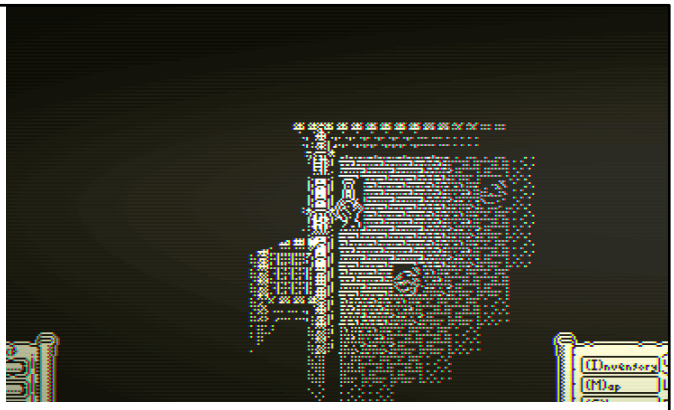
Procedural Content Generation for Computer Games

Lecture 5 – Mazes, Dungeons and RPGs

Vojtěch Černý
cerny@gamedev.cuni.cz

Dungeons!

- One of the early examples of PCG are dungeons:
  - present in a lot of RPGs
  - Rogue, Diablo, …
  - Dungeons and Dragons

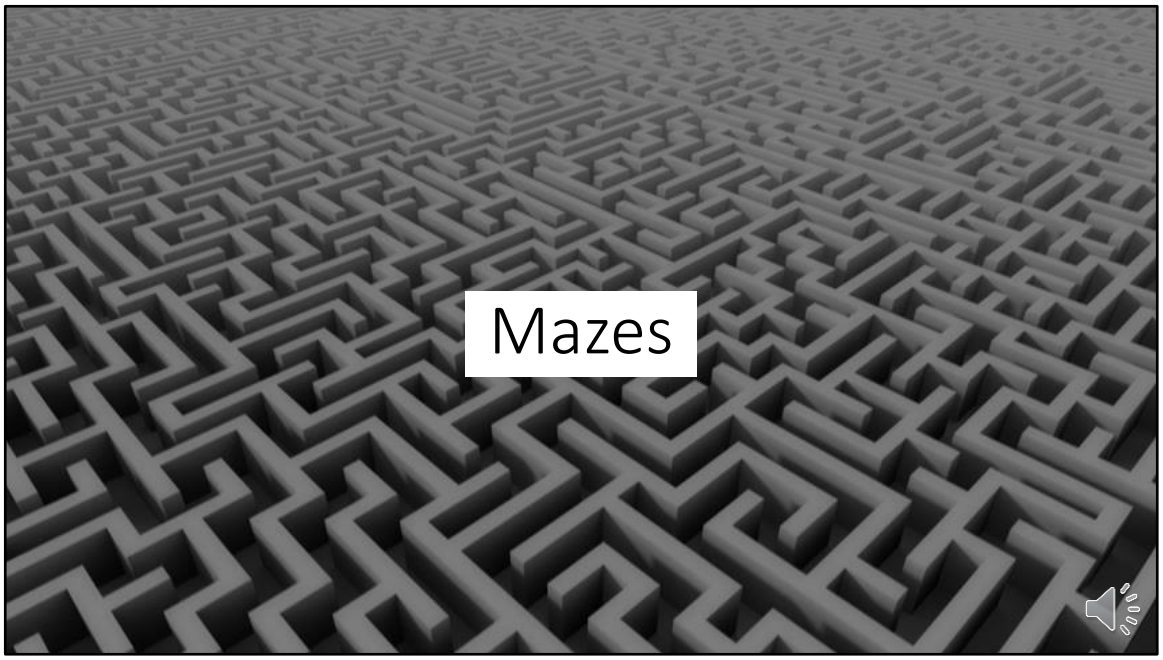


- There's a lot of reasons to have them procedural
  - Replayability
  - Unexpected experiences
  - Random looting
  - …

Image from http://wolfgame.com/games/silverquestgaiden.html

Agenda

- Maze generation
- Dungeon generation
  - Moving from mazes to dungeons
- Common RPG
  - Procedural items
  - Procedural encounters
  - Dungeon puzzles

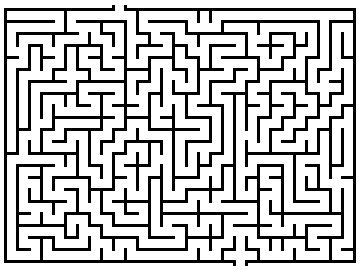
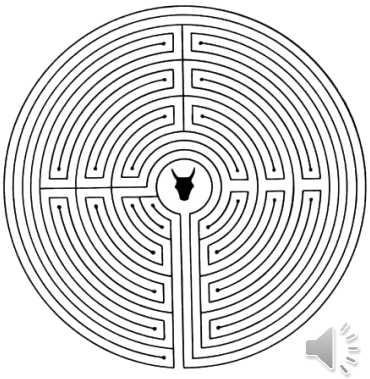

Mazes

Mazes

- In some ways, a simple version of dungeons
- There, however may be a different objective:
  - You may not want the player get “completely lost” in a dungeon
- We will consider a maze a simpler dungeon
- Definition: Maze is a graph, with one vertex selected as beginning and one vertex selected as goal.

Different kinds of mazes

- You may want the nodes to form a nicely viewable shape:
  - **dimension**: 2D, 3D, 4D, …
  - **topology**: rectangle, sphere, toroid, teleports…
  - **tessellation**: orthogonal, hex, triangular, …
  - **routing**: perfect, braid, …
  - **attributes**: run, river, elite, …
  - …

A beautiful list of various features of mazes can be found here:
http://www.astrolog.org/labyrnth/glossary.htm
Orthogonal tessellation is the most common maze form, and we'll only deal with that one in this lecture. Most of the algorithms can work in other topologies (on any graph).

