## Existing Scenes and Packages

Scenes

* PlatformerWorld - An initial setup for a basic platformer. The cube moves using the Horizontal and Jump axes as defined in the project input settings. (Defaults are A/Left Arrow, D/Right Arrow, and Space.)
* PolygonWorld - An assignment meant for another class, so likely of limited use. The circle travels along a path defined by mouse clicks.
* RPGBattle - A secondary scene that is loaded when required by RPGWorld. When loaded, a turn-based RPG battle is initiated - modeled after late 80s to mid 90s games such as Final Fantasy, EarthBound, Phantasy Star, Dragon Warrior / Dragon Quest, etc. Enemy monsters take turns randomly attacking player characters. Player characters take turns by selecting an ability with the mouse, and then clicking on an enemy (or a player, if you're into losing). All actions are recorded into a log structure, which can be saved to disk.
* RPGWorld - Runs a pre-scripted, linear story using contextual clicks as input. Clicking on the ground will move the player character to the mouse's location. Clicking on an item or NPC will move the character to the object and then attempt to contextually advance the story. If the current story event involves the clicked item or character, the event will occur. The game cannot advance while the dialogue window is open - clicking continue will close it. Pressing I will open the player's inventory. Pressing S will open the script for the current story.
* SkillStudyScenario - Essentially a scene that runs a series of battles independently of RPGWorld with some additional GUI widgets involved to collect extra data from players.

Packages and File System

Where possible, all assets (code, art, data, etc) are grouped into subfolders indicating the scene and, in the case of code, package they belong to. There is some overlap, and Unity can't actually attach scripts (MonoBehaviors) that belong to anything other than the default namespace, so this organization isn't perfect.

* AI - Assets that are generated and used by the Rain{ONE} package.
* Art Assets - All raw textures, materials, and animated sprite data.
* Code Assets - Scripts and libraries, further broken down into rough packages. All files are written in C#.
* Data Assets - XML files containing domain information regarding creatures, abilities, items, and stories. Altering these will change the properties of available domain data in RPGBattle and RPGWorld.
* [Necromancer GUI](http://u3d.as/content/ironbound-studios/necromancer-gui/1Ax) - Third party (free) GUI skin used by RPGWorld and RPGBattle. Attach the GUITestScript (unfortunately Javascript) to an object for a demo of all its features.
* [Orthello](http://www.wyrmtale.com/products/unity3d-components/orthello) - Third party (free) sprite package. The depth field for sprites has been converted from int to float for the purposes of strong-arming sprites to render in a plane other than x/y - if you choose to update the package and wish to use sprites with the Common.SpriteProjector script, you'll need to reimplement this change. Documentation is available: [user manual](http://www.wyrmtale.com/orthello) and [class reference](http://www.wyrmtale.com/ref/orthello/annotated.html).
* Physics Assets - Contains materials defining physically simulated properties of objects. Only the PlatformerWorld and PolygonWorld scenes use these.
* Prefabs - All prefabs in the system are contained here. A prefab is kind of like a semi-instantiated GameObject and functions as a template for creating copies of itself. See the [Prefab documentation page](http://docs.unity3d.com/Documentation/Manual/Prefabs.html) for more information.
* [RAIN](http://rivaltheory.com/product) - Contains the Rain{ONE} third party (not-free) editor and runtime libraries. Documentation is available [on their website](http://support.rivaltheory.com/documentation) - a link is also available through the editor's menu tree: RAIN >> Help >> Read The Documentation
* Standard Assets - Any standard packages that come with Unity import their assets to this location. The only standard asset currently in the project is Terrain.
* Terrain Assets - Contains the generated terrain data for RPGWorld's terrain object.

## Project Settings

Editor Settings - Edit >> Project Setting >> Editor

The version control mode is currently set to "Meta Files," which vastly simplifies the project file system and makes it feasible to use external version control. If you make a new project, you'll probably want to make sure this is enabled. [Documentation](http://docs.unity3d.com/Documentation/Manual/ExternalVersionControlSystemSupport.html).

Input Settings - Edit >> Project Settings >> Input

Defines all the virtual axes and buttons use by all scenes in the project. The current project uses the default settings, which have way more axes/buttons than are actually used. If scripts within the system use these axes/buttons rather than checking actual keys, changing them here will allow you to alter input settings without changing code. See [the documentation page](http://docs.unity3d.com/Documentation/Manual/Input.html) for more details.

Layer Settings- Edit >> Project Settings >> Tags

Defines the tags and layers available to all scenes in the project. These can then applied to any object in a scene in both the Tag and Layer fields at the top of the object's inspector. Layers can be used to cull objects from the scene when rendering or making calls to the physics engine. Tags are somewhat less useful, but can still make it easier to find objects in your scene, particularly when scripts are used to create GameObjects.

Existing code relies on Layers 8 - 11 to function properly.

[Layer documentation](http://docs.unity3d.com/Documentation/Components/Layers.html). [Tag documentation](http://docs.unity3d.com/Documentation/Components/Tags.html).

