InfiniteObjects Reference

Version 1 WIP
For InfiniteObjects v0.0.1-SNAPSHOT

2013

Table of contents

Goal 3

Overview 4

iWGO configuration format 5

Expressions 6

- **Mathematical expressions** 6
- **Random number functions** 7

Variables 9

Material setters 10

- **Simple setter** 10
- **Random simple setter** 11
- **Inner-outer setter** 11
- **Random inner-outer setter** 12

Conditions 14

Annex 17

- **iWGO configuration character and formatting key** 17
- **Notes** 17

Goal

The goal is InfiniteObjects is to allow users to define dynamic world generator object without the need to predefine many variations manually. Variations are achieved through random number generators used by variables and expressions. Building a dynamic world generator object is accomplished by assembling shapes. The user defines the positions and properties of the shape, as well as the materials to use when generating them by using the variables and expressions. InfiniteObjects takes more of a programming approach to defining world generator objects, and although it is not a programming language, it does make heavy use of instructions and simple mathematics.

Overview

InfiniteObjects uses a number of components to represent the various steps and concepts necessary to generate complex world generator objects. They are three main components, each having any number of types, which provide specific functionality. InfiniteObjects includes some basic types for each component. The list of components and provided types may be found bellow. Types can be added by external plugins, and may be used exactly like the provided ones.

1. Material setter
    a. Simple
    b. Random simple
    c. Inner-outer
    d. Random inner-outer
2. Condition
    a. Cuboid
    b. Sphere
3. Instruction
    a. Shape
    b. Block
    c. Repeat

Generation of a world generator object (which in this document will be referred to as a WGO) is done in three steps.

1. Evaluate the main variables
2. Evaluate each condition, terminating if any return false
3. Execute each instruction

Defining an InfiniteObjects WGO (which in this document will be referred to as an iWGO) is done using a YAML configuration. YAML is used as many Minecraft sever owners maybe familiar with it. It is also easy to read, and the Bukkit and SpoutAPI APIs provide an API for it.

iWGO configuration format

The following is the basic format for the iWGO YAML configurations. The key for the meaning of the characters and formatting used can be found at the end of the document. This format doesn't include any information specific to component types.

name: [name]

variables:
    *[name]: [**value**]*
    {…}

setters:
    [name]:
        type: [type]
        properties:
            (properties)
    {…}

conditions:
    [name]:
        type: [type]
        properties:
            (properties)
    {…}

instructions:
    [name]:
        type: [type]
        variables:
            *[name]: [**value**]*
            {…}
        properties:
            (properties)
    {…}

Expressions

Expressions are used to provide dynamic numeric quantities to the various properties. They are two types of expressions: mathematical expressions, a generic way to define values through scalars, variables, operators and functions; and random number functions, used to provided faster calculation of random numbers which can only be used for generating a random integer or floating point number between a maximum and a minimum.

Mathematical expressions

Mathematical expressions can be as simple as a unique scalar, or very complex, with various functions and variables. Evaluation of these expressions is done using the exp4j library, modified for InfiniteObjects. As part of these modifications, constants and better support for custom functions and operators have been added. A number constant, functions and operators are provided by the library and InfObjects. Please refer to the tables bellow for the complete lists.

Operators	
Operator	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Negation	‘ or -
Remainder	%
Power	^
Operation priority	() or [] or {}

Constants	
Operator	Expression
π (Pi)	PI
e (Euler’s number)	E

Functions	
Name	Expression
Floor	floor(x)
Ceiling	ceil(x)
Absolute	abs(x)
Square root	sqrt(x)
Fast square root	fsqrt(x)
Logarithm (base e)	log(x)
Sine	sin(x)

Cosine	cos(x)
Tangent	tan(x)
Arcsine	asin(x)
Arccosine	acos(x)
Arctangent	atan(x)
Random integer within [min, max]	ranI(min, max)
Random floating point within [min, max[	ranF(min max)

Mathematical expressions are written the standard linear way.

'ranI(5,10) + floor(5 * sin(angle / 2)) % 3

Operator precedency is respected. Here, "angle" is a variable, defined in the iWGO configuration under a "variables" node, either at the root, or in an instruction. It will be substituted by its value during evaluation. Variables need to be declared before they can be used.

Scientific notation is also supported.

(14 * 10 ^5) * (3 * 10 ^ -3)
can be written as
14e5 * 3e-3

A single scalar is a valid mathematical expression.

42

Random number functions

When only random numbers are required, a special notation can be used. This will lead the iWGO loader to use a different higher performance way of handling the random number generation. The notation is similar to the mathematical expression random number functions and the resulting number is identical for the same "min" and "max" arguments.

Random number functions	
Function	Expression
Random integer within [min, max]	ranI=min-max
Random floating point within [min, max[	ranF=min-max

ranI=5-10
ranF=0-1

These cannot be mixed with mathematical expressions and do not support variables.

