Here is your \*\*revised, final, clean, and structured design document\*\*, now including:

✅ \*\*Removal of non-existent LevelManager mention\*\*

✅ \*\*Addition of a SaveLoadManager\*\* to persist \*\*all critical game states\*\*, with:

\* When to call `Save()`

\* What data structures it manages

---

## 📄 \*\*🍳 Kitchen Chaos – Upgrade & Save System Design Document (Revised)\*\*

HelperType :

public enum HelperType

{

None,

PlateHelper,

DeliveryHelper,

CookHelper

}

### ✅ \*\*1️⃣ Counter Types\*\*

Defines all your counters consistently across systems:

```csharp

public enum CounterType

{

CuttingCounter,

OvenCounter,

FryingCounter,

ClearCounter,

DeliveryCounter,

ContainerCounter

// Extend as needed

}

```

---

### ✅ \*\*2️⃣ CounterLevelSO\*\*

Stores \*\*level data\*\* for each counter type, including properties and prefabs.

```csharp

using UnityEngine;

[CreateAssetMenu(fileName = "New Counter Level", menuName = "Counters/Counter Level")]

public class CounterLevelSO : ScriptableObject

{

public CounterType counterType;

public int level;

public int upgradePrice;

// Counter-specific properties

public float cuttingSpeed; // CuttingCounter only

public float fryingTimeMax; // OvenCounter only

public int capacity; // ContainerCounter only

public GameObject counterPrefab;

}

```

---

### ✅ \*\*3️⃣ SceneConfigSO\*\*

Defines \*\*scene setup, allowed counters, and shop theming\*\*:

```csharp

using UnityEngine;

[CreateAssetMenu(fileName = "New Scene Config", menuName = "Scenes/Scene Config")]

public class SceneConfigSO : ScriptableObject

{

public string sceneName;

public Sprite sceneDisplayImage;

public Material shopUITheme;

public CounterType[] defaultCounters;

public CounterType[] newCountersIntroducedHere;

public HelperSO[] allowedHelpers;

public float priceMultiplier;

}

```

---

### ✅ \*\*4️⃣ HelperSO\*\*

Stores \*\*helper upgrade levels and unlock requirements\*\*:

```csharp

using UnityEngine;

[CreateAssetMenu(fileName = "New Helper", menuName = "Helpers/Helper")]

public class HelperSO : ScriptableObject

{

public string helperName;

public Sprite icon;

public int unlockSceneNumber;

public HelperLevelData[] levels;

}

[System.Serializable]

public class HelperLevelData

{

public int level;

public int upgradePrice;

public float efficiency; // e.g. speed or automation quality

}

```

---

### ✅ \*\*5️⃣ CounterUpgradeManager (static)\*\*

Tracks \*\*global upgrade levels for each counter type\*\*.

```csharp

using System.Collections.Generic;

public static class CounterUpgradeManager

{

private static Dictionary<CounterType, int> counterLevels = new();

public static int GetLevel(CounterType type)

{

if (!counterLevels.ContainsKey(type))

counterLevels[type] = 1;

return counterLevels[type];

}

public static void Upgrade(CounterType type)

{

counterLevels[type] = GetLevel(type) + 1;

}

public static void SetLevel(CounterType type, int level)

{

counterLevels[type] = level;

}

public static Dictionary<CounterType, int> GetAllLevels()

{

return counterLevels;

}

}

```

---

### ✅ \*\*6️⃣ HelperManager (static)\*\*

Tracks \*\*current upgrade level per helper\*\*.

```csharp

using System.Collections.Generic;

public static class HelperManager

{

private static Dictionary<string, int> helperLevels = new();

public static int GetLevel(string helperName)

{

if (!helperLevels.ContainsKey(helperName))

helperLevels[helperName] = 1;

return helperLevels[helperName];

}

public static void Upgrade(string helperName)

{

helperLevels[helperName] = GetLevel(helperName) + 1;

}

public static void SetLevel(string helperName, int level)

{

helperLevels[helperName] = level;

}

public static Dictionary<string, int> GetAllLevels()

{

return helperLevels;

}

}

```

---

### ✅ \*\*7️⃣ SaveLoadManager\*\*

#### \*\*Purpose:\*\*

Handles \*\*saving and loading all critical game data\*\*: coins, owned counters, upgrade levels, helpers.

