Incoming! Random Obstacle Generator for Unity
Documentation

70:30
http://70-30.de
info@70-30.de

Introduction

Thank you for buying Incoming! - A very flexible random obstacle generator for Unity to cast any kind of obstacles for your tetris game, runner, platformer, space shooter or any other game you like. Incoming! has many options and settings you can easily adapt to your project.

Overview

This documentation provides basic information about Incoming! and how to configure the generator. It is separated in two parts - a quick guide how to get started and a more detailed part with all important information and what to do step by step.
For any comments, suggestions or bugs please write a mail to info@70-30.de. If you like the unity package, please vote for it in the asset store :)

HINT: Incoming! works hand in hand with other templates from 70:30. If you own Shift! the 3D menu controller, you can import Incoming! additionally and just add the features to your project - or the other way around. Incoming! uses the same game setup system as Shift! so if you call a game state with the „Seventythirty.cs" controlling script, both Shift! and Incoming! will interact.

Configuring the generator - in general

Basically Incoming! offers the following components:
- A **7030Controller** prefab you can drag and drop in your scene. It is used to define all your menu/game states (main menu, settings, pause, playing, and so on). The prefab is available in all 70:30 products, but not implicitely necessary for the use of Incoming! You can use it if you like (see the example scene for it's use) to benefit from it's advantages, but you dont need to
- A **ObstacleGenerator** prefab - the actual generator
- An **ObstacleTemplate** prefab you can use to create your own obstacles
- A demo scene for you to see Incoming! integrated in a project

The workflow in short to configure Incoming! will look as follows:

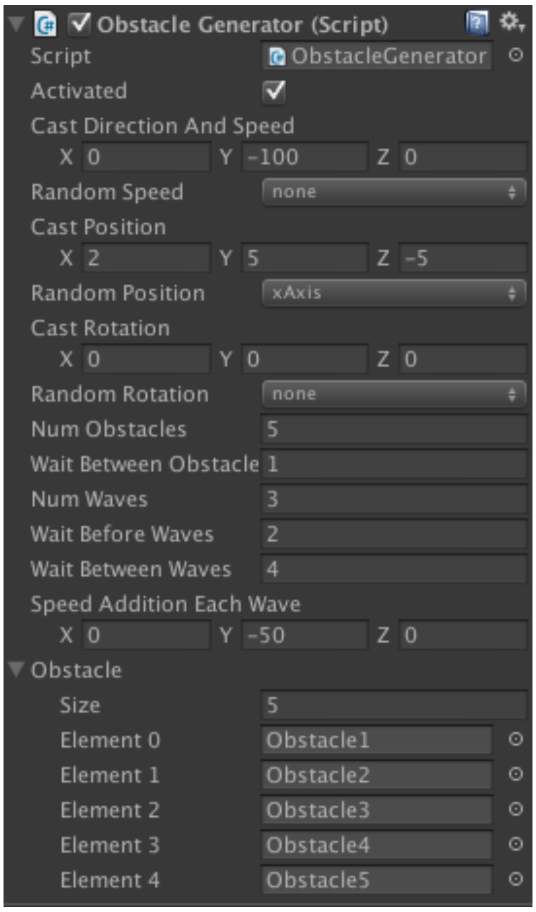
1) Drag the **ObstacleGenerator** prefab in your scene
2) Configure it as you like
3) Create obstacles and assign them to the generator

Configuring the generator - in detail

If you would like to see the generator in a project, please try the example scene provided with the package. Use the sliders to see some options of the generator variable during use. It should be pointed out that the sliders in the demo scene are just examples and the generator itself provides a lot more settings.

1) Drag the ObstacleGenerator prefab in your scene
To get started, drag the prefab in your scene found in the „prefabs“ folder.

2) Configure it as you like
When you click on the prefab, it will look like this.


