Developer Manual

Current Technology

SteamVR 2.0.0
Unity 2018.3.11

Introduction

The current tools listed may not be at the versions listed. Unity is considered stable, and using slightly different versions will not affect API usage. If the Unity version is vastly different, Unity will automatically update code or notify you of needed changes.

SteamVR is subject to change, but Valve is projected to maintain older, legacy version of their code via Github.

Before building the project from scratch, make for these two dependencies are installed and configured correctly. You may also need to configure controller bindings - though the settings should carry over from per-created configuration files.

Installation

Installation is simple. First download and install Unity Hub from https://store.unity.com/download for your operating system of choice. Launch Unity Hub and under the Installs tab, download Unity version 2018.3.11. Next download and install GitHub from https://desktop.github.com/. Inside the GitHub application, execute `git clone https://github.com/scouter238/Maze_Runner.git`. Back inside Unity Hub, click open and set the Unity project folder to the folder `Unity Project` within the newly created Maze_Runner directory. After this is done, Locate the start scene and play!

Adding Features

All code and prefabs are stored in their respective folders - Source code is contained within the source folder, and prefabs are contained in the prefab folder. The overall build is very simple in construction but with one major consideration needed: at the start of the game, the player object becomes attached to the root of the DoNotDestroyOnLoad co-scene. This scene exists *with* the current scene, and cannot be destroyed. Objects that are to be moved from scene to scene (e.i.

From the world to the maze or vice versa) will be destroyed unless added to the DoNotDestroyOnLoad co-scene.

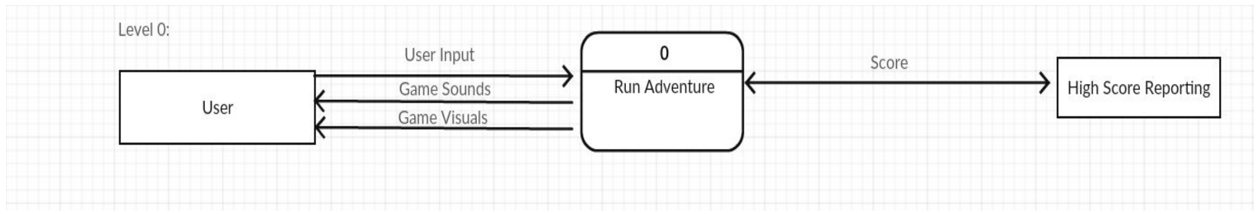
All other features will be mostly independent from the existing structure of code, and will most likely not cause any conflicts. If they do, please take some time to explore the existing content - read comments and code to understand the system.

