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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Интерфейсы, динамический полиморфизм

Студентка гр.1381 _____ Рымарь М.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Изучить понятие динамического полиморфизма. Реализовать систему событий, используя интерфейс, общий для всех событий. Эти события необходимо разделить на две группы, которые будут унаследованы от интерфейса события.

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, дающих информацию о типе события).
- Для создания события можно применять абстрактную фабрику/прототип/строитель.

Выполнение работы.

В ходе выполнения лабораторной работы все заголовочные файлы и файлы с реализацией были разделены на несколько директорий для удобства. Были выделены такие директории, как *control*, *event*, *field*, *player*. В директории *event* в начале написания работы находился только заголовочный файл с инициализацией интерфейса события. В ходе написания работы были добавлены классы, отвечающие за события игрока, события поля, их вывод и взаимодействие пользователя с программой.

Из прошлой работы остался интерфейс *Event*, от него наследуются классы *eventField* и *eventPlayer* – два класса событий, первое из них изменяет поле, а второе характеристики игрока.

В классе *eventPlayer* есть метод *execute()*, который изменяет своё *protected* поле *Player* player* в зависимости от дочернего класса. От *eventPlayer* наследуются: *eventKey*, *eventTrap*. Первое событие увеличивает количество ключиков на 1 в характеристиках игрока (поле *key*) и приводит к выигрышу, второе событие уменьшает характеристику здоровья игрока (поле *health*) на 1.

В классе *eventField* есть метод *execute()*, который изменяет своё *protected* поле *Field* field* в зависимости от дочернего класса. От *eventField* наследуются:

eventWalls, *eventTeleport*. Первое событие добавляет новые непроходимые клетки на поле, второе событие телепортирует игрока в случайное место на поле.

Создан класс *eventCreator*, который устанавливает события на определённые клетки на поле.

В классе *cellView* были добавлены символы, обозначающие каждое из событий. За событие стены отвечает символ «w», за телепорт на случайную клетку на поле «*», за ключ – «k», за ловушку – «t».

В классе *Controller* добавлены методы *checkWin()* и *checkLoss()*, которые проверяют характеристики игрока, чтобы завершить игру.

В классе *commandReader* добавлены два константных метода: *printWin()*, *printLoss()*. Методы печатают строку-сообщение о победе или проигрыше, тем самым сообщая о завершении игры.

Тестирование.

Интерфейс командной строки при генерировании поля стандартных размеров с событиями на клетках представлен на рисунке 1.

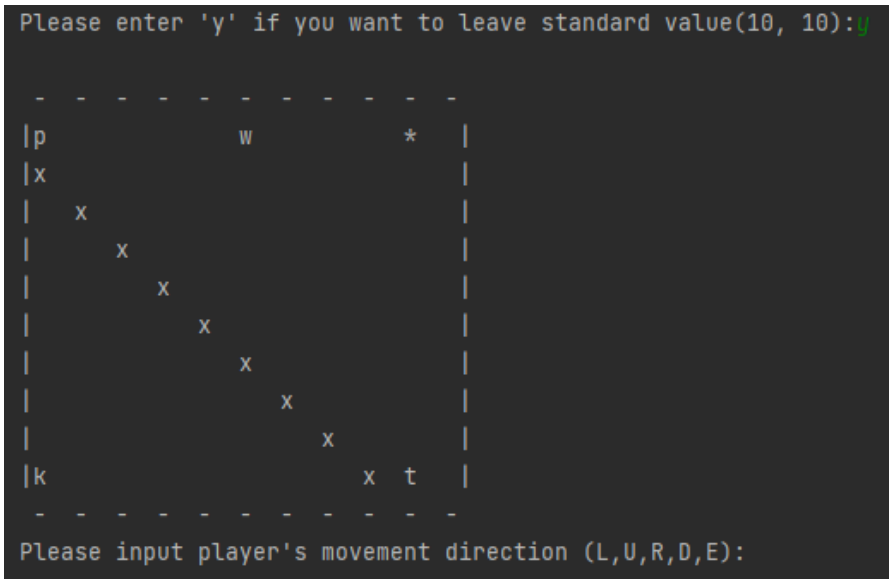

Рисунок 1 – Генерирование стандартного поля с событиями

Интерфейс командной строки при попадании на клетку с ключом “k” представлен на рисунке 2. Программа завершается победой игрока (достаточно одного ключа для победы).