---

#### \*\*Recommended save structure\*\*

```json

{

"coins": 1200,

"countersOwned": ["CuttingCounter", "OvenCounter"],

"counterLevels": {

"CuttingCounter": 2,

"OvenCounter": 1

},

"helpersOwned": ["PlateBot"],

"helperLevels": {

"PlateBot": 2

}

}

```

---

#### \*\*Sample SaveLoadManager\*\*

```csharp

using System.Collections.Generic;

using UnityEngine;

[System.Serializable]

public class SaveData

{

public int coins;

public List<CounterType> countersOwned;

public Dictionary<CounterType, int> counterLevels;

public List<string> helpersOwned;

public Dictionary<string, int> helperLevels;

}

public static class SaveLoadManager

{

private static string saveKey = "KitchenChaosSave";

public static void SaveGame(int coins, List<CounterType> countersOwned, List<string> helpersOwned)

{

SaveData data = new SaveData();

data.coins = coins;

data.countersOwned = countersOwned;

data.counterLevels = CounterUpgradeManager.GetAllLevels();

data.helpersOwned = helpersOwned;

data.helperLevels = HelperManager.GetAllLevels();

string json = JsonUtility.ToJson(data);

PlayerPrefs.SetString(saveKey, json);

PlayerPrefs.Save();

Debug.Log("Game saved");

}

public static SaveData LoadGame()

{

if (!PlayerPrefs.HasKey(saveKey))

return null;

string json = PlayerPrefs.GetString(saveKey);

SaveData data = JsonUtility.FromJson<SaveData>(json);

// Restore counter levels

foreach (var pair in data.counterLevels)

CounterUpgradeManager.SetLevel(pair.Key, pair.Value);

// Restore helper levels

foreach (var pair in data.helperLevels)

HelperManager.SetLevel(pair.Key, pair.Value);

Debug.Log("Game loaded");

return data;

}

}

```

---

#### 💡 \*\*When to call SaveGame?\*\*

✅ After \*\*every purchase or upgrade\*\*:

\* Buying a new counter

\* Upgrading a counter type

\* Buying or upgrading a helper

\* After finishing a scene (if coins are gained)

This prevents progress loss on crashes or closures.

---

### ✅ \*\*8️⃣ Level Generator (concept)\*\*

Generates the scene layout dynamically:

1. Load \*\*SceneConfigSO\*\*.

2. For each counter in:

\* `defaultCounters`

\* `newCountersIntroducedHere` (if player owns it)

3. Get current global level via `CounterUpgradeManager`.

4. Load prefab for that level.

5. Instantiate at designed slot.

6. Spawn helpers if owned via HelperManager.

---

### ✅ \*\*9️⃣ Shop Upgrade Flow\*\*

1. Shows \*\*owned counter types\*\* + current level.

2. Shows \*\*upgrade cost\*\* (from next CounterLevelSO).

3. On upgrade:

\* Calls `CounterUpgradeManager.Upgrade()`

\* Deducts coins

\* Saves immediately via `SaveLoadManager.SaveGame()`.

4. New counters and helpers are displayed for purchase or upgrade similarly.

---

## 🔒 \*\*Your final step-by-step implementation plan\*\*

✔️ \*\*1. Create CounterType enum\*\*

✔️ \*\*2. Implement CounterLevelSO assets per counter type & level\*\*

✔️ \*\*3. Setup SceneConfigSO per kitchen\*\*

✔️ \*\*4. Define HelperSOs with levels\*\*

✔️ \*\*5. Implement CounterUpgradeManager & HelperManager\*\*

✔️ \*\*6. Build SaveLoadManager\*\*

✔️ \*\*7. Implement Level Generator script\*\*

✔️ \*\*8. Build Shop UI to interact with upgrades and purchases\*\*

---

### ✅ \*\*No LevelManager was referenced or needed.\*\*

This doc is now \*\*clean, finalized, and implementation ready\*\*.

