What is ProD?

**Short Introduction:** This package supplies you with several scripts that help you create procedurally generated maps using a 2-D grid system. The package is designed with roguelike development in mind and contains a structure that supports this.

Quickstart?

Go ahead and double click on the **TestScene** provided in the ProD folder. **Press Play** to run the scene in Unity. In this scene you may click on the **DungeonGenerator** game object and press the **Generate** button in the inspector. You won’t be able to press Generate if Unity is not in Play mode. Pressing Generate button will generate a dungeon map that you can navigate using the mini map on the right or numpad keys via the player on the left.

If you have a scene of your own already and you want to use things in your own game, head to the **Prefabs** folder and dump the prefabs you might want to use into your own scene. If you are missing an object or a script in your scene check the **Console Messages**. It usually will tell you what’s wrong.

**For all other things, it is highly recommended that you read and follow the tutorial down below in this manual at least once.**

What’s inside ProD? (In Alphabetical Order)

**Warning! In this free version we excluded most of the scripts from paid version of ProD. The blue-highlighted scripts under this topic are inaccessible in the free version. You may access the highlighted assets only if you buy the paid version of ProD.**

**BaseGenerator.cs** – This is the mother script to all generator scripts. It holds common methods other generator scripts use.

**CameraDragAndZoom.cs -** This is a script that will let you use a camera as a zoomable and scrollable minimap once you place it on said camera.

**CameraManager.cs** – This is an additional script that will help you set game resolution and/or track an in-game player object. It is referenced and used by BaseGenerator.cs type scripts. You may further expand this script to suit your project’s needs or disable it completely if you don’t want the camera to set resolution or track a player object. To get pixel perfect resolution in your roguelike game you may read further below. This script will also let you switch between Orthographic and Perspective camera modes.

**CavernGenerator.cs** – This script allows you to create cavern formations. In case you wish to have a cavern map to play in, you may use this script.

**Cell.cs** - This is the script that contains crucial information about a cell. Type and Address of a cell is stored in here. Cell.cs acts as an ID card for a cell.

**CellManager.cs** – CellManager.cs is placed on prefabs and only and only holds a reference to a Cell.cs script in other words CellManager.cs is only a container for Cell.cs. The reason for this is so that we can create Cell.cs scripts without having to instantiate Game Objects in a scene, which allows us to have maps as data sets and not as Game Objects in a scene. Whenever you want to materialize your map by instantiating Game Objects, you should place a CellManager.cs on your prefab and assign a corresponding Cell from your mapArray.

**CellTypeParameters.cs** - This is actually a struct disguised as a class for ease of use. CellTypeParameters is used between ProD\_Data.cs and ProD\_Window.cs in saving and transferring bulk data about all cell types. Data consists parameters such as name, color, prefab and texture of a cell type. This data is later on called by scripts that instantiate cells.

**CreatePlane.cs** - This script was used to replace high-poly count planes with low-poly count planes in Unity3D. This script is not written by Gray Lake Studios. Author of this script is Michael Garforth. Here’s a link to script’s original source: <http://wiki.unity3d.com/index.php?title=CreatePlane>

**DirectionManager.cs** – This script gives an object 4 directions and ability to alter them for navigation. These directions are used by the generator maps and especially by maze algorithms for traversing the map. (Each cell prefab should have a DirectionManager.cs on it.)

**DungeonGenerator.cs** – This is the main script that contains most algorithms for creating a dungeon map. It will create a 2-D array that contains all map information. This array has references to Cell.cs scripts that make up the rooms, corridors, doors, stairs and all other types of cells. During the execution of this script the map will materialize and game objects will be instantiated in world-space using cell data provided by ProD\_Data. All information regarding the created map is stored in a mapArray variable in DungeonGenerator.cs and is available for use outside itself, publicly.

**Editor Extended Scripts: (BaseGeneratorEditor.cs, CameraManagerEditor.cs...)** – All scripts in the Editor folder with Editor at the end of their name are there to enhance the Inspector interface with buttons, warnings and other layout related elements supported by Editor class of Unity3D.

**MazeGenerator.cs** – This is an isolated part of DungeonGenerator.cs. This script will create a maze that connects every odd tile to each other on the map. In case you wish to have only a maze and not a dungeon with rooms, you may use this script do so.

**OrientationManager.cs** - This singleton script is useful when you have tileable 3-D prefabs for your cells or other prefabs you want to modify or rotate on the go. If you want to use a wall prefab and have it rotate and/or change shape according to its surroundings, this script is the guy. For more detail please read the comments on top of the script.

**OrientationVisualizer.cs** - This script is placed on 3-D prefabs you want modified or rotated by OrientationManager.cs. For example if you want your Door model to rotate according to the Wall cells around it, then you need to place this script on your prefab with the Door model on it.

**PlayerMovement.cs** – This is a template script for you to be able to move a player in the map. It consists of different moving schemes and a simple check for looking at tiles and deciding if they are free to be occupied by player on player’s next move.

**PlayerStats.cs** – This is a simple script for assigning attributes to your player. You may store stats, skills and other attributes here.

