



GAME DEVELOPMENT > PROCEDURAL CONTENT GENERATION



Create a Procedurally Generated Dungeon Cave System

by [CaptainKraft](#) 19 Aug 2013

Difficulty: Intermediate Length: Long Languages: English ▼

Procedural Content Generation

Platform Agnostic

Level Design

2D Games



Tile-Based Games

Haxe

For many, procedural generation is a magical concept that is just out of reach. Only veteran game developers know how to build a game that can create its own levels... right? It may *seem* like magic, but PCG (procedural content

generation) can be learned by beginner game developers. In this tutorial, I'll show you how to procedurally generate a dungeon cave system.

What We're Going to Cover

Here's a SWF demo that shows the kind of level layouts that this technique can generate:

Click the SWF to generate a new level.

Learning the basics usually means a lot of Google searching and experimentation. The problem is, there are very few *simple* guides on how to get started. For reference, here are some excellent sources of information on the subject, which I have studied:

- [Complete Roguelike Tutorial \(Python and libtcod\)](#)
- [Grid-Based Dungeon Generation](#)
- [PCG Wiki](#)

Before getting into the details, it's a good idea to consider how we're going to solve the problem. Here are some easy-to-digest chunks that we will use to keep this thing simple:

1. Randomly place your created content into the game world.
2. Check that the content is placed in a spot that makes sense.
3. Check that your content is reachable by the player.
4. Repeat these steps until your level comes together nicely.

Once we've worked through the following examples, you should have the necessary skills to experiment with PCG in your own games. Exciting, eh?

Where Do We Place Our Game Content?

The first thing we are going to do is randomly place the rooms of a procedurally generated dungeon level.

In order to follow along, it's a good idea to have a basic understanding of how tile maps work. In case you need a quick overview or a refresher, [check out this tile map tutorial](#). (It is geared towards Flash but, even if you aren't familiar with Flash, it's still good for getting the gist of tile maps.)

Creating a Room to Be Placed in Your Dungeon Level

Before we get started we have to fill our tile map with wall tiles. All you need to do is iterate through every spot in your map (a 2D array, ideally) and place the tile.

We also need to convert the pixel coordinates of each rectangle to our grid coordinates. If you want to go from pixels to grid location, divide the pixel coordinate by the tile width. To go from grid to pixels, multiple the grid coordinate by the tile width.

For example, if we want to place our room's top left corner at `(5, 8)` on our grid and we have a tile width of `8` pixels, we would need to place that corner at `(5 * 8, 8 * 8)` or `(40, 64)` in pixel coordinates.

Let's create a `Room` class; it might look like this in Haxe code:

```
01 class Room extends Sprite {
02     // these values hold grid coordinates for each corner of the room
03     public var x1:Int;
04     public var x2:Int;
05     public var y1:Int;
06     public var y2:Int;
07
08     // width and height of room in terms of grid
09     public var w:Int;
10     public var h:Int;
11
12     // center point of the room
13     public var center:Point;
14
15     // constructor for creating new rooms
16     public function new(x:Int, y:Int, w:Int, h:Int) {
17         super();
18
19         x1 = x;
20         x2 = x + w;
21         y1 = y;
22         y2 = y + h;
23         this.x = x * Main.TILE_WIDTH;
24         this.y = y * Main.TILE_HEIGHT;
25         this.w = w;
26         this.h = h;
27         center = new Point(Math.floor((x1 + x2) / 2),
28                             Math.floor((y1 + y2) / 2));
29     }
30
31     // return true if this room intersects provided room
32     public function intersects(room:Room):Bool {
33         return (x1 <= room.x2 && x2 >= room.x1 &&
34                 y1 <= room.y2 && room.y2 >= room.y1);
35     }
36 }
```

We have values for each room's width, height, center point position, and four corners' positions, and a function that tells us whether this room intersects another one. Also note that everything except for the x and y values are in our

grid coordinate system. This is because it makes life much easier to use small numbers each time we access the room values.

