**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. 1303 |  | Дауди И. Я. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2022

**Цель работы:**

Изучить основы ООП, 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.

Примечания:

* 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:

*Cell:*

Поля:

*CellType type —* тип клетки (Player, Wall, Coin, Standard).

*Event\* event* — указатель на объект события, который будет сделано в следующей лабораторной работе.

Методы:

*Cell():* конструктор, инициализирует тип поля стандартным по умолчанию, а событие — nullptr.

*void set\_type(CellType type):* принимает желаемый тип клетки в качестве аргумента и создает её.

*CellType get\_type() const:*  возвращает тип клетки.

*void set\_event(Event\* event)* — принимает указатель на событие и устанавливает его.

*void update(Player& player)* — если игрок находится на ячейке и у ячейки есть событие, событие вызывает функцию "Execute" из класса Event.

*CellView:*

определяет, как выглядит клетка.

Поля:

*char cell\_view:* символ, который будет представлять клетку.

Методы:

*explicit CellView(const Cell& cell):* конструктор, принимает объект Cell в качестве аргумента и с помощью оператора switch записывает в cell\_view символ, который он должен содержать.

*char get\_view() const* : возвращает значение атрибута cell\_view.

*Field:*

Поля:

*int height:* высота игрового поля.

*int width:* ширина игрового поля.

pair<int, int> player\_position: координаты игрока.

vector<vector<Cell>> field: игровое поле.

Методы:

*explicit Field(int a = 10, int b = 10):*  конструктор, принимает высоту и ширину в качестве аргументов, инициализирует поле и позицию игрока в {0,0} по умолчанию.

*void swap(Field& other)* : меняет поля классов местами

*Field(const Field &other)* : конструктор копирования

*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():* Помещает игрока в координаты {0,0}, затем создает клетки поля, по умолчанию все клетки стандартные, затем, используя генератор случайных чисел и оператор switch, перезаписывает некоторые клетки на стену или монету.

*void change\_player\_position(Player::Directions direction)*: Принимает направление игрока в качестве аргумента и с помощью оператора switch изменяет положение игрока. Затем проверяет, находится ли игрок на границе поля, и помещает его в нужное место. Затем проверяет, пытается ли игрок стоять у стены, и позволяет ему двигаться или просто остается на месте.

*int get\_height() const*,

*int get\_width()const*,

*std::vector<std::vector<Cell>> get\_field() const:*

возвращают атрибуты класса .

*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.

*Player:*

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.

*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.

*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.

*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.

**Вывод:**

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