**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**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0382 |  | Андрющенко К.С |
| Преподаватель |  | Жангиров Т. Р |

Санкт-Петербург

2021

**Цель работы.**

Реализация класса игрока, содержащего набор изменяющихся характеристик. Осуществление взаимодействия игрока с различными клетками поля. Создание классов-наследников элементов клетки Item и Enemy.

**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).

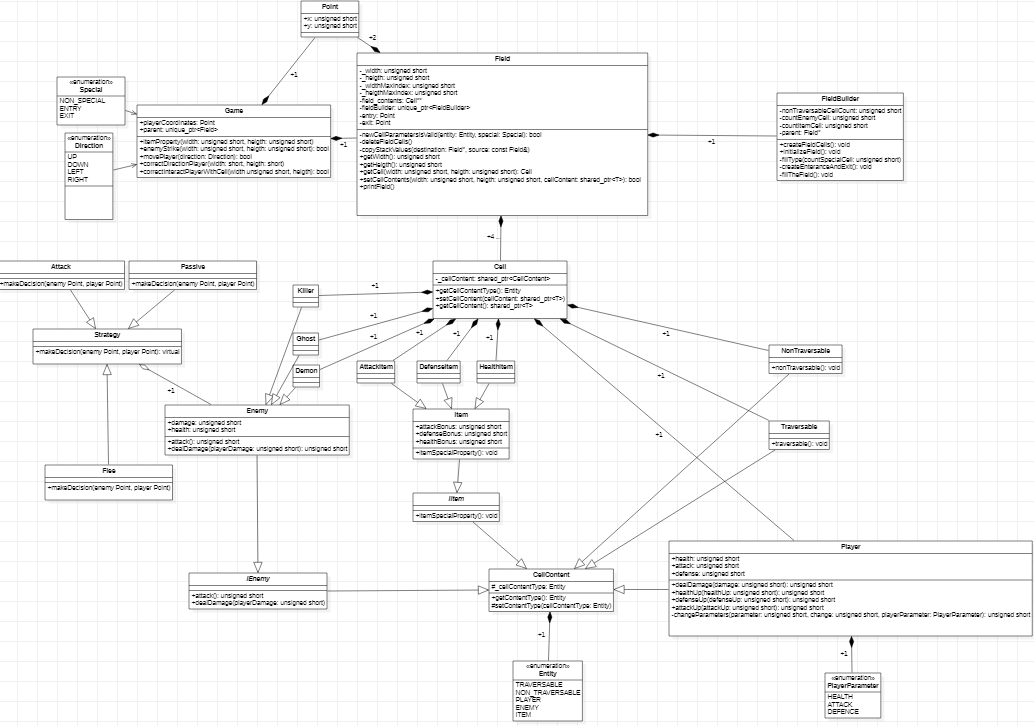
## Вещь - объект, который просто располагается на поле и не перемещается. Вещей на поле может быть больше одной.

## 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.

**Выполнение работы.**

UML-диаграмма для данной лабораторной работы см. Рисунок 1.

 Рисунок 1 – UML-диаграмма лабораторной работы №2

## Описание диаграммы.

## Перечисления:

*Direction* хранит информацию о возможном направлении движения.

*Point* хранит информацию о координатах специальной клетки.

*PlayerParameter* хранит информацию о том, какими свойствами обладает класс Player.

*Special* хранит информацию о том, что находится на клетке.

*Point* хранит информацию о координатах объекта на поле.

## Классы, созданные и модифицированные в данной лабораторной работе, их методы и свойства.

Класс Game:

Заголовочный файл см. ПРИЛОЖЕНИЕ A.

Класс, который управляет перемещением игрока по полю, изменением его состояния и взаимодействием игрока с другими элементами клетки.

Свойства класса.

*Point playerCoordinates* - Координаты игрока на поле.

*vector<Point> enemyPosotions –* Вектор с координатами всех врагов на поле.

Методы класса.

*void itemProperty(unsigned short width, unsigned short heigth) -* Взаимодействие с предметом. При попадании игрока на клетку с предметом, состояние игрока меняется.

*bool fight(unsigned short width, unsigned short heigth)* – Взаимодействие с врагом. Изменение характеристик врага и игрока при попадании их на одну клетку.

*bool movePlayer(Direction direction)* – Функция движения игрока.