Okay, we have the framework for a room in place. Now how do we procedurally generate and place a room? Well, thanks to built-in random number generators, this part isn't too difficult.

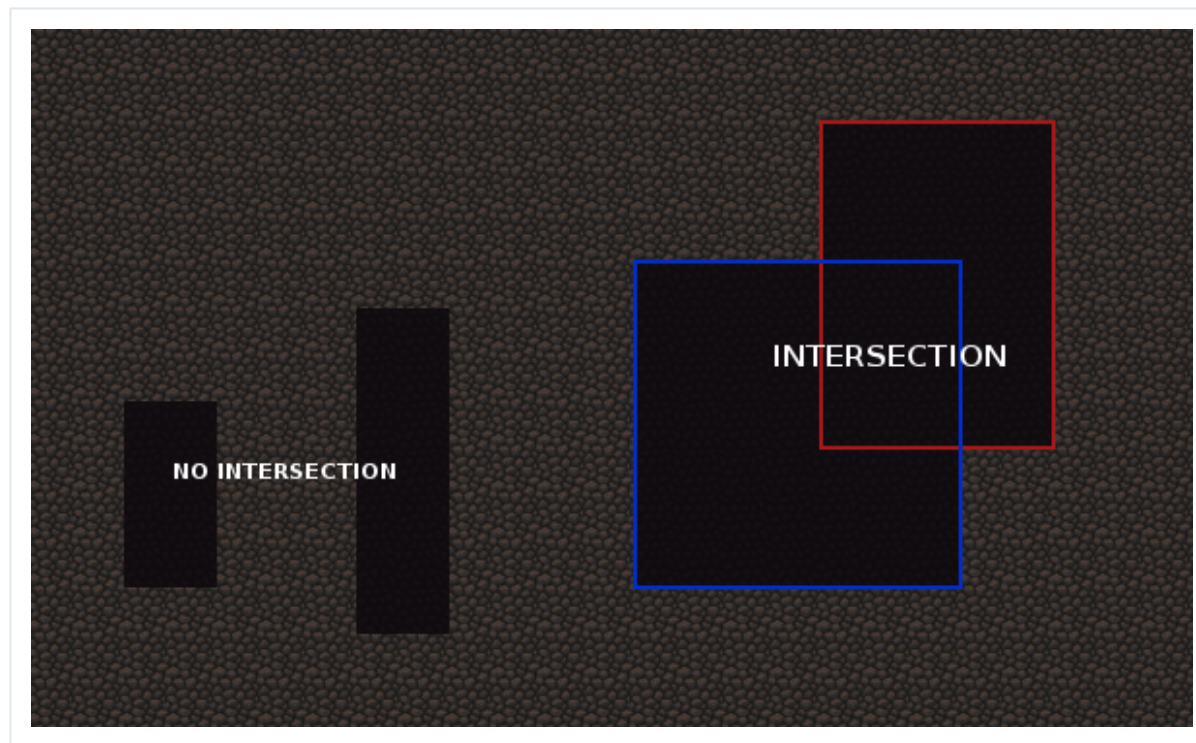
All we have to do is provide random x and y values for our room within the bounds of the map, and give random width and height values within a predetermined range.



Does Our Random Placement Make Sense?

Since we are using random locations and dimensions for our rooms, we are bound to overlap with previously created rooms as we fill our dungeon. Well, we already coded up a simple `intersects()` method in order to help us take care of the problem.

Each time we attempt to place a new room, we simply call `intersects()` on each pair of rooms within the entire list. This function returns a Boolean value: `true` if the rooms are overlapping, and `false` otherwise. We can use that value to decide what to do with the room we just attempted to place.



