Spelunky's prcedural level generation system

Eduard Zalyaev
DS2-2016
Innopolis University
Innopolis, Russia
e.zalyaev@innopolis.ru

I. Introduction

Spelunky is an adventure game made by Derek Yu that thematically comes from old stories about adventurers that explore ancient ruins, searching ancient ruins of long forgotten civilizations for treasure and glory. The most famous example of such an adventurer is Indiana Jones. And so you take the same role. And you will die a lot. Is it supposed to be fun? What makes it so special?

II. Inspirations

Most of the Spelunky's mechanics take inspiration from roguelike games - dungeon crawlers with random level generation, permadeath and a lot of variation. Although games of this genre are highly replayable they lack in visual appeal and interface accessibility for casual players. This shouldn't a surprise when all graphics are rendered as ASCII symbols. There are platformers on the other hand with simple controls that also require precision ,pre-made layers which layouts can be memorized. Spelunky attempted to make careful mix of two: substituting memorization of layots with procedurally generated mazes while at the same time remaining accessible and appealing platformer.

Level:4 Hits:29(29) Str:16(16) Gold:718 Armor:5 Exp:4/76
SCORE-000900 TIME 0465 BLK 1-02
PLAYER
ENEMY
P=02

Fig. 1. Original Rogue and Castlevania

III. Level generation

Every level in Spelunky is built out of a simple 4x4 grid of rooms and each of room is made up of 10x8 tiles. This simplifies the generation since you dont need to worry about whether a room is going to overlap another room or how to connect them. Additionally there is a variation in the settings. You got four different areas: mines, jungle, ice caves and lastly the Temple. Since those are natural or ruined environments they can be easily molded into various levels without looking out of place. That helps when youre generating levels algorithmically when a repetitive setting like an office building would make you bored way quicker and possibly limit player's movement. Each area is represented by four levels summing up to sixteen in total. So final decision of room's template comes down to area of the level, and the adjacent rooms layout. Two nonadjacent templates can be connected with a corridor for instance.

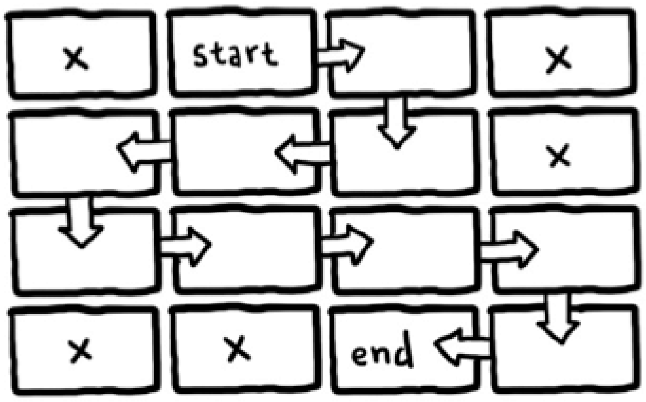

Fig. 2. Spelunky's level template

There is a set of general rules that generator has to follow in order to create a playable level:

- Each level starts at the door on the top row of the grid and end on the bottom row where exit door will be placed.
- There must be a way to get from the starting point to the exit without using tools like bombs, rope or any special items. Everything that does not lie on this path can be separated with walls or remain inaccessible.
- Generator must create a layout in such a way that eliminates any possibility for player to get stuck. By doing otherwise player would have to use of special items to get out. However if such a place is created after level generation (in other words by player interactions) it's ok.

- Adjacent rooms connection should be hard to notice but it is not prioritized.

.

One might question why we're heading strictly downward and not upward or even in variations of the two. That's because it is easier to let the player fall down and still have an option to progress rather than think about how a path that can be used to climb up can be generated.

The generator's first priority is to make sure that there is a path from the entrance to the exit that is can be accessed without the use of bombs, rope or any special items. This makes playes success less dependent on luck as well as reduces the chance of getting stuck. Saved bombs and ropes can then be used on something else and not wasted on badly generated levels.

