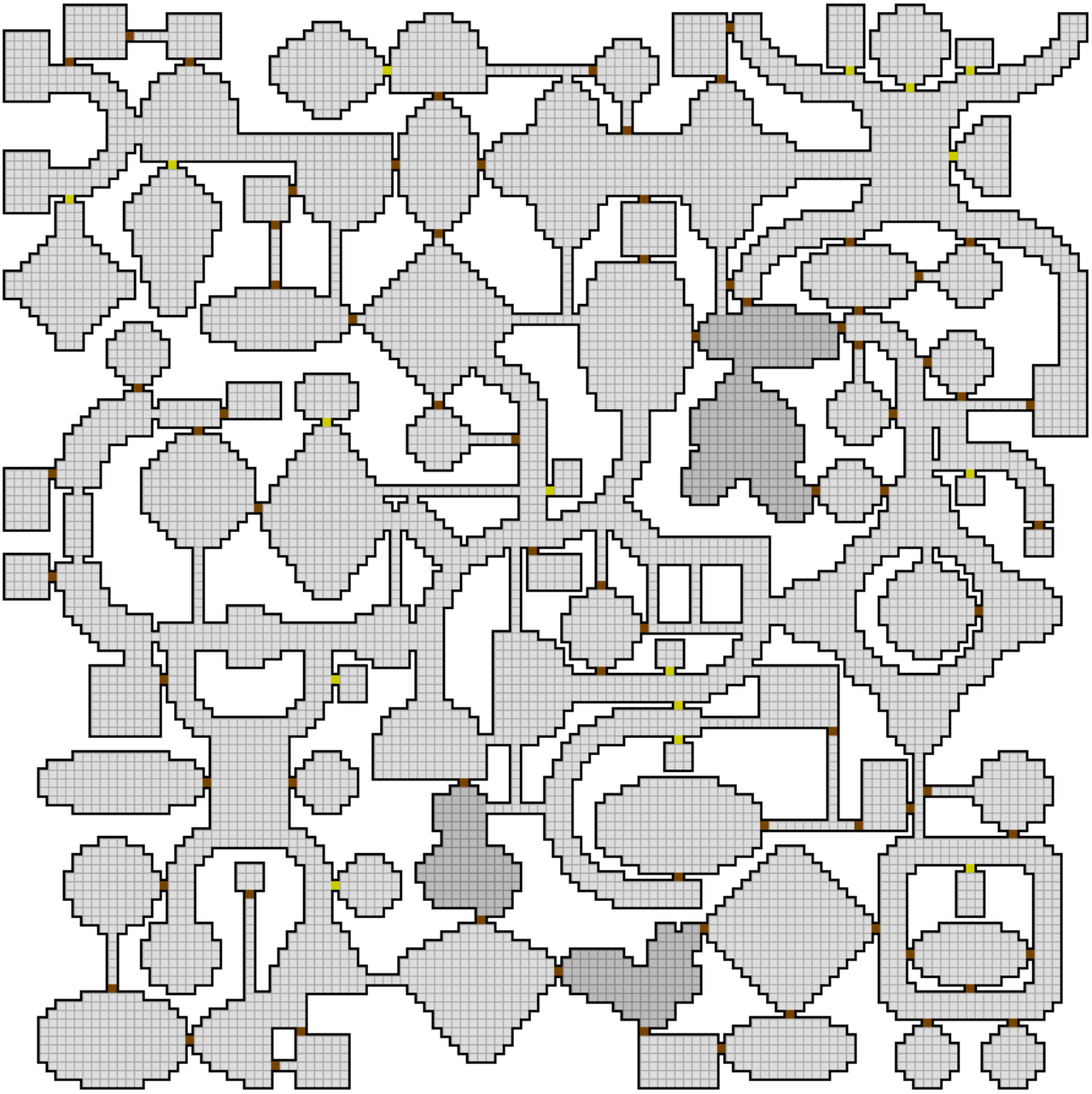
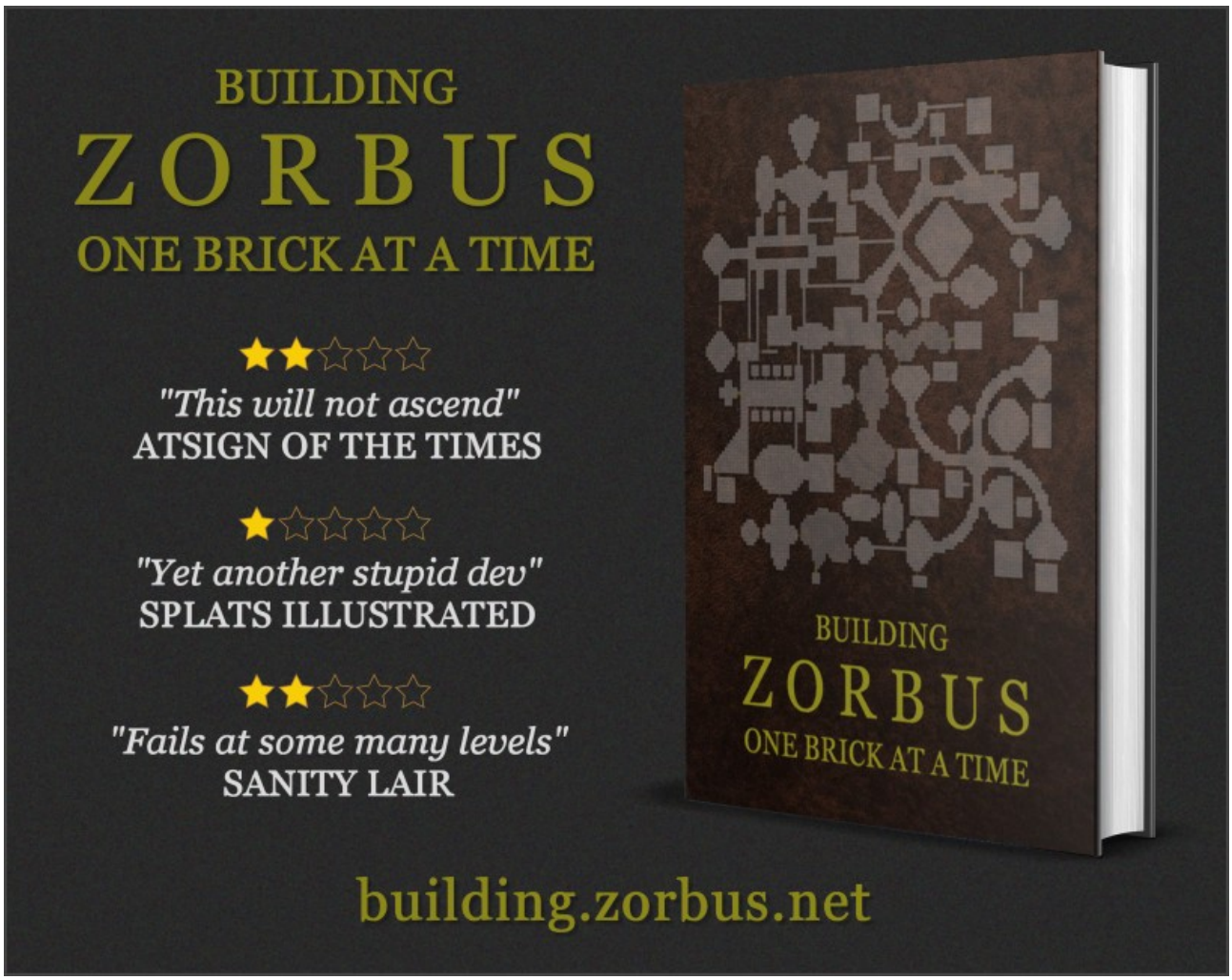


BUILDING
ZORBUS
ONE BRICK AT A TIME

updated

20-Oct-2022

Contents



Foreword

This ”book” documents various development related things of the roguelike game Zorbus.

There’s no code in this book, the game is closed source, and there’s no easy way for players to edit or modify it, but still, other hobby developers might find the methods and other stuff documented here interesting.

This was meant to be more comprehensive, sort of closure to the game project, but I just don’t have the time nor interest for more than this at the moment. I might however update this in the future.

Zorbus dev.

I can be contacted at joonas@zorbus.net.

The game can be found at www.zorbus.net.

Updates to this book can be found at building.zorbus.net.

General stuff

Using dice notation for various things

The game uses tabletop RPG dice notation to represent numeric ranges for weapon damage, effect amounts, duration, and many other things. If you're not familiar with it, and want to know more, check this Wikipedia article: https://en.wikipedia.org/wiki/Dice_notation.

I've extended the basic notation to represent damage elements (blunt, pierce, slash, cold, fire, etc.), and to use being statististics (experience level, skill ranks, etc.), and dungeon statistics (dungeon level) as modifiers.

Elemental damage

```
fire: 3d8 | per_magic: 1
```

= fire damage 3d8 + 1 per every point in *Magic*-skill

Elemental damage with multiple elements

```
pierce: 2d5,   cold: 2d5 | per_magic: 1/2
```

= pierce damage 2d5 + cold damage 2d5 + 1 per every 2 points in *Magic*-skill

Damage with modifiers against being type

```
blunt: 1d8,   against: undead, against_hit: 2, against_multiplier: 2
```

= blunt damage 1d8 + extra +2 to hit and 2 x damage against undead

Percentual chance to create specific loot in a treasure room

```
10 | per_dungeon_level: 3
```

= 10 + 3 per dungeon level

External data files

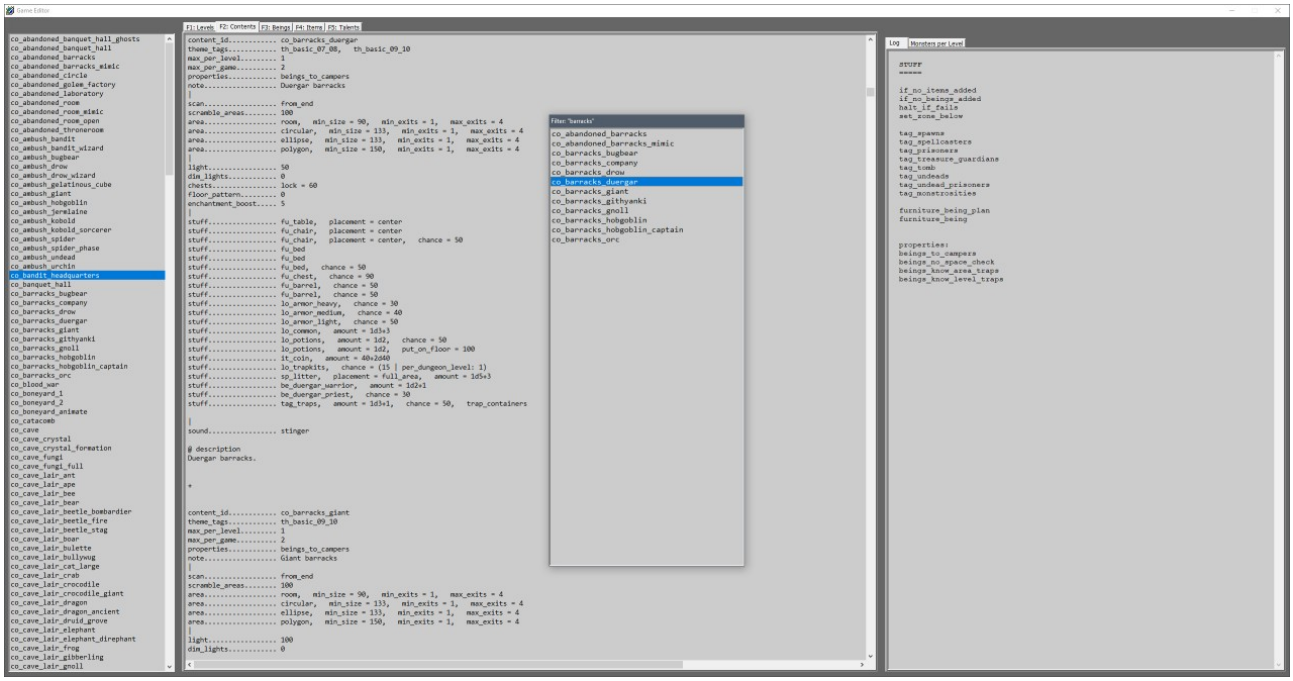
All game data is in 5 external text files:

- Levels — Dungeon structure, every dungeon level has its own entry
- Contents — Room contents
- Beings — Being statistics, emotion- and fear-presets, descriptions and dialogue
- Items — Item statistics and descriptions
- Talents — Talent statistics and descriptions

There's also an external binary data file for vaults (prefabricated areas, drawn with an external editor).

