**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 |  | Войнов А.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

**Цель работы.**

Изучить классы и основы ооп в с++, научиться создавать их, объявлять поля и методы в языке. Разобраться в работе модификаторов доступа, научиться выбирать походящие.

**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 очков жизни, координаты и скорость.
2. PlayerController – класс, отвечающий за перемещение игрока по полю.
3. Sides – enum, содержащий направления перемещения (вверх, вниз, влево и вправо).

Все классы объявлены в соответствующих заголовочных файлах, а их реализация представлена в .cpp файлах. Исходный код см. в приложении А.

В классе Player созданы следующие поля:

1. int speed\_ - скорость игрока, то есть количество клеток, которые игрок может пройти за раз. В конструкторе инициализируется значением константы START\_SPEED.
2. int health\_points\_ - количество очков здоровья. Если это значение становится меньшим или равным 0, то игра заканчивается. Имеет максимальное значение равное значению MAX\_HEALTH\_POINTS. В конструкторе инициализируется значением константы MAX\_HEALTH\_POINTS.
3. int score\_ - количество очков, которые набрал игрок, используются для обозначения успешности прохождения игры. В конструкторе класса инициализируется нулём. Значение может быть, как отрицательным, так и положительным и не имеет ограничение на размер.
4. int x\_coordinate\_ - координата персонажа по оси х. В конструкторе инициализируется значением константы START\_X.
5. int y\_coordinate – координата персонажа по оси y. В конструкторе инициализируется значением константы START\_Y.

Также в классе были объявлены константы. Они нужны для удобства разработки. Если придётся менять одно из постоянных значений во всех частях кода, достаточно будет поменять значение константы.

1. const int MAX\_SPEED = 5 – константа, обозначающая максимальную скорость.
2. const int MAX\_HEALTH\_POINTS = 10 – константа, обозначающая максимальное количество очков здоровья.
3. const int START\_X = 0 - константа, обозначающая положение персонажа по оси х в начале игры.
4. const int START\_Y = 0 - константа, обозначающая положение персонажа по оси y в начале игры.
5. const int START\_SPEED = 1 - константа, обозначающая начальную скорость персонажа игрока.

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\_points\_, speed\_, x\_coordinate\_ и y\_coordinate\_ равными соответствующим константам, а score\_ равным 0.
2. int set\_health\_points(int health\_points) – сеттер для поля health\_points. Если переданное значение меньше или равно 0, то количество очков здоровья устанавливается равным 0 и возвращается 1. Это означает, что игра окончена, иначе возвращает 1. Если количество здоровья больше максимального, то в health\_points\_ будет сохранено значение максимального количества здоровья.
3. int set\_speed(int speed) – сеттер для скорости перемещения игрока. Проверяет, что передано положительное число, иначе возвращает -1 и не изменяет скорость игрока. В случае, если переданное число больше максимальной скорости, оно приравнивается к максимальному её значению. Если скорость изменена успешно возвращает 0.
4. Остальные метода – сеттеры и геттеры соответствующих полей, названные в соответствии с рекомендациями из google c++ style guide.

В классе PlayerController создано одно поле player\_, содержащее указатель на объект класса Player. В классе реализованы следующие методы:

1. PlayterController(Player \*player\_) – конструктор, принимающий указатель на объект игрока и сохраняющий его в соответствующем поле.
2. int move(Sides side) – метод, отвечающий за перемещение игрока по полю. В качестве аргумента принимает значение enum`а Sides и перемещает игрока в заданном направлении на количество клеток, равное скорости игрока. Для этого в переменных x и y сохраняются координаты игрока, а в speed скорость. Далее с помощью конструкции switch-case изменяется одна из координат. С помощью метода set\_coordinates() класса Player изменённые данные сохраняются в полях объекта игрока. В случае успешного выполнения возвращает 0. В дальнейшем метод будет возвращать разные код ошибки -1 в случае, если игрок пытается выйти за пределы поля.
3. int add\_health\_points(int health\_change) – вызывает сеттер поля health\_points\_ класса Player, с его помощью увеличивает количество очков здоровья на health\_change единиц. В случае если количество очков здоровья после изменения становится равно 0, выводит фразу “game over”, это тестовая обработка окончания игры, которая в будущем будет заменена, и возвращает 1. Иначе возвращает 0.
4. void add\_score(int score\_change) – вызывает сеттер поля score\_ класса Player, с его помощью увеличивает количество очков игрока на score\_change.
5. std::pair<int,int> get\_coordinates() – метод обращается к координатам х и у в объекте класса игрока и возвращает их, обёрнутыми в класс pair стандартной библиотеки. Pair – контейнер, содержащий 2 объекта любых типов, которые задаются с помощью шаблона.

