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, конструкторов и методов классов.

Студент гр. 1381 Смирнов Д. Ю.

Преподаватель Жангиров Т. Р.

Санкт-Петербург
2022

Цель работы.

Изучить основы объектно-ориентированного программирования, научиться реализовывать простые классы и связывать их между собой.

Задание.

Реализовать прямоугольное игровое поле, состоящее из клеток. Клетка - элемент поля, которая может быть проходима или нет (определяет, куда может стать игрок), а также содержит какое-либо событие, которое срабатывает, когда игрок становится на клетку. Для игрового поля при создании должна быть возможность установить размер (количество клеток по вертикали и горизонтали). Игровое поле должно быть зациклено по вертикали и горизонтали, то есть если игрок находится на правой границе и идет вправо, то он оказывается на левой границе (аналогично для всех краев поля).

Реализовать класс игрока. Игрок - сущность, контролируемая пользователем. Игрок должен иметь свой набор характеристик и различный набор действий (например, разные способы перемещения, попытка избежать событие, и так далее).

Требования:

- Реализован класс игрового поля
- Для игрового поля реализован конструктор с возможностью задать размер и конструктор по умолчанию (то есть конструктор, который можно вызвать без аргументов)
- Реализован класс интерфейс события (в данной лабораторной это может быть пустой абстрактный класс)
- Реализован класс клетки с конструктором, позволяющим задать ей начальные параметры.
- Для клетки реализованы методы реагирования на то, что игрок перешел на клетку.
- Для клетки реализованы методы, позволяющие заменять событие. (То есть клетка в ходе игры может динамически меняться)

- Реализованы конструкторы копирования и перемещения, и соответствующие им операторы присваивания для игрового поля и при необходимости клетки
- Реализован класс игрока минимум с 3 характеристиками. И соответствующие ему конструкторы.
- Реализовано перемещение игрока по полю с проверкой допустимости на переход по клеткам.

Примечание:

- При написании конструкторов учитывайте, что события должны храниться по указателю для соблюдения полиморфизма
- Для управления игроком можно использовать медиатор, команду, цепочку обязанностей

Выполнение работы.

Классы:

1. *Cell* — класс клетки. Имеет публичное поле CellType – перечисление всех видов клеток

Поля:

CellType type — тип клетки (что находится на ней)

Event* event — указатель на объект события, который в следующей лабораторной работе будет срабатывать

Методы:

Cell() — конструктор, инициализирует поле type

void set_type(CellType type1) — метод позволяющий задать тип клетки.

CellType get_type() const — позволяет получить тип клетки

void set_event(Event* event) — устанавливает событие

void update(Player& player) — вызывает у события виртуальный метод Execute(), если событие есть на клетке

2. *Event* — абстрактный класс(интерфейс) — класс, от которого будут наследоваться другие события (понадобится для следующей лабораторной работе).

Методы:

1) virtual void Execute() = 0 — метод запускающий событие.

2) virtual ~Event() = 0 — деструктор.

3. *Field* — класс поля

Поля:

std::vector<std::vector<Cell>> field — двумерный массив, состоящий из объектов Cell

int height — высота

int width — ширина

std::pair<int, int> player_position — координаты игрока

Методы:

explicit Field(int a = 10, int b = 10) — конструктор, инициализирующий все поля класса

Field(const Field &other) — конструктор копирования

void swap(Field& other) — меняет поля классов местами

Field& operator=(const Field &other) — оператор присваивания. Реализован с помощью конструктора копирования и метода swap

Field(Field&& other) — конструктор перемещения. В реализации использовался метод swap

Field& operator=(Field&& other) — оператор присваивания перемещения, реализован с использовался метода swap

void generate_field() — случайным образом генерирует тип клеток

void change_player_position(Player::Directions direction) — меняет позицию игрока, если на клетку можно наступить

int get_height() const, int get_width() const, std::vector<std::vector<Cell>> get_field() const - геттеры полей

void check_position(std::pair<int, int> pair) — проверяет клетку на проходимость

4. *CellView* — определяет, как клетка выглядит

Поля:

char cell_view — символ вида клетки

Методы:

explicit CellView(const Cell& cell) — конструктор, основываясь на типе клетки заполняет поле cell_view

char get_view() const — возвращает поле cell_view

5. *FieldView* — отвечает за вывод поля

Методы:

void write_field(const Field& field) const — выводит поле, для обработки использует класс CellView

void write_horizontal_border(int width) const — выводит горизонтальную границу

6. *Player* — класс игрока.

Поля:

int hearts, int power, int coins - поля характеристики игрока

enum Directions – перечисление вариантов шагов игрока

Методы:

explicit Player(int hearts = 3, int power = 1) — конструктор, заполняет поля характеристик игрока

int get_hearts() const, int get_power() const, int get_coins() const — геттеры

void set_hearts(int heart), void set_power(int dmg), void set_coins(int coin) — сеттеры

7. *Game* — основной класс игры, отвечает за её запуск и зацикливает игру

Поля:

Field field — содержит игровое поле

FieldView field_view — содержит класс, отвечающий за отображение игрового поля

Mediator mediator — содержит класс прослойку между контроллером и считывателем команд.

Методы:

Game() — конструктор класса, инициализирует поля field_view и mediator, затем использует медиатор для заполнения конструктора игрового поля.

Start() — начинает игру и зацикливает её.

void reaction(Player::Directions move) — метод вызывает изменение позиции игрока и вызывает отрисовку игрового поля.

8. *Mediator* — класс посредник, обрабатывающий запросы главного класса *Game*

Поля:

CommandReader reader — объект класса отвечающего за пользовательский ввод

Методы:

Mediator() — инициализирует поле

std::pair<int, int> field_size() — метод выполняет последовательность команд направленных на получение размера игрового поля.

Directions move() — возвращает какой шаг был шаг сделан пользователем.

9. CommandReader — класс, считывающий данные

Методы:

int read_number() const — считывает число с консоли

Directions get_step() const — считывает char из консоли и преобразует к одному из вариантов из перечисления

Использовался паттерн объектно-ориентированного программирования MVC (Model-View-Controller).

Контроллер игры (class Game) получает ход игрока из Mediator’a и реагирует на это изменяя Модель поля (class Field), посредством вызова соответствующего метода change_player_position. После изменения Модели используется Представление(class FieldView), который обращаются к Модели за обновленными её данными, после чего их и отображает.

UML диаграмма классов представлена в приложение А.

Выводы.

Были изучены основы объектно-ориентированного программирования. В ходе лабораторной работы были созданы классы, отвечающие за игрока, клетки поля, поле, их вывод и взаимодействие пользователя с игрой.

ПРИЛОЖЕНИЕ А

UML диаграмма классов