Check back at the `intersects()` function. You can see how the x and y values overlap and return `true`.

```
01 | private function placeRooms() {  
02 |     // create array for room storage for easy access  
03 |     rooms = new Array();
```

```

04 // randomize values for each room
05 for (r in 0...maxRooms) {
06     var w = minRoomSize + Std.random(maxRoomSize - minRoomSize + 1);
07     var h = minRoomSize + Std.random(maxRoomSize - minRoomSize + 1);
08     var x = Std.random(MAP_WIDTH - w - 1) + 1;
09     var y = Std.random(MAP_HEIGHT - h - 1) + 1;
10
11     // create room with randomized values
12     var newRoom = new Room(x, y, w, h);
13
14     var failed = false;
15     for (otherRoom in rooms) {
16         if (newRoom.intersects(otherRoom)) {
17             failed = true;
18             break;
19         }
20     }
21     if (!failed) {
22         // local function to carve out new room
23         createRoom(newRoom);
24
25         // push new room into rooms array
26         rooms.push(newRoom)
27     }
28 }
29 }
30 }

```

The key here is the `failed` Boolean; it's set to the return value of `intersects()`, and so is `true` if (and only if) your rooms are overlapping. Once we break out of the loop, we check this `failed` variable and, if it's false, we can carve out the new room. Otherwise, we just discard the room and try again until we've hit our maximum number of rooms.

How Should We Handle Unreachable Content?

The vast majority of games that use procedurally generated content strive to make all of that content reachable by the player, but there are a few people out there who believe that this isn't necessarily the best design decision. What if you had some rooms in your dungeon that the player could only rarely get to but could always see? This might add an interesting dynamic to your dungeon.

Of course, no matter which side of the argument you are on, it's probably still a good idea to make sure that the player can always progress through the game. It would be pretty frustrating if you got to a level of the game's dungeon and the exit was completely blocked off.

Considering that most games shoot for 100% reachable content, we'll stick with that.

Let's Handle That Reachability

By now, you should have a tile map up and running and there should be code in place to create a variable number of rooms of varying sizes. Look at that; you already have some clever procedurally generated dungeon rooms!

Now the goal is to connect each room so that we can walk through our dungeon and eventually reach an exit that leads to the next level. We can accomplish this by carving out corridors between the rooms.

We will need to add a `point` variable to the code to keep track of the center of each room created. Whenever we create and place a room, we determine its center and connect it to the previous room's center.

