning($"化妆品 {itemID} 在缓存中未找到!");
                    }
                }
                else
                {
                    slotImages[i].gameObject.SetActive(false);
                }
            }
        }
public void ApplySkin()
{
    Debug.Log($"<color=yellow>═══ ApplySkin 开始 ═══</color>");
    Debug.Log($"GameObject: {gameObject.name}");
    Debug.Log($"data: {(data == null ? "NULL" : "OK")}");
    Debug.Log($"backgroundImage: {(backgroundImage == null ? "NULL" : "OK")}");
    if (data == null)
    {
        Debug.LogWarning("OrderBagData is null, cannot apply skin");
        return;
    }
    string effectiveSkinID = OrderBagSkinManager.GetEffectiveSkinID(data);
    Debug.Log($"<color=cyan>有效皮肤ID: {effectiveSkinID}</color>");
    Debug.Log($"data.isPerfectOrder: {data.isPerfectOrder}");
    Debug.Log($"data.currentSkinID: {data.currentSkinID}");
    Debug.Log($"<color=yellow>开始加载皮肤: {effectiveSkinID}</color>");
    OrderBagSkinSO skin = OrderBagSkinManager.GetSkin(effectiveSkinID);
    Debug.Log($"<color=yellow>皮肤加载结果: {(skin == null ? "NULL" : skin.skinName)}</color>
");
    if (skin == null)
    {
        Debug.LogWarning($"<color=red>✕ 无法加载皮肤: {effectiveSkinID}，尝试使用默认皮肤
</color>");
        skin = OrderBagSkinManager.GetDefaultSkin();
        Debug.Log($"默认皮肤加载结果: {(skin == null ? "NULL" : skin.skinName)}");
    }
    if (skin == null)
    {
        Debug.LogError("<color=red>✕ 连默认皮肤都无法加载！请检查 Resources/OrderBagSki
ns 文件夹</color>");
        return;
    }
    Debug.Log($"<color=green>✓ 皮肤加载成功: {skin.skinName}</color>");
    Debug.Log($"backgroundSprite: {(skin.backgroundSprite == null ? "NULL" : skin.backgroundS
prite.name)}");
    Debug.Log($"tintColor: {skin.tintColor}");
    if (backgroundImage != null && skin.backgroundSprite != null)
    {
        Debug.Log($"<color=cyan>准备应用背景图...</color>");
        Debug.Log($"当前背景图: {(backgroundImage.sprite == null ? "NULL" : backgroundImag
e.sprite.name)}");
        backgroundImage.sprite = skin.backgroundSprite;
        backgroundImage.color = skin.tintColor;
        Debug.Log($"<color=green>✓✓✓ 订单包 [{gameObject.name}] 应用皮肤成功！</color>
");
        Debug.Log($"新背景图: {backgroundImage.sprite.name}");
        Debug.Log($"新颜色: {backgroundImage.color}");
    }
    else
    {
        Debug.LogError($"<color=red>═══ 应用皮肤失败 ═══</color>");
        if (backgroundImage == null)
        {
            Debug.LogError($"<color=red>✕ backgroundImage 为 NULL！</color>");
        }
        else
        {
            Debug.Log($"<color=green>✓ backgroundImage 正常: {backgroundImage.name}</c
olor>");
        }
        if (skin.backgroundSprite == null)
        {
            Debug.LogError($"<color=red>✕ skin.backgroundSprite 为 NULL！皮肤: {skin.skinN
ame}</color>");
            Debug.LogError($"<color=red>请检查 Skin_perfect.asset 是否设置了 Background S
prite！</color>");
        }
        else
        {
            Debug.Log($"<color=green>✓ skin.backgroundSprite 正常: {skin.backgroundSprite.n
ame}</color>");
        }
    }
    Debug.Log($"<color=yellow>═══ ApplySkin 结束 ═══</color>");
}
        public void OnBeginDrag(PointerEventData eventData)
        {
            originalPosition = rectTransform.anchoredPosition;
            originalSiblingIndex = transform.GetSiblingIndex();
            originalParent = transform.parent;
            canvasGroup.alpha = 0.6f;
            canvasGroup.blocksRaycasts = false;
            CreateInsertLinePreview();
            orderArea.SelectOrderBag(this);
        }
        public void OnDrag(PointerEventData eventData)
        {
            if (canvas == null) return;
            rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;
            UpdateInsertLinePreview(eventData.position);
        }
```