**Тестирование**

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

1. int main() – создаёт объекты Player и PlayerController, затем вызывает две других функции.
2. void testPlayer(Player \*player) – демонстрирует работу класса Player в различных ситуациях.
3. void testPlayerController(PlayerController player\_controller) - демонстрирует работу класса PlayerController в различных ситуациях.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Изменения | Комментарии |
| 1) | Player \*player = new Player();  player->health\_points(); | health\_points\_=10  return code 0 | Вызванный конструктор инициализировал поле health\_points\_ значением 10. |
| 2) | player->set\_speed(10) | speed\_=5  return code 0 | Скорость не может стать больше максимальной |
| 3) | player->set\_speed(-10) | speed\_=5  return code -1 | Скорость не может быть отрицательной, поэтому она не изменилась. Метод вернул код -1, это значит, что были переданы неверные данные |
| 4) | PlayerController player\_controller = PlayerController(\*player);  player\_controller.move(Sides::DOWN);  player\_controller.move(Sides::RIGHT);  player\_controller.move(Sides::UP);  player\_controller.move(Sides::LEFT); | Координаты игрока:  x y  0 0  0 2  2 2  2 0  0 0 | Персонаж игрока прошёл по квадрату. |
| 5) | player\_controller.add\_health\_points(10); | health\_points\_=10  return code 0 | Количество очков здоровья не может стать больше максимального. |
| 6) | player\_controller.add\_health\_points(-2); | health\_points\_=8  return code 0 | Количество очков здоровья уменьшилось на 2. |
| 7) | player\_controller.add\_health\_points(-20); | health\_points\_=0  return code 1 | Количество очков здоровья не может упасть ниже нуля.  Метод вернул код 1, что означает поражение игрока. |

Вывод:

Были изучены классы и основы ооп в языке с++. Созданы классы с соответствующими полями, методами и модификаторами доступа для них, удовлетворяющие заданию лабораторной работы.

**Приложение А**

Исходный код программы

Название файла: Player.h

#ifndef PLAYER\_H

#define PLAYER\_H

#include "Sides.h"

#include <string>

#include <iostream>

class Player {

public:

Player();

int health\_points();

int set\_health\_points(int health\_points);

int score();

void set\_score(int score);

int x\_coordinate();

int y\_coordinate();

void set\_coordinates(int x, int y);

void set\_x\_coordinate(int x);

void set\_y\_coordinate(int y);

int speed();

int set\_speed(int speed);

private:

int speed\_;

int health\_points\_;

int score\_;

int x\_coordinate\_;

int y\_coordinate\_;

const int MAX\_SPEED = 5;

const int MAX\_HEALT\_POINTS = 10;

const int START\_X = 0;

const int START\_Y = 0;

const int START\_SPEED = 1;

};

#endif

Название файла: Player.cpp

#ifndef PLAYER\_CPP

#define PLAYER\_CPP

#include "../headers/Player.h"

Player::Player() {

health\_points\_ = MAX\_HEALT\_POINTS;

x\_coordinate\_ = START\_X;

y\_coordinate\_ = START\_Y;

score\_ = 0;

speed\_ = START\_SPEED;

}

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

if(health\_points <= 0) {

health\_points\_ = 0;

return 1;

}else if(health\_points >= MAX\_HEALT\_POINTS) {

health\_points = MAX\_HEALT\_POINTS;

}

health\_points\_ = health\_points;

return 0;

}

int Player::set\_speed(int speed) {

if(speed <= 0) {

return -1;

}else if(speed >= MAX\_SPEED) {

speed\_ = MAX\_SPEED;

return 0;

}

speed\_ = speed;

return 0;

}

void Player::set\_score(int score) {

score\_ = score;

}

void Player::set\_y\_coordinate(int y) {

y\_coordinate\_ = y;

}

void Player::set\_x\_coordinate(int x) {

x\_coordinate\_ = x;

}

void Player::set\_coordinates(int x, int y) {

x\_coordinate\_ = x;

y\_coordinate\_ = y;

}

int Player::x\_coordinate() {

return x\_coordinate\_;

}

int Player::y\_coordinate() {

return y\_coordinate\_;