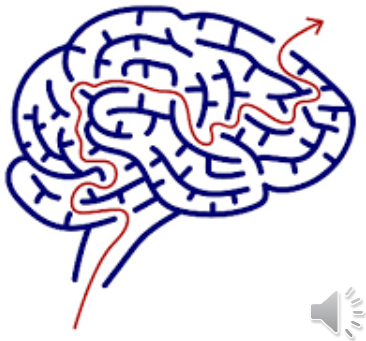
Perfect maze – only one path between any two points, all cells accessible.
Braid maze – no dead ends (always loops).
"Run" factor – long straight stretches
"River" factor – less, but long dead ends
"Elite" factor – percentage of the (shortest) solution path to exit. There is a type of maze with Elite = 1.0 – unicursal mazes. Just one snaking path throughout. The classical Minotaur's Labyrinth is often depicted as a unicursal maze.

Fun fact: there is a nice nod towards the Minotaur maze in Inception. Elliot Page is even named Ariadne in the movie.

Objective of a good maze

- Traditionally, to provide a challenge navigating from start to end

- But, could also be:
  - aesthetically pleasing
  - provide good exploration
  - …



How to solve a maze

- You can apply basic CS knowledge:
  - Breadth-first, depth-first, heuristic-guided search, …
- But as a designer, you should worry about how humans can solve mazes
  - Depth-first is still applicable, but may be tricky with loops
  - Wall-following (fails with loops)
  - Filling dead-ends (fails with loops)
  - Heuristics, e.g. distance to goal
  - Also, it is different if you have an overview of the maze or not

https://arxiv.org/abs/1307.5713 (Understanding Humans' Strategies in Maze Solving, Zhao and Marques, 2013)
It mentions that humans make a combination of exploration and guidance (heuristics). Which is basically search, but the interesting take is, that we alternate between the two processes.

It may seem that loops make mazes harder. And they often do – a lot of approaches fail on them!
It is just slightly counter-intuitive, since
a) they usually produce multiple paths to goal.
b) if you add a shortcut, you create a cycle. But shortcuts should make the game simpler

How to generate a maze

- Despite mazes being quite a simple object, there is a plethora of approaches for generation
- Not-only do mazes have several types, there's also additional metrics you may want to pay attention to:
  - Number of paths to goal (perfect mazes have just 1)
  - Length from start to goal
  - Runs (whether there are long straight paths in the maze)
  - Patterns, such as spirals
  - …

There is probably nothing more thorough on mazes than this https://www.astrolog.org/labyrnth.htm

Check out the Maze Algorithms page, it contains a lot of information. Also, feel free to get Daedalus (still works on Win10) and experiment with maze creation.

Maze generation categorization

- By representation:
  - wall-adding
  - passage-carving
  - bitmap
- By the kind of maze
  - We'll mostly talk about perfect maze generation
  - They can be post-processed into imperfect mazes

We'll only encounter a few examples, if you want to research this more - http://www.astrolog.org/labyrnth/algrithm.htm

How to generate a perfect maze

- Perfect = no duplicate paths, no inaccessible areas
- Equivalently - the maze graph is a tree

- Simple solution – consider the "full" graph having random weights on edges and run a minimum spanning tree algorithm
- Kruskal's (greedy) algorithm is useful, you can simple take a random edge with each step of the algorithm

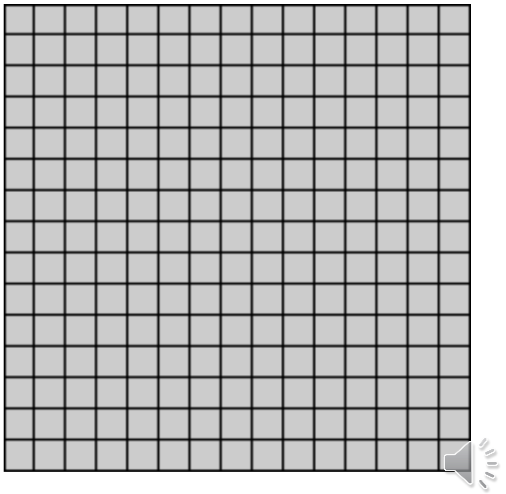
Another popular minimum spanning tree algorithm for mazes is Prim's.

The grids on these slides are animated – maybe switch to presentation mode to see it.
They are taken from James Buck's website, it has even more - http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

How to generate a perfect maze

- Perfect = no duplicate paths, no inaccessible areas
- Equivalently - the maze graph is a tree

- Simple solution – consider the “full” graph having random weights on edges and run a minimum spanning tree algorithm
- Kruskal’s (greedy) algorithm is useful, you can simple take a random edge with each step of the algorithm

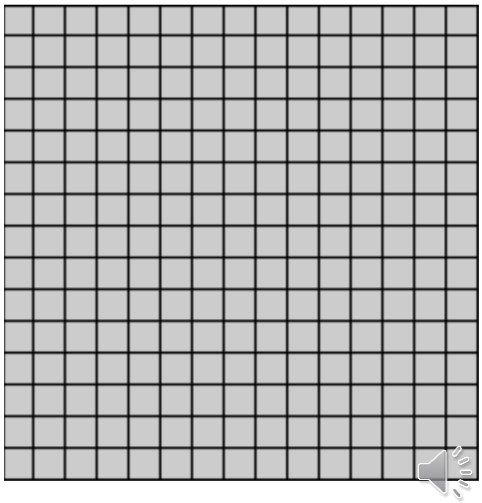


Another popular minimum spanning tree algorithm for mazes is Prim’s.

The grids on these slides are animated – maybe switch to presentation mode to see it.
They are taken from James Buck’s website, it has even more - http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

How to generate a perfect maze 2

- Just a simple idea – just run backtracking with random node selection
- Mazes will have large “river” factor: less, but quite long, dead-ends



You can place the exit at the furthest away node during the backtracking.