Obstacle Generator (Script)
Script ObstacleGenerator
Activated ✓
Cast Direction And Speed
X 0 Y -100 Z 0
Random Speed none
Cast Position
X 2 Y 5 Z -5
Random Position xAxis
Cast Rotation
X 0 Y 0 Z 0
Random Rotation none
Num Obstacles 5
Wait Between Obstacle 1
Num Waves 3
Wait Before Waves 2
Wait Between Waves 4
Speed Addition Each Wave
X 0 Y -50 Z 0
Obstacle
Size 5
Element 0 Obstacle1
Element 1 Obstacle2
Element 2 Obstacle3
Element 3 Obstacle4
Element 4 Obstacle5

Let‘s go through the options.

Activated:
trigger this on or off to start or end the generator.

Cast Direction And Speed:
This Vector3 will define the direction and the speed in one. By entering a value in an axis, the obstacles will be cast in this direction. The higher ne number, the faster the obstacles will fly. If you want to cast in the opposite direction, use negative values. Of course you can enter values in several fields if you want diagonal/3D space casting.

Random Speed:
By selecting an option here you mark the axis to use random values, depending on what you endered in the field. If you entered 100 in Y above, each obstacle will be cast with a random speed between 0 an 100.

Cast Position:
This will set where the obstacles will be cast in your space. Default settings are top of the screen.

Random Position:
This will be important for a lot of games. If you make a tetris game with blocks coming from the upper screen, you will like to have the objects cast at random x coordinates. If you have entered 5 in the X field above, obstacles will be cast at integer values between -5 and 5, making a nice random-objects-from-above-effect.

Cast Rotation:
The rotation of the objects in euler angles. Use Z values if you make a 2D game that is focused from the side.

Random rotation:
Selecting a field for random rotation will cast the objects in the selected axis with 0-359 degrees at random.

Num Obstacles:
The number of obstacles cast each wave.

Wait Between Obstacles:
The time to wait between obsacle casting. A value of 1 will cast one obstacle per second.

Num Waves:
The number of casting waves. If the value is 3 and Num Obstacles is 5, there will be three waves, each casting five obstacles. Then the generator will not do anything until it is deactivated and activated again.

Wait Before Waves:
The time it will take until the waves will start. Actually the time before the generator will start casting anything when it is activated.

Wait Between Waves:
The time it will take until the next wave starts. A value of 4 will delay each next wave for 4 seconds.

Speed Addition Each Wave:
This Vector3 works similar as the „Cast Direction And Speed“ Vector3, just that it‘s the speed that will be added every wave. Let‘s say you have a tetris game with several blocks each wave and you want to increase the difficulty after each wave by a speed of 50, you can enter the value and it will be added to the Cast Direction And Speed value, making the obstacles fly faster over time.

Obstacle:
This array of game objects will allow you to enter the number of obstacles you want to assign. Then you can drag and drop any game objects you want in the element slots. By default, 5 example obstacles are assigned. You can replace them with your own objects.

3) Create obstacles and assign them to the generator
Now you can create your own obstacles and add them to the generator. Drag and drop an **ObstacleTemplate** into your scene which has the required components rigidbody and DestroyByOption.cs script already attached.
The script lets you choose the option to remove obstacles again after they were cast, so your game will not start to lag due to increasing numbers of obstacles that fly towards eternity.

The script offers following options:

Destroy Out Of Screen:
Flag if you want an object to be removed when it‘s out of the screen.

Destroy By Collision:
Flag if you want an object to be removed when it collides with any other object. You have modify the script if you want to specify this.

Destroy By Time:
Probably one of the best options. Just remove the obstacle when a certain amount of seconds has passed. Enter your seconds in the field. If you enter 0, the obstacle will not be destroyed by time.

Name Of Destroy Boundary:
Some games use a game object that defines a boundary of the game, i.e. using a box collider covering the game area. If you want to remove your obstacles this way, just enter the name of the boundary object in this field.

Now the rest is up to you. When you create a 2D game, you can use the default sprite renderer for your obstacle sprite. But of course you can add any child objects to the prefab like 3D objects to fit your game.
Play around with the options of the **ObstacleGenerator** to see all it's settings and how they work and create your own game with it.

We hope you benefit from Incoming! Please write questions, suggestions, bugs or anything else to info@70-30.de.

70:30