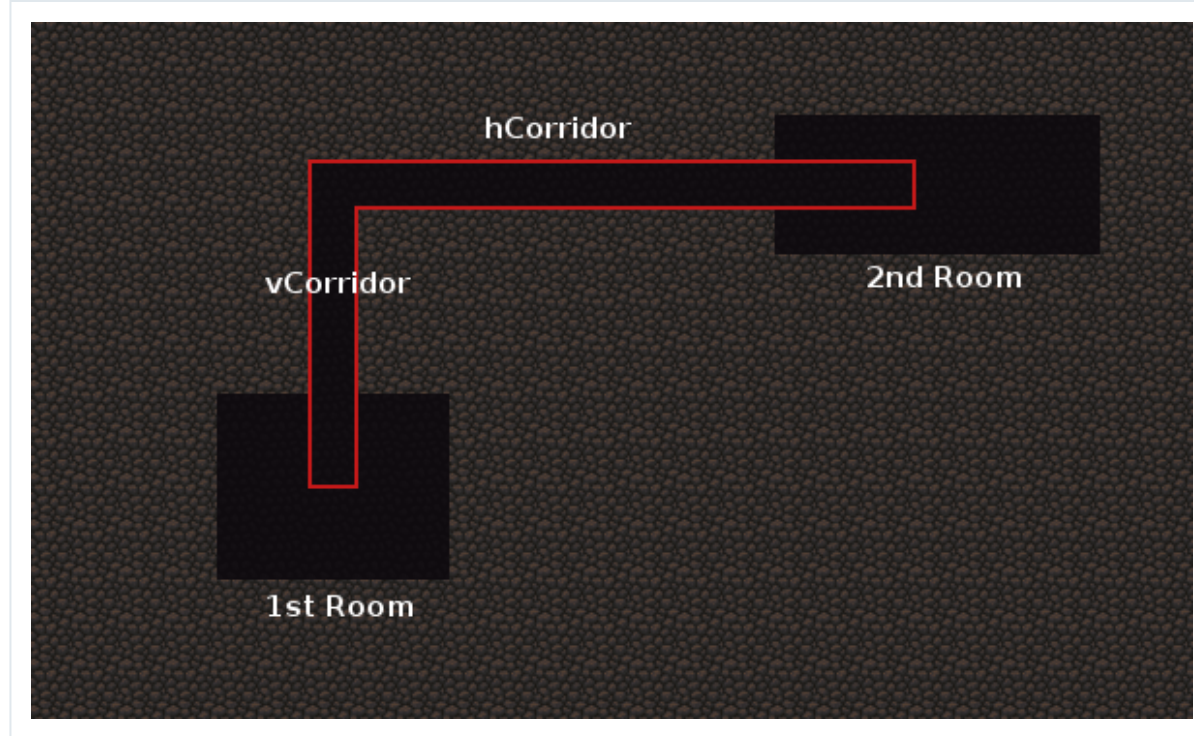
First, we'll implement the corridors:


```

01 private function hCorridor(x1:Int, x2:Int, y) {
02     for (x in Std.int(Math.min(x1, x2))...Std.int(Math.max(x1, x2)) + 1
03         // destroy the tiles to "carve" out corridor
04         map[x][y].parent.removeChild(map[x][y]);
05
06         // place a new unblocked tile
07         map[x][y] = new Tile(Tile.DARK_GROUND, false, false);
08
09         // add tile as a new game object
10         addChild(map[x][y]);
11
12         // set the location of the tile appropriately
13         map[x][y].setLoc(x, y);
14     }
15 }
16
17 // create vertical corridor to connect rooms
18 private function vCorridor(y1:Int, y2:Int, x) {
19     for (y in Std.int(Math.min(y1, y2))...Std.int(Math.max(y1, y2)) + 1
20         // destroy the tiles to "carve" out corridor
21         map[x][y].parent.removeChild(map[x][y]);
22
23         // place a new unblocked tile
24         map[x][y] = new Tile(Tile.DARK_GROUND, false, false);
25
26         // add tile as a new game object
27         addChild(map[x][y]);
28
29         // set the location of the tile appropriately
30         map[x][y].setLoc(x, y);
31     }
32 }

```

These functions act in nearly the same way, but one carves out horizontally and the other vertically.



Connecting the first room to the second room requires a `vCorridor` and an `hCorridor`.

We need three values in order to do this. For horizontal corridors we need the starting x value, the ending x value, and the current y value. For vertical corridors we need the starting and ending y values along with the current x value.

Since we are moving from left to right we need the two corresponding x values, but only one y value since we won't be moving up or down. When we move vertically we will need the y values. In the `for` loop at the beginning of each function, we iterate from the starting value (x or y) to the ending value until we have carved out the entire corridor.