How to generate a random perfect maze

Wilson's algorithm

1. Carve a (random) point in the maze
2. Start a random walk from any point
   - Moving backwards = retracting
   - If the random walk touches the carved area, carve the whole walk
3. Unless you have a perfect maze, goto 2

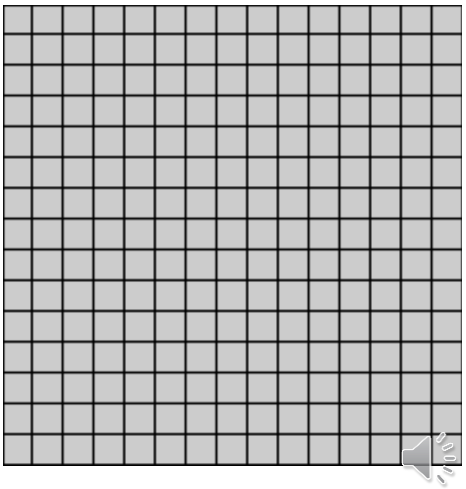
This algorithm provides a uniformly selected example of all perfect mazes.

It takes a long while to complete, but will terminate, even in a large example. Symmetric random walks in 2D always terminate https://math.stackexchange.com/questions/536/proving-that-1-and-2-d-simple-symmetric-random-walks-return-to-the-origin-with.

How to generate a random perfect maze

Wilson's algorithm

1. Carve a (random) point in the maze
2. Start a random walk from any point
   - Moving backwards = retracting
   - If the random walk touches the carved area, carve the whole walk
3. Unless you have a perfect maze, goto 2



This algorithm provides a uniformly selected example of all perfect mazes.

It takes a long while to complete, but will terminate, even in a large example. Symmetric random walks in 2D always terminate https://math.stackexchange.com/questions/536/proving-that-1-and-2-d-simple-symmetric-random-walks-return-to-the-origin-with.

Do you even want a perfect maze?

Pros

- Whole space is used
- Should be difficult to solve

Cons

- Wall-following will solve your maze
- Loops can make mazes harder
- No shortcuts
- Just a subspace of all mazes

Braid mazes – mazes with no dead ends. Everything ends in a cycle.

Mazes for dungeons

- You usually don’t want the player to get completely lost
  - Except if you’re cruel
- You don’t want to bore the player too much by long dead-ends
  - Even if there was some reward
- Cycles can help you a lot, but they may bring a lot of complications

Have you ever encountered Lost Woods in Zelda games?

Cycles in mazes

- Cycles can be difficult
- Adding them in the right way can reduce or increase difficulty

- Simple option – remove some walls
  - randomly
  - in dead ends

- Braid maze generation algorithm
  - add random walls to the maze that don't create a dead-end

There are also plenty other maze generation methods with interesting properties, we really mentioned just a few.

E.g. Sidewinder – generates the maze from top to bottom. Requires little memory and can generate infinite mazes.

Dungeons

Dungeons

- Made famous by Dungeons and Dragons
- Key feature in many RPGs

You can build your dungeon like a maze, but maybe you don't want to

- Though some kinds of mazes may not be as confusing
- Dungeons are usually formed with rooms and (optionally) corridors

In old RPGs and roguelikes, a dungeon was an underground structure for the player to explore. Modernly, a dungeon can be any enclosed space, such as a building (towers, castles, …).

Definition from PCG book: labyrinthic environments, consisting mostly of interrelated challenges, rewards and puzzles, tightly paced in time and space to offer highly structured gameplay progressions

Already known approaches

For dungeons with rooms, few methods were already mentioned:

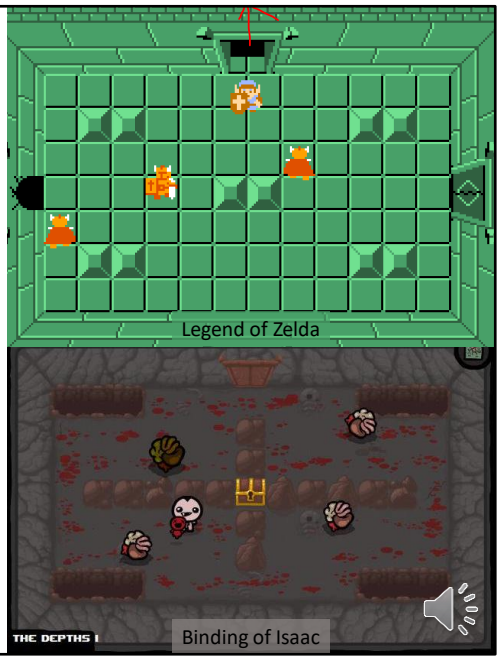
- Searching (Evolutionary Algorithms) with various encodings
- Shape / graph grammars, cellular automata
- Answer Set Programming
- Wave Function Collapse (with postprocessing)

Caves of Qud use Wave Function Collapse for dungeon/overworld generation.

Cellular automata are usually used only if you want organic environment – most often natural caves.
But, there also exist cellular automata that generate decent mazes
https://rejbrand.se/rejbrand/article.asp?ItemIndex=421,
https://english.rejbrand.se/rejbrand/article.asp?ItemIndex=422

Maze-like dungeons

- Zelda-like dungeons are formed from rectangular rooms, which could be assembled similarly to orthogonal mazes
- Obviously, you need another pass (or templating) to add items, encounters, …



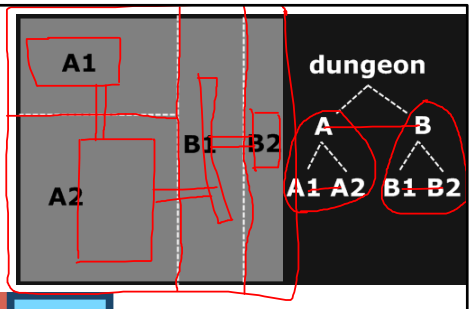
Yes, Zelda games don't generate their dungeons (at runtime, at least). But there's a Zelda-lika called Lenna's Inception, which does! It even generates the overworld map.

