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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «ООП»
Тема: Интерфейсы, полиморфизм

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы.

Научится применять на практике полиморфизм при решении задач.

Задание.

Могут быть три типа элементов располагающихся на клетках:

Игрок - объект, которым непосредственно происходит управление. На поле может быть только один игрок. Игрок может взаимодействовать с врагом (сражение) и вещами (подобрать).

Враг - объект, который самостоятельно перемещается по полю. На поле врагов может быть больше одного. Враг может взаимодействовать с игроком (сражение).

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 программы и выполнения программы были созданы следующие классы :

Класс ***IAutonomy***:

Интерфейс, который дает своим наследникам возможность обновляться. Т.е. в дальнейшем все объекты с этим интерфейсом будут записаны в один массив и поочередно обновлятся с помощью метода ***update()*** в бесконечном цикле. Флаг ***m_alive*** нужен для того чтобы обозначить жив объект или его нужно удалить.

Если объект будет удалять сам себя , то мы не сможем узнать об этом на уровне беск. цикла и удалить объект из списка обновляемых объектов.

Поэтому используется флаг ***m_alive*** . Возможно в дальнейшем переделывание системы оповещения о смерти объекта с флага на события (паттерн Наблюдатель).

Класс ***ICreature***:

Интерфейс, для существ которые имеют такие параметры как макс. здоровье, текущее здоровье, броня, урон . Интерфейс предоставляет методы для изменения этих параметров (а также реализацию по умолчанию этих методов). И метод для расположении на поле.

Является наследником интерфейса ***IAutonomy*** так как любое существо является обновляемым объектом.

Между классом ***ICreature*** и ***Field*** организованна двухсторонняя связь (за счет указателей через класс ***ICell***), для того чтобы объект ***ICreature*** мог через указатель на поле обратится к какой нибудь клетке и через клетку мог обратится к другому объекту содержащемся на клетке (например для того чтобы узнать тип этого объекта)

Класс ***Player***:

Класс игрока , реализующий интерфейс ***ICreature***. Оставляет реализацию по умолчанию методов унаследованных от ***ICreature***. Только добавляет возможность изменять максимальное количество XP (класс ***ICreature*** не имел такого функционала).

Реализует метод для взаимодействия с предметами ***IItem*** . Реализует виртуальный метод ***update()***. В котором регенерирует здоровье игрока и проверяет достиг ли игрок конечной клетки.

Метод ***setLocation()*** находится в privat секции для того чтобы расположить объект на поле можно было только через строителя BilderField , это сделано для того чтобы игрок всегда создавался на стартовой клетке .

Классы ***VerticalScorpion, HorizontalSkeleton, CleverAlien***:

Классы врагов , наследуются от интерфейса ***ICreature***. Отличаются только характеристиками , поведением и внешним видом.

- Класс ***VerticalScorpion*** , враг который перемещается только вертикально.
- Класс ***HorizontalSkeleton*** , враг который перемещается только горизонтально.
- Класс ***CleverAlien*** , враг который идет в сторону игрока, если игрок подошел слишком близко.

Если на их пути встречается герой , атакуют его , если другое существо или стена то разворачиваются или ждут.

Класс ***IItem***:

Интерфейс предметов , которые при взаимодействии как-то влияют на существо и исчезают с поля после взаимодействия.

Важно отметить, что клетка владеет ресурсом предмета ***IItem***, для того чтобы объект использующий предмет мог удалить его через клетку. Так предмет не нужно обновлять на верхних слоях иерархии программы нет ссылок на предмет и поэтому его можно спокойно удалить.

С существом ***ICreature*** так не получится. Так как существо необходимо постоянно обновлять, то имеется ссылка на верхних уровнях иерархии(там где беск. цикл). Если существо удалится , то ссылки будут указывать на удаленную память . Именно поэтому клетка не имеет ресурс на существо находящееся на нем. Она имеет информацию о существе не более. Удаление существа происходит на более верхних уровнях, после извлечения ссылок на этот объект.

Классы ***Sword, Shield, Croissant***:

Классы предметов , наследуются от интерфейса ***IItem***. Реализуют метод ***affect()*** , который и прописывает как предмет влияет на существо использующее его.

- Класс ***Sword*** повышает урон.
- Класс ***Shield*** повышает броню.
- Класс ***Croissant*** повышает текущее здоровье.

