Here you will find the documentation for the world generator asset package. It is not necessary to go through all the documentation below. This documentation aim at covering as extensively as possible this resource package in order to facilitate its integration with existing/soon to be projects .

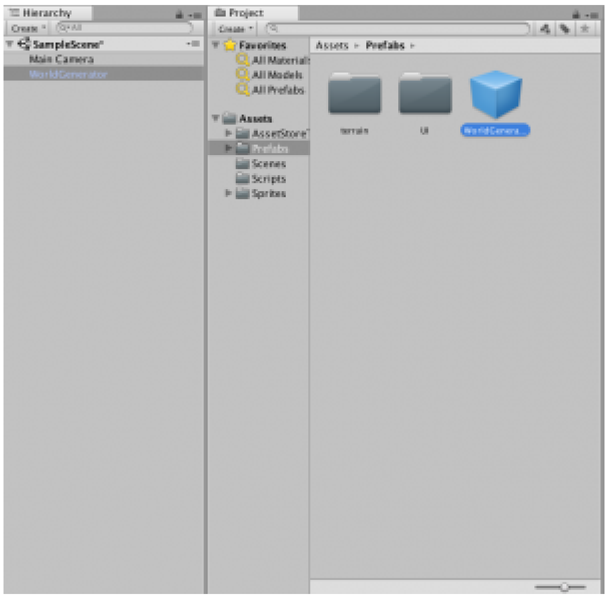
Using it the most basically only require the drag and drop of a prefab into your scene hierarchy ☺

Note: If you go through the more technically detailed part of this documentation, some design choices may feel odds or cumbersome and they probably are. Each design decision have been made in a way to increase the performances of the rendering system, in order to allow massive map size and increased unzoom level meanwhile keeping some decent performances. Or to be able to be easily modified without having to dig too deep into the core of the script.

If you have any questions, bugs or suggestions please contact us at support@moula.world

How to use (basics)

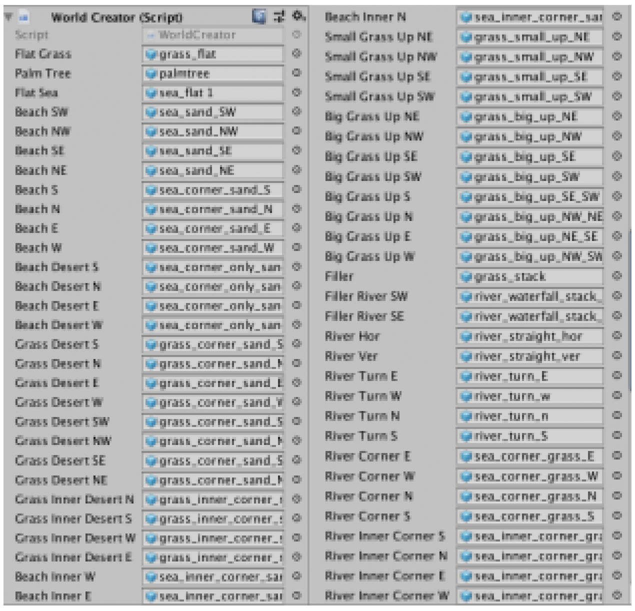
Just drag the World generator prefab from the prefab folder into your scene tree and you're good to go!



Scripts options

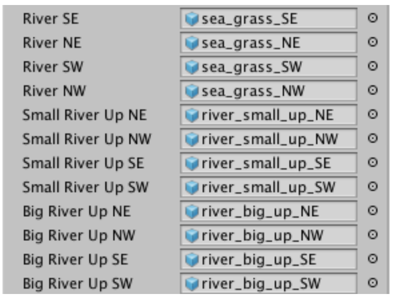
WorldCreator.cs

The world creator script come with a lot of options and it might seems overwhelming at first but most of them are linked to related prefab in order to made it easier to read and change when using it in your own projects.



