Home Architect

Documentation

Table of Content

Preamble **3**
- Developer note 3
- What to expect in the future updates 4
- Bug finding and report 4

Introduction **5**
- How it works 5
- Wall structure 5
- How to start 6
- Important concepts 7
  - Position objects 7
  - How to place a door or a window 8
  - Setting roof styles 9
    - Using the method “setRoofStyle” 9
    - Adding a module 9
  - General rules to follow 9
- Unity interface 10

Methods **11**
- ◆ addDoor() [1/2] 11
- ◆ addDoor() [2/2] 11
- ◆ addDoubleStair() [1/2] 12
- ◆ addDoubleStair() [2/2] 12
- ◆ addFence() [1/3] 12
- ◆ addFence() [2/3] 13
- ◆ addFence() [3/3] 13
- ◆ addFloor() [1/2] 13
- ◆ addFloor() [2/2] 13
- ◆ addFloorEverywhere() 13
- ◆ addRoom() [1/2] 14
- ◆ addRoom() [2/2] 14
- ◆ addSingleStair() [1/2] 14
- ◆ addSingleStair() [2/2] 14
- ◆ addWall() [1/2] 15
- ◆ addWall() [2/2] 15
- ◆ addWideEntrance() 15
- ◆ addWindow() [1/2] 15
- ◆ addWindow() [2/2] 16
- ◆ autoBuildRoof() 16
- ◆ Build() 16
- ◆ buildRoof() 16
- ◆ isInside() [1/2] 16
- ◆ isInside() [2/2] 16
- ◆ isInsideRoom() 17
- ◆ mapRoofs() 17

◆ mergeRooms() 17
◆ removeWall() [1/2] 17
◆ removeWall() [2/2] 17
◆ setElevation() 17
◆ setRoofStyle() 17
◆ setStairStyle() 18
◆ setTextures() 18

Preamble

Developer note

Thank you for using Home Architect. This documentation is intended to help you approach how this engine works, as well as presenting an exhaustive list of functions and tools comprised in the asset.

If you have any comments or suggestions, please feel free to reach me at:

julien.sacrez@outlook.com

I have designed this tool alone and it represents a fair amount of my time as a developer. If you are in possession of an illegal copy, thanks for using it anyway! However, please consider buying a license on the asset store if you can afford to, has it will help me continue to maintain its development and add new features in the future.

Good luck with your projects! Best regards,

Julien SACREZ

What to expect in the future updates

- Diagonal room and floor by floor splitting option for ¾ gameview [Ongoing development]
- More roof options [Ongoing development]
- Further optimisations [Research]
- Furniture management [Research]
- Building aging [Research]
- Allowing negative positions [Pending]

If you have any suggestions, feel free to send me an e-mail

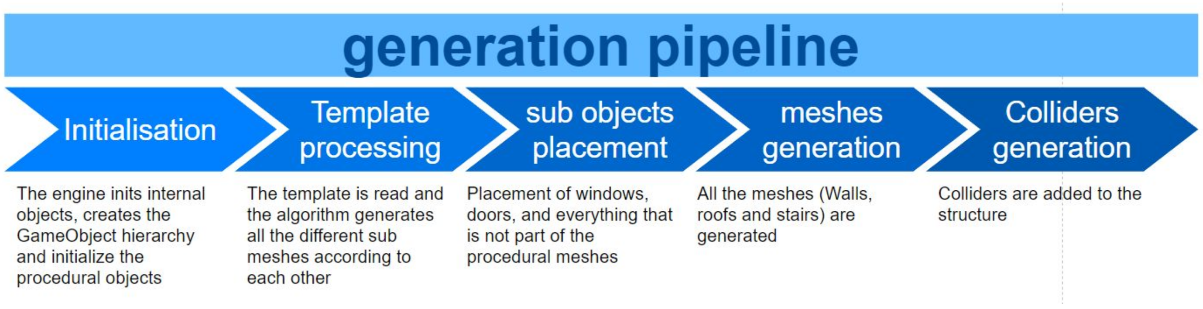
Bug finding and report

This asset is still very young, and might contains inconsistencies, bugs and errors in its code or documentation. If you find one, please contact me at the address above, I will do my best to resolve them as quick as possible.

