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, полиморфизм

Студент гр. 0303 _____ Мыратгелдиев А. М.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2021

Цель работы.

Научиться реализовывать полиморфные функции и абстрактные классы (интерфейсы классов).

Задание.

Требования:

- Реализовать класс игрока. Игрок должен обладать собственными характеристиками, которые могут изменяться в ходе игры. У игрока должна быть прописана логика сражения и подбора вещей. Должно быть реализовано взаимодействие с клеткой выхода.
- Реализовать три разных типа врагов. Враги должны обладать собственными характеристиками (например, количество жизней, значение атаки и защиты, и т.д. Желательно, чтобы у врагов были разные наборы характеристик). Реализовать логику перемещения для каждого типа врага. В случае смерти врага он должен исчезнуть с поля. Все враги должны быть объединены своим собственным интерфейсом.
- Реализовать три разных типа вещей. Каждая вещь должна обладать собственным взаимодействием на ход игры при подборе. (например, лечение игрока). При подборе, вещь должна исчезнуть с поля. Все вещи должны быть объединены своим собственным интерфейсом.
- Должен соблюдаться принцип полиморфизма

Потенциальные паттерны проектирования, которые можно использовать:

- Шаблонный метод (Template Method) - определение шаблона поведения врагов
- Стратегия (Strategy) - динамическое изменение поведения врагов
- Легковес (Flyweight) - вынесение общих характеристик врагов и/или для оптимизации
- Абстрактная Фабрика/Фабричный Метод (Abstract Factory/Factory Method) - создание врагов/вещей разного типа в runtime
- Прототип (Prototype) - создание врагов/вещей на основе "заготовок"

Выполнение работы.

Была создана структура *Attribute*, для хранения общих свойств живых существ(здоровье, урон, броня и т.д.).

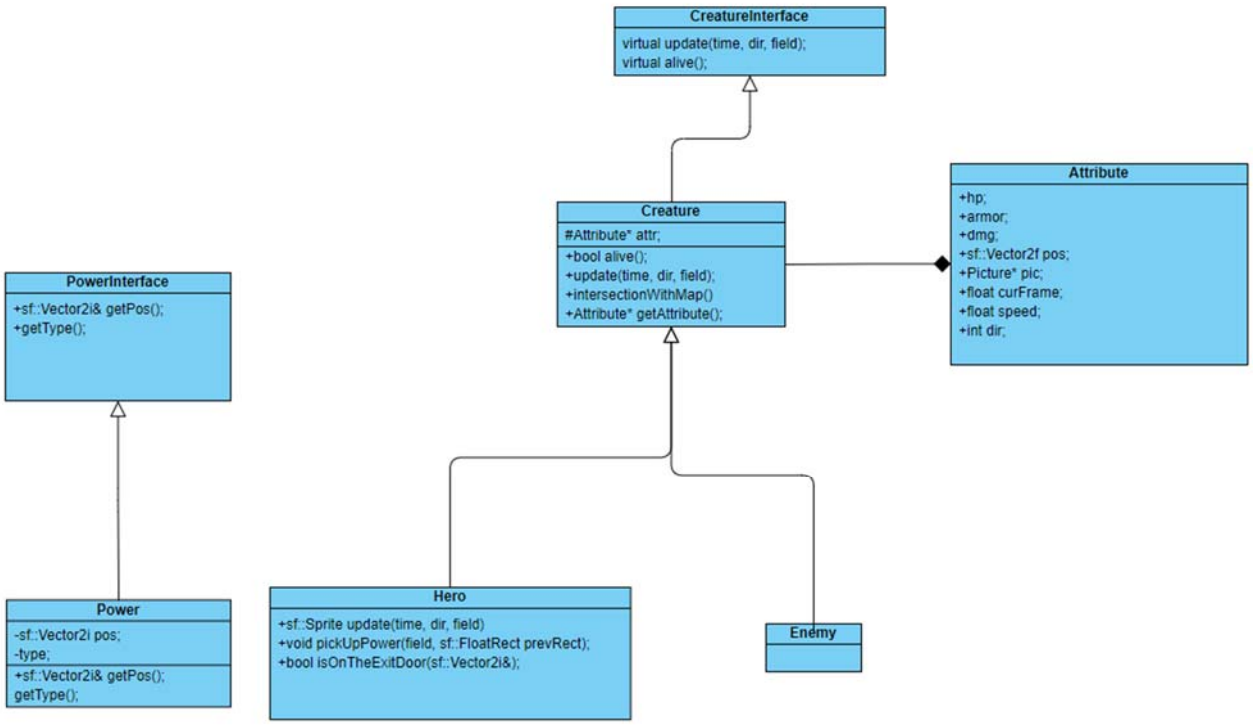
Был создан класс *Creature*, который будет определять общее поведение живых существ (героя и врагов), а также его интерфейс *Creature_Interface*. Он имеет методы *alive(), update(), intersectionWithMap(), getAttribute().* Метод *alive()* определяет живо ли существо или нет. Методы *update()* и *intersectionWithMap()* отвечают за передвижение по карте и взаимодействие с ней (столкновение со стенкой, чтобы не выходил за границы карты).

Класс *Enemy* наследуется от класса *Creature* и содержит структуру *Attribute.*

В классе *Hero*, который наследуется от *Creature*, переопределен метод *update()*, чтобы герой мог собирать предметы (бусты) и взаимодействовать с клеткой выхода.

Класс *Power* имеет поля *pos*, который хранит его месторасположение на карте и тип *type* (восстанавливает здоровье, или увеличивает броню или урон), а также геттеры для этих полей.

UML-диаграмма классов.



Выводы.

Были написаны классы *CreatureInterface, Creature, Hero, Enemy, PowerInterface, Power* и структура *Attribute.* Каждое существо (герой и враги) обладает собственным интерфейсом и набором характеристик. Игрок может подбирать вещи, которые в свою очередь меняют характеристику игрока (броня, здоровье, урон). Также для отображения героя на экране и реализации графического пользовательского интерфейса, была использована библиотека SFML.