МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Конструкторы и деструкторы

Студент гр. 2300 _____ Жохов К.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2023

Цель работы.

Изучить понятия «конструктор» и «деструктор», получить навыки их написания. Реализовать конструкторы копирования и перемещения. Разработать классы клетки и поля, в класс управления игроком добавить взаимодействие с полем.

Задание.

а) Создать класс клетки игрового поля. Клетка игрового поля может быть проходимой или нет, тем самым определяя возможность игрока встать на эту клетку. Возможность задать проходимость клетки должна быть реализована через конструктор и через метод клетки. В будущем в клетке будет храниться указатель на интерфейс события.

б) Создать класс игрового поля. Игровое поле представляет собой прямоугольник из клеток (двумерный массив). В учебных целях, клетки хранятся как чистый массив на указателях (использовать контейнеры stl запрещено в этой лаб. работе). Размер поля передается в конструктор поля, в котором динамически выделяется память под массив клеток. Также должна быть возможность вызвать конструктор поля без аргументов. Так как происходит выделение память, то необходимо реализовать деструктор в котором будет происходить очистка память.

Также добавить в игровое поле добавьте информацию о входе (где в начале появляется игрок) и выходе (куда игрок должен дойти)

в) В класс управления игрока добавить взаимодействия с полем. При перемещении игрока должна быть проверка на проходимость клетки, если клетка непроходима, то перемещение не должно производиться.

Выполнение работы.

1. Класс `Cell`. Представляет собой клетку игрового поля, приватные поля `passability` и `event` содержат соответственно вид клетки (проходимая или нет) и указатель на игровое событие.

- Конструктор `Cell(bool passable = true , Interface *event = nullptr)`. Модификатор доступа – `public`. Принимает на вход необязательные параметры (потенциальные значения полей `passability` и `event`).

- Конструктор копирования `Cell(const Cell &other)`. Модификатор доступа – `public`. Принимает на вход константную ссылку на объект, из которого нужно скопировать данные. Копирует данные в создаваемый объект, оставляя объект-источник неизменным.

- Оператор присваивания с копированием `Cell& operator = (const Cell &other)`. Модификатор доступа – `public`. Принимает на вход константную ссылку на объект, из которого нужно скопировать данные. Создаётся временный объект-копия, с которым объект-приёмник обменивается данными. Возвращает ссылку на объект-прёмник.

- Конструктор перемещения `Cell(Cell &&other)`. Модификатор доступа – `public`. Принимает на вход rvalue объект, с которым происходит обмен данных.

- Оператор присваивания с перемещением `Cell& operator = (Cell &&other)`. Модификатор доступа – `public`. Принимает на вход rvalue объект, с которым происходит обмен данных. Возвращает ссылку на объект-приёмник.

- Константный метод `bool getPassability() const`. Модификатор доступа – `public`. Не принимает аргументов. Возвращает значение типа `bool` в зависимости оттого, является ли клетка проходимой.

- Метод `void setPassability(bool value)`. Модификатор доступа – `public`. Принимает на вход значение типа `bool`, которое должно быть установлено в поле `passability` клетки.

- Деструктор `~Cell()`. Модификатор доступа – `public`. Очищает память, выделенную для хранения игрового события.

2. Класс `Field`. Представляет собой игровое поле. В приватных полях `cell_arr, size, start, finish` хранится соответственно указатель на двумерный массив клеток, размер игрового поля, точка входа игрока, точка выхода игрока.

- Метод `bool checkSize(int width, int height)`. Модификатор доступа – `private`. Принимает на вход потенциальные размеры игрового поля, проверяет их валидность. Возвращает `true` или `false`.

- Конструктор `Field(int width = DEF_SIZE, int height = DEF_SIZE)`. Модификатор доступа – `public`. Принимает на вход потенциальные размеры игрового поля, передаёт на проверку функции `checkSize()`. Если функция проверки вернула `true`, инициализируются поля, ответственные за хранение размера поля, координат точки входа и точки выхода. Выделяется динамическая память для хранения двумерного массива клеток. В противном случае бросается ошибка `std::invalid_argument`.

- Метод `std::pair<int, int> getSize()`. Модификатор доступа – `public`. Не принимает аргументов, возвращает размер игрового поля.

- Метод `std::pair<int, int> getStart()`. Модификатор доступа – `public`. Не принимает аргументов, возвращает координаты точки входа игрока.

- Метод `std::pair<int, int> getFinish()`. Модификатор доступа – `public`. Не принимает аргументов, возвращает координаты точки выхода игрока.