Now that we have the corridor code in place, we can change our `placeRooms()`

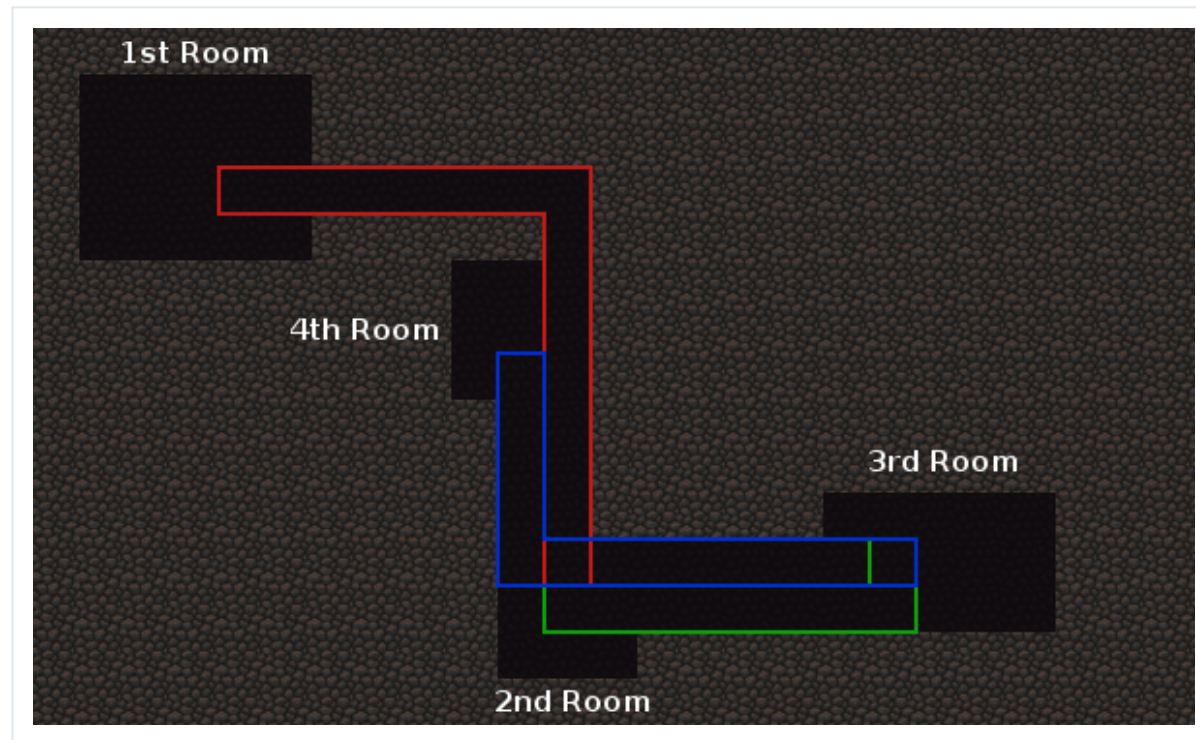
function and call our new corridor functions:

```
01 private function placeRooms() {
02 // store rooms in an array for easy access
03 rooms = new Array();
04
05 // variable for tracking center of each room
06 var newCenter = null;
07
08 // randomize values for each room
09 for (r in 0...maxRooms) {
10     var w = minRoomSize + Std.random(maxRoomSize - minRoomSize + 1);
11     var h = minRoomSize + Std.random(maxRoomSize - minRoomSize + 1);
12     var x = Std.random(MAP_WIDTH - w - 1) + 1;
13     var y = Std.random(MAP_HEIGHT - h - 1) + 1;
14
15     // create room with randomized values
16     var newRoom = new Room(x, y, w, h);
17
18     var failed = false;
19     for (otherRoom in rooms) {
20         if (newRoom.intersects(otherRoom)) {
21             failed = true;
22             break;
23         }
24     }
25     if (!failed) {
26         // local function to carve out new room
27         createRoom(newRoom);
28
29         // store center for new room
30         newCenter = newRoom.center;
31
32         if(rooms.length != 0){
33             // store center of previous room
34             var prevCenter = rooms[rooms.length - 1].center;
35
36             // carve out corridors between rooms based on centers
37             // randomly start with horizontal or vertical corridors
38             if (Std.random(2) == 1) {
39                 hCorridor(Std.int(prevCenter.x), Std.int(newCenter.x),
40                     Std.int(prevCenter.y));
41                 vCorridor(Std.int(prevCenter.y), Std.int(newCenter.y),
```

```

42         Std.int(newCenter.x));
43     } else {
44         vCorridor(Std.int(prevCenter.y), Std.int(newCenter.y),
45             Std.int(prevCenter.x));
46         hCorridor(Std.int(prevCenter.x), Std.int(newCenter.x),
47             Std.int(newCenter.y));
48     }
49 }
50 }
51 if(!failed) rooms.push(newRoom);
52 }
53 }

```



In the above image, you can follow the corridor creation from the first to the fourth room: red, green, then blue. You can get some interesting results depending on the placement of the rooms - for instance, two corridors next to each other make a double-wide corridor.

