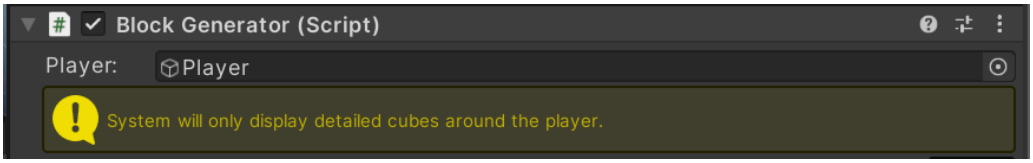
Quick Start

It is very easy to setup Procedural Cube World system:

1. Drag "SoftKitty/ProceduralCubeWorld/Prefabs/CubeWorldGenerator" prefab to your scene.
2. Drag your player gameobject to the "Player" field of "*BlockGenerator.cs*".



3. Call ***BlockGenerator.instance.GenerateRandomWorld()*** `from your script`.

Recommended project setting:

- Player>Other Settings>Color Space: set to "Linear".
  The textures and shaders are designed for Linear color space, it is recommended to set if you would like to have same visual as the preview videos and images in the Asset Store.
- Player>Other Settings>Graphics Jobs: checked
  This will help to achieve better performance in most cases.
- Physics>Layer Colllision Matrix> set the layer of "CubeWorldGenerator" gameobject not collide with itself.
  It is recommended to have "Block Generator" and all its children set as a separated layer represent the ground.

Demo Scene

Demo scene is located in :

"SoftKitty/ProceduralCubeWorld/ExampleScene/DemoScene"

The demo itself is a full simple game with the following mechanics:

- Infinite random generated game world.
- Save/Load the whole game world.
- Build/Delete Cubes.
- Full character control and camera control system.
- Teleport to random place of the world.
- Spawn random number of AI around and follow you.
- Mini map.

Involved package:

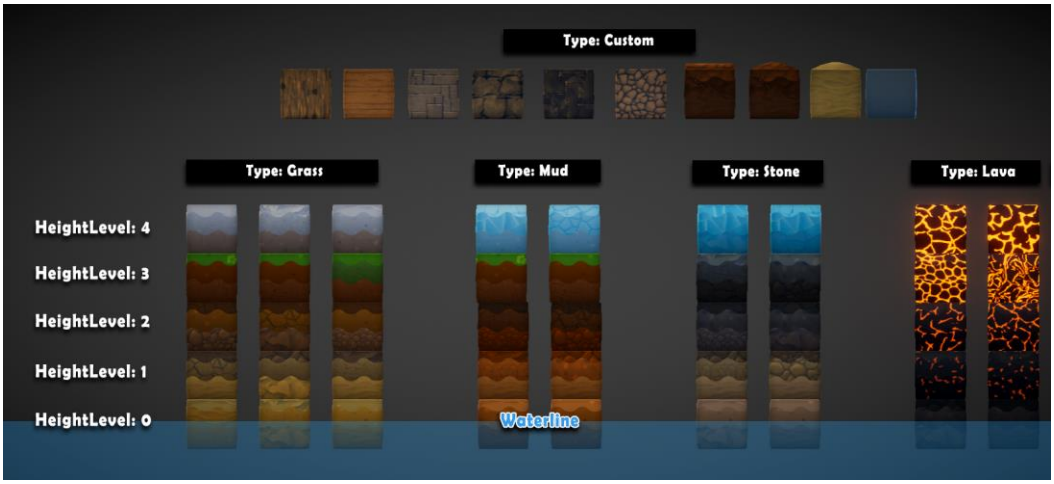
- Unity Post Processing 3.4.0 (Need to manually install)
  To install, just open package manager, select “package: Unity Registry” in the top-left drop down, search “Post Processing” in the top-right search field, select it then click “**install**”.
- Unity Starter Assets – Third Person (**You don’t need to install**)
  You **don’t need to install** this one, we used the character model and animations from this package, it is already in the project.