Each entry has an identifier with a prefix ("le_" for levels, "co_" for contents, and so on).

I created a simple editor to help editing the files. When you open the editor, all files are automatically opened. Entry identifiers are listed on the left, and you can quickly jump to a wanted entry via a jump list that can be filtered by writing a keyword.



The editor works like a compiler. When you press F9, the text files are parsed, syntax checked, and packed into a single binary datafile. If an error is found, the editor jumps to the erroneous file and line.

Tags

Tags in Zorbus are used to group data entries. Following tags are used:

- Theme groups contents
- Encounter groups beings
- Loot groups items
- Hazard groups traps
- Feature groups talents

Like data entries, these also have a prefix (”th_” for theme, ”re_” for (random) encounter, and so on).

Example of theme-tag

Part of a content entry:

```
content_id............ co_lair_kobold
theme_tags............ th_basic_01
max_per_level......... 1
max_per_game.......... 1
note.................. Kobold lair
```

This content belongs to the *th_basic_01* theme. Several other contents also have the tag.

In a level entry you could now have:

```
themes................ th_basic_01 = 2d4+5
```

Which would set 2d4+5 contents onto the dungeon level, chosen from all contents with *th_basic_01* tag.

Example of encounter-tag

Part of a being entry:

```
being_id.............. be_ape
|
unidentified_name..... ape
introduced_name....... ape
|
encounter_tags........ re_animals,   re_apes,   re_01,   re_02
```

This being has the *re_animals*, *re_apes*, *re_01* and *re_02* tags. Several other animals also have the *re_animals* tag.

In a content entry you could now have:

```
stuff................. re_animals,   amount = 1d3+3
```

The *re_01*, re_02 … *re_xx* tags are mostly used to group all beings that can be randomly placed on the level. On the first dungeon level data entry you could have:

```
stuff................. re_01,   amount = 1d10+20
```

Which would randomly create 1d10+20 beings, chosen from all beings with the *re_01* tag, and place them randomly on the level.

Example of loot-tag

Part of an item entry:

```
item_id............... it_potion_healing
name.................. Potion of Healing
|
type.................. device,   potion
|
loot_tags............. lo_potions,   lo_healing_potions
loot_amount........... 1
loot_min_dungeon...... 1
loot_max_dungeon...... 3
```

This item has the *lo_potions* and *lo_healing_potions* tags. Items *Potion of Extra Healing* and *Potion of Superior Healing* also have the *lo_healing_potions* tag.

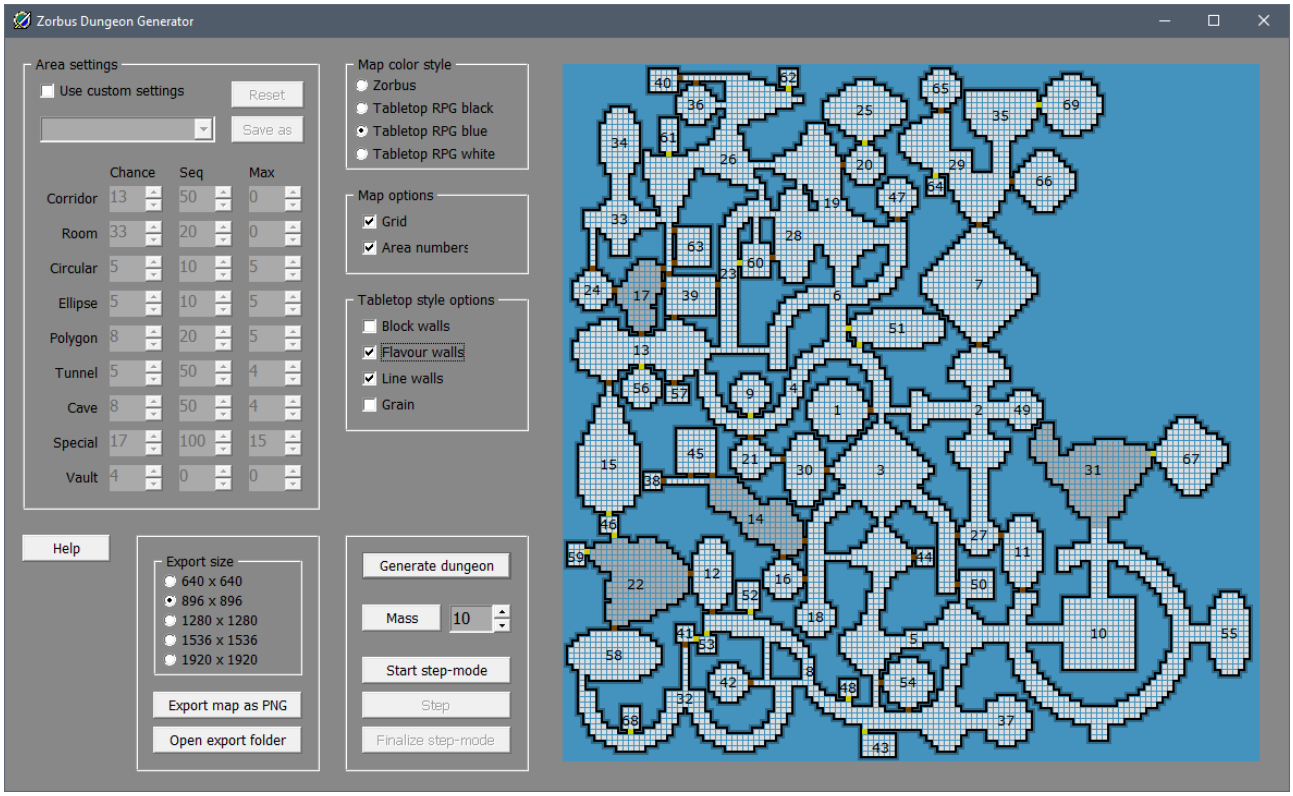
In a content entry you could now have:

```
stuff................. lo_healing_potions,   amount = 1d2
```

Which would create 1d2 items, randomly chosen from all items that have the *lo_healing_potions* tag. Note the *loot_min_dungeon* and *loot_max_dungeon* values. These control the dungeon level range where this item can be created. In this case, *Potion of Healing* will only be created on the first three dungeon levels, and after that it will be ignored when creating loot with the *lo_healing_potions* tag.

Dungeon building

The dungeon generator of the game has been released as a separate Windows tool, available at dungeon.zorbus.net. There's a step-function which adds one area at a time if you're interested in seeing how the algorithm works.



The algorithm

The dungeon building algorithm used in Zorbus is based on an article written by Mike Anderson, posted to Darren Hebden's Roguelike News website back in 1999. The article can be now found at RogueBasin.

The algorithm is very simple, easy to implement, and ensures that every area is reachable. The algorithm makes it easy to control the amount of different area types, especially corridors.

Here's the algorithm how Mike explains it:

```
In this algorithm a "feature" is taken to mean any kind of map component
e.g. large room, small room, corridor, circular arena, vault etc.

1. Fill the whole map with solid earth
2. Dig out a single room in the centre of the map
3. Pick a wall of any room
4. Decide upon a new feature to build
5. See if there is room to add the new feature through the chosen wall
6. If yes, continue. If no, go back to step 3
7. Add the feature through the chosen wall
8. Go back to step 3, until the dungeon is complete
```

Marks

I do step 3 of the algorithm a bit differently. After adding a new area to the map, I add new area spawning points to all sides of the area. In reality, the marks are added to a list, but for debugging and visualization purposes I also set the marks to a debug layer of the map. When deciding where to build next, the generator randomly picks a point from the list.

I call these connection points ”marks”.

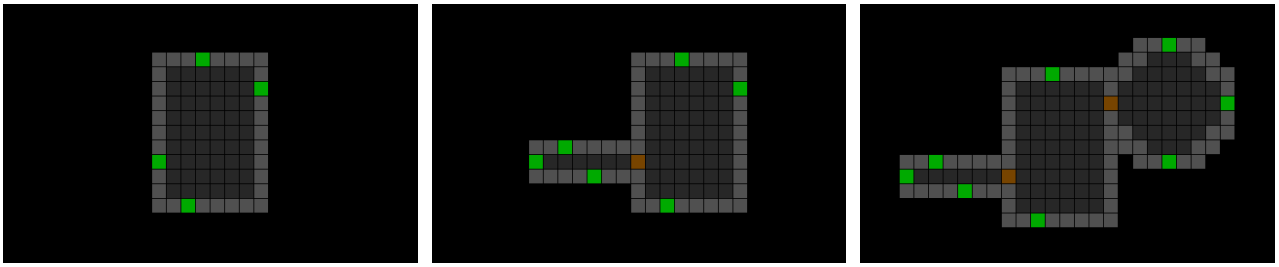
For example, when you add a room, add a connection point to all sides of the area excluding the source point side (so, a total of 4 points if it's the first area of the dungeon, 3 otherwise).

The exact mark location is randomly chosen. I won’t add marks to points too close to map edges.