Apart from that, Zelda games have a lot of nice theory about Dungeon generation. The critical path (must-go-through) is quite linear – despite feeling non-linear, dead-ends are short or contain a shortcut back, etc. I recommend reading https://www.gamasutra.com/view/feature/6582/learning_from_the_masters_level_.php.

A very comprehensive playlist on Zelda dungeons by Mark Brown - https://www.youtube.com/playlist?list=PLc38fcMFcV_ul4D6OChdWhsNsYY3NA5B2

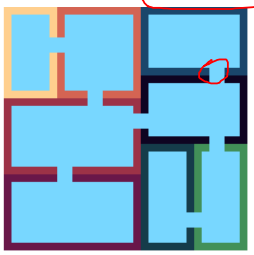
Simple methods for dungeons

Binary division



Two basic options

- Splitting preferences:
  - Split by longer axis
  - Split by shorter axis
  - Split by random axis
- Room placement
  - Take whole cell as room
  - Place room within the cell



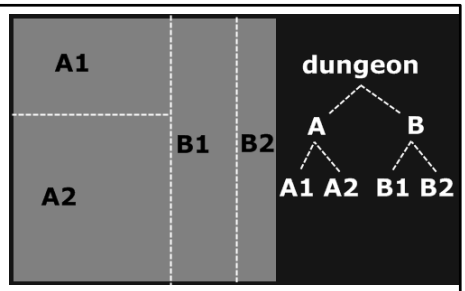
- Additional uses for the implicit tree
  - Corridors, “theme”

The good thing about this algorithm, is that it can naturally generate dungeons without corridors (if you just take the whole cell as a room), which is often tricky to accomplish.
They will also be neatly packed into a rectangle, which produces building-like results.

https://eskerda.com/bsp-dungeon-generation/
https://thingonitsown.blogspot.com/2018/11/dungeon-generator.html

Simple methods for dungeons

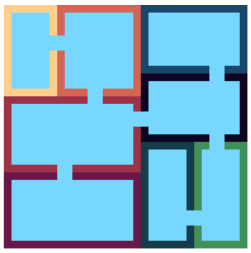
Binary division

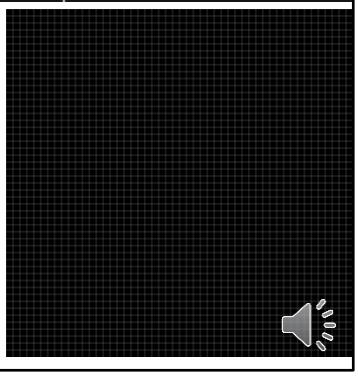


Two basic options

- Take whole cell as room
- Place room within the cell

- Splitting preferences:
  - Split by longer axis
  - Split by shorter axis
  - Split by random axis





- Additional uses for the implicit tree
  - Corridors, “theme”

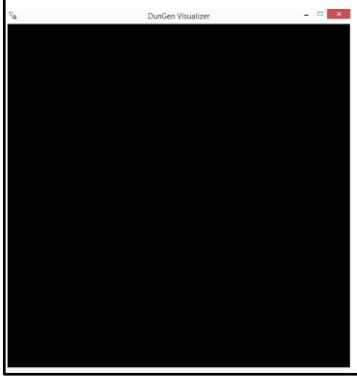
The good thing about this algorithm, is that it can naturally generate dungeons without corridors (if you just take the whole cell as a room), which is often tricky to accomplish.
They will also be neatly packed into a rectangle, which produces building-like results.

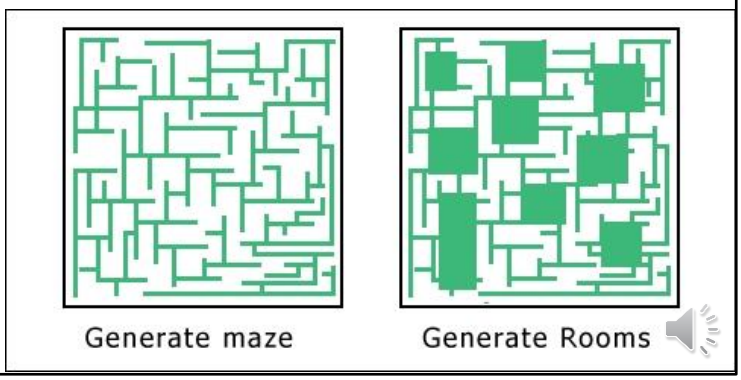
https://eskerda.com/bsp-dungeon-generation/
https://thingonitsown.blogspot.com/2018/11/dungeon-generator.html

Simple methods for dungeons

Start with maze

- Add rooms in free space of a (sparser) maze
- Overlap rooms over a maze





https://www.reddit.com/r/roguelikedev/comments/2brhl8/screenshot_saturday_08/cj87umz/

Simple methods for dungeons

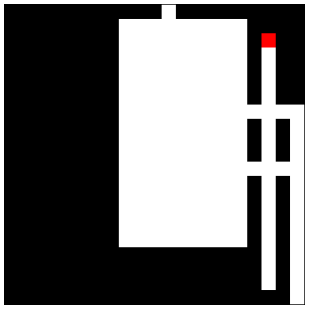
Ending with mazes

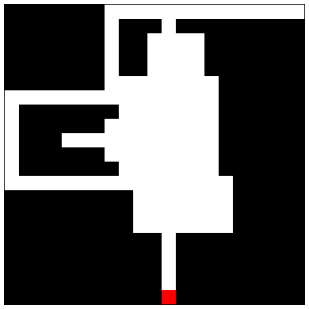
- Randomly place the rooms (or by subdivision)
- Run a maze algorithm to fill the space in-between
  - Connect to rooms immediately
  - Connect to rooms in another pass
    - e.g. to control number of connections

