DUNGEON3

Software project

Overview:

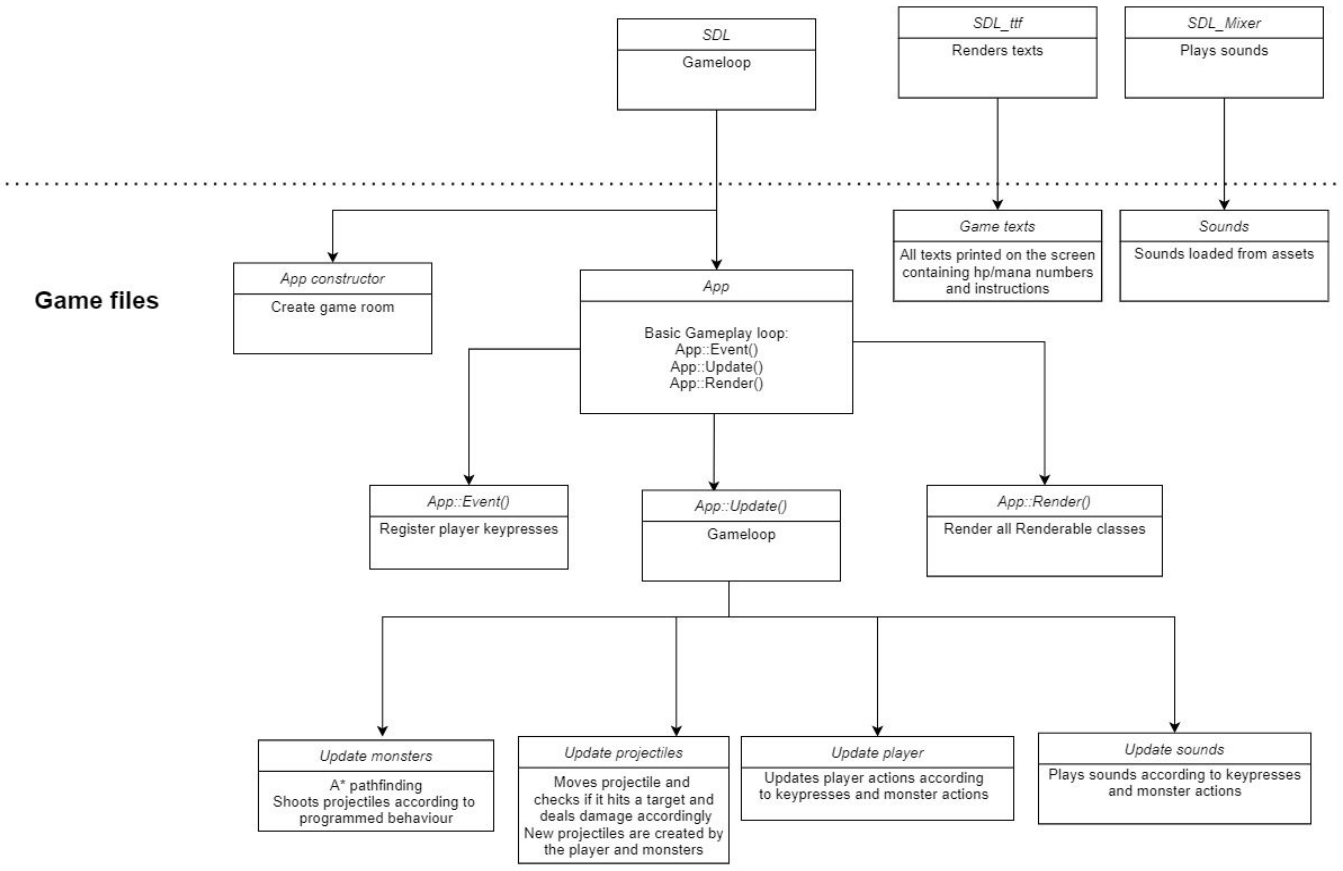
The software is a 2D dungeon crawler game, where the player can move around freely in real time inside a 2D closed environment consisting of randomly generated stages. Game rooms can contain chests with useful items or monsters which will try to kill you. The game has basically infinite floors and gets harder every floor. The goal is to reach as low a floor as possible. More about the actual gameplay is explained in the instructions part.

