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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Объектно-ориентированное программирование»

Тема: Интерфейсы, полиморфизм.

Студент гр. 0381 _____ Дзаппала Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать класс объекта Игрока, Врага и Вещей (для взаимодействия с игроком). На поле должно быть минимум по 3 врага и вещи. Игрок должен быть один, собственно тем, которым проходит игра. Попробовать реализовать какой-нибудь паттерн проектирования: Шаблонный метод, Стратегия (реализован), Легковес, Фабричный метод (реализован) / Абстрактная фабрика, Прототип; Один из архитектурных паттернов: MVC (реализован), MVP, MVVM.

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

Выполнение работы.

Была выбрана архитектура MVC — Model-View-Controller. Под Моделью, обычно понимается часть содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. В обязанности Представления (View) входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Классы Игрока (Player), Врага (Enemy) и Вещей (Item) являются частью бизнес-логики. Была создана такая иерархия классов:

- **Entity** — класс-интерфейс, объявляющий набор базовых методов, общих среди всех сущностей.
- **Icharacter** — наследуется от Entity, также является классом-интерфейсом, объявляющим уже общие методы «персонажей»;
- **Item** — наследуется от Entity, является абстрактным классом, объявляющий методы вещей.
- **LittleHealthBottle**, **BigHealthBottle**, **Ammo** — наследуются от Item, являются классами трех предметов: маленькой «аптечки», большой «аптечки» и пуль, соответственно.
- **Character** — наследуется от ICharacter, абстрактный класс, определяющий большинство общих методов «персонажей».

- **Player** — наследуется от Character, является классом Игрока. Определяет всю бизнес-логику обновления состояний игрока.
- **Enemy** — наследуется от Character, является абстрактный классом врагов.
- **Enemy1**, **Enemy2**, **Enemy3** — наследуются от Enemy, являются классами врагов, в которых определяются методы обновления состояний врагов и их передвижения по карте.
- **Weapon** — наследуется от Entity, является классом оружия Игрока.
- **Bullet** — наследуется от Entity, является классом пули, которую выстреливает объект Weapon. Определяет методы обновлений состояний пули, а также взаимодействия с врагами и стенами.

Также были созданы вспомогающие классы, определяющие паттерн «Фабричный метод» - ItemFactory и EnemyFactory.

Начнем с класса Игрока — **Player**. Какими свойствами обладает игрок? Так как Player наследуется от Character, а Character является АБК, то свойства игрока уже объявляются и определяются здесь. Поля: healthPoint (кол-во здоровья игрока, целое значение), damage (урон, целое значение), position (позиция игрока, объект Point), speed (скорость игрока, число с плавающей запятой), moveState (движение персонажа(сторона), переменная перечисления MoveDir). Это были поля класса Character, то есть также этими полями обладают враги. Поля Игрока (Player): bag (рюкзак-ранец игрока, является контейнером std::map, хранящий в себе пары ItemType (перечисление вещей) и std::vector, хранящий в себе указатели на объекты вещей, которые игрок подбирает), weapon (оружие игрока, объект Weapon), checkDir (сторона, в которую в последний раз перед остановкой смотрел игрок, является перечислением MoveDir), fireState (статус выстрела игроком пули, является bool состоянием). Итак, на этом это все поля, которыми обладает Игрок.

Какими методами обладает игрок?

- **UseItem**, принимающий в аргументы перечисление ItemType, что бы можно быть посмотреть, есть ли он в рюкзаке у игрока. В случае, если он есть, мы вытаскиваем последний такой, и вызываем у него метод Interact, который к слову, есть у всех объектов, наследуемых от Entity. Этот метод для каждого объекта определяет, что делать с сущностью, которую он принял в аргументы.
- **AddToBag**, принимающий указатель на объект, который игрок подбирает на поле. В самом методе уже с помощью typeinfo определяется, какой это тип вещи, и запихивается в рюкзак по ключом типа вещи.
- GetBag, метод, нужный только для возвращения ссылки на рюкзак, для класса, представляющего рюкзак.
- Fire, метод, возвращающий указатель на пулю в том случае, если Игрок выстрелил. Этот метод, в свою очередь, вызывает метод оружия (weapon) — ReduceMagazine, который также возвращает указатель на пулю, только он еще и проверяет, сколько в оружии есть патронов.
- Update, один из самых главных методов всех объектов, которые вообще могут двигаться. Метод принимает ссылку на поле, чтобы проверять что есть на текущей клетке; время, прошедшее в игре, для более плавной игры; два параметра высоты и ширины картинки Игрока, которые в дальнейшем будут нужны для метода InteractionWithGrid. В методе идет проверка в какую сторону движется персонаж, создаются две переменные дельт, после чего позиция игрока меняется.
- InteractionWithGrid, второй важный метод, проверяет, сталкивается ли объект со стеной. Принимает все тоже самое, что и Update, плюс две переменные дельт для проверок движения.

