Dungeons Deep Down

By Jonah Alligood

Preface

I created this game in around one to two days during my winter vacation. I don't often complete games, and I figured I'd like to challenge myself by completing a simple game for the first time in over a year (probably two years actually). My original thought was to have a sort of dungeon crawler, which is what I created. The complexity of the game I thought of would not have been very difficult to implement in the grand scheme of it all, but in the end, I wanted to focus on simplicity. I hope you enjoy playing the game and/or reading the paper.

Game Summary

As of release, the game does not have a story line, and is simply a rogue-like dungeon crawler. The player is an unnamed character pointlessly slaughtering monsters that reside in dungeons deep underground. The player gains experience by fighting monsters, and earns gold by clearing rooms. Monsters get progressively more difficult as the player levels up. The game is never won, rather the player will die, or quit, displaying the overall score at the end. The score is determined by gold gathered, experience gained, and level reached.

How It Works

Above is the formal synopsis of the game, whereas this segment of the paper will contrast on the level of formalities and simplicity. I'd like to walk through what the game is, what it means, how it functions, and possible ways to maybe exploit the simplicity of the game to become a total baller in terms of in-game gold.

This game is just a simple dungeon crawler, written in the language of C. A simple game in C is a bit of a paradox, but I enjoy writing things down at the low level, and challenging myself to complete more difficult tasks than something like Python or Java would provide me (not to say those wouldn't bring their own challenges). I've already gone over why it's simple, so I won't go into that now. I'll start with how the game starts, and go over what everything does.

The game starts the player off in a room in the middle of the map. This room is specifically set to spawn only one monster, and contains 100 gold. This is done for a couple of reasons. Firstly, it allows the room to be cleared after one battle that the player pretty much can't lose. Secondly, it allows the player to have the ability to heal a couple of times before clearing another room for more gold. This can be taken advantage of immediately, as after beating the first monster, the player levels up, increasing player stats, as well as spawnable monster stats. This happens every time the player levels up after defeating a monster with proper amounts of experience points.

After presumably beating the first monster and clearing the first room, one could then move in four different directions: North, South, East, and West. But before moving, the player could take advantage of the “look” command, which allows the player to see how many monsters and how much gold a particular room contains. The player can’t know the exact stats of the monsters, until the player moves to that room.

I’d like to take a moment to talk about the room spawning. The memory in the game for the rooms is allocated at the start of the program. Only one room is actually initialized at the start of the game, and that’s the middle of the map where the player is, and as previously stated, it is designed to spawn one monster with 100 gold. Every other room is left to be null data. As memory was already allocated, saving space wasn’t the issue. I decided that rooms should be initialized when moved to, or looked at. There would then be no point in further initializing rooms at the start, if they would be overwritten as soon as the player moved to or looked at them. Technically one could look at and move to every room on the map and they would only re-spawn new things if a room was cleared. This could make grinding for experience and gold much easier.

Monsters are spawned in rooms as rooms spawn. Monsters also spawn with stats based on what level the player is currently at. One could go around and spawn all sorts of low level monsters before going back to the middle and clearing the room. All the low level monsters would remain in the other rooms, and the player could defeat them quite easily, levelling up and not worrying about more difficult things to come their way. Of course, there’s not much fun in that, but as it’s a simple game, maybe cheating would make it a little more fun.

The fighting system wasn’t too difficult to tackle. I figured the defender’s defense stat subtracted from the attacker’s power stat would suffice for damage dealt to the defender. The least amount of damage to be dealt is 1. Spamming the first option in the fight dialog worked out pretty well for me in testing during the player’s early levels, but as I haven’t done deep testing of the game (the highest level I got for testing was 4) I’m not sure how well that technique would work at higher levels. Upon defeating a monster, the player is awarded experience points. The simple system for that was the sum of the monster’s stats.

On the topic of systems, the last thing I constructed was the levelling system. I didn’t try too hard with how stats and other items would increase, but there is a level of randomness to it. One could view the code of the `player.c` file to see the numbers between which the stats would randomly increment by.

Command List

Please just enter “help” into the game. I’ve already written it once, I’d like to not write it again. Thank you for dealing with my laziness, it’s much appreciated.

Conclusion

I completed the task of creating a simple game in the time span of about 24 hours. It’s nothing special, but it’s still functional, and could virtually go on until the program runs out of memory. I learned a few things about how to structure a game, like what data should be kept in a `struct`, what data should be passed to functions, and variables in headers need to be `static` in order to actually count for something. I also learned that my brain will explode if I try to think things through too many times. It was a great experience to have a goal in mind, and to solve the puzzle that the code presented. I enjoyed working with the code instead of against it (I guess I’d be working against the compiler but whatever). The game works the way I imagined, and that’s the best that could’ve happened. I hope you enjoy the game if you can compile it. If you’re interested in what the code does, check that out too.