Introduction

How it works

If you ever played games like the Sims, then this asset should feel familiar: The engine will build an architectural structure with basic blocks, and bake them into several meshes like wall structures, roofs, stairs and so on. The final result is defined by a set of instructions gathered into a method called “building Template”.



The engine inits internal objects, creates the GameObject hierarchy and initialize the procedural objects

The template is read and the algorithm generates all the different sub meshes according to each other

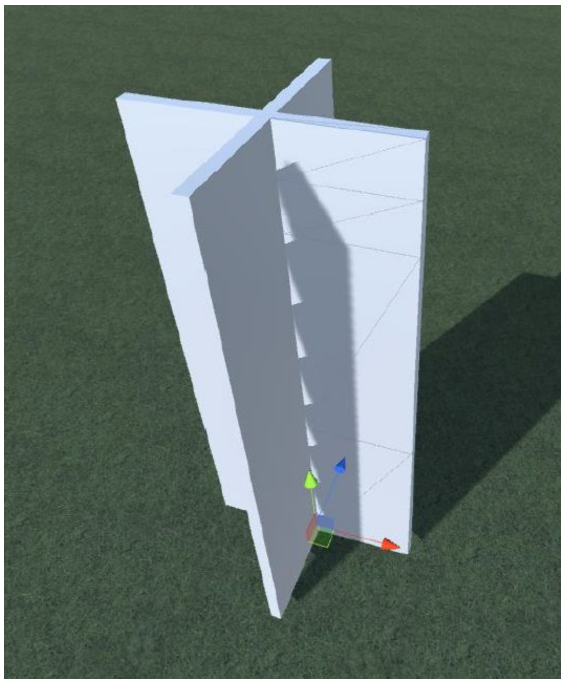
Placement of windows, doors, and everything that is not part of the procedural meshes

