

Rooms and Mazes: A Procedural Dungeon Generator



DECEMBER 21, 2014

CODE DART GAME-DEV ROGUELIKE



Several months ago I promised a follow-up to my previous blog post about [turn-based game loops](#) in [my roguelike](#). Then I got completely sidetracked by [self-publishing my book](#), *Game Programming Patterns*, and forgot all about it. I totally left you hanging.

Well, I finally got some time to think about my roguelike again and today, I'm here to... keep you hanging. Alas, you are at the mercy of my wandering attention span! Instead of game loops, today we're going to talk about possibly the most fun and challenging part of making a roguelike: generating dungeons!

Go ahead and click the little box below to see what we end up with:

Hi! I'm Bob Nystrom, the one on the left.

I wrote a book called [Game Programming Patterns](#).

You can email me at robert@nystrom.net at this site or follow me on twitter at [@munificentbob](https://twitter.com/munificentbob).

ELSEWHERE

- Code at [github](#)
- Code at [bitbucket](#)
- Tweets at [twitter](#)
- Photos at [flickr](#)
- Video at [vimeo](#)
- Posts at [google+](#)

CATEGORIES

- [code](#) 67
- [language](#) 41
- [magpie](#) 24
- [c-sharp](#) 13
- [dart](#) 13
- [game-dev](#) 12



Click it again to restart it.

Pretty neat, huh? If you want to skip the prose, the code is [here](#).

One of my earliest memories of computing is a maze generator running on my family's Apple IIe. It filled the screen with a grid of green squares, then incrementally cut holes in the walls. Eventually, every square of the grid was connected and the screen was filled with a complete, perfect maze.

My little home computer could create something that had deep structure—every square of the maze could be reached from any other—and yet it seemed to be chaotic—it carved at random and every maze was different. This was enough to blow my ten-year-old mind. It still kind of does today.

What's in a dungeon?

- [java](#) 10
- [cpp](#) 8
- [game-patterns](#) 6
- [parsing](#) 6
- [roguelike](#) 6
- [design](#) 5
- [go](#) 5
- [js](#) 4
- [book](#) 3
- [c](#) 3
- [finch](#) 3
- [python](#) 3
- [ruby](#) 3
- [blog](#) 2
- [f-sharp](#) 2
- [lua](#) 2
- [ai](#) 1
- [beta](#) 1
- [blogofile](#) 1
- [game](#) 1
- [jasic](#) 1
- [javascript](#) 1
- [music](#) 1
- [oop](#) 1
- [optimization](#) 1
- [oscon](#) 1
- [politics](#) 1
- [scheme](#) 1
- [typescript](#) 1
- [visualization](#) 1

All 73 articles...

This blog is built using [jekyll](#).
The source repo for it is [here](#).

© 2008–2014 Robert Nystrom

Procedural generation—having the game build stuff randomly instead of using hand-authored content—is amazing when it works well. You get a ton of replayability because the game is different every time. As the person implementing the game, you also get the critical feature of not knowing what you’re going to get even though you wrote the code. The game can surprise *you* too.

People get into procedural generation because it seems easier. Hand-authoring content is obviously a lot of work. If you want your game to have a hundred levels, you have to make a hundred things. But make one little random level generator and you can have a hundred levels, a thousand, or a million, for free!

Alas, it doesn’t *quite* work out that way. You see, *defining the procedure* is a hell of a lot harder than just sitting down and banging out some content. You have to take some very nebulous, artistic chunk of your brain, figure out precisely what it’s doing, and translate that to code. You’re coding a simulation of yourself.

It must balance a number of technical and aesthetic constraints. For mine, I focused on:

- It needs to be **fairly efficient**. The generator only runs when the player enters a new level, so it doesn’t have to be as *super* fast, but I still don’t want a several second pause giving the player time to question whether they should be playing a game or doing something more productive with their life.
- The dungeon needs to be **connected**. Like the mazes on my old green-screen Apple, that means from any point in the dungeon, there is a way—possibly circuitous—to any other point.

This is vital because if player has to complete a quest like “find the magic chalice” or “kill the cockatrice”, it’s pretty cruel if the dungeon drops that in some walled-off room the player can’t get to. It also avoids wasting time generating and populating areas the player can never see.

- Moreso, I want dungeons to **not be perfect**. “Perfect” in the context of mazes and graphs (which are synonymous) means there is *only one* path between any two points. If you flatten out all of the windy passages, you’ll discover your twisty maze is really just a tree all crumpled up. Passageways branch but never merge. *Im*-perfect mazes have loops and cycles—multiple paths from A to B.

This is a gameplay constraint, not a technical one. You could make a roguelike with perfect dungeons, and many simple roguelikes do that because generators for those are easier to design and implement.

But I find them less fun to play. When you hit a dead end (which is often), you have to do a lot of backtracking to get to a new area to explore. You can’t circle around to avoid certain enemies, or sneak out a back passage. Neither can the bad guys, for that matter.

