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: Интерфейсы, динамический полиморфизм.

Студент гр. 1384 _____ Тапеха В. А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2022

Цель работы.

Изучить понятие принципы полиморфизма, научиться реализовывать классы, которые в иногда являются интерфейсом, и осуществлять межклассовые отношения соблюдая полиморфизм.

Задание.

Реализовать систему событий. Событие - сущность, которая срабатывает при взаимодействии с игроком. Должен быть разработан класс интерфейс общий для всех событий, поддерживающий взаимодействие с игроком. Необходимо создать несколько групп разных событий реализуя унаследованные от интерфейса события (например, враг, который проверяет условие, будет ли воздействовать на игрока или нет; ловушка, которая безусловно воздействует на игрока; событие , которое меняет карту; и.т.д.). Для каждой группы реализовать конкретные события, которые по разному воздействуют на игрока (например, какое-то событие заставляет передвинуться игрока в определенную сторону, а другое меняет характеристики игрока). Также, необходимо предусмотреть событие “Победа/Выход”, которое срабатывает при соблюдении определенного набора условий.

Реализовать ситуацию проигрыша (например, потери всего здоровья игрока) и выигрыша игрока (добрался и активировал событие “Победа/Выход”)

Требования:

- Разработан интерфейс события с необходимым описанием методов
- Реализовано минимум 2 группы событий (2 абстрактных класса наследников события)
- Для каждой группы реализовано минимум 2 конкретных события (наследники от группы события)
- Реализовано минимум одно условное и безусловное событие (условное - проверяет выполнение условий, безусловное - не проверяет).

- Реализовано минимум одно событие, которое меняет карту (меняет события на клетках или открывает расположение выхода или делает какие-то клетки проходимыми (на них необходимо добавить события) или не непроходимыми
- Игрок в гарантированно имеет возможность дойти до выхода

Примечания:

- Классы событий не должны хранить никакой информации о типе события (никаких переменных и функций дающие информации о типе события)
- Для создания события можно применять абстрактную фабрику/прототип/строитель

Выполнение работы.

Для выполнения лабораторной работы были созданы классы, отвечающие за игрока, создание клетки поля, создания поля, их вывод и взаимодействие пользователя с программой.

Новые классы:

1) Был создан интерфейс Event, от которого будут наследоваться все интерфейсы групп событий (EventPlayer, EventField).

Интерфейс EventPlayer содержит в себе в себе метод execute, который, исходя из того, какой класс от него наследуется, изменяет поля игрока и хранит в себе protected поле Player* player. От него наследуются классы Box, Heal, Key, Trap, которые соответственно увеличивает поле xp игрока (если соблюдается условие, повышает увеличивает lvl), увеличивает поле health игрока, увеличивает поле num_of_keys, уменьшает поле health игрока на случайно количество единиц. В конструкторах этих же классов инициализирует protected поле класса-родителя.

Интерфейс EventField содержит в себе в себе метод execute(), который, исходя из того, какой класс от него наследуется, изменяет класс типа Field и хранит protected поле Field* field. От него наследуется класс SetWalls, которое

при вызове метода execute вызывает в поле метод generate_walls, который добавляет новые непроходимые клетки. И также от него наследуется класс DelWalls, который вызывает метод, удаляющий все непроходимые клетки. В конструкторах этих же классов инициализирует protected поле класса-родителя.

2) Был написан класс PlayerView, который выводит все поля игрока.

3) Были написан интерфейс Observer, который содержит содержит метод update и от которого будут наследоваться PlayerView и FieldView. Метод update выводит либо статистику игрока, либо поле.

4) Создан класс Observable, который хранит в себе все наблюдаемые объекты. Есть методы attach, detach, notify, которые добавляет в вектор наблюдаемых объектов, удаляет объект из этого вектора и вызывает update у всех элементов.

5) Создан класс EventCreator, который генерирует случайным образом все события кроме события с ключом и ставит на определенные клетки события, которые добавляют ключ.

Измененные классы.