The software allows the user to interact with the game character with given keypresses. It does not allow the user any additional control over the character. Most of the game's functions, including room generation and item stats are randomly generated meaning the player has to always adapt to the given situation.

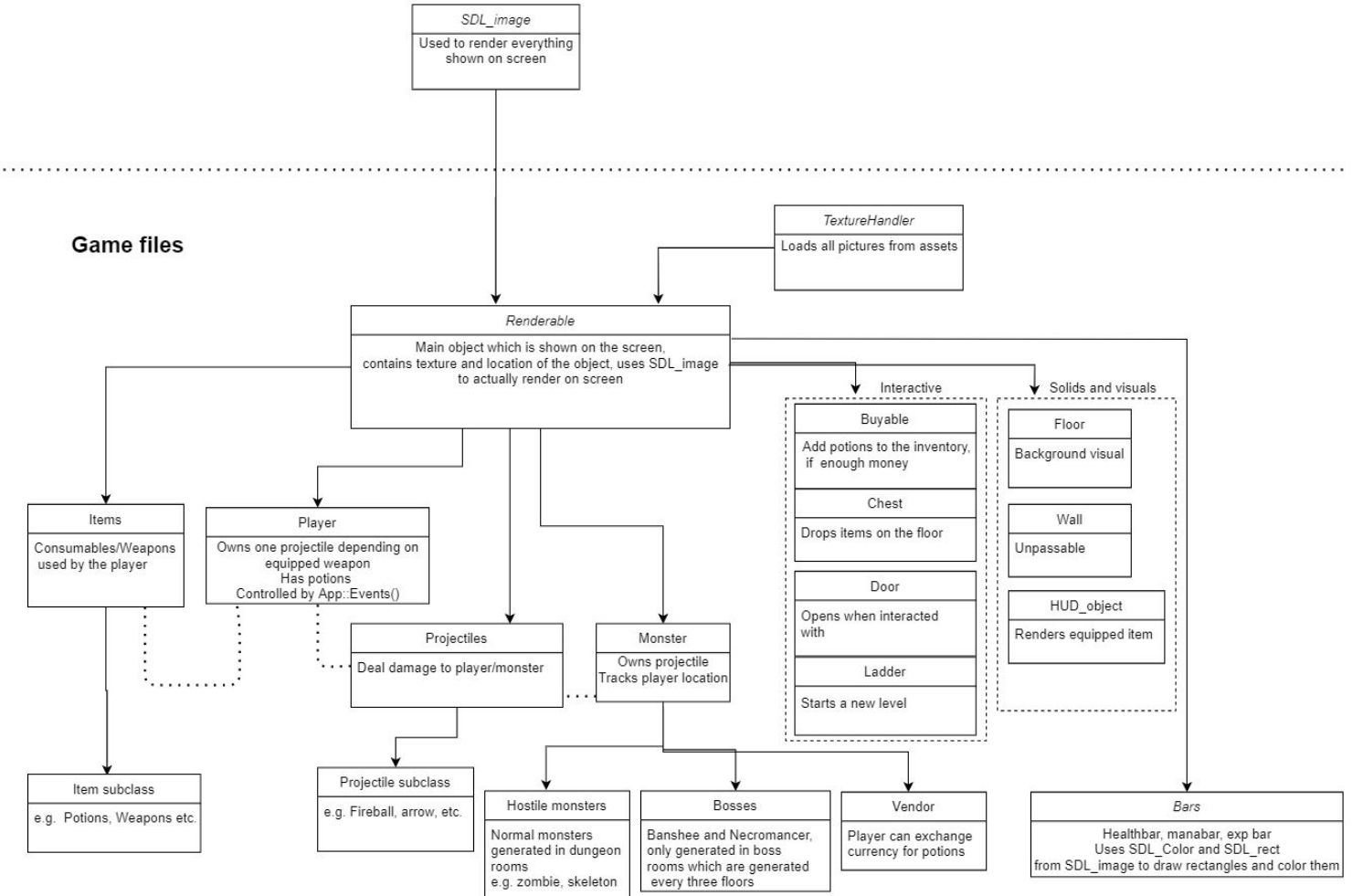
The software does not contain any inbuilt high score system and is single player only. The only competition is against the dungeon. Try to get as far as you can!