Fundamentally, games are about making decisions from a set of alternatives. At a literal level, perfect dungeons only give you one path to choose from.

- I want **open rooms**. I could make dungeons just be nothing but mazes of narrow passages, but then you could never get surrounded by a horde of monsters. It would feel claustrophobic and kill a bunch of interesting combat tactics.

Wide open areas are critical for area effect spells, and big dramatic battles. They also provide space for interesting decorations and themed areas. Vaults, pits, traps, treasure rooms, etc. Rooms are the high points of the hero’s journey.

- I want **passageways**. At the same time, I don’t want the dungeon to *just* be rooms. There are some games that create levels this way where doors directly join room to room. It works OK, but I find it a bit monotonous. I like the player feeling confined part of the time, and having narrow corridors that the player can draw monsters into is a key tactic in the game.

- All of this needs to be **tunable**. Many roguelikes have one huge multi-floor dungeon where depths vary in difficulty but not much else. My game is different. It has a number of different *areas*. Each has its own look and feel. Some may be small and cramped, others spacious and orderly.

I solve this partially by having multiple distinct dungeon generation algorithms. Outdoor areas use an entirely different process. (I should probably write about that too sometime. Look, another unfulfilled promise!) But coding a new dungeon generator from scratch for *every* area is a huge time sink. Instead, I want the generator to have a bunch of knobs and levers I can tweak so I can make a number of areas that share the same code but have their own feel.

A room with a view

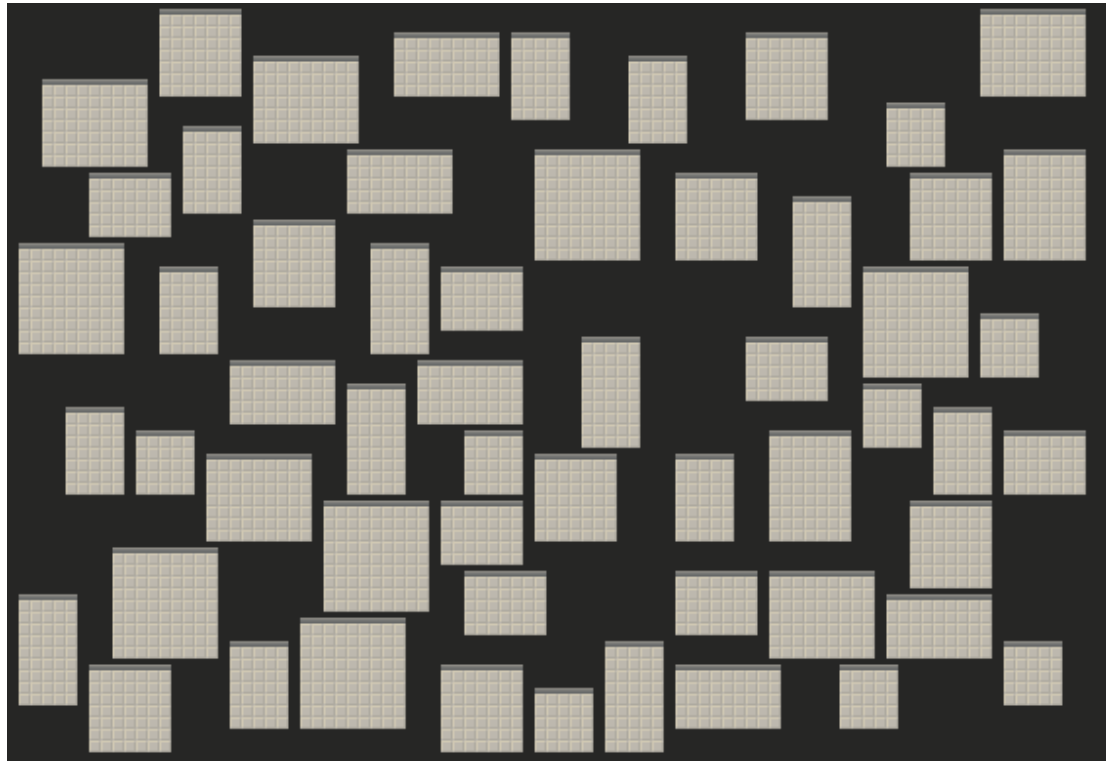
I've been working on this game pretty much forever (it's gone through four different implementation languages!) and I've tried a number of different dungeon generators. My main source of inspiration is a game called [Angband](#). The only thing I've sunk more of my life into than working on my game is playing that one.