Use your own Art Assets

It is very easy to setup your own art assets, but before we walk through the process we need to understand how the system works.

Cube Type

- There’re 5 main types for cubes: **Grass**, **Mud**, **Stone**, **Lava**, **Custom**. Only “**Custom**” will not be part of the random generated terrain, but they still will be saved if you put them into your game world. Which makes them perfect to be used as “player only cubes” in “Build Mode”, or use them for some special structures in your game (like buildings, dungeon, etc.

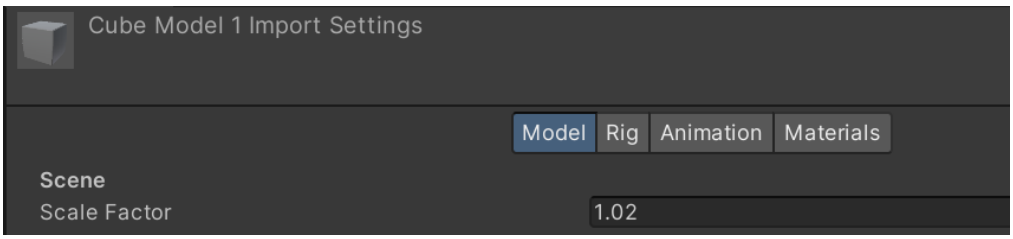


Height Level

- From 0~4 there’re 5 **height levels**, represent the height from waterline. So that you could have matched cubes in different height which make scene.
  You must have at least one cube prefab per **HeightLevel** per **Type**(except **Custom** Type), it is recommended to have 2~3 Cube prefabs each to achieve better variety, but make sure the cubes with same **Type** and **HeightLevel** looks similar enough(except **Custom** Type).

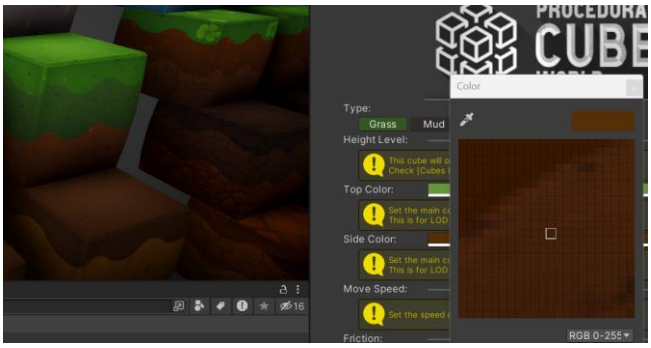
With the understanding of the above system, not we can start to set up your own cubes.

1. Make sure your cube model size is 1x1x1 meter, Then it is recommended to set the scale factor in model setting of the FBX file to 1.02 for better visual:



