INDICE

Description:

Basic Instructions:

Extended Instructions:

Features:

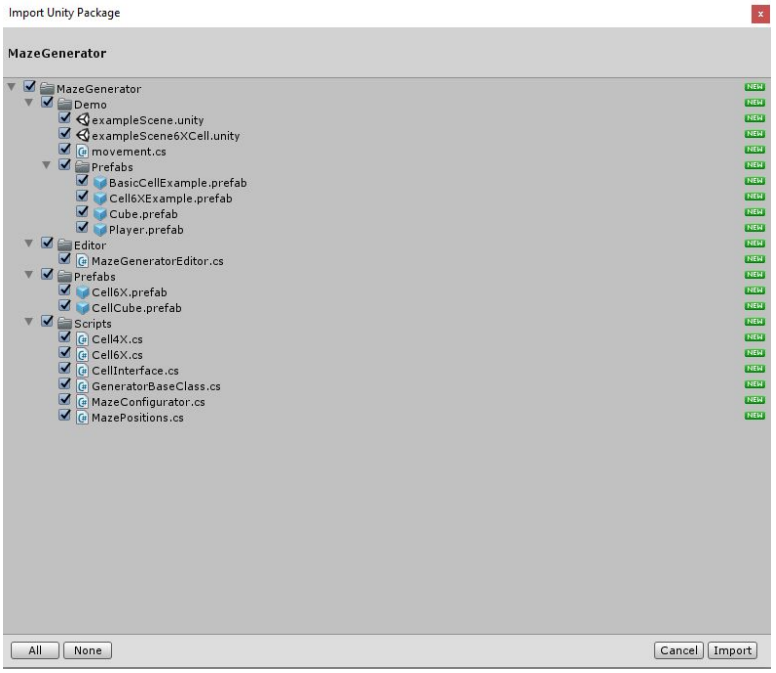
MAZE GENERATOR

Description:

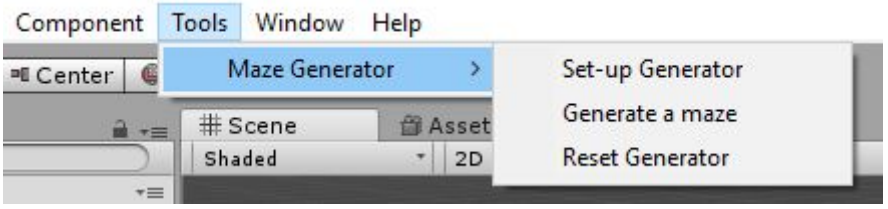
With this asset you can configure a Generator and build different types of mazes.
The generator can be used in the editor to generate Mazes, modify them and store as prefabs.
Or can be used in execution mode, different maze will be generated on each execution.

Basic Instructions:

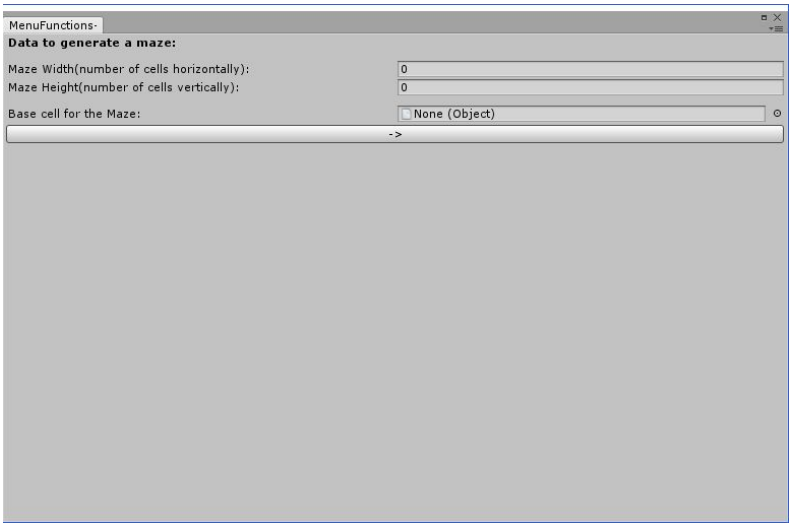
1. Import the asset package. You can deselect the Demo folder if you don't need the example scenes.



2. Set-up Generator: Once imported a new option will appear in ToolBar, the option is Maze Generator with 3 suboptions Set-Up Generator, Generate a maze and Reset Generator.



a. Choose **Set-Up Generator** to show the next screen:



Maze Width will be the number of cells horizontally in the maze. So the full width of the maze will be the width of base cell multiplied by this number.
Maze Height will be the number of cells vertically in the maze.
Base cell is the GameObject or Prefab that represent one cell of the maze. This base cell have to be a GameObject with a CellInterface Script attached, and with the parent representing the true width and height of the cell.

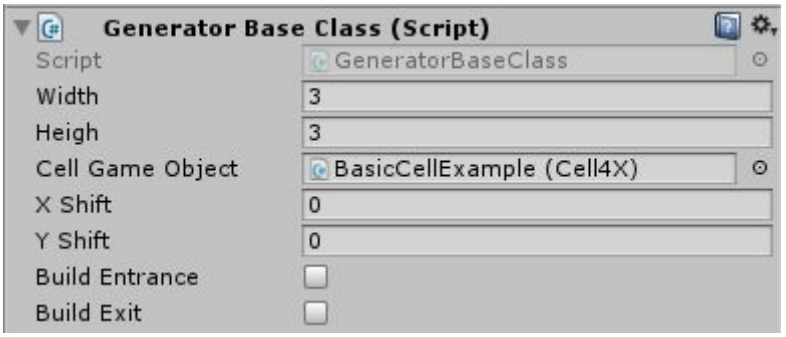
b. Click on the [->] Button and a Game Object called Generator will be created in the actual scene. If a Generator are yet in the scene this button update the basic variables of that generator with this new variables.
c. A new Button will appear in the SetUp Generator window to generate a Maze, this button will launch the generation function.

3. **Generate Maze** will generate a Maze under the generator object in the scene. You can store this object as prefab dragging the generator to the Project Tab.

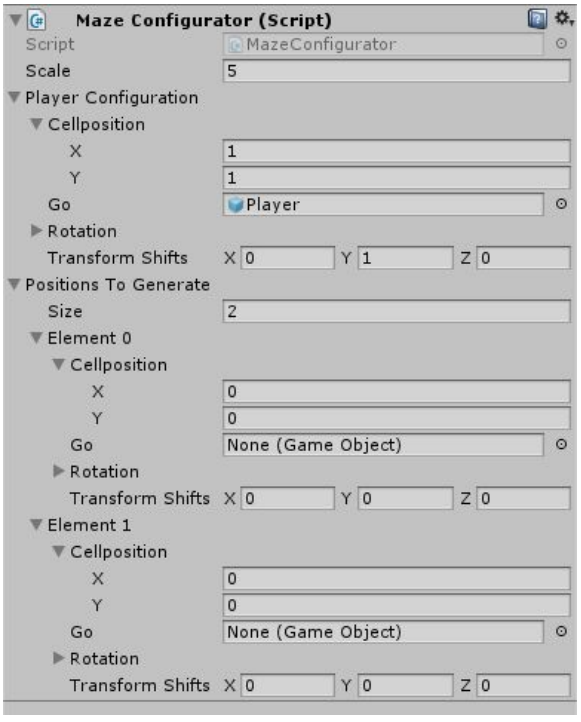
4. The last option under Maze Generator is **Reset Generator**. This option resets the Generator object to a state as if it have just setted up, erasing any maze generated in it.

Extended Instructions:

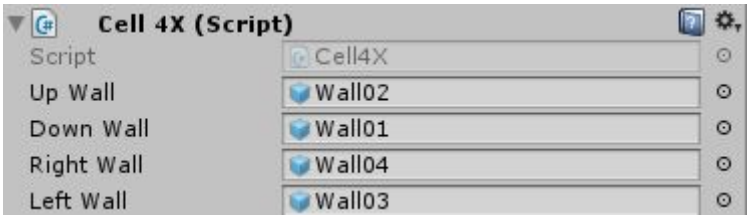
- Maze Generator: Base Script to generate the Maze. Store the size of the maze, the cell, and small changes to the generation.



  - The Width and Height is the number of cells in each direction
  - The CellGameObject is the prefab which represent the base cell for the Maze, this GameObject must have a CellInterface Component.
  - The X shift is the shifting percentage based on the base cell size, applied in the odd rows. You can change this behavior extending the Generator Base Class or changing the original.
  - The Y shift is the shifting percentabe based on the base cell size, but this shift is applied in all rows.
  - Build Entrance/Exit. If is checked the builder will destroy a wall in the first cell or in the last cell to make an entrance or a exit.
- Maze Configurator: The maze configurator is a component in the Generator which configure the scale of the maze and the place of external objects in execution mode.



    - Scale property is the scale for the Maze will be multiplied when the building ends
    - Under Player Configuration can indicate a place where the GameObject Go(usually a player type object) have to be moved or instantiated.
        - You can configure a rotation and position shifting applied over the position of generation.
    - Positions to generate is a list of objects which will appear in the maze in the positions indicated, with the rotation and with the position shifts indicated.
- Cell Configuration: Each base cell must have a Cell Interface Script(or a extended class of this). In this class you can configure a few things.



    - In base class you won't have the wall variables, when you build your own cell have to set up a number of walls and assign them.
    - In this example the cell is a 4 walls cell(a square), in the childs of this GameObject are Wall01, 02, 03 and Wall 04. You have to assign each one to his correspond variable and program the method to destroy them.
- Extendable classes
    - You can create a Maze Generator with your own rules, and methods extending from GeneratorBaseClass
    - You must extend from CellInterface if you are using a new cell with new properties, like more height than width, a cell with more walls, or cells with different behaviour. The important methods to implement of this interface are:
        - destroyWallsOnDirectionOf(int x, int y); Method to destroy the wall in the direction sended.
        - getNeighboutsNotvisitedOfCurrentCell(CellInterface[,] fullGrid); Functions that returns a list of the cells that are "touching" this cell and are not visited yet.

Features:

- Random generation
- Personalizable size
- Personalizable cells
- Extendable generator class
- Extendable cell class
- Maze generates in editor mode and in game mode
- Easy to configure