Variables

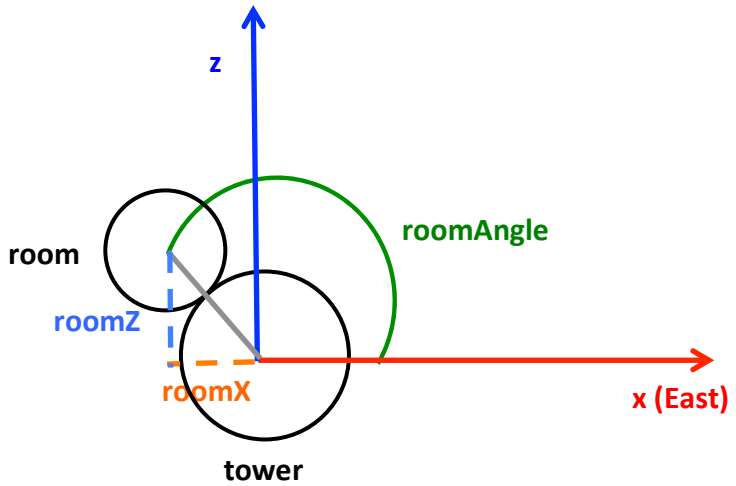
Variables are named expressions. They are defined under the "variables" nodes either at the root of the configuration (these will be referred to as global variables) or inside instructions. Global variables are evaluated once per placement. Instruction variables are evaluated every time the instruction is executed. Their purpose is to provide different attributes each time the iWGO is placed. For example: the height of a tree, the dimensions of a room or the total amount of ores in a vein. As these values are reevaluated during each placement or instruction execution, these properties will vary from object to object, creating dynamic WGOs without the need to define many variations manually. Defining a variable is very simple.

variables:
    [name]: [expression]

Example:

variables:
    towerHeight: ranI=7-11
    roomAngle: ranF(0, 2) * PI
    roomX: cos(roomAngle) * 10
    roomZ: sin(roomAngle) * 10

This declared four variables: the height of a tower, the angle from the east direction of a room attached to the side of the tower, the position on x of the room and the position on z of the room. As you can see, variables can use mathematical expressions as well as random number functions. Variables depending on other variables must be defined after their dependency. Failing to do so will result in an iWGO loading error. This means that global variables cannot depend on instruction variables. The meaning of the three last values has been schematized bellow to clarify. Please note that CamelCase has been used for the variable names. It is suggested, but not required.



Material setters

The purpose of material setters is to allow setting of blocks to various materials. Setters are provided to instructions and take care of selecting the material and applying it to the world, the instruction only telling the setter which block to set. This is especially useful when many instructions require the same block pattern. Instead of always redefining the block pattern for every instruction, it can be defined once using a material setter. Instructions do not need to make use of material setters, but all of the included ones do. Just like variables, material setters are defined under the "setters" node at the root of the configuration and are referred to using the name provided in the definition. The definition of a material setter includes the type, which can be any of the ones defined bellow, or a custom type provided by a plugin It also includes the properties, which depend on the type selected.

Included material setter types		
Name	Type	Short description
Simple	simple	Always sets the same material and data.
Random simple	random-simple	Randomly sets or not the same material and data.
Inner-outer	inner-outer	Sets a different material for the outer and inner regions.
Random inner-outer	random-inner-outer	Randomly sets or not a different material for the outer and inner regions.

The following subsections will cover all of the included material setters in more detail.

Simple setter

This is the simplest for of material setters, it will set the same material and data for all blocks.

```
setters:
    [name]:
        type: simple
        properties:
            material: [material]
            data: [data]
```

Example:

```
setters:
    jungleLeaves:
        type: simple
        properties:
            material: leaves
            data: 3
```

This defines a material setter called “jungleLeaves” of the type “simple”, which will set the material “leaves” with a data value of 3. The data node can be omitted if no particular data value should be used (default to 0).

Random simple setter

This is a slightly more complex variation of the simple setter. It will randomly set blocks or leave them unchanged, based on an odd expressed as a percentage. The odd must be an integer number greater than zero and smaller than 128 (although, since it is a percentage, anything above 100 will have the same results as 100).

```
setters:
    [name]:
        type: random-simple
        properties:
            odd: [odd]
            material: [material]
            data: [data]
```

Example:

```
setters:
    brokenCobblestone:
        type: random-simple
        properties:
            odd: 85
            material: cobblestone
```

This defines a material setter called “brokenCobblestone”. It behaves almost exactly like the simple setter, but it will only set the material “cobblestone” with data 0 85% of the time. The other 15% will remain unchanged. Again, the data key can be omitted for a default value of 0.

Inner-outer setter

Inner-outer setters are a variation of simple setters. This type will set two different materials depending on the location of the block that is being set. If the location is

considered as being in the outer region, the outer material will be used, else the inner one will be. Whether or not the region is considered as outer depends on what is using the setter. If a sphere shape is using it, then outer would be the surface, whilst inner would be the inside. This is useful for creating hallow shapes, by setting the inner material to air.

```
setters:
    [name]:
        type: inner-outer
        properties:
            outer:
                material: [material]
                data: [data]
            inner:
                material: [material]
                data: [data]
```

Example:

```
setters:
    woodedRoom:
        type: inner-outer
        properties:
            outer:
                material: wood
                data: 2
            inner:
                material: air
```

This defines an inner-outer material setter called “woodenRoom”, which will set the outer region as the material “wood” with a data value of 2. The inner material will only be air. Like the other setter, the data key can be omitted to default the data to 0. If used with a cuboid shape, this will create a wooden box, which could be used as a basic room.