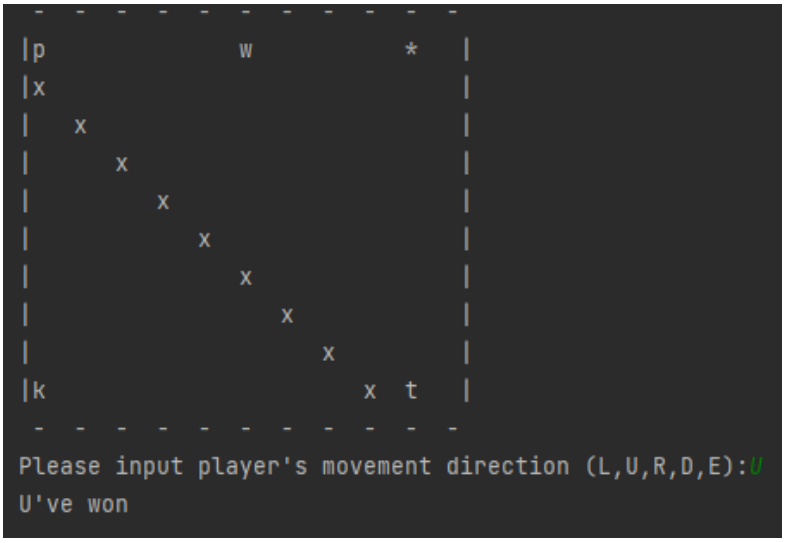

Рисунок 2 – Победа игрока при попадании на клетку с ключом “k”

Изменение поля при попадании на клетку со стеной “w” и интерфейс командной строки представлен на рисунке 3.

Рисунок 3 – Изменение поля при попадании на клетку со стеной “w”

Изменение поля при попадании на клетку с телепортом “*” и интерфейс командной строки представлен на рисунке 4.

Рисунок 4 – Изменение поля при попадании на клетку с телепортом “*”

Интерфейс командной строки при попадании на клетку с ловушкой “t” представлен на рисунке 5. Программа завершается проигрышем игрока (достаточно одной ловушки, которая отнимет одно здоровье).

