Doomlike Dungeons: How It Works

Doomlike Dungeons seems to work somewhat differently than most dungeons generations systems, therefore it only makes sense to write it down in way that doesn't require reading through 10k line of Java code.

Influences and Design Consideration

The most direct influence on Doomlike Dungeons is the Oblige level generator for Doom, specifically the version 3.x series. It is not a port and does some things quite differently however. There are good reasons for this. At the same time, it also barrow many general idea and attempts to create similar contact, which is somewhat different from that typical of Roguelike games.

Before Oblige, there was Slige, a level generator for Doom which produced notoriously levels with a notoriously linear levels topology. In contrast Oblige uses a "quest" system in which locked doors send players off on different path to find the correct key or switch, resulting in a branching tree-like structure (though some newer version and small cycles within a quest area). In addition, Oblige adds some extra side rooms and subrooms.

Oblige 3.x and 6.x also procedurally build complex rooms from shape primitives and simple motifs, rather than using prefabs, allowing a near infinite variety of room; prefab based construction was tried in the 4.x and 5.x series, but 6.x reverted to the 3.x system as the prefabs became repetitive. Yet another important feature is the use of themes, separating tech, urban, gothic-hell, and gory-hell palettes for texture, monsters, and some architextural variable, though these are buried inside lua scripts containing logic.

Doomlike Dungeons was intended to bring a similar style to Minecraft, but some changes in style had to be made. The use of shape primitives and procedurally generated rooms was made central, however, no diagonals were included so as to fit better with the blocky world. The quest system makes no sense in a game where players can simply dig through wall, so instead of a branching tree and highly cyclic network of rooms was planned, connecting a few central nodes (replacing quest goals). Since the mod would be distributed as compiled byte-code in a Java jar, themes became external, pure data config files. In addition, Doom does not spawn and despawn enemies, but simply places them on the map, and controls where the player starts so there is limit to the gear going in and the level cannot be left until completed. Since mobs will be produced by spawners and can despawn, while the player (or players) can enter any time with unpredictable gear, possibly leaving the and returning, there is no simulation of battles or calculation or gear; instead mob spawner are placed with more random difficulty and density and loot is given based loosing on room difficulties.

Another consideration involves differences in when generation occurs – both Slige and Oblige are off-line generator, creating content in a separate program which is later loaded into the game. In contrast Doomlike Dungeons is an online generator, creating content while the game is running. This forces some limitation, as extensive processing can not be done. Oblige can take several second to over half a minute to process on a fast CPU no one will complain – a 30 second lag spike is unacceptable in a running game, so processing must be limited to keep generation times to well under a second.

One important point in all of this is that I've always wanted play with the mod to be and feel like Minecraft, complete with placing and breaking blocks, tunneling, pillaring, blocking-up, etc. – there's no need to mod Minecraft into Doom, if you want to play Doom you should play Doom. Thus, there is no attempt to create things like unbreakable blocks to stop "cheating" since this is not cheating.

Dungeon Placement

In early versions dungeons were placed based on a simple random chance per chunk. However, this did not always produce good result, especially when the probability was set low. So, a new system was created. The mod groups chunks into square with a number of chunks per side determined be the dungeons frequency setting, When a chunk is generated a special RNG is generated that is seeded with the same seed as all other in the same square, and two number from 0 to length-1 (in chunks) are generated. If the chunk is the position corresponding to these number on the X and Z axis counting from the beginning of the square it get the dungeon for that square . Thus, one in generated per square of chunks, but where in it its centered is determined randomly.

The Dungeon Planner

Creating the actual dungeon involves four processes, planning the layaout / placing rooms, planning / generating the rooms, correcting errors and bad designs and building. The first two steps, dungeon planing and room layout, are interleaved, with rooms being placed and then the next room laide out before the next is placed. As rooms are placed they are put into a room list allowing access by index and list iteration; the list of real rooms starts at index one as zero is used as a null-room representing areas outside the dungeon.

Setting Up and Applying Themes

Planning out a dungeon starts with initializing some core variable for the dungeon, mostly involving themes. A theme is picked from the themes available for the biome at the center of the dungeon (a central block in the chunk that the dungeon is build around). Obtaining a theme is based on first unioning all sets of themes for all Forge biome types that apply to the biome being used, then removing all themes that are not allowed for any of those biome types.