Level generation script starts off by picking an entrance in the top row of the level grid. It is then randomly moves left or right until it hits the edge or randomly chooses to do so. Rooms that the algorithm never walks through are marked with the lowest value of zero.Rooms that lie on the path towards the exit are marked with values 1,2 or 3. This lets the game know whether a room is on the path and how exactly it will be connected to other rooms. 1 is a simple corridor, 2 is a corridor with connection to the room on the bottom and 3 have a connection to the upper cell. In the case where a 2 is on top of another 2 the bottom room will also have an opening on the top. On the bottom row the algorithm stops and marks the room it stops on as the exit.

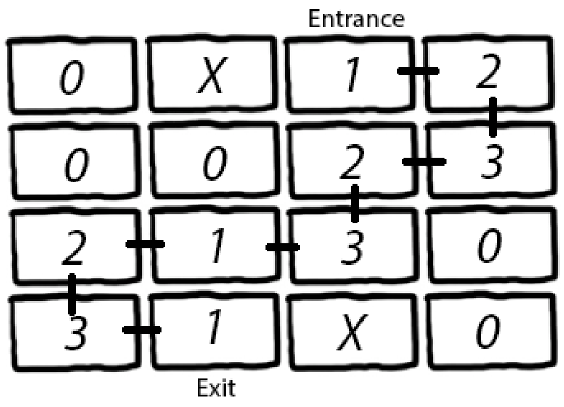

Fig. 3. Room layout example

Rooms that are not on the path (0) are treated as side rooms. The player may not necessarily be able to reach them without destroying the walls so they can be completely sealed off. Given the fact that terrain in Spelunky is destructable it makes level generation easier since we only need to care about the path from entrance to the exit. If an opening on the edge of the room is connected with opening of a side room then everything is fine, if not it's still fine. Openings that end with a wall will simply look like they were walls to begin with.

When the path is generated each room will be built out of tiles. First, it randomly selects a room template out of of six to twelve templates. Those depend on the type (1,2,3) of the room. Some of the examples will build a room with low ceiling, a ladder or a platform which can also vary in their placement these types are defined as simple string like this:

```
0000000011 0060000L040000000P110000000L1100
00000L11000000000110000000011111222 2111
```

If you will split it into lines of ten characters it would make much more sense:

```
0000000011
0060000L04
0000000P11
0000000L11
0000000L11
0000000011
0000000011
1112222111
```

Looks familiar isn't it?

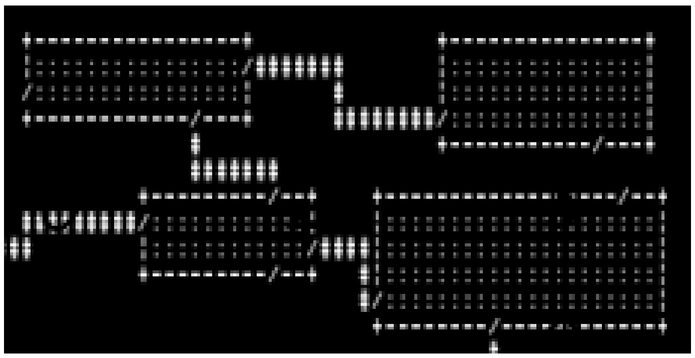

Fig. 4. Rogue's rooms layout

As you might expect zeros are empty spaces. Ones are walls, Ls are ladders, P is a wooden platform, and the 4 is a stone block. Two is a special case of 1 more specifically it has a 50 percent chance to be empty. After the room tiles were assigned sprittes will be loaded. 6 that floats in the air is a group of tiles 5x3 which are randomly selected out of a possible ten. An example of a group might be a platform:

```
01110
02220
00000
```

When the program replaces floating six in the room with this group the room will look like this:

```
0000000011
0001110L04
0002220P11
0000000L11
0000000L11
```