All the meshes (Walls, roofs and stairs) are generated

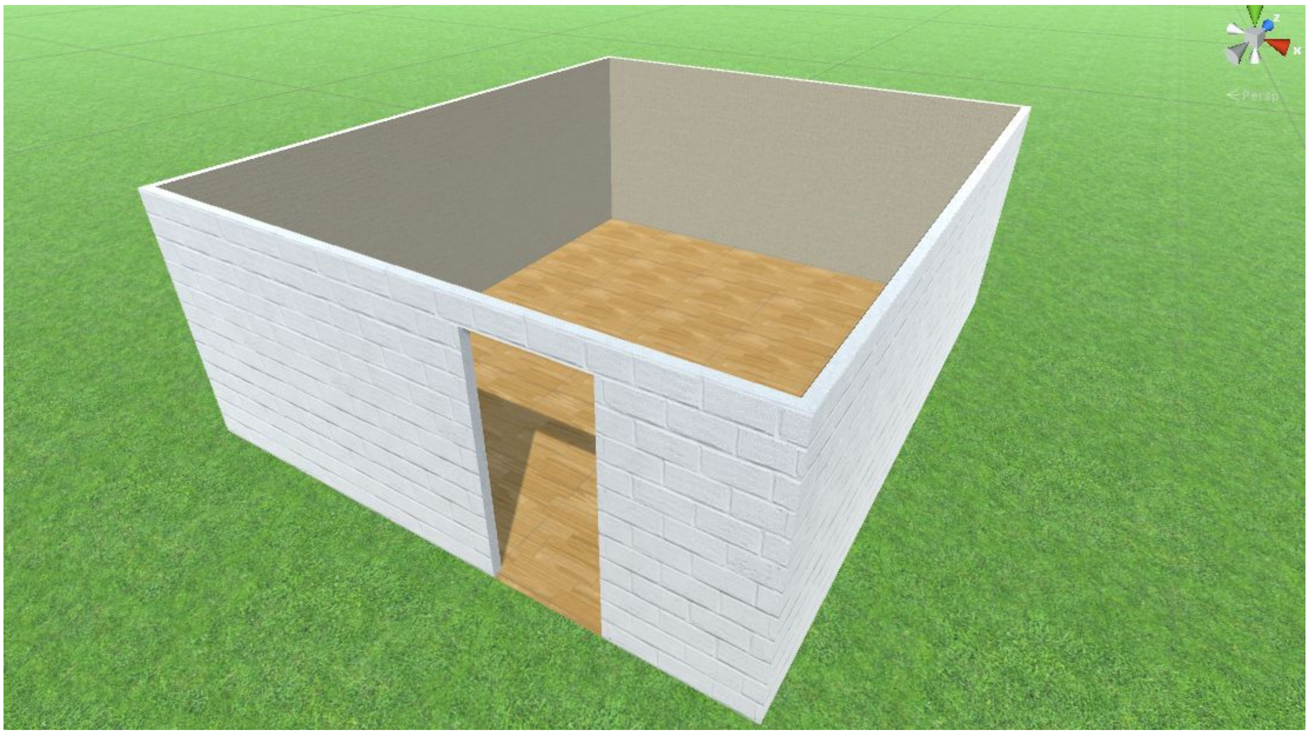
Colliders are added to the structure

Wall structure

The fundamental block used to generate the wall structure is this one:



Comprised of several sub parts, it it will arrange in form and texture to make rooms. The sub walls on the piece are defined by their cardinal orientation. Once those are in place, a typical room looks like this:



The parts that are not used by the structure won't be generated, and the different parts will be assembled into the final mesh.

How to start

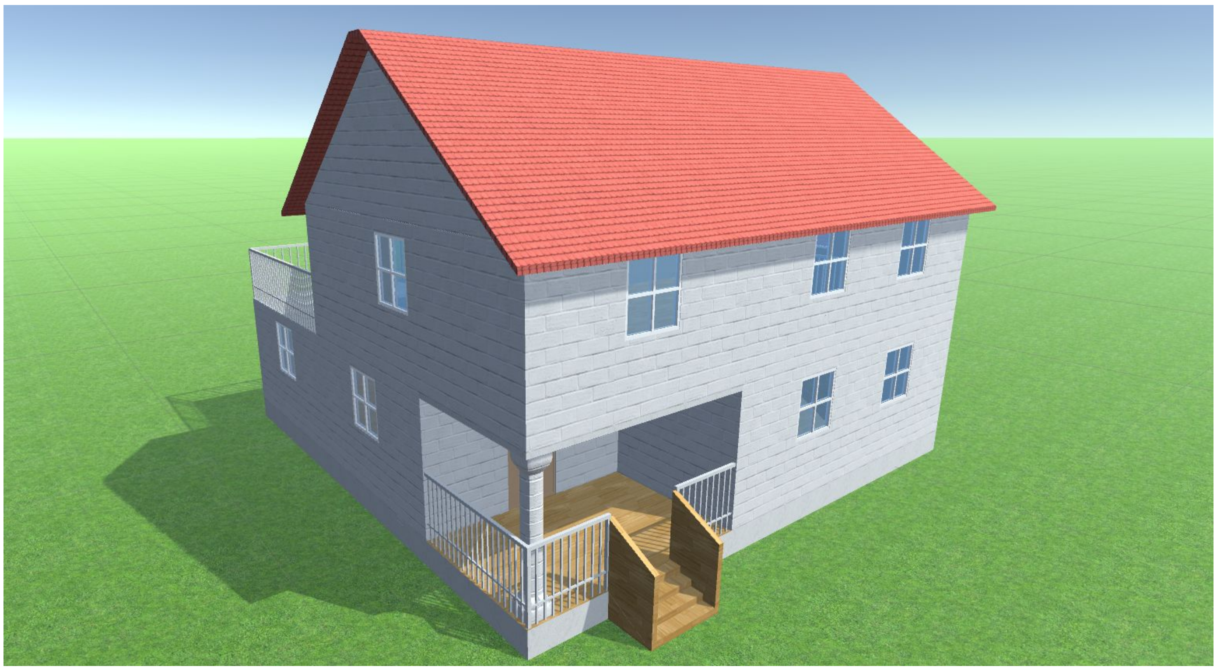
Once you downloaded the asset, you have to create a class that will inheritate the object "HouseBuilder". You will also have to add the using "HouseBuilderEngine"

```
using System;
using System.Collections;
using UnityEngine;
using HouseBuilderEngine;

public class SmallHouse : HouseBuilder
{
    public override void buildTemplate()
    {

        // Your template will be placed here
    }
}
```

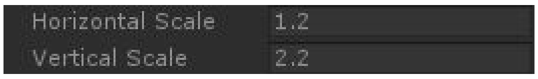
You can check out the examples provided in the asset. Example of a generation:



