**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «ООП»**

Тема: Создание классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 2383 |  | Лустенкова Д.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Создать первые классы, необходимые для реализации проекта первой игры на языке программирования С++, с требуемыми для них методами.

## Задание

Для данной лаб. работы UML диаграмма классов в отчете не обязательна.

а) Создать класс игрока. У игрока должны быть поля, которые определяют его характеристики, например кол-во жизней, очков и.т.д. Также в классе игрока необходимо реализовать ряд методов для работы с его характеристиками. Данные методы должны контролировать значения характеристик (делать проверку на диапазон значений).

б) Создать класс, передвигающий игрока по полю и работу с характеристиками. Данный класс всегда должен знать об объекте игрока, которым управляет, но не создавать класс игрока. В следующих лаб. работах данный класс будет проводить проверку, может ли игрок совершить перемещение по карте.

Примечания:

* Не забывайте для полей и методов определять модификатор доступа
* Для указания направления движения можно использовать перечисление enum или дополнительную систему классов. Использования чисел или строк является для указания направления является плохой практикой
* Делать отдельный метод под каждое направление делает класс перегруженным, и в будущем ограничивает масштабирование класса

## Выполнение работы

В качестве продукта проекта была выбрана игра с использованием GUI, целью которой будет уничтожение врагов, возникающих на поле и приближающихся к игроку для нанесения ему урона.

Для удобства работы с направлениями перемещения игрока / его врагов с помощью *enum* был создан *Direction*, который может принимать одно из следующий значений: *Top*, *Bottom*, *Right*, *Left*.

Для удобства работы с координатами была создана структура *cords*, содержащая в себе два целочисленных поля: *x\_* и *y\_*.

Так как у игрока и его врагов есть много общих свойств, создан класс entity - сущность.

class entity

Класс имеет поля для логического значения жива сущность или нет (*bool alive\_*), целочисленных значений количества жизней (*unsigned int lives\_*) и наносимого этой сущностью урона за раз (*unsigned int attack\_power\_*), а также поле координат (*cords cords\_*).

Методы класса *entity*:

* *entity()* – конструктор класса, инициализирующий поля значениями по умолчанию;
* *void move(field& f, Direction move\_direction)* – метод передвижения сущности по полю;
* void kill() – метод убийства сущности;
* void healthpointer(bool treat, unsigned int how\_many) – метод изменения значения здоровья (treat == true, прибавлять жизни; treat == false, отнимать жизни);
* unsigned int attack\_power() – метод, возвращающий текущее количество наносимого этой сущностью урона за раз, для определения, сколько жизней нужно будет снять сопернику при попадании.

Следующие два класса, а именно классы player и enemy, являются дочерними классами класса *entity*.

Для дальнейшего создания полноценного персонажа, за которого будет играть пользователь, создан класс *player*.

class player

В классе добавлено целочисленное поле уровня игрока. Переопределён конструктор класса (в нём появляется инициализация уровня игрока единицей) и метод *void kill()* (помимо базовых занулений, зануляется так же и уровень). Добавлены следующие методы:

* *cords locate()* – метод, возвращающий координаты игрока (с его помощью враги будут двигаться у нему);
* *void raise\_power(int raise)* – метод повышения наносимого игроком урона (игрок будет подбирать некие артефакты, которые будут повышать ему урон);
* *void levelup()* – метод повышения уровня игрока (вместе с его базовыми характеристиками).

Далее создан класс врага игрока – *enemy*.

class enemy

Переопределён конструктор класса (здесь он принимает координаты появления enemy) и создан метод *void move\_to\_player(player p)*, перемещающий врага к игроку по прямой. В методе использован алгоритм рисования прямой из курсовой работы по программированию 2-ого семестра.

В следующих работах у появятся дочерние классы (разные враги будут иметь разные характеристики).

Когда в проекте появятся события, при приближении "впритык" (между игроком и врагом отсутствуют клетки) одних врагов и при таком приближении, что между игроком и врагом остаётся одна клетка, других врагов, враги будут наносить урон игроку. Радиус наносимого игроком урона будет завесить от артефактов (без них нужно впритык) и также будет определён через события.

Для контроля существ создан класс movement. В дальнейшем, он будет обрабатывать события.

class controller

Класс имеет следующие поля: ссылку на существо *entity& e* и ссылку на поле *field& f*.

