Cozy Caves

A D&D Dungeon Generator

Dungeon Generation Module

Written By
Carlo Cigaral
Gavin Lim

Reviewed By
Abdulrahman Asfari

Procedural Generation Algorithms

Possible Procedural Map Generation Algorithms

Binary Space Partitioning

A common procedural map generation algorithm among roguelike games. Involves recursively partitioning the bounding space (the canvas) into smaller segments until an end criteria is met. Final room layout can be abstractly represented as a binary tree with each divided region being a node with the initial bounding space as the root node, partitions between regions representing hallways/connections between nodes, and each final region being leaf nodes once recursion is complete.

Benefits:

- Efficiently divides our space into a tree-like structure that limits the search space and overall complexity of the layout.
- Gives us a number of parameters (termination criteria/depth of tree, partitioning heuristics - doing Recursive Division instead of Binary partitioning - Read more into integrating k-D Tree algorithms into procedural generation as well, etc.) to tweak to find an optimal balance of the number of rooms - shape of the rooms - space between rooms, etc.
- Controllable complexity and performance scalability from the controllable parameters but does overly rely on manual tuning as the algorithm provides some level of complexity inherently.

Things to consider:

- Space can be easily represented as a 2d array with room layout as a tree structure however this may cause conflicts depending on the Room object data structure (Will need to discuss further).
- Due to grid-like nature of the BSP final room layouts tend toward similar layouts and grow stale (too many rooms, too few spaces between rooms, ends with grid of rooms and little to no variation) as well as the chance of maps not fitting the context of a dungeon/cave. This can also largely depend on configuration of parameters (can be avoided/mitigated through room culling or varying room shapes/layouts)

Simple Room Placement algorithm

Simple and intuitive procedural generation algorithm. Involves recursively randomly placing rooms in the bounding space while ensuring they meet some criteria (no overlapping, enough space between neighbouring rooms, etc).

Benefits:

- Room shapes and sizes seem to be more flexible to work with the Room Generation module compared to other procedural map generation algorithms.
- Lacks any detail on hallway generation/logic behind connecting rooms and so will require more research and experimentation.

- Map structure/room layout may be more modular due to randomness and may end up being more suitable within our desired context of a dungeon/cave

Things to consider:

- The straightforward implementation and lack of logic behind room placement (depending on placement criteria) means room layouts may lack complexity compared to other algorithms.
- Requires much more manual parameter tuning (maybe play around with ML style training?) as it is closer to complete randomness compared to other algorithms.
- More reliant on dynamic room shapes through the Room Generation module to add complexity and could add unnecessary workload onto another module due to inherent lack of complexity in the algorithm.

Cellular Automata

Utilises cellular automata with rules/logic similar to Conway's Game of Life to procedurally generate an organic cave-like environment. A niche idea however lacks any form of possible room generation integration and seems like it would be a completely different structure on its own. Further research and discussion could be open when approaching the extra goal of having different environments utilising different algorithms to produce more organic map layouts.

Model Inputs/Parameters to tune

These parameters should initially all be set to some default value/settings during basic implementation but be coded in a way that will allow for easy integration with user input for manual parameter tuning. Specifics as to what parameters will be available to us to set and customise will largely depend on the algorithm decided upon. A high level of customizability should be available in order to produce complex map layouts while also remaining coherent.

Termination Criteria

Regardless of the algorithm we decide on, the termination criteria of the dungeon generation determines the map complexity and should be tuned to fit within the context of the project (Too many rooms becomes incoherent, too few rooms will lack complexity). Some parameters for termination criteria to control will be:

- Maximum iterations/Depth
- Desired number of features (e.g. number of rooms)
- Quality/Fitness threshold (e.g. overall map coverage/fill percentage)

Map Dimensions

The size of the bounding space itself is a fundamental parameter and will also determine map complexity as well as processing complexity. A map with dimensions too large will quickly cause certain recursive models to become too complex. Initial implementation will use arbitrary map dimensions and later be fine tuned as we see fit.

Room placement/culling/carving constraints
Determining the criteria and logic behind room placements and when to cull rooms in certain algorithms will control the modularity and complexity of the map layout. This is going to be dependent on the chosen algorithm but will still be flexible between algorithms as well.

Random Seeding
A master seed will be used (defaulted to a random value but allow the user to change this themselves as they see fit) for map layout generation (For BSP, randomising the placement of partitions). Each time the Room Builder will be called a secondary seed will be generated through the master seed to create randomised rooms.

Map Data Structure
The Dungeon Gen Module will encapsulate all Room Objects within the dungeon and also contain its own internal grid of the map layout where Room coordinates will be kept to later be passed further onto the Rendering Module.

Communication with Room Module
This will largely depend on the type of algorithm(s) we choose to implement as to when and how the Room Module will be called.

Room Object representation on Map
Rooms will be given as a self contained object, having all necessary information per room within itself. The internal grid of the Map will only need to keep track of certain positions of individual rooms rather than the position of every single tile.

Generation Pipeline
BSP
- **Inputs -** Process parameter configurations as inputs
- **Recursive Partitioning -** Recursively partition bounding space until termination criteria is met
- **Room Culling -** According to configurations cull unwanted rooms randomly
- **Room Generation -** Use Room Builder to generate random rooms in spaces not culled
- **Hallway Connections -** TBD