**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-ориентированное программирование»**

ТЕМА: СОЗДАНИЕ классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 2300 |  | Шумов О.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

**Цель работы.**

Изучить классы и основы ООП в C++, научиться создавать их, объявлять поля и методы в языке. Разобраться в работе модификаторов доступа, научиться выбирать походящие. Написать программу, соответствующую заданию, создать необходимые классы и методы с обеспечением сохранения инварианта.

## Задание.

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.

Были созданы 3 класса:

1. Player – класс игрока, хранящий информацию о состоянии игрока: количество жизней и очков, а также координаты.
2. Directions – enum class, содержащий направления перемещения (вверх, вниз, влево, вправо).
3. Control – класс, отвечающий за перемещение игрока по полю и за работу с характеристиками.

Все классы объявлены в соответствующих заголовочных файлах, а реализация их методов представлена в .cc файлах.

В классе Player созданы следующие поля:

1. int health\_ - количество жизней. Если это значение становится меньшим или равным 0, то игра завершается. Имеет минимальное значение равное значению 0, а также максимальное равное 3. В конструкторе инициализируется значением 3.
2. int score\_ - количество очков. В конструкторе инициализируется значением 0.
3. int x\_ - координата игрока по оси х. В конструкторе инициализируется значением 0.
4. int y\_ - координата игрока по оси y. В конструкторе инициализируется значением 0.

Все перечисленные поля имею модификатор доступа private, что защищает их от нежелательного изменения из вне.

В классе Player реализованы следующие методы:

1. Player() – конструктор класса без аргументов. Устанавливает значения health\_, score\_, x\_ и y\_ равными соответствующим константам.
2. void set\_health(int health) – сеттер для поля health\_. Если переданное значение меньше или равно 0, то количество жизней устанавливается равным 0, на экран выводится надпись “You died!”, а также завершается программа. Если количество жизней больше 0, то в health\_ будет сохранено текущее значение количества жизней.
3. void display(Player player) – функция для вывода полей игрока: health\_ и score\_.
4. Остальные метода – геттеры и сеттеры соответствующих полей, названные в соответствии с рекомендациями из google c++ style guide.

В классе Direction реализовано перечисление enum в котором объявлены направления движения: вверх, вниз, вправо, влево (kUp, kDown, kLeft, kRight).

В классе Control реализовано поле player\_, которое содержит ссылку на объект класса Player. Также в классе была объявлена константа. Она нужна для удобства разработки. Если придётся менять одно из постоянных значений во всех частях кода, достаточно будет поменять значение константы.

1. const int kStepMove = 1 – константа, обозначающая значение изменения координаты по любой из осей в 2D.

В классе Control реализованы следующие методы:

1. Control(Player& player) : player\_(player) {} – конструктор, принимающий указатель на объект игрока и сохраняющий его в соответствующем поле.
2. int move(Directions direction) – метод, отвечающий за перемещение игрока по полю. В качестве аргумента принимает значение enum class direction и перемещает игрока в заданном направлении на количество клеток, равное значению изменения координаты игрока. Для этого реализована конструкция switch-case в которой изменяется одна из координат. С помощью методов: set\_x() и set\_y() класса Player изменённые данные сохраняются в полях объекта игрока.
3. void remove\_health() – метод, снимающий игроку жизнь и записывающий новое количество жизней с помощью метода set\_health().
4. void add\_score() – метод, начисляющий игроку очки и записывающий новое количество очков с помощью метода set\_score().

Также был написан makefile для удобной сборки программы.

Разработанный программный код см. в приложении А.