В методе Update также идет проверка, есть ли враг или вещь на клетке. Если есть, они беруться с клетки, и вызывается метод Interact объекта Player. В случае с вещью, она добавляется в рюкзак.

Пара слов о классах врагов. Enemy является АБК для Enemy1, Enemy2, Enemy3, они, в свою очередь, наследуют Enemy и определяют те самые методы Update и InteractionWithGrid. В методе InteractionWithGrid в зависимости от врага свой ход действий для врага.

Не далеко отходя от врагов, скажем два слова о «фабрике» врагов. EnemyFactory является интерфейсом, который потом наследуют фабрики для трех врагов. В каждом объекте фабрики есть виртуальный метод CreateEnemy, который принимает ссылку на поле, так как позиция врага выбирается псевдослучайно и нам надо запихнуть врага на клетку поля.

Что с вещами? Реализован АБК Item, в котором кроме поля с позицией объекта и геттерами/сеттерами, ничего нет. Этот класс наследуют актуальные объекты LittleHealthBottle, BigHealthBottle, Ammo. В каждом классе определен виртуальный метод Interact, чтобы взаимодействовать с игроком. Это, своего рода, паттерн «Стратегии». Неявный, но в каждом классе есть один и тот же виртуальный метод, выполняющий свои действия, что и похоже на паттерн «Стратегия».

Для вещей, также как и для врагов, реализована «фабрика», которая генерирует вещи и псевдослучайно выбирает для нее место, после чего выдает нам на него указатель. Метод называется — CreateItem.

Немного о контроллере, это «мини» класс, в котором хранится ссылка на Player и метод pControl(). Этот метод постоянно вызывается в игре, и в нем прописаны условия нажатий клавиш, в зависимости от которых, персонаж ходит в разные стороны, а также использует разные объекты из рюкзака.

Перейдем к Представлению.

Для представления всех объектов, реализован один класс, который хранит в себе ссылку на сам объект, который она представляет, спрайт с

текстурой (объекты из библиотеки SFML), шрифт и надпись (также из SFML), саму надпись и строку с путем к картинке объекта. Конструктор класса принимает ссылку на объект и строку с путем до картинки. Также присутствует метод updateView, принимающий два аргумента: объект окна SFML и ссылку на поле. Этот метод при каждой итерации игры вызывается для каждого объекта.

Также, был создан отдельный класс для представления рюкзака, что бы игрок понимал, что у него есть. PlayersBagView содержит в себе ссылку на сам рюкзак, поле gui, являющейся контейнером с паром ItemType и еще одной парой, текстурой и спрайтом; fontsAndText, контейнер с типом предмета и парой шрифта с текстом. Класс содержит метод UpdateView, принимающий окно SFML, которое рисует все надписи.

И, самый главный класс, который и совершает всю магию. Класс Game, содержащий в себе указатель на Окно, Player, EntityView для игрока, список из пар указатель на врага и EntityView, список из пар указатель на объект и EntityView, список из пар указатель на пули и EntityView, указатель на поле (FieldGrid) и его представление (GridView), указатель на контроллер и представление рюкзака. Класс содержит методы для инициализации каждого хранимого поля, метод для обновления состояний всех объектов (UpdateObjects), метод для «рендера» всех объектов (RenderObjects), а также метод Display, который и показывает нам все на экране.

Выводы.

Проделана немалая и интересная работа по составлению иерархий классов, а также продумывания всей работы игры на данной стадии. Были реализованы классы для Игрока, врагов и вещей.

ПРИЛОЖЕНИЕ А

UML ДИАГРАММА КЛАССОВ