Angband is fantastically old. When it forked off of Moria, Nancy Kerrigan had just taken a round of melee damage from a club-wielding troll. On machines of that time, it was much harder to make a fast dungeon generator, and Angband's is pretty simple:

1. Sprinkle a bunch of randomly located, non-overlapping rooms.
2. Draw random corridors to connect them.

To ensure rooms don't overlap, I just discard a room if it collides with any previously placed one. To avoid a possible infinite loop, instead of trying until a certain number of rooms are successfully *placed*, I do a fixed number of *attempts*

to place rooms. Failure becomes more common as the dungeon gets fuller—after all, you can only fit so many rooms in a given area—but tuning this gives you some control over room density, like so:



A dark and twisty passageway

Most of the dungeon generators I've written start with this. The hard part, by far, is making good passageways to connect them. That's really what this post is about—a neat way to solve that problem.

Angband's solution is brute force but surprisingly effective. It picks a pair of rooms—completely ignoring how far apart they are—and starts a passageway that wanders randomly from one (hopefully) to the other. It's got a few clever

checks to keep things from overlapping too much but passageways can and do cut through other rooms, cross other passages or dead end.

I tried implementing that a number of times but (likely failures on my part) never got to something I really liked. The corridors I ended up with always looked too straight, or overlapped other stuff in unattractive ways.

Then, a few months ago, I stumbled onto a [description of a dungeon generator](#) by [u/FastAsUcan](#) on the [/r/roguelikedev](#) subreddit. His generator, [Karcero](#), is based on [Jamis Buck's](#) dungeon generator. If you've ever done any procedural dungeon generation, you know—or should know—who Buck is. He's got a ton of great articles on random mazes.

Years ago, I remember seeing an actual [dungeon generator](#) he wrote for use with pen-and-paper Dungeons & Dragons. Unlike most of his maze stuff, this had actual rooms, and the results looked great.

But, at the time, I didn't know how it *worked*. How do you go from mazes to open winding corridors and rooms? I tucked this open question away in the corner of my mind and immediately forgot about it.

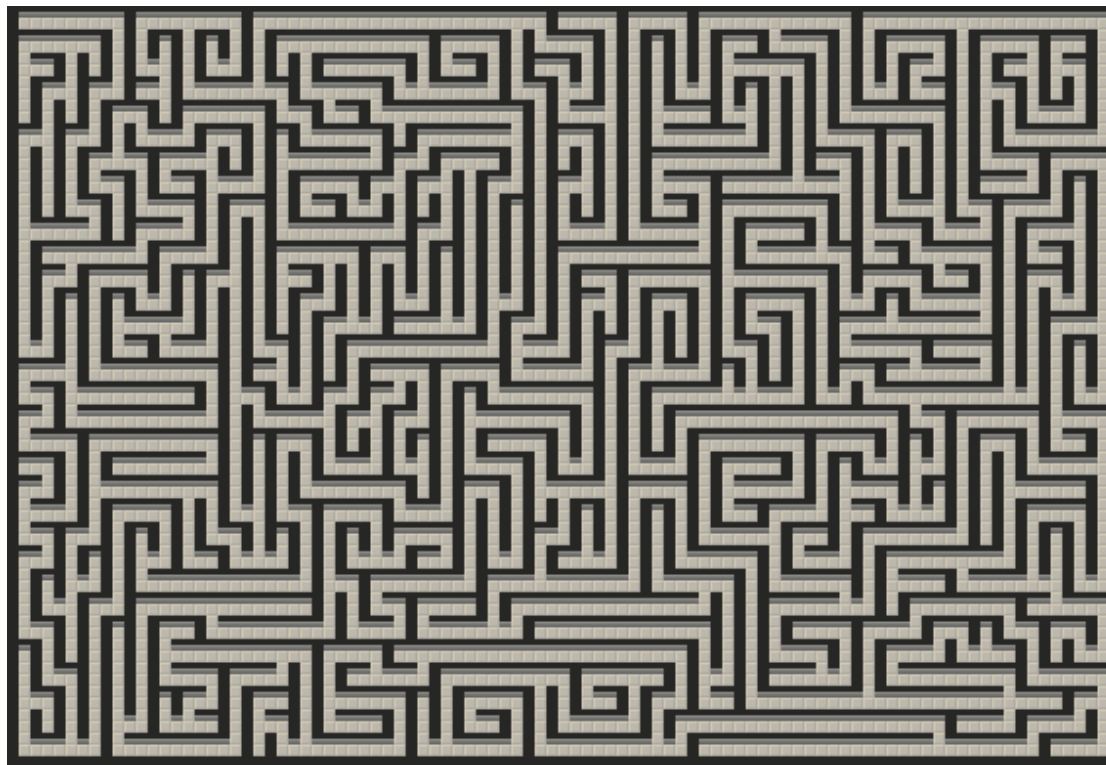
The post by FastAsUcan provides the answer. It works like so:

1. Make a perfect maze. There are a number of different algorithms for this, but they're all fairly straightforward.
2. Make the maze *sparse*. Find dead end passages and fill them back in with solid rock.
3. Pick some of the remaining dead ends and cut holes in them to adjacent walls. This makes the maze imperfect. (Remember, this is a good thing!)
4. Create rooms and find good locations to place them. "Good" here means not overlapping the maze but *near* it so you can add a door and connect it.

