Vilnius Gediminas Technical University
Department of Information Technology

Coursework Report
Ooventures

Student: Zehra İrem Kuyucu, ITfuc-21
Lecturer: Julius Čepukėnas

April 26, 2022

Contents

1 Introduction

Ooventures, stands for "OOP adventures", is a very minimalistic rougelike[1] game. The main goal of the player is to defeat all monsters and be the last entity standing on the map. The map is generated based on terminal size. The number of monsters spawned, health and attack points differ based on the difficulty setting.

Ooventures is available under WTFPL license on Github.

2 Libraries

The standard library functions are provided by the GNU C++ Standard Library.[2]

Ncurses library is used for the text-based user interface.[3]

3 Classes

3.1 Map

This class represents the map in the background. The constructor of this class initializes Ncurses and colors. `Map::print()` prints the map, a dotted background and a frame that is as big as the terminal size. The destructor function restores the terminal by ending Ncurses mode.

```
1 class Map {
2         public:
3                 void print();
4                 Map();
5                 ~Map();
6 };
```

3.2 Entity

This class represents any entity that is spawned on the map. Entities are capable of moving and dealing damage. The `Monster` and `Player` classes are inherited from this class. Using the `protected` keyword instead of `private` allows the members `symbol` and `health` to be accesible within these derived classes. `move(int c)` is a virtual member function, because the derived classes contain different definitions of it. `symbol` contains the character that is printed on screen each time the entity moves.

The subtraction operator is overriden in this class. Whenever two `Entity` type objects are subtracted from each other, the initial one's health score is lowered based on the latter's damage score. This way two entities on the map can deal damage to each other.

```
1  class Entity {
2          protected:
3                  char symbol;
4                  int health;
5
6          public:
7                  Position pos;
8                  unsigned int damage;
9
10                 void print();
11                 virtual void move(int c) = 0;
12                 Entity(char sym, int hp, int ap);
13
14                 int operator - (Entity const &obj) {
15                         health -= obj.damage;
16                         return health;
17                 }
18 };
```

3.3 Player

This class represents the entity that is controlled by the player. `move(int c)` takes a key code as an argument and moves accordingly. `aroundMonster()` returns the coordinates of a monster that is next to the player. `using Entity::Entity` allows `Player` class objects to be initialized using the same constructor from the `Entity` class.

```
1 class Player : public Entity {
2         public:
3                 void move(int c);
4                 Position aroundMonster();
5                 using Entity::Entity;
6 };
```

3.4 Monster

This class represents monsters whose movements are out of the player's control. `move(int c)` randomizes monsters' movements. The key code of the player is still passed to this function, because once in a while monsters run away from the player. The constructor of the `Entity` class is used, yet again.

```
1 class Monster: public Entity {
2         public:
3                 void move(int c);
4                 using Entity::Entity;
5 };
```

3.5 Game

This class represents the current game and its state. It contains the monsters, the player and the map. The constructor function cares about the difficulty level. Based on the level different number of monsters are spawned, the entities have different health and attack points. `state` contains the current game state: 0 is for ongoing, 1 is for loss and 2 is for player victory. `getMonsterNearby(Position mPos)` returns an iterator to the monster who is next to the player. Later on damage is dealt to that monster. `nextTurn(int c)` moves entities and updates state. `print()` prints the map and the entities.

```
1  class Game {
2          private:
3                  Player *player;
4                  Map *map;
5                  std::vector<Monster> monsters;
6
7          public:
8                  int state;
9                  std::vector<Monster>::iterator
                      getMonsterNearby(Position mPos);
10                 int nextTurn(int c);
11                 void print();
12                 Game(int difficulty);
13                 ~Game();
14 };
```

4 Building and Running

Ooventures can be built using the provided Makefile.
`$ cd ooventures && make`
The produced binary requires a difficulty level to be specified.
`$ ./ooventures [difficulty level 1 to 3]`
The difficulty levels are as described:

1. Easy.
2. Medium.
3. Hardcore.

The game has two outcomes, the player has won or the player has been slain. The first occurs when the player kills all the monsters (monsters vector is empty), the latter occurs when the player dies (runs out of health points) when there are still monsters.

5 Limitations

This program does not handle the `SIGWINCH` signal and therefore is unable to resize in case the terminal is resized. The terminal size must not be changed or else the window might be broken.

In case of a fight between the player and a monster, the player is unable to move until the fight comes to an end.

References

[1] Rougelike on Wikipedia. *https://en.wikipedia.org/wiki/Roguelike*

[2] The GNU C++ Library. *https://gcc.gnu.org/onlinedocs/libstdc++*

[3] NCURSES - New Curses. *https://invisible-island.net/ncurses*