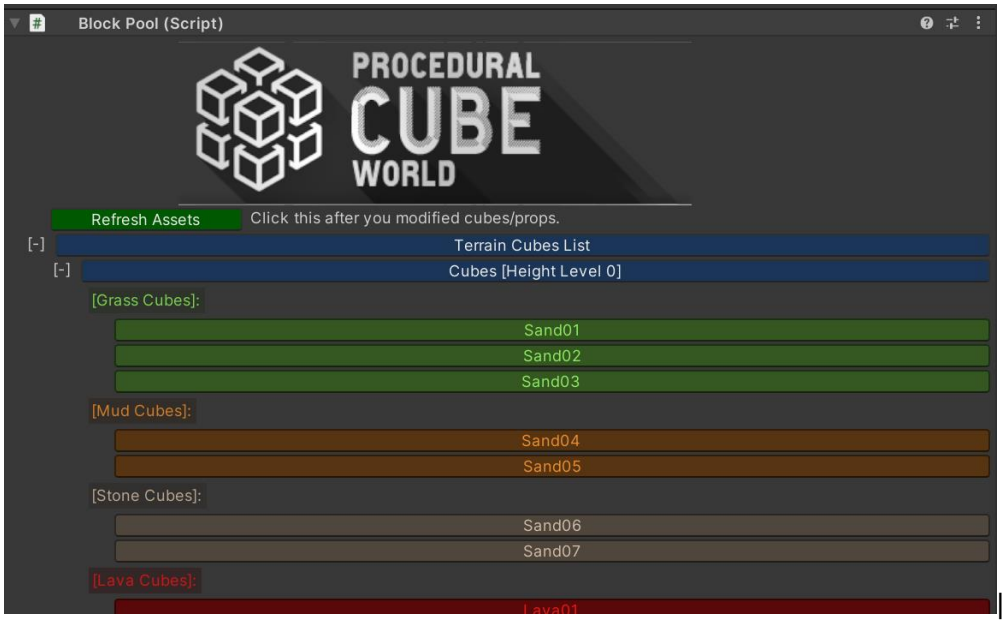
2. Make sure your cube's rotation and size looks correct with "Position"/"Rotation" of Transform set to 0,0,0 and "Scale" of Transform set to 1,1,1
3. Make sure the material of you cube have "Enable GPU Instancing" checked for better performance.
4. Add "Box Collider" to your cube gameobject.
5. Add "*BlockInfo.cs*" component to your cube gameobject, set the "**Type**" and "**Height Level**" for it.

"Top color" and "Side color" will be used in mini map and advanced LOD system, a easy way to setup it is click on the color tab to open the color picker, then click on the Eyedropper icon to capture the color from the cube model in the scene.



"*Move Speed*" and "*Friction*" setting will not directly change the movement of your character, you will need implant some code in your own player control script. Check "*DemoPlayerControl.cs*" as an example.

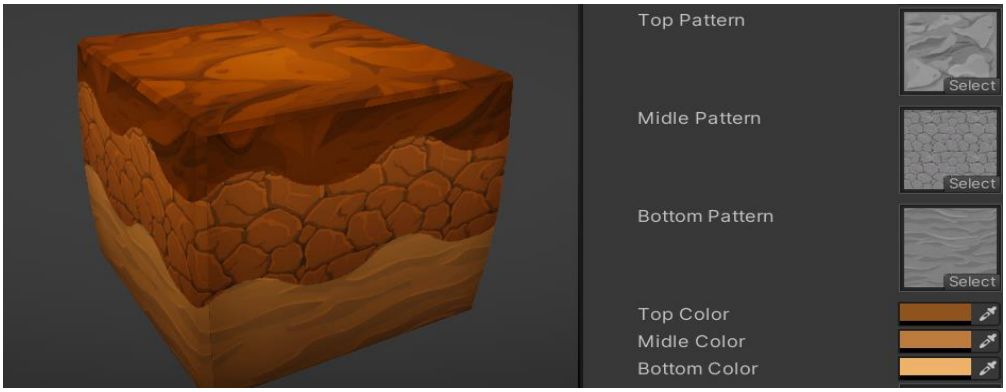
6. That's all for one cube, after you setup all your cubes, drag them as child transform of "CubeWorldGenerator" prefab, Click on "Refresh Assets" button in "*BlockPool.cs*" interface.



The script will auto setup everything, then you can check the "Terrain Cubes List" to verify if all cubes belong to their correct sort.