1) В классе Field был переписан метод make_field под новую реализацию. Теперь там создается объект класса EventCreator, который генерирует события. В этом же методе по определенному алгоритму ставятся непроходимые клетки в поле, проверив перед этим на наличие событий в клетке. Также в этом классе были добавлены методы generate_walls(), delete_walls(). Первый генерирует случайным образом новые непроходимые клетки, второй удаляет все такие клетки.

2) В класс Controller были добавлены методы end_game() и win_game(), которые проверяют конец игры.

3) В класс CommandReader были добавлены методы print_end_game() и print_win_game(), которые печатают конец игры.

Тестирование программы.

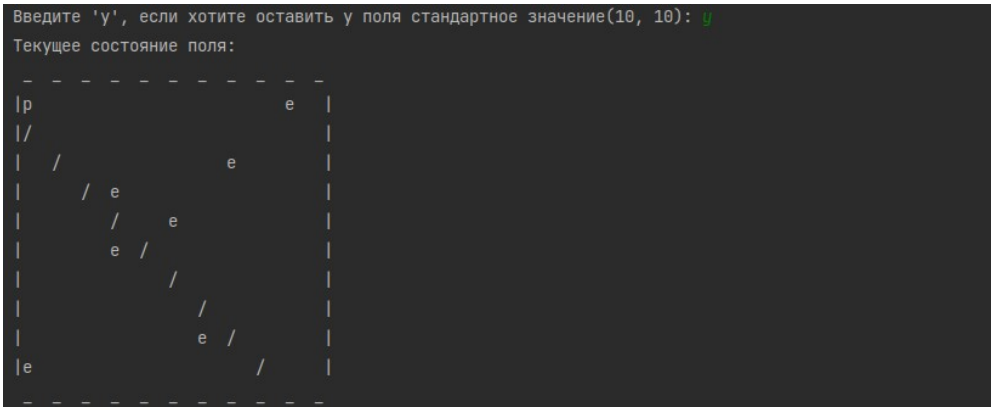

Рис. 1. Тестирование программы

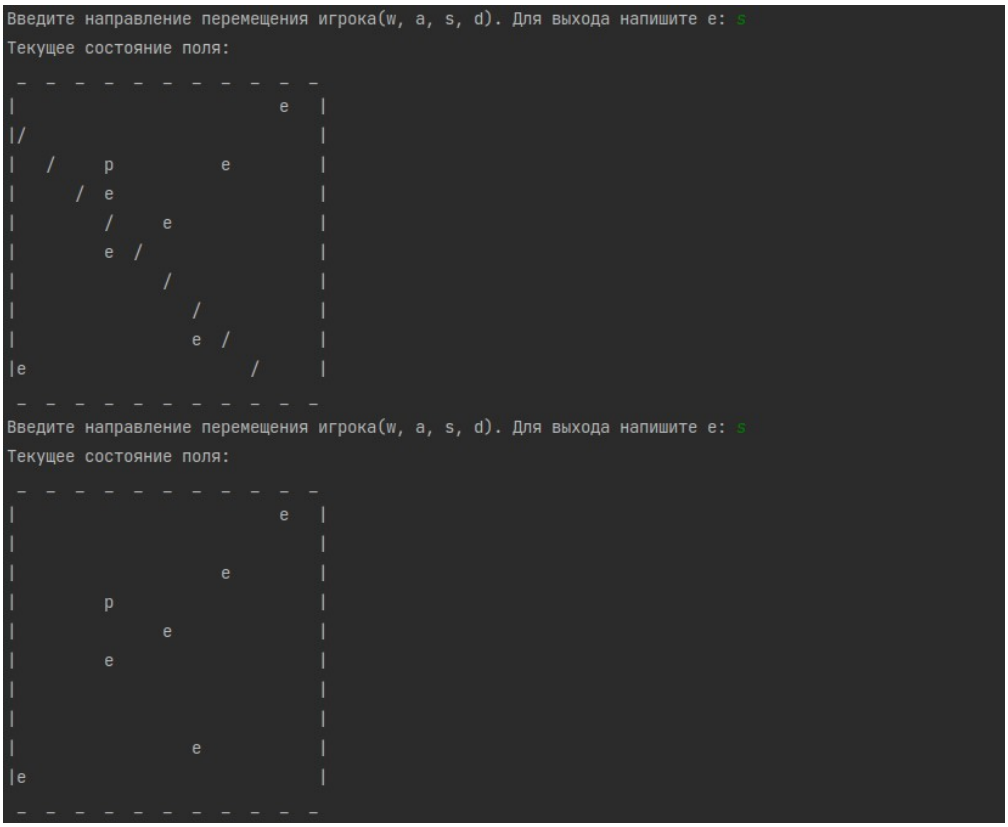

Рис. 2. Тестирование программы

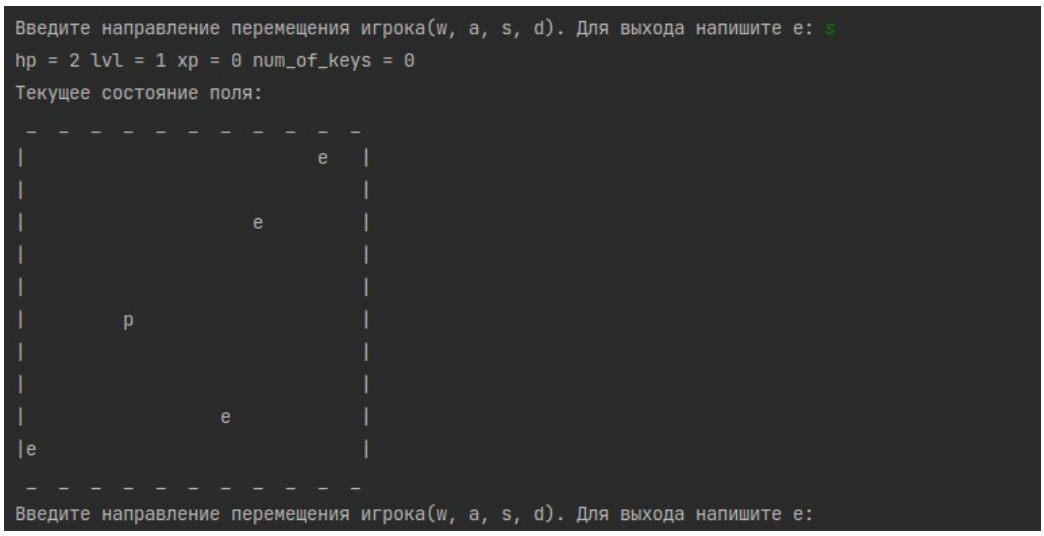

Рис. 3. Тестирование программы.

