

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

Кривенцова Л.С.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Изучить понятие интерфейса и принцип полиморфизма, научиться реализовывать классы (в уместных случаях обобщённые интерфейсом) и осуществлять межклассовые отношения соблюдая полиморфизм.

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

Основные теоретические положения.

ООП

- Концепцию предложили Оле-Йохан Даль и Кристен Ньюгором
- Появилась из языка ALGOL
- Сохранение фрейма в динамической памяти
- Локальные переменные сохранялись после выхода из функции
- Полиморфизм через указатели на функции

Абстракция

- Отображение только существенной информации о мире с сокрытием деталей и реализации
- Выделение интерфейса
- Единицей абстракции может быть класс или файл

Инкапсуляция

- Связь кода и данных
- Защита от внешнего воздействия
- Основа инкапсуляции в ООП - класс

Наследование

- Механизм, с помощью которого один объект перенимает свойства другого
- Позволяет добавлять классу характеристики, делающие его уникальным
- Поддержка понятия иерархической классификации
- Уменьшение количества дублирующего кода

Полиморфизм

- Реализация принципа: Один интерфейс – множество реализаций
- Механизм, позволяющий скрыть за интерфейсом общий класс действий
- Виды полиморфизма:

- ✓ Статический

- ✓ Динамический
- ✓ Параметрический

Методы

- Функции определенные внутри структуры
- Отличие заключается в прямом доступе к полям структуры
- Обращение к методу аналогично обращению к полям

Неявный указатель this

- Методы реализованы как обычные функции, имеющие дополнительный параметр
- Неявный параметр является указателем типа структуры и имеет имя this
- Можно считать, что настоящая сигнатура методов следующая:
- Позволяет обратиться к полям объекта при перекрытии имен

Объявление и определение методов

- Как и для обычных функций можно разделять объявление и определение
- Объявление выносится в заголовочный файл (.h)
- Определение выносится в исходный файл (.cpp)

Перегрузка функций

- Определение нескольких функций с одинаковым именем, но отличающимся списком параметров (типами и/или количеством)

Инвариант класса

- Публичный интерфейс – список методов, доступный внешним пользователям класса
- Инвариант класса – набор утверждений, которые должны быть истинны применительно к любому объекту данного класса в любой момент времени, за исключением переходных процессов в методах объекта
- Для сохранения инварианта класса:
- Все поля должны быть закрытыми
- Публичные методы должны сохранять инвариант

Конструкторы

- Специальная функция объявляемая в классе
- Имя функции совпадает с именем класса
- Не имеют возвращаемого значения
- Предназначены для инициализации объектов

Списки инициализации

- Предназначены для инициализации полей
- Инициализации происходит в порядке объявления полей

Деструктор

- Специальные функции, объявляемые в классе
- Имя функции совпадает с именем класса, плюс знак ~ в начале
- Не имеют возвращаемого значения и параметров
- Предназначены для освобождения используемых ресурсов
- Вызывается автоматически при удалении экземпляра класса / структуры

Выполнение работы.

Ход решения:

1. Объявляются перечисления:

`enum TYPE{PASSABLE, NOPASS, IN, OUT};` - Перечисление состояний клетки – проходимая, непроходимая, вход и выход.

`enum OBJECT{NONE, FOOD, BOX, PORTAL, SMALLMONSTER, MEDIUMMONSTER, LARGEMONSTER};` - Перечисление элемента клетки: отсутствие, еда, клад, портал и три вида монстров.

`enum GO{RIGHT, LEFT, UP, DOWN, STOP, START};` - Перечисление позиции/направления движения персонажей: вправо, влево, вниз, вверх, неподвижно и стартовая позиция.

2. В интерфейс *Cell* были добавлены следующие обновления:

а. Чистые виртуальные функции:

`virtual TYPE GetType() = 0;` - геттер (получает значение полей, имеющих модификатор доступа *privat*) для доступа к полю типа клетки (поле будет реализовано в классах-наследниках).

`virtual OBJECT GetObject() = 0;` - геттер для получения значения поля,

хранящего тип элемента клетки (поле будет реализовано в классах-наследниках).

virtual void SetObject(OBJECT t)=0; - сеттер для установления значения поля, хранящего тип элемента клетки (поле будет реализовано в классах-наследниках).

virtual void SetKey(bool rich) = 0; - сеттер для установления значения поля, хранящего логическое значение ключа (содержит клетка ключ или нет) (поле будет реализовано в классах-наследниках).

virtual bool GetKey() = 0; - геттер для получения значения поля, хранящего логическое значение ключа (содержит клетка ключ или нет) (поле будет реализовано в классах-наследниках).

virtual void SetStep(GO step) = 0; - сеттер для установления значения поля, хранящего следующий шаг элемента игровой клетки (поле будет реализовано в классах-наследниках).

virtual GO GetStep() = 0; - геттер для получения значения поля, хранящего следующий шаг элемента игровой клетки (поле будет реализовано в классах-наследниках).

3. В класс клетки игрового поля *Cellule* (наследник *Cell*) были добавлены обновления:

а. Поля с модификатором доступа *private*:

OBJECT ossiru; - поле хранит тип элемента игровой клетки – один из элементов перечисления *OBJECT*.

bool treasure; - поле хранит логическую переменную, обозначающую наличие ценного клада на игровой клетке.

GO nextstep; - поле, хранящее следующий шаг элемента игровой клетки (один из вариантов перечисления *GO*).

б. Реализуются методы класса с модификатором доступа *public*:

Cellule(const Cellule& other); - Конструктор копирования класса. Принимает в качестве входного параметра константную ссылку на объект этого же класса. С помощью списка инициализации создаваемому объекту присваиваются данные – члены класса из исходного объекта.

Cellule& operator = (const Cellule& other); - Перегрузка оператора присваивания копированием. Принимает в качестве входного параметра константную ссылку на объект этого же класса. После проверки на попытку присвоить объект самому себе полям создаваемого объекта присваиваются поля объекта копирования.

Cellule(Cellule&& other); - Конструктор перемещения класса. Принимает в качестве параметра ссылку *rvalue* на тип класса. С помощью с помощью функции *swap* стандартной библиотеки значения полей исходного объекта присваиваются текущему объекту.

Cellule& operator=(Cellule&& other); - Перегрузка оператора присваивания перемещением. Принимает в качестве параметра ссылку *rvalue* на тип класса, возвращает ссылку на тип класса. После проверки на попытку присвоить объект самому себе, с помощью функции *swap* стандартной библиотеки значения полей исходного объекта присваиваются текущему объекту. Возвращается ссылка на текущий объект.

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

void SetObject(OBJECT t) final; - Ничего не возвращает, присваивает полю *оссиру* один из доступных типов элемента клетки. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

void SetKey(bool rich) final; - Ничего не возвращает, присваивает полю *treasure* значение логической переменной, говорящей о наличии ценностей на клетке. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

bool GetKey() final; - Возвращает значение логической переменной, говорящей о наличии ценностей на клетке. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

void SetStep(GO step) final; - Ничего не возвращает, присваивает полю *step* следующий шаг элемента клетки: направление движения или состояния покоя.

Предполагается отсутствие дальнейшего переопределения в случае наследования (final).

GO GetStep() final; - Возвращает значение переменной, хранящей следующий шаг элемента клетки: направление движения или состояния покоя.. Предполагается отсутствие дальнейшего переопределения в случае наследования (final).

4. Обновлён класс внешнего представления клетки игрового поля *CelluleView*.

Переопределен конструктор:

explicit CelluleView(Cellule& one, Player& someone); - Теперь он также принимает ссылку на объект игрока, чтобы вывести его внешнее представление, получив доступ к координатам.

5. Определяется интерфейс элемента игровой клетки *Interior*.

Содержит одну чистую виртуальную функцию:

virtual OBJECT GetObject() = 0; - геттер для доступа к полю типа данного элемента.

6. Определяется интерфейс персонажей игры - *Characters*, наследуемый от более широкого интерфейса *Interior*.

а. Объявляются чистые виртуальные функции:

virtual OBJECT GetObject() = 0; - описана в интерфейсе-родителе.

virtual void SetHealth(int a) = 0; - сеттер для установления значения поля здоровья персонажа.

virtual int GetHealth() = 0; - геттер для получения значения поля здоровья персонажа.

virtual int GetAttack() = 0; - геттер для получения значения поля атаки персонажа.

7. Определяется класс игрока *Player* – реализация интерфейса *Characters*.

а. Определяются поля с модификатором доступа *private*:

int health; - целое число, хранит количества жизней игрока.

int attack; - целое число, хранит значение атаки персонажа (какой урон

наносится врагам).

Coordinates location{};- поле хранит структуру координат, где находится игрок в данный момент игры.

bool key; - поле хранит значение логической переменной, определяющей наличие у игрока ключа для выхода с поля (победы).

OBJECT type; - поле хранит тип элемента игровой клетки (у игрока всегда *NONE*).

int coins; - целое число, хранит количество монет, собранных игроком.

b. Реализуются методы класса с модификатором доступа *public*:

explicit Player(); - Конструктор класса, в котором инициализируются поля.

void SetHealth(int a); - Ничего не возвращает, присваивает полю *health* количество жизней игрока на данный момент игры.

int GetHealth(); - Возвращает значение переменной, хранящей количество жизней игрока на данный момент игры.

int GetAttack(); - Возвращает значение переменной, хранящей урон, наносимый игроком в сражениях.

void SetKey(bool yes); - Ничего не возвращает, принимает и присваивает полю *key* состояние, говорящее о наличии/отсутствии ключа у игрока.

bool GetKey() const; - Возвращает значение переменной, хранящей количество жизней игрока на данный момент игры.

OBJECT GetObject(); - Возвращает значение переменной, хранящей тип элемента игрока.

void SetLocal(Coordinates place); - Ничего не возвращает, принимает и присваивает полю *location* актуальное местоположение персонажа на игровом поле.

Coordinates GetLocal(); - Возвращает значение переменной, хранящей актуальное местоположение персонажа на игровом поле.

void SetCoins(int number); - Ничего не возвращает, принимает и присваивает полю *coins* количество монет у игрока.

int GetCoins(); - Возвращает значение переменной, хранящей количество монет у игрока.