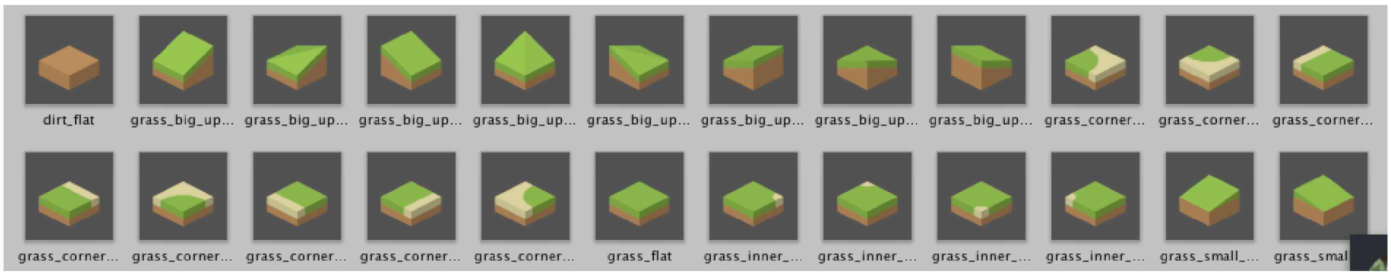
World Creator (Script)

Script	WorldCreator
Flat Grass	grass_flat
Palm Tree	palmtree
Flat Sea	sea_flat 1
Beach SW	sea_sand_SW
Beach NW	sea_sand_NW
Beach SE	sea_sand_SE
Beach NE	sea_sand_NE
Beach S	sea_corner_sand_S
Beach N	sea_corner_sand_N
Beach E	sea_corner_sand_E
Beach W	sea_corner_sand_W
Beach Desert S	sea_corner_only_san
Beach Desert N	sea_corner_only_san
Beach Desert E	sea_corner_only_san
Beach Desert W	sea_corner_only_san
Grass Desert S	grass_corner_sand_S
Grass Desert N	grass_corner_sand_N
Grass Desert E	grass_corner_sand_E
Grass Desert W	grass_corner_sand_W
Grass Desert SW	grass_corner_sand_S
Grass Desert NW	grass_corner_sand_N
Grass Desert SE	grass_corner_sand_S
Grass Desert NE	grass_corner_sand_N
Grass Inner Desert N	grass_inner_corner_
Grass Inner Desert S	grass_inner_corner_
Grass Inner Desert W	grass_inner_corner_
Grass Inner Desert E	grass_inner_corner_
Beach Inner W	sea_inner_corner_sa
Beach Inner E	sea_inner_corner_sa
Beach Inner N	sea_inner_corner_sa
Small Grass Up NE	grass_small_up_NE
Small Grass Up NW	grass_small_up_NW
Small Grass Up SE	grass_small_up_SE
Small Grass Up SW	grass_small_up_SW
Big Grass Up NE	grass_big_up_NE
Big Grass Up NW	grass_big_up_NW
Big Grass Up SE	grass_big_up_SE
Big Grass Up SW	grass_big_up_SW
Big Grass Up S	grass_big_up_SE_SW
Big Grass Up N	grass_big_up_NW_NE
Big Grass Up E	grass_big_up_NE_SE
Big Grass Up W	grass_big_up_NW_SW
Filler	grass_stack
Filler River SW	river_waterfall_stack_
Filler River SE	river_waterfall_stack_
River Hor	river_straight_hor
River Ver	river_straight_ver
River Turn E	river_turn_E
River Turn W	river_turn_w
River Turn N	river_turn_n
River Turn S	river_turn_S
River Corner E	sea_corner_grass_E
River Corner W	sea_corner_grass_W
River Corner N	sea_corner_grass_N
River Corner S	sea_corner_grass_S
River Inner Corner S	sea_inner_corner_gr
River Inner Corner N	sea_inner_corner_gr
River Inner Corner E	sea_inner_corner_gr
River Inner Corner W	sea_inner_corner_gr