The magic step, and the piece I was missing, is *sparseness*. A normal maze fills every single square of the world, leaving no areas where you can fit a room. The trick that Jamis and FastAsUcan do here is to carve the whole maze and then *uncarve* the dead ends.

Doing that is actually pretty easy. A dead end is just a tile that has walls on three sides. When you find one of those, you fill that tile back in. That may in turn make the tile it connects to a dead end. Keep doing this until you run out of dead ends and you'll end up with lots of solid area where rooms can be placed.

Of course, if you do that starting with a perfect maze and run to completion, you'll erase the whole maze! A perfect maze has no loops so *everything* is a dead end if you follow passages long enough. Jamis' solution is to not erase *all* of the dead ends, just some. It stops after a while. Something like this:



Corridors to leave: 1000

Once you do that, you can start placing rooms. The process Jamis uses for this is interesting. He picks a room size and then tries to place it on every single location in the dungeon. Any location that overlaps a room or passageway is discarded. The remaining positions are “ranked” where rooms that are near passageways are better. It then picks the best position and places the room there, and puts some doors between the room and the passage.

Rinse, lather, repeat and you’ve got yourself a dungeon.

Rooms *then* mazes

I went ahead and coded this up exactly as described. It went OK, but I found that the process of placing rooms was pretty slow. It works well for dungeons of the small size you do for a tabletop role-playing game, but not so much at the scale of a computer roguelike.

So, I did some tinkering and came up with a slight variation. My contribution is pretty minor, but I thought it would be worth writing down. (Honestly, I just think it’s fun to watch animated dungeon generators, and the prose is pure fluff.)

Where Buck and Karcero start with the maze and then add the rooms, mine does things in the opposite order. First, it places a bunch of random rooms. Then, it iterates over every tile in the dungeon. When it finds a solid one where an open area *could* be, it starts running a maze generator at that point.

Maze generators work by incrementally carving passages while avoiding cutting into an already open area. That’s how you ensure the maze only has one solution. If you let it carve into existing passages, you’d get loops.

This is conveniently exactly what you need to let the maze grow and fill the odd shaped areas that surround the rooms. In other words, a maze generator is a randomized [flood fill](#) algorithm. Run this on every solid region between the

rooms and we're left with the entire dungeon packed full of disconnected rooms and mazes.



Each color here represents a different region of connected tiles.

Looking for a connection

All that remains is to stitch those back together into a single continuous dungeon. Fortunately, that's pretty easy to do. The room generator chooses odd sizes and positions for rooms so they are aligned with the mazes. Those in turn fill in all of the unused area, so we're assured that each unconnected region is only a single tile away from its neighbors.

After filling in the rooms and mazes, we find all of those possible *connectors*. These are tiles that are:

1. Solid rock.
2. Adjacent to two regions of different colors.

Here they are highlighted:



We use these to tie the regions together. Normally we think of the entire dungeon as a graph with each tile a vertex, but we're going to go up a level of abstraction. Now, we treat each *region* of tiles as a single vertex and each connector as an edge between them.

If we use *all* of the connectors, our dungeon would be way too densely connected. Instead, we carve through just the connectors we need to get each region connected to the whole *once*. In fancy terms, we're finding a *spanning tree*.

The process is pretty straightforward:

1. Pick a random room to be the main region.

2. Pick a random connector that touches the main region and open it up.

In the demo, it does that by placing a door, but you can do an open passageway, locked door, or magical wardrobe. Be creative.

Note that this process is agnostic about rooms and mazes. It just deals in “regions”. That means it can connect rooms directly to other rooms sometimes. You can avoid that if you want, but I find the resulting dungeons more fun to play.

3. The connected region is now part of the main one. Unify it. In the demo, I use a little flood fill algorithm to color in the newly merged region because it looks pretty. In a real implementation, you don’t need to mess with tiles. Just make a little data structure that tracks “region X is now merged”.

4. Remove any extraneous connectors. There are likely other existing connectors that connect the two regions that just merged. Since they no longer connect two separate regions and we want a spanning tree, discard them.

5. If there are still connectors left, go to #2. Any remaining connectors imply that there is still at least one disconnected region. Keep looping until all of the unconnected regions are merged into the main one.

Earlier, I said that I don’t want a perfect dungeon because they make for crappy gameplay. But, since this creates a spanning tree, that’s exactly what we’ve got. We only allow a single connector between any two regions so our dungeon *is* a tree and there’s only a single path between any two points.

Fixing that is pretty simple. In step 3, when we cull the unneeded connectors, we give them a *slight* chance of being opened up. Something like:

```
if (rng.oneIn(50)) _carve(pos, CELL_DOOR);
```

This occasionally carves an extra opening between regions. That gives us the imperfect loops we want to make the dungeon more fun to play in. Note that this is also easily tunable. If we make the chance more likely, we get more densely connected dungeons.