Software structure:

External libraries



External libraries



Instructions for building and using the software:

How to compile:

For Linux:

Install CMake, git and a compiler if you don’t already have them.

Run `run.sh` with sudo permissions, this will install required packages which are `SDL2`, `SDL2_image`, `SDL2_mixer` and `SDL2_ttf`, download SDL2 CMake dependencies and compile the game. Binary `dungeon` of the game will be generated at `build/dungeon`.

One could also execute the commands manually

Install git and cmake with package manager

```
apt-get install git
```

```
apt-get install cmake
```

```
apt-get install build-essential
```

Install required SDL packages using the package manager

```
apt-get install libsdl2-dev
```

```
apt-get install libsdl2-image-dev
```

```
apt-get install libsdl2-ttf-dev
```

```
apt-get install libsdl2-mixer-dev
```

Get CMake support for SDL2

```
git clone https://gitlab.com/aminosbh/sdl2-cmake-modules.git cmake/sdl2
```

Make build directory and use cmake and make to compile the game

```
mkdir build
```

```
cd build
```

```
cmake ..
```

```
make
```

Run the game with

```
./dungeon
```

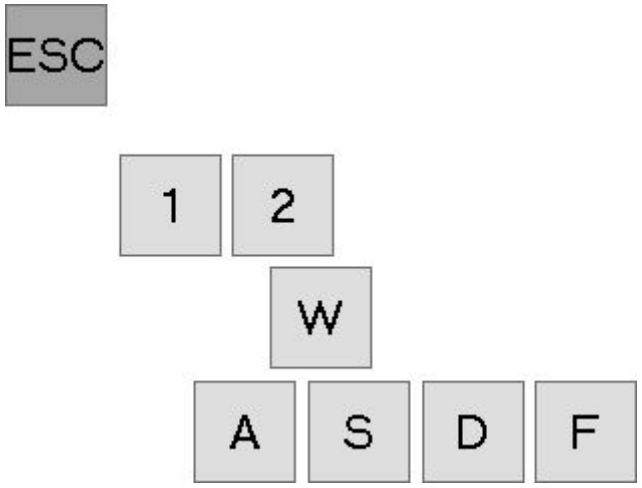
For Windows:

Install Visual Studio 16 2019 and open the solution file (.sln). Select dungeon3 as a startup project and run with windows local debugger. This will start the game and create an executable at x64/Debug or x64/Release. You could also copy the exe file to src/ directory to run the game from the executable since all the required .dll files and assets are located there.

.vcxproj files and lib/SDL2 are solely used for windows development and compiling.

User guide:

Controls and basics:



Keyboard layout of the controls

W, A, S, D - Movement keys
F - Interact
1 - Use a health potion
2 - Use a mana potion

Esc - Exit the game
Left mouse button - Attack

This is you, the player. Your objective might be unclear for now, but read on and you will soon find out.