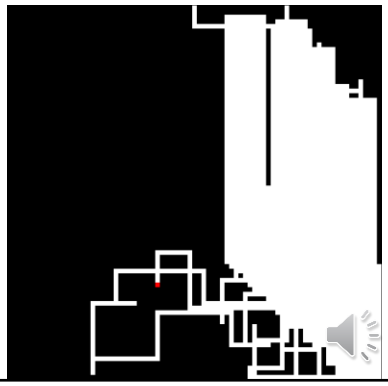
Check out Bob Nystrom's creation https://journal.stuffwithstuff.com/2014/12/21/rooms-and-mazes/ - it contains an interesting idea where he completely cuts-off all dead-ends

Simple agent-based method

- An agent (digger) starts digging in the dungeon
- With a (small) probability, he places a room, or makes a turn
  - without any “smartness”, the results are quite poor







The algorithm creates many overlapping rooms, and dead-end corridors.

Also, results get much worse as the space gets larger. Random-walks in 2D often bias themselves into one part of the space (in short periods).

Look-ahead agent-based method

- Look-ahead and place corridors and rooms only if none are there yet
- Add back-tracking to avoid dead-ends
- Alternative of the backtracking method for mazes

Adding control

- The simpler approaches don't give much control to the creator
- Often you want to influence the generator:
  - Add a hand-designed room
  - Add mandatory checkpoints
  - Add solvable lock-and-key puzzles
  - Constraints (length of shortest path, placement of boss key, …)
  - …

Hand-designed rooms in dungeons

- Why?
  - Some things are hard to generate procedurally
  - e.g. boss-fights, realistic building layouts
- In some approaches, this is quite simple:
  - maze-first, maze-last, agent-based, …
- Slightly more tricky in binary division
  - Make the division such that the custom room fits

A hand-designed structure (red) in Cogmind

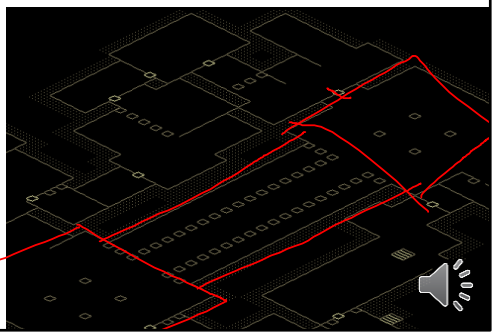
A great reddit thread about using PCG in some of the current rogue-likes: https://www.reddit.com/r/roguelikedev/comments/9pebns/faq_friday_75_procedural_generation/

Mark Johnson describing his interaction of handmade and generated in Ultima Ratio Regum (with data from several other games). https://youtu.be/Z_1T6f2ThzI

Case-study: Diablo

- *Fans have reverse-engineered Diablo's source code*

- Different algorithms for different areas
- Tile-based approach
- Small number of pre-authored *sets*
- Multiple passes
- Restart if fail
  - Probably quick-fix

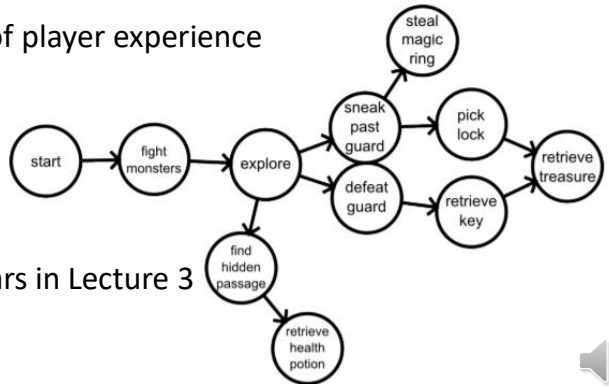


sets = pre-authored combinations of tiles
Switching to a completely different PCG algorithm mid-game is quite an approach for large-scale variability. Obviously, effective. Also obviously, hard to tune.

https://www.boristhebrave.com/2019/07/14/dungeon-generation-in-diablo-1/

Sophisticated constraints in dungeons

- A common approach to putting more hand-guidance to dungeons is to use a “mission graph”
- A high-level representation of player experience



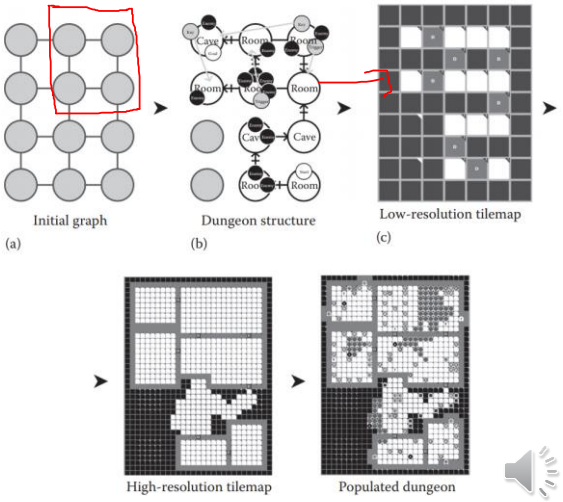
- We have seen graph grammars in Lecture 3

Turning graph grammars into content

- Four generic options:
  - Graph first, then independently space to fit it in
  - Graph first, nodes and edges also prescribe how to generate space (as prescription)
  - Graph and space at the same time
  - Space first, generate graph in that space
- None of these approaches is simple