Once a theme is picked it is used to pick architectural feature from the theme and pick a default blocks the parts of the dungeon, along with the size category of the dungeon. Some other classes are also initializing, such the list of rooms eligible to grow new rooms.

One very important structure created at this time is the 2D map of the dungeon. The map contains a number of 2D arrays that hold information such as floor / ceiling / wall blocks, floor and ceiling heights, and room IDs (indicies in the main RoomList) that areas belong to. Very early on a more abstract planning system was considered, but this was deemed an over-complication, and so was replaced by mapping the dungeon as it was designed to keep track of areas used directly and later to build from.

Node Rooms: Entrances and Destinations

Once core dungeon features have been determined a small number of node room (aka, hub-rooms) are placed at random locations in the map. These rooms are always on the larger end of the available room sizes. Some of these will later become entrance rooms, with no loot or spawner and a way out, while other will become destination rooms (aka, “boss rooms” or “loot / treasure rooms”) with extra spawner and loot and at least one harder mob spawner. Besides the main RoomList, nodes are also place in a small array for node specific processing, especially node connection.

Connecting the Nodes

Once all the node rooms are built they are then connected. To determine which nodes to connect, the first is place in a separate array and is connected to another node picked at random for the others and then moved to same array as the first; thus there are two node arrays, one for connected nodes and one for unconnected nodes. After this, the process is repeated, connected a randomly chosen node from those that are connect to a random node from those that are not connected. This is based on the logic that if you can reach A from B and B from C then you can reach A from C by at least one route (through B). Thus, all nodes ultimately connected, though the exact connection and routes are randomized.

To connect one node to another rooms are sprouted off from the two nodes being connected, each taking a turn. Each new room created moves closer to the other on either the x or z axis, though with axis is determined randomly with a weighted function in which the weight of each direction is its overall difference from on that axis. This causes the series of rooms to home on its target while keeping the specific steps unpredictable. As this is done, each nodes rooms are added to a list of rooms known to connect to it. If an attempt to add a room moves into a space already occupied by an existing room, that room is added to the list and the next addition room will sprout from that room. When either nodes moves into a room already in the other nodes list the two are considered connected. This does mean an extra partial route with a dead end can be formed, which is intended and desired. Note that it's also possible, though rare, for the rooms to grow into an area where they don't have room to create new rooms and are stuck, making it possible, though very rare, for the connection to fail – this is, of course, not a desired outcome and is fixed later in dungeon planning.

Growing Side Rooms

Once all the nodes are (at least in theory) connected, extra side rooms are added. To do this, the current list of rooms is iterated, doors available in each room are given a chance to grow new rooms, sprouted off to the side as extra areas. New rooms are put into new a list to replace the one being used, so that no room gets more than one chance, but is also possible for a series of multiple new rooms to emerge from one. This, called the GrowthCylce, is repeated until either there are now more available doors to grow from (the list of new rooms is empty, the last cycle having produced no more candidates) or the maximum number of allowed rooms for the dungeons size have bee reached.

The Room Planner

The room planner may create either a typical room (built from shape primitives) or occasionally a cave-like room using cellular automata. It was originally intended to have several atypical options, including small mazes and small set of rooms created from a single large room by partitioning (the atypical rooms being created from special big rooms) so as to vary the dungeon style. However, mazes and room sets (partioned areas) have not been implemented so far, and for most of the mods history only typical rooms were implemented, caves being a relatively recent addition.

Planting the Seed

Creating a new room starts with defining a space for it. To do this a system is used in which a 1x1 block seed of a room is placed that then tries to grow into full sized room. The target size is randomly determined in advance, so room seeds do not all try to grow the maximum allowed size and a variety of sizes is essentially guaranteed. The room side with first randomly pick with axis to expand on first, either x or z, so as to avoid a bias toward long, narrow rooms being found along one dimension and not the other. The seed will try to add new tiles (columns of blocks) to its area in each direction in the chosen dimension until it reaches it target size of both ends are blocked by existing room or the edge of the dungeon map. The, it will try to expand with rows of of tiles in the other direction until that dimension target size is reached or the addition of rows is blocked on both side. If either dimension is less than five, the room is abandoned for not having enough room to be interesting. Otherwise, the claimed area is used to determine the center coordinate and dimension of the new room and this data is passed to the Room class where its used to create the actual room.