1. Вызывает функцию проверки положения желаемой клетки перехода (bool checkPlayerDirection(short width, short heigth)), которая определяет не выходит ли игрок за границы поля, находится ли он на клетке выхода.
2. Вызывает функцию проверки взаимодействия игрока с объектом, находящимся на желаемой клетке перехода (bool correctInteractPlayerWithCell(unsigned short width, unsigned short heigth)), если враг, то происходит бой, элемент – удаление элемента с поля и изменение характеристик врага, непроходимая клетка – игрок остается на месте, проходимая – осуществляется переход.
3. Вызывает функцию, изменяющую положение игрока на поле (старая клетка содержит указатель на Traversable, новая – на Player).
4. Изменяет свойство *Point playerCoordinates* в зависимости от нового положения игрока.
5. Вызывает функцию *void startTurn()* , которая перебирает вектор существующих врагов и меняет их положение, проверяя переход на желаемую клетку следующими функциями:

*bool checkEnemyDirection(Point enemyPosition, Point desiredPoint);*

*void moveEnemy(Point enemyPosition);*

*void moveEnemyCell(short currentX, short currentY, short desiredX, short desiredY);*

*bool interactEnemyWithCell(unsigned short width, unsigned short heigth, Point enemyPosition);*

Класс Strategy:

Заголовочный файл см. ПРИЛОЖЕНИЕ B.

После каждого хода игрока враги перемещаются по полю (*void startTurn())* , направление, в котором будет перемещен враг выбирается стратегией. По умолчанию стратегия врага Flee – враги сдвигаются вниз по полю. Выход за границы поля и взаимодействие с элементами клетки проверяются.

Класс Enemy:

Установка стратегии движения врага по полю и получение текущей стратегии.

1. *void setStrategy(std::shared\_ptr<Strategy> strategy)*

*{*

*\_strategy = strategy;*

*}*

2*. std::shared\_ptr<Strategy> getStrategy();*

Заголовочный файл см. ПРИЛОЖЕНИЕ C.

Класс Item:

Метод *void Item::itemSpecialProperty()* выводит свойства конкретного предмета, с которым взаимодействует игрок.

Заголовочный файл см. ПРИЛОЖЕНИЕ D.

## Отношение между классами.

Отношение между абстрактным классом *IEnemy* и CellContent – наследование, т.к. *IEnemy* является частной формой CellContent. Также наследниками CellContent являются классы *IItem*, NonTraversable, Traversable и Player.

Классы, представляющие различную стратегию поведения (Passsive, Flee, Attack) наследуются от класса стратегии Strategy.

Отношение между классом Item и абстрактным классом *IItem* – реализация. Item реализует поведение, заданное поставщиком *IItem*.

Отношение между классом Enemy и абстрактным классом *IEnemy* – реализация. IEnemy реализует поведение, заданное поставщиком *IEnemy*.

Отношение между классами Enemy и Killer, Ghost, Demon (типы врагов) - наследование. Классы врагов имеют разные характеристики. Все враги объединены своим собственным интерфейсом.

Отношение между классами Item и AttackItem, DefenseItem, HealthItem (типы предметов) - наследование. Классы предметов имеют разные характеристики. Все предметы объединены своим собственным интерфейсом.

Отношение между классами Killer, Ghost, Demon, AttackItem, DefenseItem, HealthItem, NonTraversable, Traversable, Player и классом Cell - композиция, т.к. данные классы являются клетками (контейнер) и не имеют смысла вне их. Если клетки будут уничтожены, то их содержимое тоже.

Класс поля(Field) “хранится” в классе Game – отношение – композиция.

Класс поля(Field) – контейнер для класса FieldBuilder(композиция).

Свойства классов-наследников классов Enemy и Item переопределены, каждый – со своим набором характеристик.

*IItem* – абстрактный класс, методы которого – вывод характеристик предмета.

*IEnemy* – абстрактный класс, методы – возврат урона, наносимого врагом, и установка нового значения здоровья после встречи с игроком.

Item – наследник класса *IItem*, свойства которого – значения, прибавляемые к показателю здоровья, атаки и защите игрока при их подборе. В данном классе переопределена функция, выводящая характеристики предмета.

Описание классов.

Класс Game управляет созданием поля и корректным перемещением игрока по нему, обрабатывает крайние случаи – выход за границы поля. Сначала создается поле «пустых» клеток необходимых размеров (Game), а затем, установка параметров клеток - предварительное конструирование поля с необходимым параметрами (FieldBuilder).

При переопределении параметров клетки происходит проверка на их соответствие. При вызове деструктора происходит удаление созданных указателей, что предотвращает утечку памяти. В остальных случаях – использование умных указателей.

## Проверка классов.

Приложим скриншоты тестов, находящихся в main(), результат и при необходимости памяти после выполнения тестов.