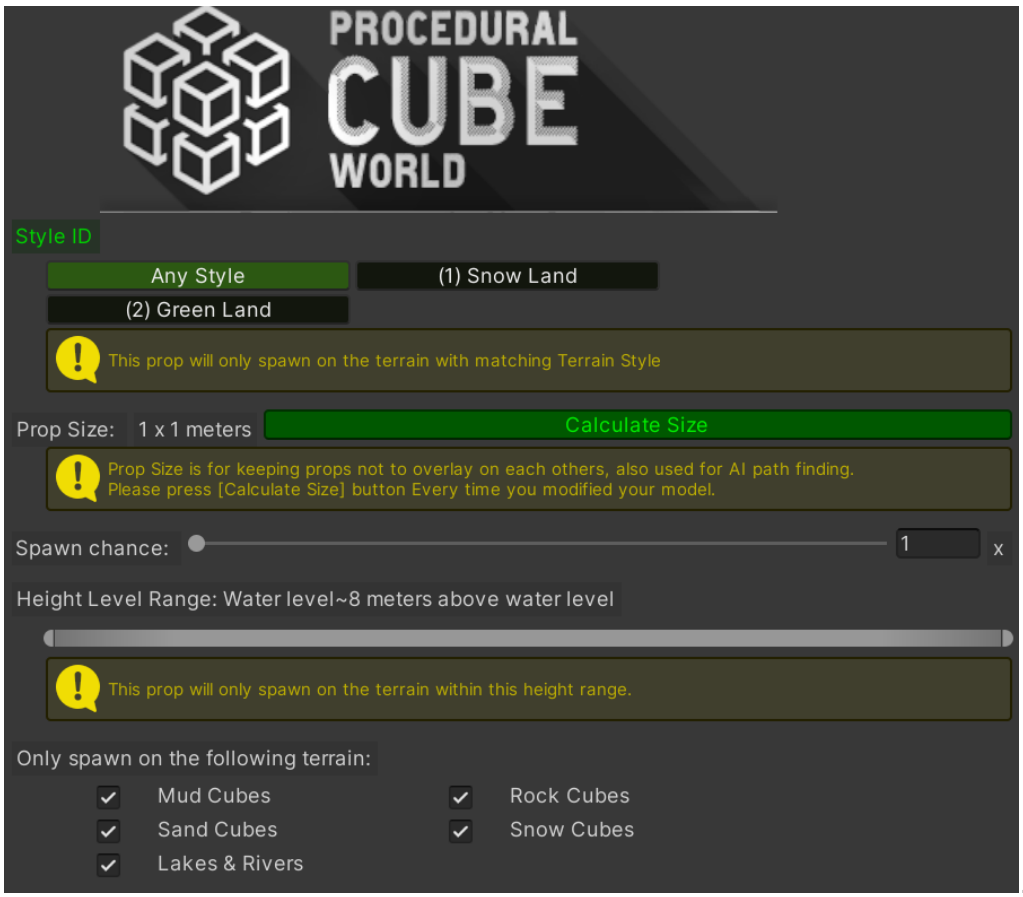
There is another simple way to make your own cubes:

- The shader of the cube is designed for easy customization use. Simply select Top/Middle/Bottom Pattern texture in the material inspector(It is recommended to use tileable grayscale texture as the pattern). Then pick Top/Middle/Bottom colors as you like. We also provided 5 version of Main Textures for different base shading, along with 3 different cube models. You can find them in "SoftKitty/ProceduralCubeWorld/Textures" and "SoftKitty/ProceduralCubeWorld/Models" folder.



