✅ \*\*Continuing from your upgrade system design – implementing the structured architecture now.\*\*

Here is the \*\*final structured system plan\*\* with clear components for your upcoming Unity implementation:

---

## 🏗️ \*\*1. Counter Upgrade System Architecture\*\*

### 🔹 \*\*1.1 CounterType Enum\*\*

Defines all counter types in your game.

```csharp

public enum CounterType

{

CuttingCounter,

OvenCounter,

FryingCounter,

ClearCounter,

DeliveryCounter,

ContainerCounter,

// add future counters here

}

```

---

### 🔹 \*\*1.2 CounterLevelSO\*\*

Each counter level is defined as a ScriptableObject, containing:

\* CounterType

\* Level number

\* Int price

\* Counter-specific properties (speed, capacity, frying time, etc.)

\* Visual prefab for this level

✅ \*\*Example:\*\*

```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; // For CuttingCounter

public float fryingTimeMax; // For FryingCounter

public int capacity; // For ContainerCounter

// etc.

public GameObject counterPrefab; // Prefab with upgraded visuals

}

```

---

### 🔹 \*\*1.3 CounterInstanceData\*\*

Tracks each counter placed in a scene and its state:

```csharp

[System.Serializable]

public class CounterInstanceData

{

public string uniqueID; // For saving/loading each instance

public CounterType counterType;

public int currentLevel;

public Vector3 position;

public Quaternion rotation;

}

```

✅ \*\*Note:\*\* Generate uniqueID on placement for precise save/load mapping.

---

### 🔹 \*\*1.4 CounterDatabaseSO (Optional)\*\*

A database to easily access all CounterLevelSOs in your game for runtime loading and editor tools.

---

### 🔹 \*\*1.5 Milestone Global Boost System\*\*

A manager that:

\* Checks when \*\*all instances of a counter type reach a target level\*\*.

\* Unlocks a \*\*global boost\*\* (e.g. +10% cutting speed for all CuttingCounters).

✅ \*\*Implementation concept:\*\*

\* Maintain a `Dictionary<CounterType, int>` tracking highest milestone unlocked.

\* On upgrading a counter, check if all instances reached the milestone threshold.

\* Apply global stat modifier.

---

## 🏗️ \*\*2. Helper System\*\*

### 🔹 \*\*2.1 HelperSO\*\*

ScriptableObject per helper type with upgrades:

```csharp

using UnityEngine;

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

public class HelperSO : ScriptableObject

{

public string helperName;

public int unlockSceneNumber;

public Sprite icon;

public HelperLevelData[] levels;

}

[System.Serializable]

public class HelperLevelData

{

public int level;

public int upgradePrice;

public float efficiency; // e.g. bot speed, reaction time, etc.

}

```

---

✅ \*\*Plate Bot Helper example:\*\*

\* Level 1: Places plates slowly

\* Level 2: Places plates faster

\* Level 3: Anticipates next order plates proactively

---

### 🔹 \*\*2.2 Helper Behaviours\*\*

Design each helper as a MonoBehaviour script implementing its specific logic, reading its \*\*HelperSO current level data\*\*.

---

## 🏗️ \*\*3. SceneConfigSO\*\*

Defines each scene’s data and restrictions:

```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[] allowedCounters;

public HelperSO[] allowedHelpers;

public CounterInstanceData[] defaultCounters;

public float priceMultiplier; // To increase prices per scene progressively

}

```

---

## 🏗️ \*\*4. Pricing Progression Strategy\*\*

Here is your recommended \*\*scaling table\*\* based on previous discussions:

| Scene | New Counter Base Price | Upgrade Price Multiplier |

| ----- | ---------------------- | ------------------------ |

| 1 | 50 coins | x1.0 |

| 2 | 100 coins | x1.2 |

| 3 | 150 coins | x1.5 |

| 4 | 250 coins | x2.0 |

| 5 | 400 coins | x2.5 |

| 6 | 600 coins | x3.0 |

| 7 | 850 coins | x3.5 |

| 8 | 1200 coins | x4.0 |

✅ Apply \*\*SceneConfigSO.priceMultiplier\*\* dynamically for each scene’s shop pricing.

---

## 🏗️ \*\*5. Data Flow Diagram (textual)\*\*

1. \*\*Player opens Scene Selection →\*\* loads list of SceneConfigSOs

2. \*\*Player selects Scene →\*\* loads that SceneConfigSO:

\* Allowed counters

\* Allowed helpers

\* Scene shop UI theme

\* Default counters

3. \*\*Shop Button Pressed →\*\*

\* Displays allowed counters and upgrades (from CounterLevelSO)

\* Applies price multipliers

\* Shows helpers and their upgrade options

4. \*\*Player purchases upgrade →\*\*

\* Deducts global coins

\* Updates CounterInstanceData or HelperSO level

\* Checks milestone global boost conditions

5. \*\*Gameplay starts →\*\*

\* Loads CounterInstanceData for placements and levels

\* Applies milestone global boosts

\* Activates helpers at their current level

6. \*\*On game end →\*\*

\* Saves updated CounterInstanceData and helper levels persistently.

---

## 🚀 \*\*Next steps\*\*

✅ You are now ready to:

\* Define \*\*each counter type’s levels and properties\*\* systematically

\* Design \*\*helper upgrade levels and behaviors\*\*

\* Build \*\*save/load manager\*\* to persist instance upgrades and helper progress.

---

### 🔔 \*\*When you’re ready\*\*

Tell me:

\*\*“Let’s define counter levels and helper upgrade plans systematically.”\*\*

We will then:

1. Draft the \*\*full levels table\*\* for each counter.

2. Design \*\*helper upgrade trees\*\*.

3. Schedule integration tasks for your upcoming Unity development sprints efficiently.