```
public void OnEndDrag(PointerEventData eventData)
{
    isDragging = false;
    canvasGroup.alpha = 1f;
    canvasGroup.blocksRaycasts = true;
    int newIndex = GetInsertIndex(eventData.position);
    transform.SetSiblingIndex(newIndex);
    DestroyInsertLinePreview();
    if (orderArea != null)
    {
        orderArea.OnOrderBagReordered();   // 调用 OrderAreaUI 的方法
    }
}
public void OnPointerClick(PointerEventData eventData)
{
    if (!isDragging)
    {
        orderArea.SelectOrderBag(this);
    }
}
public void SetSelected(bool selected)
{
    isSelected = selected;
    if (outline != null)
    {
        outline.enabled = selected;
    }
}
private void CreateInsertLinePreview()
{
    if (insertLinePreview == null)
    {
        insertLinePreview = new GameObject("InsertLine");
        insertLinePreview.transform.SetParent(transform.parent, false);
        Image lineImage = insertLinePreview.AddComponent<Image>();
        lineImage.color = new Color(1f, 1f, 0f, 0.5f);
        RectTransform lineRect = insertLinePreview.GetComponent<RectTransform>();
        lineRect.sizeDelta = new Vector2(5, 200);
    }
}
private void UpdateInsertLinePreview(Vector2 screenPosition)
{
    if (insertLinePreview == null) return;
    int insertIndex = GetInsertIndex(screenPosition);
    RectTransform lineRect = insertLinePreview.GetComponent<RectTransform>();
    if (insertIndex == 0)
    {
        Transform firstChild = transform.parent.GetChild(0);
        RectTransform firstRect = firstChild.GetComponent<RectTransform>();
        lineRect.anchoredPosition = firstRect.anchoredPosition - new Vector2(135, 0);
    }
    else
    {
        Transform prevChild = transform.parent.GetChild(insertIndex - 1);
        RectTransform prevRect = prevChild.GetComponent<RectTransform>();
        lineRect.anchoredPosition = prevRect.anchoredPosition + new Vector2(135, 0);
    }
}
private void DestroyInsertLinePreview()
{
    if (insertLinePreview != null)
    {
        Destroy(insertLinePreview);
```

```
                insertLinePreview = null;
            }
        }
        private int GetInsertIndex(Vector2 screenPosition)
        {
            int insertIndex = transform.GetSiblingIndex();

            for (int i = 0; i < transform.parent.childCount; i++)
            {
                Transform child = transform.parent.GetChild(i);
                if (child == transform) continue;

