**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: **Отслеживание изменений**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2303 |  | Волков И.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

**Цель работы.**

Создание класса, который будет отслеживать все изменения в игре и направлять их туда, куда нужно, вызывать нужные методы специализированных классов и т.д.

**Задание.**

а) Реализовать класс, который связывается с игрой, и отслеживает изменения в игре: перемещение игрока, победа или выигрыш, срабатывание событий. Данный класс должен реагировать на изменения и отрисовывать игровое поле, а также выводить информацию для игрока (например, предлагать начать новую игру).

б) При отрисовке поля должна считываться информация с поля и об игроке, и в зависимости от расположения происходит вывод представления поля в терминал. В представлении поля непроходимые клетки, игрок, события должны отображаться различными символами. Игрок, события, клетки и другие игровые сущности не должны знать ничего о том, каким символом они отрисовываются. За выбор символа отвечает класс выполняющий отрисовку

Примечания:

Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания

При проверке типов события можно использовать dynamic\_cast / typeinfo, либо сделать связывание модель-представление

**Выполнение задания.**

Созданные классы: lvl, I\_Visualise, Visualise, Menu

Menu реализует взаимодействие с игроком в меню – это выбор уровня и выход из игры. В классе только один метод – std::string MainMenu() – возвращает путь к папке с данными об уровне или пустую строку, если игрок решил закрыть игру.

Класс lvl – создаёт уровень, запускает игру, отслеживает изменения, в данной работе изменения – это ввод пользователем каких-либо команд, исполнение их и вызов отрисовки.

Поля:

* Player player – объект игрока
* Field field – объект игрового поля
* Controller contr – объект контроллера
* I\_PlyerInput pi – объект класса пользовательского ввода
* I\_Visualise v – объект класса визуализации поля
* std::map<int, Controlls> crls – словарь с командой и её кодовым представлением
* std::map<Symbols, char> visuals – словарь с символами на понятном программе языке и их версией для отрисовки для пользователя
* long long frameRate – время, которое должно пройти между 2-мя шагами (2-мя кадрами отрисовки)

Методы:

* lvl(std::string path, long long frameRate) – при инициализации на вход подаётся положение папки с файлами уровня и предполагаемое время кадра в миллисекундах, здесь же файлы читаются и данные записываются в поля
* void start() – запускает уровень (отслеживает пользовательский ввод, запускает отрисовку)
* ~lvl() – удаляет данные из полей

I\_Visualise – интерфейс для объекта вывода информации для пользователя.

Абстрактные методы:

* virtual void printField() = 0 – выводит игровое поле каждую игровую итерацию
* virtual void GameOver(int steps) = 0 – выводит информацию при поражении игрока
* virtual void Win(int steps) = 0 – выводит информацию при победе игрока

Visualise – является реализацией интерфейса I\_Visualise, выводит к консоль игровое поле и информацию по игроку.

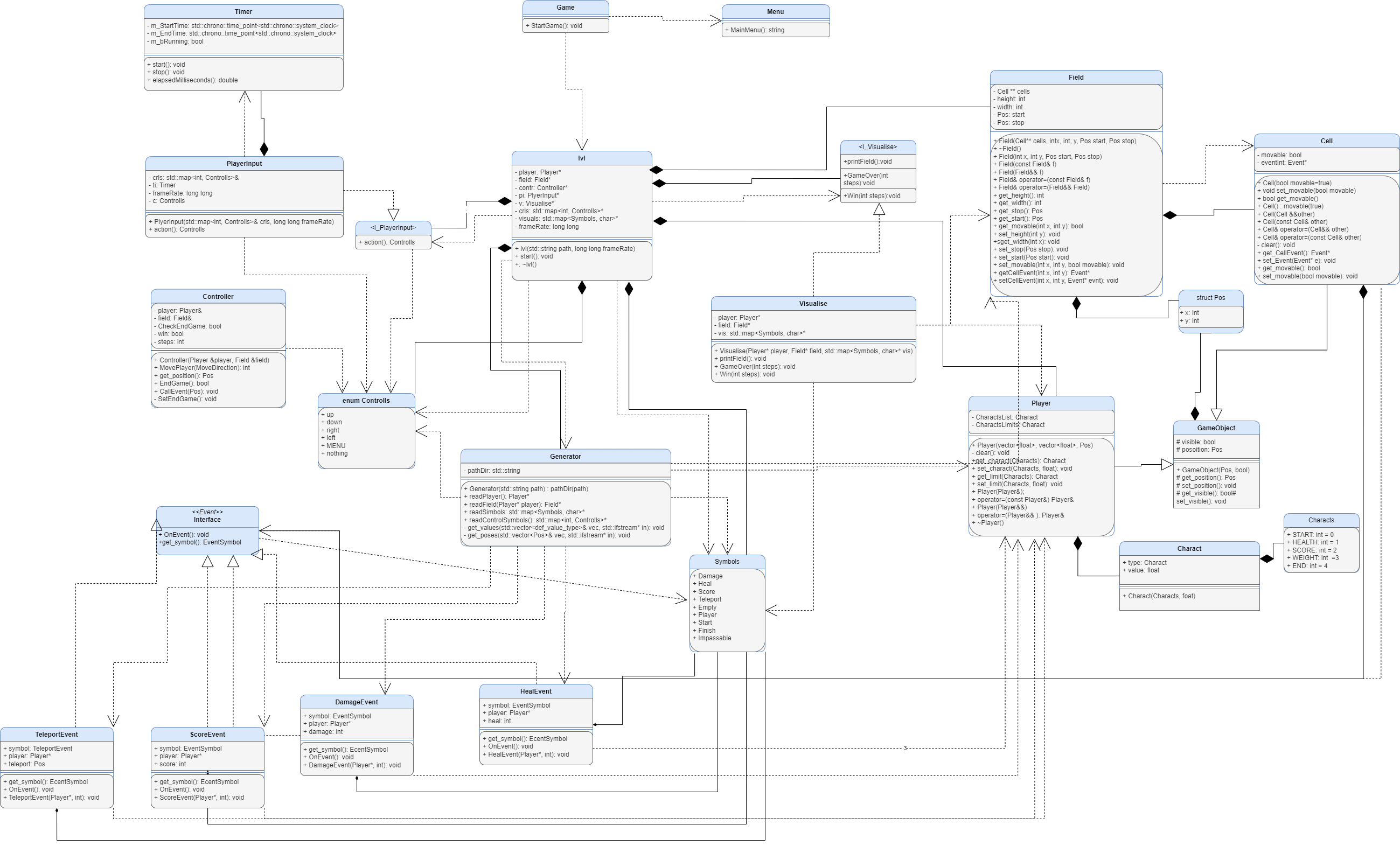
Поля:

