

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.0382

Литягин С.М.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить применение интерфейсов, полиморфизм.

Задание.

Могут быть три типа элементов располагающихся на клетках:

1. Игрок - объект, которым непосредственно происходит управление. На поле может быть только один игрок. Игрок может взаимодействовать с врагом (сражение) и вещами (подобрать).
2. Враг - объект, который самостоятельно перемещается по полю. На поле врагов может быть больше одного. Враг может взаимодействовать с игроком (сражение).
3. Вещь - объект, который просто располагается на поле и не перемещается. Вещей на поле может быть больше одной.

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.

Для реализации графического интерфейса используется SFML библиотека.

В ходе работы были созданы или изменены следующие классы:

1. В классе Cell были добавлены следующие методы:

- метод GetObjectType() возвращает тип объекта, находящего на ней
- метод GetType() возвращает указатель на объект, находящийся на ней
- метод SetObject(Object* object) устанавливает указатель объекта поля на переданный объект

2. Класс интерфейса элемента клетки Object был изменен. Содержит следующие виртуальные методы:

- метод SetCoord(int x, int y) для установки объекта на координаты
- метод GetCoord() возвращает указатель на массив с координатами объекта
- метод SetType(ObjectType value) устанавливает тип объекту
- метод SetHealth(int health) устанавливает здоровье объекту
- метод GetHealth() возвращает здоровье объекта

- метод `Interaction(Object* unit)` проводит взаимодействие с объектом
- метод `GetType()` возвращает тип объекта

3. Класс интерфейса юнитов `Unit` наследуется от интерфейса `Object`.

Содержит следующие виртуальные методы:

- метод `SetForce(int damage)` для установки урона, наносимого юнитом
- метод `GetForce()` для получения урона, наносимого юнитом
- метод `SetMaxHealth(int maxHealth)` для установки максимального здоровья
- метод `IsAlive()` для проверки юнита (жив или нет)
- метод `Move(Field* field, int x, int y)` реализует перемещение юнита

4. Классы юнитов `Ent`, `Eye`, `Spider` наследуются от класса интерфейса `Unit`. Описание классов в файлах `Ent.cpp`, `Eye.cpp`, `Spider.cpp`, определение в `Ent.h`, `Eye.h`, `Spider.h`. Имеет поля `int health`, `int force`, `bool is_alive`, `int MaxHealth`, `ObjectType type`, `int coord[2]`. Содержат следующие методы:

- переопределенный метод `SetCoord(int x, int y)` для установки объекта на координаты
- переопределенный метод `GetCoord()` возвращает указатель на массив с координатами объекта
- переопределенный метод `SetType(ObjectType value)` устанавливает тип объекту
- переопределенный метод `SetHealth(int health)` устанавливает здоровье объекту
- переопределенный метод `GetHealth()` возвращает здоровье объекта
- переопределенный метод `Interaction(Object* unit)` проводит взаимодействие с объектом
- переопределенный метод `GetType()` возвращает тип объекта
- переопределенный метод `SetForce(int damage)` для установки урона, наносимого юнитом

- переопределенный метод `SetMaxHealth(int maxHealth)` для установки максимального здоровья
- переопределенный метод `IsAlive()` для проверки юнита (жив или нет)
- переопределенный метод `GetForce()` для получения урона, наносимого юнитом
- переопределенный метод `Move(Field* field, int x, int y)` реализует перемещение юнита

5. Юниты создаются при использовании паттерна Абстрактная фабрика. Для этого были реализованы: класс интерфейса фабрик `ObjectFabric`, наследованные от него классы `EntFabric`, `EyeFabric`, `SpiderFabric` (производят юнитов `ent`, `eye`, `spider` соответственно). Описание классов в файлах `ObjectFabric.cpp`, `EntFabric.cpp`, `EyeFabric.cpp`, `SpiderFabric.cpp`, определение в `ObjectFabric.h`, `EntFabric.h`, `EyeFabric.h`, `SpiderFabric.h`. Содержат следующий метод:

- метод `CreateUnit()` возвращают указатель на созданные объекты

5. Класс `Game`. Добавлено два метода:

- метод `MoveHero(Field* field, Unit* hero, int x, int y)` для перемещения игрока.
- метод `MoveEvil(Field* field, Unit** evils)` для перемещения юнитов из массива `evil`

6. Класс интерфейса вещей `Thing`. Описание класса в файле `Thing.cpp`, определение в `Thing.h`. Содержит следующий виртуальный метод:

- метод `Interaction(Hero* obj)` для взаимодействия с игроком

7. Классы вещей, наследуемые от интерфейса `Thing`, `Candy`, `Coin`, `Axe`. Имеет поля `int hp` (в классе `Candy`), `int damage` (в классе `Axe`), `int value` (в классе `Coin`), `int coord[2]` (в классах `Candy`, `Axe`, `Coin`). Содержат следующий переопределенный метод:

- переопределенный метод `Interaction(Hero* obj)` для взаимодействия с игроком

7. 8. Класс игрока Hero. Наследуется от интерфейса Unit. Имеет поля int health, int force, int coins, bool is_alive, int MaxHealth, ObjectType type, int coord[2]. Содержат следующие методы:

- переопределенный метод SetCoord(int x, int y) для установки объекта на координаты
- переопределенный метод GetCoord() возвращает указатель на массив с координатами объекта
- переопределенный метод SetType(ObjectType value) устанавливает тип объекту
- переопределенный метод SetHealth(int health) устанавливает здоровье объекту
- переопределенный метод GetHealth() возвращает здоровье объекта
- переопределенный метод Interaction(Object* unit) проводит взаимодействие с объектом
- переопределенный метод GetType() возвращает тип объекта
- переопределенный метод SetForce(int damage) для установки урона, наносимого юнитом
- переопределенный метод SetMaxHealth(int maxHealth) для установки максимального здоровья
- переопределенный метод IsAlive() для проверки юнита (жив или нет)
- переопределенный метод GetForce() для получения урона, наносимого юнитом
- переопределенный метод Move(Field* field, int x, int y) реализует перемещение юнита