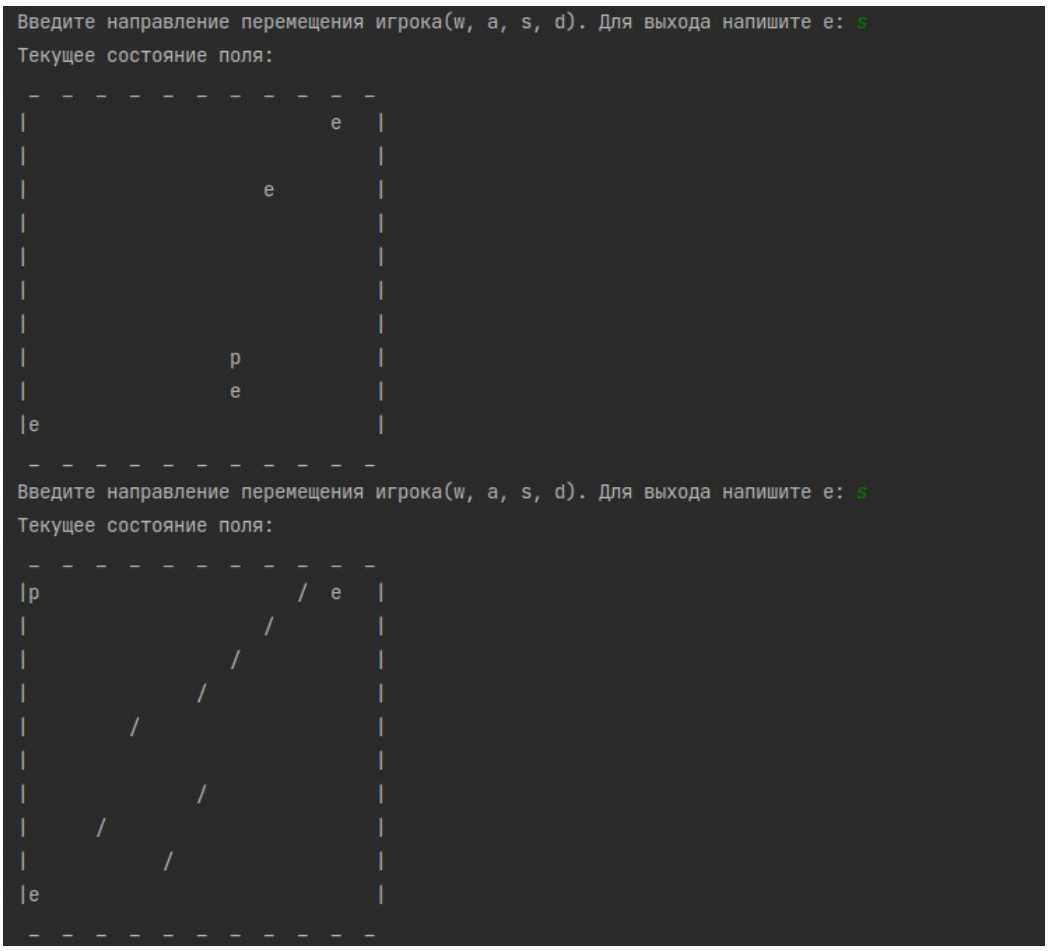

Рис. 4. Тестирование программы

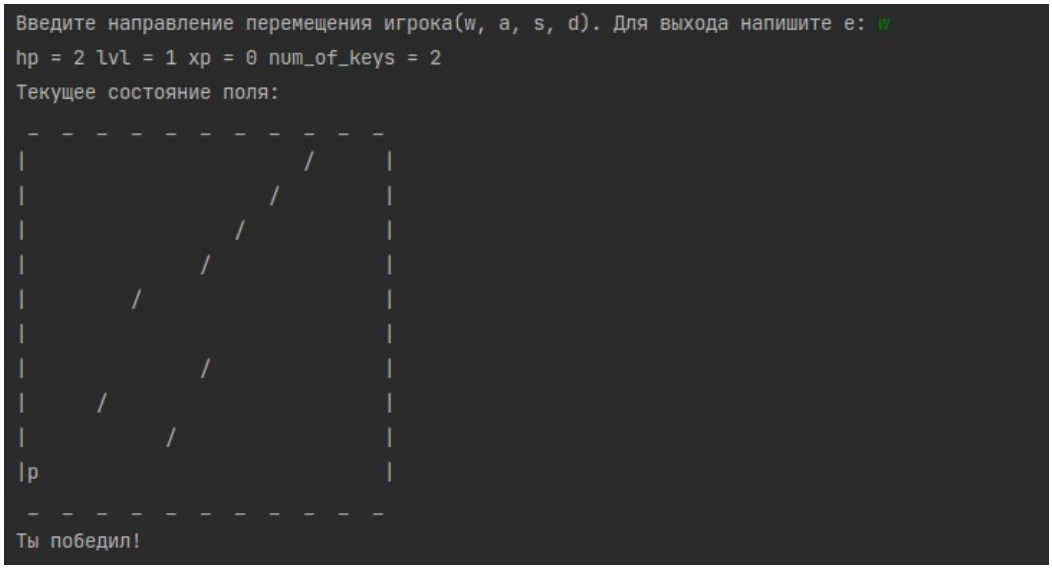

Рис. 5. Тестирование программы.

UML-диаграмма межклассовых отношений.