Random inner-outer setter

Random inner-outer setters are a mix of inner-outer setter and random simple setters. Like the first, they will set two different materials depending on the location of the block that is being set, which can be either outside or inside. Like the second, they will randomly set blocks or leave them unchanged, based on an odd expressed as a percentage (greater than zero, less or equal to 100). This odd can be set independently for both the outside and inside material.

```
setters:
    [name]:
        type: random-inner-outer
```

properties:
outer:
odd: [odd]
material: [material]
data: [data]
inner:
odd: [odd]
material: [material]
data: [data]

Example:

```
setters:
    abandonnedStoneBrickRoom:
        type: random-inner-outer
        properties:
            outer:
                odd: 90
                material: stone_brick
                data: 2
            inner:
                odd: 10
                material: spider_web
```

This defines a random inner-outer setter called “abandonnedStoneBrickRoom”, which will set the outer region as the material “stone_brick” with a data value of 2. The inner material will contain a few blocks with the material “spider_web”. Like the other setters, the data key can be omitted to default the data to 0. If used with a cuboid shape, this will create a stone box with broken walls containing a few spider webs, which could be used as a basic abandoned room.

Conditions

Conditions are used to verify if the location at which the iWGO is being placed it valid. They are many criteria that can be used to determine whether or not an area is suitable for a WGO, but the simplest is to verify a volume to see which materials it contains. Depending on what is found, the condition will be true or false. The types included with InfiniteObjects are based on this idea.

This section will cover the included types. Bellow is the configuration format for the types.

```
conditions:
    [name]:
        type: [type]
        properties:
            mode: ["include" | "exclude"]
            size:
                x: [x]
                y: [y]
                z: [z]
            position:
                x: [x]
                y: [y]
                z: [z]
            check:
                - [material]
                {...}
    {...}
```

Conditions can be omitted if not required. Two types are included with InfiniteObjects. These vary only by their shape, and not by how they work.

Included condition types	
Name	Type
Cuboid	cuboid
Sphere	sphere

For example, if the goal were to create a tree, it would be necessary to verify that it would not cut through mountain overhangs or other floating features, or be submerged. A condition for this would check a cuboid volume, about as big as the tree, for the presence of stone, dirt, grass, water or any other terrain material. If the volume includes any of these materials, the condition would be false, as the tree would be placed within. This means that the condition is excluding the materials listed above. Defining such a condition would look like so.

conditions:

```
volumeContents:
        type: cuboid
        properties:
                mode: exclude
                size:
                        x: 5
                        y: height
                        z: 5
                position:
                        x: -2
                        y: 0
                        z: -2
                check:
                        - stone
                        - dirt
                        - grass
                        - water
```

This defines a condition called “volumeContents” which will be true if it excludes stone, dirt, grass and water. The volume is a cuboid of length 5, height “height” (this is a variable for the height of the tree) and width 5. The volume’s lower corner is at -2 on x, 0 on y and -2 on x. The position is not the middle of the cuboid, but its lower corner.

Continuing with the example, a tree should only be placed on dirt or grass. To check this, we create a second condition that will be true if the volume of the blocks directly under the trunk of the tree includes only the grass and dirt materials.

(continued)

```
        surfaceContents:
                type: cuboid
                properties:
                        mode: include
                        size:
                                x: 1
                                y: 1
                                z: 1
                        position:
                                x: 0
                                y: -1
                                z: 0
                        check:
                                - dirt
                                - grass
```

This creates a condition called “surfaceContents”, which will be true if the cuboid volume of length 1, height 1 and depth 1, positioned at 0 on x, -1 on y and 0 on z, contains only grass and dirt.

The sphere shape will not be covered here, but it works the same way, only the position is at the center, and not the lower corner. This also means that the size values for x, y and z represent the radiuses on these axes. Having a radius twice as big on z compared to x and y will result in an elliptical volume twice as deep as it is large or high.

Finally, they are only two accepted modes: “exclude” and “include”. Plugins cannot add more for the “cuboid” and “sphere” types.

Annex

iWGO configuration character and formatting key

- Italic elements are optional
- Bold elements must be mathematical expressions or random number functions
- Underlined elements can only be constant scalars (no expressions)
- Elements between brackets are to be defined by the user, unless between quotes. The "|" character means that one, but not both, of the quoted terms must be used.
- Elements between parentheses represent a section of the configuration, which cannot be defined as it depends on the type of component.
- An ellipsis between braces mean that the key above at the same rank and its children are to be repeated (unless in italics). For lists, it means that more items can be added to the list

Notes

- "InfObjects" and "InfiniteObjects" can be used interchangeably, and refer to the same software and project, the official name being the second.