Методы класса:

* *controller(entity& somebody, field& somewhere)* – конструктор класса, принимающий существо и поле, на котором оно будет находиться и двигаться;
* *void move(Direction move\_direction)* – метод перемещения существа по полю.

main()

В функции main() на данный момент реализации проекта находятся тестовые элементы создания объектов класса и методов их обработки.

Программный код приведен в приложении А.

## UML-диаграмма классов

entity

* # bool alive\_;
* # uint32\_t lives\_;
* # uint32\_t attack\_power\_;
* + entity();
* + void move(field& f, Direction move\_direction);
* + void kill();
* + void healthpointer(bool treat, unsigned int how\_many);
* + unsigned int attack\_power();

controller

* # entity& e;
* # field& f;
* # cords\_ xy;
* + controller(entity &somebody, field &somewhere, cords xy);
* + cords get\_cords\_();
* + void move(Direction move\_direction);

field

* # uint32\_t height\_;
* # uint32\_t width\_;
* # cell\*\* cells\_;
* + field(uint32\_t h, uint32\_t w);
* + field();
* + field copy();
* + cell\_type touch\_cell(int x, int y);
* + ~field();

Классы предстоящей лабораторной работы №2:

cell

* # cords tp\_cords\_;
* + cell(bool wall);
* + cell();
* + bool touch();

enemy

* + enemy();
* + cords step\_to\_move\_to\_player(int my\_x, int my\_y, int p\_x, int p\_y);

player

* # uint32\_t level\_;
* + player();
* + void kill();
* + void raise\_power(int raise);
* + void levelup();

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | player p1;  p1.kill();  field f1 = field(100, 100);  field f2 = f1.copy();  controller c\_p1(p1, f1, {1, 1});  c\_p1.move(Right);  std::cout << "p1: " << c\_p1.get\_cords\_().x\_ << ' ' << c\_p1.get\_cords\_().y\_ << '\n'; | p1: 2 1  p1: 22 21 | Задача выполнена корректно. |
|  | for (int i = 0; i < 20; i++)  {  c\_p1.move(Right);  c\_p1.move(Top);  }  std::cout << "p1: " << c\_p1.get\_cords\_().x\_ << ' ' << c\_p1.get\_cords\_().y\_ << '\n';  enemy e1;  controller c\_e1(e1, f1, {3, 4});  e1.step\_to\_move\_to\_player(c\_e1.get\_cords\_().x\_, c\_e1.get\_cords\_().y\_, c\_p1.get\_cords\_().x\_, c\_p1.get\_cords\_().y\_);  return 0; | 13 83  13 82  13 81  13 80  14 79  14 78  14 77  14 76  14 75  14 74  14 73  15 72  15 71  15 70  15 69  15 68  15 67  15 66  16 65  16 64  16 63  16 62  16 61  16 60  16 59  17 58  17 57  17 56  17 55  17 54  17 53  17 52  18 51  18 50  18 49  18 48  18 47  18 46  19 45  19 44  19 43  19 42  19 41  19 40  19 39  20 38  20 37  20 36  20 35  20 34  20 33  20 32  21 31  21 30  21 29  21 28  21 27  21 26  21 25  22 24  22 23  22 22 | Задача выполнена корректно. |

## Выводы

Продумана логика игры. Написаны классы сущности, её подклассы: игрока и врага, класс движения.

Сделаны первые шаги в реализации проекта игры на языке программирования С++.

# **Приложение А Исходный код программы**

Название файла: entity.h

#ifndef ENTITY\_H

#define ENTITY\_H

#include "../field/field.h"

#include <cstdint>//

class entity

{

private:

protected:

bool alive\_;

uint32\_t lives\_;

uint32\_t attack\_power\_;

public:

entity();

void kill();

void add\_lives\_(int how\_many);

uint32\_t get\_attack\_power\_();

};

#endif

Название файла: entity.cpp

#include "entity.h"

entity::entity()

: alive\_ {true},

lives\_ {100},

attack\_power\_ {10}

{

}

void entity::kill()

{

alive\_ = false;

lives\_ = 0;

}

void entity::add\_lives\_(int how\_many)

{

if (alive\_)

{

lives\_ += how\_many;

if (lives\_ <= 0)

this->kill();

}

}

uint32\_t entity::get\_attack\_power\_()

{

return attack\_power\_;

}

Название файла: player.h

#include "entity.h"

class player : public entity

{

private:

protected:

uint32\_t level\_;

public:

player();

void kill();

void raise\_power(int raise);

void levelup();

};

Название файла: player.cpp

#include "player.h"

player::player()

: level\_ {1}

{

}

void player::kill()