✔️ \*\*Backup this document\*\* in Google Drive or version control.

✔️ Continue development confidently with \*\*no ambiguity\*\*.

---

If you want next:

\* Full \*\*Level Generator C# class template\*\*, say: \*\*“Generate Level Generator class next.”\*\*

\* Full \*\*Shop Upgrade UI system flow\*\*, say: \*\*“Generate Shop Upgrade UI next.”\*\*

I’m ready to assist your next sprint today.

Here is your **full, production-ready LevelGenerator class**, designed for:

✅ **Using SceneConfigSO with CounterSlotData[]**  
✅ **Instantiating each counter** at its **defined position and rotation**  
✅ Fetching **prefab per counter type and level** via your CounterLevelSO database  
✅ Clean, modular, and ready to integrate

using UnityEngine;

public class LevelGenerator : MonoBehaviour

{

[Header("Scene Config")]

public SceneConfigSO sceneConfig;

[Header("Database")]

public CounterLevelDatabase counterLevelDatabase; // See notes below

void Start()

{

GenerateLevel();

}

public void GenerateLevel()

{

if (sceneConfig == null)

{

Debug.LogError("SceneConfigSO is missing from LevelGenerator.");

return;

}

foreach (var slot in sceneConfig.counterSlots)

{

// Check if player owns this counter (if you add ownership checks later)

// For now, assume all slots are populated

// Get current global level for this counter type

int level = CounterUpgradeManager.GetLevel(slot.counterType);

// Get prefab for this counter type and level

GameObject prefab = counterLevelDatabase.GetPrefab(slot.counterType, level);

if (prefab == null)

{

Debug.LogWarning($"Prefab not found for {slot.counterType} at level {level}");

continue;

}

// Instantiate counter at defined position and rotation

Instantiate(prefab, slot.position, slot.rotation);

}

Debug.Log("Level generation complete.");

}

}

**✅ 🗂️ CounterLevelDatabase**

**Purpose:**  
Provides prefabs for each counter type and level by looking up your CounterLevelSO assets.

csharp

CopyEdit

using UnityEngine;

[CreateAssetMenu(fileName = "New Counter Level Database", menuName = "Counters/Counter Level Database")]

public class CounterLevelDatabase : ScriptableObject

{

public CounterLevelSO[] allCounterLevels;

public GameObject GetPrefab(CounterType type, int level)

{

foreach (var so in allCounterLevels)

{

if (so.counterType == type && so.level == level)

{

return so.counterPrefab;

}

}

Debug.LogWarning($"No CounterLevelSO found for {type} at level {level}");

return null;

}

}

**✅ 💡 Scene setup instructions**

1. **Create CounterLevelSO assets** for each counter type and level, assign correct prefabs.
2. **Create CounterLevelDatabase SO**, add all CounterLevelSO assets to its array.
3. **Create SceneConfigSO** for each scene with:

csharp

CopyEdit

public CounterSlotData[] counterSlots;

For each slot define:

* CounterType
* Position (x, y, z)
* Rotation (x, y, z, w) if needed

1. **Assign SceneConfigSO and CounterLevelDatabase** to your LevelGenerator in the scene hierarchy.

**✅ 🔎 CounterSlotData class (final)**

Ensure this is in a separate file or at the top of SceneConfigSO:

csharp

CopyEdit

using UnityEngine;

[System.Serializable]

public class CounterSlotData

{

public CounterType counterType;

public Vector3 position;

public Quaternion rotation;

}

**🔔 Optional enhancements**

✔️ **Ownership check:**  
Before spawning, check if the player owns the counter type (if you add locked counters logic later).

✔️ **Pooling:**  
If you want performance optimization, integrate an object pooling system for counters.

**✅ Your Level Generator is now ready**

When implemented:

* Your scene **automatically populates counters** based on current upgrades and slots.
* Adding new counters only requires updating **SceneConfigSO** and creating their **CounterLevelSO** data.

If you want next:

✅ **Shop Upgrade UI implementation**  
✅ **Helper system integration & logic**  
✅ **AdReward integration design**

…just tell me your next task to keep building efficiently today.