**Тестирование.**

Для тестирования программы был создан файл main.cc, в котором есть 3 функции:

1. int main() – создаёт объекты Player и Control, затем вызывает две других функции.
2. void test\_player(Player& player) – демонстрирует работу класса Player в различных ситуациях.
3. void test\_control(Control control) - демонстрирует работу класса Control в различных ситуациях.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Изменения | Комментарии |
| 1) | Player main\_player;  player.set\_health(2);  player.set\_score(10);  player.set\_x(4);  player.set\_y(4); | health\_ = 2  score\_ = 10  x\_ = 4  y\_ = 4 | Все методы в классе Player выполнены правильно. |
| 2) | Player main\_player;  Control controller(main\_player);  control.remove\_health();  control.add\_score(10); control.move(Directions::kDown); control.move(Directions::kRight); | health\_ = 2  score\_ = 10  x\_ = 1  y\_ = 1 | Все методы в классе Control выполнены правильно. |

## Вывод.

В ходе лабораторной работы были изучены классы и основы ооп в языке с++. Также были созданы необходимые классы, изучены способ передачи в поле объекта ссылки на другой объект, применение enum для задания направления в классе *Player*, добавлено несколько проверок в сеттерах приватных полей класса для сохранения инварианта.

**Приложение А**

Исходный код программы

Название файла: player.h

#ifndef OOP\_HEADERS\_PLAYER\_H\_

#define OOP\_HEADERS\_PLAYER\_H\_

class Player {

public:

int get\_health() const;

int get\_score() const;

int get\_x() const;

int get\_y() const;

void set\_health(int health);

void set\_score(int score);

void set\_x(int x);

void set\_y(int y);

void display() const;

Player();

private:

int health\_;

int score\_;

int x\_, y\_;

};

#endif

Название файла: player.cc

#include <iostream>

#include "headers/player.h"

int Player::get\_health() const { return health\_; }

int Player::get\_score() const { return score\_; }

int Player::get\_x() const { return x\_; }

int Player::get\_y() const { return y\_; }

void Player::set\_health(int health) {

health\_ = health;

if (health == 0) {

display();

std::cout << "You died!" << std::endl;

exit(0);

}

}

void Player::set\_score(int score) { score\_ = score; }

void Player::set\_x(int x) { x\_ = x; }

void Player::set\_y(int y) { y\_ = y; }

void Player::display() const {

std::cout << "Health: " << health\_

<< " Score: " << score\_ << std::endl;

}

Player::Player() {

health\_ = 3;

score\_ = 0;

x\_ = 0;

y\_ = 0;

}

Название файла: directions.h

#ifndef OOP\_HEADERS\_DIRECTIONS\_H\_

#define OOP\_HEADERS\_DIRECTIONS\_H\_

enum class Directions {

kUp,

kDown,

kLeft,

kRight

};

#endif

Название файла: control.h

#ifndef OOP\_HEADERS\_CONTROL\_H\_

#define OOP\_HEADERS\_CONTROL\_H\_

#include "player.h"

#include "directions.h"

class Control {

public:

void move(Directions direction);

void remove\_health();

void add\_score(int step\_score);

Control(Player& player);

private:

const int kStepMove = 1;

Player& player\_;

};

#endif

Название файла: control.cc

#include "headers/player.h"

#include "headers/directions.h"

#include "headers/control.h"

void Control::move(Directions direction) {

switch (direction) {

case Directions::kUp:

player\_.set\_y(player\_.get\_y() - kStepMove);

break;

case Directions::kDown:

player\_.set\_y(player\_.get\_y() + kStepMove);

break;

case Directions::kLeft:

player\_.set\_x(player\_.get\_x() - kStepMove);

break;

case Directions::kLeft:

player\_.set\_x(player\_.get\_x() + kStepMove);

break;

default:

break;

}

}

void Control::remove\_health() {

player\_.set\_health(player\_.get\_health() - 1);

}

void Control::add\_score(int step\_score) {

player\_.set\_score(player\_.get\_score() + step\_score);

}

Control::Control(Player& player) : player\_(player) {}

Название файла: main.cc

#include <iostream>

#include "headers/player.h"

#include "headers/directions.h"

#include "headers/control.h"

int main() {

Player main\_player;

Control controller(main\_player);

return 0;

}

Название файла: makefile

all: game clean

game: player control main

g++ player.o control.o main.o -o game

main: main.cc headers/player.h headers/directions.h headers/control.h

g++ -c main.cc

player: player.cc headers/player.h

g++ -c player.cc

control: Control.cc headers/player.h headers/directions.h headers/control.h

g++ -c control.cc

clean:

rm main.o player.o control.o