Mission graph solutions

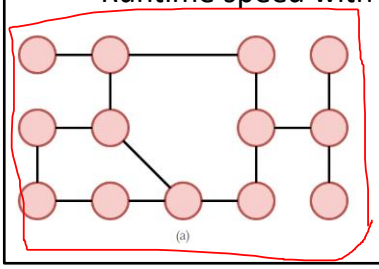
- Unexplored (Joris Dormans)
- Mission graph together with space
- 2D grid with cycles

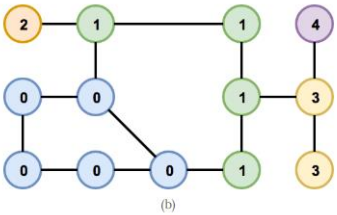


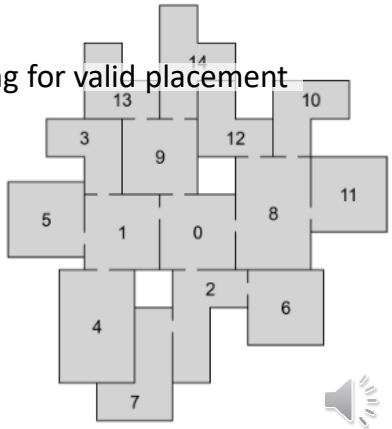
To know more detail, check chapter 9 – “Cyclic Generation” in the “Procedural Generation in Game Design” book.

Mission graphs with custom rooms

- Tight composition of handcrafted rooms by graph
- Ondřej Nepožitek, diploma thesis (in progress)
- Decomposition of graph into chains and searching for valid placement
- Heuristic-pruned search
- Runtime speed with reasonable sized graphs



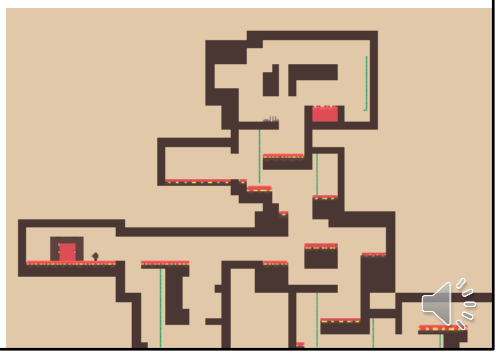




For more info, check https://ondra.nepozitek.cz/blog/
Builds on paper by Ma et al. (http://chongyangma.com/publications/gl/index.html)

Side-note: Dungeons vs. platformers

- Through simple lenses, dungeons and platformer levels only differ in one aspect – gravity
- Therefore, many dungeon / maze approaches may work for platformer levels
  - Just avoid long vertical stretches



Ondřej Nepožitek's generator, platformer version

Some things may be more complicated, platformer levels have usually more constrained design (only few places where to bind levels), but the approaches can be similar.

Procedural Items

Not calling it loot, because items of in-game shops are usually distinguished from loot and can also be procedurally generated.

Why do it?

- Can provide a lot of variability and replayability (Diablo 2 is still played)
- Excitement of finding rare things
- Loot economy is interesting

Why not do it?

- It may be hard to do well
- Some part of gameplay will be just examining items

Also, Path of Exile (PoE) devs mentioned that the best items from early PoE still haven't been found in meaningful quantities.

Procedural loot

Common method – pick from a selection of base items, add some random modifiers

- Pairing with affixes (Dagger of Gods, Burning Sword)

Worry about:

- Variability
- Underpowered / nonsensical items
- Power scaling

Nonsensical and weird items may make your game more interesting. Items that just aren't good may provide only boredom. As always, playtest!

The item generation can easily become quite complicated – check https://diablo2.diablowiki.net/Item_Generation_Tutorial
For interactive examples – go https://xteare.itch.io/path-of-exile-item-generator, https://poecraft.com/itemmods

If you're making a game that is heavily dependent on loot, you'll probably spend a lot of time developing it and come up with your own method. We will just mention some common things to look out for.

Large-scale variability

- If both a sword and an axe can have the same modifiers, player may not feel any difference between them
- Consider creating some themes or constraints

Borderlands uses a multitude of weapon manufacturers to provide themes in weapon design.
The approach is building blocks (preset parts of weapon, e.g. scope, clip, …) with some ranges on how powerful certain blocks can be.

This variability is also used in the RPG Sproggiwood.

Reasonably powered items

- If a high-level item randomly gets +1 cold-damage, is it meaningful at all?
- Make your items meaningful

(minimal ranges for stat modifiers)

Path of Exile is a game that builds on the items quite a lot (it is a mainly PvE game that tries to have people playing forever) and it has to keep producing exciting new item possibilities. https://www.youtube.com/watch?v=tmuy9fyNUjY

Power scaling

- In most RPGs, player scales (logarithmically / other concave function)
- How long will a good drop be useful?
  - Most often, you want to balance between short / long time

If the player can only experience some loot for a short time, they can lose the appeal of looting, since the items won't provide much gameplay.
If the time is too long, however, they aren't provided gameplay variability from the loot system.

How rare?

- It depends on the experience you're aiming at
  - Measure avg. hours to grind item (20+ hours may be long)

In MMO's, you can have items that a player is just not likely to get – still interesting for the economy:

- In World of Warcraft, Big Love Rocket has 0.03% drop rate where you can only try once a day in 2 weeks of a year
- In Path of Exile, Mirror of Kalandra has less than 0.001% drop rate (but can happen anywhere)
  - Drops once in the order of 10^4 hours of gameplay

WoW had items in the game that were actually unobtainable, just to mess with dataminers.
If you used every opportunity to get the Big Love Rocket, it would take you about 238 years on average. There are probably better multiple-lifetimes goals.

Loot tables

Non-programming designers want to work on loot -> you may need a table with a large number of parameters:

- drop probability (weight)
- drop only once
- drop locations
- drop always on
- ….