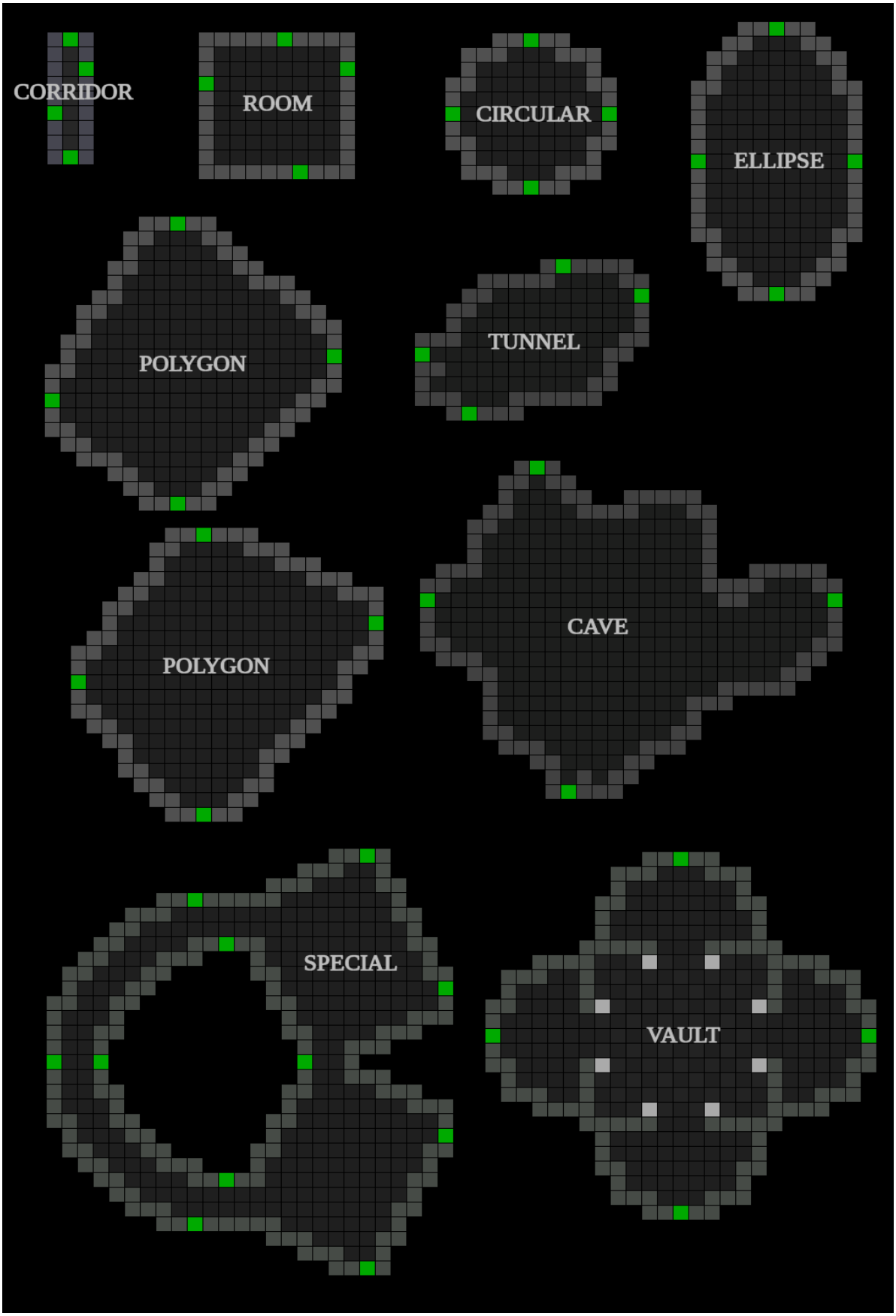
Marks for prefabs are set with an external tool, the Vault Editor.

Marks have these properties:

- X- and Y-coordinate
- Direction (1 - 4 = N, E, S, W)
- Master area type
- Force area type (optional, force the next area to be of certain type)
- Corridor end (set if this point is the dead end of a corridor)



Green blocks are marks. Brown blocks are doors.


CORRIDOR
ROOM
CIRCULAR
ELLIPSE
POLYGON
TUNNEL
POLYGON
CAVE
SPECIAL
VAULT

Area types

The dungeon generator has the following area types:

- Corridor
- Room
- Circular
- Ellipse
- Polygon
- Tunnel
- Cave
- Special
- Vault

For corridors, there’s a chance that side corridors are automatically added making it a crossroads.

Polygons are shaped from connecting randomly placed points.

Caves are natural formations. Tunnels are also natural formations, but more longish, with a chance that a cave is added at the end point. Tunnels and caves are created with the Drunkard Walk algorithm.

Specials are prefabs, mostly hallways, crossroads, dividers, etc., mostly not used for content. The curved corridors you seen in the game maps are specials.

Vaults are prefabs, thronerooms etc. for special content.

Specials and vaults are created with an external tool, the Vault Editor. Prefabs are rotated towards the direction they will be drawn at. Vault Editor precalculates the prefab dimensions so that prefabs that would go over the map edges can be skipped when randomly picking from the alternatives. Prefabs are grouped into ”rectangular”, ”curvy”, ”prison”, etc. groups. Each level entry has a list of prefab groups that the level uses.

Each level entry has a weighted list of chances per area type. The sequential value is a percentual chance that a same type of area can be spawned from an area.

```
area_chances.......... co = 014%,  ro = 033%,  ci = 006%,  el = 005%,  po = 006%,  tu = 005%,  ca = 008%,  sp = 017%,  va = 004%
|
area_sequential....... co = 050%,  ro = 020%,  ci = 010%,  el = 010%,  po = 020%,  tu = 050%,  ca = 050%,  sp = 100%,  va = 000%
area_max_amounts...... co =    0,  ro =    0,  ci =    5,  el =    5,  po =    5,  tu =    4,  ca =    4,  sp =   15,  va =    0
```

Area list

Each added area is stored to an area list with its area type, coordinates, size, number of exits, etc. info. This list is then used when choosing areas for content and many other things.

Dungeon post-processing

Several methods are used to post-process the map:

- Corridor dead ends are terminated by creating areas at the end of the corridors, or the corridors are extended if they can breach another area. If a corridor with a single entry point can not be terminated with these methods, that entry point is turned into a secret door, and later ”treasure corridor” content is applied to the area.
- Wider corridors are extended if there’s an area that can be breached.
- Corridors between areas are added to connect them.
- Chance of areas being merged by removing shared walls.
- Small areas (small rooms and small prefabs) are added at marks in cave- and vault-areas.
- If there are common walls between areas, floor / doors / secret doors are added to them.
- Some doors are changed to secret doors. Secret doors are not created at the end of corridors as they seem too obvious.

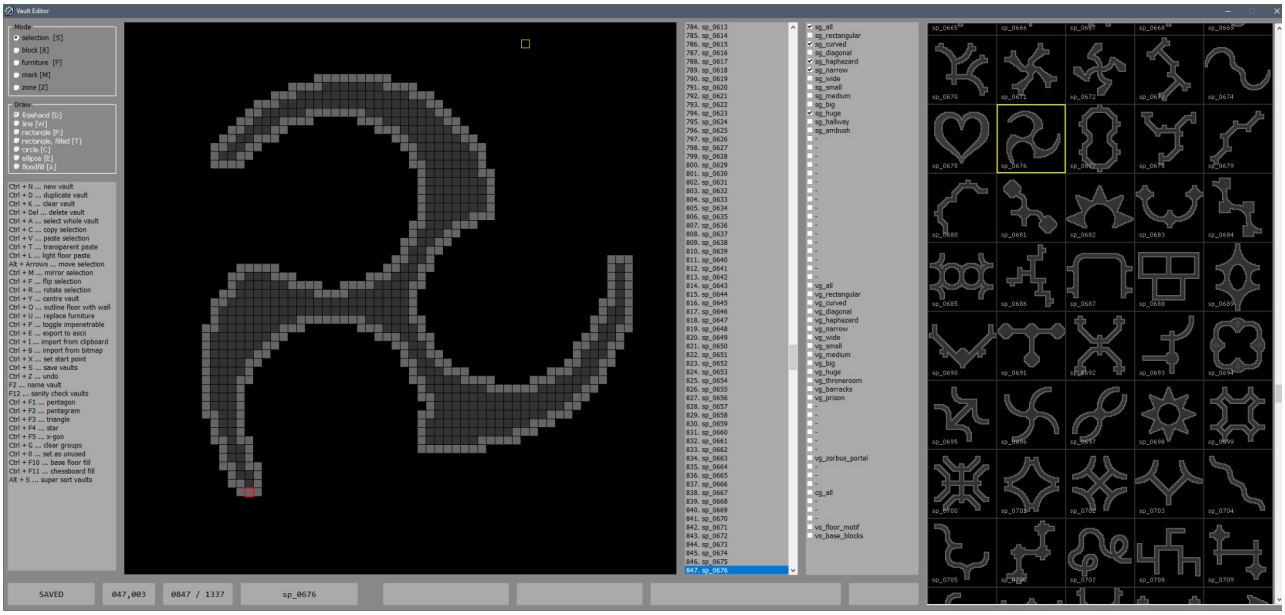
After post-processing, the number of exits (no-door exits, doors, locked doors, secret doors) in each area is calculated and stored in the area list. This information is later used when adding content to the areas.

Post-processed map. Red tiles are walls turned into floor. Yellow tiles are secret doors.

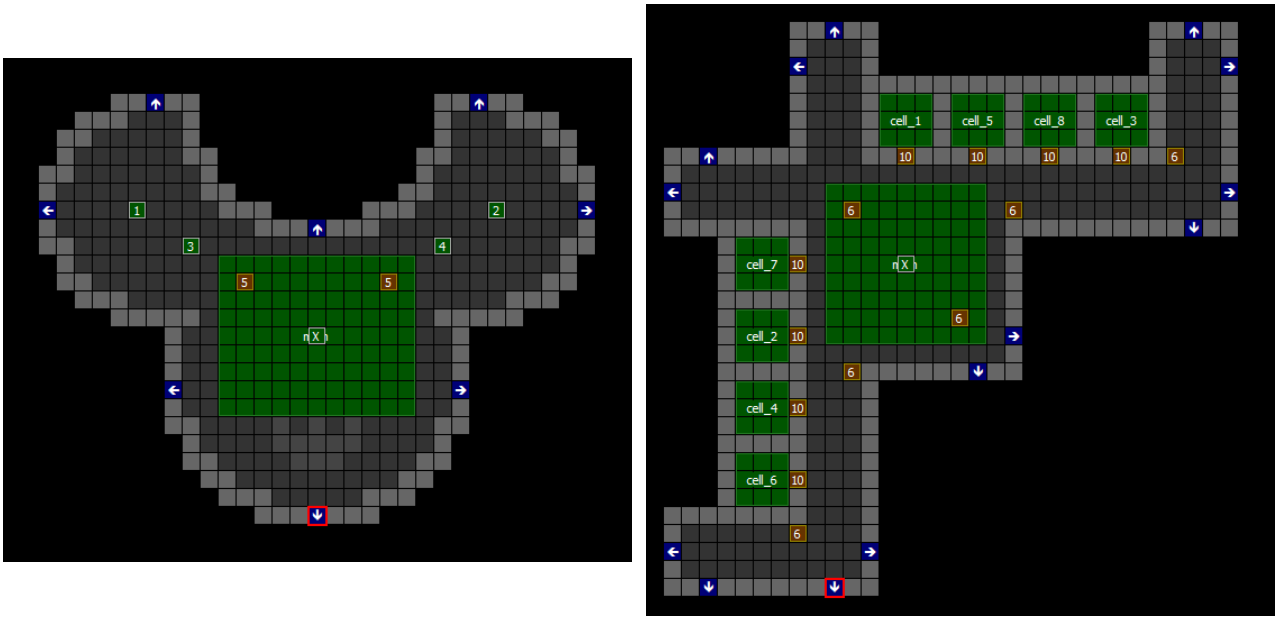
Vault Editor

Prefabs are created with an external tool, the Vault Editor.

If you are a game developer, you might be interested in zorbus_vaults.zip. It's a collection of all the prefab areas used in the generator, in plain ASCII-format, free to use (CC0 Creative Commons License).



The Vault Editor has basic drawing and rotating capabilities. Bitmap images and single truetype font glyphs can be imported to be used as a base for drawing.



Prefabs. Arrows on blue background are marks. Red frame marks the starting point. Numbers with brown background are furniture. Grids with green background are spots and zones, where content stuff can be placed.

Adding content to dungeon areas

Contents are described in an external text file. Most contents are tagged with one or several theme tags, for example *th_basic_01* or *th_rare_01*.