                RectTransform childRect = child.GetComponent<RectTransform>();
                if (childRect != null)
                {
                    Vector2 localPos;
                    RectTransformUtility.ScreenPointToLocalPointInRectangle(
                        transform.parent.GetComponent<RectTransform>(),
                        screenPosition,
                        canvas.worldCamera,
                        out localPos
                    );
                    if (localPos.x < childRect.anchoredPosition.x)
                    {
                        return i;
                    }
                }
            }
            return transform.parent.childCount - 1;
        }
        public OrderBagData GetData()
        {
            return data;
        }
        public void SetData(OrderBagData newData)
        {
            data = newData;
            RefreshDisplay();
        }
        public OrderBagConfigSO GetConfig()
        {
            return config;
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class PrefabLibraryUI : MonoBehaviour
    {
        [Header("UI组件")]
        [SerializeField] private Transform libraryContainer;          // 库物
        [SerializeField] private GameObject libraryItemPrefab;         // 库物
        [SerializeField] private TMP_InputField searchInput;           // 搜索
        private LevelEditorManager manager;
        private List<CosmeticLibraryItem> activeLibraryItems = new List<CosmeticLibraryItem>();
        private StackingAreaUI stackingArea;
        public void Initialize(LevelEditorManager manager)
        {
            this.manager = manager;
            stackingArea = FindObjectOfType<StackingAreaUI>();
            SetupButtonListeners();
            LoadAllCosmetics();
        }
        private void SetupButtonListeners()
        {
            if (searchInput != null)
            {
                searchInput.onValueChanged.AddListener(OnSearchChanged);
            }
        }
        private void LoadAllCosmetics()
        {
            ClearLibrary();
            if (manager == null)
            {
                Debug.LogError("LevelEditorManager 未初始化！");
                return;
            }
            var allCosmetics = manager.GetAllCosmeticItems();
            if (allCosmetics == null || allCosmetics.Length == 0)
            {
                Debug.LogWarning("没有找到任何化妆品数据！");
                return;
            }
            foreach (var cosmetic in allCosmetics)
            {
                CreateLibraryItem(cosmetic);
            }
            Debug.Log($"✓ 化妆品库已加载 {allCosmetics.Length} 个物品");
        }
        private void CreateLibraryItem(CosmeticItemSO cosmetic)
        {
            if (libraryItemPrefab == null || libraryContainer == null)
            {
                Debug.LogError("化妆品库配置不完整！");
                return;
            }
            GameObject itemObj = Instantiate(libraryItemPrefab, libraryContainer);
            CosmeticLibraryItem libraryItem = itemObj.GetComponent<CosmeticLibraryItem>();
            if (libraryItem == null)
            {
                libraryItem = itemObj.AddComponent<CosmeticLibraryItem>();
            }
            libraryItem.Initialize(cosmetic, stackingArea);
            activeLibraryItems.Add(libraryItem);
        }
        private void OnSearchChanged(string searchText)
        {
            if (string.IsNullOrEmpty(searchText))
            {
                foreach (var item in activeLibraryItems)
                {
                    item.gameObject.SetActive(true);
                }
            }
            else
            {
                string lowerSearch = searchText.ToLower();
                foreach (var item in activeLibraryItems)
                {
                    bool matches = item.CosmeticData != null && item.CosmeticData.itemNam
e.ToLower().Contains(lowerSearch);
                    item.gameObject.SetActive(matches);
                }
            }
        }
        private void ClearLibrary()
        {
            foreach (var item in activeLibraryItems)
            {
                if (item != null && item.gameObject != null)
                {
                    Destroy(item.gameObject);
                }
            }
            activeLibraryItems.Clear();
        }
    }
}
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;
using System.Linq;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class StackingAreaUI : MonoBehaviour
    {
        [Header("UI按钮")]
        [SerializeField] private Button generateCosmeticsBtn;
        [SerializeField] private Button clearButton;
        [Header("堆叠区域边界")]
        [SerializeField] private float worldBoundsMinX = -8f;
        [SerializeField] private float worldBoundsMaxX = 8f;
        [SerializeField] private float worldBoundsMinY = -4f;
        [SerializeField] private float worldBoundsMaxY = 4f;
        [Header("化妆品容器")]
        [SerializeField] private Transform cosmeticContainer;
        [Header("拖动预览")]
        [SerializeField] private GameObject dragPreviewPrefab;
        private GameObject dragPreviewInstance;
        private LevelEditorManager manager;
        private List<GameObject> activeCosmeticObjects = new List<GameObject>();
        public void Initialize(LevelEditorManager manager)
        {
            this.manager = manager;
            SetupButtonListeners();
        }
        private void SetupButtonListeners()
        {
            if (generateCosmeticsBtn != null)
generateCosmeticsBtn.onClick.AddListener(OnGenerateCosmetics);
            if (clearButton != null)
                clearButton.onClick.AddListener(ClearAll);
        }
        private void OnGenerateCosmetics()
        {
            var orderArea = FindObjectOfType<OrderAreaUI>();
            if (orderArea == null || !orderArea.HasOrderBags())
            {
                Debug.LogError("请先在订单区生成订单包！");
                return;
            }
            var orderBags = orderArea.GetAllOrderBags();
            var sortedOrderBags = orderBags.OrderBy(bag => bag.orderIndex).ToList();
            Debug.Log($"<color=yellow>═══ 生成化妆品 - 订单包排序检查 ═══</color>");
            Debug.Log($"<color=yellow>原始顺序: {string.Join(", ", orderBags.Select(b => $"{b.c
onfigID}(idx:{b.orderIndex})"))}");
            Debug.Log($"<color=yellow>排序后: {string.Join(", ", sortedOrderBags.Select(b => $"
{b.configID}(idx:{b.orderIndex})"))}");
            var instances = manager.cosmeticGenerator.GenerateWithOrderBagTempSlots(
                sortedOrderBags,  // 🔧 使用排序后的列表
                worldBoundsMinX,
                worldBoundsMaxX,
                worldBoundsMinY,
                worldBoundsMaxY