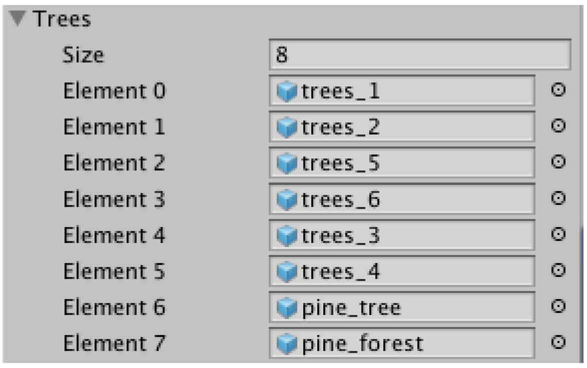


River SE	sea_grass_SE
River NE	sea_grass_NE
River SW	sea_grass_SW
River NW	sea_grass_NW
Small River Up NE	river_small_up_NE
Small River Up NW	river_small_up_NW
Small River Up SE	river_small_up_SE
Small River Up SW	river_small_up_SW
Big River Up NE	river_big_up_NE
Big River Up NW	river_big_up_NW
Big River Up SE	river_big_up_SE
Big River Up SW	river_big_up_SW

The images above refers to related prefab. You can easily replace the prefabs with your own assets, by replacing and linking the corresponding prefab. The prefabs are named according to the tile they reference in the world creator. Here is a screenshot of some of the included prefabs:



Trees families:



▼ Trees

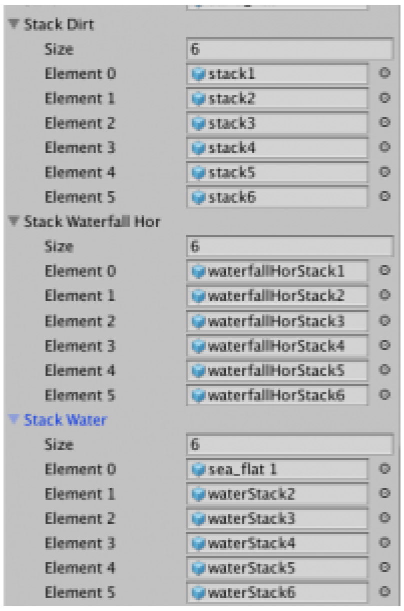
Size	8
Element 0	trees_1
Element 1	trees_2
Element 2	trees_5
Element 3	trees_6
Element 4	trees_3
Element 5	trees_4
Element 6	pine_tree
Element 7	pine_forest

In order to allow as many kind of trees as possible but still providing variations, the trees are handled in a specific way. As seen in the picture above an array represent the trees. This array must always have an even number of elements. The tree array is broke down into trees families which consist of 2 prefabs per tree family in order to bring more visual variety. Starting from the index 0, two adjacent index represent one family. Indexes 0 and 1 will be one family, 2 and 3 another one, and so on. If you wish to use only one type of tree or one family of tree, feel free to duplicate the same prefab. The tree array doesn't have a maximum amount of objects.

Trees are always grouped by families during the generation process.

Here is an example of 2 trees families. In order to work properly their index should be 0,1,2,3.

Elevation and “stacks” system:



Stack Dirt

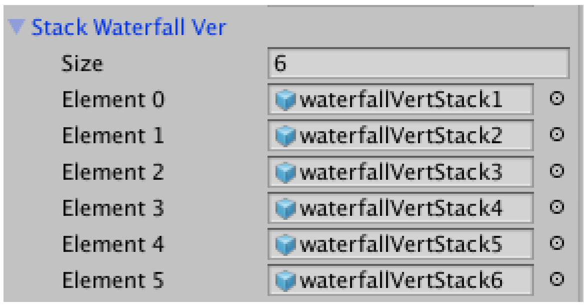
Size	6
Element 0	stack1
Element 1	stack2
Element 2	stack3
Element 3	stack4
Element 4	stack5
Element 5	stack6