In a level entry you could have:

```
themes.............. th_basic_01 = 2d4+5,   th_rare_01 = 1d10-9
```

Which would set 2d4+5 contents onto the dungeon level, chosen from all contents with *th_basic_01* tag, and 1d10-9 contents chosen from all contents with *th_rare_01* tag.

You can also force contents:

```
force_contents........ co_ambush,   co_prison
```

Content descriptions

Content descriptions started as simple lists of stuff I wanted to be created into an area, but later on more keywords and control were added.

Contents are almost never tied to certain area type or any single prefab. Instead, the content description has a list of wanted area types with requirements.

```
content_id............ co_gnoll_barracks
theme_tags............ th_basic_03,   th_basic_04,   th_basic_05_06
|
area.................. room,   min_size = 90,   min_exits = 1,   max_exits = 4
area.................. circular,   min_size = 133,   min_exits = 1,   max_exits = 4
area.................. ellipse,   min_size = 133,   min_exits = 1,   max_exits = 4
area.................. polygon,   min_size = 150,   min_exits = 1,   max_exits = 4
area.................. vault,   min_size = 150,   min_exits = 1,   max_exits = 4
```

This content wants a room area type of at least size 90, with a minimum of 1 exit, and maximum of 4 exits. The content generator makes a list of such areas that haven’t yet had content added to them, randomly selects one, then generates the wanted stuff into it. If no such area exists, the generator then tries the next area type in the list, circular areas in this case.

You can give a percentual chance that the order of the wanted areas list is scrambled.

```
scramble_areas........ 30
```

These control max amounts of this content per level / per run. (0 is unlimited)

```
max_per_level......... 2
max_per_game.......... 0
```

With *enchantment_boost* you can boost the chance that the armor and weapons created in this content are enchanted (plusses or branded).

```
enchantment_boost..... 5
```

Choosing an area for the content

In the map generation phase the generator added each area to an area list with its area type, coordinates, size, exits, and so on. This is where the area list gets very useful.

The content generator makes a list of all areas on the level that fill the requirements of the content, then one area from the list is randomly picked. After the content is added, the area is marked as used.

Adding content stuff

Stuff is things that are created into the selected area. Stuff can be furniture, floor symbols, beings, items, loot (selected from a group of items with the declared loot-tag), traps, hazards (selected from a group of traps with the declared hazard-tag), decorative splashes (bones, blood, litter, etc.).

Stuff placement can be set (center, top wall, etc.), otherwise it's random.

```
stuff.............. fu_table,   placement = center
stuff.............. fu_chair,   placement = center,   chance = 50

stuff.............. lo_bones,   placement = full_area,   amount = 1d3+1
```

For prefabs, you can set a wanted zone, otherwise the main zone is used. If a prefab is selected as the target area, the code that checks if an area is suitable for this content also checks that the needed zones exist in the prefab. For example, if the content needs a prefab with prison cells, then only those kind of areas areas are considered suitable.

```
stuff.............. be_jermlaine_ambusher,   zone = vz_ambush_1
stuff.............. be_rat_giant_ambusher,   zone = vz_ambush_2

stuff.............. tag_prisoners,   zone = vz_cell_1,   chance = 80
stuff.............. tag_prisoners,   zone = vz_cell_2,   chance = 80
```

Selection keywords form a list from which one is randomly picked and created.

```
stuff.............. sy_fangs_5x5,   selection_start
stuff.............. sy_pentacle_5x5,   selection
stuff.............. sy_pentagram_5x5,   selection
stuff.............. sy_sun_5x5,   selection
stuff.............. sy_sunrise_5x5,   selection_end
```

Stuff can be also be listed in level entries. A random area is picked from all areas on the level.

```
stuff.............. re_01,   amount = 20+2d5,   min_exits = 1,   no_beings
stuff.............. ha_traps_1,   amount = 1d3,   clear_area_index
stuff.............. it_bones_humanoid,   placement = full_area,   amount = 5+3d5
```

Content example: Gnoll barracks

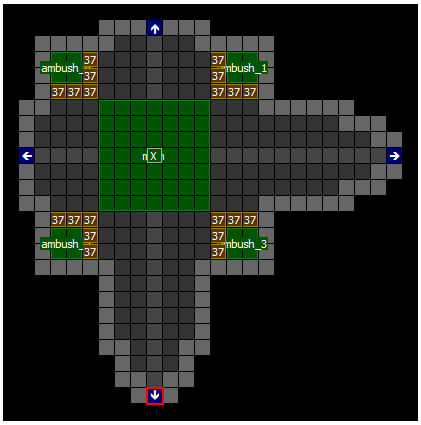
```
content_id..... co_gnoll_barracks
theme_tags..... th_basic_03,   th_basic_04,   th_basic_05_06
max_per_level..... 1
max_per_game..... 1
properties..... beings_to_campers
note..... Gnoll barracks
|
scan..... from_end
scramble_areas..... 100
area..... room,   min_size = 90,   min_exits = 1,   max_exits = 4
area..... circular,   min_size = 133,   min_exits = 1,   max_exits = 4
area..... ellipse,   min_size = 133,   min_exits = 1,   max_exits = 4
area..... polygon,   min_size = 150,   min_exits = 1,   max_exits = 4
|
light..... 100
dim_lights..... 0
chests..... lock = 20
floor_pattern..... 0
enchantment_boost..... 0
|
stuff..... sy_crushed_5x5
|
stuff..... fu_table,   placement = center
stuff..... fu_chair,   placement = center
stuff..... fu_chair,   placement = center,   chance = 50
stuff..... fu_bed
stuff..... fu_bed
stuff..... fu_bed,   chance = 50
stuff..... fu_chest,   chance = 90
stuff..... fu_barrel,   chance = 50
stuff..... fu_barrel,   chance = 50
stuff..... lo_armor_medium,   chance = 40,   amount = 1d2
stuff..... lo_common,   amount = 1d3+3
stuff..... lo_potions,   amount = 1d2,   chance = 30
stuff..... lo_potions,   amount = 1d2,   put_on_floor = 100
stuff..... lo_trapkits,   chance = (15 | per_dungeon_level: 1)
stuff..... lo_utility_potions,   chance = 25
stuff..... it_coin,   amount = 20+2d20
stuff..... sp_litter,   placement = full_area,   amount = 1d5+3
|
stuff..... be_gnoll_leader,   amount = (0 | per_dungeon_level: 1/4)
stuff..... be_gnoll,   amount = (1d2 | per_dungeon_level: 1/2)
|
stuff..... tag_traps,   amount = 1d3+1,   chance = 30,   trap_containers
|
sound..... stinger
```

Content example: Hidden dead end treasure corridor

```
content_id............ co_treasure_corridor
theme_tags............ th_secret_areas_01,   th_secret_areas_02,   th_secret_areas_03
max_per_level......... 0
max_per_game.......... 0
properties............ beings_to_campers,   no_blink_into
|
scan.................. from_end
scramble_areas........ 0
area.................. corridor,   max_exits = 0,   min_size = 9
|
light................. 0
dim_lights............ 100
chests................ -
floor_pattern......... 0
enchantment_boost..... 5
|
stuff................. fu_chest
stuff................. lo_armor_heavy,   chance = (10 | per_dungeon_level: 3)
stuff................. lo_armor_medium,   if_previous_failed
stuff................. lo_common,   amount = 3+1d5,   chance = 60
stuff................. it_coin,   amount = (5+3d10 | per_dungeon_level: 10)
stuff................. lo_potions,   chance = (20 | per_dungeon_level: 1)
stuff................. lo_ammunition,   chance = 60
stuff................. lo_rare_ammunition,   chance = (20 | per_dungeon_level: 2)
stuff................. lo_magic_devices,   chance = (5 | per_dungeon_level: 3)
stuff................. lo_rare_melee,   chance = (0 | per_dungeon_level: 1)
stuff................. lo_rare_ranged,   chance = (0 | per_dungeon_level: 1)
stuff................. lo_scrolls,   chance = (10 | per_dungeon_level: 2)
stuff................. lo_books,   chance = 10
stuff................. it_coin,   amount = 30+2d30
stuff................. tag_traps,   amount = 1d3-1
stuff................. tag_traps,   chance = 80,   trap_containers
|
sound................. stinger
```

Note how the area requirement is a corridor with max 0 exits, meaning that this content is only created in corridors that are behind a secret door without further exits. This is a nice way to add something to dead end corridors that the dungeon post-processing was unable to terminate.

Content example: Ambush

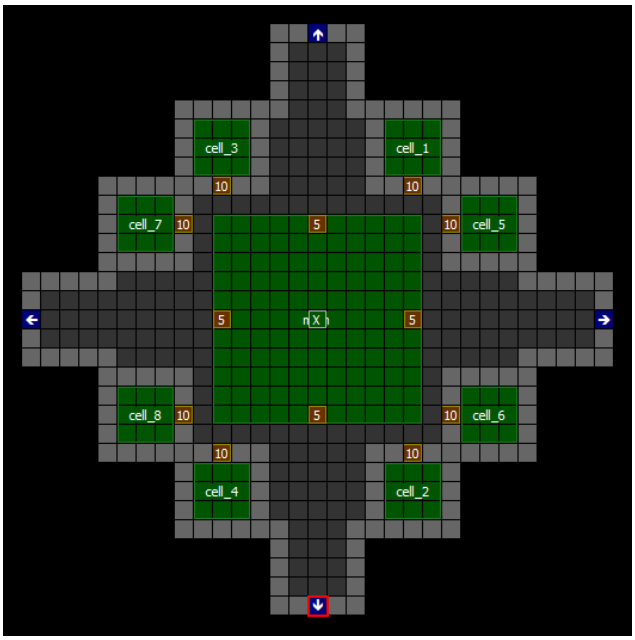



```
content_id..... co_ambush
theme_tags..... th_specials_01,   th_specials_02
max_per_level.... 1
max_per_game..... 1
properties..... beings_to_campers,   beings_always_awake
|
scan..... from_end
scramble_areas..... 0
area..... special
|
light..... 0
dim_lights..... 0
dim_lights..... 0
chests..... -
floor_pattern..... 0
enchantment_boost..... 0
|
stuff..... tr_ambush,   amount = 8+1d4,   trap_difficulty = 100
stuff..... ls = le_kobold,   le = le_giant_2,   tr_slime,    amount = 1d2+1
|
stuff..... be_jermlaine_ambusher,   zone = vz_ambush_1
stuff..... be_rat_giant_ambusher,   zone = vz_ambush_1
stuff..... be_jermlaine_ambusher,   zone = vz_ambush_2
stuff..... be_rat_giant_ambusher,   zone = vz_ambush_2,   chance = 30
stuff..... be_jermlaine_ambusher,   zone = vz_ambush_3,   chance = 80
stuff..... be_rat_giant_ambusher,   zone = vz_ambush_3,   chance = 60
stuff..... be_jermlaine_ambusher,   zone = vz_ambush_4,   chance = 70
stuff..... be_rat_giant_ambusher,   zone = vz_ambush_4,   chance = 50
|
stuff..... sp_litter,   placement = full_area,   amount = 1d5+3
stuff..... sp_remains_pool,   placement = full_area,   amount = 2d4+5
stuff..... lo_bones,   placement = full_area,   amount = 1d3+1
```

The area layout is a prefab, created with the Vault Editor. The green zone in the middle is where the ambush traps are placed. Walking on one of these traps triggers the ambush, meaning that the walls marked with brown background and number 37 will vanish, and the rest of the ambush traps in the area are removed. Beings waiting in ambush are placed in special ambush zones.