8. Определяется интерфейс вещей, доступных в игре - *Item*, наследуемый от более широкого интерфейса *Interior*.

a. Объявляются чистые виртуальные функции:

virtual OBJECT GetObject() = 0; - описана в интерфейсе-родителе.

virtual void interplay(Player& person) = 0; - функция взаимодействие игрока с вещью.

virtual ~Item(){}; - деструктор.

9. Определяется класс еды/аптечки *Heal* – реализация интерфейса *Item*.

a. Определяется поле с модификатором доступа *private*:

OBJECT type; - поле хранит тип элемента игровой клетки (у аптечки всегда *FOOD*).

b. Реализуются методы класса с модификатором доступа *public*:

Heal(): type(FOOD); - конструктор класса, инициализирующий поле списком инициализации.

void interplay(Player& person) final; - Ничего не возвращает, принимает ссылку на объект игрока, и через сеттер прибавляет к здоровью игрока число, получаемое случайно (в диапазоне 0 - 10) с помощью функции стандартной библиотеки - *rand()*. Так как по правилам игры максимальное количество жизней игрока – 10, если после аптечки показатель здоровья превышает норму, он устанавливается на значение 10. Таким образом еда лечит игрока. Предполагается отсутствие дальнейшего переопределения метода в случае наследования (*final*)

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

10. Определяется класс сундука *Box* – реализация интерфейса *Item*.

a. Определяются поля с модификатором доступа *private*:

OBJECT type; - поле хранит тип элемента игровой клетки (у сундука всегда *BOX*).

bool full; - поле хранит логическую переменную, обозначающую наличие клада(ключа) в сундуке.

bool open; - поле хранит логическую переменную, обозначающую был ли открыт этот сундук.

b. Реализуются методы класса с модификатором доступа *public*:

Box(bool filled): full(filled), type(BOX); - конструктор класса, инициализирующий поле списком инициализации. Принимает логическую переменную – наличие сокровища внутри сундука.

void interplay(Player& person) final; - Ничего не возвращает, принимает ссылку на объект игрока, и выстраивает взаимодействие сундука с ним следующим образом:

Открытие сундука стоит 1 монету. Если у игрока достаточно момент (проверяется через геттер), и после открытия оказывается, что сундук не пустой - через сеттер игрок получает ключ.

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

bool GetOpen(); - Возвращает логическую переменную – состояние сундука (был открыт или нет).

11. Определяется класс портала *Teleport* – реализация интерфейса *Item*.

a. Определяются поля с модификатором доступа *private*:

OBJECT type; - поле хранит тип элемента игровой клетки (у портала всегда *PORTAL*).

Coordinates firstplace; - поле хранит структуру координат, где находится один из входов в портал.

Coordinates secondplace; - поле хранит структуру координат, где находится второй из входов в портал.

b. Реализуются методы класса с модификатором доступа *public*:

Teleport(Coordinates points[2]): firstplace(points[0]), secondplace(points[1]), type(PORTAL){}; - конструктор класса,

инициализирующий поля списком инициализации. Принимает координаты клеток, на которых расположены входы в портал.

void interplay(Player& person) final; - Ничего не возвращает, принимает ссылку на объект игрока, и с помощью сеттера перемещает игрока ко входу другого портала. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

12. Определяется интерфейс врагов *Enemy*, наследуемый от более широкого интерфейса *Characters*.

Объявляются чистые виртуальные функции:

virtual void fight(Player& hero) = 0; - функция битвы.

virtual void hit(Player& hero) = 0; - функция успешного выстрела.

virtual void miss(Player& hero) = 0; - функция промаха.

virtual void SetHealth(int a) = 0; - описана в интерфейсе-родителе.

virtual int GetHealth() = 0; - описана в интерфейсе-родителе.

virtual int GetAttack() = 0; - описана в интерфейсе-родителе.

virtual OBJECT GetObject() = 0; - описана в интерфейсе-родителе.

virtual ~Enemy() {}; - деструктор.

13. Определяется класс малого монстра *Smonster* – реализация интерфейса *Enemy*.

а. Определяются поля с модификатором доступа *private*:

int health; - целое число, хранит количества жизней врага.

int attack; - целое число, хранит значение атаки персонажа (какой урон наносится игроку).

OBJECT type; - поле хранит тип элемента игровой клетки (у маленького врага всегда *SMALLMONSTER*).

int assailable[5]; - поле хранит массив, обозначающий конечности и их состояние (0 - неуязвимость, 1 - уязвимость).

bool goon; - поле хранит логическую переменную, обозначающую, должна ли битва продолжиться.

б. Реализуются методы класса с модификатором доступа *private*:

void hit(Player& hero); - функция принимает ссылку на игрока. Здоровье врага (*health*) уменьшается на значение поля атаки игрока. Если после этого у монстра не осталось жизней, выводится поздравительная надпись и игрок получает одну монету. Полю *goon* присваивается значение *false*. В противном случае выводится подбадривающая надпись о том, что битву нужно продолжить.

void miss(Player& hero); - функция принимает ссылку на игрока и ничего не возвращает. Выводит сообщение о промахе, здоровье игрока уменьшается на значение поля атаки монстра (*attack*). Если после этого у игрока не осталось жизней, полю *goon* присваивается значение *false* и игра завершается.

с. Реализуются методы класса с модификатором доступа *public*:

Smonster(): health(5), attack(1), type(SMALLMONSTER), goon(true); - конструктор класса. Списком инициализации присваивает полям значения, ничего не принимает. Случайным образом заполняется массив поля *assailable* (чтобы в нём хранилось три единицы и два нуля в случайном порядке).

void SetHealth(int a) final; - Ничего не возвращает, присваивает полю *health* количество жизней монстра на данный момент игры. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

int GetHealth() final; - Возвращает значение переменной, хранящей количество жизней монстра на данный момент игры. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

int GetAttack() final; - Возвращает значение переменной, хранящей урон, наносимый монстром в сражении. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

void fight(Player& hero); - Ничего не возвращает, принимает ссылку на объект игрока. Выводит инструкцию к сражению и количество жизней обоих соперников. Если битва продолжается (*goon = true*) то в зависимости от попадания вызывает функции *hit* или *miss*.

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

14. Определяется класс среднего монстра *Mmonster* – реализация интерфейса *Enemy*.

а. Определяются поля с модификатором доступа *private*:

int health; - целое число, хранит количества жизней врага.

int attack; - целое число, хранит значение атаки персонажа (какой урон наносится игроку).

OBJECT type; - поле хранит тип элемента игровой клетки (у маленького врага всегда *MEDIUMMONSTER*).

int assailable[5]; - поле хранит массив, обозначающий конечности и их состояние (0 - неуязвимость, 1 - уязвимость).

bool goon; - поле хранит логическую переменную, обозначающую, должна ли битва продолжиться.

б. Реализуются методы класса с модификатором доступа *private*:

void hit(Player& hero); - метод принимает ссылку на игрока. Здоровье врага (*health*) уменьшается на значение поля атаки игрока. Если после этого у монстра не осталось жизней, выводится поздравительная надпись и игрок получает одну монету. Полю *goon* присваивается значение *false*. В противном случае выводится подбадривающая надпись о том, что битву нужно продолжить.

void miss(Player& hero); - метод принимает ссылку на игрока и ничего не возвращает. Выводит сообщение о промахе, здоровье игрока уменьшается на значение поля атаки монстра (*attack*). Если после этого у игрока не осталось жизней, полю *goon* присваивается значение *false* и игра завершается.

с. Реализуются методы класса с модификатором доступа *public*:

Mmonster(): health(3), attack(2), type(MEDIUMMONSTER), goon(true); - конструктор класса. Списком инициализации присваивает полям значения, ничего не принимает. Случайным образом заполняется массив поля *assailable* (чтобы в нём хранилось три единицы и два нуля в случайном порядке).

void SetHealth(int a) final; - Ничего не возвращает, присваивает полю *health* количество жизней монстра на данный момент игры. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

int GetHealth() final; - Возвращает значение переменной, хранящей количество жизней монстра на данный момент игры. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*)

int GetAttack() final; - Возвращает значение переменной, хранящей урон, наносимый монстром в сражении. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*)

void fight(Player& hero); - Ничего не возвращает, принимает ссылку на объект игрока. Выводит инструкцию к сражению и количество жизней обоих соперников. Если битва продолжается (*goon = true*), подстреленная конечность уничтожается (соответствующей ей ячейке массива *assailable* устанавливается значение, равное нулю) и в зависимости от попадания вызывает функции *hit* или *miss*.

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

15. Определяется класс большого монстра *Lmonster* – реализация интерфейса *Enemy*.

а. Определяются поля с модификатором доступа *private*:

int health; - целое число, хранит количества жизней врага.

int attack; - целое число, хранит значение атаки персонажа (какой урон наносится игроку).

OBJECT type; - поле хранит тип элемента игровой клетки (у маленького врага всегда *LARGEMONSTER*).

int assailable[3]; - поле хранит массив, обозначающий конечности и их состояние (0 - неуязвимость, 1 - уязвимость).

bool goon; - поле хранит логическую переменную, обозначающую, должна ли битва продолжиться.

b. Реализуются методы класса с модификатором доступа *private*:

void hit(Player& hero); - метод принимает ссылку на игрока. Здоровье врага (*health*) уменьшается до нуля. Печатается поздравительная надпись и игрок получает две монеты. Полю *goon* присваивается значение *false*.

void miss(Player& hero); - метод принимает ссылку на игрока и ничего не возвращает. Выводит сообщение о промахе, здоровье игрока уменьшается на значение поля атаки монстра (*attack*). Если после этого у игрока не осталось жизней, полю *goon* присваивается значение *false* и игра завершается.

c. Реализуются методы класса с модификатором доступа *public*:

Lmonster(): health(100), attack(4), type(LARGEMONSTER), goon(true); - конструктор класса. Списком инициализации присваивает полям значения, ничего не принимает. Массив поля *assailable* инициализируется нулями, но одной из ячеек массива (выбранной случайным образом) присваивается значение, равное единице.

void SetHealth(int a) final; - Ничего не возвращает, присваивает полю *health* количество жизней монстра на данный момент игры. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

int GetHealth() final; - Возвращает значение переменной, хранящей количество жизней монстра на данный момент игры. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

int GetAttack() final; - Возвращает значение переменной, хранящей урон, наносимый монстром в сражении. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

void fight(Player& hero); - Ничего не возвращает, принимает ссылку на объект игрока. Выводит инструкцию к сражению и количество жизней обоих соперников. Если битва продолжается (*goon = true*), и в зависимости от попадания вызывает функции *hit* или *miss*.