We added some variables for tracking the center of each room and we attached the rooms with corridors between their centers. Now there are multiple non-overlapping rooms and corridors that keep the entire dungeon level connected. Not bad.





We're Done With Our Dungeon, Right?

You've come a long way building your first procedurally generated dungeon level, and I'm hoping you've realized that PCG isn't some magical beast that you will never have a chance to slay.

We went over how to randomly place content around your dungeon level with simple random number generators, and a few predetermined ranges for keeping your content the right size and roughly in the right place. Next, we discovered a very simple way to determine if your random placement made sense by checking for overlapping rooms. Lastly, we talked a little bit about the merits of keeping your content reachable and we found a way to ensure that your player can reach every room in your dungeon.

The first three steps of our four step process are finished, which means that you have the building blocks of a great dungeon for your next game. The final step is down to you: you must iterate over what you learned to create more procedurally generated content for endless replayability.

There Is Always More to Learn

The method for carving out simple dungeon levels in this tutorial only scratches the surface of PCG and there are some other simple algorithms that you can easily pick up.

My challenge for you is to start experimenting with the beginnings of your game that you created here and to do some research into more methods to change up your dungeons.

One great method for creating cave levels is using [cellular automata](#), which has infinite possibilities for customizing dungeon levels. Another great method to learn is [Binary Space Partitioning \(BSP\)](#), which creates some wicked looking gridlike dungeon levels.

I hope this gave you a good jump start into procedural content generation. Make sure to comment below with any questions you have, and I'd love to see some examples of what you are creating with PCG.

Related Posts

- [Generate Random Cave Levels Using Cellular Automata](#)



CaptainKraft

When I was 10 years old I decided that I was going to become a game developer. The "growing up" part hasn't happened yet but I love making games anyway. I also like to talk about gamedev over on GamesKraft and Twitter.

 [captainkraft](#)

 [FEED](#)  [LIKE](#)  [FOLLOW](#)  [FOLLOW](#)

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Game Development tutorials. Never miss out on learning about the next big thing.

[Update me weekly](#)

Download Attachment

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

[Translate this post](#)


Advertisement

25 Comments

Gamedevtuts+

 Login ▾

 Recommend 5

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



Alberto Gómez R • 4 years ago

Amazing!! It was exactly what I was looking for :D

7 ^ | v • Reply • Share ›



Joshua Homer • 4 years ago

Have a quick question pertaining to this code:

```
// return true if this room intersects provided room
```

```
public function intersects(room:Room):Bool {  
    return (x1 <= room.x2 && x2 >= room.x1 && y1 <= room.y2 && room.y2 >= room.y1);  
}
```

Should the return actually read:

```
return (x1 <= room.x2 && x2 >= room.x1 && y1 <= room.y2 && y2 >= room.y1);  
where room.y2 is actually y2? Just wondering...
```

Love the read so far and am implementing this concept into Javascript and easelJS

3 ^ | v • Reply • Share ›



Bide • 4 years ago



Nice tut thank you

3 ^ | v • Reply • Share ›



Animebryan • 2 years ago

Can you offer a tutorial for javascript? I'm using RPG Maker MV & no ones seems to be working on a plugin that can do this.

1 ^ | v • Reply • Share ›



Michael James Williams Mod ➔ Animebryan • 2 years ago

JavaScript and ActionScript aren't too dissimilar, so you might be able to straight copy some of the code over... have a go and let me know how you get on!

^ | v • Reply • Share ›



Animebryan ➔ Michael James Williams • 2 years ago

Well I don't know how to program in any language (no experience) & was wondering if someone could make a RPG Maker MV version of it?

There have been people on the RPG Maker forums who got around to making a dungeon generator for VX Ace but no one seems to want to take on the task of doing it for MV.

I want to learn how write javascript plugins for RPG Maker MV but I don't know where to start & I don't have internet at home & have limited time to be on the internet in the computer lab that I'm in.