Note that this content is not tied to any single prefab, but any prefab that fills the requirements, in this case the four ambush zones where the ambushers will be placed. The game has currently over 60 different prefab maps for ambushes, and this content can be placed in any of them.

Content example: Prison




```
content_id..... co_prison
theme_tags..... th_specials_04,   th_specials_05_06
max_per_level..... 1
max_per_game..... 1
properties..... beings_to_campers
note..... Prison
|
scan..... from_end
scramble_areas..... 0
area..... vault,   min_exits = 0
|
light..... 100
dim_lights..... 0
chests..... -
floor_pattern..... 50
enchantment_boost..... 0
|
stuff..... fu_table,   placement = center
stuff..... fu_chair,   placement = center
stuff..... fu_chair,   placement = center,   chance = 60
stuff..... fu_chest,   chance = 70
|
stuff..... lo_common,   amount = 1d4
stuff..... lo_trapkits,   chance = (15 | per_dungeon_level: 1)
|
stuff..... lo_bones,   placement = full_area,   amount = 1d4+3,   put_on_floor = 100
stuff..... sp_blood_pool,   placement = full_area,  amount = 2d3+3
stuff..... sp_litter,   placement = full_area,  amount = 1d8+5
|
stuff..... be_human_bandit_leader,   chance = (40 | per_dungeon_level: 10)
stuff..... be_human_bandit,   amount = (1d2 | per_dungeon_level: 1/2)
|
stuff..... tag_prisoners,   zone = vz_cell_1,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_2,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_3,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_4,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_5,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_6,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_7,   chance = 80
stuff..... tag_prisoners,   zone = vz_cell_8,   chance = 80
```

Note that this content is not tied to any single prefab, but any prefab that fills the requirements, in this case the eight prison cell zones where the prisoners will be placed. The game has currently 30 different prefab maps for prisons, and this content can be placed in any of them.

Being emotions

The emotion system used in the game was inspired by an article, ”Need driven AI”, written by Björn Bergström, available at RogueBasin.

Beings have emotion- and fear-values against other beings set in the Beings-datafile. Being type is used as an identifier. Type can for example be race, alignment, faction, same race, all except same race, and so on.

```
being_id.............. be_githyanki_warrior
|
unidentified_name..... githyanki warrior
introduced_name....... githyanki warrior
|
type.................. humanoid,   githyanki,   male,   medium,   evil
|
emotions.............. same_race = fanatic_love,
                       mindflayer = fanatic_hate,
                       good = fanatic_hate,
                       player_and_companions = fanatic_hate
```

Certain factions and being types hate each other, for example demons and devils are always fighting each other in a neverending Blood War. This brings more life into the dungeon as not every creature is after the player character.

Some beings get gradually more hostile against other beings that stay on their birth area, or just stay too long near them.

```
being_id.............. be_bear_brown
|
unidentified_name..... brown bear
introduced_name....... brown bear
|
type.................. animal,   bear,   large,   neutral,
                       hostile_on_home_trespassing,   gets_hostile
|
emotions.............. bear = fanatic_love,
                       all_except_same_race = mild_hate
```

Being’s fear-value against other being gets increased when the being gets damaged, its master is killed, something kills a large amount of its friends in a single round, and so on. Depending on the being’s morale-value, it might flee, go berserk, or just ignore the fear. Most beings get a morale boost from superiority.

The emotion- and fear-tables also store the target being’s last seen location. A fleeing being avoids that location for a time.

Creatures with positive emotions against each other trade hate-emotions (and at the same time, the last seen locations of hate targets). Occasionally the player will see and hear these emotion exchanges in speech bubbles and sound effects.

Goblin A might flee from the player, and when running away it might meet its friend, goblin B, who then gets the hate-emotion from goblin A with player’s last seen location. If goblin A would have met more friends, its morale could have been restored so much that it would return at the player’s last seen location with its friends.

Interactive dialogue

The game has interactive dialogue in that sense that beings react to things happening to them and to things happening around them. Dialogue is shown with speech bubbles.

Dialogue is grouped to various situations, for example:

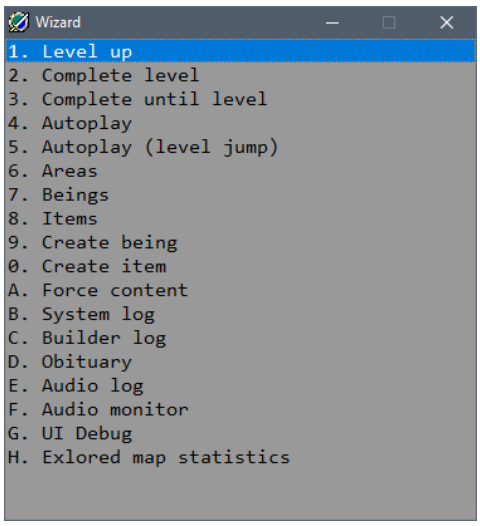
- Friendly introduction
- Hostile introduction
- Gets angry
- Kills another being
- Friend is killed
- Flees
- Player is hiding in same area
- Tells friend how much he hates the player
- Friend comments info about player
- Follower springs trap
- Follower sees master spring a trap
- Idle chatter

When a being goes into some state, or sees something happen, the dialogue usually isn’t immediate, but instead a situation counter is set. The counter is decreased each game round, and the being tries to say the dialogue while the counter is alive. If the counter goes to zero, the need for dialogue is forgotten. There are priorities and cooldowns for dialogue types, some are said only once, and since there are limited amount of visible speech bubbles on the screen, there might not be space for a bubble.

Debugging the game

The developer version of the game places the game window in the upper left corner of the screen. Below the game window is a full sized debug window with a map of the level and various logs. The map is updated as beings move around.

With the various hotkeys you can jump to wanted area number or level, save and load the game state, and so on. The developer version jumps straight into the game without any menus, so loading the previous state is fast.



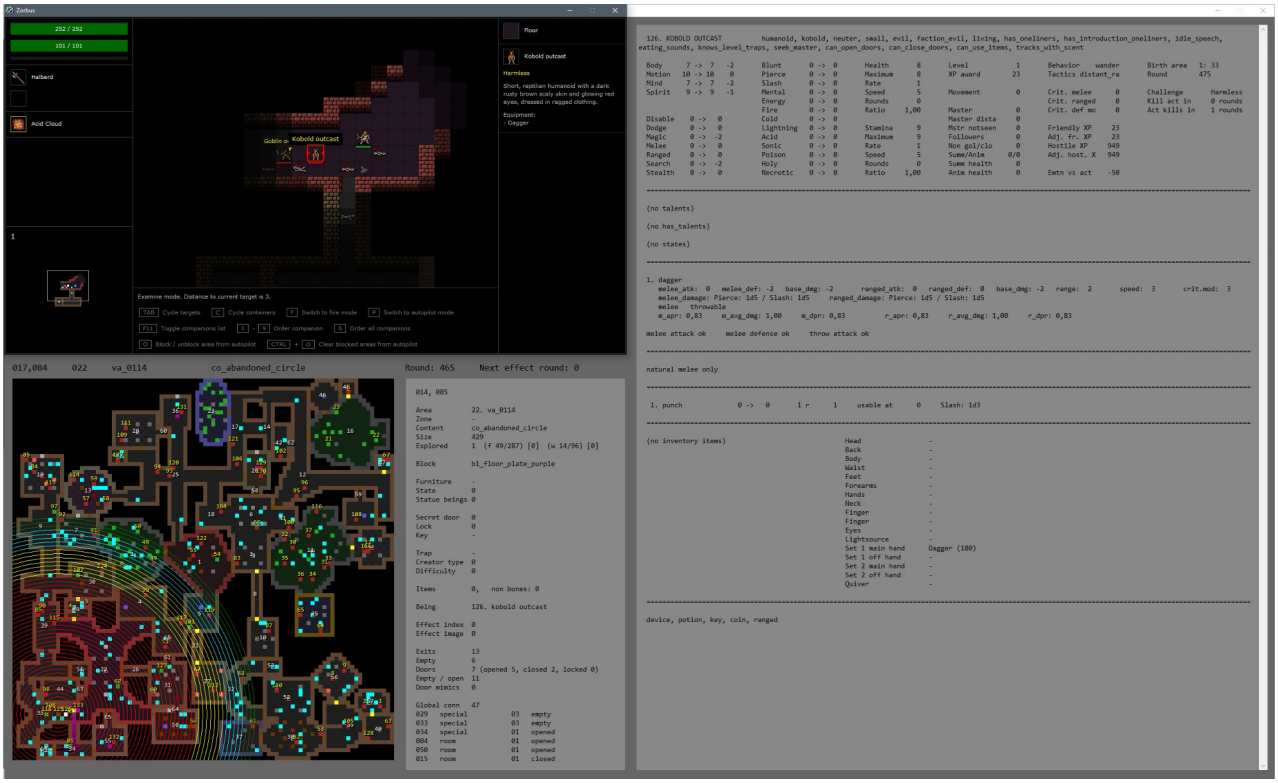
Pressing a hotkey brings up a wizard-menu.

You can level up the character to wanted experience level.

”Complete level” sets the player character to AI control and explores the whole dungeon level, then returns control to player.

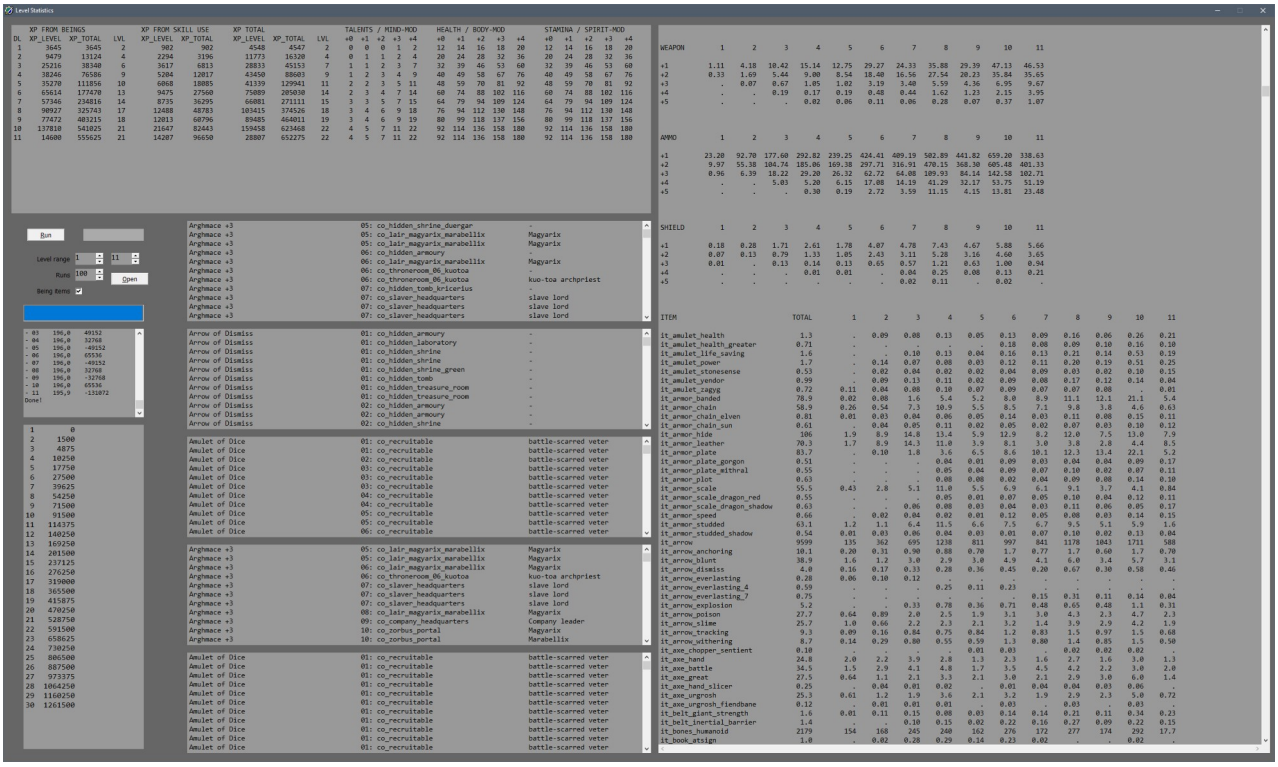
”Autoplay” autoplays the game in AI control, and starts again from the start when the last dungeon level is explored. Very, very useful for debugging and stress testing the game.