Important concepts

Position objects

Position objects are used throughout a building template to place entities, like rooms, doors or windows. It works very much like a Vector3, but represents distance in cell units, and has integers for x:y:z values. Those values are used by the tool and converted to Unity units, according to the scales defined on the interface. For example, a Position(4, 1, 2), when the asset is configured like so:



Horizontal Scale	1.2
Vertical Scale	2.2

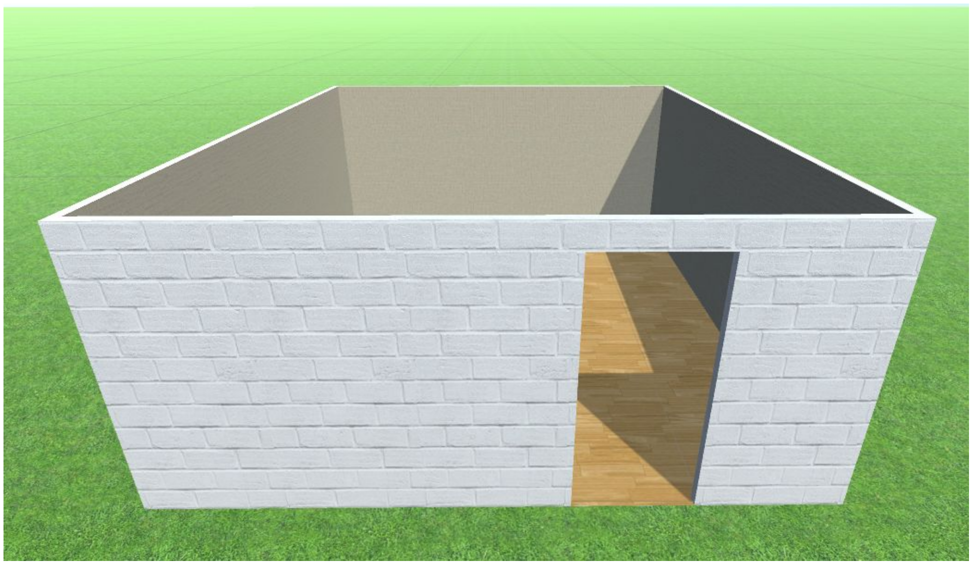
Will be translated to: Vector3(4.8, 2.2, 2.4). If you want to translate Positions to Unity units, you can proceed like this:

```
(Position).toVector3(RealDimensionsVector)
```

`RealDimensionsVector` is a Vector3 that represents the scale of the model defined above, so in this example it will be Vector3(1.2, 2.2, 1.2). `toVector3` multiply each axis of the position by the corresponding axis in the vector passed as parameter.