1 ^ | v • Reply • Share ›



Michael James Williams Mod → Animebryan • 2 years ago

This is way outside of my expertise, I'm afraid. I recommend you start by learning JavaScript itself, even if it's for the general web rather than for games specifically. (There are plenty of great books out there so you can learn offline -- try Head First JavaScript <http://shop.oreilly.com/pro...> to start with.)

Once you know the language, you can learn how to use it with RPG Maker MV -- or with something like Phaser, which is entirely JavaScript; you'll have more options open to you.)

And then you should be able to come back to this tutorial, understand the core concepts, and translate it to JavaScript in a way that suits you! Good luck!

^ | v • Reply • Share ›



Bobby • 2 years ago

Fantastic article, a true inspiration to aspiring game developers such as myself. Well done sir.

1 ^ | v • Reply • Share ›



Winter Mute • 3 years ago

I only read the first part of your tutorial about generating the rooms, and from there I just went crazy and did my own thing (I scanned the rest of it just before posting this comment). Anyways, that little nudge sent me over the deep end, and this is the result (my first Java program :P)



1 ^ | v • Reply • Share ›



Winter Mute → Winter Mute • 3 years ago

I'm building a player object to wander around the caves later today, and after that I'm going to be adding three more dimensions to my primary map array, one as a Z index for floors, another to store interactive objects, and a final one

one as a Z-index for floors, another to store interactive objects, and a final one to house mobs (and the player). Thanks a lot for push, I was really confused how to get into PCG, but the spark from this ignited a fire in my mind.

2 ^ | v • Reply • Share ›



Winter Mute ➔ Winter Mute • 3 years ago

Oh errr, this was done in Java, I'm trying to learn to program games and I'm a huge sucker for roguelikes.

^ | v • Reply • Share ›



Tom Gattenhof • 4 years ago

Thanks for the tutorial, as a Flash guy moving to HTML5 I used the ideas you put forward to start my own little dungeon generation engine. Many thanks!

1 ^ | v • Reply • Share ›



lewis lepton • 7 months ago

great article.

the source code is not available though - may be worth making it available again so new learners can get to grips with such a practice without having to decipher what is written ;)

^ | v • Reply • Share ›



Jota Vilchez • 8 months ago

Sorry but i guess placeRooms function have a memory leak, one "new" with possible no "delete" :(

```
if (!failed)
{
    .....
    // push new room into rooms array
    roomLayer.rooms.push_back(newRoom);
}
else
{
    delete newRoom; <----- missing
}
```

^ | v • Reply • Share ›

Avatar This comment was deleted.



Liam Bury → Guest • 3 years ago

It's the Java version of what in C# is `foreach(itemCalledX in someList)`.

^ | v • Reply • Share ›



adsf → Liam Bury • 3 years ago

Close. The correct Java syntax uses `:` instead of the word `in`.

2 ^ | v • Reply • Share ›



Thomas Henry Huxley • 3 years ago

I've spent a considerable amount of time on this idea, and I came to the conclusion that the best and most efficient way to generate a level is to run a randomized snake-like function through the area, have it randomly place room markers, then drop the pre-built rooms in. I did it all in Python using Blender's game engine. I do need to modify it to change the way the ceiling is built though, so it will be easier to build multiple levels. I have all the source code on my website. www.tachufind.com under the Game menu.

I was a big fan of the game Moria, similar to Angeband, which had rooms with internal parts, like columns, inner rooms, etc., and I wanted to replicate that.

^ | v • Reply • Share ›



ZigZalgo • 3 years ago

Hey I know it's been quite some time since you posted this, But I cannot for the life of me understand the placing of the corridors, could you perhaps explain it a little more for me?

^ | v • Reply • Share ›



Lee Stemkoski • 3 years ago

Thanks for writing this! Your article is very clearly written and the diagrams are great. I used your approach to create a rogue-like game, and inspired by your article I wrote up a short tutorial on how to implement this in Construct 2. If you're interested, it's available online at <http://www.scirra.com/tutorial...>

Thanks again!