Создание поля заданных размеров (ручное заполнение при создании). Обработаем случай взаимодействия с врагом, предметом, непроходимой клеткой, выходом за границы поля. см.Рисунок 2:

Main():



Output:

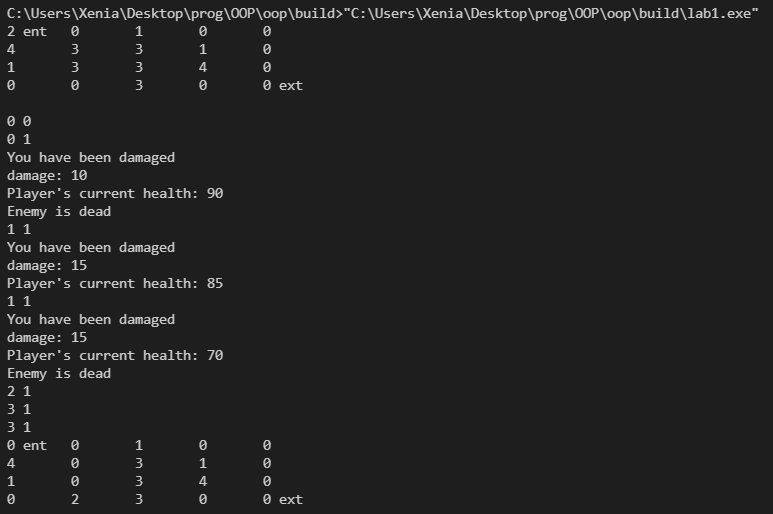


Рисунок – Движение игрока по полю, обработка исключений.

Проверка выхода за границы поля, взаимодействие игрока с врагом и предметом, столкновение врага с проходимыми и непроходимыми клетками, предметами см. Рисунок 3:

Main():

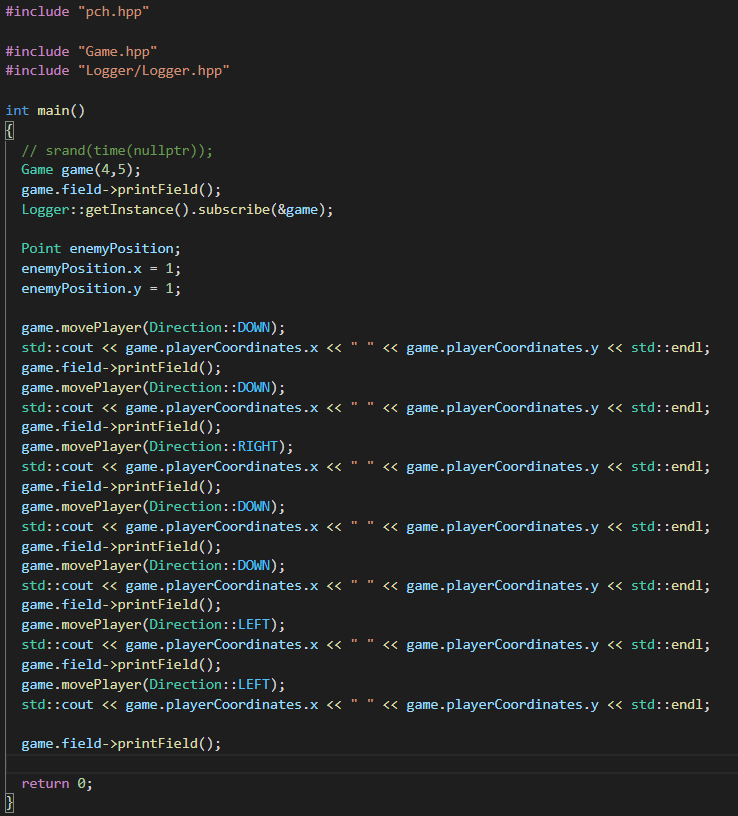


Рисунок - Движение игрока по полю, обработка исключений.