Uncarving

If we stop here, we'll get dungeons that are packed chock full of maze corridors, most of which are dead ends. That has a certain sadistic appeal, but isn't exactly what I'm going for. The last remaining step is the "sparseness" pass described earlier.

Now that we've got all of our rooms connected to each other, we can remove all of the dead ends in the maze. When we do that, the mazes are reduced to just the winding set of passageways needed to connect the rooms to each other. Every corridor is guaranteed to go somewhere interesting.

What we ended up with

In summary:

1. Place a bunch of random non-overlapping rooms.
2. Fill in the remaining solid regions with mazes.
3. Connect each of the mazes and rooms to their neighbors, with a chance to add some extra connections.
4. Remove all of the dead ends.

I'm pretty happy with it so far. It's not perfect, though. It tends to produce annoyingly windy passages between rooms. You can tune that by tweaking your

maze generation algorithm, but making the passageways less windy tends to make them wander to the edge of the dungeon, which has its own strange look.

The fact that rooms and mazes are aligned to odd boundaries makes things simpler and helps fill it in nicely, but it does give the dungeon a bit of an artificially aligned look. But, overall, it's an improvement over what I had before, and the dungeons it makes seem to be pretty fun to play.

If you want to see for yourself, you can play the game [right in your browser](#). The code for these demos is [here](#), but it's pretty gnarly. Making them animated adds a lot of complexity. Instead, [here](#) is the much cleaner implementation my game uses.

As a bonus for making it this far, here's a super dense giant dungeon. I find it hypnotic:



80 Comments **journal**

 **Login** ▾

 **Recommend** 17

 **Share**

Sort by Best ▾



Join the discussion...

LOG IN WITH


OR SIGN UP WITH DISQUS 

Name



Marcel Popescu • 3 years ago

Amazing article... makes me want to start implementing a game just so that I can use it. Unfortunately, I have about 50 other things I want to write :)

33   • Reply • Share ›



Lyndon • 3 years ago

I played around with mazes for a while (without looking into sparseness) and one fun thing I found was that playing around with your entropic source of randomness could lead to very different maze "characteristics" for each maze type.

The simplest way I did this was to have a lazy sequence of numbers consumed by my algorithm, by default using a standard randomisation algorithm, but also allowing user-supplied sequences that were cycled.

Here's a couple of examples:



16   • Reply • Share ›



munificent Mod ↗ Lyndon • 3 years ago

Yes! There's a lot of simple things you can do to tweak the maze generator. Mine using something called the "growing tree" algorithm. It basically maintains a list of "live" cells, which are corridors that may have neighboring cells that are solid rock that can be carved into. Every iteration, it chooses a live cell, removes it from the list and processes it. If it has no solid neighbors, it's just discarded. Otherwise, it carves a new live cell into a randomly chose neighboring direction, then gets re-added to the list.

You can dramatically change the output just by tweaking how you pull cells from the list. Assuming you add them to the end, if you remove from the beginning, you basically get a breadth-first-search with short, highly branching passages, like a coral fan. Choose from the end and you get DFS, with long meandering paths that rarely branch. Weight towards the middle and you can blend between those two extremes.

5 ^ | v • Reply • Share ›



Maximov Anatoly • 3 years ago

Last example can be a very good screensaver=)

10 ^ | v • Reply • Share ›



Matt Clegg • a year ago

I'm not sure if the author will see this but I'm attempting to port this to Java and having some issues with the connectRegions method.

Here's my current method: <http://pastebin.com/W9WTEkQT>

I've been struggling with this line (and similar lines) from your dart source:
`var regions = connectorRegions[connector].map((region) => merged[region]);`

In my code it keeps pointing to an array index out of bounds exception.

3 ^ | v • Reply • Share ›



Jeremy Stanley • 3 years ago

I went ahead and reimplemented this (very non-performantly for now) using React and plain Javascript and ended up feeling the same way about the

windyness of the maze as you had expressed. I added an extra step towards the end which is a bit costly but reduces slack in the maze. See <http://fishi.es/dungeon> or <http://fishi.es/dungeon/demo> to see it in being drawn. Here's the source for the slack-reduction algorithm: <https://gist.github.com/azu...>

3 ^ | v • Reply • Share ›



Tanner Van De Walle → Jeremy Stanley • a year ago

Woah! Slack removal looks great!

^ | v • Reply • Share ›



stan10785 → Jeremy Stanley • 2 years ago

Like it!

^ | v • Reply • Share ›



munificent Mod → Jeremy Stanley • 3 years ago

Oh, wow. That looks really really cool!

^ | v • Reply • Share ›



Shi ChuanGuan • 3 years ago

This is amazing. I wrote a similar maze / dungeon engine back in the 80s in BASIC. My code back then lacked the random rooms yours has. Kudos!