The Existing Code

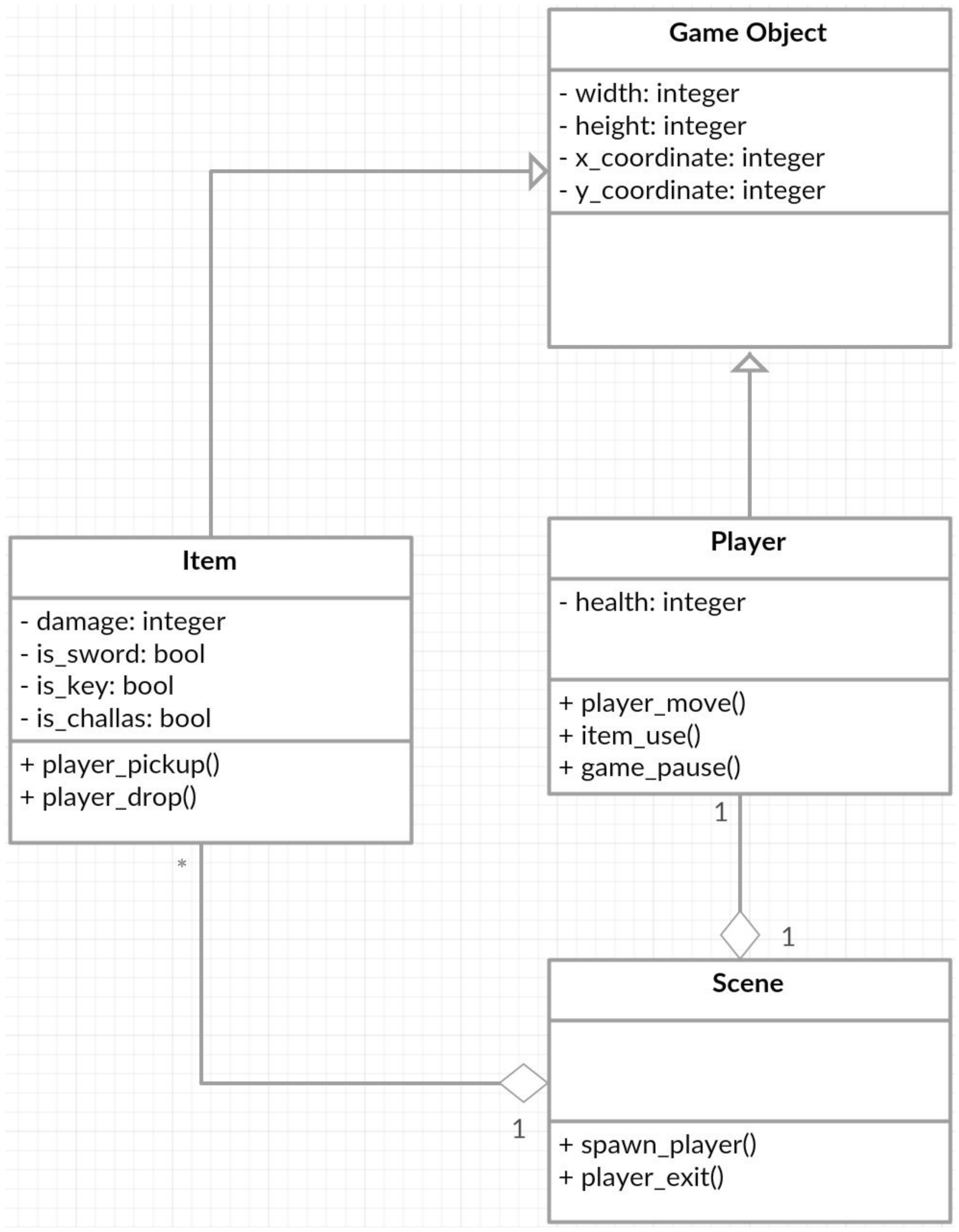
As mentioned above, almost every features' code operates independently of another features' code. This allows the features in the game to exhibit low coupling and high cohesion. From an very high level view, a game manager manages the scenes within our game while each feature manages game objects and scripts within those scenes (e.i A DragonAI script is responsible for the instantiation, moving, attacking… etc features of the dragon game objects within a scene and needs very little information from other features to run).

The original concepts for how our team envisioned coding our features, laid out in the following diagrams, do not accurately represent how the current existing code in the game looks. With more experience using the Unity engine and scripting in C#, we abandoned many aspects of the diagrams we created before development began. To fully comprehend the differences between the beforehand envisioned code and the existing code, feel free to compare the following diagrams with the code existing in the GitHub.

