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 _____ Белокобыльский И. В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Научиться реализовывать межклассовые связи при помощи динамического полиморфизма с использованием абстрактных классов и интерфейсов.

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.

Класс EventsController связывает фабрики событий с внешним кодом. Он позволяет для получения конкретного события не подключать каждый файл фабрики по отдельности, а при помощи методов produce или getFactory получить доступ к событию или фабрике соответственно. Также дает возможность создать пустое событие, о чем будет написано ниже.

Интерфейс IEvent обобщает события как таковые для их вызова в клетках с обеспечением динамического полиморфизма.

Абстрактный класс EventChainLink реализует поведение события как звена паттерна цепочки обязанностей. Данный паттерн был выбран с целью возможности создания из простых событий сложных конструкций. Для реализации условных событий проверяется на равенство true передаваемая в объект коллбэк-функция. В случае, если необходима проверка условия на первом же звене необходимо перед ним поставить пустое событие, к которому добавить проверку функцией обратного вызова. Такой подход обусловлен тем, чтобы избежать копирования кода: так как метод dispatch реализован в конкретных событиях, там же вызывается передача управления другому событию, то выполнение условия пришлось бы проверять в каждом событии. В данной реализации достаточно вызвать метод dispatchNext, если событие не подразумевает конец игры.

Интерфейс IEventFactory необходим для динамического полиморфизма фабрик событий – объектов-наследников EventChainLink.

В конкретных фабриках реализован метод produce, принимающий на вход строку – тип необходимого события, а возвращает конкретное событие.

Конкретное событие с целью соблюдения принципа единственной ответственности взаимодействует только с необходимым ей объектом, а не с посредником, как это делают фабрики событий. Они представляют собой небольшие блоки, которые могут быть использованы как по отдельности, так и вместе в группах событий.

Было реализовано 4 абстрактных класса для событий следующих типов:

1. Взаимодействие клетками
2. Взаимодействие с картой целиком
3. Взаимодействие с состоянием игры
4. Взаимодействие с игроком.

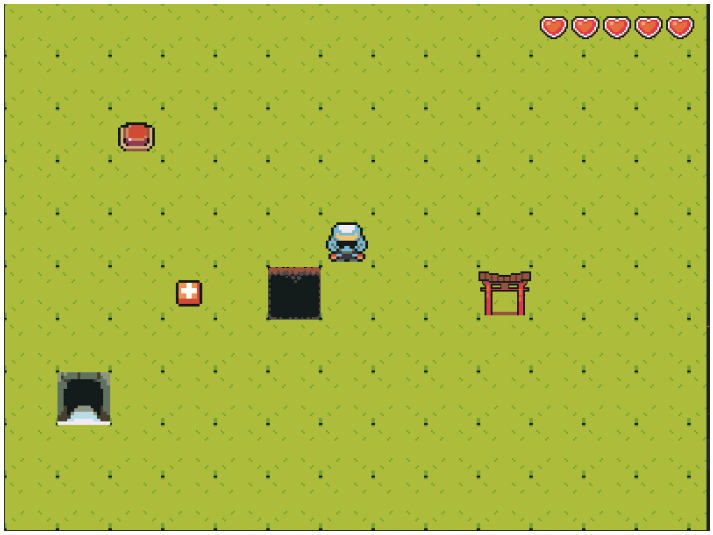
Всего создано 5 конкретных событий:

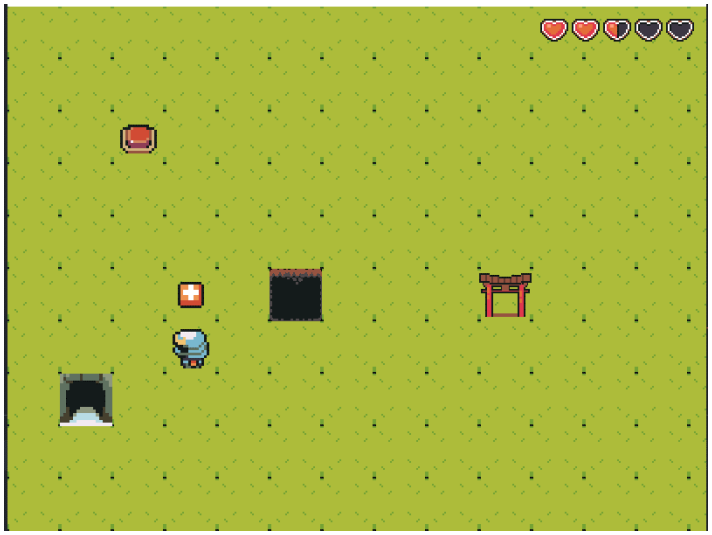
1. «Заключение» игрока в квадрат из непроходимых клеток
2. Изменение карты
3. Победа
4. Уменьшение здоровья игрока
5. Увеличение здоровья игрока