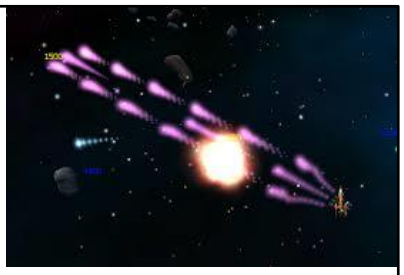
Also, to comfort unlucky players, consider a “pity-timer”:

- Maximum interval between some quality / rarity of drops

This is a good practice in using random numbers in a lot of procedural content. If you want something to happen in 1 / 5 of cases, it may be better to take a random number from 1-10 and make it happen on that case, than to just make it happen with 20% probability. One 2020 GMTK video goes into details of randomness in games: https://youtu.be/dwI5b-wRLic

Example of implementation – opening boosters in Hearthstone uses pity-timers for legendaries and epics. You are guaranteed to get a legendary at least once each 20 booster packs. It also recently added that you do not get cards from boosters, that you already own the max usable amount of. This helps to reduce the grief of unlucky outliers.

Interaction with item system



- Common approaches:
  - crafting
  - modularity (upgrading, assembling, socketing, …)
  - items / effects that increase item drop chance or rarity

- More advanced:
  - Path of Exile's maps with modifiers
  - Galactic Arms Race's adaptivity to player

Enemy encounters

Fun fact: in Pokémon games, encounter system and loot system is the same thing.

Necessary?

- You need some way to place enemies
- Worry about: difficulty and enjoyment

Approaches you already know:

- Make enemies part of the terrain
- Building blocks
- Templates
- Shape grammars
- …

(Flow: The Psychology of Optimal Experience, Mihaly Csikszentmihalyi, 1990)

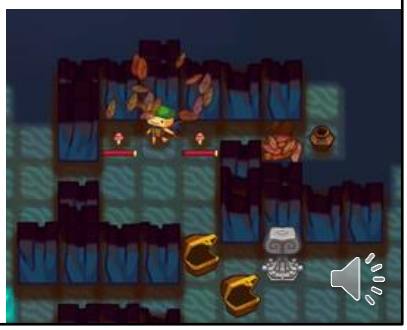
Simple methods

- Place random enemies throughout the dungeon
- Make them more difficult the further player is from the start
- Adapt to player level

- It may be worth to place groups of enemies
  - Generally, surrounding things with more things provides more natural feel
- Keep theme in a single room

Difficulty adjustment

- Usually, some pre-determined pacing (lvl1 enemies in dungeon level 1, …)
- Special – out-of-depth encounters
  - Rare chance to find an enemy that is usually spawned much later
  - Big reward
  - Should be optional
  - May provide more variability
  - The enemy may dominate the encounter
- Also, difficulty scaling
  - Adjust difficulty to player level



Sproggiwood features out-of-depth encounters

Side-note: Enemy waves

Enemy can spawn in waves in RPG, strategy and other genres

Spawning on:

- timer
  - designed pressure and pacing
  - issues if player too strong / too weak
- killing whole wave
  - alternates pressure / release
- killing enemy
  - spawn new (1-2) enemies on each kill
  - continuously increasing pressure
- low hitpoint total of current wave
  - simple adaptive method
  - avoids stalling

Tower-defense (a subgenre of strategy games) is also well known for spawning enemy waves.

Notable addition – allowing player to spawn next wave early (for bonus, e.g. Kingdom Rush)... mostly solves the issue with players being too strong (both from overleveling or just unexpected skill)

For more info about enemy waves, check chapter 14 – Procedural Enemy Waves in Procedural Generation in Game Design.

Side-note: Enemy waves

Enemy can spawn in waves in RPG, strategy and other genres

Spawning on:

- timer
  - designed pressure and pacing
  - issues if player too strong / too weak
- killing whole wave
  - alternates pressure / release
- killing enemy
  - spawn new (1-2) enemies on each kill
  - continuously increasing pressure
- low hitpoint total of current wave
  - simple adaptive method
  - avoids stalling

Diablo 3 cursed chests – spawn on killing enemy

Tower-defense (a subgenre of strategy games) is also well known for spawning enemy waves.

Notable addition – allowing player to spawn next wave early (for bonus, e.g. Kingdom Rush)... mostly solves the issue with players being too strong (both from overleveling or just unexpected skill)

For more info about enemy waves, check chapter 14 – Procedural Enemy Waves in Procedural Generation in Game Design.

Enemy AI

- In most RPG games, AI is scripted and doesn't adapt
- The goal may be to make it "believable" instead of super-smart
- For more information, check out Cyril Brom's and Jakub Gemrot's course "Human-like artificial agents" (NAIL068)

- Simple behaviors may do the job
- Notable exceptions – Neverwinter Nights, Shadow of War

Procedural puzzles

(not only) in dungeons

This section may become a full lecture in the following years.

Why?

- You want to have an endless supply of puzzles
- You don’t want the player to google the solutions

Why not?

- It is one of the hardest things to generate procedurally

Approach 1: Don’t do it

- All things considered, maybe hand-making your puzzles is the best approach for everybody (cheaper, higher quality)
- Nethack (rogue-like) contains fixed sokoban puzzles

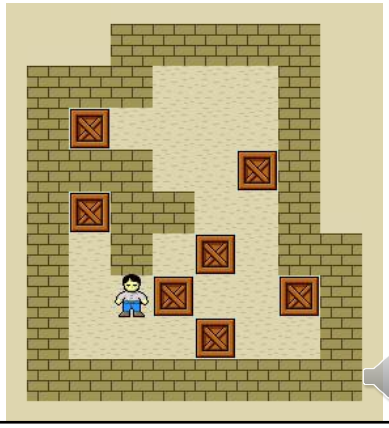
The sokoban player starts on the left staircase (**>**) and has to push 12 boulders (**o**) into the holes (**^**) to get to the right staircase (**<**)

Approach 2: Random

- Some quick puzzles can work with a random setup
- Might not guarantee solvability
- Examples:
  - Desktop dungeons
  - Minesweeper
  - Solitaire
  - Jigsaw puzzles

Approach 3: Reverse from solution

- Generate a solution, than make reverse moves to get puzzle
  - Random, or search for far-away state
- Might not guarantee challenge
- Examples:
  - Sokoban
  - Mini-game in Opus Magnum
  - Lloyd's 15
  - Rubik's cube



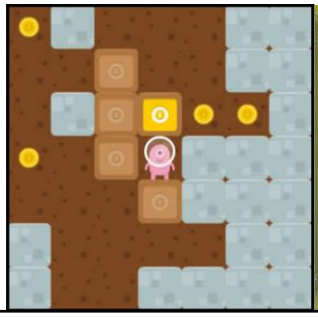
Note that finding the longest path in a graph is NP-hard.