The wizard has various ways to list stuff from the current level, or to create new stuff.



Examine mode shows information of the map tile and being under the cursor. There's a second page of being information that lists the being's emotion- and fear-values.

In this mode you can press S to select a being, then move the cursor to a wanted location, and place the being there by pressing M. With the previously shown wizard-menu you can create beings and items. This is very useful for creating a specific debug scenario. Game state can be saved with Ctrl+S, and loaded with Ctrl+L.



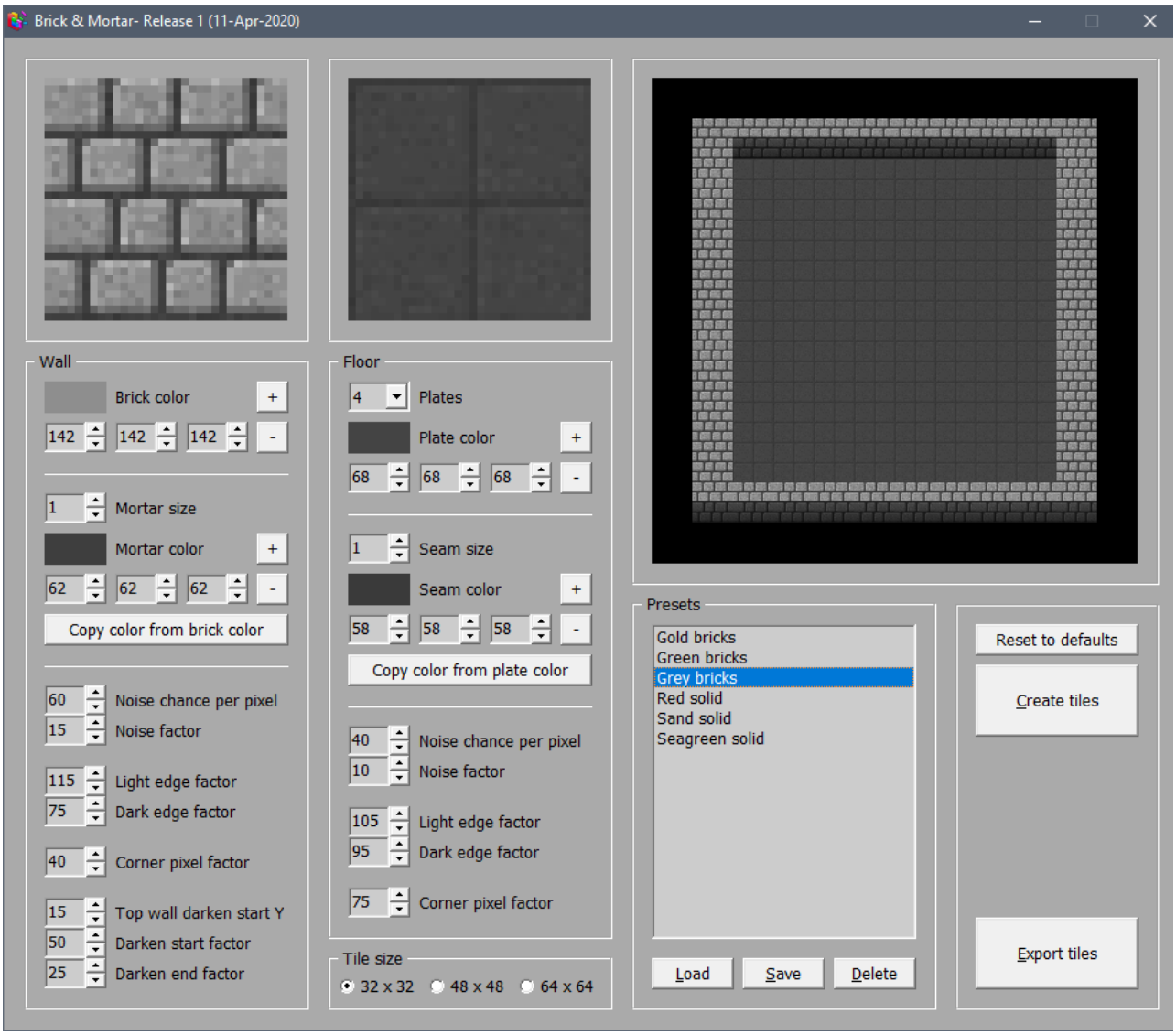
Level Statistics tool creates dungeon levels for wanted amount of runs, and calculates the average amount of created content, beings, items, etc., then listing the total amount and per level amounts.

Average experience per level is also calculated, and from that, the average character level per dungeon level.

Very useful for getting statistics on various things, adjusting numbers, and also for stress testing the dungeon generator for possible errors and memory leaks.

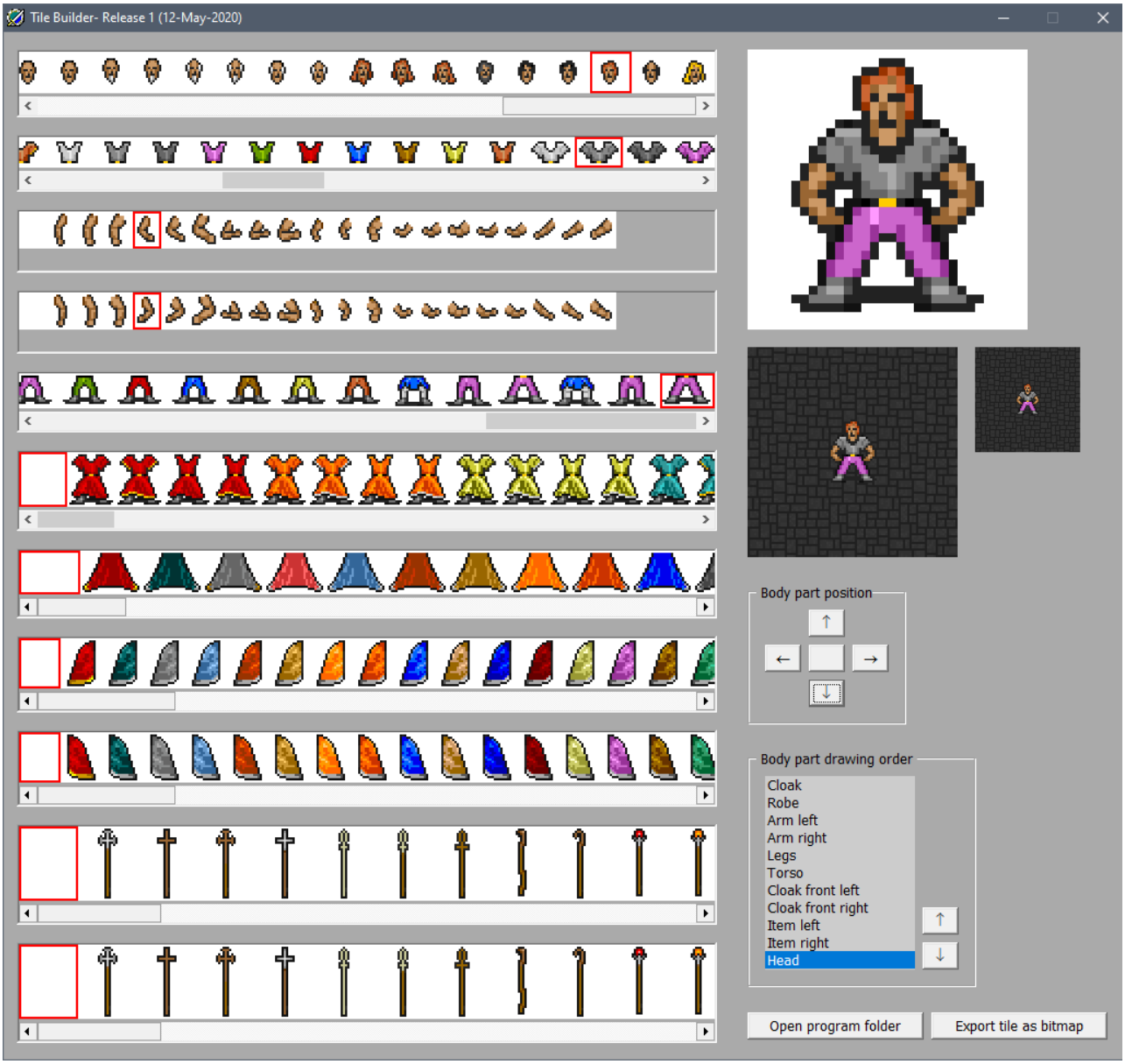
Miscellaneous external tools

Several external tools were created during development.



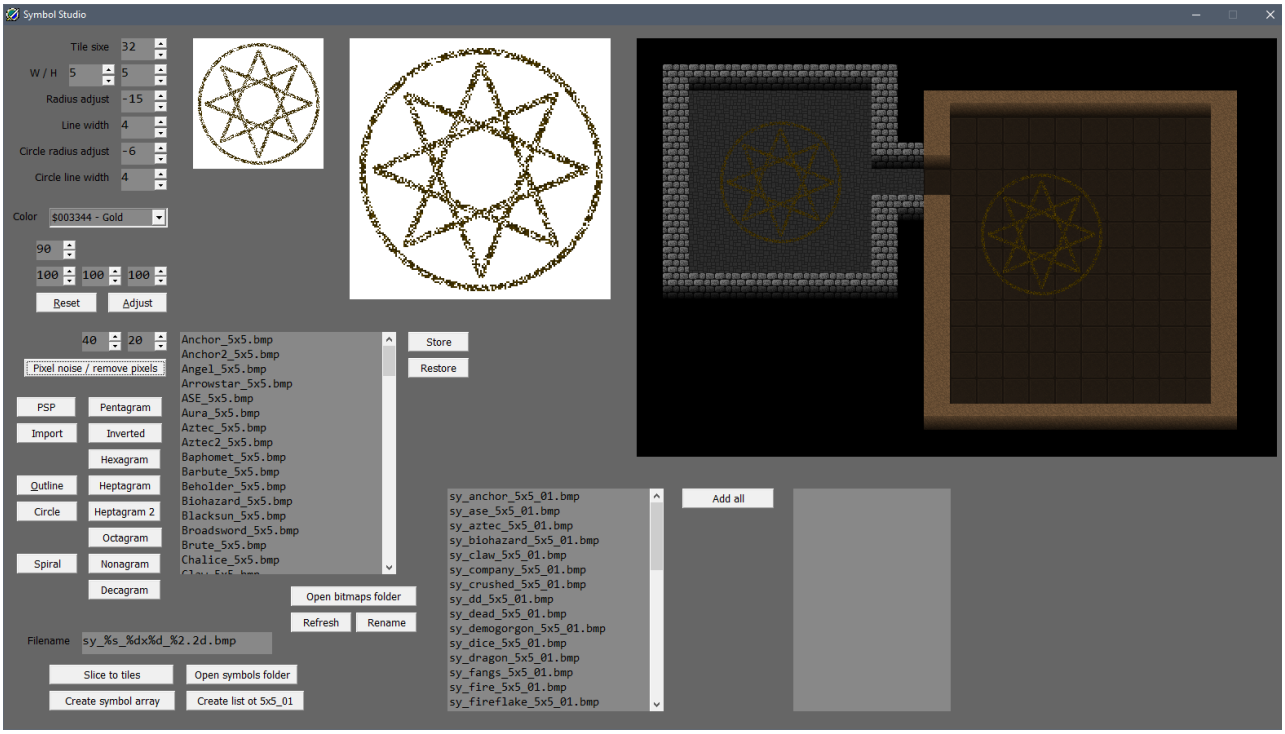
Brick & Mortar is a tool for creating wall and floor tiles for games with tile graphics.

