***Asset Management***

Assets need to be retrieved and unloaded. This project needs a system and paradigm of when and how asset bundles are retrieved

* Asset Bundling
  + Assets will be organized and built into asset bundles
  + Bundles can/will store prefabs, sprites, audio, etc.
  + Using asset bundles system, asset bundles can be loaded to deliver requested assets by name
  + Retrieving an asset bundle is (probably) not immediate and should probably be handled in an asynchronous process
  + Retrieving a single asset is also (probably) not immediate
    - We can test a synchronous load
    - If the stutter becomes a problem, an asynchronous operation can be explored
* Consider Game Context
  + Scenes
    - Main Menu
    - Cutscene
    - Gameplay Level
  + UI
    - Menus
    - Hud UI
    - Common reusable UI
  + Art/audio assets
* Questions
  + How do we deal with asset dependencies?
    - Use AssetBundle.LoadAsset<AssetBundleManifest>(“AssetBundleManifest”);
    - The AssetBundleManifest stores an array of dependencies for the assets within that bundle
  + How do we determine what assets are needed in each scene/game state?
  + Should the game be run with a state machine? Yes.
* Game State Machine
  + Game State
    - List of states that can be transitioned to
      * Transition Id
      * Game State
    - Required assets list
      * Needed UI Prefabs
      * Needed World Assets Prefabs
      * Needed Scriptable Objects
      * Assets that need to be pooled(?)
      * Loading these asset bundles will try to load other asset dependencies
    - Parent Game State
    - Required Scene (Probably should be on “leaf” child state)
  + Game State Machine Behavior
    - Enter(GameState nextState)
      * Calls Exit on current state
      * Sets new state
      * Prepares next state
    - Exit()
    - Event OnEnter
    - Event OnExit
* Asset Manager
  + Loaded asset bundles should be stored in a dictionary by asset bundle
  + Game State machine should request asset bundle loads
    - Compile a list of assets to be loaded when entering a new state
    - Register assets in the pooled object manager
    - Register UI in the UI Manager
    - Deregister assets in the pooled object manager
    - Deregister UI in the UI Manager
  + Asset Manager will attempt to load the list of assets
  + If successful, the asset should be stored in a dictionary for further use
    - Asset Name
    - The Asset
    - The Asset Type(???)
      * Perhaps multiple dictionaries should be created?
* Pooled Object Manager
  + Should reference the asset manager for asset loading
  + Register/Deregister functionality
  + Use/Return functionality