How to place a door or a window

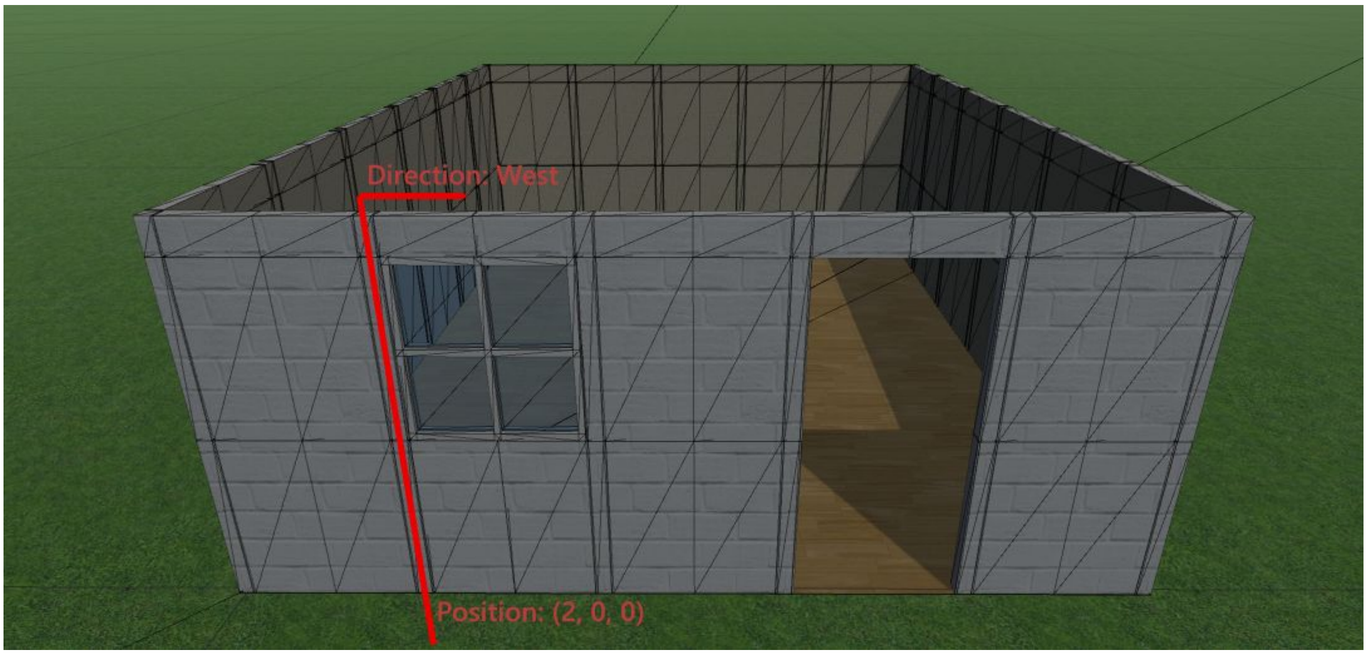
Consider this room:



The wall in front of us is facing south, and we want to place a window through this wall, next to the door. The corner of the room is at (0, 0, 0). All we want to do is:

```
placeWindow(new Position(1, 0, 0), Directions.WEST);
```

And we get this result:



Keep in mind that objects placed in the walls are not centered, because we are dealing with wall parts assembled together. Therefore a window oriented west will have to fit in a wall *facing* either south or north.

Setting roof styles

There is several ways to configure your roofs

Using the method "setRoofStyle"

Using the method "`setRoofStyle()`" will change the style of every roof of the current building after the call. Refer to the documentation below on how to use it

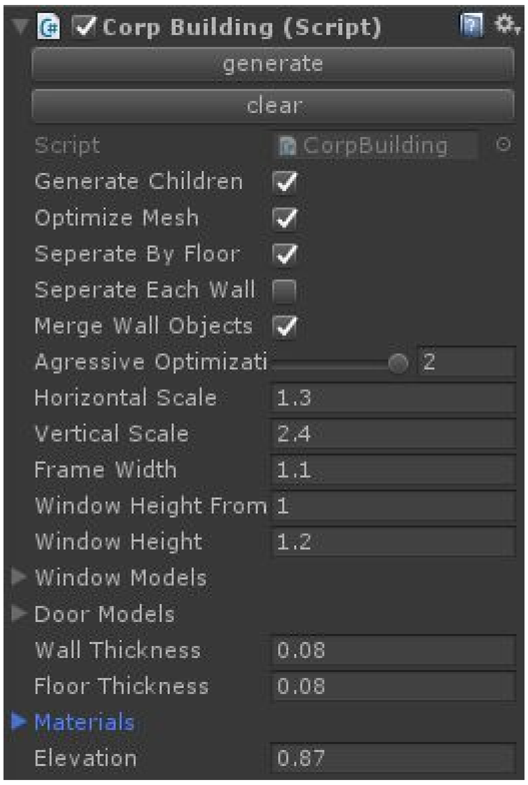
Adding a module

You can add the module `RoofStyle` to the object holding your house script. If you choose that option, `setRoofStyle()` will stop being taken into account

General rules to follow

- This tools does not allow negative numbers, so negative Position() objects will return an exception. If you need to have space toward west and south of the center of you model, you may want to offset your gameobject in that direction and work with greater positive numbers. You can find an example of that with the dungeon generator example.
- X and Z scales are tied together, and cannot not be independent
- This tool is optimized to work with the default scale. Extreme ratios are possible, but not recommended.
- When working on multiple buildings, it is recommended to use one module for each building.