**ProD\_Data.cs** - This is the script that holds all information that comes from your ProD Window. You may open the ProD Window by going to Window Tab and clicking on ProD. Upon placing a texture, naming a cell variable or changing the color of a cell in the ProD Window make sure to click the SAVE button. ProD\_Data.cs will permanently store your settings to ProD\_Data.prefab and make it available for you when you generate your map. You may access this data bank through public methods and inquire information about your cell types.

**ProD\_Window.cs** - This script works closely with ProD\_Data.cs and draws the window interface. See ProD\_Data.cs.

**QuadTreeDungeonGenerator.cs** - (This is an initial version that still needs work.) This script generates a dungeon by dividing the given map space into 4 subspaces for every space in it’s domain. The script will continue doing this for the number of quadTreeDepth specified in the script.

**QuadTreeNode.cs** - This a node used by QuadTreeDungeonGenerator.cs. Every node has four child nodes and is a member of a Quadratic Tree used by QuadTreeDungeonGenerator.cs

**Room.cs** - This script is actually a struct that keeps information about a room. It’s being used by generator type classes to store data about all individual rooms in a map so that ProD users can benefit from spawning chests, monsters or events in rooms if they wish to do so.

**VisualGenerator.cs** – This script is another version of DungeonGenerator.cs. The script is made up of co-routines that will run the map generation by using yield statements. This way developers may observe the creation of their map in real-time and feel awesome.

**WebPlayerGUI.cs** - WebPlayerGUI.cs is responsible of the GUI sliders and buttons you might have seen in the webplayer version. It’s a simple interface for other users to see and benefit if they decide to create a simple interface for their own projects.

How do I use this package in my project?

1. **First and foremost, experience the test scene:** We recommend that you open the **TestScene** and press **Play**. You may initiate a map by pressing the **Generate Dungeon** button on **DungeonGenerator** game object’s inspector only and only when you are in Play mode. We advise you to play with the values you see in the inspector to get a feel of what the variables do in DungeonGenerator.cs.
2. **Make sure you know what BaseGenerator.cs and its children do:** BaseGenerator.cs and its children are the heroes of ProD. They contain the most basic methods for map generation. These methods are reusable by any new map generator script you may create. BaseGenerator.cs is thoroughly commented and it’s recommended that you **read at least all of the method names in this class once** before using ProD.
3. **Last, learn how to operate and benefit from using the ProD Window:** ProD Window is there to help you organize your cell types and what textures and prefabs these cell types use. You may **read the ‘a’ part of the example below to get a good understanding of how the ProD Window functions**. Following the example verbatim is, of course, optional.
4. **Here’s an EXAMPLE scenario in which ProD is being used**:
   1. Let’s say you want to create your very own procedural map with a river running through mountains and fields.
      1. First, you should plan out what your algorithms will do. This is the tough part of the job and where you spend time reading about making procedural river generation and more. There are sources online with example codes and explanations. Here’s one I like from *Amit Patel* <http://www-cs-students.stanford.edu/~amitp/gameprog.html>
      2. Once you know enough about what algorithms you will use for your map generator, you should decide what types of cells you will need. In our example we use the following cell types: *Abyss*, *River*, *Mountain* and *Field*.
      3. Considering you don’t have models, textures and no other assets yet: You may open the **ProD Window** from **Windows** drop down menu, located on top in Unity, and set your **Replacement Type** to **Textures**. Place the ProD/Resources/**TopDownTile.prefab** in the **Generic Prefab** field on **ProD Window**. **TopDownTile.prefab** is included in the ProD package to act as a simple prefab with a plane mesh. We will lay down tiles to make our map and this prefab will be used for every cell there is.
      4. Next you should punch in *4* as your **Array Size** for **Cell Types** and write your **Cell Types’** names here. You may write *Abyss*, *River*, *Mountain* and *Field* for each blank field.
      5. Now go ahead and set the **Color** field of every **Cell Type**. We may use *black* for *Abyss*, *blue* for *River*, *red* for *Mountain* and *green* for *Field*. You may also add a texture in the **Texture** field for every **Cell Type** but if you choose not to, the flat texture on **TopDownTile.prefab** will be used instead.
      6. Finally click **SAVE** to permanently save the data you entered!
   2. So far, so good!
      1. We made plans to make our own procedural script.
      2. We extracted relevant data ProD will need.
      3. We walked through basics of storing said data in ProD.
      4. Now, it’s time to learn how we can use this data for our script!
   3. Now, let’s move on and code our script. You are on your own at this step, however, BaseGenerator.cs provides you with useful methods.
      1. Let’s start with creating your class. Go ahead and create a C# script and name it *RiverGenerator* Now double click your RiverGenerator.cs and replace default MonoBehaviour inheritance with BaseGenerator so that you can benefit from some of the generation methods ProD already gives you. Before we move on further, take a look at methods inside BaseGenerator.cs. Least you should do is read the method names.
      2. BaseGenerator.cs includes a variety of methods that help you create, delete, scan and modify your map. Once you skim through the class you will have a better idea if you need a new method or you can use an inherited method in your code.
      3. Next, you should carefully plan your class’ flow. This is the toughest part, as I already stated, since you have to either learn your algorithm from other sources such as books, articles or code snippets or take some time and make your own. In the link I gave you there already is an explanation on how to generate rivers procedurally, so feel free to read and learn from it. <http://www-cs-students.stanford.edu/~amitp/gameprog.html>
      4. Pretty much the rest is on your shoulders!