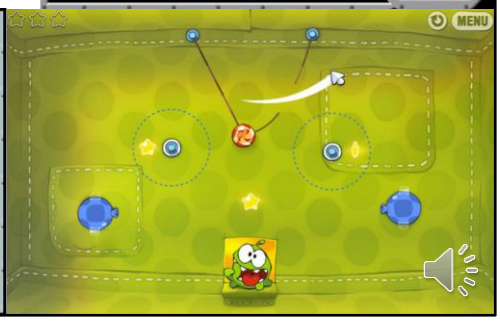
Sokoban generation: https://www.youtube.com/watch?v=ljj6rAaM4A8
Included video from: https://en.wikipedia.org/wiki/Sokoban

Approach 4: Sophisticated!

This may be time consuming

- Answer set programming
- Monte Carlo tree search
- Evolutionary algorithms
- …





Answer set programming in Refraction (A case study of expressively constrainable level design automation tools for a puzzle game, Smith et al, 2012)
Sokoban with MCTS (Generating Sokoban Puzzle Game Levels with Monte Carlo Tree Search, Kartal et al, 2016)
Context-free grammatical evolution (Automatic generation and analysis of physics-based puzzle games, Shaker et al, 2013)
Mazes with EA (Automatic generation of game elements via evolution, Daniel Ashlock, 2010)

Dungeon summary

Take-away points

- There is a plenty of methods for generating dungons
  - Even mazes have a bunch of different algorithms
  - Pick a method that best fits target gameplay
- Try to have large-scale variability (use themes)
- Good items systems take a lot of work
- Enemies are usually placed very simply
- Procedural puzzles are hard

Course takeaways

Procedural content generation

- can have a lot of value for your game
- **may** not be difficult to implement
  - you should carefully evaluate it
- is quite an unexplored territory
- → promotes innovation
- …
- is fun!

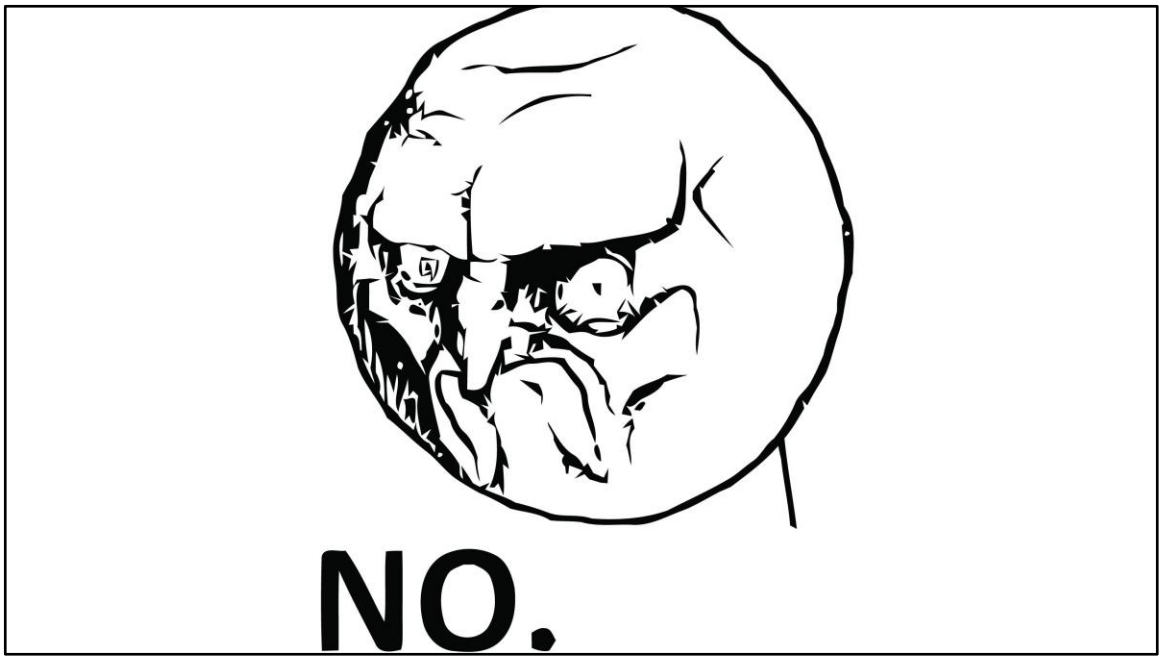
You should now know the pros/cons and should carefully evaluate with each of your games, if you want to use it ☺

Video sources:
Caves - https://www.reddit.com/r/proceduralgeneration/comments/dnezn4/update_on_my_cave_generator/
Town - https://www.reddit.com/r/proceduralgeneration/comments/fjnerf/2d_city_generation_for_my_game/
Lizard Animation + Water - https://www.reddit.com/r/proceduralgeneration/comments/frr643/procedural_animations_with_3d_collisions_in_a/

Q & A

cerny@gamedev.cuni.cz
discord.gg/Zts98PGw6z

The end


NO.