GameObjects such as buildings, enemies, pickups, etc., are each individually encapsulated as an “Entity”.

**The PlayerAvatar entity is initialised with:**

* Entity Attributes module
  + Health
  + Health Max
  + Character Level = 1
  + Walk Speed
  + Experience Threshold
  + Contact Damage
  + Walk Animation Speed
* Level System module
  + Level-up thresholds
  + Per Level
    - Effect Bonuses
    - Ability Changes
* Player Controller module
  + Starting Abilities
  + Camera elevation range

**Destructible Buildings are initialised with:**

* Entity Attributes module
  + Health
  + Damage Reduction
* CollapseOnDeath module
  + Time to Live
  + Fall Speed
  + Dust Cloud Prefab
* SpawnOnDeath module
  + Spawn Radius = 0
  + Origin Offset = 0,0,0
  + Must or Lottery Spawn with an appropriate Rubble prefab (to suit size)
* SpawnOnDeath module
  + Spawn Radius = 2.5 (as appropriate)
  + Origin Offset
  + Must or Lottery Spawn with Collectables
* Map Tile Preset module (if building is being pre-placed)
  + Size (as appropriate)
* Shake module

**Enemy forces are initialised with:**

* Entity Attributes module
  + Health
* CollapseOnDeath module
  + TTL
  + Fall Speed
* WarpOnce module
* Patrol module
* Some form of fire controller

**Collectable entities are initialised with:**

* Collectable module
  + Rotation speed
  + Collectable By
    - Player, Enemy, etc
  + Contents
    - Experience, etc

## Coding Conventions and Folder Structure.

All scripts are contained under the Assets/Scripts folder. Standard C# naming conventions and namespace-folder conventions apply.

### Major Namespaces

* {global} – Utilities that should be available globally
* AgentAI – “Utility” AI, weighted decision making state machine using collections of Tasks and Actions
* Canvas – HUD/UI controller scripts
* Gameplay – Scripts/Monobehaviours that affect entities directly
* MapGen – Procedural Map Generation (rather, ‘population’) tools
* Scriptables – Classes derived from ScriptableObject

### Folder Structure

Under Assets:

* \_3PP Assets – Any Asset Store downloads
* \_Scenes – All Scenes
* Animations – All animations
* Audio – All sounds
* Gizmos – Utility folder for Unity editor customization, holds things like PNG images for ScriptableObject icons
* Materials – Any non-mesh-specific materials and textures
* Models – Contains all meshes and mesh-specific textures, split into categories:
  + /Model Category – (eg, Buildings, Enemies)
    - /Model Instance – FBX files, mesh-specific material and textures
* Mutators – Specific instances of Mutator-type ScriptableObjects
* Prefabs – Should follow a similar structure to Models, but additionally include UI and effects
* Scripts – Follows C# namespace-folder conventions
* Terrain – Any Unity-Terrain objects (underlying ‘map’ terrains)

## Scene Setup

Minimum Scene requirements:

Player’s Avatar requires:

* PlayerController (script) - Handles user-character motion, idle timer and 3P camera
* EntityAttributes (script) – Tracks Health, Experience, Character Level, etc.
* LevelSystem (script) – Listens to EntityAttributes for Experience changes, updates CharacterLevel and ExperienceThreshold in EntityAttributes.

Canvas (UI Collection):

* HudController (script) – Links from Player’s EntityStats to specified UI elements
* SliderHook (script) – Attached to a UI.Slider object. Watches a specified EntityAttributes module for specified Value and MaxValue changes, updates the UI.Slider element that it’s attached to.

Terrain piece with ‘walkable’ area at Y-level zero.

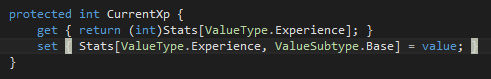
“Monster”-type entities (Player or NPC) **must not** be nested or else Collectable script will not trigger correctly.

All Interactive entities (Buildings, tanks, etc) require an EntityStats (script) to define their Health. Also CollapseOnDeath (script) to fell building and de-spawn the ‘dead’ entity. SpawnOnDeath (script) must be added to generate Pickups from killing other entities.

## System Architecture Overview

### ValueCollection

Maintains a dictionary of Attribute Types and their base and modifier values. Most often this class is wrapped in MonoBehaviours to attach it to objects. The most common example of this is EntityStats, which generally is used to store Health and Experience. ValueCollection raises a ValueChanged event when a value (base or modifier) changes.



### Collectable

This class is attached to entities that can be ‘collected’. It contains a collection of attributes that will be added to the collecting entity’s existing Attributes.