**F.A.Q.**

1. **Does ProD generate maps in Play Mode or before I press play, in the Editor Mode?**
   1. **ProD can do both**, however, the test scene in the package will only let you generate maps in the play mode, that is after you press Play icon in Unity.
   2. Since ProD can generate maps in 2-D string or Cell arrays without instantiating any game objects it’s possible for you to make maps before you run the Play mode. To do this go ahead and override or rewrite the Generate method in the generation script you’ll be using and omit the instantiation methods.
2. **I generate my map, but I get transparent GameObjects! What’s the problem?**
   1. You should check the alpha values in the Color fields of your CellType entries on ProD Window. If everything looks alright there, you should check the type of shader you are using for your materials.
3. **I can’t get pixel-perfect textures in my game. How do I resolve this issue?** 
   1. Your texture size, your object’s scale and your resolution need to be in sync for getting pixel-perfect images in Unity3D. Follow these steps:
      1. Switch your camera to Orthographic. This will display everything in 2-D.
      2. Go to import settings of your textures and raise their compression to a higher resolution. Your texture’s Filter Type should be Point and its Format should be highest possible.
      3. Power of 2 is your friend. You get warped pixels when they need to split into half. For example I use a 32x32 pixel PNG file that is attached to a prefab with scale of X:32 Y:1 Z:32. I display this in a resolution of 512x512 using 512 as my camera size.
      4. If these don’t help you, google and unity forums are your friend.
4. **What’s your e-mail?**
   1. [**Unity3D@graylakestudios.com**](mailto:Unity3D@graylakestudios.com)
5. **I e-mailed you but you didn’t reply. What's up with that?**
   1. I usually reply immediately. I might be very busy due to the fact that I’m working at my own start-up. Feel free to e-mail me again in case I missed it.

**I want to contact you!**

You may contact us via [**Unity3D@graylakestudios.com**](mailto:Unity3D@graylakestudios.com) and send us your feedback, complaint, praise or questions. We are an independent development studio and we hope to reply to you as soon as we can, however, given the circumstances we may be very busy and reply late depending on our schedule. ProD is being developed slowly and you may follow the latest development news on our blog: <http://produnity.blogspot.nl/>

**Update Log: ProD v 2.13**

* Added sample 3D models and textures.
* Added OrientationManager.cs and OrientationVisualizer.cs.
  + Now you may rotate doors and tile 3D models by modifying them automatically, according to the cells around them. This is also demonstrated in the package with the current 3D models.
* Added QuadTreeNode.cs and QuadTreeDungeonGenerator.cs.
  + Added an unfinished version of QuadTreeDungeonGenerator.cs. This scripts is still under construction.
* Added CameraDragAndZoom.cs
  + Added a script that let’s you make a draggable and scrollable minimap. This is the same script we use in the WebPlayer version of ProD.
* Added WebPlayerGUI.cs
  + Added the GUI script used in making the WebPlayer for users to learn from.
* Changed contents of CameraManager.cs
  + Cleaned some redundant methods and tightened the script.
* Added Room.cs
  + Now you may keep track of the rooms you generate in a dungeon in case you want to spawn things in them.
* Changed contents of BaseGenerator.cs
  + Modified the ReplacePrefabs method and integrated it with OrientationManager.cs script.

**Update Log: ProD v 2.0**

* Changed contents of CameraManager.cs
  + Removed automatic resolution resizing method since it didn’t resize perfectly for all sizes.
  + Added track and lock on player object capability.
  + Added a new resolution setting capability.
* Added ProD Window.
  + Added ProD\_Window.cs, ProD\_Data.cs & CellTypeParameters.cs
  + Now you may set cell types, colors, prefabs and textures in the ProD Window and use them in any map generation. This allows users to generate maps using their own prefabs or textures.
* Removed CellManager.cs
  + ProD\_Data.cs took the responsibility of storing relevant data CellManager.cs had before.
* Changed contents of PlayerMovement.cs
  + Reduced the complexity of script.
  + Added capability to move by using all numpad keys.
  + Added a boolean to switch between only cursor keys and numpad keys.
* Changed contents of all BaseGenerator.cs related classes.
  + Added ConvertUnreachableWallsTo() method that transforms wall tiles if they are surrounded by wall tiles on all 8 grids around them.
  + Cell objects are now generated and immediately set to be children of map objects in scene. These parent objects create a permanent address for access by code. In our case we access them for deletion of old cells on generation of a new map upon an existing one.
  + Added PlacePlayer() so users can place a player object by toggling a boolean in generator scripts.
  + Added a DestroyPlayer() method for destroying the player on map re-generation.
  + Added ReplaceCellsWithPrefabs() method for replacing your cells with textures or prefabs you may set before.
  + Tweaked all scripts for compatibility with ProD Window.
* Added an example tile set for Texture Type Generation.
* Added placeholder prefabs for Prefab Type Generation.