Stack Waterfall Hor

Size	6
Element 0	waterfallHorStack1
Element 1	waterfallHorStack2
Element 2	waterfallHorStack3
Element 3	waterfallHorStack4
Element 4	waterfallHorStack5
Element 5	waterfallHorStack6

Stack Water

Size	6
Element 0	sea_flat 1
Element 1	waterStack2
Element 2	waterStack3
Element 3	waterStack4
Element 4	waterStack5
Element 5	waterStack6



Stack Waterfall Ver

Size	6
Element 0	waterfallVertStack1
Element 1	waterfallVertStack2
Element 2	waterfallVertStack3
Element 3	waterfallVertStack4
Element 4	waterfallVertStack5
Element 5	waterfallVertStack6

The elevation system use a stack system in order to save on processing time and display elevation. The stack size must match the **highest** option -1 and each prefab in the stack array must be the accurate height representation for that elevation. If the highest option is set to 7 as by default, the stacks size must be 6.

For example: for an height of 4, the 4th index of the corresponding stack array will be used to display the elevation.

There is 4 different stacks depending on the current element being elevated. Rivers might use “waterfall stack” or “water stack” meanwhile mountain will use the “dirt stack”. The stack used depend on the surroundings of the current cell.

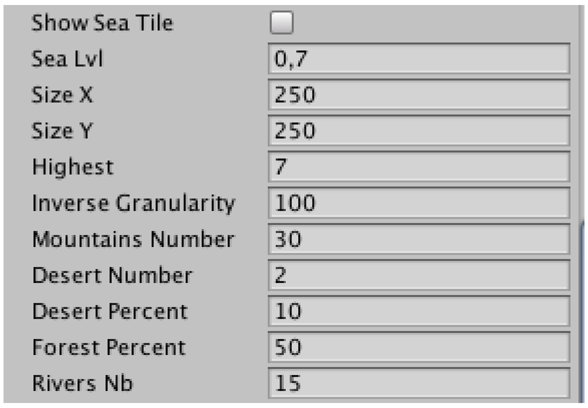
Below is a screenshot of one of the waterfall stack prefabs:

waterfallVertS...
waterfallVertS...
waterfallVertS...
waterfallVertS...
waterfallVertS...
waterfallVertS...

respectively corresponding to height of 1,2,3,4,5,6. The actual tile is displayed on top of the stack tile.

Important: Maximum elevation supported at the moment is 6 (highest value of 7) as it would exceed the tile size otherwise. You can try higher elevation at your own risks.

World generation options:



Show Sea Tile	☐
Sea Lvl	0,7
Size X	250
Size Y	250
Highest	7
Inverse Granularity	100
Mountains Number	30
Desert Number	2
Desert Percent	10
Forest Percent	50
Rivers Nb	15

Here are the options used in order to generate the world.

- Show sea tile: will display the ocean tile if checked (useful if your sea tiles are a plain color and can then be replaced by a background color).
- Sea Lvl: Define the ratio sea/land tile of the map. Value between 0.0 – 1.0. Minimum and maximum value should be avoided as they will result in no sea or no land. 0,7 for example stand for 70% of sea as by default.
- Size X: The map size on the X axis. If you use the object pooling script, there is little limitation to the map size you can use.
- Size Y: The map size on the Y axis.
- Highest: The highest elevation allowed on the map. Remember that it should be +1 of the actual highest elevation. Here the highest allowed height is then 6.
- Inverse Granularity: The highest it is the bigger landmasses will be. The lowest will result in smaller islands.
- Mountains Number: The number of mountains chains to cross the map.
- Desert Number: The number of deserts on the map.
- Desert Percentage: Percentage of landmass tiles used to fill desert.
- Forest Percentage: The amount of landmass tiles (excluding deserts) filled with trees.
- Rivers Nb: The number of rivers to be drawn. Rivers always start up high and try to make their way down. They can fail and end up in lake.

Note: The world generator will try its best to fit those criteria, however certain maps can result in impossibilities to meet specific conditions and therefore the result can be slightly different (less than the number of deserts or rivers or forests etc..).