Unity interface


Corp Building (Script)

generate

clear

Script	CorpBuilding
Generate Children	✓
Optimize Mesh	✓
Seperate By Floor	✓
Seperate Each Wall	☐
Merge Wall Objects	✓
Agressive Optimizati	2
Horizontal Scale	1.3
Vertical Scale	2.4
Frame Width	1.1
Window Height From	1
Window Height	1.2
▶ Window Models	
▶ Door Models	
Wall Thickness	0.08
Floor Thickness	0.08
▶ Materials	
Elevation	0.87

- **Optimize mesh**: Concatenate the original panes used to make the walls. This option should always be activated, but can be used to understand how the engine works or debug generations. You can activate "shaded wireframe" mode in the scene viewer to see the differences
- **Separate by floor**: Generate each floor separately in ordered containers in the hierarchy
- **Separate each wall**: If this option is checked, the walls of rooms are going to be separate objects
- **Merge wall objects**: Merges objects instantiated in walls like doors and windows.
- **Aggressive optimization**:
  - value = 0: Doesn't remove any triangle
  - value = 1: remove the triangles on top of walls. This options is good for standard generations where roofs don't need to be hidden for example
  - value = 2: removes every horizontal triangles on top of walls and in window frames. This options is useful if you want to produce decorative models, with statics windows and doors.
- **Horizontal scale**: the horizontal stretch (X and Z) of the model
- **Vertical scale**: The height of one floor
- **Frame width**: The width of a door or window
- **Window height from floor**: distance between the floor and the window
- **Window height:** self explanatory

- **Window models / Door models**: list of models to be used by the tool, see corresponding methods for specific use
- **Materials**: The material list for the model's walls, floors, and stairs.
- **Elevation**: Height of the buffer between the ground and the bottom of the model.

Methods

You will find below the documentation for each public method available to you. You can also find examples of those methods being used in the example folder.

◆ addDoor() [1/2]

```
void RoomArchitectEngine.RoomArchitect.addDoor ( Position    position,
                                                 Directions  direction,
                                                 bool        inverted = false,
                                                 int         doorID = DEFAULT
                                               )
```

Places a door through a wall junction Example: if a door is added at [0;0;0] with direction north, the door will be placed between [0;0;0] and [0;0;1]

Parameters

position	position of the door
direction	Direction to which the door is parallel to
inverted	If true, this parameters allows flipping the door 180° by its center
doorID	ID corresponding to doors in the door's list on the unity interface

◆ addDoor() [2/2]

```
void RoomArchitectEngine.RoomArchitect.addDoor ( WallPart  wall1,
                                                 WallPart  wall2,
                                                 bool      inverted = false,
                                                 int       doorID = DEFAULT
                                               )
```

Places a door through a wall junction Example: if a door is added between [O;O;O] and [0;0;1], it will be placed in the walls between those points, parralel to the north direction.

Parameters

wall1	first wall
wall2	second wall
inverted	If true, this parameters allows flipping the door 180° by its center
doorID	ID corresponding to doors in the door's list on the unity interface

◆ addDoubleStair() [1/2]

```
DoubleStair addDoubleStair ( Vector3     bottomPosition,
                             Vector3     topPosition,
                             int         stepNumber,
                             Directions  orientation
```

)

Add a double stairway to the building. This method use real Unity units, and has a cell unit equivalent

Parameters

bottomPosition	Bottom left position of the stair
topPosition	top right position of the stair
stepNumber	number of steps in the stairway
orientation	Cardinal orientation. If the steps points north, the lowest point will be south, and the highest point north

◆ addDoubleStair() [2/2]

```
DoubleStair addDoubleStair ( Position    bottomPosition,
                             Position    topPosition,
                             int         stepNumber,
                             Directions  orientation
                           )
```

Add a double stairway to the building. This method use cell units, and has a real Unity unit equivalent

Parameters

bottomPosition	Bottom left position of the stair
topPosition	top right position of the stair
stepNumber	number of steps in the stairway
orientation	Cardinal orientation. If the steps points north, the lowest point will be south, and the highest point north

◆ addFence() [1/3]

```
void addFence ( Vector3     position,
                float       length,
                Directions  direction,
                int         inBetweenBarAmount
              )
```

Adds a fence to the model

Parameters

position	Starting point of the fence in Unity units
length	total length in Unity units
direction	Cardinal direction
inBetweenBarAmount	Number of bars inbetween the extremeties