* Player& player – ссылка на игрока
* Field& field – ссылка на игровое поле
* std::map<Symbols, char> vis – словарь с сопоставлением типу отрисовываемой информации char символа

Методы:

* Visualise(Player& player, Field& field, std::map<Symbols, char> vis) – конструктор, инициализация обязательных полей
* void printField()– отрисовка игрового поля
* void GameOver(int steps) – сообщение по проигрыше, принимает на вход количество шагов, которые сделал игрок, и выводит их
* void Win(int steps) – сообщение о победе, так же выводит пройденное количество шагов

**UML**



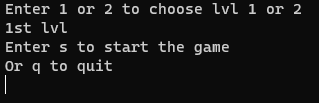
**Выводы.**

Реализованы необходимые классы с конструкторами, деструкторами и функционалом.

**Тестирование.**

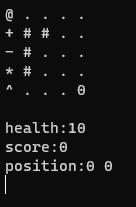
Проверка работы классов:

Здесь запущена программа, ввод – 1, выбран 1-ый уровень.



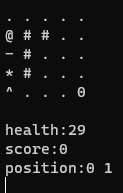
ввод – s

Запущен уровень, он отрисовывается корректно, программа ждёт ввода игрока

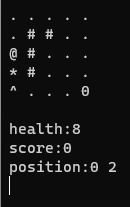


Сразу после ввода перерисовывается игровое поле, для ввода достаточно просто ввести символ, без подтверждения (нажатия enter).

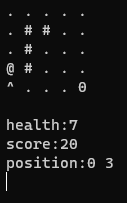
ввод – s



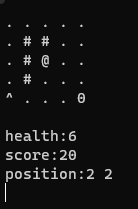
ввод – s



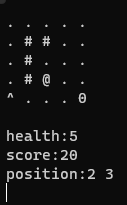
ввод – s



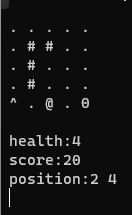
ввод – s



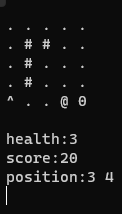
ввод – s



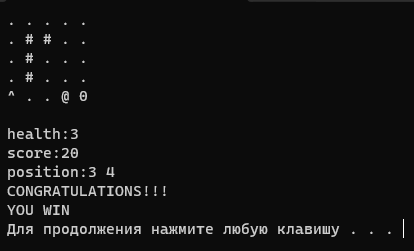
ввод – s



ввод – d



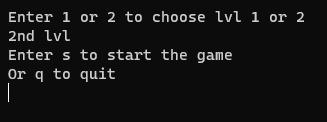
ввод – d



Все функции ввода работают корректно, отрисовка тоже, класс игры выполяет свою задачу.

Игра запущена второй раз, это 2-ой уровень.

ввод – 2



Ввод - s



Правила те же, что и в первом уровне (правила были перечислены на прошлой лабораторной работе). Сам уровень больше, загружается из файла.