Unity Preferences - Edit >> Preferences

General settings for the editor as a whole. Notable options are "General >> Always Show Project Wizard" and "External Tools >> External Script Editor." The project wizard will simplify your workflow a bit if you have multiple projects on the machine and are consistently switching between them. Do yourself a favor and set the external script editor to VisualStudio and only use MonoDevelop (which installs with Unity) if you're on a Mac. You'll still be forced to do any breakpoint debugging via MonoDevelop, but in addition to VS being a generally superior IDE, almost all of the code within the Code Assets folder is tagged with VS Intellisense.

## Scene Initialization

Empty Object

Unity uses a [component-based architecture](http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/) and is thus unable to execute any code you've written unless it's attached to a GameObject as a class that inherits from MonoBehavior (also referred to as a script). While convenient in most cases, it doesn't provide an intuitive way to execute code that doesn't need any representation within the virtual world. You can use GameObject >> Create Empty to create an object with no components attached to it. Place it at the origin of the scene and attach all your managers, factories, generators, pathers, or whatever you need to this object.

Existing code interchangeably names these as "Root" or "Null" objects. Obviously, this is merely a suggestion - everything will still work as intended if you scatter these scripts randomly over all the objects in your scene.

Camera

Every new scene comes with a main camera object. When rendering 2D, ensure that the projection is set to Orthographic. The culling mask can be used to view only selected objects in the scene, but for most purposes this should be set to "Everything."

A simple camera script can be found in Common.FollowCam which tracks a given GameObject in the scene. It supports zooming and can be used to track object on either the x/y or x/z plane.

Orthello

If you intend to use the sprite functionality present in Orthello, you'll need to do a little work to get the scene set up. [Orthello documentation](http://www.wyrmtale.com/orthello/starting-a-scene) explains this better than this document can.

Recast

A component provided by Rain{ONE} that automatically generates navigation grids for the purpose of pathing. If you intend to use the pathing solution provided by Rain{ONE}, this is probably your best option - as the other methods involve authoring waypoints by hand. Again, the [Rain{ONE} user manual](http://support.rivaltheory.com/usermanual.pdf) explains this better - see the section on Recast, page 53.

## Creating a Character

AI Agent - RAIN >> Create AI

The Rain{ONE} package provides a nice base agent that should save you some unwanted work. You can use either SimpleMinds or BehaviorTreeMinds to run an agent. Simple Minds contain multiple behaviors, which all execute each frame while BehaviorTreeMinds provide a method for authoring, well, BehaviorTrees. If you want the most control over your AI, you should probably use a SimpleMind. If you don't mind conforming to the structure of a BehaviorTree, BTMinds can probably save you some development time when creating complicated behaviors.

As a quick guide, select the object you want to use as an Agent and RAIN >> Create AI >> (whichever). Make sure that you locate the generated PathManager child object and set it to use a navigation grid rather than the default waypoint collection. Probably a good idea to actually point it at the proper NavGrid file while you're at it.

Full details are available in the [Rain{ONE} user manual](http://support.rivaltheory.com/usermanual.pdf) on page 4.

Moving the Agent

If you've properly configured the PathManager script attached to the agent's child object, moving the agent is relatively easy. In the editor, you can set the "Move Target" to either a specific location in the scene as a Vector3, or to the transform of another object, in which case it will move to that object's position.

During runtime (from the scope of a MonoBehavior), the move target can be set as follows:

GameObject other;

PathManager pather = this.gameObject.GetComponentInChildren<PathManager>();

pather.moveTarget.TransformTarget = other.transform;

An example of this can be found at RPGWorldEngine.Scripts.PlayerController.HandleClick() . A warning: once a PathManger's move target is set, it will continually seek out this position - it isn't cleared upon arrival. As a result, an agent that is teleported (by setting transform.position) will then beginning moving back to its move target.

If you wish to change the movement speed of an agent, alter the "Max Speed" field of the RAINAgent script attached to the agent's root object.

Spriting

See the [relevant area of the Orthello documentation](http://www.wyrmtale.com/orthello/sprites) for how to instantiate and use sprites. Note however, that this version of Orthello will accept float values for the depth field.

However, there is a limitation to Orthello that needs to be worked around if you intend to use terrain objects and sprites in the same project. The Unity terrain package is optimized to exist in the x/z plane and cannot be rotated. As it would happen, Orthello is optimized to exist in the x/y plane and (out of the box) casts the z dimension to an int. Additionally, Orthello overrides (or otherwise hooks into) all the transform fields, making it difficult to solve the issue directly.

To get around this, you'll have to basically create an invisible virtual agent and set a sprite to follow the agent. Create an empty root object to store sprites. Set its Rotation Vector as follows: X 90 Y 0 Z 0. Add whatever sprite prefab you need as a child object of this root node. Add a SpriteProjector script to this newly created sprite object and point the "Follow Target" field at the agent's gameobject. The sprite will now follow the agent's transform. You should probably remove any rendering components attached to the agent's object.

## Creating the Virtual World

Terrain - Terrain >> Create Terrain

The [Unity Terrain documentation](http://docs.unity3d.com/Documentation/Manual/Terrains.html) covers how to work with terrain objects. However, as a shortcut, instantiating your terrain object by using a prefabricated asset, such as the one found in the Terrain Assets folder (or another you make yourself), will save you the step of initializing all its textures again.

Spriting

Placing sprites in the world operates much the same as placing them on characters, but you don't necessarily need to attach the SpriteProjector script to them, as they don't move.