◆ addFence() [2/3]

```
void addFence ( Position    position,
                float       length,
                Directions  direction,
                int         inBetweenBarAmount
              )
```

Adds a fence to the model

Parameters

position	Starting point of the fence in cell units
length	total length in Unity units
direction	Cardinal direction
inBetweenBarAmount	Number of bars inbetween the extremeties

◆ addFence() [3/3]

```
void addFence ( Position    position,
                int         length,
                Directions  direction,
                int         inBetweenBarAmount
              )
```

Parameters

position	
length	
direction	
inBetweenBarAmount	Number of bars inbetween the extremeties

◆ addFloor() [1/2]

```
void addFloor ( Rectangle  rect )
```

adds a piece of floor to the building. The rectangles will be placed according to it's lower left corner

◆ addFloor() [2/2]

```
void addFloor ( Position  bottomLeft,
                Position  topRight
              )
```

adds a piece of floor to the building. The rectangles will be placed according to it's lower left corner

Parameters

rect	

◆ addFloorEverywhere()

```
void addFloorEverywhere (   )
```

Add floors for every room created

◆ addRoom() [1/2]

```
Room addRoom ( Position  position,
               int       width,
               int       depth
             )
```

Adds a room to the building queue.

Parameters

position	BottomLeft Corner of the room
width	
depth	
wallThickness	

◆ addRoom() [2/2]

```
Room addRoom ( Position  bottomLeft,
               Position  topRight
             )
```

Adds a room to the building queue.

Parameters

bottomLeft	BottomLeft Corner of the room
topRight	topRight Corner of the room
wallThickness	

◆ addSingleStair() [1/2]

```
SingleStair addSingleStair ( Vector3     bottomPosition,
                             Vector3     topPosition,
                             int         stepNumber,
                             Directions  orientation
                           )
```

Add a single stairway to the building. This method use real Unity units, and has a cell unit equivalent

Parameters

bottomPosition	Bottom left position of the stair
topPosition	top right position of the stair
stepNumber	number of steps in the stairway
orientation	Cardinal orientation. If the steps points north, the lowest point will be south, and the highest point north

◆ addSingleStair() [2/2]

```
SingleStair addSingleStair ( Position    bottomPosition,
                             Position    topPosition,
                             int         stepNumber,
                             Directions  orientation
                           )
```

Add a single stairway to the building. This method use cell units, and has a real Unity unit equivalent

Parameters

bottomPosition	Bottom left position of the stair
topPosition	top right position of the stair
stepNumber	number of steps in the stairway
orientation	Cardinal orientation. If the steps points north, the lowest point will be south, and the highest point north

◆ addWall() [1/2]

```
void HouseBuilderEngine.HouseBuilder.addWall ( Position  p1,
                                               Position  p2
                                             )
```

Adds a set of walls between p1 and p2. Positions must be vertically or horizontally aligned

◆ addWall() [2/2]

```
void HouseBuilderEngine.HouseBuilder.addWall ( Position    position,
                                               Directions  direction,
                                               int         length
                                             )
```

Adds a set of walls. Positions must be vertically or horizontally aligned

◆ addWideEntrance()

```
void RoomArchitectEngine.RoomArchitect.addWideEntrance ( Position    wall1,
                                                         Position    wall2,
                                                         GameObject  door = null,
                                                         bool        inverted = false
                                                       )
```

Places a wide opening through several walls Example: if an entrance is added between [O;O;O] and [0;0;7], the function will create the equivalent of a wide door opening (like a garage door for example) into all the wall junctions present between the two points the GameObject (being optionnal) will be placed at the bottom center of the opening

Parameters

wall1	first wall
wall2	last wall
door	Optional GameObject to place into the entrance, at the bottom center
inverted	If true, this parameters allows flipping the window 180° by its center

◆ addWindow() [1/2]

```
void RoomArchitectEngine.RoomArchitect.addWindow ( Position    position,
                                                   Directions  direction,
                                                   float       customHeightBuffer = 0,
                                                   bool        inverted = false,
                                                   int         windowID = DEFAULT
                                                 )
```

Places a window through a wall junction Example: if a window is added at [O;O;O] with direction north, the window will be placed between [0;0;0] and [0;0;1]

Parameters

position	position of the door

direction	Direction to which the door is parallel to
inverted	If true, this parameters allows flipping the window 180° by its center
doorID	ID corresponding to windows in the window's list on the unity interface

◆ addWindow() [2/2]

```
void RoomArchitectEngine.RoomArchitect.addWindow ( WallPart  wall1,
                                                   WallPart  wall2,
                                                   float     customHeightBuffer,
                                                   bool      inverted = false,
                                                   int       windowID = DEFAULT
                                                 )
```

Places a window through a wall junction Example: if a window is added between [O;O;O] and [0;0;1], it will be placed in the walls between those points, parralel to the north direction.

Parameters

wall1	first wall
wall2	second wall
inverted	If true, this parameters allows flipping the window 180° by its center
doorID	ID corresponding to windows in the window's list on the unity interface

◆ autoBuildRoof()

```
void autoBuildRoof ( ROOFTYPE  roofType = ROOFTYPE.DEFAULT )
```

Automatically generates a roof for every part that would need to be capped with a roof

◆ Build()

```
void Build ( )
```

Initiate the generation. Refer to documentation section "Building Process" for more information
WARNING: This method should not be called inside **buildTemplate()**;

◆ buildRoof()

```
void buildRoof ( Roof  roof )
```

Builds a single roof

◆ isInside() [1/2]

```
bool isInside ( Position  tile,
                int       floorOffset = 1
              )
```

returns wether or not the tile is inside any room

Parameters

tile

Returns

◆ isInside() [2/2]

```
bool isInside ( Vector3  tile,
                int      floorOffset = 1
```

)

checks wether or not the 'tile' parameter indoors

◆ isInsideRoom()

```
bool isInsideRoom ( Position  tile,
                    Room      room,
                    int       floorOffset = 1
                  )
```

checks wether or not the 'tile' parameter is inside 'room'

◆ mapRoofs()

```
List<Roof> mapRoofs (   )
```

Maps every top surface eligible to have roof

Returns

a list of singleRoof objects

◆ mergeRooms()

```
void mergeRooms ( Room  room1,
                  Room  room2
                )
```

Merges two rooms by deleting any crossing walls

NOTE: Room newRoom = new Room() does not produce a valid object to pass to mergeRooms(), the method takes Room objects that have been initialized and returned by addRoom();

◆ removeWall() [1/2]

```
void HouseBuilderEngine.HouseBuilder.removeWall ( Position  p1,
                                                  Position  p2
                                                )
```

Removes a set of walls between p1 and p2. Positions must be vertically or horizontally aligned

◆ removeWall() [2/2]

```
void HouseBuilderEngine.HouseBuilder.removeWall ( Position    position,
                                                  Directions  direction,
                                                  int         length
                                                )
```

Removes a set of walls. Positions must be vertically or horizontally aligned

◆ setElevation()

```
void setElevation ( float  elevation )
```

This method creates a buffer beneath the house and elevate the whole model

◆ setRoofStyle()

```
void RoomArchitectEngine.RoomArchitect.setRoofStyle ( ROOFTYPE  roofType,
                                                      float     height = 0.8f,
```

float flatTopSize = 0.03f
)

Sets a style for every following roof after the call. Note that "Height" and "flatTopeSize" parameters will be considered only for "default" roof type

Parameters

roofType	Type of the roof. ROOFTYPE.NONE will cancel the generation
height	height of the roof based on its width. height = 1 will generate a 45� angled roof
flatTopSize	Percentage of flat space between the 2 slopes.

◆ setStairStyle()

```
void setStairStyle ( RAILSTYLE  railStyle = RAILSTYLE.FULL,
                     bool       fullBottom = false
                   )
```

Stairs can have different styles based on the enumerator RAILSTYLE. You can also specify if the stair must be connected to the ground through all its length with the boolean "fullBottom"

◆ setTextures()

```
void setTextures ( int  indoorTexture = -1,
                   int  outdoorTexture = -1,
                   int  floorTexture = -1,
                   int  stairSteptexture = -1,
                   int  stairRailTexture = -1,
                   int  roofTexture = -1,
                   int  edgeTexture = -1,
                   int  ceilingTexture = -1
                 )
```

Sets textures for every generated meshes in the building. You can set a particular texture by using the following: setTextures(floorTexture:4) See generation examples for specific uses