Output(консоль):

2 ent 0 3 0 0

1 0 4 4 0

1 3 0 1 0

0 0 0 0 0 ext

0 1

0 ent 2 0 3 0

1 0 4 4 0

1 0 3 1 0

0 0 0 0 0 ext

Non traversable for enemy

0 2

0 ent 0 2 0 3

1 0 4 4 0

1 0 3 1 0

0 0 0 0 0 ext

MOVEMENT: Player's current health is max

MOVEMENT: Player's current defence: 2

MOVEMENT: Player's current attack: 20

MOVEMENT: Attack: 0

MOVEMENT: Defense: 1

MOVEMENT: Health: 5

Non traversable for enemy

1 2

0 ent 0 0 0 3

1 0 2 4 0

1 0 3 1 0

0 0 0 0 0 ext

MOVEMENT: Player's current health: 100

MOVEMENT: Player's current defence: 0

MOVEMENT: Player's current attack: 25

MOVEMENT: Attack: 5

MOVEMENT: Defense: 0

MOVEMENT: Health: 0

Non traversable for enemy

1 3

0 ent 0 0 0 3

1 0 0 2 0

1 0 3 1 0

0 0 0 0 0 ext

Non traversable for enemy

1 4

0 ent 0 0 0 3

1 0 0 0 2

1 0 3 1 0

0 0 0 0 0 ext

You have been damaged

BATTLE: Damage: 10

MOVEMENT: Player's current health: 90

BATTLE: Emeny is dead.

Non traversable for enemy

Non traversable for enemy

0 4

0 ent 0 0 0 2

1 0 0 0 0

1 0 3 1 0

0 0 0 0 0 ext

Non traversable for enemy

0 4

0 ent 0 0 0 2

1 0 0 0 0

1 0 3 1 0

0 0 0 0 0 ext

**Вывод.**

В ходе работы были написаны классы поля, стратегии, определяющей поведение врагов, игры, отвечающей за взаимодействие объектов поля и перемещение игрока и врагов.

ПРИЛОЖЕНИЕ A

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Game.hpp**

#pragma once

#include "Ghost.hpp"

#include "Killer.hpp"

#include "Demon.hpp"

#include "AttackItem.hpp"

#include "HealthItem.hpp"

#include "DefenseItem.hpp"

#include "Field.hpp"

#include "Cell.hpp"

#include "CellContent.hpp"

#include "Logger/Publisher.hpp"

#include "Logger/Logger.hpp"

class Field;

class Cell;

class CellContent;

class Game : public Publisher

{

public:

  Game(unsigned short width, unsigned short heigth);

  Game(const Game& other) = delete;

  Game(Game&& other) = delete;

  Game& operator=(const Game& other) = delete;

  Game& operator=(Game&& other) = delete;

  ~Game();

  std::shared\_ptr<Field> field;

  /// @brief Взаимодействие с предметом.

  void itemProperty(unsigned short width, unsigned short heigth);

  /// @brief Взаимодействие с врагом.

  bool fight(unsigned short width, unsigned short heigth);

  /// @brief Движение игрока.

  bool movePlayer(Direction direction);

  /// @brief Проверка клетки перехода.

  bool checkPlayerDirection(short width, short heigth);

  /// @brief Проверка корректности положения желаемой клетки на поле.

  bool correctInteractPlayerWithCell(unsigned short width, unsigned short heigth);

  /// @brief Координаты игрока на поле.

  Point playerCoordinates;

  bool checkEnemyDirection(Point enemyPosition, Point desiredPoint);

  void moveEnemy(Point enemyPosition);

  void movePlayerCell(short currentX, short currentY, short desiredX, short desiredY);

  void moveEnemyCell(short currentX, short currentY, short desiredX, short desiredY);

  bool interactEnemyWithCell(unsigned short width, unsigned short heigth, Point enemyPosition);

  std::vector<Point> enemyPosotions;

private:

  void startTurn();

};

ПРИЛОЖЕНИЕ B **Исходный код программ**

**Файл Strategy.hpp**

#pragma once

class Strategy

{

public:

  ///@brief Функция принятия решения стратегией. Обращения к данной функции вызавет запуск одной из функций, определяющей поведение врага.

  virtual Direction makeDecision(Point enemy, Point player) = 0;

};

ПРИЛОЖЕНИЕ C **Исходный код программ**

**Файл Item.hpp**

#include "CellContent.hpp"

class Item : public IItem

{

public:

  Item() { }

  virtual ~Item() { }

  virtual void itemSpecialProperty();

  unsigned short attackBonus = 0;

  unsigned short defenseBonus = 0;

  unsigned short healthBonus = 0;

};

# ПРИЛОЖЕНИЕ

**Исходный код программ**

**Файл Item.hpp**

#pragma once

#include "IItem.hpp"

#include "CellContent.hpp"

#include "Logger/LogPublisher.hpp"

#include "Logger/Logger.hpp"

class Item : public IItem, public LogPublisher

{

public:

  Item();

  virtual ~Item();

  /// @brief Вывод характеристик предмета.

  virtual void itemSpecialProperty();

  unsigned short attackBonus = 0;

  unsigned short defenseBonus = 0;

  unsigned short healthBonus = 0;

};

# 