Notes

In this prototype, I’ve built a solid and flexible game architecture using several key principles:

1. **Modular Design:**
   - I’ve broken down the game into distinct components, each handling a specific aspect like UI management, inventory, or game data. This separation keeps everything organized and makes it easier to maintain or expand in the future.
2. **Event-Driven Communication:**
   - To keep things flexible and decoupled, I use an event-driven system. The GlobalEventManager acts as a hub for game events, like picking up items or changing screens. Components can listen for these events and respond without being tightly linked to each other, which helps in maintaining a clean and modular codebase.
3. **UI Management:**
   - The UISpawner script takes care of creating and managing UI screens. It listens for events that trigger screen changes and keeps track of active screens. This way, UI elements are efficiently handled, and their updates are managed centrally.
4. **Inventory Handling:**
   - InventoryScreenController: Manages the inventory UI, including slot creation and item updates. It listens for events like item pickups and game state changes to keep the UI in sync.
   - ItemSlotController: Manages individual slots within the inventory, handling things like item display and drag-and-drop operations. It interacts with other scripts via events to update slot contents and handle user interactions.
   - ItemSlotStruct: Defines the structure of item slots, including item details and the actions that can be performed on them.
5. **Data Persistence:**
   - InventoryData and SaveSystem: These handle saving and loading inventory data. InventoryData stores information about the items in the inventory, and the SaveSystem manages the actual saving and loading of this data. This ensures that your inventory remains intact across game sessions.
6. **State Management:**
   - I use a state machine to manage different game states. Scripts like GlobalStateMachine and GameState handle transitions between these states, ensuring that the game behaves consistently whether you’re in the main menu, playing, or dealing with a game over.
7. **ScriptableObjects for Data:**
   - ScriptableObjects are used for defining game data like items. This approach allows me to manage data assets easily within Unity, supporting a flexible and data-driven design without hardcoding values directly in the scripts.

The 3D design and overall design were taken from an old prototype created by my friend, Rim Hamdi