OBJECT GetObject() final; – Возвращает тип элемента, находящемся на данной клетке игрового поля. Предполагается отсутствие дальнейшего переопределения в случае наследования (*final*).

16. Обновлён класс игрового поля *Field*.

а. Определяются поля с модификатором доступа *private*:

Coordinates Monsters*; - поле, хранящее массив структур *Coordinates* – координаты всех клеток, на которых располагаются монстры.

int monstercount; - поле, хранящее количество монстров на карте, для удобства работы с массивом местонахождения монстров.

Coordinates teleports[2]; - поле, хранящее массив структур *Coordinates* – координаты обоих входов в портал.

б. Реализуются методы класса с модификатором доступа *public*:

void MakeInOut(); - Метод класса, обеспечивающий появление клеток входа и выхода в игровом поле. Значение поля класса координат входа инициализируются с помощью функции *rand()* (в диапазоне размерности поля). Полю объекта *Cellule::Type* (хранящемуся в матрице игрового поля *Field::field* по индексу только что сгенерированных координат) через сеттер присваивается тип клетки «Вход» (*.SetType(IN)*). Аналогично генерируется клетка «Выхода» (в цикле *while* координаты генерируются случайным образом до тех пор, пока потенциальная клетка «Выхода» не будет лежать достаточно далеко от уже установленной клетки «Входа». Дальность – разность координат должна составлять минимум 20% длины координатной оси (длины и ширины)).

void MakeObjects(); - Метод класса, устанавливающий проходимость клеток. В ней запускается вложенный цикл *for*, в котором в каждой итерации создаётся структура координат игровой клетки (*Coordinates*), и с 10% шансом клетке игрового поля с координатами, хранящимися в созданной структуре присваивается (с помощью сеттера) тип клетки *NOPASS* – такая игровая клетка становится непроходимой.

void MakeItems(); - метод класса, распределяющий вещи по игровому полю. Генерирует их местоположение с помощью функции *rand()*, пока не получит, случайные координаты, по которым расположена свободная клетка.

void MakeEnemies(); - метод класса, распределяющий врагов по игровому полю. Генерирует их местоположение с помощью функции *rand()*, пока

не получит, случайные координаты, по которым расположена свободная клетка.

bool Access(Coordinates presentstate); - метод класса, проверяющий возможность передвижения объекта, находящегося на этой клетке поля, по его заявленной траектории. Принимает координаты нужной клетки. Сверяет, находятся ли новые координаты в диапазоне поля, а также состояние клетки, располагающейся по новым координатам. В случае соблюдения условий, возвращает *true*, иначе - *false*.

int GetHeight(); - геттер для доступа к полю *height*.

int GetWidth(); - геттер для доступа к полю *width*.

int GetMonsterCount(); - геттер для доступа к полю *monstercount*.

*Cellule** GetField();* - геттер для доступа к полю *field*.

Coordinates GetTeleports();* - геттер для доступа к полю *teleports*.

Coordinates GetMonsters();* - геттер для доступа к полю *Monsters*.

17. Объявляется класс внешнего представления клетки игрового поля *FieldView*.

a. Определяются поля с модификатором доступа *private*:

Field& GameField; - ссылка на объект поля, внешнее представление которого и будет храниться в объекте текущего класса.

Player& hero; - ссылка на объект игрока игры.

b. Реализуются методы класса с модификатором доступа *public*:

explicit FieldView(Field& one, Player& character): GameField(one), hero(character); - конструктор, принимающий ссылки на текущие поле и игрока. Списком инициализации присваивает полям начальные значения.

void PrintBorder() const; - Константный метод, является вспомогательным для метода *Print()*, в котором и вызывается. Печатает на экран горизонтальную границу поля. Этот функционал вынесен в отдельный метод только во избежание дублирования кода.

void Print() const; - Константный метод, печатающий актуальное состояние игрового поля на экран, получающее информацию о поле и его клетках через геттеры.

18. Объявляется класс игры *Game*, который связывает пользователя и внутренние классы, доступ к которым не должен предоставляться непосредственно.

a. Определяются поля с модификатором доступа *private*:

Field game; - поле, хранящее объект класса игрового поля.

Player person; - поле, хранящее объект класса игрока текущей игры.

b. Реализуются методы класса с модификатором доступа *public*:

explicit Game(); - конструктор класса, в нём создаются объекты классов поля и игрока, а также вызывается метод начала игры.

void NextCondition(Cellule& locate, Field& game); - метод класса, в котором определяется следующее направление движение монстра. С помощью оператора *switch* разбивается на случаи разных врагов, и в соответствии с логикой их передвижения (см. Правила игры) им определяется новое направление движения.

void Step(GO side, Coordinates begin, int k, Field& game); - метод класса, отвечающий за шаг передвижения. Принимает сторону, куда двигать персонажа, начальные координаты, индекс монстра в массиве врагов и ссылку на объект класса игрового поля. С помощью оператора *switch* разбивается на случаи, и определяются новые координаты объекта. Переприсваиваются необходимые поля и таким образом монстр передвигается на нужную клетку поля.

void EnemyMove(Field& game); - метод класса, отвечающий за ход врагов. Принимает ссылку на объект класса игрового поля. Запускает цикл, в котором вызывает метод определения следующего состояния врага и метод, непосредственно передвигающий персонажа на клетку.

void GameMove(Field& game); - метод класса, отвечающий за игровой ход. Принимает ссылку на объект класса игрового поля. Вызывает функцию передвижения врагов и осуществляет взаимодействие игрока с элементами клетки, на которой он оказался (с помощью оператора *switch* разбивается на случаи, при встрече вещи/врага создаётся объект нужного класса, запускается его метод, отвечающий за взаимодействия с игроком и если предмет исчерпан, он исчезает с

поля).

void Start(Field& game); - Метод класса, отвечающий за запуск игры. Устанавливает начальное состояние для функции *rand()* (*srand(time(NULL))*), вызывает методы, генерирующие состояния и элементы клеток, задаёт игроку начальную позицию (с помощью сеттера), запускает цикл игрового процесса, в котором чередуются функции печати поля на экран и игрового хода.

19. Обновлено главная функция *main()*.

Game firstGame; - создаётся объект класса игры.

Правила и логический ход игры:

Формирования поля:

На поле заданного размера (высота и ширина) появляются две клетки: вход и выход (их координаты получаются случайным образом, с учётом того, чтобы эти клетки не располагались рядом). С вероятностью 10% на каждой клетке появляется стена и делает участок непроходимым. Враги и вещи также появляются случайным образом, и их количество зависит от размеров поля.

Принцип игры:

Игрок появляется на клетке входа. Его задача – покинуть поле через клетку выхода. Однако открыть дверь выхода можно, только имея при себе ключ. Ключ спрятан в одном из сундуков, расположенных на поле. Открыть сундук (и забрать ключ, либо уйти ни с чем) можно, заплатив одну монету. Монеты игрок зарабатывает, сражаясь с монстрами, встречающимися ему на пути.

На поле можно подобрать еду, которая моментально лечит здоровье на случайную величину, не превышая лимит жизней игрока (10). Также на поле расположены два портала, позволяющие игроку телепортироваться с одной такой точки, на другую. При удачном расположении помогут игроку быстрее перемещаться по полю или обойти монстров.

Монстры:

Перемещаются, так же как и игрок, на одну клетку за игровой ход. Траектория пути маленького монстра – по кругу из четырех клеток (Вверх – вправо – вниз – влево, при необходимости пропуская некоторые шаги: например, если он

должен идти вправо, но там стена или предмет – он пойдёт вниз и так далее по циклу перемещения). Если монстр заблокирован – со всех сторон - он останавливается. Средний и большой монстр действуют аналогично, но средний по горизонтальной траектории (вправо, пока не упрётся в стенку, затем влево и т.д.), а большой – по направлению к игроку.

Персонажи (игрок и монстры) не могут передвигаться по диагонали.

Сражение с монстрами:

Боевая система игры – текстовая и устроена следующим образом:

Средний монстр имеет пять конечностей (две ноги, две руки и грудь), стрельба по трём из которых наносит врагу урон. Три уязвимых места определяются случайным образом и неизвестны игроку. При встрече с монстром система оповещает игрока о том, какого противника он встретил и предлагает выстрелить в одну из конечностей. При каждом промахе враг бьёт игрока (на значение, равное атаке врага). При попадании, монстру наносится урон (равный атаке игрока), а конечность уничтожается от удара, и теперь стрельба по ней не будет приносить урон. У среднего монстра три жизни – таким образом, игроку предстоит обнаружить и попасть по всем трём боевым точкам, пока враг не забрал все жизни. При самом неудачном раскладе и выигрышной стратегии игрока (если он выстрелит в обе неуязвимые точки, но при этом не будет бессмысленно стрелять в них дальше) игрок потеряет 4 жизни. После поражения враг исчезает с игрового поля.

Маленький монстр – аналог среднего, но его болевые точки не уничтожаются после удара. Следовательно, если игрок нашёл уязвимое место врага, он может стрелять в него, пока здоровье противника не иссякнет. При самом неудачном раскладе и выигрышной стратегии игрока игрок потеряет 2 жизни.

Большой монстр имеет всего три конечности – сердце, голова и живот. Урон наносится при ударе только по одному из указанных мест (болевая точка определяется случайным образом), зато этот удар – всегда смертельный. В худшем случае (но при выигрышной стратегии) игрок потеряет 8 жизней.