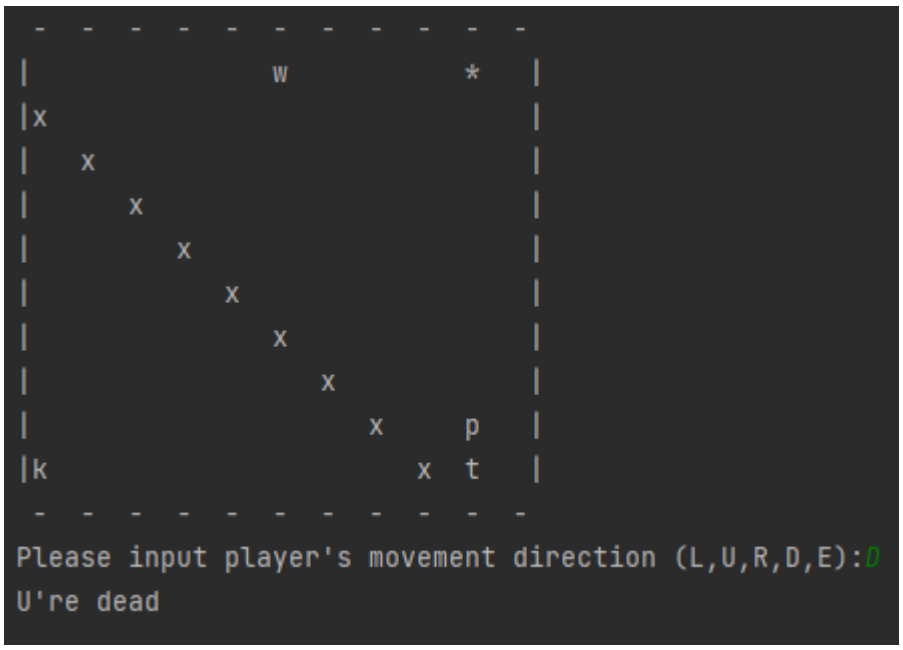

Рисунок 5 – Проигрыш игрока при попадании на клетку с ловушкой “t”

UML-диаграмма межклассовых отношений.

На рисунке 6 представлена UML-диаграмма межклассовых отношений.

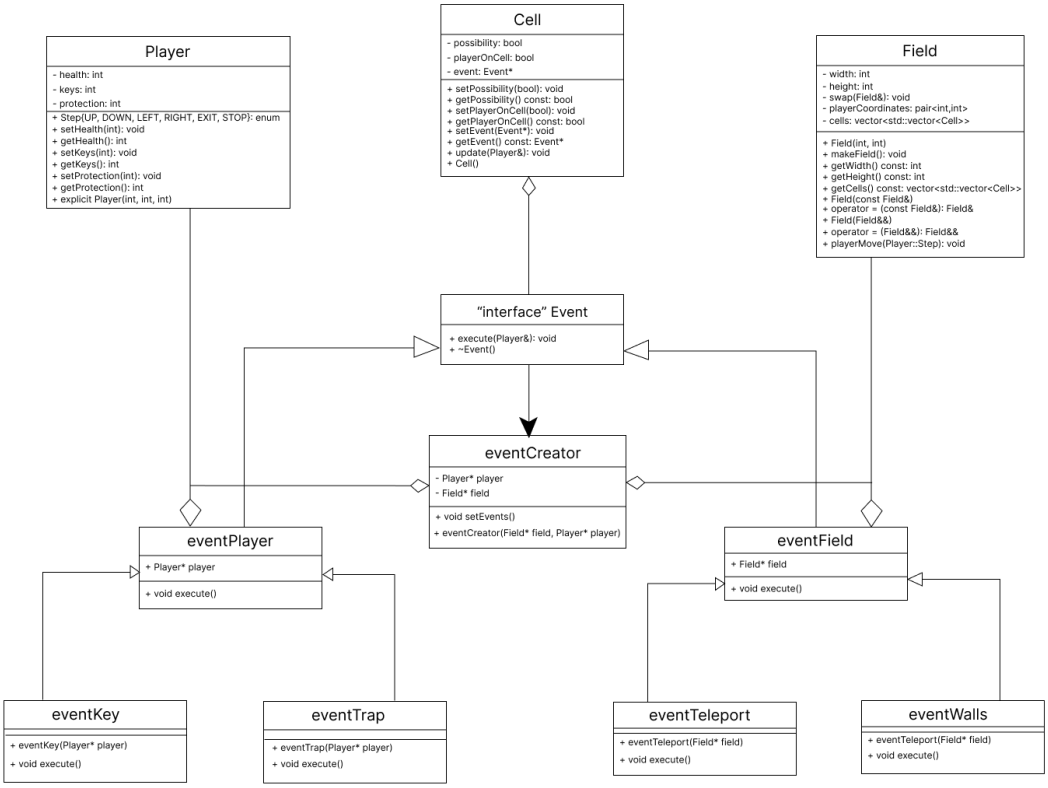

Рисунок 6 – UML-диаграмма

Выводы.

В ходе выполнения лабораторной работы было изучено понятие динамического полиморфизма. Также была реализована система событий с помощью общего для всех событий интерфейса. События были разделены на две группы, которые унаследованы от интерфейса события.