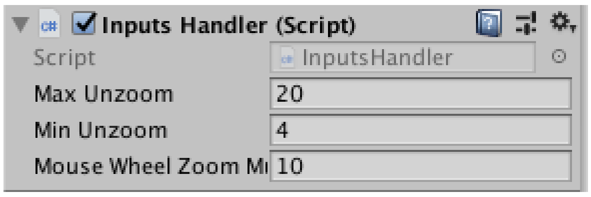
InputsHandler.cs

The input handler is responsible for providing basics movements on the map.

Controls:

- Arrow keys, wsad: move around the map
- Mouse wheel: Zoom/Unzoom
- Space: Generate a new world

Options:


Inputs Handler (Script)

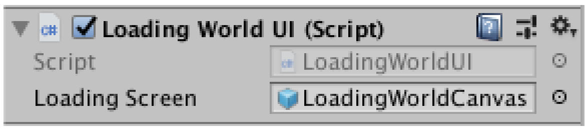
Script	InputsHandler
Max Unzoom	20
Min Unzoom	4
Mouse Wheel Zoom Mı	10

The options are pretty much self explanatory.

- Max unzoom: The maximum orthogonal size allowed for the camera (the higher it is, the lower the performances will be. It is not recommended to keep it as high as 20 in production environnment. Please see the “Performances are bad! I get low FPS!” section of the FAQ for more details).
- Min unzoom: The minimum orthogonal size allowed for the camera
- Mouse Wheel Zoom Multiplier: The ratio of increase/decrease of the zoom when the mouse wheel is used.

LoadingWorldUI.cs

Responsible for the loading screen.


Loading World UI (Script)

Script	LoadingWorldUI
Loading Screen	LoadingWorldCanvas

- Loading Screen: Link to the prefab for the UI loading screen.

Scripts Roles

The role of each script is as follow:

Game manager (GameManager.cs):

The game manager script only purpose is to instantiate the other scripts and do some basics UI refreshing when the world generation progresses.

- instantiate InputsHandler.cs
- instantiate Render.cs
- instantiate LoadingWorldUI.cs

The game manager script will then update the loadingWorldUI in LateUpdate(); as well as the first render of Render.cs

Inputs Handler (InputsHandler.cs):

Capture the inputs from keyboard and mouse in Update(); and refresh the current view if there is a need to do so through GameManager.instance.gameRenderer.SwitchInstances ();

Render (Render.cs):

- Instantiate WorldCreator.cs

The render script is responsible for the object pooling rendering system of the world creator. It can initialize the world creator, clear the pool and refresh the current viewport.

World Creator (WorldCreator.cs):

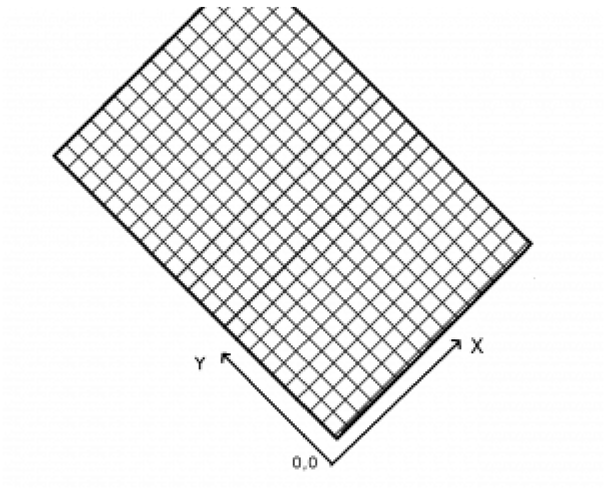
The world creator script generate the world based on the parameters supplied to the script. The most important function is: Initialize(); Which will initialize a new world.

World Creator important functions and properties

In order to make things easier to be reused in a different project, the world creator come with a set of functions and properties.