Во время битвы с любым из монстров можно сбежать, и тогда сражение

прекратится. Однако в этом случае игрок не получит награду, а противник не исчезнет с поля. К этому монстру можно будет вернуться и начать битву заново, но за время отсутствия игрока, враг восстановит свои жизни до максимума.

Пример битвы см. в «Результатах работы программы».

Внешний вид игры:

Поле, состояние игрока (ХП, монеты) печатаются на экране и дублируются с каждым новым ходом. Игровые обозначения:

Пробел – пустая проходимая клетка;

/ - стена, клетка непроходима;

1 – маленький монстр;

2 – средний монстр;

3 – большой монстр;

\$ - клад;

+ - еда (аптечка);

@ - телепорт;

F – игрок;

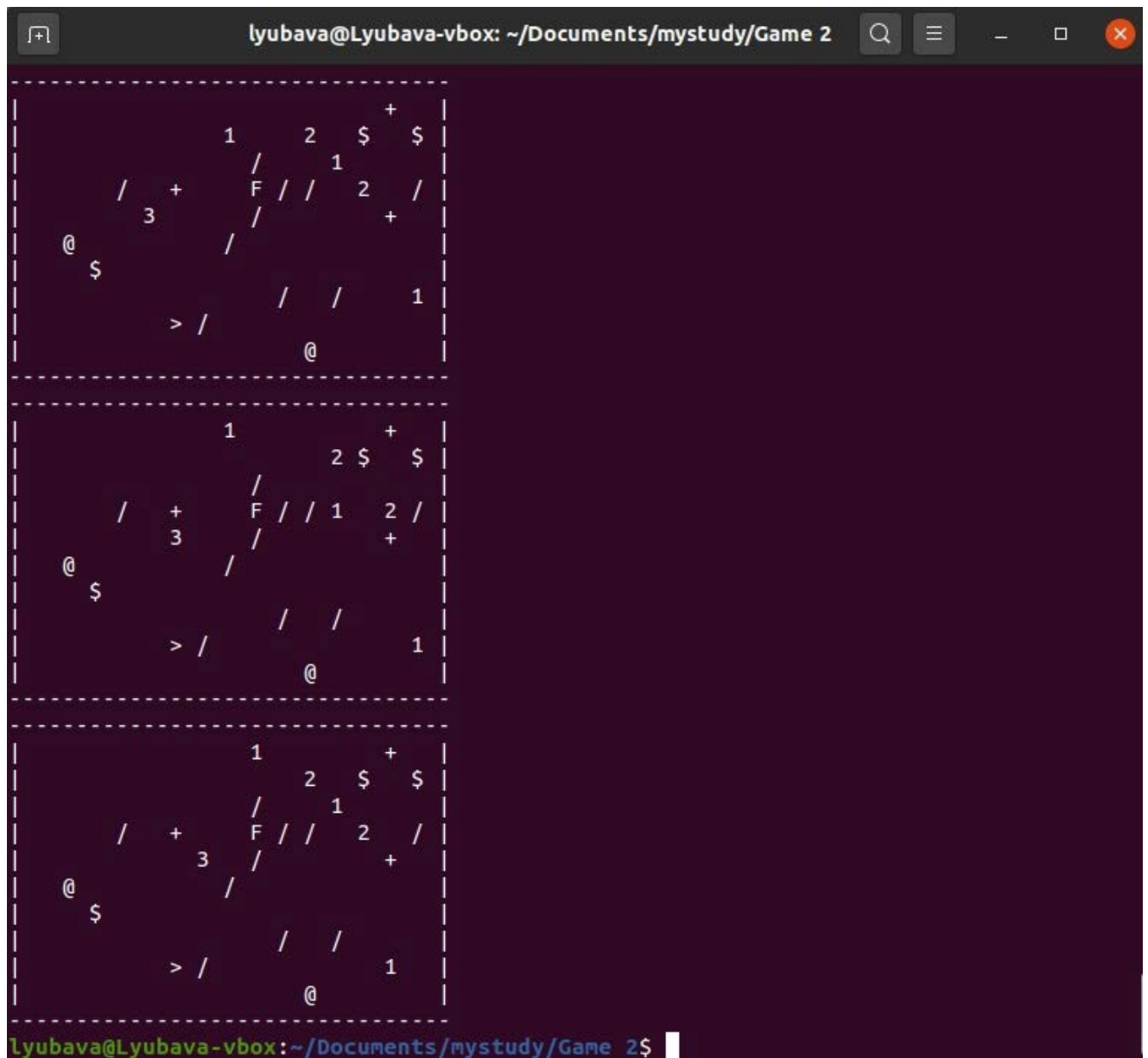
^ - вход;

> - выход;

| _ - границы поля.

Результат работы программы:

Рис 1. – демонстрация работы программы в терминале Ubuntu.



```
lyubava@Lyubava-vbox: ~/Documents/mystudy/Game 2
+-----+
|      1      2  $  +  $ |
|      /      /  /  1  2  / |
|    / +      F / / 2  +  / |
|  @  3      /      +      |
|  $      /      /      /  |
|      > /      /      1  |
|              @          |
+-----+
|      1      2  $  +  $ |
|      /      /  /  1  2  / |
|    / +      F / / 2  +  / |
|  @  3      /      +      |
|  $      /      /      /  |
|      > /      /      1  |
|              @          |
+-----+
|      1      2  $  +  $ |
|      /      /  /  1  2  / |
|    / +      F / / 2  +  / |
|  @  3      /      +      |
|  $      /      /      /  |
|      > /      /      1  |
|              @          |
+-----+
lyubava@Lyubava-vbox:~/Documents/mystudy/Game 2$
```

```

-----
Before you is a Usual Monster.
Your HP:  10      HP of the monster:  3
Where to shoot?
To the left hand: press 1
To the right hand: press 2
To the left leg: press 3
To the right leg: press 4
To the chest: press 5
Escape: press 0
2
This hand is destroyed.
Now it's useless to shoot at it.
Before you is a Usual Monster.
Your HP:  10      HP of the monster:  2
Where to shoot?
To the left hand: press 1
To the right hand: press 2
To the left leg: press 3
To the right leg: press 4
To the chest: press 5
Escape: press 0
4
This leg is destroyed.
Now it's useless to shoot at it.
Before you is a Usual Monster.
Your HP:  10      HP of the monster:  1
Where to shoot?
To the left hand: press 1
To the right hand: press 2
To the left leg: press 3
To the right leg: press 4
To the chest: press 5
Escape: press 0
3
You missed.
Try again.

```

```

You missed.
Try again.
Before you is a Usual Monster.
Your HP:  8      HP of the monster:  1
Where to shoot?
To the left hand: press 1
To the right hand: press 2
To the left leg: press 3
To the right leg: press 4
To the chest: press 5
Escape: press 0
5
The shot was fatal.
Who would have thought that such a giant had a weak point?
Chest is destroyed.
Now it's useless to shoot at it.

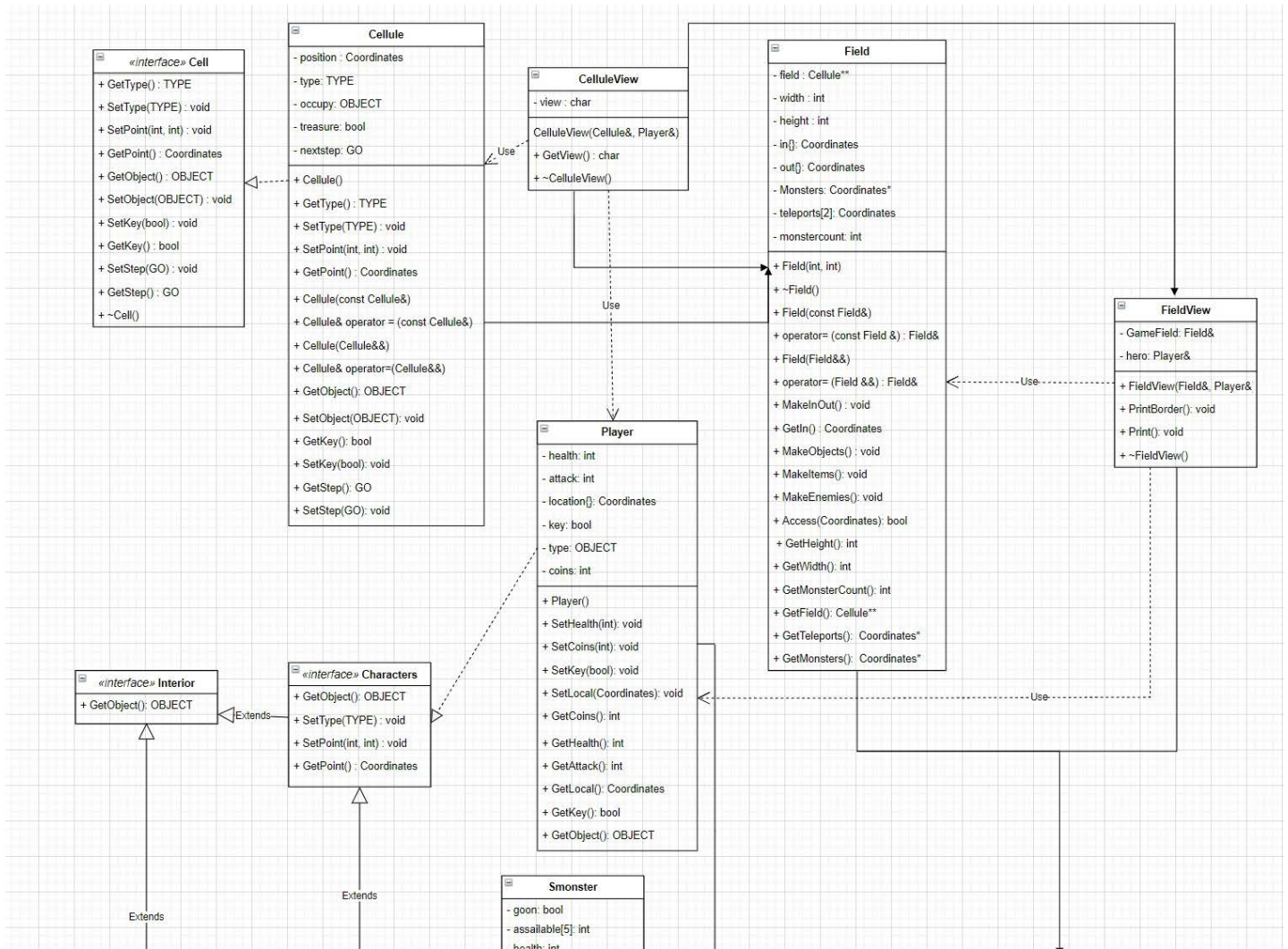
```