            );
            if (instances == null || instances.Count == 0)
            {
                Debug.LogError("化妆品生成失败！");
                return;
            }
            LoadInstances(instances);
            Debug.Log($"<color=green>✓ 成功生成 {instances.Count} 个化妆品！</color>");
            UpdateAllItemsClickableState();
        }
        public void LoadInstances(List<CosmeticInstanceData> instances)
        {
            ClearAll();
            Debug.Log($"<color=yellow>开始加载 {instances.Count} 个化妆品实例...</color>");
            var allCosmetics = manager.GetAllCosmeticItems();
            int successCount = 0;
            foreach (var instanceData in instances)
            {
                var cosmeticSO = System.Array.Find(allCosmetics, c => c.cosmeticID == instan
ceData.cosmeticID);
                if (cosmeticSO == null)
                {
                    Debug.LogWarning($"未找到化妆品: {instanceData.cosmeticID}");
                    continue;
                }
                if (cosmeticSO.prefab == null)
                {
                    Debug.LogWarning($"化妆品 {instanceData.cosmeticID} 没有预制件！");
                    continue;
                }
                GameObject cosmeticObj = Instantiate(cosmeticSO.prefab, cosmeticContainer);
                cosmeticObj.transform.position = instanceData.position;
                cosmeticObj.transform.rotation = Quaternion.Euler(0, 0, instanceData.rotation);
                cosmeticObj.name = $"{cosmeticSO.itemName}_Layer{instanceData.layer}";
                SpriteRenderer sr = cosmeticObj.GetComponent<SpriteRenderer>();
                if (sr != null)
                {
                    sr.sortingOrder = instanceData.layer;
                }
                CosmeticEditorItem editorItem = cosmeticObj.GetComponent<CosmeticEditorIte
m>();
                if (editorItem == null)
                {
                    StackingItem stackingItem = cosmeticObj.GetComponent<StackingItem>();
                    if (stackingItem == null)
                    {
                        editorItem = cosmeticObj.AddComponent<CosmeticEditorItem>();
                        editorItem.SetData(instanceData, cosmeticSO);
                    }
                    else
                    {
                        stackingItem.Initialize(instanceData, cosmeticSO, this);
                    }
                }
                else
                {
                    editorItem.SetData(instanceData, cosmeticSO);
                }
                activeCosmeticObjects.Add(cosmeticObj);
                successCount++;
                Debug.Log($"✓ 加载化妆品 [{successCount}]: {cosmeticSO.itemName}, 位置: {instanceData.position}, 旋转: {instanceData.rotation:F4}°, 层级: {instanceData.layer}");
            }
            Debug.Log($"<color=green>已加载 {activeCosmeticObjects.Count} 个化妆品到场景中（成功: {successCount}/{instances.Count}）</color>");
        }
        public void ClearAll()
        {
            foreach (var obj in activeCosmeticObjects)
            {
                if (obj != null)
                {
                    Destroy(obj);
                }
            }
            activeCosmeticObjects.Clear();
            HideDragPreview();
            Debug.Log("堆叠区已清空");
        }
        public List<CosmeticInstanceData> GetAllInstances()
        {
            List<CosmeticInstanceData> instances = new List<CosmeticInstanceData>();
            Debug.Log($"<color=cyan>GetAllInstances 被调用, activeCosmeticObjects 数量: {activeCosmeticObjects.Count}</color>");
            foreach (var obj in activeCosmeticObjects)
            {
                if (obj == null)
                {
                    Debug.LogWarning("发现空的化妆品对象，跳过");
                    continue;
                }
                var editorItem = obj.GetComponent<CosmeticEditorItem>();
                if (editorItem != null)
                {
                    editorItem.UpdatePositionToData();
                    var data = editorItem.GetInstanceData();
                    instances.Add(data);
                    Debug.Log($"  - 获取数据: {data.cosmeticID}, 层级: {data.layer}, 旋转: {data.rotation:F1}°");
                    continue;
                }
                var stackingItem = obj.GetComponent<StackingItem>();
                if (stackingItem != null)
                {
                    stackingItem.UpdatePositionToData();
                    var data = stackingItem.GetInstanceData();
                    instances.Add(data);
                    Debug.Log($"  - 获取数据: {data.cosmeticID}, 层级: {data.layer}, 旋转: {da
ta.rotation:F1}°");
                    continue;
                }
                Debug.LogWarning($"化妆品对象 {obj.name} 没有 CosmeticEditorItem 或 Sta
ckingItem 组件！");
            }
            Debug.Log($"<color=green>GetAllInstances 返回 {instances.Count} 个化妆品数据<
/color>");
            return instances;
        }
        public List<GameObject> GetAllCosmeticObjects()
        {
            return new List<GameObject>(activeCosmeticObjects);
        }
        public void UpdateAllItemsClickableState()
        {
            var instances = GetAllInstances();
            foreach (var obj in activeCosmeticObjects)
            {
                if (obj == null) continue;
                var editorItem = obj.GetComponent<CosmeticEditorItem>();
                if (editorItem != null)
                {
                    bool isClickable = IsItemClickable(editorItem.GetInstanceData(), instances);
                    editorItem.SetClickable(isClickable);
                    continue;
                }
                var stackingItem = obj.GetComponent<StackingItem>();
                if (stackingItem != null)
                {
                    bool isClickable = IsItemClickable(stackingItem.GetInstanceData(), instance
s);
                    stackingItem.SetClickable(isClickable);
                }
            }
        }
        private bool IsItemClickable(CosmeticInstanceData item, List<CosmeticInstanceData> allI
tems)
        {
            if (item == null) return false;
            foreach (var other in allItems)
            {
                if (other.instanceID == item.instanceID)
                    continue;
                if (other.layer > item.layer)
                {
                    float distance = Vector3.Distance(item.position, other.position);
                    if (distance < 1.5f)
                    {
                        return false;
                    }
                }
            }
            return true;
        }
        public void CreateCosmeticFromLibrary(CosmeticItemSO cosmeticSO, Vector3 worldPosi
tion)
```

