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. 0304 _____ Никитин Д.Э.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить работу с абстрактными классами, научиться применять полиморфизм в языке программирования C++ на примере создания игрока, монстров и предметов, а также логики их взаимодействия.

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
- Гарантировать отсутствие утечки памяти.

Выполнение работы.

Исходный код программы приведен в приложении А. Диаграмма классов приведена в приложении Б.

1. Создание(или использование готовых) вспомогательных интерфейсов, позволяющих упростить работу с игровыми объектами:

sf::Drawable — интерфейс библиотеки для отрисовки sfml, используется для отрисовки объекта через переопределение метода draw.

Prototype — интерфейс, позволяющий создавать клон объекта не вдаваясь в подробности его реализации. Это возможно за счет метода clone, в котором происходит копирование полей и выдача указателя на новый объект.

TextureGetter — интерфейс, использующийся для получения определенной текстуры через метод getTexture за счет получения текстуры через сиглтоновый класс текстур.

2. Создание интерфейса GameObject, следующего интерфейсам sf::Drawable, Prototype, TextureGetter. Это позволяет задать основное поведение для всех объектов: получение текстуры, клонируемость, а также рисуемость.

3. Создание абстрактных классов Item и Creature следующих интерфейсу GameObject.

Item — абстрактный класс, задающий поведение всех предметов, а именно:

- хранит в себе координаты
- Переопределен метод draw, позволяющий отрисовать существо по координатам

- Виртуальные методы clone, getTexture все еще не переопределены.

Creature — абстрактный класс, задающий поведение всех существ, а именно:

- наличие показателей здоровья, брони и атаки и их последующее изменение(changeHealth, changeArmor, changeAttack) и получение(геттеры);
- наличие move_manager - а, а также геттера для его получения. (CreatureMoveManager — абстрактный класс, следующий интерфейсу Moveable(возможность перемещения объекта за счет метода move), хранящий в себе координаты в виде пары, а также состояние объекта(уже передвигался или нет). Конструктор принимает координаты, переопределен метод move, используются геттеры(для состояния и координат) и сеттеры(для состояния)).
- Виртуальные методы clone, getTexture все еще не переопределены.
- Переопределен метод draw, позволяющий отрисовать существо по имеющимся в move_manager-е координатам

4. Созданы следующие классы: Player и Enemy, наследующиеся от Creature.

Player — класс игрока, конструктор принимает координаты, переопределены методы clone и getTexture.

Enemy — абстрактный класс, в котором все еще не переопределены методы clone и getTexture.

5. Созданы классы врагов, которые наследуются от Enemy и обладают собственными характеристиками: ShadowWarrior, KillerAnt, Spider. Во всех классах конструктор принимает координаты. Переопределены методы clone и getTexture.

6. Создан абстрактный класс Potion, наследующийся от Item. Зелье обладает эффектом либо какой-то силы, либо какой-то продолжительности.

Для хранения эффекта используется поле effect. Создан геттер для эффекта. Методы clone и GetTexture не переопределены.

7. Созданы зелья, которые наследуются от Potion и обладают собственными характеристиками . Во всех классах конструктор принимает координаты. Переопределены методы clone и getTexture.

8. Создан абстрактный класс Spawner. Он хранит в себе игровое поле, но не управляет временем его жизни. Виртуальный метод spawn позволяет создать нужный объект на игровом поле.

9. Созданы 3 типа спавнера, наследующихся от Spawner:

- PlayerSpawner
- EnemiesSpawner
- ItemsSpawner

У них конструктор принимает GameField& и переопределен метод spawn(проход по клеткам поля и помещение в них созданного объекта с определенной вероятностью).

10. Создан класс EventManager. Основная цель его создания — обработка всевозможных событий на игровом поле. Взаимосвязь с классом игры обусловлена тем, что Game — синглтон, благодаря этому при смерти игрока или достижении игроком клетки финиша происходит изменение состояния игры(not_ended становится false, т. е. игра завершена). Поля класса — игровое поле и игрок. Конструктор класса принимает игровое поле и игрока. Основные типы событий:

- Нажатие клавиши: KeyPressed: происходит движение игрока в направлении заданном пользователем, после этого двигаются все враги.
- Подбор предмета игроком: pickUp.
- Атака Creature по Creature: attack.

- Движение игрока/врага: move: всевозможные проверки и возможные события: аттака, передвижение, выход из игры(для игрока)
- Движение всех врагов: enemiesMove.
- Проверки на то, что:
  - ➔ Движение в поле: checkMoveInField
  - ➔ Движение не в стену: checkNotWall
  - ➔ Движение не во врага: checkNotEnemy
  - ➔ Движение не в предмет: checkNotItem
  - ➔ Движение не в игрока: checkNotPlayer
  - ➔ Игрок пришел в клетку финиша: isFinishCell

Тестирование.