Thanks for making me smile. Made my day seeing this, brought me back to my teens. ^_^

3 ^ | v • Reply • Share ›



Suzanne Atkinson → Shi ChuanGuan • 4 months ago

FUnny but I also had a random maze generator back in the 80s that I used to enter a competition and got 2nd place.

17 ^ | v • Reply • Share ›



Peeling • 3 years ago

As regards the disagreeable windiness between rooms: looking at the output you could get rid of most of it thus:

Trace each linear corridor section (terminated by branches or rooms)

Once you have the start and end of a section, retrace your steps. If you find a

point where you could dig through one block to make a shortcut to an earlier part of the section, do so, and fill in the unwanted part. Continue until you reach the start of the section.

Repeat for all linear corridor sections.

3 ^ | v • Reply • Share ›



munificent Mod ➔ Peeling • 3 years ago

This is a really cool idea. Thanks!

1 ^ | v • Reply • Share ›



Marc • 3 years ago

Thanks for this brilliant article and hypnotizing mazes! Great side project for the xmas holidays :-)

3 ^ | v • Reply • Share ›



Marrt • 2 years ago

Thank you for sharing this, i converted it to C# for use in Unity

<http://forum.unity3d.com/th...>

-i added a function to streamline corridors between branch points and doors and a function that sorts the list of connectors to give room-to-room connectors a better chance to be connected first (= less dead-end-rooms)

2 ^ | v • Reply • Share ›



Anon • 3 years ago

This is pretty cool, but I'm not too fond of the end result, from a gameplay point of view. This seems **WAY** too dense, rooms and corridors everywhere. There's no way I can get familiar with the map as a player, there's just too much of it, none of it is special.

Your mazes have very little in the way of blank, empty space. And the pathways between two rooms can be chaotic twisting things that look like they'd get really boring to go through.

Who carves tunnels that go in a completely wrong direction and make the player walk half the map when you really just needed to go two units forward?

The actual maze levels in nethack are widely considered to be the most tedious and boring. for example.

Neat project, but I wouldn't play it like this.

2 ^ | v • Reply • Share ›



Nobody On An Oddessy ➔ Anon • 3 years ago

I fixed that with a simple tweak. After the dead-end pass, I went through and checked every wall tile's (linear) neighbors. If it had 2 or more hallways as neighbors and all of its neighbors were of the same region, it became a hallway of the same region.

The result is still space filling, but the paths look and feel natural. You also get large spaces in the halls that, with a little work, you could easily turn into more rooms.

1 ^ | v • Reply • Share ›



nathar • 3 years ago

Great article!! Just for the discussion, why dart??

2 ^ | v • Reply • Share ›



munificent Mod ➔ nathar • 3 years ago

Two reasons:

1. I'm a member of the Dart team at work, so it's the language I use the most these days.

2. I think it's a really great language for stuff like this. I think people coming from almost any language won't have too much trouble understanding the code, it's a lot cleaner than JS, and yet it still runs in a browser.

5 ^ | v • Reply • Share ›



James Mussett • a year ago

An absolutely brilliant article! I've used a lot of the techniques mentioned above in my own code. The following adds additional steps like stripping out unnecessary walls and edges to create a much more open-plan dungeon.



1 ^ | v • Reply • Share ›



munificent Mod ➔ James Mussett • a year ago

Oh, wow. That looks really nice!

^ | v • Reply • Share ›



Jordan W • 2 years ago

Thank you for the great explanation and exploration of this topic! I loved the animations, and I agree that the maze generation is hypnotic.

I've become more familiar with Dart by reading through your dungeon generator code, but I am having trouble (I think), understanding this line:

```
var regions = connectorRegions[connector]
```

```
.map((region) => merged[region]);
```

Could you explain it to me? I thought it returned a simple iterable list that contains the set of regions that connector is attached to, but my code isn't working correctly (I'm trying to port it to C#), and I think this might be the problem. Thanks again!

1 ^ | v • Reply • Share ›



munificent Mod ➔ Jordan W • 2 years ago

Conveniently, someone else asked about this a while back. Here's the explanation:

connectorRegions is a Map in Dart, which is like Dictionary in C#. It associates keys with values. Here, the key is a Vec, a position in the dungeon, and the value is the set of regions the tile at that position is next to. Each region is identified by a number, so the value type is Set<int>, which is like HashSet<int> in C#. This map tells you which tiles are next to more than one isolated region in the dungeon. These tiles will be used to carve passages that connect the option regions together.

connector is a Vec, one key randomly chosen from connectorRegions.

It's the connector that we're carving open to connect its neighboring regions together.

Now we have:

```
connectorRegions[connector]
```

[see more](#)

2 ^ | v • Reply • Share ›



Brandon Beauchene → munificent • a year ago

im confused with the same line but porting to java. hashmap doesnt have a map method and whats the point of using the anonymous method? how would i implement the usage of map() you show in java to get the proper regions object?