UML диаграмма классов:

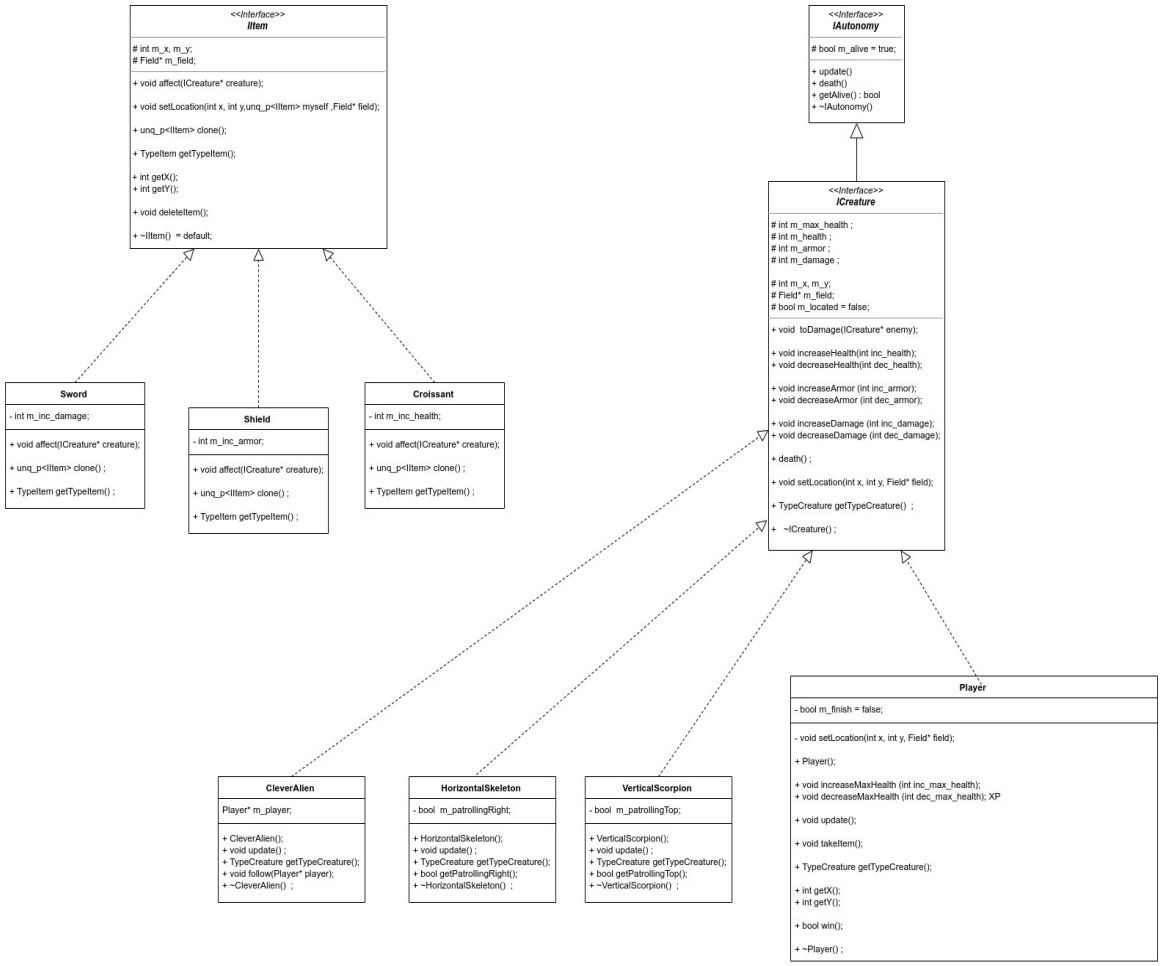

Рисунок 1: UML диаграмма классов

Тестирование.

main.ccp(самое важное):

```
//инициализация объектов
auto alien = std::make_unique<CleverAlien>();
auto sword = std::make_unique<Sword>();
unq_p<Player> player ;

//получение игрока
player = builder.buildStartCell(13, 7);

alien->follow(player.get());
alien->setLocation(10, 5, field1.get());

while ( true ){
    if (player->getAlive()){
        if(player->win()){
            break;
        }
        player->update();
    } else {
        break;
    }

    if (alien->getAlive()){
        alien->update();
    }
    view.rendering();
}
```

Результат:

Утечек памяти не обнаружено.

Рисунок 2: Исследование на утечки памяти

Программа работает корректно.

Выводы.

Был получен опыт в применении на практике принципа полиморфизма. Разработана программа, выполняющая поставленное задание.