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.

Студентка гр. 1304 _____ Чернякова В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

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.

Описание архитектурных решений и классов.

Классы.

Класс *Player*: реализован класс, отвечающий за характеристики игрока. Поля *health, food, score, resource* – задают основные черты игрока. В методах реализован конструктор *Player(),* в котором задаются значения соответствующих полей: *10, 5, 10, 0*. Для удаления объектов также присутствует деструктор *~Player()*. Для получения значения характеристик игрока в процессе игры созданы одноименные методы-геттеры с приставкой *get*, методы являются константными, так как внутри них никаких изменений не происходит, они постоянны.

Класс *Cell*: реализован класс клетки, который отвечает за характеристику каждой клетки, составляющей поле. Присутствуют два поля *active* и *barrier*. Первое отвечает за активность клетки, то есть наступил ли на нее игрок. Второе является проверкой: проходима ли клетка. Реализован конструктор по умолчанию *Cell() = default*, обычный конструктор *Cell(bool active, bool barrier)*, в котором значения каждого поля *false* и деструктор *~Cell()* для удаления объекта класса. Также созданы геттеры и сеттеры для получения значения. Сеттер для *active* определяет – активная ли клетка на данный момент *this->active ? this->active = false : this->active = true* если да, то она становится не активной, иначе активной.

Класс *Field*: создан класс поля, реализующий полностью игровое поле. Поля *height, width* – отвечают за его размеры, *cell_x* и *cell_y* – положение клетки на поле и *Cell** *cells* – двумерный массив из клеток, то есть само поле. Конструкторы *Field(int height, int width)* от значений, вводимых пользователем с клавиатуры и *Field()* - по умолчанию поле *10* на *10*. В теле конструктора

создается поле из клеток: с помощью функции *rand()* на клетку устанавливается барьер, до этого проверяя, что мы не находимся на крайних позициях поля, чтобы игрок гарантированно мог ходить. Клетка *cell[0][0]* становится активной, так как на нее помещается игрок. Конструктор копирования *Field::Field(const Field& field),* конструктор перемещения *Field::Field(Field&& field) noexcept*, основанный на *move* семантике. Также происходит переопределение оператора присваивания *Field& Field::operator=(const Field& field)* для осуществления копирования и перемещения соответственно. Деструктор для очищения *Field::~Field()*. Реализованы соответствующие геттеры и сеттеры. Методы передвижения *moveUp/Down/Left/Right*(передается шаг, который надо совершить, который равен единице): предыдущая клетка становится неактивной, координата новой клетки рассчитывается по следующей формуле: *(height/width + cell_y/cell_x -/+ step)%height/width* – это гарантирует зацикливание поля, для клетки с новыми координатами вызывается метод клетки *setActive()*, и она становится активной.

Класс *CellView*: класс обрисовки клетки поля. Представлен только методами – конструктор по умолчанию *CellView() = default* и *void printCell(Cell cell)*. *printCell()* проверяет, является ли клетка активной – тогда выводится *"|P|"*, то есть игрок, иначе проверяется, есть ли барьер с помощью метода клетки *getBarrier()*, если да рисуется *"|R|"*, иначе *"| |"*.

Класс *FieldView*: класс обрисовки самого поля, состоящего из клеток. Для работы с консолью создано поле - *HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE)*. Методы: конструктор по умолчанию *FieldView() = default* и сами методы обрисовки *void printLine(int width)*, *void printField(Field& field)*. *printLine()* выводит рамку поля и разграничивает клетки. *printField()*: так как в ходе игры метод вызывается несколько раз и поле рисуется заново после изменений(например, передвижений), для очистки используется *system("cls")*, далее создается объект класса *CellView,* и в ходе

вывода поля вызывается соответствующий метод *printCell()* для правильной обрисовки клетки в зависимости от ее свойств.

Класс *Enum class Direction*: в нем объявлены возможные команды игры, такие как передвижение и выход – *LEFT, RIGHT, UP, DOWN, EXIT, NONE*.

Класс *CommandReader:* класс считывания команд пользователя с клавиатуры. Представлен только методами. Считывание высоты и ширины поля: *getFieldHeight()* и *getFieldWidth()*. Считывание команды-символа: *getDirection()* метод возвращает значения типа класса *Durection*, считывание происходит за счёт функции *_getch()*, которая ожидает нажатия клавиши, после которого она немедленно возвращает значение.

Касс *Controller:* класс обработки считанных *CommandReader* команд. Представлен только методами. *getFieldHeight()* и *getFieldHeight()* – проверка корректности введенных значений с клавиатуры для поля. Перемещение игрока *movePlayerUp/Down/Left/Right()*, в нем происходит проверка возможности перемещения игрока, так осуществляется оценка проходимости клетки, то есть не установлен ли на ней барьер, если нет, то происходит перемещение игрока с помощью метода поля *field.moveUp/Down/Left/Right(),* в конце метода обрисовывается поле после внесенных изменений. *getAction()* в зависимости от полученного значения *Direction* осуществляет либо перемещение либо выход из игры.