Map concept:

The isometric map should be seen as a slightly rotated normal 2d grid, here is a visualization of the x and y axis as implemented in the world generation script.



World creator properties:

float decalX, float decalY : Used to calculate the isometric ratio to space up the tile, you can increase it slightly if you wish to have the grid displayed.

int generateProgress: Used to calculate the progress of the generation of the map, it is increased manually in the word creation script during the generation process. Default to -1, must reach 100 at the end of the generation.

String generateProgressText: A string describing the current step of the generation. Is overwritten at specific steps. As of now it is hardcoded in the script when the % of the generation progresses.

int grid[x,y]: The elevation grid. It is an Int array representing the elevation of the terrain. 0 means sea, the others numbers means the height. Highest+1 value represent a desert. Some static properties are available to test the value more easily.

int objects[x,y]: The objects grid. It works the same as the world grid. 0 means empty, the other values indicate the array index of the tree array. trees.Length+1 value represent a palm tree.

Vector2 rivers[]: A list a vector2 representing the grid cell crossed by a river or lake.

int DESERT: reference the desert value for easier comparison.

int FLAT_GRASS: static value representing a grass tile of lowest height.

int SEA: static value representing the sea in the grid.

int PALM_TREE: reference the palm tree value in the objects array for easier comparison.

GameObject instances[x,y,z]: reference to the prefab tile used for each grid cell. The z value range is between 0-2. 0 is the terrain tile, 1 is the stack tile and 2 is the object on top of the tile (tree as included).

Vector3 instancesPositions[x,y,z]: Store the transform position of each grid object as a vector3. z is the same as the instance array explained above.

int instancesIndexes[x,y,z]: current index in the pool if any. -1 if there is none. It is automatically filled by the rendering process.

Sprite instancesSprites[x,y,z]: used to store the sprites in order to easily switch it by another sprite during the render process. Might feel like a duplicate of instances but it is there to accelerate calculations at runtime. It is automatically filled.

Vector2 translations[]: List of (x,y) vectors modifications in the nine possible directions. Center being (0,0).

int INFERIOR_EQUAL, int INFERIOR, int EQUAL, int SUPERIOR_EQUAL, int SUPERIOR: constants used for comparison, please see the hasAdjacent();method detailed below.

int FOUR_DIRECTION_MASK[], int HEIGHT_DIRECTION_MASK[], int NINE_POSITIONS_MASK[], int NW_CORNER[], int SW_CORNER[], int NE_CORNER[], int SE_CORNER[]: predefined masks used to check for adjacent cells.

Respectively:

- Four directions (north, west, est, south)
- Height directions (north west, north, north east, west, east, south west, south, south east)
- nine positions (north west, north, north east, west, center, east, south west, south, south east)
- North west corner (north west, north, west)
- South west corner (west, south west, south)
- North east corner (north, north east, east)
- South east corner (east, south, south east)

for further information on masks, please see the hasAdjacentInList(); and hasAdjacent(); methods detailed below.

Note: some properties have been omitted are they are used strictly for generation purpose and can then be avoided. If you wish to edit the world generation script, please use the comments in the code to understand the particular use of properties and methods not detailed in this document.

World creator methods:

GenerateWorld(): The main generation function, will generate the grid based on the script options. Not to be called as is, please use the Initialize(); function instead.

int hasAdjacentInList(List<Vector2> inList, Vector2 pos, int[] mask): Find the number of adjacent objects in a list (the rivers list for example) and returns it.

- inList: The list to look in
- pos: position of the current object
- mask: the mask to be applied. For further details, please see masks below.

int hasAdjacent(int[,] inArray, Vector2 pos, int val, int[] mask = null, int operand = EQUAL): Return the number of adjacent cells in a two-dimensional array based on a set of conditions.

