Challenges and Framework

This time I wanted to go over a few of the challenges we’re expecting to face in development of the IsoEngine and how I plan to tackle them. As well I want to start laying out the framework the IsoEngine will work within.

Classically Isometric engines are built on the premise of a 2D tile based system. This allows for very easy data management as each tile can hold information about it’s position, terrain type, and what objects are currently occupying that space. It also lends itself well to pathfinding algorithims as each tile is essentially the node for the pathfinder to traverse and locate the end goal.

As beneificial as these points are, there are also some limitations. As it is a tile based system, artists are generally restricted to creating tile based art which can be repeated over and over to form the map. While scalable and extensible, you can run into a lot of repetition and thus boring locales. Some ways to combat this are to have hand crafted environment maps that sit below the actual tile based grid. A level designer will mark which areas of the map are non-collidable and the player character (and any other objects in the world) operate on an invisible tile based grid that happens to line up with the environment map.

With the tile based system, the common way to handle objects and more importantly object depth sorting is that each object gets placed on a certain tile and then sorted so that objects with a higher screen Y value (towards the bottom of the screen) are drawn last to preserve the illusion of depth. This works perfectly or objects that only take up one tile in terms of their width and depth but what about larger objects like a table, bed, or car?

Beause the object is larger than one tile, but is referenced by the origin of the object, the entire object spans multiple depth values. With objects placed in visual proximity but falling between the depth values, conflicts arrise because the larger object can only have one depth value. It must be either in front or behind the other object despite the face that it needs to be partly behind and partly in front.

It’s incredibly tedious to break up an object into one tile sections so that sorting can occur properly and is a daunting task for an artist to accomplish.

Solutions:

Depth Sorting:

While splitting an large object is simply not feasible from an art perspective, we may be able to split the objects at run time. The first step is to depth sort the objects normally based on their origin screen y value. Again running through the painter’s algorithm. At this point we analyze the screen based bounding boxes of each object. Those objects that have screen based bounding boxes that intersect are then checked to determine if they’re isometric bounding boxes