^ | v • Reply • Share ›



munificent Mod → Brandon Beauchene • a year ago

connectorRegions is a map of Vecs to sets of tiles. connectorRegions[connector] looks up a Vec in that map and returns the Set<int> associated with it. It's the Set that has .map(). That iterates over every number in the set and passes it to the lambda. Then it returns a new iterable with the result of that. In imperative Java, it would be something like:

```
List<int> regions = new ArrayList<>();
for (int region : connectorRegions.get(connector)) {
    regions.add(merged.get(region));
}
```

^ | v • Reply • Share ›



stan10785 → Jordan W • 2 years ago

I was just getting started with the XNA RPG tutorial in Monogame and saw Roguesharp but was not happy with the map. This article was fun if a bit beyond my current thinking. It would be fun to try in C#

^ | v • Reply • Share ›



stan10785 → Jordan W • 2 years ago

Did you happen to get this to work in C#? I was looking at RogueSharp and was disappointed in the layout and was looking for an alternative. Reading the article and seeing the animations - I was impressed. Just wondered if there was a link to code somewhere

^ | v • Reply • Share ›



Jordan W → stan10785 • 2 years ago

I did indeed get it working in C#! I am now figuring out the slack reduction algorithm, written in javascript, posted below by another poster. I'm self taught and doing this in my (fleeting) spare time, so it took me a week or so to port the code from dart to C# (mostly because I was unfamiliar with the more advanced parts of C#). Feel free to ask me for help with porting it if you get stuck if you want to try and port it too.

^ | v • Reply • Share ›



stan10785 → Jordan W • 2 years ago

Jordan

I may do that. I was looking at the shorter code submitted in *.js by someone in the discussion - I am fighting for spare time too and not a power C# programmer!! Learning opportunities

^ | v • Reply • Share ›



Hasalm • 3 years ago

Excelent procedure, and even better explanation thereof.

Other than the slack in the corridors, the only problem i have with this design is that its possible to pass from one end of the dungeon to the other without entering a single room, making it look like they were just tacked onto a maze rather than them being part of the dungeon. Rooms that are adjacent to two sections of the same(or nearly the same) branch could(at a probability) ensure that they have connections to both and then remove the redundant section of hallway. If performed before the dead end removal step, each would only have to remove a single non-branching tile.

1 ^ | v • Reply • Share ›



Rune Skovbo Johansen • 3 years ago

Nice explanation! Incidentally, it's the same approach I use in my game The Cluster. That's a platformer and it has much fewer rooms but also does first rooms, then full maze, then cull some of the maze dead ends down to fit a certain fill ratio.

I use a special maze algorithm I designed myself which optionally makes the maze sparse as its being built. When enabled, the carving of passages prefer tiles that are not near existing passages. The preference can also be influenced by a probability "heat map". It took some work to make it performant - I guess I should write a blog post about it too at some point.

1 ^ | v • Reply • Share ›



munificent Mod ➔ Rune Skovbo Johansen • 3 years ago

Yes, I would definitely read it!

^ | v • Reply • Share ›



Jai Shaw • 3 years ago

Nice, I had a thought to expand on this adding some points with some weight in your random room placement, on a big map this would give you one or more clusters of rooms with some flung out near the edges. Just a thought :)

1 ^ | v • Reply • Share ›



Meteor Fury • 3 years ago

This is unbelievable. I think I have clicked the last maze generator you have posted here like 100 times. Sat back, relaxed, and began to drool.

1 ^ | v • Reply • Share ›



Ted Mielczarek • 3 years ago

I enjoyed this other article that I read recently that takes a different tack, with some pre-designed room bits being stitched together by an algorithm:

<http://www.magicaltimebean....>

I wonder how well some combination of these two approaches would work? Thanks for the awesome visualizations and for publishing your source code!

1 ^ | v • Reply • Share ›



Paul Taylor • 3 years ago



Daniel Ioussignant • 3 years ago

What a great way to explain this, using canvas.

1 ^ | v • Reply • Share ›



damien • 3 years ago

Hmm, I might have approached this from a constructionist perspective. How might such a dungeon have been built? Maybe it started as a mine, then grew into a crypt, then into a proper dungeon.

Starting as a mine would give you longish passageways and a fair number of dead end side-tunnels.

Expanding into a burial crypt, those side-tunnels would be expanded.

Then as it morphs into an actual dungeon, the side tunnels become rooms.

1 ^ | v • Reply • Share ›



FrankenPC . • 3 years ago

Brilliant. I wonder if it's possible to sort of force a little order into the chaos. Like use some pattern templates so there appears to be some thought going into the design of the dungeon. Like a human actually designed it and not random seeds.

1 ^ | v • Reply • Share ›



J Kenneth King ➔ FrankenPC . • 3 years ago

Constraint propagation solvers! Or simulation (ie: simulate the workers, forces like erosion, etc and let it run for a few thousand cycles).