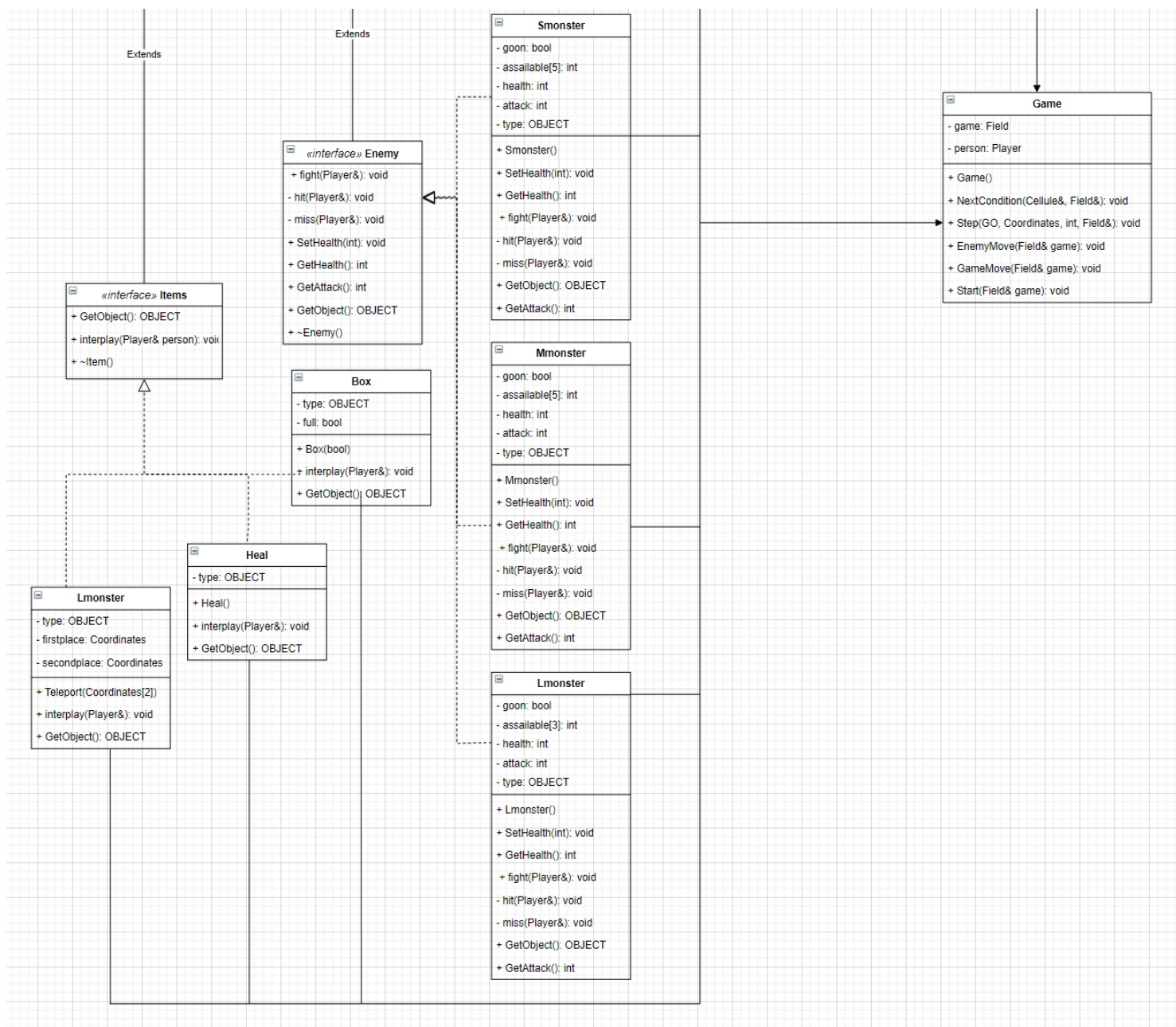
```

-----
| F  /          > 2      |
|          @          /   |
|          /          3 /  |
| 1      +      2  $  @ /  |
| $      $      /      1  |
|          +          ^   |
|      + / 1 /          |
|      /                |
|-----|

```


Рис 2. – UML-диаграмма.





Программный код см. в приложении А.

Выводы.

Были изучены понятия

интерфейса и принцип полиморфизма, получены навыки реализовывать классы (в уместных случаях обобщённые интерфейсом) и осуществлять межклассовые отношения соблюдая полиморфизм.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Game.h"
```

```
int main() {
    Game firstGame;
    return 0;
}
```

Название файла: structs.h

```
#pragma once
enum TYPE{PASSABLE, NOPASS, IN, OUT};
enum OBJECT{NONE, FOOD, BOX, PORTAL, SMALLMONSTER, MEDIUMMONSTER,
LARGEMONSTER};
enum GO{RIGHT, LEFT, UP, DOWN, STOP, START};

struct Coordinates{
    int x;
    int y;
};
```

Название файла: Cell.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
class Cell{
public:
    virtual TYPE GetType() = 0;
    virtual void SetType(TYPE t) = 0;
    virtual void SetPoint(int x, int y)=0;
    virtual Coordinates GetPoint() = 0;
    virtual OBJECT GetObject() = 0;
    virtual void SetObject(OBJECT t)=0;
    virtual void SetKey(bool rich) = 0;
    virtual bool GetKey() = 0;
    virtual void SetStep(GO step) = 0;
    virtual GO GetStep( ) = 0;
    virtual ~Cell(){};
};
```

Название файла: Cellule.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Cell.h"
```

```
class Cellule: public Cell {
```

```

public:
    Cellule();
    Cellule(const Cellule& other);
    Cellule& operator = (const Cellule& other);
    Cellule(Cellule&& other);
    Cellule& operator=(Cellule&& other);
    TYPE GetType() final;
    void SetType(TYPE t) final;
    void SetPoint(int a, int b) final;
    Coordinates GetPoint() final;
    OBJECT GetObject() final;
    void SetObject(OBJECT t) final;
    bool GetKey() final;
    void SetKey(bool rich) final;
    GO GetStep() final;
    void SetStep(GO step) final;

private:
    Coordinates position{};
    TYPE type;
    OBJECT occupy;
    bool treasure;
    GO nextstep;
};

    Название файла: Cellule.cpp
#include "Cellule.h"
    Cellule::Cellule():position({0,0}),    type(PASSABLE),    occupy(NONE),
nextstep(STOP), treasure(false){}

    Cellule::Cellule(const Cellule& other) : position(other.position),
type(other.type),
                                occupy(other.occupy),
nextstep(other.nextstep),    treasure(other.treasure){    //    Конструктор
копирования
    }

    Cellule& Cellule::operator = (const Cellule& other) { // Оператор
присваивания копированием
        if (this != &other){
            position = other.position;
            type = other.type;
            occupy = other.occupy;
            nextstep = other.nextstep;
            treasure = other.treasure;
        }
        return* this;
    }
    Cellule::Cellule(Cellule&& other) { // Конструктор перемещения
        std::swap(this->position, other.position);
        std::swap(this->type, other.type);
        std::swap(this->occupy, other.occupy);
        std::swap(this->nextstep, other.nextstep);
        std::swap(this->treasure, other.treasure);
    }

```

```

    Cellule& Cellule::operator=(Cellule&& other) { // Оператор присваивания
переменением
        if (this != &other) {
            std::swap(this->position, other.position);
            std::swap(this->type, other.type);
            std::swap(this->occupy, other.occupy);
            std::swap(this->nextstep, other.nextstep);
            std::swap(this->treasure, other.treasure);
        }
        return* this;
    }

    TYPE Cellule::GetType() {
        return this->type;
    }

    void Cellule::SetType(TYPE t) {
        this->type = t;
    }

    void Cellule::SetPoint(int a, int b) {
        this->position.x = a;
        this->position.y = b;
    }
    Coordinates Cellule::Cellule::GetPoint() {
        return this->position;
    }
    OBJECT Cellule::GetObject() {
        return this->occupy;
    }

    void Cellule::SetObject(OBJECT t) {
        this->occupy = t;
    }
    bool Cellule::GetKey() {
        return this->treasure;
    }

    void Cellule::SetKey(bool rich) {
        this->treasure = rich;
    }

    GO Cellule::GetStep() {
        return this->nextstep;
    }

    void Cellule::SetStep(GO step) {
        this->nextstep = step;
    }
}

Название файла: CelluleView.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>

```

```
#include "structs.h"
#include "Cell.h"
#include "Cellule.h"
#include "Player.h"
```

```
class CelluleView {
public:
    explicit CelluleView(Cellule& one, Player& someone);
    char GetView() const;
    virtual ~CelluleView();
private:
    char view;
};
```

Название файла: CelluleView.cpp

```
#include "CelluleView.h"

CelluleView::CelluleView(Cellule& one, Player& someone) {
    switch (one.GetType())
    {
        case PASSABLE:
            switch (one.GetObject()) {
                case BOX: {
                    this->view = '$';
                    break;
                }
                case FOOD: {
                    this->view = '+';
                    break;
                }
                case PORTAL: {
                    this->view = '@';
                    break;
                }
                case SMALLMONSTER: {
                    this->view = '1';
                    break;
                }
                case MEDIUMMONSTER: {
                    this->view = '2';
                    break;
                }
                case LARGEMONSTER: {
                    this->view = '3';
                    break;
                }
                case NONE: {
                    this->view = ' ';
                    break;
                }
            }
            break;
        case NOPASS:
            this->view = '/';
    }
```