Applying Themes

When a new room is created a the block to be used are first determined. This is done with a series of decisions designed to created varying degrees of consistency and transition types between rooms. A room starts out with the block of the room and was branched from (for the initial nodes, these are the default block for the dungeon). The dungeons consistency level (picked based on its theme) is used to determine if changes should be made. If so, each block category (wall, floors, etc.) goes through a series of decisions, all based on consistency, to determine if it should inherit from the previous room, take the dungeon default, or pick another block available in th dungeons theme. The purpose of this is to simulate varying design aesthetics: highly consistent dungeons will use the same blocks in all or most rooms, moderate consistency dungeons are likely to form into zones with similar blocks and gradual transitions in which change progress gradually with some blocks changing while other are kept and changed later, while low consistency dungeons will have a very random appearance with rooms changing erratically and frequently.

Adding Features

Typical rooms are built using shape-primitives, most of which resemble letter of the Latin alphabet modified to have a squared shape, though full rectangles are available for most purposes. Once again the theme derived architectural variable are used. First an overall level and type of room symmetry is determined, which might mirror along either or block axes (x or z), across the mid-line ("transpose" symmetry), or on a rotational bases (180 degree "rotational" or 90 degree "swirled"); this often create the illusion the weird construction are planned, and also create more of them by restricting the area available to place shapes so the are more likely to pile up. A themed dungeons variable ("vertical") are also used int determining the ceiling height relative to the floor.

Next, it is determined whether or not to have a whole room pattern, a single shape that is used for the room. Whole room patterns are based on considering the dungeons complexity variable in such a way that they are rare with low or high complexity but common with moderate levels. Whole room patterns are also picked from those that are compatible with the rooms symmetry (no symmetry can lead to any). Examples of whole room symmetries include rooms with a single central liquid pool and those in which an 'S' shape bridges a liquid.

If rooms do not have a whole room pattern than a number of features are added based on the dungeon wide complexity level. Examples of feature include platforms, areas cut out from the room as internal wall, pits / depression in the floor, and pools of liquid (or liquid stand-ins). This means the low complexity dungeons will have a lot of big empty or mostly empty "box" room, while high complexity dungeons will have a lot of complex and sometimes strange rooms.

The selection of feature is done by first shuffling a list of feature selectors, each of which will determine whether or not a specific feature type will be placed on that pass through the list. If it is, no other feature will be considered on that pass. The randomization is to prevent a global bias toward some feature types that would exist chance for each were tested for in a consistent, hard coded order, and also gives rooms a there own preferences, differing from other rooms. The probability of selecting a particular feature when its chance comes up is a dungeon wide variable derived from the theme. Thus, both the themed dungeon variable and the rooms shuffle contribute to the likely hood of various feature being used.

Doors (really doorways or arches) are also added, being randomly placed randomly in the walls, and registered a list of doors for further processing, including potentially being used to make new rooms, potentially being removed if they go no where, and in testing room passibility.

Monsters and Loot

The Room class also picks spawners and chests, placing each on a list to be placed after the dungeon is built. First the number of spawner is determined, considering if the room is an entrance (none), a destination (multiple), or neither (either option, randomly), then the difficulty of the monster is determined based on the difficulty setting from the configuration file (and randomness) and a mob of that difficulty is chosen. For destination rooms, an extra spawner is placed in the center of the room in the floor (to be under a chest), which is always of a harder type. The room is assigned a difficulty level based on the hardest mob, with a +1 bonus for having multiple spawners and a +1 bonus for being a destination.

Once spawners have been placed chests are added. The number of chests and the value of loot in each is based on the rooms difficulty level. Destination rooms also get an extra chest on top the harder spawner. Chests may be of three types: gear (like the weapon / ammo placements in Oblige), health (like the health / armor placements in Oblige, but with food and no armor), and treasure (a reward, like the gold, etc., found in rogluelike and RPG games, since these voluntary dungeons do not have escape to another level are a goal). Gear and loot chests can be found in any room, and even rooms without spawners can sometime have “weak” chests with some junk, but only destination rooms have a treasure chest (exactly one).

Cave Rooms

Cave rooms are handled differently, and use a common for of cellular automata originally developed for roguelike games and also used in many version of Oblige. Some intentional difference are made. A room sized grid (2D array) is filled with ones and zeroes, and each of these is replaced by either a one or zero based on summing the cell and all its neighbors and testing against a threshold value – a common system, though modified to include the cell itself for simplicity and to have a threshold that is randomly adjusted slightly for variable generation. Once this is done, it is repeated with the ones from the previous step being carried over as a kind of mask, to select non-wall areas where the floor and/or ceiling may be incremented or decremented by one to create a more interesting 3D cave. The number of additional passes is determined by the dungeons vertical variable.