The player

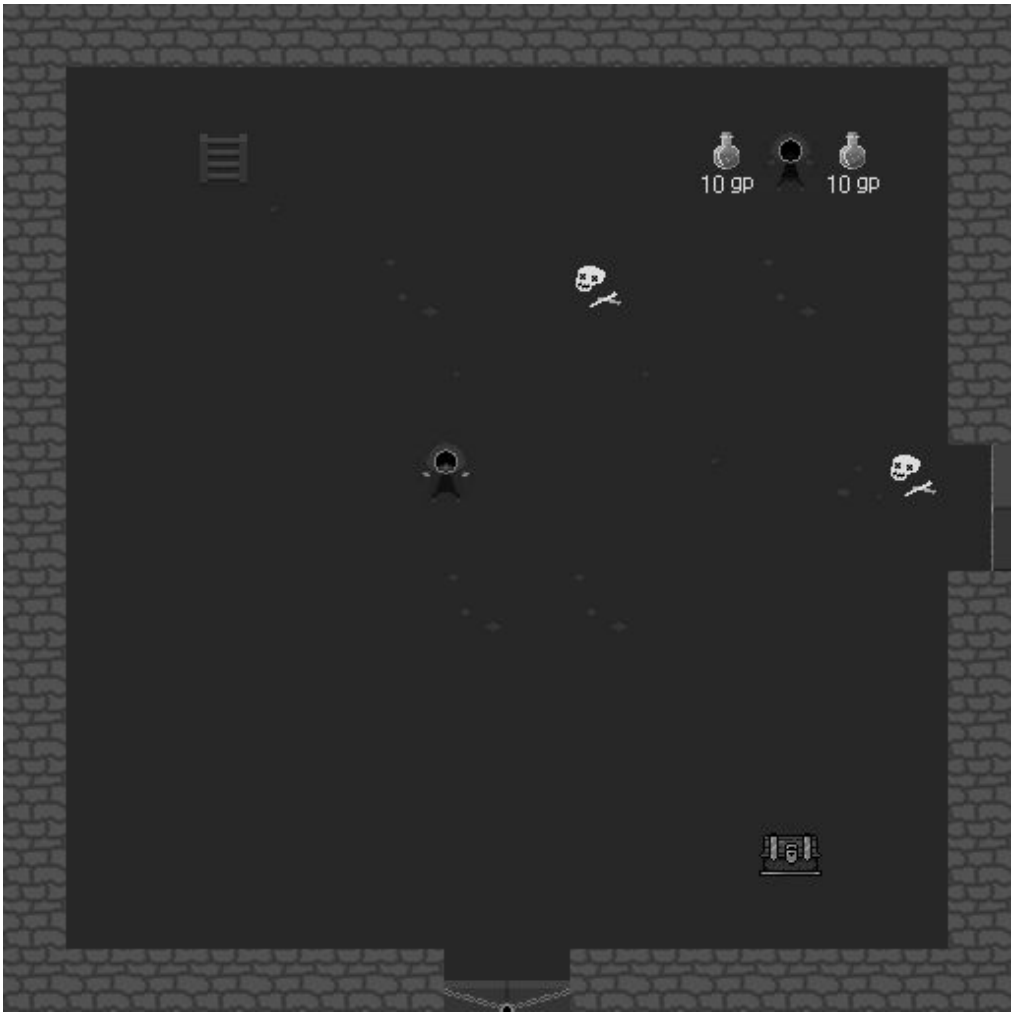
The objective:

The objective of the game is to fight your way through the endless dungeon, clearing levels and fighting different types of monsters.

You start the game at the main menu, you should familiarize yourself with the controls before moving on.

Three difficulty settings, easy, medium and hard.

To select the difficulty setting of your choice, go down the ladder that represents it.

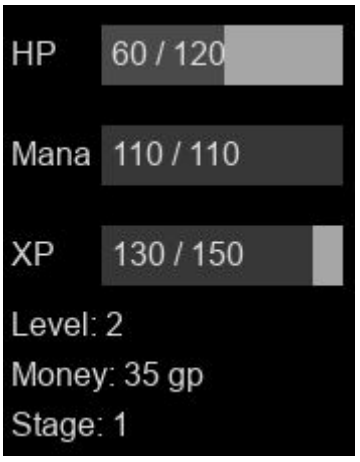


The starting room of each normal level

You should now find yourself in a different room. Each normal stage starts at the bottom of a ladder in the starting room. Your primary goal now is to find a new ladder that leads you to the next level. Should you manage to reach the end ladder, you will reach a new stage until you reach the boss stage (2 stages). Defeat the boss and the cycle repeats until you die. To move on from this room, interact with the doors to reveal new parts of the dungeon.

A typical monster encounter and reaching the end ladder

In different rooms of the dungeon you will find monsters that will try to kill you and chests that contain potions and/or weapons. Monsters will either try to get up close and personal with you or try to attack you from afar. Should they manage to hit you, you will lose health points until you have none and you will die. The game will restart if this happens. Upon killing a monster you are rewarded with 5 gold pieces and some experience points. After gaining enough experience, you will level up and your health and mana will increase slightly. You should see all of this information in the user interface.