Download [bm.zip](bm.zip).

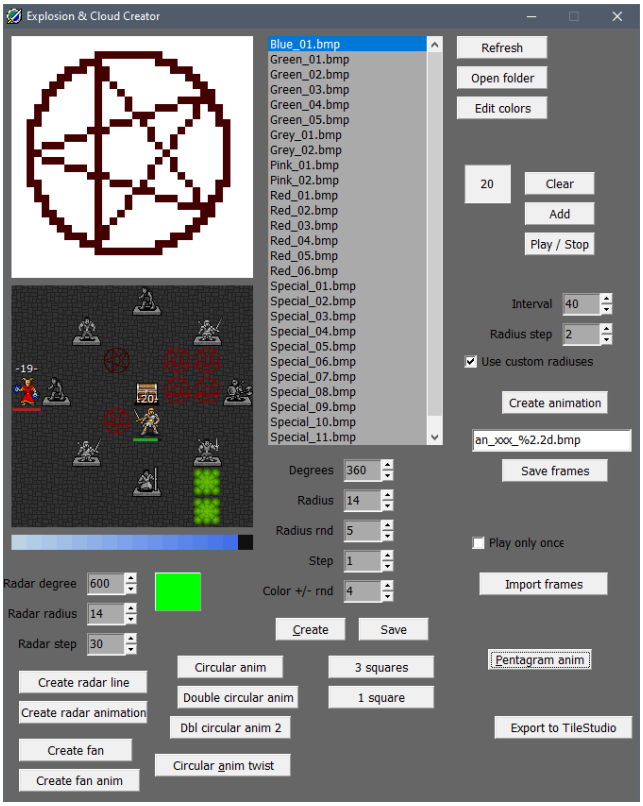


Tile Builder is a tool for building tiles from body parts drawn by David E. Gervais. The body part images are taken from the work files of his Angband tileset.

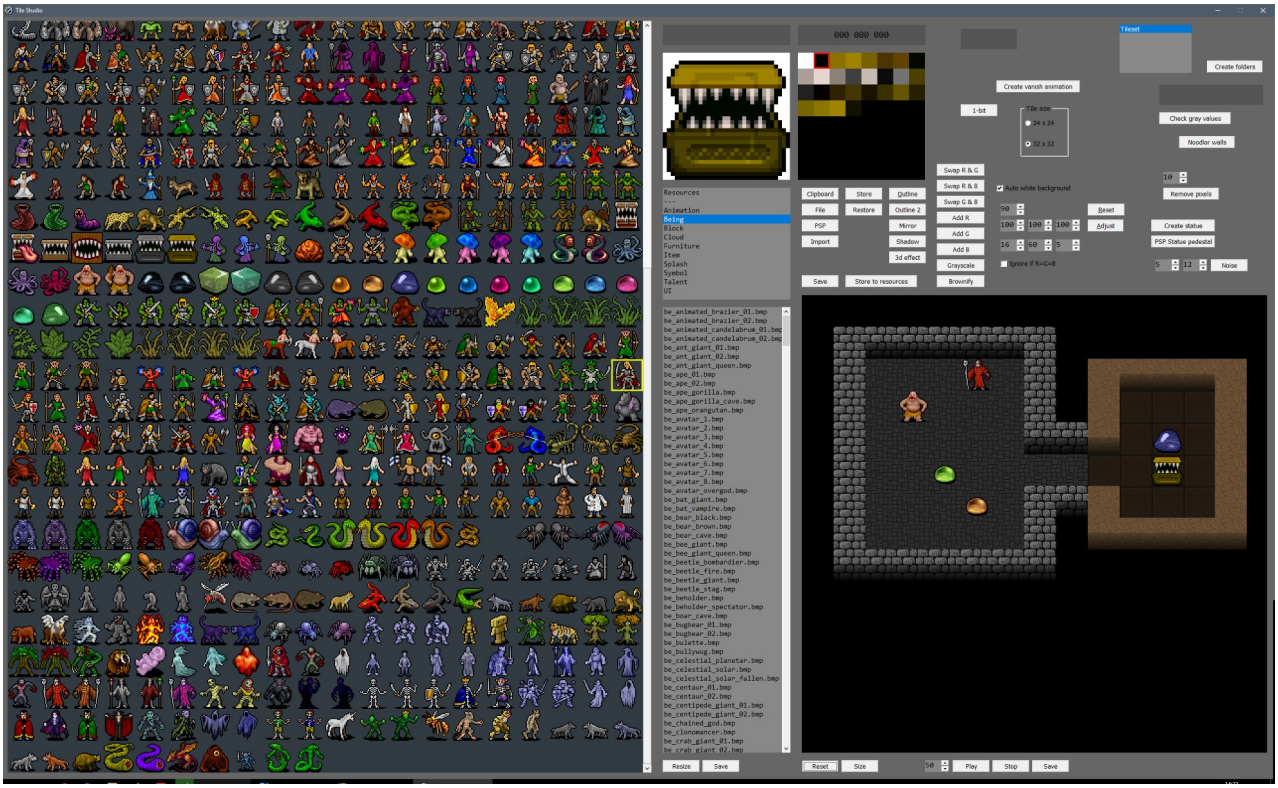
Download tb.zip.



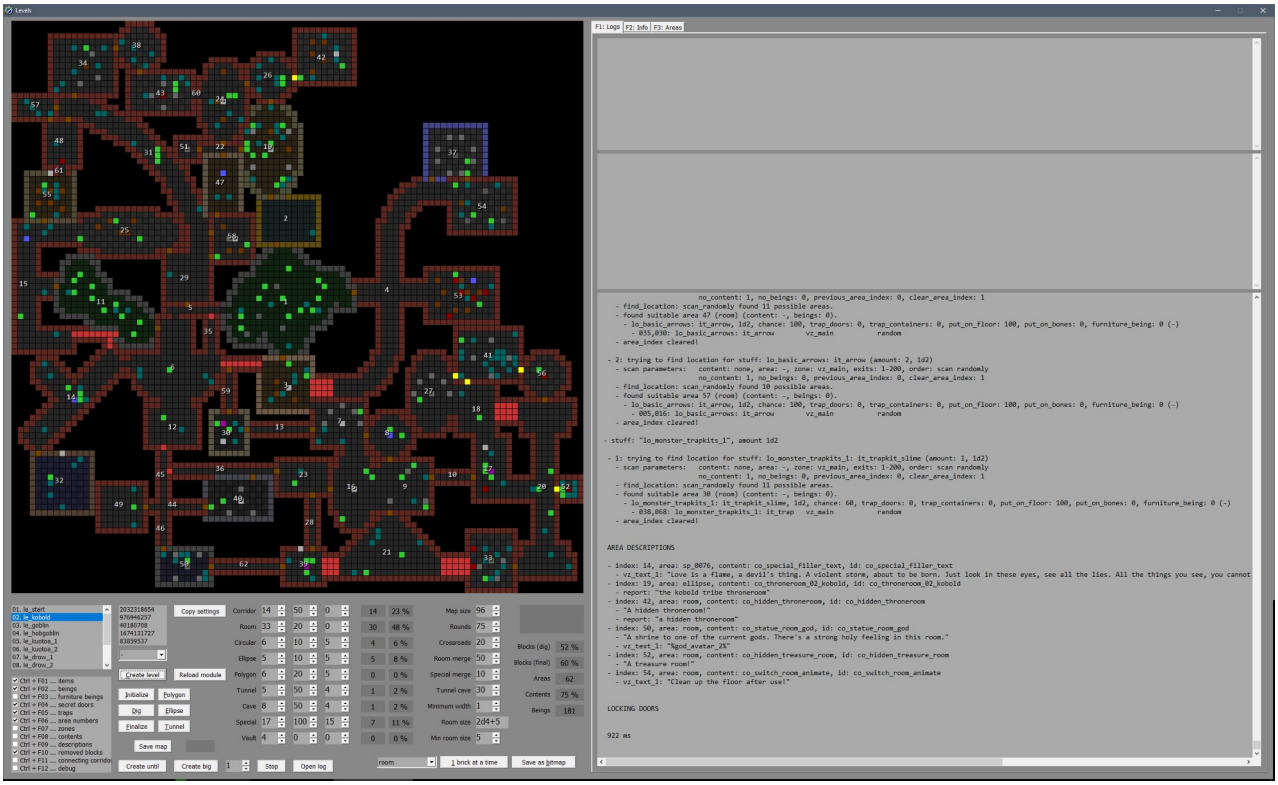
Symbol Studio for creating floor symbols, both procedurally and from bitmap images.