Look at the source for Brogue too. It can design levels with magic puzzles, locked vaults, mechanical traps (not just a "take some damage tile"), and unique situations like captured NPCs which you can free (or not of course).

1 ^ | v • Reply • Share ›



FrankenPC . ➔ J Kenneth King • 3 years ago

I'll check Brogue out. Thanks for the tip!

^ | v • Reply • Share ›



olsn • 3 years ago

VERY cool!

1 ^ | v • Reply • Share ›



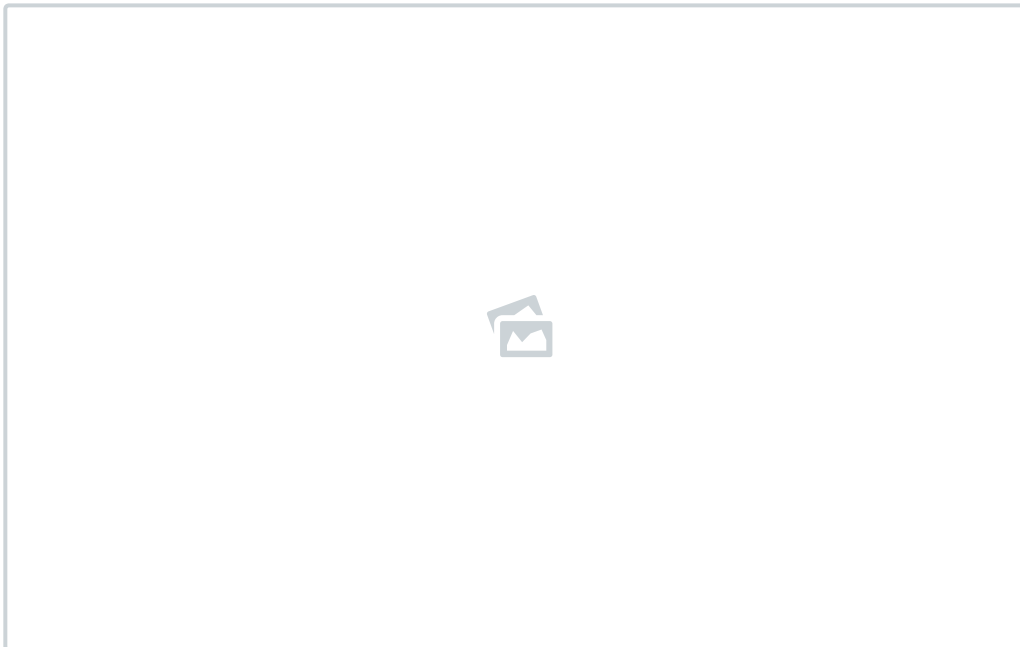
IvaintheMCbuilder • a month ago

Been trying to achieve something like this for a while by myself, I guess this is far more useful.

^ | v • Reply • Share ›



Max Paine • 8 months ago



Your article is pretty good, however i don't like this boring winding passages between rooms. My idea is to put rooms just like you do, then connect

[see more](#)

^ | v • Reply • Share ›



Tanner Van De Walle • a year ago

Just realized how old this post is, but I have been doing a little reading into dungeon generation for a game idea, and this was extremely helpful! Thank you!

^ | v • Reply • Share ›



Xepherys • a year ago

I've been trying to convert this to C# for use in Unity, but I'm having some difficulty with the growMaze() portion. Any chance you could look at what I have so far and make some recommendations?

<http://www.labyrintheer.com...>

Thanks!

^ | v • Reply • Share ›



Pr0l4p5 • a year ago

Wonderful article ! This is exactly what I was looking for ! started to implement it in JavaScript, will post another comment with my progress ! I might add some other room shapes but I'll have to get a polygon.distanceTo method I guess ^^ Thanks a lot for this ! Cheers from France (2 years late) !

^ | v • Reply • Share ›



grumbunger • a year ago

hmm awesome, my post did not show

^ | v • Reply • Share ›



grumbunger • a year ago

this is a great inspiration the room layouting is exactly what I am up to, but the halls I didn't know what I might do. I was thinking of generating rooms off of opening the doors. but to layout the whole dungeon would be optimum.

^ | v • Reply • Share ›



Mr.Zhao • a year ago

Wonderful!

Recently I am make one game,i'm troubled that how to random create one map,it give me some idea ,but i also have several problem.

for example ,how to create one maze road,the text describe it only one word!

wish for best information about how to create one maze road!

by the way ,is anyidea that create one maze game in unity3D!

^ | v • Reply • Share ›



davidbackeus • 2 years ago



Any suggestions for articles / literature on procedural generation of vast open worlds such as those found in Cataclysm or Dwarf Fortress?

^ | v • Reply • Share ›

Load more comments



Subscribe



Add Disqus to your site

Add DisqusAdd



Privacy