Doors, spawners, and chests are placed just as in typical rooms. Most cave-like rooms are given a cave block, usually smooth stone for most default themes, but may occasionally have typical wall blocks, creating a strange, chaotic room that isn't fully cave like in appearance.

Pass Two: Dungeon Improvements and Corrections

In many version the planning would be done after all the rooms were built. However, though this produced good dungeons most of the time, it could also produce duds. To fix this, some quality control was added.

Finding and Fixing Doors

The first step in improving dungeon plans as to analyze and fix doors. The doors in each rooms are tested to see if they go anywhere; if there is no other room on the other side they are removed (previous to the step, dungeons were littered with door to nowhere with stone and dirt visible in them). One door to each connected room is selected for use in the next step, the passibility test.

Passibility

Passibility is tested using the A* algorithm (which I will not describe here – there are many resource explaining A*), treating each tile / block column as a vertex and adjacency along either the x or z axis as edges (but not diagonal, as a player can't squeeze between block corners). The cost of moving from one tile to another is 1 + 16*changes, while the heuristic is simply the Manhattan distance (distance on x and z summed) to the target door. The purpose is find the route that requires the least change (preferably none) the current layout. Once a least change route between the start and target doors, the route is traced back from target to start, making necessary changes as it goes and marking the tiles as “AStarred,” the latter being used to ensure these always have a floor block during building.

Routes between doors are process in much the same ways as routs between node rooms in planning the dungeon layout: If A connect to B and B connects to C, the A connects C. Thus, each door is tested to be sure it connects to exactly one other door.

As a side benefit, this also creates mazy tunnels and bridges in many rooms.

Connectivity

Another far less common but very serious problem is failure for all rooms to connect, leaving a small disconnected section of rooms that a player is unlikely to even guess are there, or worse, have a sole entrance into such an area with most of the dungeon hidden. Thus, though the problem is actually quite rare, it was not one to ignore.

To fix this problem, the rooms are first tested for connectivity using a breadth first search, in which each room is considered a vertex and a set of one or more doors between rooms as an edge. Failure to find all rooms from the first leads to and attempt to find the rest from the first room among those not found, and this is continued until all rooms have been found, effectively sorting the rooms into the already connected sections. Then, a slightly different implementation of A* is used to connect the center of a random node room in each section to the center of a node room in each other section. This implementation of A* differs from the one previous used within room in that it is allowed to use any tile in the dungeon map, even those of the null room (id=0). Leaving the dungeon proper for a zero tile has an added cost of 512, discouraging quickly leaving the mapped area and making long tunnels, but favoring shorter tunnels between geometrically closer areas. In addition, it adds a cost of +7 to any tile that is not marked as AStarred, so as to encourage using previously found routes through rooms. When processing it s corrections, leaving the established dungeon rooms creates a tunnel which is technically made part of the room it exited from, and continues the tunnel until it finds the next room.

Building the Dungeons

Building the Main Dungeons

With the planning now complete, the main dungeon is built into the world. This is done by simply iterating through the 2D arrays found in the dungeon map, removing blocks to create empty areas, (re)placing blocks to create walls, floors and ceilings. Doors are created by removing the three wall blocks at the bottom. During this phase, degeneracy (determined a room characteristic durring room generation) is also applied, allowing the possibility that some blocks may not be placed if they would replace air; degeneracy is more common for walls and ceilings than for floors, and floors cannot degenerate in an entrance room or in a tile marked as AStarred, ensure passibility remain and entrance room have full floors.

Once the dungeon is built, the list of rooms is iterated and spawners and chests are placed. Chests are also filled at this step, placing loot from the relevant list with a chance of an extra item from a randomly selected list (gear, health, or treasure). When gear is placed it has a chance to be enchanted, the higher the starting value the higher the chance; if so 0-30% of the loots value is converted to levels for enchanting, and the rest is left to determine the list to pick the actual item from.

Adding Entrances

Lastly, entrances are added above entrance rooms. Entrances may be either ladders, spiral stairs, or rooms / ruins with one those inside. The chance of an entrance room being a ruin is based on the dungeons degeneracy variable, while the blocks used are the dungeons default blocks. These structures are built entirely be procedures without the use of a map, prefab, or template.