}

int Player::speed() {

return speed\_;

}

int Player::health\_points() {

return health\_points\_;

}

int Player::score() {

return score\_;

}

#endif

Название файла: PlayerController.h

#ifndef PLAYER\_CONTROLLER\_H

#define PLAYER\_CONTROLLER\_H

#include "Player.h"

#include "Sides.h"

#include <utility>

class PlayerController {

public:

PlayerController(Player \*player);

int move(Sides side);

int add\_health\_points(int health\_change);

void add\_score(int score\_change);

std::pair<int, int> get\_coordinates();

private:

Player \*player\_;

};

#endif

Название файла: PlayerController.cpp

#ifndef PLAYER\_CONTROLLER\_CPP

#define PLAYER\_CONTROLLER\_CPP

#include "../headers/PlayerController.h"

PlayerController::PlayerController(Player \*player) {

player\_ = player;

}

void PlayerController::add\_score(int score\_change){

player\_->set\_score(player\_->score() + score\_change);

}

int PlayerController::move(Sides side) {

int speed = player\_->speed();

int x = player\_->x\_coordinate();

int y = player\_->y\_coordinate();

switch (side) {

case Sides::UP:

y -= speed;

break;

case Sides::DOWN:

y += speed;

break;

case Sides::LEFT:

x -= speed;

break;

case Sides::RIGHT:

x += speed;

break;

}

player\_->set\_coordinates(x, y);

return 0;

}

int PlayerController::add\_health\_points(int health\_change) {

if(player\_->set\_health\_points(player\_->health\_points() + health\_change) == 1) {

std::cout << "game over\n";

return 1;

}

return 0;

}

std::pair<int, int> PlayerController::get\_coordinates() {

return std::make\_pair(player\_->x\_coordinate(), player\_->y\_coordinate());

}

#endif

Название файла: Sides.h

#ifndef SIDES\_H

#define SIDES\_H

enum Sides {

UP, DOWN, LEFT, RIGHT

};

#endif

Название файла: main.cpp

#include "../headers/PlayerController.h"

#include <iostream>

#include <cassert>

void testPlayer(Player \*player) {

int error\_code;

std::cout << "health points on start: " << player->health\_points() << '\n';

error\_code = player->set\_speed(10);

std::cout << "player took powerUp so his speed increased: " << player->speed() << '\n'<<

"set\_speed returns code " << error\_code << "\n\n";

error\_code = player->set\_speed(-10);

std::cout << "something went wrong and set\_speed() got argument under 0, but speed would not change: " << player->speed() << '\n' <<

"set\_speed returns code " << error\_code << "\n\n";

error\_code = player->set\_speed(2);

std::cout << "and finally his speed is " << player->speed() << "\n\n\n";

}

void testPlayerController(PlayerController player\_controller, Player \*player) {

int error\_code;

std::cout << "player walked across the square\n";

std::cout << player\_controller.get\_coordinates().first << ' ' << player\_controller.get\_coordinates().second << '\n';

player\_controller.move(Sides::DOWN);

std::cout << player\_controller.get\_coordinates().first << ' ' << player\_controller.get\_coordinates().second << '\n';

player\_controller.move(Sides::RIGHT);

std::cout << player\_controller.get\_coordinates().first << ' ' << player\_controller.get\_coordinates().second << '\n';

player\_controller.move(Sides::UP);

std::cout << player\_controller.get\_coordinates().first << ' ' << player\_controller.get\_coordinates().second << '\n';

player\_controller.move(Sides::LEFT);

std::cout << player\_controller.get\_coordinates().first << ' ' << player\_controller.get\_coordinates().second << '\n';

error\_code = player\_controller.add\_health\_points(10);

std::cout << "\nafter add 10 hp nothing changed because player has full hp: " << player->health\_points() << '\n' <<

"method returns code " << error\_code << "\n\n";

error\_code = player\_controller.add\_health\_points(-2);

std::cout << "player was careless and got damage so his hp decreased " << player->health\_points() << '\n'<<

"method returns code " << error\_code << "\n\n";

error\_code = player\_controller.add\_health\_points(-20);

std::cout << "Player got a lot of damage, so his hp is " << player->health\_points() << '\n'<<

"method returns code " << error\_code << "\n\n";

}

int main() {

Player \*player = new Player();

PlayerController player\_controller = PlayerController(player);

testPlayer(player);

testPlayerController(player\_controller, player);

}