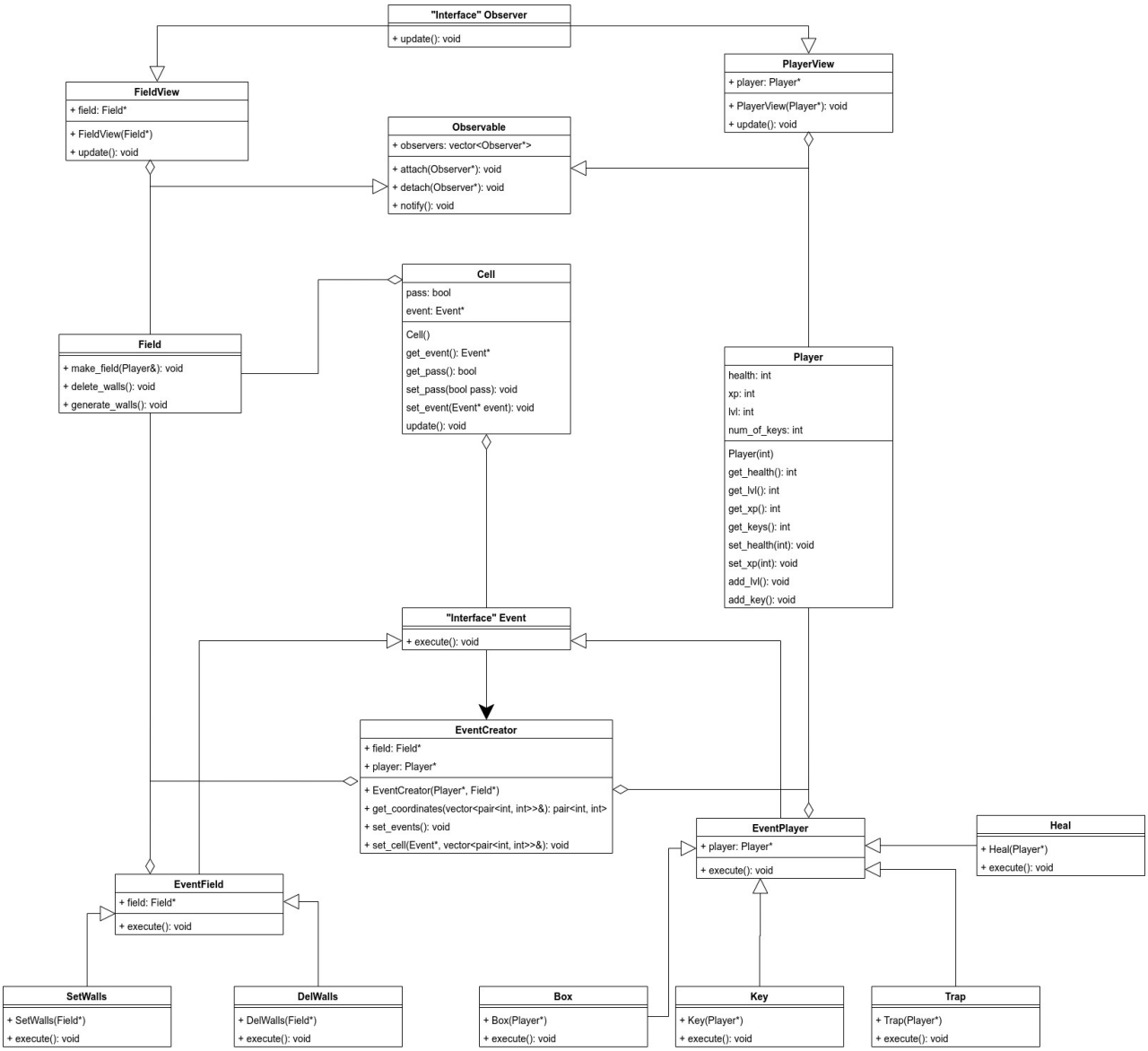

Рис. 6 UML-диаграмма.

Выводы.

Я изучил понятие принципы полиморфизма, научился реализовывать классы, которые иногда являются интерфейсом, и осуществлять межклассовые отношения, соблюдая полиморфизм.