Unused Ideas

Big Rooms, Mazes, and Room-Sets

Early version of the mod had special big room, with target dimensions between one and two time the usual maximum. These were removed because they increased connectivity problems, connectivity testing not yet having been implemented. These could possibility make a return.

Big rooms were intended to be the place to put mazes, partitioned room sets, and most cave-like areas, as well as having some left as over side rooms. Typically generated big room also had (and would have) more features added. These are unused ideas most like to be added in the future.

Variable Room Altitude

Early on it was planned that eventually the base floor height would vary between rooms (not to be confused with floor height in general – this would have been differences between rooms in the height of the main floor, before platforms and depression are added). This has mostly been abandoned on the basis that it would like involve a fair amount of processing and not really increase the enjoyability of the dungeons.

Multiple Levels

The original plan never included multi-level dungeons (put differently, they were to resemble single levels, not episodes). However, the idea has been toyed with since, and would be pretty easy to do – simply create a dungeon with exactly one entrance, and in the node farthest from the entrance would be a stair of ladder down to the (only) entrance to the next level. The problem is that, though not hard to do, it would multiply all planning and block-placing, along with triggered block updates, by the number of levels in the dungeon. While this would not take as long to build as a dungeon per level (part of the built time is actually vanilla terrain and world generation to create chunks to contain parts of the dungeon that are built into more distant unexplored areas), it would still almost certainly take substantially longer and most like cause very noticeable lag spikes. As a result, multi-level dungeons are not a priority, but something to possibly experiment with and likely never to be added.

Micro-Prefabs

Another idea that was not in the original plan but has been considered is the addition of small prefabs, representing such thing as fountains or pieces of furniture (not whole rooms). This would allow hand-build decoration to be added to the dungeons without removing the relatively pure procedural approach to room building. The prefad templates would probably be similar to those used by The Ruins mod and some parts of Fomivre's mods, but and extra complexity would be needed. Ideally, it would be possible to have both hard-saved blocks and blocks to be chosen from themes. It was also be required than they could be anchored to the floor, sealing, or both, with a repeated middle block, so as to make such things complex pillars possible. It would also, ideal, allow for stretch along some axes, with a field to show which axes can and cannot be stretched.

This is probably one fo the more likely feature to added, if I have time and am in the mood, but not a priority, and relatively complex to code (basically amounting to adding a more advanced clone of The Ruins into this mod). You should not hold your breath waiting for this to appear.

Mini-Game Mode: Extra-Dimensional Dungeons on Demand

Yet another idea I toyed with was adding a Bukkit-style command that could (if allowed in the config, probably not by default) take the player to an on-demand dungeon in a special dungeon dimension, allowing more dungeon play without forcing the player to look farther and farther from home for new dungeons (and on a server, potentially plunder distant areas where another player might later settle). Though I'm tempted to play with the idea just to learn about making dimensions, this is unlikely to be added to the mod as it seems more like something that belongs in a plugin, probably with hook for others. This is unllikely to ever be a real feature, at least not in this mod.

Per-Chunk Building

One last idea that has been considered very recently is sever dungeon maps and data with their central chunk, and having new chunks search for near by dungeons when generated. This way, only the portion of a dungeon found in a chunk would be built with that chunk, and no other chunks (other than the dungeon center for a newly found dungeon) would be generated. This would actually increase work slightly, but remove lag spike caused by generating large amounts of extra terrain at once with the dungeon. It would also lead to more consistent generation, as currently areas closer to player are placed in existing chunks where all other generation has occurred, while distant areas are generated before anything else. As a result of the better consistency in generation order dungeon would conform better the one-to-one mapping between seeds and worlds (currently, finding a dungeon from a different directions can cause subtle differences because the chunk have been generated in a different order). However, this would be a huge overhaul and have disadvantage as well, such as technically (slightly) more world-gen processing over all and producing some bloated chunks that might be slower to load. Some additional advantages, however, include making multi-level more feasible, and making it possible to remove the lower limit to dungeon density (aka, FrequencyScale) since the danger of the game trying to generated the whole word to hold overlapping dungeons would longer exist.

This is something I might experiment with someday, but due to the size and complexity of the overhaul involved, and the fact that I simply have other time demands its not really all that likely.