Ultimate Isometric Toolkit - Quick Guide

for version 2.1

Introduction

This is a quick guide on how to use the Ultimate Isometric Toolkit and understand the basics of the isometric projection. Create a new project, import the Ultimate Isometric Toolkit from the Asset Store Window and create a new scene.

Isometric coordinate system

The coordinate system used in this toolkit is slightly different from the coordinate system we are used to work with in unity. The x,y,z axis are still all perpendicular to each other but are rotated around the y- and x-axis [See Figure 1]. The y-axis remains to be the up axis. To convert a given Vector $\vec{v}(x, y, z)$ from the isometric coordinate system to unity's coordinate system we use

`Isometric.IsoToScreen(v)`

and

`Isometric.ScreenToIso(v)`

vice versa.

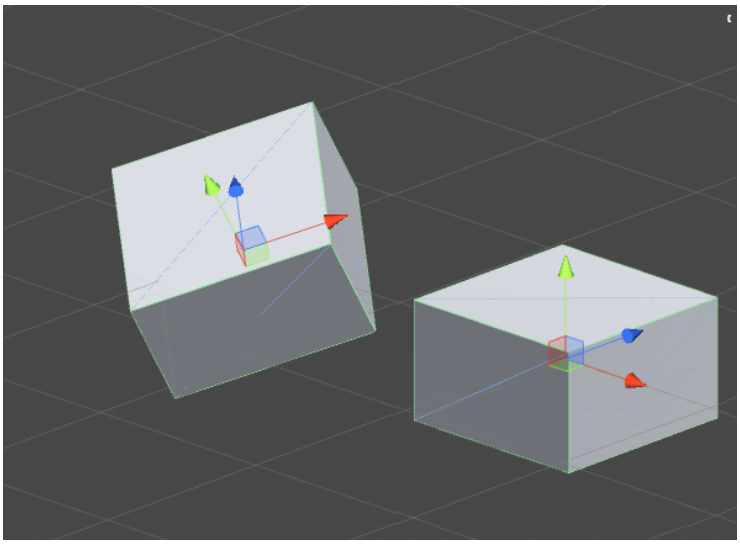

Figure 1: Isometric coordinate system [left], regular unity's coordinate system [right]

The IsoSorting component

Add a new empty GameObject to the scene, add the *IsoSorting* component to it, then add the *Heuristic* component and drag it onto the Sorting Strategy field in the inspector window.

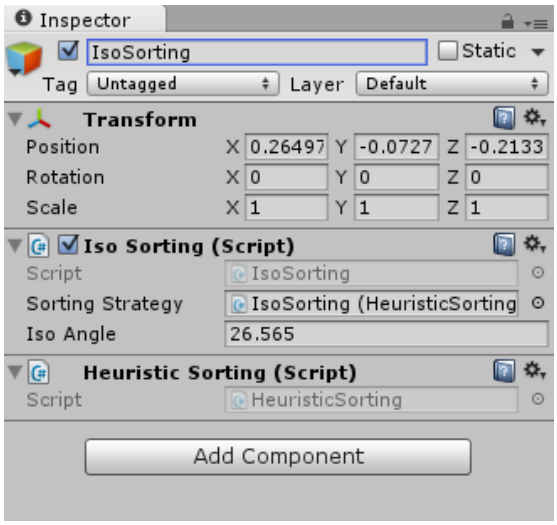

Figure 2: Proper IsoSorting Setup

The IsoAngle α is optional and depends on the sprites you use in your game. Common angles are 26.565 also known also *2:1 iso* and 30 degrees. If you do not know the exact value for your set of sprites you have to fiddle around with the angle until the projection (white bounding box) lines up with the bounds of your sprites.

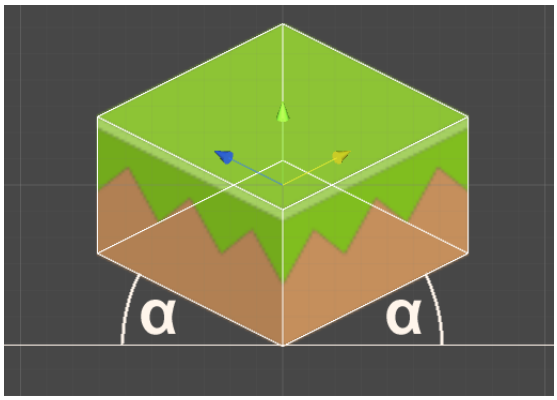

Figure 3: IsoAngle α (here 26.565)

There are multiple ways on how to achieve proper isometric sorting, each of which makes different assumptions about your game. It is therefore crucial to choose the right *sorting strategy*. The isometric toolkit offers three sorting strategies right now.

Heuristic Sorting

Fastest sorting strategy with smallest cpu overhead that requires all sprites to be of relatively equal size. Use for games with the maximum number of sprites that do not differ much in size. Sorting overhead can be neglected.

Tiled Sorting

Second fastest sorting strategy that requires all sprites to be tiled (sliced into 1x1 pieces). Use for games with a large number of sprites. *This is the sorting strategy you would use in most cases.*

Dependency Sorting

Worst in terms of cpu overhead, best in overall compatibility. Allows objects to be of any size. Use for games with up to a couple of hundred sprites.

Please note that perfect isometric sorting in all cases can not be achieved. All sorting strategies have edge cases in which visually correct sorting is impossible. That is not only for this toolkit but for any sorting approach. The provided sorting strategies are just the most common approaches.