```

        break;
    case OUT:
        this->view = '>';
        break;
    case IN:
        this->view = '^';
        break;
    default:
        this->view = '?';
        break;
    }
    if (someone.GetLocal().x == one.GetPoint().x &&
someone.GetLocal().y == one.GetPoint().y) this->view = 'F';
}

```

```

char CelluleView::GetView() const {
    return this->view;
}
CelluleView::~CelluleView(){};

```

Название файла: Interior.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
class Interior{
public:
    virtual OBJECT GetObject() = 0;
};

```

Название файла: Characters.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Interior.h"
class Characters: Interior{
public:
    virtual OBJECT GetObject() = 0;
    virtual void SetHealth(int a) = 0;
    virtual int GetHealth() = 0;
    virtual int GetAttack() = 0;
};

```

Название файла: Player.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Characters.h"
class Player: Characters{
public:
    explicit Player();

```

```

void SetHealth(int a);
int GetHealth();
int GetAttack();
void SetKey(bool yes);
bool GetKey() const;
OBJECT GetObject();
void SetLocal(Coordinates place);
Coordinates GetLocal();
void SetCoins(int number);
int GetCoins();

private:
    int health;
    int attack;
    Coordinates location{};
    bool key;
    OBJECT type;
    int coins;
};

Название файла: Player.cpp
#include "Player.h"
Player::Player() {
    location = {0,0};
    health = 10;
    key = false;
    attack = 1;
    type = NONE;
}
void Player::SetHealth(int a){
    this->health = a;
}
int Player::GetHealth() {
    return this->health;
}
int Player::GetAttack() {
    return this->attack;
}
void Player::SetKey(bool yes) {
    this->key = yes;
}
bool Player::GetKey() const{
    return this->key;
}
OBJECT Player::GetObject(){
    return this->type;
}
void Player::SetLocal(Coordinates place) {
    this->location = place;
}
Coordinates Player::GetLocal() {
    return this->location;
}
void Player::SetCoins(int number) {
    this->coins += number;
}

```



```

    int Player::GetCoins() {
        return this->coins;
    }

```

Название файла: Enemy.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Characters.h"
#include "Player.h"
class Enemy: Characters{
public:
    virtual void fight(Player& hero) = 0;
    virtual void hit(Player& hero) = 0;
    virtual void miss(Player& hero) = 0;
    virtual void SetHealth(int a) = 0;
    virtual int GetHealth() = 0;
    virtual int GetAttack() = 0;
    virtual OBJECT GetObject() = 0;
    virtual ~Enemy() {};
};

```

Название файла: Smonster.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Enemy.h"

class Smonster: public Enemy {
public:
    Smonster();
    void SetHealth(int a) final;
    int GetHealth() final;
    int GetAttack() final;
    OBJECT GetObject() final;
    void fight(Player& hero);
private:
    int health;
    int attack;
    OBJECT type;
    int assailable[5];
    bool goon;
    void hit(Player& hero);
    void miss(Player& hero);
};

```

Название файла: Smonster.cpp

```

#include "Smonster.h"
Smonster::Smonster():    health(5),    attack(1),    type(SMALLMONSTER),
goon(true){
    for (int i = 0; i < 5; i++) {
        assailable[i] = 1;
    }
    assailable[rand() % 5] = 0;
}

```

```

        int random = rand() % 5;
        while (assailable[random] == 0){
            random = rand() % 5;
        }
        assailable[random] = 0;
    }
    void Smonster::SetHealth(int a){
        this->health = a;
    }
    int Smonster::GetHealth(){
        return this->health;
    }
    int Smonster::GetAttack(){
        return this->attack;
    }
    OBJECT Smonster::GetObject(){
        return this->type;
    }

    void Smonster::fight(Player& hero){
        int body;
        while (health > 0 && goon){
            char wound;
            std::cout << "Before you is a Monster child." <<std::endl;
            std::cout << "Your HP:  " << hero.GetHealth() << "\t\tHP of the
monster:  " << this->health << std::endl;
            std::cout << "Where to shoot?" <<std::endl;
            std::cout << "To the left hand: press 1" <<std::endl;
            std::cout << "To the right hand: press 2" <<std::endl;
            std::cout << "To the left leg: press 3" <<std::endl;
            std::cout << "To the right leg: press 4" <<std::endl;
            std::cout << "To the chest: press 5" <<std::endl;
            std::cout << "Escape: press 0" <<std::endl;
            std::cin >> wound;
            switch (wound) {
                case '0':
                    goon = false;
                    break;
                case '1':
                    body = 0;
                    break;
                case '2':
                    body = 1;
                    break;
                case '3':
                    body = 2;
                    break;
                case '4':
                    body = 3;
                    break;
                case '5':
                    body = 4;
                    break;
                default:
                    body = -1;
            }
        }
    }

```

```

        break;
    }
    if (goon && body >= 0 && assailable[body] == 1) {
        hit(hero);
    }
    else {
        if (goon) miss(hero);
    }
}

void Smonster::hit(Player& hero) {
    this->health = health - hero.GetAttack();

    if (health <= 0){
        goon = false;
        std::cout << "The shot was fatal." << std::endl;
        std::cout << "Who would have thought that such a giant had a
weak point?" << std::endl;
        hero.SetCoins(1);
    }
    else {
        std::cout << "You hurt him! Keep it up!" << std::endl;
    }
}

void Smonster::miss(Player& hero){
    std::cout << "You missed." <<std::endl;
    hero.SetHealth(hero.GetHealth() - attack);
    if (hero.GetHealth() < 0){
        goon = false;
        std::cout << "The enemy is stronger than you..." <<std::endl;
        //Game Over
    }
    else
        std::cout << "Try again." <<std::endl;
}

```

Название файла: Mmonster.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Enemy.h"

class Mmonster: public Enemy {
public:
    Mmonster();
    void SetHealth(int a) final;
    int GetHealth() final;
    int GetAttack() final;
    OBJECT GetObject() final;
    void fight(Player& hero);
private:
    int health;
    int attack;
}

```

```

    OBJECT type;
    int assailable[5];
    bool goon;
    void hit(Player& hero);
    void miss(Player& hero);
};

    Название файла: Mmonster.cpp
#include "Mmonster.h"
    Mmonster::Mmonster():    health(3),    attack(2),    type(MEDIUMMONSTER),
goon(true){
    for (int i = 0; i < 5; i++) {
        assailable[i] = 1;
    }
    assailable[rand() % 5] = 0;
    int random = rand() % 5;
    while (assailable[random] == 0){
        random = rand() % 5;
    }
    assailable[random] = 0;
}
void Mmonster::SetHealth(int a){
    this->health = a;
}
int Mmonster::GetHealth(){
    return this->health;
}
int Mmonster::GetAttack(){
    return this->attack;
}
OBJECT Mmonster::GetObject(){
    return this->type;
}

void Mmonster::fight(Player& hero){
    int body;
    while (health > 0 && goon){
        char wound;
        std::cout << "Before you is a Usual Monster." <<std::endl;
        std::cout << "Your HP:    " << hero.GetHealth() << "\t\tHP of the
monster:    " << this->health << std::endl;
        std::cout << "Where to shoot?" <<std::endl;
        std::cout << "To the left hand: press 1" <<std::endl;
        std::cout << "To the right hand: press 2" <<std::endl;
        std::cout << "To the left leg: press 3" <<std::endl;
        std::cout << "To the right leg: press 4" <<std::endl;
        std::cout << "To the chest: press 5" <<std::endl;
        std::cout << "Escape: press 0" <<std::endl;
        std::cin >> wound;
        switch (wound) {
            case '0':
                goon = false;
                break;
            case '1':
                body = 0;
                break;

```

```

        case '2':
            body = 1;
            break;
        case '3':
            body = 2;
            break;
        case '4':
            body = 3;
            break;
        case '5':
            body = 4;
            break;
        default:
            body = -1;
            break;
    }
    if (goon && body >= 0 && assailable[body] == 1) {
        assailable[body] = 0;
        hit(hero);
        switch (body) {
            case 0:
            case 1:
                std::cout << "This hand is destroyed." << std::endl;
                break;
            case 2:
            case 3:
                std::cout << "This leg is destroyed." << std::endl;
                break;
            case 4:
                std::cout << "Chest is destroyed." << std::endl;
                break;
        }

        std::cout << "Now it's useless to shoot at it." << std::endl;
    }
    else {
        if (goon) miss(hero);
    }
}

void Mmonster::hit(Player& hero){
    this->health = health - hero.GetAttack();

    if (health <= 0){
        goon = false;
        std::cout << "The shot was fatal." << std::endl;
        std::cout << "Who would have thought that such a giant had a  
weak point?" << std::endl;
        hero.SetCoins(1);
    }
}

void Mmonster::miss(Player& hero){
    std::cout << "You missed." <<std::endl;
    hero.SetHealth(hero.GetHealth() - attack);
    if (hero.GetHealth() < 0){

```

```

        goon = false;
        std::cout << "The enemy is stronger than you..." <<std::endl;
        //Game Over
    }
    else
        std::cout << "Try again." <<std::endl;
}

Название файла: Lmonster.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Enemy.h"

class Lmonster: public Enemy {
public:
    Lmonster();
    void SetHealth(int a) final;
    int GetHealth() final;
    int GetAttack() final;
    OBJECT GetObject() final;
    void fight(Player& hero);
private:
    int health;
    int attack;
    OBJECT type;
    int assailable[3];
    bool goon;
    void hit(Player& hero);
    void miss(Player& hero);
};

Название файла: Lmonster.cpp
#include "Lmonster.h"
    Lmonster::Lmonster(): health(100), attack(4), type(LARGEMONSTER),
        goon(true){
        std::cout << health <<std::endl;
        for (int i = 0; i < 3; i++) {
            assailable[i] = 0;
        }
        assailable[rand() % 3] = 1;
    }
    void Lmonster::SetHealth(int a){
        this->health = a;
    }
    int Lmonster::GetHealth(){
        return this->health;
    }
    int Lmonster::GetAttack(){
        return this->attack;
    }
    OBJECT Lmonster::GetObject(){
        return this->type;
    }
}

```