Результаты тестирования программы представлены на рис. 1 и рис. 2

Рисунок 1

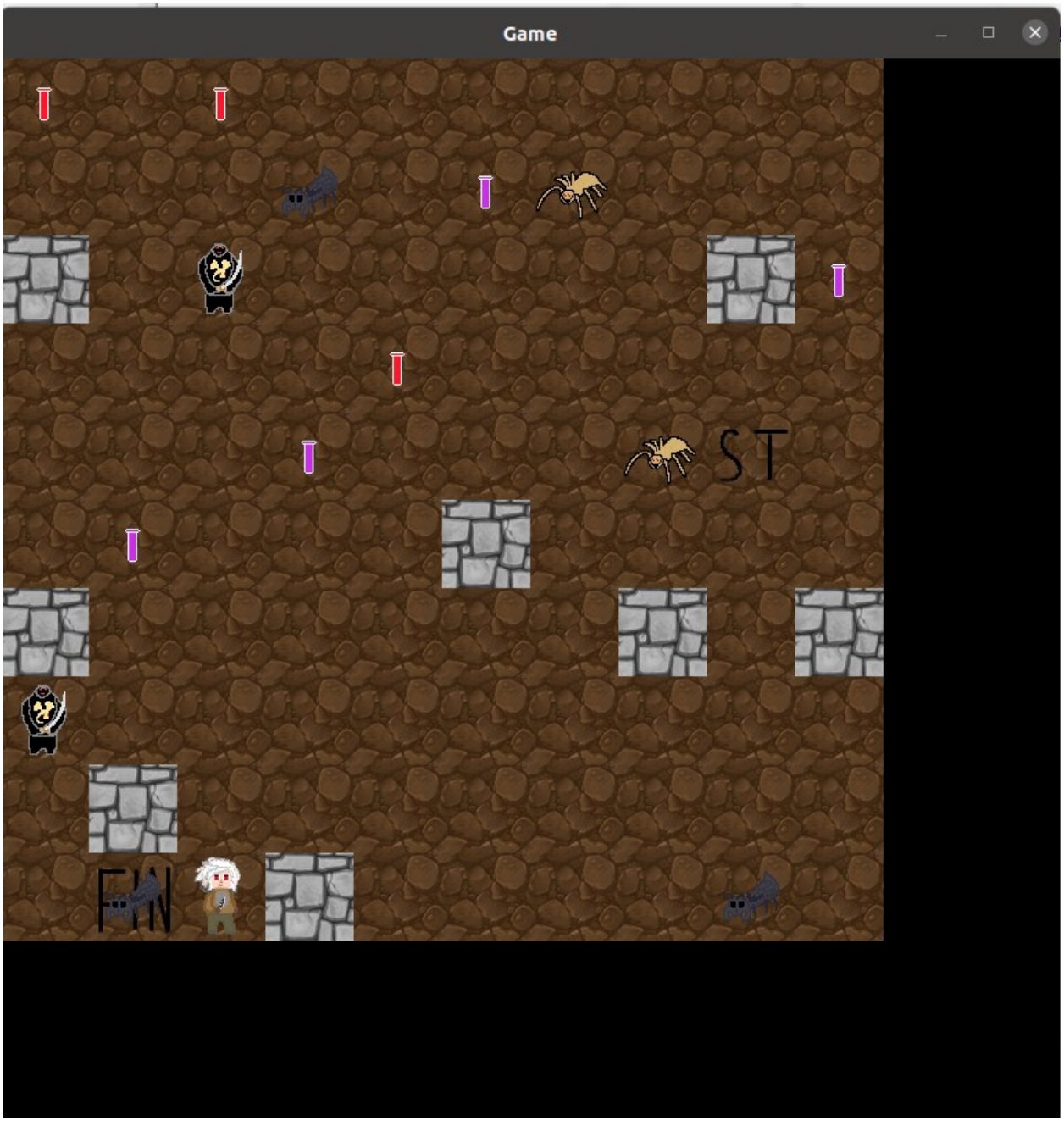

Рисунок 2

Выводы.

Был изучен процесс создания абстрактных классов в языке C++, применение полиморфизма, а также разработка с использованием паттернов проектирования.

Была разработана программа, создающее игровое поле с клетками, содержащими объекты внутри. Клетки подразделяются на типы: клетка

старта, клетка финиша, обычная клетка и стена. Разработан графический интерфейс для вывода результата на экран.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Ссылка на гитхаб: https://github.com/Fovteltar/Game.git

ПРИЛОЖЕНИЕ Б

UML-ДИАГРАММА

Ссылка на диаграмму: https://miro.com/app/board/o9J_lssZB5c=/