0000000011
0000000011
1112222111

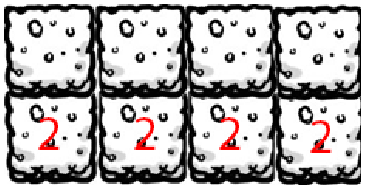

Fig. 5. Platform tile assignment

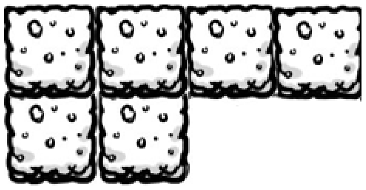

Fig. 6. Possible outcome

Aforementioned groups of tiles come in two types: the ones that float in the air like our example and the ones that lie on the floor. There are also special cases like jungle plants that hang on the ceiling. All these elements contribute to an overall illusion of variety when in reality this can be achieved with small amounts of assets.

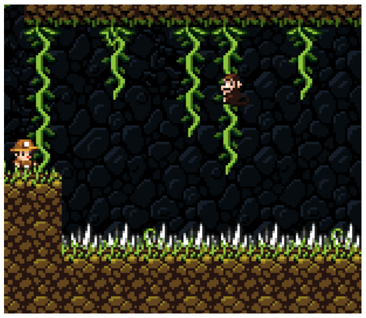

Fig. 7. Special asset example

Once room generator has completed room generation it is time to populate our rooms with monsters, traps, items, and treasures. An entity generator checks each floor tile and randomly decides whether to generate an entity on or around the tile or not while taking surrounding tile layout into consideration. For example item crates, treasure chests, gems, and gold will more likely to spawn in single tile spaces which are surrounded by walls in three out of the four directions. After the last step is done the level is populated with all sorts of the deadly obstacles and rewards.

IV. Conclusion

Shifting focus from pure A to B traversal throughout the level allows player to make other decisions that would influence overall score. Do you leave the Damsel and sacrifice an extra life that might be taken away by a spider that will fall onto your head? Will you spend money on the special items or rob the shop and deal with the consequences. Or maybe you want will spend extra time collecting gems before the ghost appears. These decisions would have been far less interesting if you had already played the level before. But Spelunky forces you to be aware of the situation and plan ahead with it's unique levels and situations. You have no other choice but to master mechanics instead of levels because only they will keep you alive for longer.

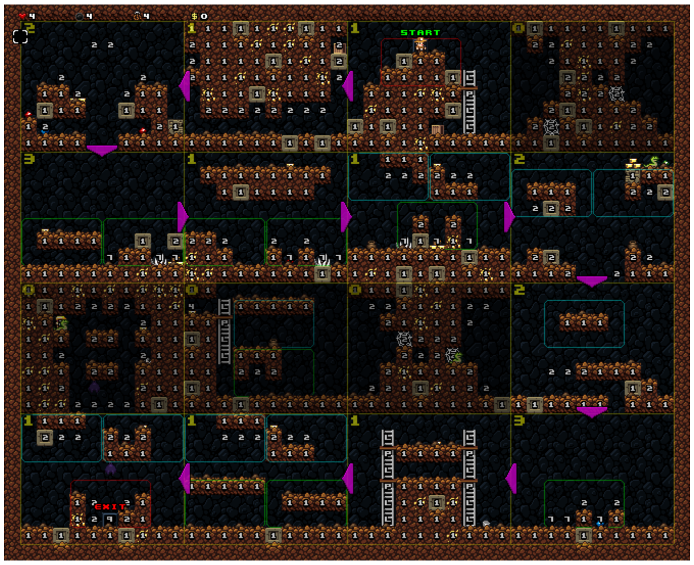

Fig. 8. Fully constructed level

References

[1] D. Yu The Full Spelunky on Spelunky https://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky

[2] D. Kazemi Spelunky level generator Part 1 http://tinysubversions.com/spelunkyGen

[3] D. Kazemi Spelunky level generator Part 2 http://tinysubversions.com/spelunkyGen2

[4] R.Terrell A Spelunky Game Design Analysis https://www.gamasutra.com/blogs/RichardTerrell/20121115/181646/A_Spelunky_Game_Design_Analysis__Pt_2.php

[5] Spelunky Documentary https://www.youtube.com/watch?v=jv434Xyybqc