```

void Lmonster::fight(Player& hero){
    int body;
    while (health > 0 && goon){
        char wound;
        std::cout << "Before you is a Big Monster." <<std::endl;
        std::cout << "Your HP:  " << hero.GetHealth() << "\t\tHP of the
monster:  " << this->health << std::endl;
        std::cout << "Where to shoot?" <<std::endl;
        std::cout << "To the head: press 1" <<std::endl;
        std::cout << "To the heart: press 2" <<std::endl;
        std::cout << "Into the belly: press 3" <<std::endl;
        std::cout << "Escape: press 0" <<std::endl;
        std::cin >> wound;
        switch (wound) {
            case '0':
                goon = false;
                break;
            case '1':
                body = 0;
                break;
            case '2':
                body = 1;
                break;
            case '3':
                body = 2;
                break;
            default:
                body = -1;
                break;
        }
        if (goon && body >= 0 && assailable[body] == 1) {
            hit(hero);
        }
        else {
            if (goon) miss(hero);
        }
    }
}

void Lmonster::hit(Player& hero){
    this->health = 0;
    goon = false;
    std::cout << "The shot was fatal." << std::endl;
    std::cout << "Who would have thought that such a giant had a weak
point?" << std::endl;
    hero.SetCoins(2);
}

void Lmonster::miss(Player& hero){
    std::cout << "You missed." <<std::endl;
    hero.SetHealth(hero.GetHealth() - attack);
    if (hero.GetHealth() < 0){
        goon = false;
        std::cout << "The enemy is stronger than you..." <<std::endl;
        //Game Over
    }
}

```

```

        else
            std::cout << "Try again." <<std::endl;
    }

    Название файла: Item.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Interior.h"
#include "Player.h"
class Item: Interior{
public:
    virtual OBJECT GetObject() = 0;
    virtual void interplay(Player& person) = 0;
    virtual ~Item(){};
};

    Название файла: Box.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Item.h"
class Box: public Item{
public:
    Box(bool filled);
    void interplay(Player& person) final;
    OBJECT GetObject() final;
    bool GetOpen();
private:
    OBJECT type;
    bool full;
    bool open;
};

    Название файла: Box.cpp
#include "Box.h"
    Box::Box(bool filled): full(filled), type(BOX){}

    void Box::interplay(Player& person){
        if (full && person.GetCoins() > 0){
            person.SetKey(true);
            person.SetCoins(-1);
            open = true;
        }
        else {
            std::cout << "Oops! .. Not enough coins." << std::endl;
            std::cout << "Please get the money and come back." << std::endl;
        }
    }

    OBJECT Box::GetObject(){
        return this->type;
    }
    bool Box::GetOpen(){

```



```

        return this->open;
    }
    Название файла: Heal.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Item.h"
class Heal: public Item{
public:
    Heal();
    void interplay(Player& person) final;
    OBJECT GetObject() final;
private:
    OBJECT type;
};
    Название файла: Heal.cpp
#include "Heal.h"
    Heal::Heal(): type(FOOD){}

    void Heal::interplay(Player& person){
        int a = rand() % 10;
        if (a + person.GetHealth() > 10)
            person.SetHealth(10);
        else person.SetHealth(a + person.GetHealth());
    };

    OBJECT Heal::GetObject(){
        return this->type;
    }
    Название файла: Teleport.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Item.h"
class Teleport: public Item{
public:
    Teleport(Coordinates points[2]);

    void interplay(Player& person) final;

    OBJECT GetObject() final;

private:
    OBJECT type;
    Coordinates firstplace;
    Coordinates secondplace;
};
    Название файла: Teleport.cpp
#include "Teleport.h"
    Teleport::Teleport(Coordinates points[2]): firstplace(points[0]),
secondplace(points[1]), type(PORTAL){}

```

```

void Teleport::interplay(Player& person){
    if (person.GetLocal().x == firstplace.x && person.GetLocal().y ==
firstplace.y)
        person.SetLocal(secondplace);
    else person.SetLocal(firstplace);
}

OBJECT Teleport::GetObject(){
    return this->type;
}

```

Название файла: Field.h

```

#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Cellule.h"
class Field {
public:
    Field(int w = 10, int h = 10);
    ~Field();

    Field(const Field &other);

    Field& operator = (const Field &other);

    Field(Field&& other);

    Field& operator=(Field&& other);

    void MakeInOut();

    Coordinates GetIn();

    void MakeObjects();

    void MakeItems();

    void MakeEnemies();

    bool Access(Coordinates presentstate);

    int GetHeight();

    int GetWidth();

    int GetMonsterCount();

    Cellule** GetField();

    Coordinates* GetTeleports();

    Coordinates* GetMonsters();

```

```
private:
    Cellule** field;
    int width;
    int height;
    Coordinates in{};
    Coordinates out{};
    Coordinates* Monsters;
    Coordinates teleports[2];
    int monstercount;
};
```

Название файла: Field.cpp

```
#include "Field.h"
#include <cmath>

Field::Field(int w = 10, int h = 10){
    width = w;
    height = h;
    monstercount = 0;
    field = new Cellule*[height];
    for (int i = 0; i < height; i++){
        field[i] = new Cellule[width];
        for (int j = 0; j < width; j++){
            field[i][j].SetPoint(j, i);
        }
    }
    Monsters = new Coordinates[10];
}

Field::~Field() {
    for (int i = 0; i < height; i++) {
        delete[] field[i];
    }
    delete[] field;
    delete[] Monsters;
}

Field::Field(const Field &other) : width(other.width),
height(other.height),
in(other.in), out(other.out), Monsters(new
Coordinates[monstercount]), monstercount(other.monstercount), field(new
Cellule*[height]) { // Конструктор копирования
    for (int i = 0; i < height; i++) {
        field[i] = new Cellule[width];
        for (int j = 0; j < width; j++){
            field[i][j] = other.field[i][j];
        }
    }
    for (int k = 0; k < monstercount; k++){
        Monsters[k] = other.Monsters[k];
    }
}

Field & Field:: operator = (const Field &other) { // Оператор
присваивания копированием
```

```

    if (this != &other){
        for (int i = 0; i < height; i++){
            delete[] field[i];
        }
        delete[] field;
        delete[] Monsters;
        width = other.width;
        height = other.height;
        in = other.in;
        out = other.out;
        monstercount = other.monstercount;
        field = new Cellule* [height];
        for (int i = 0; i < height; i++) {
            field[i] = new Cellule[width];
            for (int j = 0; j < width; j++){
                field[i][j] = other.field[i][j];
            }
        }
        Monsters = new Coordinates[monstercount];
        for (int k = 0; k < monstercount; k++){
            Monsters[k] = other.Monsters[k];
        }
    }
    return *this;
}

Field::Field(Field&& other){ // Конструктор перемещения
    std::swap(this->width, other.width);
    std::swap(this->height, other.height);
    std::swap(this->in, other.in);
    std::swap(this->out, other.out);
    std::swap(this->Monsters, other.Monsters);
    std::swap(this->monstercount, other.monstercount);
    std::swap(this->field, other.field);
}

Field & Field:: operator=(Field&& other) { // Оператор присваивания
перемещением
    if (this != &other) {
        std::swap(this->width, other.width);
        std::swap(this->height, other.height);
        std::swap(this->in, other.in);
        std::swap(this->out, other.out);
        std::swap(this->Monsters, other.Monsters);
        std::swap(this->monstercount, other.monstercount);
        std::swap(this->field, other.field);
    }
    return *this;
}

void Field::MakeInOut() {
    in = {rand() % width, rand() % height};
    field[in.y][in.x].SetType(IN);
    out = {rand() % width, rand() % height};
    while (abs(out.x - in.x) < round(0.2* width) || abs(out.y - in.y) <

```

```

round(0.2* height) ){

    out = {rand() % width, rand() % height};
}
field[out.y][out.x].SetType(OUT);
}

Coordinates Field::GetIn() {
    return this->in;
}

void Field::MakeObjects() {
    int chance;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            Coordinates any = {j, i};
            chance = rand() % 100 + 1;
            if ((chance <= 10) && (field[any.y][any.x].GetType() != IN)
&& (field[any.y][any.x].GetType() != OUT)) //10% chance
                field[any.y][any.x].SetType(NOPASS);
        }
    }
}

void Field::MakeItems() {
    teleports[0] = {rand() % width, rand() % height};
    while (field[teleports[0].y][teleports[0].x].GetType() != PASSABLE)
        teleports[0] = {rand() % width, rand() % height};
    teleports[1] = {rand() % width, rand() % height};
    while (field[teleports[1].y][teleports[1].x].GetType() != PASSABLE)
        teleports[1] = {rand() % width, rand() % height};

    field[teleports[0].y][teleports[0].x].SetObject(PORTAL);
    field[teleports[1].y][teleports[1].x].SetObject(PORTAL);

    int number = round(0.02 * width * height);
    Coordinates treasure;
    treasure = {rand() % width, rand() % height};
    while (field[treasure.y][treasure.x].GetType() != PASSABLE ||
field[treasure.y][treasure.x].GetObject() != NONE)
        treasure = {rand() % width, rand() % height};
    field[treasure.y][treasure.x].SetObject(BOX);
    field[treasure.y][treasure.x].SetKey(true);
    for (int i = 0; i < number - 1; i++) {
        treasure = {rand() % width, rand() % height};
        while (field[treasure.y][treasure.x].GetType() != PASSABLE ||
            field[treasure.y][treasure.x].GetObject() != NONE)
            treasure = {rand() % width, rand() % height};
        field[treasure.y][treasure.x].SetKey(false);
        field[treasure.y][treasure.x].SetObject(BOX);
    }
    Coordinates healing;
    for (int i = 0; i < number; i++) {
        healing = {rand() % width, rand() % height};
        while (field[healing.y][healing.x].GetType() != PASSABLE ||
field[healing.y][healing.x].GetObject() != NONE)

```

```

        healing = {rand() % width, rand() % height};
        field[healing.y][healing.x].SetObject(FOOD);
    }