- inArray: The array to look into
- pos: the position of the object
- val: the value to be looked for
- mask: the mask to apply (for further informations on masks, please see masks below)
- operand: the operand to be applied to the value looked for.

for example: if the operand is EQUALand the value is DESERT hasAdjacent();will return the number of cells containing desert depending of the provided mask. If the operand is SUPERIOR_EQUAL and the value is SEA it will return all the adjacent cells being higher than sea level depending of the mask provided.

`Draw()`: Fill the instances and the instancesPositions arrays with the correct values depending on their surroundings.

Initialize(): Initialize the world generator and generate a new world.

Masks:

Masks are a one dimensional array of 9 elements representing the nine possible positions relative to the current value. They are organized as follow:

[NORTH_WEST, NORTH, NORTH_EAST, WEST, CENTER, EST, SOUTH_WEST, SOUTH, SOUTH EAST].

The value for each of the values can be only 0 or 1.

0 meaning it is skipped and 1 meaning it is included. Looking for the height possible direction around a cell would then result in the following mask:

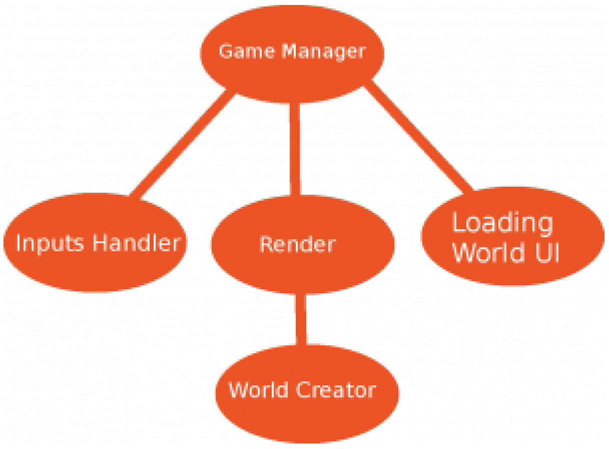
[1,1,1,1,0,1,1,1,1]

Some useful masks such as this one are already defined in the script, as detailed in the properties listing.

Note: Some method have been omitted are they are not necessary to be able to use the world generator script. If you wish to make some modification to the script please use the comments provided in the source code in order to understand how they operate.

Scripts organization

If you want to dig deeper into the scripts hierarchy, here is how they are organized.



Using the Object Pooling Rendering and the World Creator without game manager and Inputs Handler

If you wish to use the Render.cs script and want to generate a new world but don't want to use the provided GameManager.cs and InputHandlers.cs you need to call the method **FirstRender();** of the Render.cs script to first setup the initial viewport and then **SwitchInstance();** every time you wish to refresh the current viewport.

If you wish to regenerate the world you would just have to call the **ClearRender();** function of the Render.cs script which will empty the pool and reinitialize the world.

ClearRender(); Shouldn't be called if it's the first time you are generating a new world.

Here is a simplistic example:

```
// generate a new world
void Start() {
        // get the Render.cs script
        gameRenderer = GetComponent<Render> ();
        // initialise it
        gameRenderer.Initialize ();
}
```

```
// regenerate the world
void generateANewWorld() {
        // will clear the pool and automatically regenerate the world
        gameRenderer.ClearRender ();
}

// use the object pooling to refresh the current viewport
void Update() {
        // get the current progression of the world generation
        if (gameRenderer.world.generateProgress === 100) {
                // process the first viewport
                gameRenderer.FirstRender ();
                // reinitialise the generation process to not call firstRender a second
                time.
                gameRenderer.world.generateProgress = -1;
        } else {
                // put your movement condition here and then call switchInstances when the
                viewport needs to be refreshed
                gameRenderer.SwitchInstances ();
        }
}
```

Using only the World Creation Script

If you wish to only use the world creation script and none of the other script provided, just discard all the other scripts and call the Initialize(); function of the WorldCreator.cs script whenever you want to generate a new world. Be aware that you will have to use your own rendering system as nothing will be rendered by default, only the grid and objects arrays will be populated.

In order to build your own rendering system you should use the instances array and instancesPositions array as a bare minimum. As explained above those contains references to the prefabs used for the map and their respective positions.

