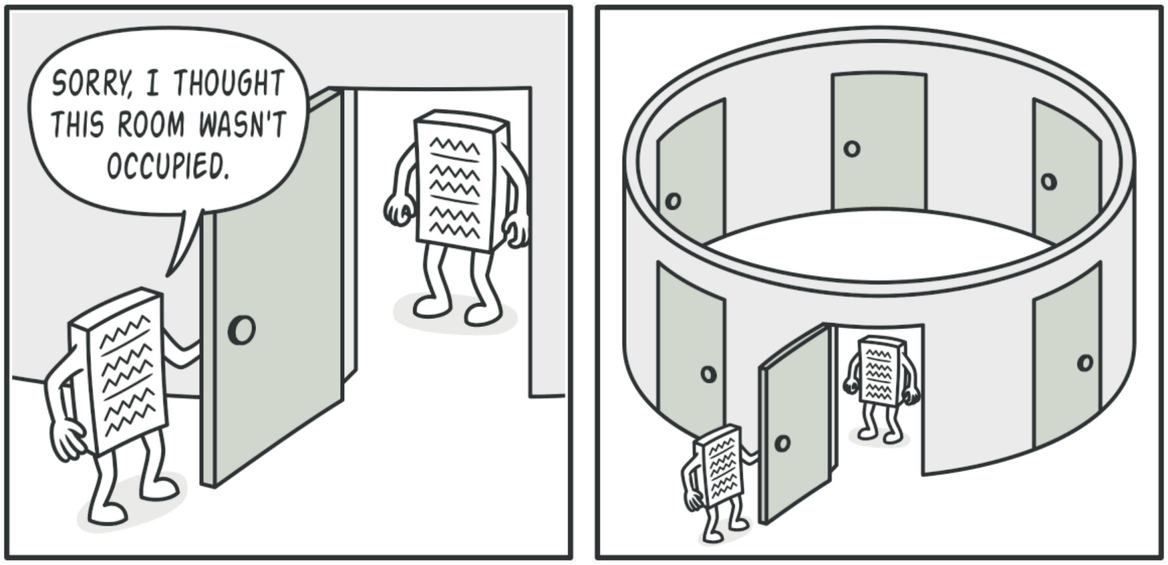
Summary for Task2

Singleton Design pattern:

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.



Ensure that a class has just a single instance.

Provide a global access point to that instance.

Singleton Design pattern in Unity:

This **generic Singleton** implementation in **Unity (C#)** ensures that only **one instance** of a class exists at a time and persists across scene changes.

1. Class Declaration

```
public class SINGLETON<T> : MonoBehaviour where T : MonoBehaviour
```

- This is a **generic class** (`SINGLETON<T>`), meaning it can be used for **any MonoBehaviour-derived class**
- The `where T : MonoBehaviour` constraint ensures that `T` must be a class that **inherits from MonoBehaviour**.

---

2. Static Instance Variable

```
public static T Instance;
```

- `Instance` is a **static variable**, meaning it belongs to the class itself, not an individual object.
- This ensures there is **only one shared instance** across the entire game.

3. Awake Method

```
private void Awake()
{
    RegisterSingleton();
}
```

- `Awake()` is called automatically **when the script is first initialized** in Unity.
- Calls `RegisterSingleton()` to set up the singleton instance.

4. RegisterSingleton Method

```
protected void RegisterSingleton()
{
    Debug.Log("Registering Singleton");

    if (Instance == null)
    {
        Instance = this as T;
        DontDestroyOnLoad(gameObject);
    }
    else
    {
        Destroy(gameObject);
    }
}
```

Execution Steps:

1. **Logs "Registering Singleton"** in the Unity Console.
2. **Checks if `Instance` is null:**
    - If `Instance` is `null`, it means this is the **first** instance.

    - Assigns `this` (the current object) as the singleton `Instance`.
    - Calls `DontDestroyOnLoad(gameObject);`, ensuring the object **persists** across scene changes.
3. **If an instance already exists**:
    - The newly created object **is destroyed** using `Destroy(gameObject);`, preventing duplicates.

Common Use Cases of Singletons in Games

System	Purpose
Game Manager	Handles game states (start, pause, game over)
Audio Manager	Controls background music and sound effects
UI Manager	Manages HUD, health bars, and score display
Input Manager	Centralizes player input handling
Save System	Manages player progress and settings
Networking Manager	Handles multiplayer connections