- Метод `void setStart(int x_start, int y_start)`. Модификатор доступа – `public`. Принимает на вход потенциальные координаты новой точки входа игрока, проверяет их валидность. В зависимости от результата проверки будет установлена новая точка входа или выброшена ошибка `std::invalid_argument`.

- Метод `void setFinish(int x_finish, int y_finish)`. Модификатор доступа – `public`. Принимает на вход потенциальные координаты новой точки выхода игрока, проверяет их валидность. В зависимости от результата проверки будет установлена новая точка выхода или выброшена ошибка `std::invalid_argument`.

- Константный метод `bool checkCoordinates(int x, int y) const`. Модификатор доступа – `public`. Принимает на вход координаты точки, для которой возвращается `true` или `false` в зависимости оттого, находится точка внутри поля или нет.

- Константный метод `const Cell& getCell(int x, int y) const`. Модификатор доступа – `public`. Возвращает константную ссылку на клетку на основе полученных координат. Если переданные координаты не лежат внутри поля, бросается ошибка `std::out_of_range`.

- Метод `void setPassability(int x, int y, bool value)`. Модификатор доступа – `public`. Принимает на вход координаты клетки и значение, которое нужно ей установить. В зависимости оттого, лежит клетка внутри поля или нет, будет установлено переданное значение или выброшена ошибка `std::out_of_range`.

- Деструктор `~Field()`. Модификатор доступа – `public`. Очищается память, выделенная под двумерный массив клеток.

3. Класс `Controller`. Теперь содержит ещё одно private поле `field`, хранящее ссылку на игровое поле.

- Метод `void setCoordinates(int x, int y)`. Модификатор доступа – `public`. Принимает на вход новые потенциальные координаты игрока. Добавлена проверка на валидность координат и проходимость клетки. Если координаты лежат внутри поля и клетка с такими координатами проходима, то будут установлены новые координаты, в противном случае координаты останутся прежними.

4. Класс `Interface`. Представляет собой игровое событие. Содержит 2 чисто виртуальных `public` метода, которые будут переопределены в классах-наследниках в последующих лабораторных работах.

5. Класс `HealthEvent`. Наследник класса `Interface`. Содержит 2 переопределённых `public` метода, один из которых будет описан ниже.

- Метод `HealthEvent* clone()`. Модификатор доступа — `public`. Не принимает аргументов, возвращает указатель на динамическую память, выделенную под текущий объект класса. Требуется для осуществления копирования класса клетки.

Makefile. С целью сделать код более читаемым и для удобства модификации программа была разделена на смысловые части. Для сборки программы написан Makefile.

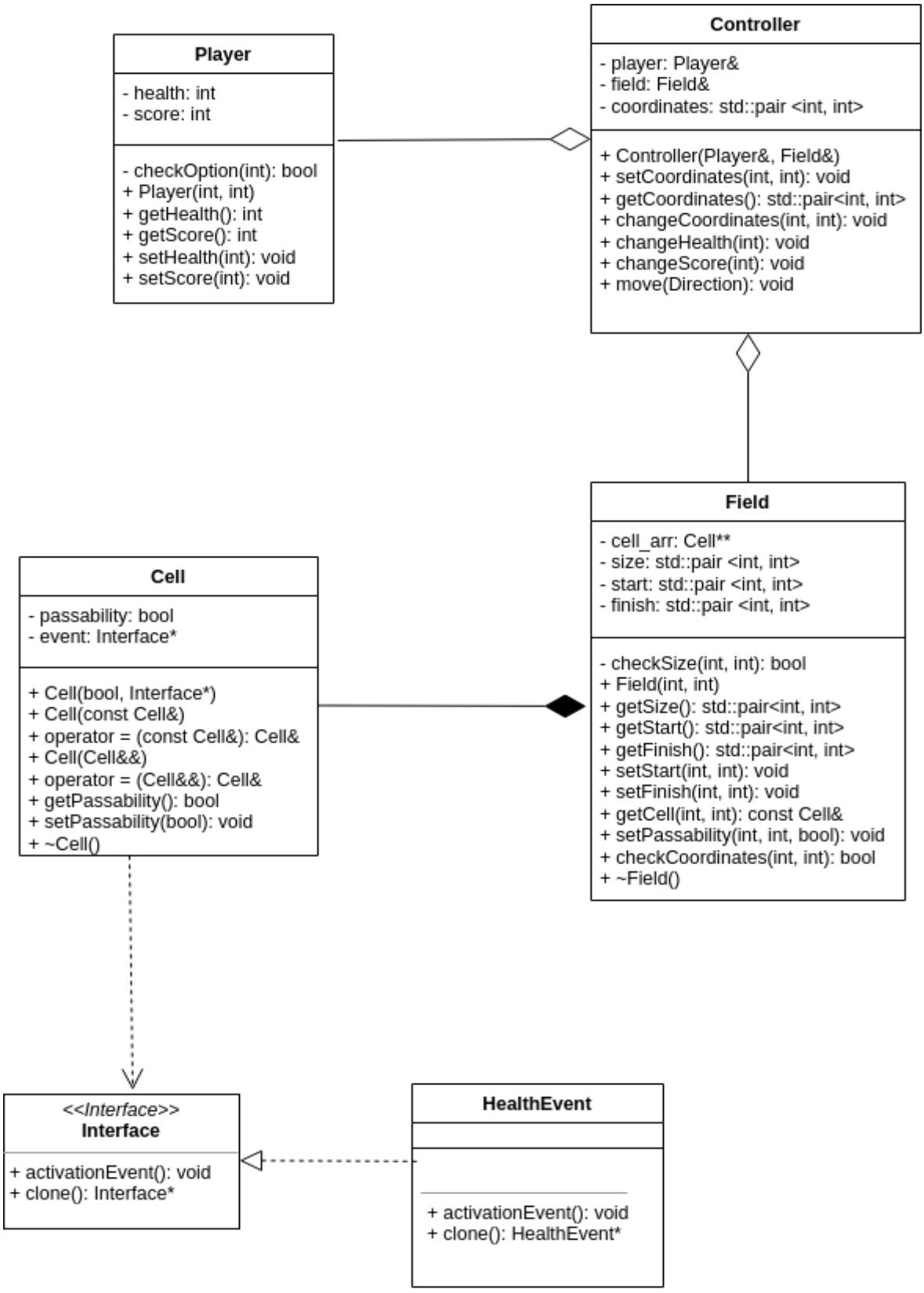
Разработанные UML-диаграммы классов см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе лабораторной работы разработаны классы клетки и игрового поля, добавлено взаимодействие класса управления игроком и игрового поля. Реализованы конструкторы копирования и перемещения. Положено начало разработке классов, отвечающих за игровые события.