The IsoTransform Component

IsoTransforms (former IsoObjects) extend the regular Transform component similar to the RectTransform component. The IsoTransform component replaces the Transform component in the inspector window. Every GameObject in your Scene that has to hold information like a position or size must have an IsoTransform component attached.

Drag a sprite into the scene and add the IsoTransform component to it. [See figure 4]

Show bounds when active will show the rectangular bounds of this GameObject

Position the isometric position of this GameObject

Size the isometric size of this GameObject

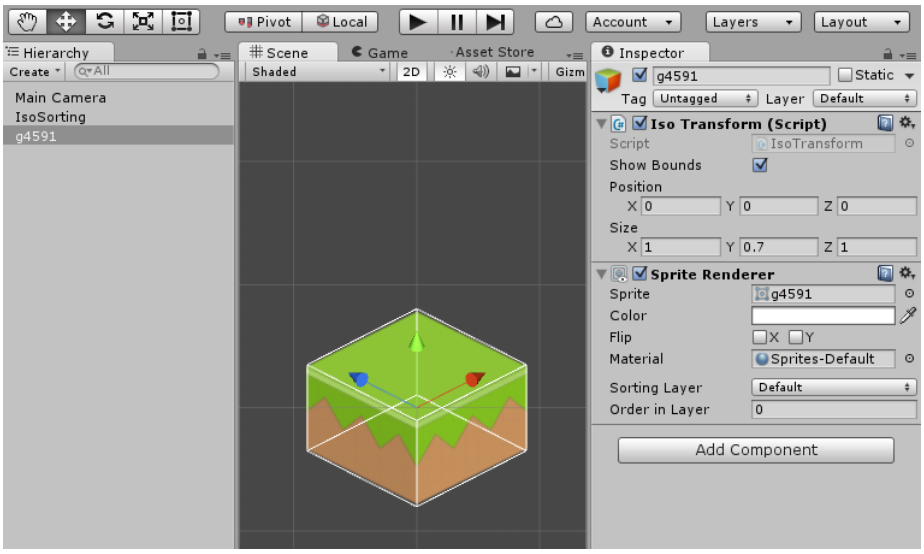

Figure 4: IsoTransform

For ease of use all sprites have properly set up prefabs in the respective *Prefabs* folder that you can add to the scene.

Important: When working with custom sprites make sure to adjust the PixelPerUnit value in your sprite's import settings so that with an IsoTransform component attached the x and z size are both equal to 1 for ground tiles and such.

IsoSnapping

Bring up the IsoSnapping utility tool by pressing CTRL+L. The IsoSnapping tool allows you to evenly place IsoTransforms in your scene.

Auto Snap when active will position evenly across the scene

Snap Value IsoTransforms will snap to the closest multiple of x, y, z

Snap Selection Snaps the current selection in the hierarchy

Converting Mouse/Touch Inputs

A common use case is how to get the object that was clicked/dragged/touched by some user input. The process is very much the same for all kinds of inputs because they all refer to a screen space point. We do the same as we would normally do in Unity (see [here]). We add an IsoCollider to all objects that we want to raycast

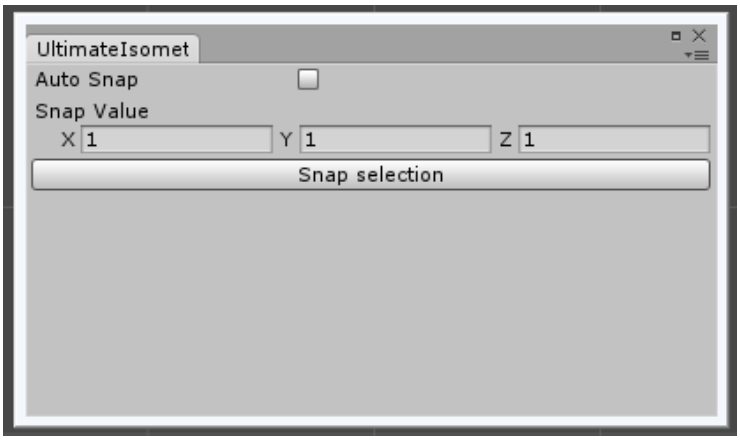

Figure 5: IsoSnapping tool

against in our scene. We then use [ScreenPointToRay] to construct a ray from our input. Since we are dealing with a different coordinate system in isometric perspective we have to rotate both the ray's origin and direction. The Isometric class offers some utility functions to construct such a ray. We then raycast against our scene using the IsoPhysics class. If an IsoCollider was hit we can get its name, distance, hitpoint, etc.

```
// Update is called once per frame
void Update () {

        //do an isometric raycast on left mouse click
        if (Input.GetMouseButtonDown(0)) {

            //mouse input ray in isometric coordinate system
            var isoRay = Isometric.ScreenSpaceToIsoRay(Input.mousePosition);

                IsoRaycastHit isoRaycastHit;
                if (IsoPhysics.Raycast(isoRay, out isoRaycastHit)) {
                        Debug.Log("we hit " + isoRaycastHit.Collider.name
                        + " at " + isoRaycastHit.Point);
                }
        }

}
```

Additional Resources

User Documentation Extended documentation

Youtube Youtube channel with additional tutorials

Doxygen Online class overview

Trello Add your feature requests, bug reports, etc.