```
        {
            if (cosmeticSO == null || cosmeticSO.prefab == null)
            {
                Debug.LogError("化妆品数据或预制件为空！");
                return;
            }
            var instanceData = new CosmeticInstanceData
            {
                cosmeticID = cosmeticSO.cosmeticID,
                position = worldPosition,
                rotation = Random.Range(0f, 360f),
                layer = GetNextAvailableLayer(),
                isLock = false,
                instanceID = System.Guid.NewGuid().ToString()
            };
            GameObject cosmeticObj = Instantiate(cosmeticSO.prefab, cosmeticContainer);
            cosmeticObj.transform.position = worldPosition;
            cosmeticObj.transform.rotation = Quaternion.Euler(0, 0, instanceData.rotation);
            cosmeticObj.name = $"{cosmeticSO.itemName}_Layer{instanceData.layer}";
            SpriteRenderer sr = cosmeticObj.GetComponent<SpriteRenderer>();
            if (sr != null)
            {
                sr.sortingOrder = instanceData.layer;
            }
            CosmeticEditorItem editorItem = cosmeticObj.GetComponent<CosmeticEditorItem>
();
            if (editorItem == null)
            {
                var stackingItem = cosmeticObj.GetComponent<StackingItem>();
                if (stackingItem == null)
                {
                    editorItem = cosmeticObj.AddComponent<CosmeticEditorItem>();
                    editorItem.SetData(instanceData, cosmeticSO);
                }
                else
                {
                    stackingItem.Initialize(instanceData, cosmeticSO, this);
                }
            }
            else
            {
                editorItem.SetData(instanceData, cosmeticSO);
            }
            activeCosmeticObjects.Add(cosmeticObj);
            UpdateAllItemsClickableState();
            Debug.Log($"从库中创建化妆品: {cosmeticSO.itemName} at ({worldPosition.x:F2}, {
worldPosition.y:F2}), 旋转: {instanceData.rotation:F1}°");
        }
        private int GetNextAvailableLayer()
        {
            int maxLayer = 0;
            foreach (var obj in activeCosmeticObjects)
            {
                if (obj == null) continue;
                var editorItem = obj.GetComponent<CosmeticEditorItem>();
                if (editorItem != null)
                {
                    var data = editorItem.GetInstanceData();
                    if (data != null && data.layer > maxLayer)
                        maxLayer = data.layer;
                }
                var stackingItem = obj.GetComponent<StackingItem>();
                if (stackingItem != null)
                {
                    var data = stackingItem.GetInstanceData();
                    if (data != null && data.layer > maxLayer)
                        maxLayer = data.layer;
                }
            }
            return maxLayer + 1;
        }