{

level\_ = 0;

}

void player::raise\_power(int raise)

{

attack\_power\_ += raise;

}

void player::levelup()

{

if (alive\_)

{

level\_ += 1;

lives\_ += 100;

attack\_power\_ += 10;

}

}

Название файла: enemy.h

#include "entity.h"

class enemy : public entity

{

private:

protected:

public:

enemy();

cords step\_to\_move\_to\_player(int my\_x, int my\_y, int p\_x, int p\_y);

};

Название файла: enemy.cpp

#include "enemy.h"

enemy::enemy()

{

}

cords enemy::step\_to\_move\_to\_player(int my\_x, int my\_y, int p\_x, int p\_y)

{

int x1 = my\_x; // enemy x

int y1 = my\_y; // enemy y

int x2 = p\_x; // player x

int y2 = p\_y; // player y

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int d = dx - dy;

int x1\_is\_lower\_x2; int y1\_is\_lower\_y2;

if (x1 < x2)

x1\_is\_lower\_x2 = 1;

else

x1\_is\_lower\_x2 = -1;

if (y1 < y2)

y1\_is\_lower\_y2 = 1;

else

y1\_is\_lower\_y2 = -1;

int x = x1; int y = y1;

while(x != x2 || y != y2)

{

int doble\_d = 2 \* d;

if (doble\_d > -dy)

{

d -= dy;

x += x1\_is\_lower\_x2;

}

if (doble\_d < dx)

{

d += dx;

y += y1\_is\_lower\_y2;

}

return {x, y};

// for the test, that's really a straight line from the enemy to the player

/\*

cords\_.x\_ = x;

cords\_.y\_ = y;

std::cout << cords\_.x\_ << ' ' << cords\_.y\_ << '\n';

\*/

}

}

Название файла: controller.h

#include "entities/player.h"

#include "entities/enemy.h"

enum Direction {Top, Bottom, Right, Left};

class controller

{

private:

protected:

entity &e;

field &f;

cords cords\_;

public:

controller(entity &somebody, field &somewhere, cords xy);

cords get\_cords\_();

void move(Direction move\_direction);

};

Название файла: controller.cpp

#include "controller.h"

controller::controller(entity& somebody, field& somewhere, cords xy)

: e {somebody},

f {somewhere},

cords\_ {xy}

{

// Now we can manage events

}

cords controller::get\_cords\_()

{

return cords\_;

}

void controller::move(Direction move\_direction)

{

if (move\_direction == Top){

if (f.touch\_cell(cords\_.x\_, cords\_.y\_ + 1) != Barrier)

cords\_.y\_ += 1;

}

if (move\_direction == Bottom){

if (f.touch\_cell(cords\_.x\_, cords\_.y\_ - 1) != Barrier)

cords\_.y\_ -= 1;

}

if (move\_direction == Right){

if (f.touch\_cell(cords\_.x\_ + 1, cords\_.y\_ ) != Barrier)

cords\_.x\_ += 1;

}

if (move\_direction == Left){

if (f.touch\_cell(cords\_.x\_ - 1, cords\_.y\_ ) != Barrier)

cords\_.x\_ -= 1;

}

// if Spikes

if (f.touch\_cell(cords\_.x\_, cords\_.y\_) == Spikes)

{

e.add\_lives\_(-100);

}

// if Teleport

if (f.touch\_cell(cords\_.x\_, cords\_.y\_) == Teleport)

{

cords\_ = f.get\_tp\_cords\_of\_cell(cords\_.x\_, cords\_.y\_);

}

}

Название файла: main.cpp

#include "controller.h"

int main()

{

player p1;

p1.kill();

field f1 = field(100, 100);

field f2 = f1.copy();

controller c\_p1(p1, f1, {1, 1});

c\_p1.move(Right);

std::cout << "p1: " << c\_p1.get\_cords\_().x\_ << ' ' << c\_p1.get\_cords\_().y\_ << '\n';

for (int i = 0; i < 20; i++)

{

c\_p1.move(Right);

c\_p1.move(Top);

}

std::cout << "p1: " << c\_p1.get\_cords\_().x\_ << ' ' << c\_p1.get\_cords\_().y\_ << '\n';

enemy e1;

controller c\_e1(e1, f1, {3, 4});

e1.step\_to\_move\_to\_player(c\_e1.get\_cords\_().x\_, c\_e1.get\_cords\_().y\_, c\_p1.get\_cords\_().x\_, c\_p1.get\_cords\_().y\_);

return 0;

}