It is strongly recommended to have a look at the SwitchInstances();function of the Render.csscript in order to have an rendering example to use as a base.

FAQ

What should I do if I encounter a bug or an unexpected behavior?
Please write to us at support@moula.world and we’ll get it fixed as soon as possible.

Which map sizes are supported?

Any kind of map size are supported, if you are using the object pooling provided with the Render.cs script the performances shouldn't be impacted by the size of the map. Be aware that the largest the map, the longest time it will take to be generated and the higher will be the memory usage of the game. Other than that FPS should remain stable regardless of the map size.

Something isn't clear in the documentation and i would need extra informations.

If what you don't understand is related to the scripts please first start by looking at the comments inside the script. If it's something else or your issue isn't still fixed drop us a mail at support@moula.world with a description of your issue and suggestion on how to improve this documentation. This documentation is a work in progress and will be updated following the recommendations and the need of the users.

How can i add more objects than just trees?

You will need to write your own algorithm in order to populate your new objects on the map. You can take a look at the way current objects are populated and modify it to suit your need. The generation would have to take part in the GenerateWorld(); function of the WorldCreator.cs script and you will need to populate the instances and instancesPositions with the proper prefabs. You can have a look at how it is done in the Draw(); function of the WorldCreator.cs script. Once that's done and your objects are properly added to the objects[,] array, the rendering system will render them automatically.

How can i add more varieties of trees?

Just add new prefabs to the tree array. Remember than the tree array length must be an even number and a family of tree consist of 2 prefabs with indexes following each others. You can duplicate the same prefab if you wish to have only one sprite for a family.

How can i add one or several characters moving around?

Just add it to the objects[,] array and populate the proper values in the instancesand instancesPositions arrays. Update it's index in those array when it moves and the rendering system will render it properly.

How to handle collision?

Use the grid[,] array of the WorldCreator.cs script to get the height of a specific cell and therefore decide if your character is allowed to move there or not. You can use the function hasAdjacentInList(); to find surrounding rivers or more generally hasAdjacent(); with either the grid or the object array to find what is surrounding your character.

Performances are bad! I get low FPS!

Please check your unzoom max level. By default the unzoom max level is set extremely high (20) which force your computer to render a more than 100×100 grid on each movement! It is a value which is intended for testing only and it is recommended to keep the maximum unzoom at 10 at most. Be aware that a game like Civilization V on its highest map size generate a 128×80 world map. An unzoom value of 20 forces your computer to generate a much bigger map than that on a single viewport and recalculating it on every frame…

What are the mechanics in place to improve performances?

There is a lot of bigger and smaller tweaks implemented in order to get the better of the Unity engine. First the render use an object pooling system which avoid generating thousands of objects when generating the map but instead reuse the already instantiated objects. Here are some more:

- Some functions as the Mathf.Min() and Mathf.Max() have been discarded and replaced by more basic logical conditions.
- Sprites are stored in a separate array in order to minimise the calls to getComponent.
- The rendering system calculate the difference in viewport between the current viewport and the last one, in order to render / hide only the difference between the two.
- The elevation system is based on “stack” sprite in order to reduce the number of required GameObjects.
- Tiles and Stacks which aren’t visible are discarded from the rendering process to lower further the number of required GameObjects.
- The rendering of sea tiles is optional in order to provide static background / background color when it matches the visual style of the map.
- and much more…

Obviously nothing is perfect and if you have a suggestion on how to improve things even further, we would be glad to hear about it! Contact us at support@moula.world.

How easy it is to use custom assets?

Assets have been broke down into single entities to make them easier to replace. If you wish to use your own assets you will have to recreate all the prefabs used and link them to the correct corresponding GameObject of the WorldCreation script. It should be pretty straightforward as you can easily look up all the sprite used.

What assets are included by default?

The assets included by default are a modified version of the Kenney assets that you can find here: http://kenney.nl/assets. According to their license you are free to reuse them in your game as well.