^ | v • Reply • Share ›



kaappi • 3 years ago

Thanks for the tutorial. It wasn't really the thing I was looking for, but it was still very helpful, clear and fun ^^

^ | v • Reply • Share ›



jeff • 4 years ago

create some c# based on this tutorial

^ | v • Reply • Share ›



MrPoulet • 4 years ago

great ! 5 stars

^ | v • Reply • Share ›



Ron Krepps • 4 years ago

I don't quite understand this code:

```
public function new(x:Int, y:Int, w:Int, h:Int) {  
    super();
```

```
    x1 = x;
```

```
    x2 = x + w;
```

```
    y1 = y;
```

```
    y2 = y + h;
```

```
    this.x = x * Main.TILE_WIDTH;
```

```
    this.y = y * Main.TILE_HEIGHT;
```

```
    this.w = w;
```

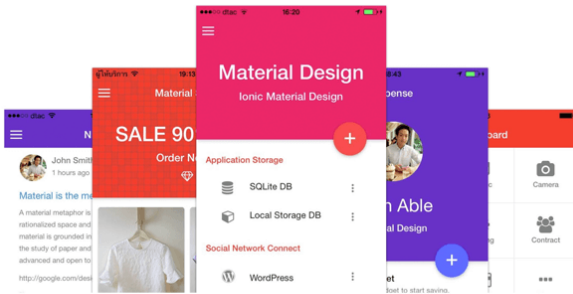
```
    this.h = h;
```

```
    center = new Point(Math.floor((x1 + x2) / 2)
```


Advertisement

LOOKING FOR SOMETHING TO HELP KICK START YOUR NEXT PROJECT?

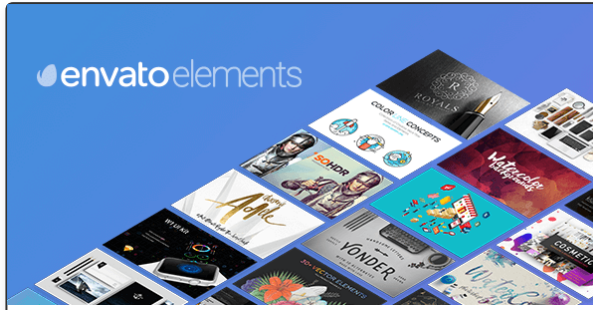
Envato Market has a range of items for sale to help get you started.



Mobile Game Templates
From \$4

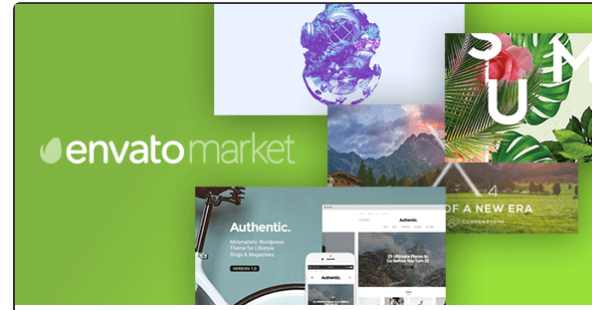


HTML5 Games
From \$7



Unlimited Downloads
Only \$29/month

Get access to over 18,000 creative assets on Envato Elements.



Over 9 Million Digital Assets

Everything you need for your next creative project.

ENVATO TUTS+

[About Envato Tuts+](#)
[Terms of Use](#)
[Advertise](#)

JOIN OUR COMMUNITY

[Teach at Envato Tuts+](#)
[Translate for Envato Tuts+](#)
[Forums](#)
[Community Meetups](#)

HELP

[FAQ](#)
[Help Center](#)



tuts+

24,641
Tutorials

1,062
Courses

16,702
Translations

[Envato.com](#) [Our products](#) [Careers](#)

© 2017 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

Follow Envato Tuts+