Класс *Application*: класс создания игры. Определен полем *game*, отвечающем за статус игры: *true* или *false*. Реализованы методы. Конструктор, в котором значение поля *game = true,* то есть игра идет, и деструктор. *start()* осуществляет создание игры и начало игрового процесса. Создаются объекты классов *Controller, CommandReader, Field, FieldView* и *Player*. С помощью метода класса *FieldView printField()* выводится обрисованное поле. Пока значения поля *game* имеет значение *true*: с помощью метода *Controller getAction()* принимаются и обрабатываются действия, вводимые

пользователем, далее поле обрисовывается и выводится заново *printField()* на основе обработанных действий.

Файл Game.cpp: запуск игры. Создается объекта класс *Application* – *application*. У данного объекта вызывается метод *start()* – игра началась.

ПРИЛОЖЕНИЕ.

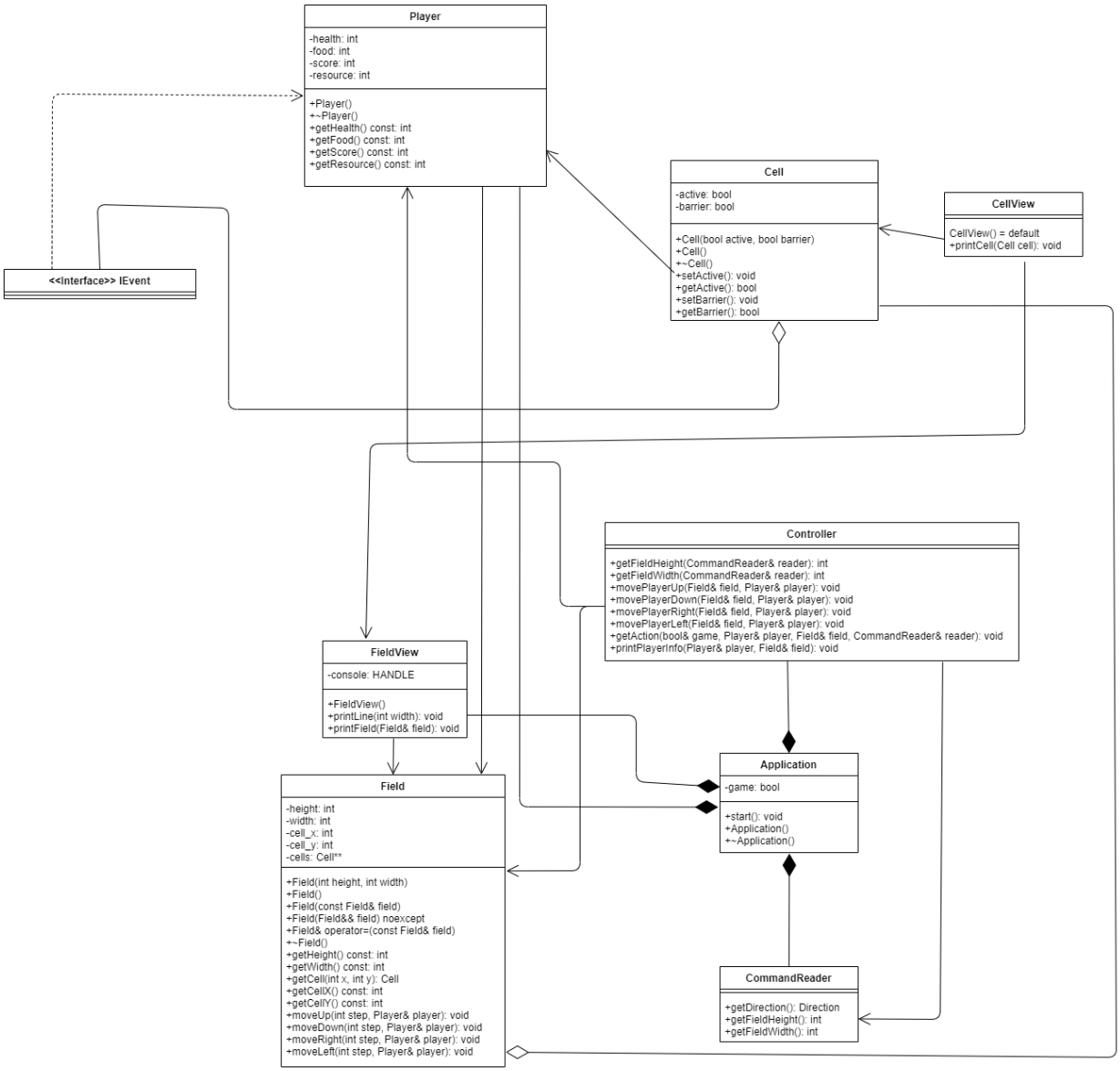

Рисунок 1 – UML диаграмма классов.