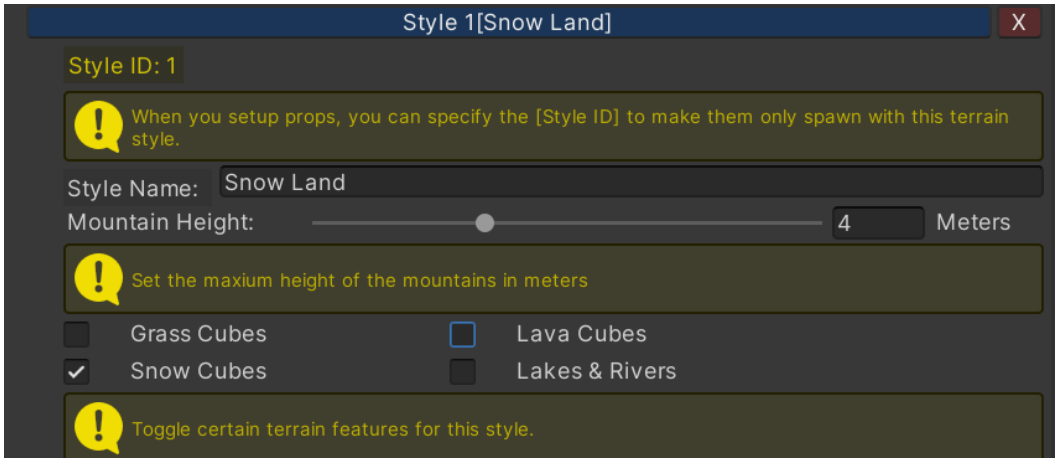
Setup Your Own Props

1. Frist make sure the pivot of your prop gameobject is in the bottom, if the pivot of the fbx file you have is not in the bottom you can put the model under an empty gameobject, and move it to align the bottom of the model with the position of its parent.
2. Add *PropInfo.cs* component to the root gameobject of your prop, setup the values as you wish, then click on the "**Calculate Size**" button to calculate the size of your prop.
3. Add a Collider to it.
4. That's all for one prop, after you setup all your props, drag them as child transform of "CubeWorldGenerator" prefab, Click on "**Refresh Assets**" button in "*BlockPool.cs*" inspector.



Terrain Variety

We have a system called "**Terrain Style**" to define how the terrain looks. You can setup multiple styles in the inspector of CubeWorldGenerator prefab.



There're two different modes to apply these styles to the world:

1. Randomly apply the styles to create visual variety.
2. Mapping the style with a texture which we called "**World Map**".

You can switch and adjust the mode in the inspector of CubeWorldGenerator prefab.

World Map

If you choose the "**World Map**" mode, you can only have maximum 4 styles, the idea is to use the RGBA channel of the World Map texture to define the **Styles ID**:

- Red channel = **Style1**
- Green channel = **Style2**
- Blue channel = **Style3**
- Alpha channel = **Style4**

The brighter the channel is, the more chance the matching style will be picked.

The cube in the center of the world will use the center pixel of the world map for style mapping, and the "World Map Scale" will define how many meters in the world space will offset 1 pixel in the mapping texture.

Make sure you world map texture is in TGA format.

Other Systems

- **Path Finding System**:
  We provided a low cost path finding system for NPC/enemy character to navigate in the random generated cube world. Please check:
  "SoftKitty/ProceduralCubeWorld/ExampleScene/Resources/ProceduralCubeWorld/ Enemy" prefab to learn how to use it.
- **Build System**:
  We provided build system for player/developer to build their own saveable world, Please check :
  "SoftKitty/ProceduralCubeWorld/ExampleScene/DemoScript/BuildSystem/ BuildControl.cs**"** to learn how to integrate it.
- **Save/Load System**:
  The generated world and custom build cubes can be save/load with a json file by calling:
  "*SaveWorld(string _path)*" and "*LoadSavedWorld(string _path)*" of *BlockGenerator.cs*
- **Minimap System**:
  The system will auto generate minimap textures for the terrain, simply use
  "*RegisterMinimapCreateCallback(OnMinimapCreateCallback _callback)*"
  "*RegisterMinimapDeleteCallback(OnMinimapDeleteCallback _callback)*"
  to hook your mini map UI with it. Please check:
  "SoftKitty/ProceduralCubeWorld/ExampleScene/DemoScript/UI/ MinimapUI.cs**"**
  to learn how to integrate it.