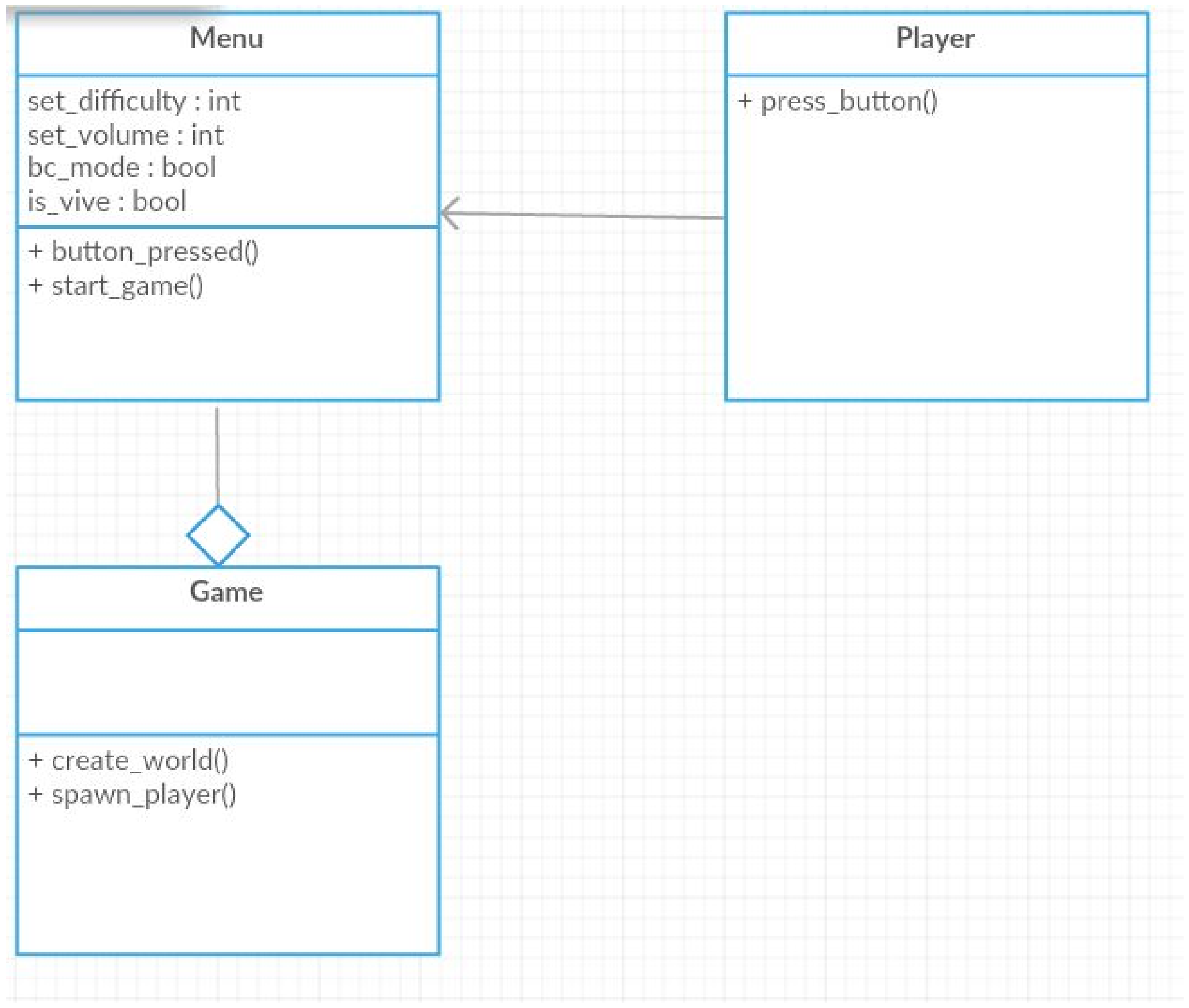
Context Level Diagram



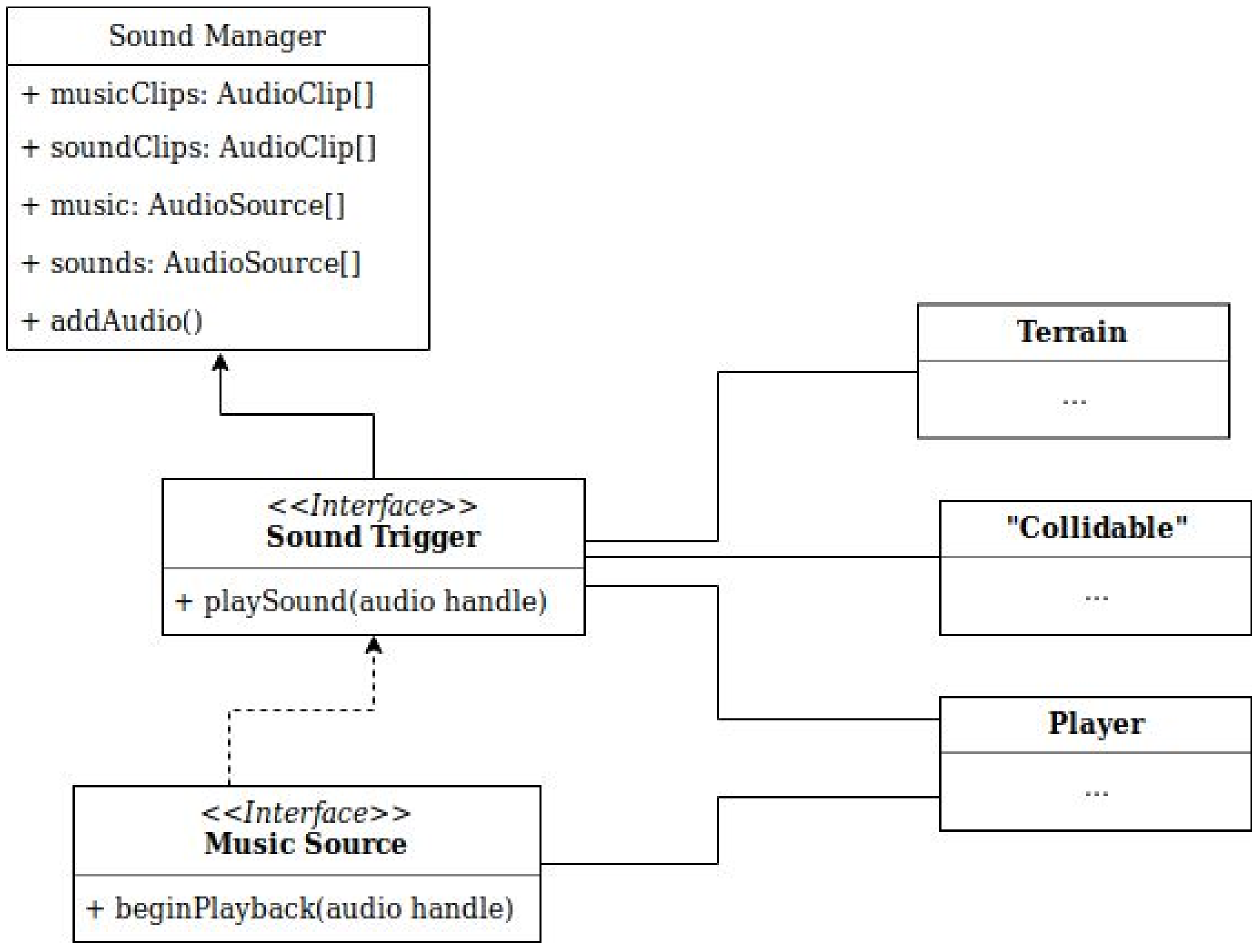
Player Class Diagram



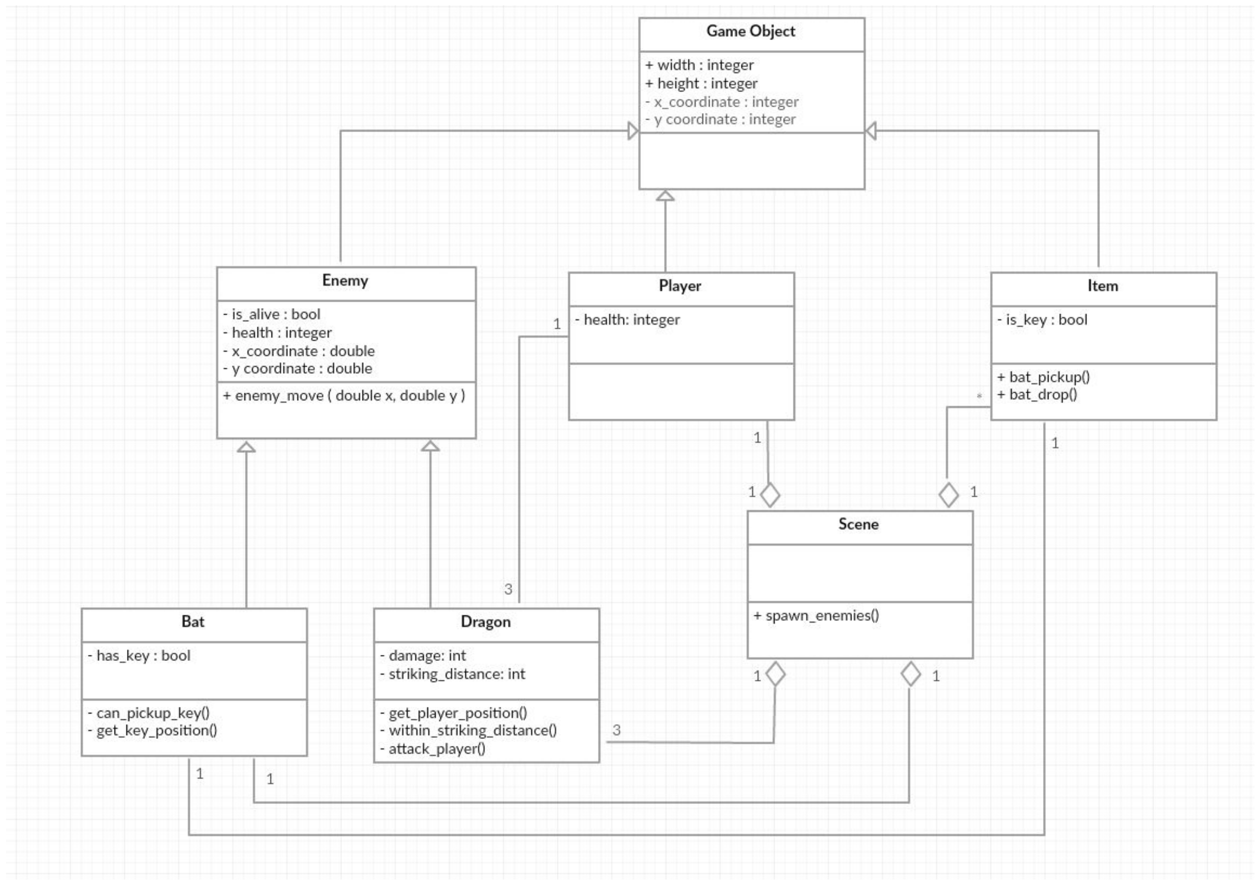
Menu Class Diagram



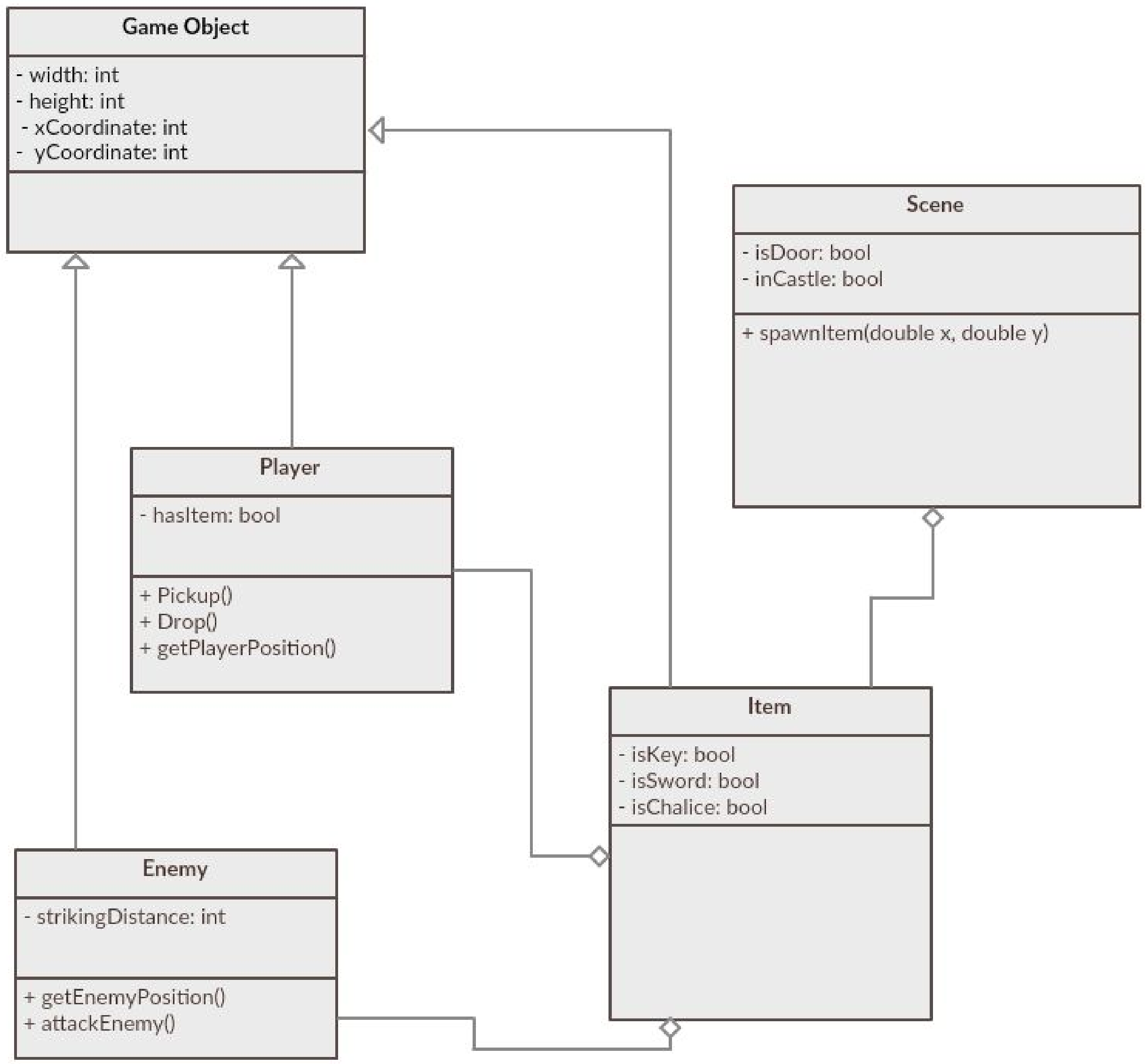