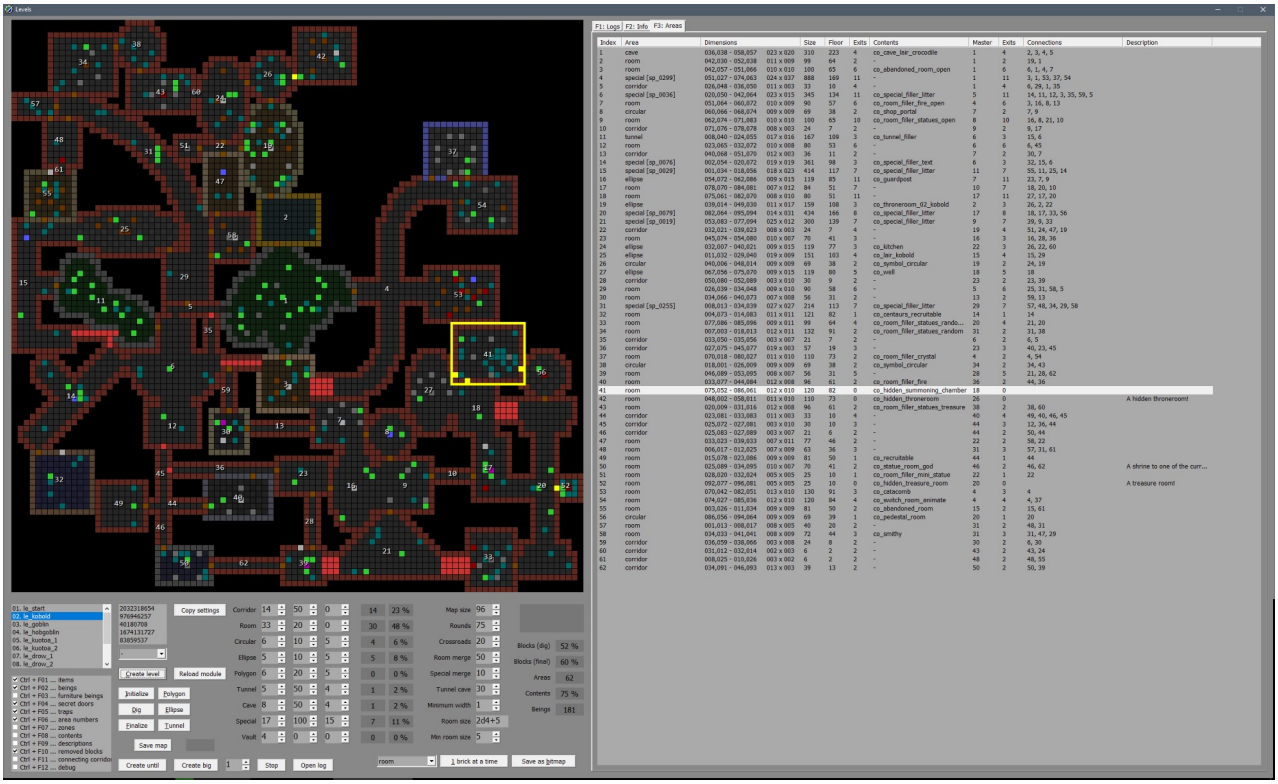
Explosion & Cloud Creator for procedurally creating various animations, mostly for spell effects. The rotating pentagram and summoning circle animations were created with this.

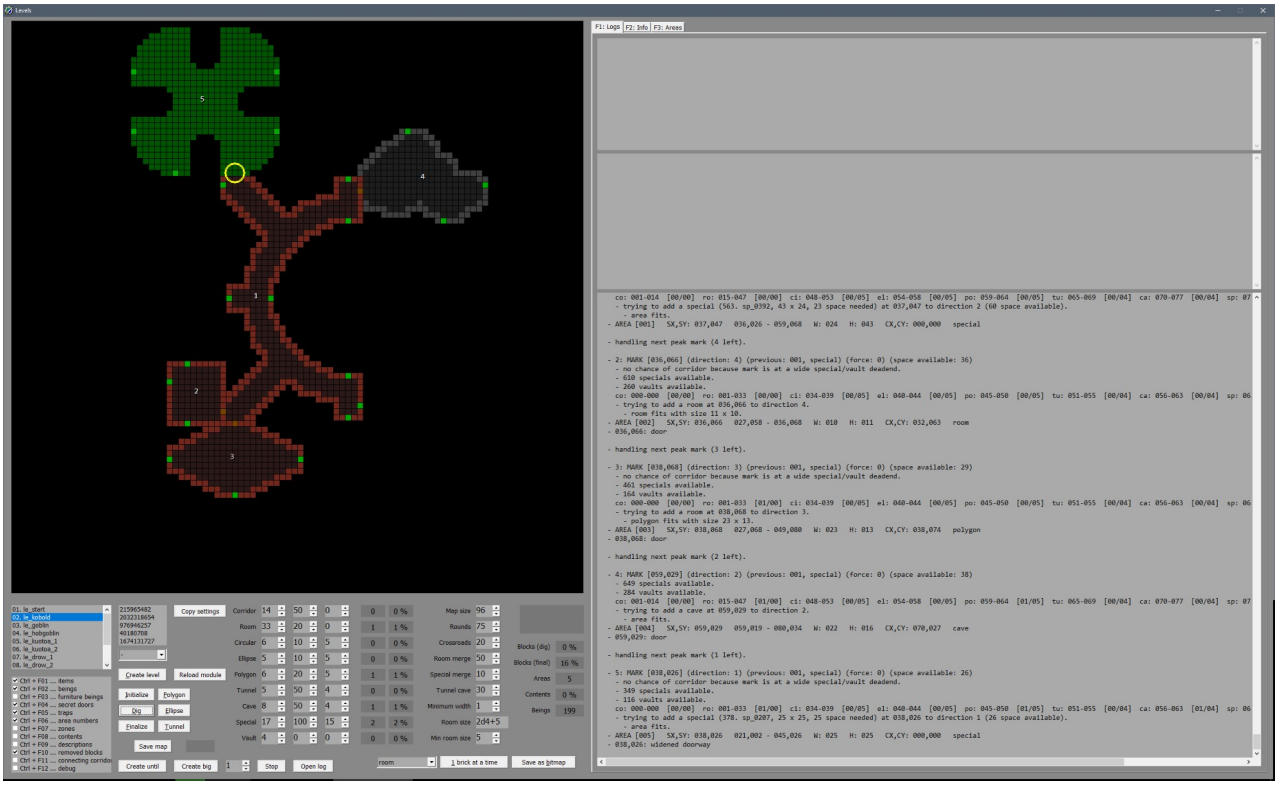


Tile Studio for editing and previewing the tiles. Can do some basic image manipulation, palette swapping, and so on.

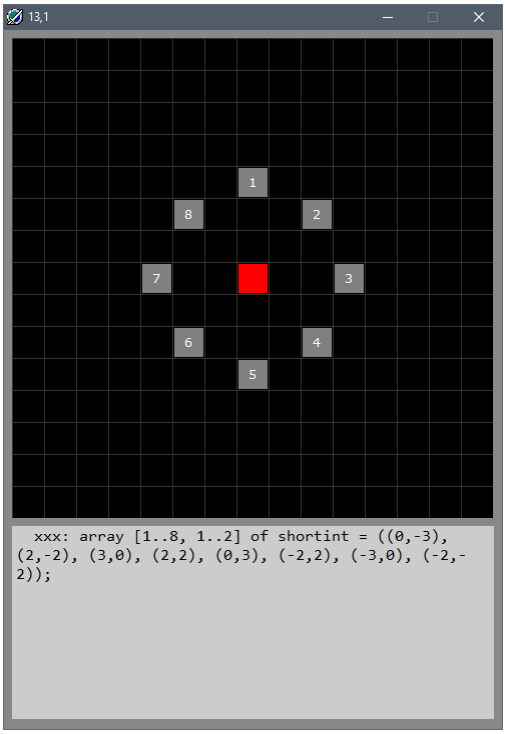


Level Tester has been invaluable in debugging the dungeon generator. Seeds can be used to reproduce problematic levels.

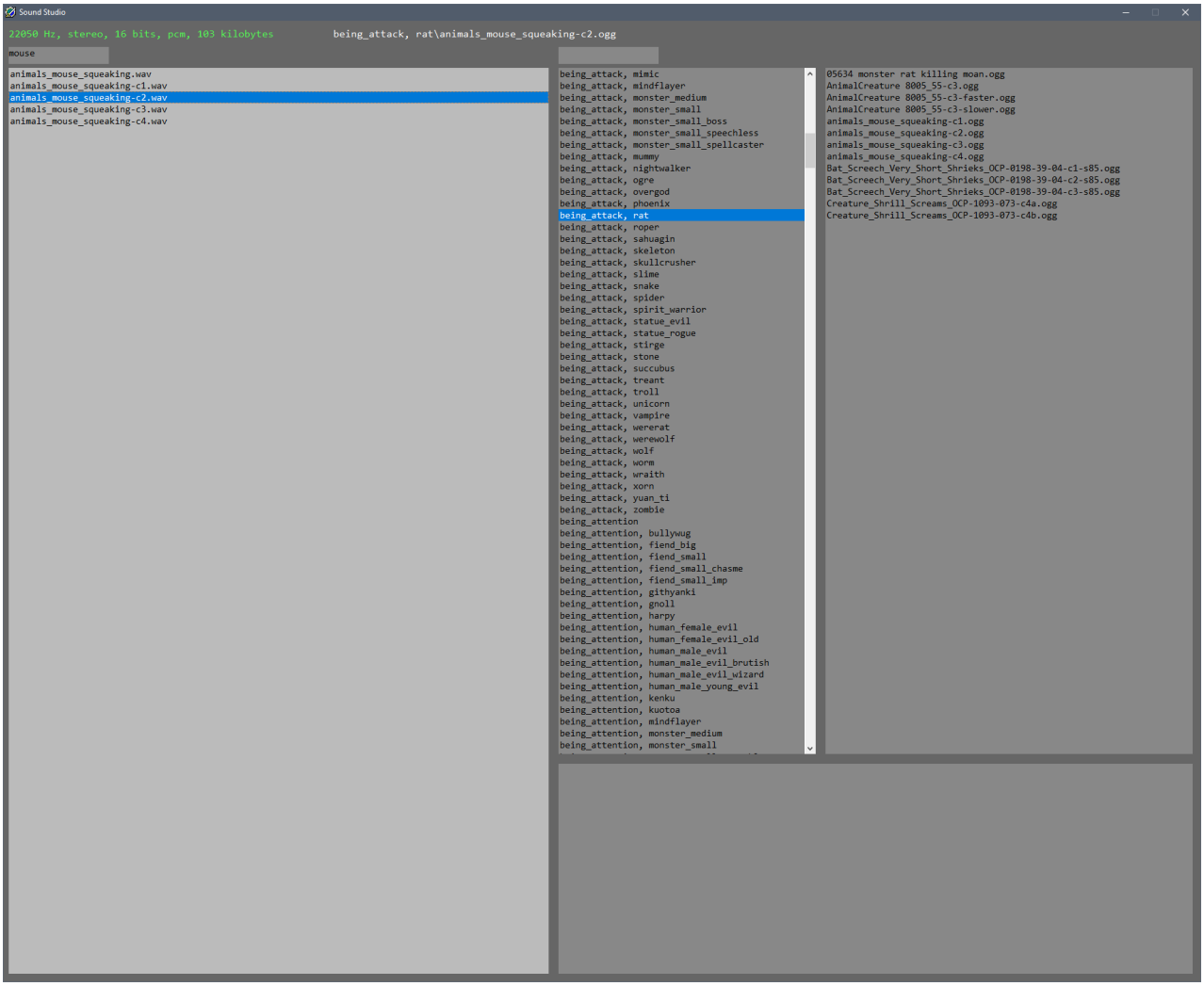




Level Tester in step-mode, adding one area at a time.



The game uses a lot of offset coordinate tables to check things from the map, and to place things on it. I needed something that could easily output the coordinates of series of offsets, so I created this offset tool where you can click the wanted grids, and it will create the coordinate array code.



The game has over 4000 sound effects, and to get there I went through a ridiculous amount of sound effects. Windows file explorer was way too clumsy to handle tens of thousands of resource files, thus Sound Studio was born. Yeah, it's not pretty, but it gets the job done.

This tool can be operated fully with the keyboard, you can press TAB to switch between the lists (resources / target folder / target folder contents), and use hotkeys to play, open, and convert files. There are hotkeys and keyword fields for filtering the resource files and target folders.

Pressing F1 executes command line utility FFmpeg to convert the selected wav-file from the leftmost list to an ogg-file, placing it in the target folder (middle list), of which contents are listed in the rightmost list.

Pressing F3 / F4 opens the wav-file in Goldwave / Audacity for editing.

I didn't want the game to have thousands of files in its release version, so Ctrl+S combines all the sound files to a giant, single datafile.