ПРИЛОЖЕНИЕ А
UML-ДИАГРАММЫ КЛАССОВ



ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Interface *event = new HealthEvent; Cell cell1(true, event); std::cout << cell1.getPassability() << '\n'; cell1.setPassability(false); std::cout << cell1.getPassability() << '\n'; Cell cell2 = cell1; Cell cell3(Cell(false, new HealthEvent));	1 0	Проверка работоспособности метода setPassability(), копирования и перемещения

2.	Field field(15, 10);
	std::pair <int, int> field_size = field.getSize();
	field.setStart(3, 4);
	field.setFinish(9, 8);
	field.setPassability(5, 5, false);
	std::pair <int, int> entry = field.getStart();
	std::pair <int, int> exit = field.getFinish();
	std::cout << "size: " << field_size.first << '\t' << field_size.second << '\n';
	std::cout << "start: " << entry.first << '\t' << entry.second << '\n';
	std::cout << "finish: " << exit.first << '\t' << exit.second << '\n';
	std::cout << field.getCell(5, 5).getPassability() << '\n';
		size: 15 10
		start: 3 4
		finish: 9 8
		0
			Проверка работоспособности основополагающих методов класса игрового поля

3.	Player player; Field field; Controller controller(player, field); std::pair <int, int> field_size = field.getSize();	size: 10 10 start: 0 0 finish: 9 9 coordinates before 0 0 coordinates after 0 0	Проверка работоспособности методов класса игрового поля и

	std::pair <int, int> entry = field.getStart();
	std::pair <int, int> exit = field.getFinish();
	std::cout << "size: " << field_size.first << '\t' << field_size.second << '\n';
	std::cout << "start: " << entry.first << '\t' << entry.second << '\n';
	std::cout << "finish: " << exit.first << '\t' << exit.second << '\n';
	std::pair <int, int> crds1 = controller.getCoordinates();
	std::cout << "coordinates before " << crds1.first << '\t' << crds1.second << '\n';
	controller.move(Direction::down);
	std::pair <int, int> crds2 = controller.getCoordinates();
	std::cout << "coordinates after " << crds2.first << '\t' << crds2.second << '\n';
			класса управления игроком

4.	Field field(88, -99);	Invalid size of playing field!	Проверка обработки исключений
5.	Field field(10, 10); field.setStart(10, 8);	Entry coordinates should be inside the playing field!	Проверка обработки исключений
6.	Field field(10, 10); field.setPassability(5, 10);	Invalid coordinates There is no cell with such coordinates	Проверка обработки исключений
7.	Field field(10, 10); field.getCell(11, 6);	Invalid coordinates There is no cell with such coordinates	Проверка обработки исключений