Scripting Manual

BlockGenerator.cs

public void **RegisterMinimapCreateCallback(OnMinimapCreateCallback _callback)**

- When a map piece is created, the callback will be called with following parameters: (int _offsetX, int _offsetY,int _key, Texture2D _tex), offsetX and offset is the position value in pixel for this map piece, _tex is the minimap texture of the map. Store the map piece with the _key, so you can manage them.

public void **RegisterMinimapDeleteCallback(OnMinimapDeleteCallback _callback)**

- When a map is deleted, the callback will be called with the key, so you can removed the paired minimap piece by the key.

public void SaveWorld(string _path)

- Save the current generated world with the path as parameter. Make sure the path is valid before calling it. Player position will be saved as well.

public void LoadSavedWorld (string _path)

- Load the current generated world with the path as parameter. Make sure the path is valid before calling it. Player will be teleported to the saved position.

public void GenerateRandomWorld ()

- Reset the current world and generate a new random world.

public void ResetCurrentWorld ()

- Reset the current world, all cubes and props will be removed.

public void Teleport(Vector3 _pos)

- Teleport the player to the given position.

public static int GetIslandKey(int _x, int _y)

- The generated world is sliced into multiple pieces with 200x200(meter) size, those pieces called island in the script. After an island is created, it will be put into a Dictionary with a key. This function is to get the key via the position of the island. The position is not actual transform position but a row and column id. For example, the first island is (0,0), the other island on its east is (-1,0), the one on its west is (1,0), the one on its north is (0,1), the one on its south is (0,-1)

public void UnloadIsland(int key)

- Manually remove an island by its key, normally you don't need to do this, all islands are auto managed by player's position.

public void Teleport(Vector3 _pos)

- Teleport the player to the given position.

public bool isInitialized()

- Check if the world data is accessible.

public float GetRunningSpeedByPosition(Vector3 _pos)

- Get the running speed(0~1) by the cube setting with the position provided.

public float GetFrictionByPosition (Vector3 _pos)

- Get the friction value(0~1) by the cube setting with the position provided.

public BlockInfo GetCubeByPosition (Vector3 _pos)

- Get the cube in the surface by the position provided, the y value of the position doesn't matters, it will always return the top cube.

public isLand GetIslandByPosition (Vector3 _pos)

- Get the island by the position provided.

public bool GetSurfaceWalkableByPosition (Vector3 _pos)

- Check if the cube with the position provided is walkable, meaning if there is any prop cover it.

public float GetSurfaceHeightByPosition (Vector3 _pos)

- Get the world space height value of surface by the position provided.

BlockPool.cs

public BlockInfo CreateBlock(BlockType _type, int _level,Transform _parent,int _angle, int _rnd, Vector3 _localPos, int _uid)

- Create a new cube by the parameters provided. _angle is 0~3 represent 4 directions, _rnd is 0~100 for randomness, _uid is for Custom Type cubes only.

public BlockInfo GetBlockPrefab(BlockType _type, int _level, int _uid)

- Get the cube prefab by the parameters provided._uid is for Custom Type cubes only.

class BlockCube (DataClass.cs)

public void BuildNewInstance(BlockInstance _newInstance, bool _load,bool _modifyData)

- If you want manually build a new cube, first you need get the BlockInfo.cs script of the top cube at the position you want to build. Then you can access BlockCube class via <BlockInfo>()._blockInstance. _blockCube to call this function. You can create _newInstance data via BlockInstance.CreateInstance(). Set _load to true if this new cube is close to player, set it to false if it's not in player's view. Set _modifyData to true if you want this new cube to be able to saved with the world.

public void RemoveInstance(bool _modifyData = true)

- If you want manually remove a cube, first you need get the BlockInfo.cs script of the top cube at the position you want to build. Then you can access BlockCube class via <BlockInfo>()._blockInstance. _blockCube to call this function. Set _modifyData to true if you want this action to be able to saved with the world.

class BlockInstance (DataClass.cs)

public void CreateInstance(BlockPool.BlockType type, int level, Transform parent, int angle, int rnd, Vector3 localPos, byte localY,int _uid=0)

- Create a new instance data by this function. _angle is 0~3 represent 4 directions, _rnd is 0~100 for randomness, _uid is for Custom Type cubes only.

public Color GetTopColor()

- Get the top color of this cube.

public Color GetSideColor()

- Get the side color of this cube.