Sound Manager Class Diagram



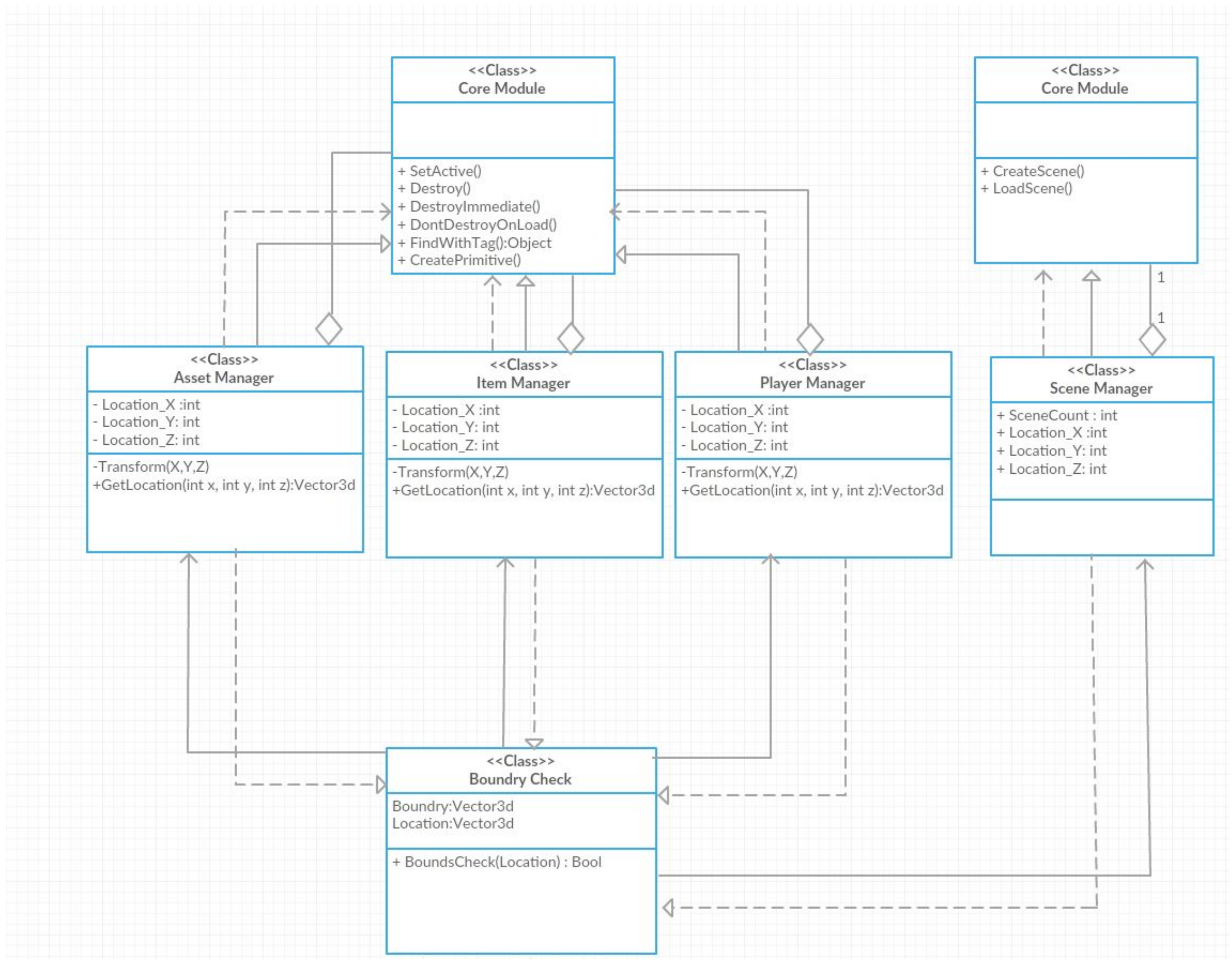
Enemy Class Diagram



Item Manager Class Diagram



Scene Manager Class Diagram



<<Class>>
Core Module
+ SetActive()
+ Destroy()
+ DestroyImmediate()
+ DontDestroyOnLoad()
+ FindWithTag():Object
+ CreatePrimitive()
<<Class>>
Core Module
+ CreateScene()
+ LoadScene()
1
1
<<Class>>
Asset Manager
- Location_X :int
- Location_Y: int
- Location_Z: int
-Transform(X,Y,Z)
+GetLocation(int x, int y, int z):Vector3d
<<Class>>
Item Manager
- Location_X :int
- Location_Y: int
- Location_Z: int
-Transform(X,Y,Z)
+GetLocation(int x, int y, int z):Vector3d
<<Class>>
Player Manager
- Location_X :int
- Location_Y: int
- Location_Z: int
-Transform(X,Y,Z)
+GetLocation(int x, int y, int z):Vector3d
<<Class>>
Scene Manager
+ SceneCount : int
+ Location_X :int
+ Location_Y: int
+ Location_Z: int
<<Class>>
Boundry Check
Boundry:Vector3d
Location:Vector3d
+ BoundsCheck(Location) : Bool