}

void Field::MakeEnemies() {
    Coordinates monster;
    int number;
    OBJECT whichone;
    for (int j = 0; j < 3; j++){
        switch (j) {
            case 0 :
                number = round(0.02 * width * height);
                whichone = SMALLMONSTER;
                break;
            case 1:
                number = round(0.01 * width * height);
                whichone = MEDIUMMONSTER;
                break;
            case 2:
                number = 1;
                whichone = LARGEMONSTER;
                break;
        }
        for (int i=0; i < number; i++) {
            monster = {rand() % width, rand() % height};
            while (field[monster.y][monster.x].GetType() != PASSABLE ||
field[monster.y][monster.x].GetObject() != NONE)
                monster = {rand() % width, rand() % height};
            field[monster.y][monster.x].SetStep(START);
            Monsters[monstercount++] = {monster.x, monster.y};
            field[monster.y][monster.x].SetObject(whichone);
        }
    }
}

bool Field::Access(Coordinates presentstate){
    switch (field[presentstate.y][presentstate.x].GetStep()) {
        case RIGHT:
            presentstate.x++;
            break;
        case LEFT:
            presentstate.x--;
            break;
        case UP:
            presentstate.y--;
            break;
        case DOWN:
            presentstate.y++;
            break;
    }
    if ((presentstate.x >= 0) && (presentstate.x < width) &&
        (presentstate.y >= 0) && (presentstate.y < height) &&
        (field[presentstate.y][presentstate.x].GetType() == PASSABLE) &&
        (field[presentstate.y][presentstate.x].GetObject() == NONE))

```

```

        return true;
    else return false;
}

int Field::GetHeight() {
    return this->height;
}

int Field::GetWidth() {
    return this->width;
}

int Field::GetMonsterCount() {
    return this->monstercount;
}

Cellule** Field::GetField() {
    return this->field;
}

Coordinates* Field::GetTeleports() {
    return this->teleports;
}

Coordinates* Field::GetMonsters() {
    return this->Monsters;
}

Название файла: FieldView.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "CelluleView.h"
#include "Field.h"

class FieldView {
public:
    explicit FieldView(Field& one, Player& character);

    void PrintBorder() const;

    void Print() const;

    virtual ~FieldView();
private:
    Field& GameField;
    Player& hero;
};

Название файла: FieldView.cpp
#include "FieldView.h"
    FieldView::FieldView(Field& one, Player& character): GameField(one),
    hero(character){}

    void FieldView::PrintBorder() const {

```

```

        for (int i = 0; i < GameField.GetWidth() + 1; i++)
            std::cout << "--";
        std::cout << '-' << std::endl;
    }

    void FieldView::Print() const {
        PrintBorder();
        for (int i = 0; i < GameField.GetHeight(); i++){
            std::cout << "| ";
            for (int j = 0; j < GameField.GetWidth(); j++) {
                std::cout << CelluleView(GameField.GetField()[i][j],
hero).GetView() << ' ';
            }

            std::cout << '|' << std::endl;
        }
        PrintBorder();
    }
    FieldView::~FieldView(){};
Название файла: Game.h
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "structs.h"
#include "Player.h"
#include "Field.h"
#include "Box.h"
#include "Heal.h"
#include "Teleport.h"
#include "Smonster.h"
#include "Mmonster.h"
#include "Lmonster.h"
#include "FieldView.h"
class Game {
public:
    explicit Game();
    void NextCondition(Cellule& locate, Field& game);
    void Step(GO side, Coordinates begin, int k, Field& game);
    void EnemyMove(Field& game);
    void GameMove(Field& game);
    void Start(Field& game);
private:
    Field game;
    Player person;
};
Название файла: Game.cpp
#include "Game.h"
Game::Game(){
    Player person;
    Field game( 15, 10);
    Start(game);
}

void Game::NextCondition(Cellule& locate, Field& game){

```



```

int constant;
constant = 3;
GO array[4] = {RIGHT, DOWN, LEFT, UP};
switch (locate.GetObject()) {
    case SMALLMONSTER: {
        switch (locate.GetStep()) {
            case START:
                for (int i = 0; i <= constant; i++) {
                    locate.SetStep(array[i]);
                    if (game.Access(locate.GetPoint()))
                        break;
                }
                if (!game.Access(locate.GetPoint()))
                    locate.SetStep(STOP);
                break;
            default:
                do{
                    for (int i = 0; i <= constant; i++){
                        if (array[i] == locate.GetStep()){
                            if (i < constant)
                                locate.SetStep(array[i+1]);
                            else
                                locate.SetStep(array[i-constant]);
                            break;
                        }
                    }
                } while (!game.Access(locate.GetPoint()));
                break;
        }
    }
    case MEDIUMMONSTER:{
        switch (locate.GetStep()) {
            case START:
                for (int i = 0; i <= constant; i += 2) {
                    locate.SetStep(array[i]);
                    if (game.Access(locate.GetPoint()))
                        break;
                }
                if (!game.Access(locate.GetPoint()))
                    locate.SetStep(STOP);
                break;
            default:
                if (!game.Access(locate.GetPoint())) {
                    if (locate.GetStep() == array[0])
                        locate.SetStep(array[2]);
                    else if (locate.GetStep() == array[2])
                        locate.SetStep(array[0]);
                }
                break;
        }
        break;
    }
    case LARGEMONSTER:{
        if (abs(person.GetLocal().x - locate.GetPoint().x) <
            abs(person.GetLocal().y - locate.GetPoint().y)){

```

```

        if(person.GetLocal().y > locate.GetPoint().y)
locate.SetStep(DOWN);
        if(person.GetLocal().y < locate.GetPoint().y)
locate.SetStep(UP);
    }
    else{
        if(person.GetLocal().x > locate.GetPoint().x)
locate.SetStep(RIGHT);
        if(person.GetLocal().x < locate.GetPoint().x)
locate.SetStep(LEFT);
    }
    if (!game.Access(locate.GetPoint()))
        locate.SetStep(STOP);
    break;
}
}

void Game::Step(GO side, Coordinates begin, int k, Field& game){
    switch (side) {
        case RIGHT:
            game.GetMonsters()[k] = {begin.x + 1, begin.y};
            break;
        case LEFT:
            game.GetMonsters()[k] = {begin.x - 1, begin.y};
            break;
        case UP:
            game.GetMonsters()[k] = {begin.x, begin.y - 1};
            break;
        case DOWN:
            game.GetMonsters()[k] = {begin.x, begin.y + 1};
            break;
    }
    if (side != START && side != STOP){
        game.GetField()[begin.y][begin.x].SetObject(game.GetMonsters()[k].y[
game.GetMonsters()[k].x].SetObject(game.GetField()[begin.y][begin.x].GetOb
ject());
        game.GetField()[begin.y][begin.x].SetStep(game.GetMonsters()[k].y[
game.GetMonsters()[k].x].SetStep(game.GetField()[begin.y][begin.x].GetStep
());
        game.GetField()[begin.y][begin.x].SetStep(STOP);
        game.GetField()[begin.y][begin.x].SetObject(NONE);
    }
}

void Game::EnemyMove(Field& game)
{
    for (int k = 0; k < game.GetMonsterCount(); k++) {
        NextCondition(game.GetField()[begin.y][begin.x].GetStep(),
game.GetMonsters()[k].x, game);
        Step(game.GetField()[begin.y][begin.x].GetStep(),
game.GetMonsters()[k].x, game);
    }
}

void Game::GameMove(Field& game) {

```

```

Coordinates back;
EnemyMove(game);
back = person.GetLocal();
/*
 *
 */
switch
(game.GetField()[person.GetLocal().y][person.GetLocal().x].GetType()) {
    case NOPASS:
        person.SetLocal(back);
        break;
    case IN:
        person.SetLocal(back);
        break;
    case OUT:
        if (person.GetKey()) {
            std::cout << "Congratulations! You completed the game"
<< std::endl;
        }
        else {
            std::cout << "Oh no! The door is locked.\n";
            std::cout << "Please get the key and come back here!"
<< std::endl;
        }
        break;
    case PASSABLE: {
        if
(game.GetField()[person.GetLocal().y][person.GetLocal().x].GetObject() !=
NONE)
            switch
(game.GetField()[person.GetLocal().y][person.GetLocal().x].GetObject()) {
                case BOX: {
                    Box
keybox(game.GetField()[person.GetLocal().y][person.GetLocal().x].GetKey())
;
                    keybox.interplay(person);
                    if (keybox.GetOpen()){

game.GetField()[person.GetLocal().y][person.GetLocal().x].SetObject(NONE);

game.GetField()[person.GetLocal().y][person.GetLocal().x].SetKey(false);
                    }
                    break;
                }
                case FOOD: {
                    Heal healing;
                    healing.interplay(person);

game.GetField()[person.GetLocal().y][person.GetLocal().x].SetObject(NONE);
                    break;
                }
                case PORTAL: {
                    Teleport teleportation(game.GetTeleports());
                    teleportation.interplay(person);
                    break;
                }
                case SMALLMONSTER: {

```

```

        Smonster enemy;
        enemy.fight(person);
        if(enemy.GetHealth() <= 0) {
game.GetField()[person.GetLocal().y][person.GetLocal().x].SetObject(NONE);
        }
        break;
    }
    case MEDIUMMONSTER: {
        Mmonster enemy;
        enemy.fight(person);
        if(enemy.GetHealth() <= 0) {
game.GetField()[person.GetLocal().y][person.GetLocal().x].SetObject(NONE);
        }
        break;
    }
    case LARGEMONSTER: {
        Lmonster enemy;
        enemy.fight(person);
        if(enemy.GetHealth() <= 0) {
game.GetField()[person.GetLocal().y][person.GetLocal().x].SetObject(NONE);
        }
        break;
    }
}
}
break;
}
}

void Game::Start(Field& game) {
    srand(time(NULL));
    game.MakeInOut();
    game.MakeObjects();
    person.SetLocal(game.GetIn());
    game.MakeItems();
    game.MakeEnemies();
    FieldView(game, person).Print();
    for (int k = 0; k < 3; k++){
        GameMove(game);
        FieldView(game, person).Print();
    }
}

```