        public void ShowDragPreview(CosmeticItemSO cosmeticSO, Vector3 worldPosition)
        {
            if (dragPreviewPrefab == null || cosmeticSO == null) return;

            if (dragPreviewInstance == null)
            {
                dragPreviewInstance = Instantiate(dragPreviewPrefab, cosmeticContainer);
            }
            dragPreviewInstance.SetActive(true);
            dragPreviewInstance.transform.position = worldPosition;
            SpriteRenderer sr = dragPreviewInstance.GetComponent<SpriteRenderer>();
            if (sr != null && cosmeticSO.icon != null)
            {
                sr.sprite = cosmeticSO.icon;
                sr.color = new Color(1, 1, 1, 0.5f);
            }
        }
        public void ShowDragPreview(Sprite sprite, Vector3 worldPosition)
        {
            if (dragPreviewPrefab == null) return;

            if (dragPreviewInstance == null)
            {
                dragPreviewInstance = Instantiate(dragPreviewPrefab, cosmeticContainer);
            }

            dragPreviewInstance.SetActive(true);
            dragPreviewInstance.transform.position = worldPosition;

            SpriteRenderer sr = dragPreviewInstance.GetComponent<SpriteRenderer>();
            if (sr != null)
            {
                sr.sprite = sprite;
                sr.color = new Color(1, 1, 1, 0.5f);
            }
        }
        public void HideDragPreview()
        {
            if (dragPreviewInstance != null)
            {
                dragPreviewInstance.SetActive(false);
            }
        }
        public void SetItemDragging(bool isDragging)
        {
            if (isDragging)
            {
                Debug.Log("化妆品拖动开始");
            }
            else
            {
                Debug.Log("化妆品拖动结束");
                UpdateAllItemsClickableState();
            }
        }
        public void UpdateInstanceData(CosmeticInstanceData instanceData)
        {
            if (instanceData == null)
            {
                Debug.LogError("实例数据为空！");
                return;
            }
            Debug.Log($"更新实例数据: {instanceData.cosmeticID}, 新位置: ({instanceData.posit
ion.x:F2}, {instanceData.position.y:F2}), 旋转: {instanceData.rotation:F1}°");
        }
        public bool HasCosmetics()
        {
            return activeCosmeticObjects.Count > 0;
        }
    }
}
using UnityEngine;
using UnityEngine.EventSystems;
using MakeupPuzzle.Core;
namespace MakeupPuzzle.EditorLevel
{
    public class StackingItem : MonoBehaviour
    {
        private CosmeticInstanceData instanceData;
        private CosmeticItemSO cosmeticSO;
        private StackingAreaUI stackingArea;
        private bool isDragging = false;
        private Vector3 dragOffset;
        private Camera mainCamera;
        private SpriteRenderer spriteRenderer;
        private Color originalColor;
        private void Awake()
        {
            mainCamera = Camera.main;
            spriteRenderer = GetComponent<SpriteRenderer>();

            if (spriteRenderer != null)
            {
                originalColor = spriteRenderer.color;
            }
        }
        public void Initialize(CosmeticInstanceData data, CosmeticItemSO so, StackingAreaUI ar
ea)
        {
            instanceData = data;
            cosmeticSO = so;
            stackingArea = area;
        }
        private void OnMouseDown()
        {
            if (mainCamera == null) return;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(Input.mousePosition);
            mouseWorldPos.z = 0;
            dragOffset = transform.position - mouseWorldPos;
            isDragging = true;
            if (spriteRenderer != null)
            {
                spriteRenderer.color = Color.yellow;
            }
            if (stackingArea != null)
            {
                stackingArea.SetItemDragging(true);
            }
            Debug.Log($"开始拖动: {cosmeticSO?.itemName}");
        }
        private void OnMouseDrag()
        {
            if (!isDragging || mainCamera == null) return;
            Vector3 mouseWorldPos = mainCamera.ScreenToWorldPoint(Input.mousePosition);
            mouseWorldPos.z = 0;
            transform.position = mouseWorldPos + dragOffset;
        }

        private void OnMouseUp()
        {
            if (!isDragging) return;
            isDragging = false;
            UpdatePositionToData();
            if (stackingArea != null && instanceData != null)
            {
                stackingArea.UpdateInstanceData(instanceData);
                stackingArea.SetItemDragging(false);
            }
            UpdateVisual();
            Debug.Log($"拖动结束: {cosmeticSO?.itemName}, 新位置: ({transform.position.x:F
2}, {transform.position.y:F2}), 旋转: {instanceData.rotation:F1}°");
        }
        public void UpdatePositionToData()
        {
            if (instanceData != null)
            {
                instanceData.position = transform.position;
                instanceData.rotation = transform.rotation.eulerAngles.z;
            }
        }
        public void SetClickable(bool clickable)
        {
            if (spriteRenderer != null)
            {
                if (clickable)
                {
                    spriteRenderer.color = originalColor;
                }
                else
                {
                    spriteRenderer.color = originalColor * 0.6f;
                }
            }
        }
        private void UpdateVisual()
        {
            if (spriteRenderer != null)
            {
                spriteRenderer.color = originalColor;
            }
        }
        public CosmeticInstanceData GetInstanceData()
        {
            return instanceData;
        }
    }
}
```