The user interface, health (HP), mana, experience (XP), level, money and the current stage are displayed here

Should you be running low on health, use a health potion! Potions when used will restore some of your missing health or mana depending on the colour of the potion (red = health, blue = mana). Potions can also be bought from the merchant at the start of each level for 10 gold pieces.

The potion user interface, this tells you how many potions of each kind you have

The merchant and his wares

In addition to finding potions from chests, you may also find different weapons. There are three different weapon types: a sword, a bow and a magical staff. You'll start the game with a sword, but you should be able to find other types of weapons fairly quickly. These weapons will get better throughout the game and their damage will scale with your level (therefore you shouldn't just ignore the monsters).

The item user interface, weapon damage and speed displayed

These are the basics of the game. Should you be interested in the finer details of monsters and different items, read the sections about non player characters and items.

Non player characters (NPC):

Generic monsters:

Zombie / Super zombie

A zombie and a super zombie

A slow moving generic undead

Combat type: Melee
Health: 30 * (1.0 + 0.25 * difficulty)
Speed: 600 * (1.0 + 0.02 * difficulty)

Ghost / Banshee

A ghost and a banshee

A fast moving ghost, can go through walls
Combat type: Melee
Health: 20 * (1.0 + 0.25 * difficulty)
Speed: 750 * (1.0 + 0.02 * difficulty)

Skeleton / Super skeleton

A skeleton and a super skeleton

An undead archer, shoots arrows
Combat type: Ranged
Health: 20 * (1.0 + 0.25 * difficulty)
Speed: 680 * (1.0 + 0.02 * difficulty)

Skeletal mage / Lich

A skeletal mage and a lich

An undead mage, shoots shards of ice
Combat type: Magic
Health: 25 * (1.0 + 0.25 * difficulty)
Speed: 700 * (1.0 + 0.02 * difficulty)

Boss monsters:

Necromancer

The necromancer

A fast moving undead necromancer, resurrects nearby dead monsters, shoots shards of ice
Combat type: Magic
Health: 300 * (1.0 + 0.25 * difficulty)
Speed: 800 * (1.0 + 0.02 * difficulty)

Banshee overlord

The banshee overlord

A fast moving ghost, can go through walls, spawns ghosts and banshees

Combat type: Melee
Health: 250 * (1.0 + 0.25 * difficulty)
Speed: 750 * (1.0 + 0.02 * difficulty)

Other:

The merchant

The merchant

Friendly, spawns at the start of each stage
Sells both types of potions for 10 gold pieces

Note: difficulty is calculated as = stage * difficulty_multiplier
difficulty_multiplier is 0.5 for easy, 1 for medium and 2 for hard

Items:

Bow and arrow

Base damage: 4
Base speed: 0.4
(2.5 attacks per second by default)

Sword

Base damage: 6
Base speed: 0.1
(10 attacks per second by default)

Staff of fire

Base damage: 5
Base speed: 0.2
(5 attacks per second by default)

Health (red) potion

Health restoration: 20

Mana (blue) potion

Mana restoration: 20

The damage and speed values values for items are randomized as follows:

Damage = base_damage * floor(player_level + sqrt(sqrt(player_level) * (rand() % 150 + 50.0) / 100.0))

Speed = base_speed * floor(0.8 + (0.4 * (rand() % 100)) / 100.0)

Testing:

The game was mainly tested by compiling the game and noticing functionalities ingame. Many of the implementations are visual or random, which are not easily tested (or cannot be tested with testing methods such as GTEST).

For monster pathfinding unit tests with GTEST were implemented, found in “test/test.cpp” file. Several test cases were made to test the subfunctions used in pathfinding algorithm (A*). Pathfinding required separate tests, since the functionality could not be trivially seen from the behaviour of monsters.

Floor generation is also automatically tested inside the App::Generate function. The function tests if both the starting room and end room exits, then it is a valid generation. If not, then it is an illegal generation and it will discard this generation and a new level will be generated. Furthermore, should any rooms be generated on top of existing rooms, the function will change that room to the end room. This always results in a valid generation. This is crucial since the floors are generated randomly.