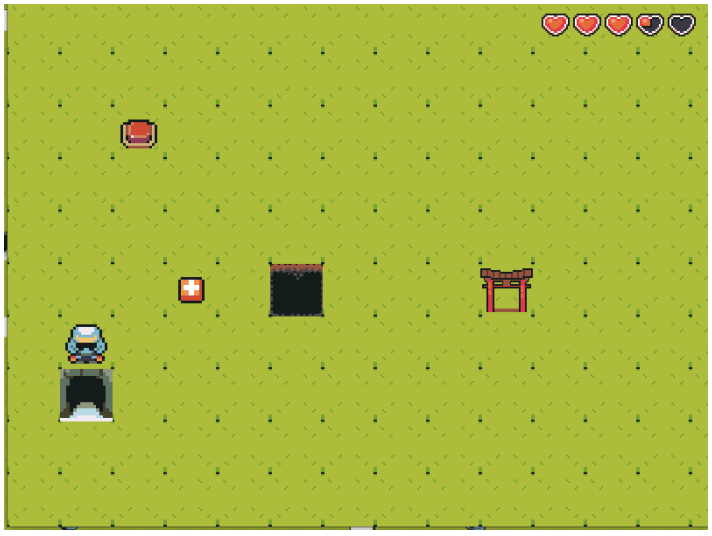
Для добавления событий на карту используется класс MapMaker, который в данной лабораторной используется скорее для тестирования имеющегося кода и будет дополнен в дальнейших лабораторных работах.

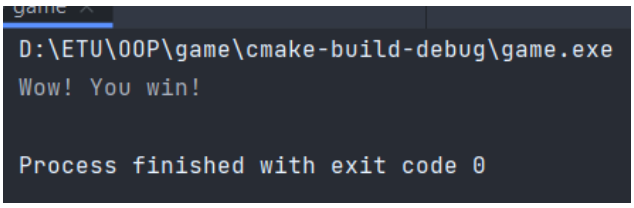
Тестирование программы.

1. Проверка работы конкретных событий

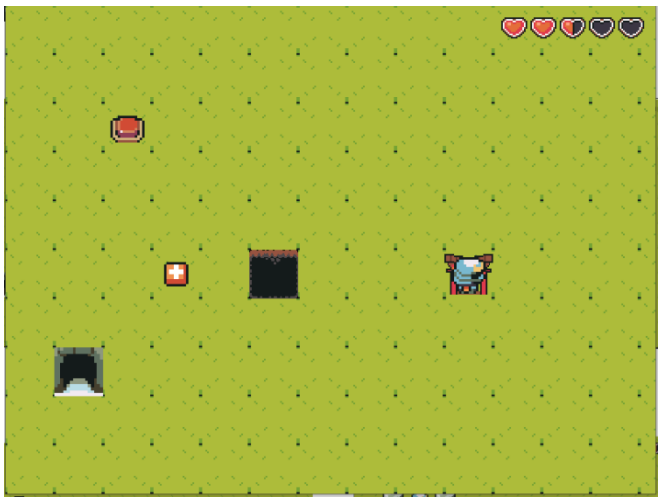




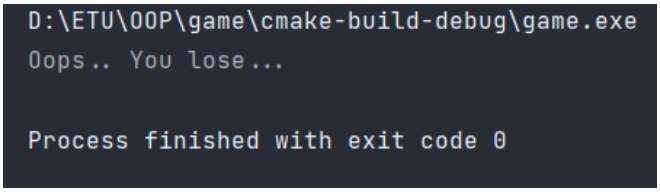




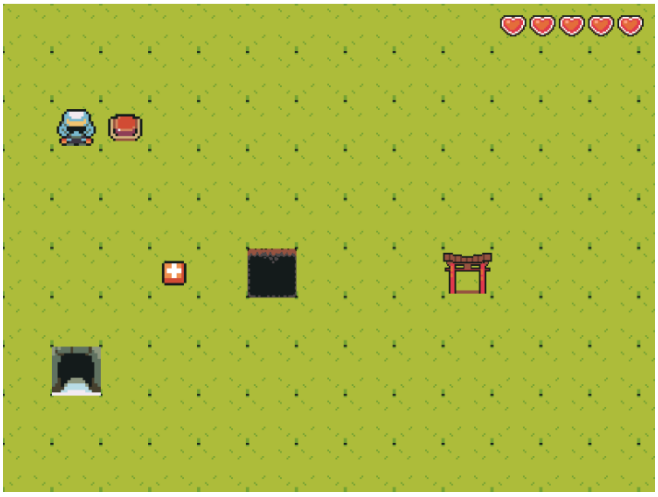
2. Проверка условий на событиях



3. Проверка возможности проигрыша



4. Проверка вызова нескольких событий последовательно



Выводы.

В рамках данной лабораторной работы была доработана программа на языке C++ из предыдущей работы. Реализован механизм событий при помощи интерфейсов и динамического полиморфизма. В работе использовались паттерны «Цепочка обязанностей» и «Абстрактная фабрика».

ПРИЛОЖЕНИЕ 1

UML-диаграмма классов: