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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Логическое разделение классов

Студентка гр. 0324 _____ Жигалова Д. А.

Преподаватель _____ Глущенко А. Г.

Санкт-Петербург
2021

Цель работы.

Знакомство с паттернами проектирования.

Основные теоретические положения.

При создании программных систем перед разработчиками часто встает проблема выбора тех или иных проектных решений. В этих случаях на помощь приходят паттерны. Дело в том, что почти наверняка подобные задачи уже решались ранее и уже существуют хорошо продуманные элегантные решения, составленные экспертами. Если эти решения описать и систематизировать в каталоги, то они станут доступными менее опытным разработчикам, которые после изучения смогут использовать их как шаблоны или образцы для решения задач подобного класса. Паттерны как раз описывают решения таких повторяющихся задач.

Концепция создания программного обеспечения с использованием паттернов, несомненно, очень важная, но относительно молодая, быть может, поэтому до сих пор нет четкого определения, что же такое паттерн. Об этом свидетельствуют непрекращающиеся дискуссии в популярной литературе и на соответствующих форумах в сети.

Фасад — это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

Фасад — это простой интерфейс для работы со сложной подсистемой, содержащей множество классов. Фасад может иметь урезанный интерфейс, не имеющий 100% функциональности, которой можно достичь, используя сложную подсистему напрямую. Но он предоставляет именно те фичи, которые нужны клиенту, и скрывает все остальные.

Фасад полезен, если вы используете какую-то сложную библиотеку со множеством подвижных частей, но вам нужна только часть её возможностей.

К примеру, программа, заливающая видео котиков в социальные сети, может использовать профессиональную библиотеку сжатия видео. Но все, что

нужно клиентскому коду этой программы — простой метод encode(filename, format).

Посредник — это поведенческий паттерн проектирования, который позволяет уменьшить связанность множества классов между собой, благодаря перемещению этих связей в один класс-посредник. Паттерн Посредник заставляет объекты общаться не напрямую друг с другом, а через отдельный объект-посредник, который знает, кому нужно перенаправить тот или иной запрос. Благодаря этому компоненты системы будут зависеть только от посредника, а не от десятков других компонентов.

Команда — это поведенческий паттерн проектирования, который превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

Хорошие программы обычно структурированы в виде слоёв. Самый распространённый пример — слои пользовательского интерфейса и бизнес-логики. Первый всего лишь рисует красивую картинку для пользователя. Но когда нужно сделать что-то важное, интерфейс «просит» слой бизнес-логики заняться этим.

В реальности это выглядит так: один из объектов интерфейса напрямую вызывает метод одного из объектов бизнес-логики, передавая в него какие-то параметры.

Паттерн Команда предлагает больше не отправлять такие вызовы напрямую. Вместо этого каждый вызов, отличающийся от других, следует завернуть в собственный класс с единственным методом, который и будет осуществлять вызов. Такие объекты называют командами.

К объекту интерфейса можно будет привязать объект команды, который знает, кому и в каком виде следует отправлять запросы. Когда объект интерфейса будет готов передать запрос, он вызовет метод команды, а та — позаботится обо всём остальном.

Классы команд можно объединить под общим интерфейсом с единственным методом запуска. После этого одни и те же отправители смогут работать с различными командами, не привязываясь к их классам. Даже больше: команды можно будет взаимозаменять на лету, изменяя итоговое поведение отправителей.

Параметры, с которыми должен быть вызван метод объекта получателя, можно загодя сохранить в полях объекта-команды. Благодаря этому, объекты, отправляющие запросы, могут не беспокоиться о том, чтобы собрать необходимые для получателя данные. Более того, они теперь вообще не знают, кто будет получателем запроса. Вся эта информация скрыта внутри команды.

Постановка задачи.

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

Выполнение работы.

Для решения поставленных задач была создана программа на языке программирования C#.

Класс базы в данной игре не предусмотрен. Однако у игрок обладает такими характеристиками как, здоровье и время. Также идет отсчет уровней.

Все взаимодействия в игре реализованы с помощью паттерна Команда. Имеется абстрактный класс Action содержащий определение функции исполнения команды и вспомогательные функции для упрощения вывода результата команды в консоль и проверки типа результата команды. Далее от этого класса наследуются команды Move, Attack и Heal, реализующие перемещение юнитов по полю с взаимодействием их с разными типами юнитов, атаку одного юнита смертного типа другого юнита смертного типа, а также восполнение здоровья смертного юнита.

Выводы.

При разработке программы был реализован набор классов для взаимодействия пользователя с юнитами, а также паттерн Команда.

ИТОГОВЫЙ КОД ПРОГРАММЫ

Исходный код файла Program.cs

```
using System;

namespace course_work
{
    class Program
    {
        static void Main(string[] args)
        {
            gameManager.Instance.BaseUnitFactory = new BaseFactory();
            gameManager.Instance.MortalUnitFactory = new MortalFactory();
            gameManager.Instance.timer = 10;
            gameManager.Instance.w = 6;
            if (gameManager.Instance.level == 0) {
                int a = gameManager.Instance.w;
                gameManager.Instance.grid = new Grid(a, a);
            }
            string message = "";
            while (true)
            {
                Console.Clear();
                gameManager.Instance.grid.draw();
                if (message != "") { Console.WriteLine(message); message = ""; }
                if (gameManager.Instance.GameOver) {
                    Console.WriteLine($"Здоровье:
{gameManager.Instance.player.health} | Время: {gameManager.Instance.timer} |
Уровень: {gameManager.Instance.level}\n0 - Выйти");
                } else {
                    Console.WriteLine($"Здоровье:
{gameManager.Instance.player.health} | Время: {gameManager.Instance.timer} |
Уровень: {gameManager.Instance.level}\nПеремещение:\nw - Шаг вверх\nd - Шаг
вправо\ns - Шаг вниз\na - Шаг влево\n0 - Выйти");
                }
                string input = Console.ReadLine();
                int key = 0;
                if (input == "0") break;
                switch (input)
                {
                    case "w" or "a" or "s" or "d" when
!gameManager.Instance.GameOver:
                        if (input == "w") key = 1;
                        else if (input == "d") key = 2;
                        else if (input == "s") key = 3;
                        else if (input == "a") key = 4;
                        Action action = new MOVE(gameManager.Instance.player, key);
                        message = action.execute();
                        break;
                    default:
```

```
                        message = Action.fail("Моя твоя не понимать, еретик!");
                        break;
                }
            }
        }
    }
}
```

Исходный код файла grid.cs

```
using System.Collections.Generic;
using System;

namespace course_work {
    public enum cellType {
        Floor = 0,
        Wall = 1,
        Player = 4,
        Warrior = 2,
        Warrior2 = 5,
        Trap = 3,
        Heal = 6,
        DoorSwitch = 7,
        Door = 8
    }

    public class Grid {
        private List<IUnit> board {get; set;}
        public int width {get; set;}
        public int height {get; set;}
        public Grid(Grid copy) {
            this.width = copy.width;
            this.height = copy.height;
            this.board = new List<IUnit>(copy.board);
        }

        public Grid(int width, int height) {
            this.board = new List<IUnit>(width*height);
            this.width = width;
            this.height = height;
            int len = width * height;
            var rand = new Random();

            byte[] map = new byte[len];
            for (int i = 1; i < this.height-1; i++) {
                for (int j = 1; j < this.width-1; j++){
                    double c = rand.NextDouble();
                    if (c >= .7) {
                        map[Grid.oneDIndex(j,i,width)] = 2;
                    } else if (c >= .5) {
                        map[Grid.oneDIndex(j,i,width)] = 3;
```

```
                } else if (c <= .05) {
                    map[Grid.oneDIndex(j,i,width)] = 6;
                }
            }
        }
        map[Grid.oneDIndex(1,1, width)] = 4;
        map[Grid.oneDIndex(width-2,height-2, width)] = 7;

        for (int i = 0; i < width; i++) {
            map[i] = 1;
        }
        for (int i = len-width; i < len; i++) {
            map[i] = 1;
        }
        for (int i = 1; i < height-1; i++) {
            if (rand.NextDouble() >= .6) {
                map[Grid.oneDIndex(0, i, width)] = 8;
                map[Grid.oneDIndex(width-1, i, width)] = 8;
            } else {
                map[Grid.oneDIndex(0, i, width)] = 1;
                map[Grid.oneDIndex(width-1, i, width)] = 1;
            }
        }

        DoorSwitch ds = new DoorSwitch();
        gameManager.Instance.doorSwitch = ds;

        foreach (var item in map)
        {
            IUnit newUnit;
            switch (item)
            {
                case 0 or 1 or 6 or 8:
                    newUnit =
gameManager.Instance.BaseUnitFactory.GetUnit((cellType)item);
                    break;
                case 2 or 4 or 3:
                    newUnit =
gameManager.Instance.MortalUnitFactory.GetUnit((cellType)item);
                    if (newUnit as Player != null) {
                        gameManager.Instance.player = (Player)newUnit;
                    }
                    break;
                default:
                    newUnit = null;
                    break;
            }
            this.board.Add(newUnit);
        }
        this.board[len - width - 2] = ds;
```

```
        }
        public void draw() {
            for (int i = 0; i < this.height; i++) {
                for (int j = 0; j < this.width; j++){
                    System.Console.Write($"{this.board[Grid.oneDIndex(j, i,
this.width)]} ");
                }
                System.Console.WriteLine();
            }
        }
        public IUnit getCellDir(IUnit unit, int dir) {
            int index = this.getUnitIndex(unit);
            IUnit r;
            switch (dir)
            {
                case 1:
                    r = this.board[index-this.width];
                    break;
                case 2:
                    r = this.board[index+1];
                    break;
                case 3:
                    r = this.board[index+this.width];
                    break;
                case 4:
                    r = this.board[index-1];
                    break;
                default:
                    r = new Wall();
                    break;
            };
            return r;
        }
        public void moveUnit(IUnit unit, int i) {
            int index = this.getUnitIndex(unit);
            this.board[index] = new Floor();
            this.board[i] = unit;
        }
        public void addUnit(IUnit unit, int i) => this.board[i] = unit;
        public void removeUnit(IUnit unit) {
            int i = this.board.IndexOf(unit);
            if (i != -1) {
                this.board[i] = new Floor();
            }
        }
        public int dir2offset(int dir) {
            int offset = 0;
            switch (dir)
            {
                case 1:
```

```
                    offset -= this.width;
                    break;
                case 2:
                    offset += 1;
                    break;
                case 3:
                    offset += this.width;
                    break;
                case 4:
                    offset -= 1;
                    break;
                default:
                    break;
            };
            return offset;
        }
        public static int oneDIndex(int x, int y, int w) => y * w + x;
        public static cellType IUnit2cellType(IUnit unit) =>
(cellType)unit.getID();
        public int getUnitIndex(IUnit unit) => this.board.IndexOf(unit);
        public IEnumerable<IUnit> Units
        {
            get {
                for (int i = 0; i < this.board.Count; i++) yield return
this.board[i];
            }
        }
        private IEnumerator<IUnit> GetEnumerator()
        {
            for (int i = 0; i < this.board.Count; i++) yield return this.board[i];
        }
    }

}
```

Исходный код файла factory.cs

```
namespace course_work
{
    public abstract class UnitFactory {
        public IUnit GetUnit(cellType UnitT) => createUnit(UnitT);
        public abstract IUnit createUnit(cellType UnitT);
    }

    public class BaseFactory : UnitFactory {
        public override IUnit createUnit(cellType UnitT) {
            switch (UnitT)
                {
                    case cellType.Floor:
                        return new Floor();
                    case cellType.Wall:
```

```
                    return new Wall();
                case cellType.Heal:
                    return new HealPotion();
                case cellType.DoorSwitch:
                    return new DoorSwitch();
                case cellType.Door:
                    return new Door();
                default:
                    System.Console.WriteLine("Can't create type " + UnitT);
                    return null;
            }
        }
    }
    public class MortalFactory : UnitFactory {
        public override IUnit createUnit(cellType UnitT) {
            switch (UnitT)
                {
                case cellType.Warrior:
                    if (gameManager.Instance.level < 5) {
                        return new Warrior();
                    } else {
                        return new Warrior2();
                    }
                case cellType.Player:
                    return new Player();
                case cellType.Trap:
                    return new Trap();
                default:
                    System.Console.WriteLine("Can't create type " + UnitT);
                    return null;
            }
        }
    }
}
```

Исходный код файла units.cs

```
namespace course_work {
    public interface IUnit
    {
        public byte getID();
    }

    public abstract class Mortal {
        public byte health { get; set; }
    }


    public class Player : Mortal, IUnit {
        public Player () {this.health = 20;}
        public byte getID() => (byte)4;
        public override string ToString() => "@";
```

```
    }

    public class Warrior : Mortal, IUnit {
        public Warrior () {this.health = 2;}

        public byte getID() => (byte)2;

        public override string ToString() => "I";
    }

    public class Warrior2 : Mortal, IUnit {
        public Warrior2 () {this.health = 4;}

        public byte getID() => (byte)5;

        public override string ToString() => "V";
    }

    public class Trap : Mortal, IUnit {
        public Trap () {this.health = 2;}
        public byte getID() => (byte)3;

        public override string ToString() => ":";
    }
    public class HealPotion : IUnit {
        public HealPotion() {}
        public byte getID() => (byte)6;
        public override string ToString() => "+";
    }
    public class DoorSwitch : IUnit {
        public delegate void activate();
        public event activate activationEvent;
        public DoorSwitch() {}
        public void toggle() {
            if (activationEvent != null) activationEvent();
        }
        public byte getID() => (byte)7;
        public override string ToString() => "%";
    }
    public class Door : IUnit {
        public string gfx = "H";
        public bool isOpen = false;
        public Door() { if (gameManager.Instance.doorSwitch != null)
gameManager.Instance.doorSwitch.activationEvent += onSwitchToggle;}
        public void onSwitchToggle() {
            gameManager.Instance.doorSwitch.activationEvent -= onSwitchToggle;
            this.isOpen = true;
            this.gfx = "_";
        }
        public byte getID() => (byte)8;
```

```
        public override string ToString() => this.gfx;
    }
    public class Floor : IUnit {
        public Floor () {}
        public byte getID() => (byte)0;

        public override string ToString() => ".";
    }

    public class Wall : IUnit {
        public Wall () {}
        public byte getID() => (byte)1;

        public override string ToString() => "#";
    }
}
```

Исходный код файла commands.cs

```
using System;

namespace course_work {
    public abstract class Action {
        public abstract string execute();
        public static string fail(string a) {
            return "ПРОВАЛ: " + a;
        }
        public static string damage(string a) {
            return "УРОН: " + a;
        }
        public static string success(string a) {
            return "УСПЕХ: " + a;
        }
    }

    public class MOVE : Action {
        public IUnit unit;
        public int dir;
        public MOVE(IUnit unit, int dir) {
            this.unit = unit;
            this.dir = dir;
        }
        public override string execute() {
            Grid g = gameManager.Instance.grid;
            IUnit cell = g.getCellDir(unit, dir);
            cellType cellT = Grid.IUnit2cellType(cell);


            gameManager.Instance.timer -= 1;
            gameManager.Instance.CheckTimer();
            if (gameManager.Instance.player.health == 0) {
```

```
                gameManager.Instance.GameOver = true;

gameManager.Instance.grid.removeUnit((IUnit)gameManager.Instance.player);
                return fail("Game Over");
            }

            switch (cellT)
            {
                case cellType.Floor:
                    g.moveUnit(unit, g.getUnitIndex(unit) + g.dir2offset(dir));
                    return "";
                case cellType.Wall:
                    return fail("Не могу туда пойти!");
                case cellType.Warrior or cellType.Warrior2:
                    Action action1 = new ATTACK((Mortal)unit, (Mortal)cell, false);
                    string result1 = action1.execute();
                    if(result1.Contains("УРОН")) {
                        g.moveUnit(unit, g.getUnitIndex(unit) + g.dir2offset(dir));
                    }
                    return result1;
                case cellType.Trap:
                    Action action2 = new ATTACK((Mortal)unit, (Mortal)cell, true);
                    string result2 = action2.execute();
                    if(result2.Contains("УРОН") || result2.Contains("УСПЕХ")) {
                        g.moveUnit(unit, g.getUnitIndex(unit) + g.dir2offset(dir));
                    }
                    return result2;
                case cellType.Heal:
                    Action action3 = new HEAL();
                    g.moveUnit(unit, g.getUnitIndex(unit) + g.dir2offset(dir));
                    return action3.execute();
                case cellType.DoorSwitch:
                    ((DoorSwitch)cell).toggle();
                    g.moveUnit(unit, g.getUnitIndex(unit) + g.dir2offset(dir));
                    return success("Двери открываются, идите в них");
                case cellType.Door:
                    if(((Door)cell).isOpen) {
                        gameManager.Instance.w++;
                        gameManager.Instance.level++;
                        byte ph = gameManager.Instance.player.health;
                        gameManager.Instance.grid = new
Grid(gameManager.Instance.w, gameManager.Instance.w);
                        gameManager.Instance.timer = 10;
                        gameManager.Instance.player.health = ph;
                        return success("СЛЕДУЮЩИЙ УРОВЕНЬ");
                    } else {
                        return fail("Дверь закрыта");
                    }
                default:
                    return fail("ОшибОчка");
```

```
            }
        }
    }

    public class ATTACK : Action {
        public Mortal attacker;
        public Mortal reciever;
        public int dir;
        public bool trap;

        public ATTACK(Mortal a, Mortal r, bool t) {
            this.attacker = a;
            this.reciever = r;
            this.trap = t;
        }
        public override string execute() {
            if (!this.trap) {
                attacker.health -= reciever.health;
                attacker.health = (attacker.health >= 200) ? (byte)0 :
attacker.health;

                if (attacker.health == 0) {
                    gameManager.Instance.GameOver = true;
                    gameManager.Instance.grid.removeUnit((IUnit)attacker);
                    return fail("Game Over");
                } else {
                    gameManager.Instance.grid.removeUnit((IUnit)reciever);
                    return damage($"Игрок потерял {reciever.health} здоровья, но
победил врага");
                }
            } else {
                var rand = new Random();
                double c = rand.NextDouble();
                if (c >= .5) {
                    gameManager.Instance.timer -= (int)reciever.health;
                    gameManager.Instance.CheckTimer();
                    if (attacker.health == 0) {
                        gameManager.Instance.GameOver = true;
                        gameManager.Instance.grid.removeUnit((IUnit)attacker);
                        return fail("Game Over");
                    }
                    gameManager.Instance.grid.removeUnit((IUnit)reciever);
                    return damage($"Игрок застрял и потерял {reciever.health}
минуты времени");
                } else {
                    gameManager.Instance.grid.removeUnit((IUnit)reciever);
                    return success("Игрок успешно обезвредил ловушку");
                }
            }
        }
```

```
    }

    public class HEAL : Action {
        public HEAL() {}
        public override string execute()
        {
            gameManager.Instance.player.health = (byte)20;
            return success("Игрок восстановил всё свое здоровье");
        }
    }
}
```

Исходный код файла gameManager.cs

```
namespace course_work{
    public sealed class gameManager {
        private static gameManager _instance;
        private gameManager() {}
        public static gameManager Instance {
            get {
                if (_instance == null) {
                    _instance = new gameManager();
                    _instance.GameOver = false;
                }
                return _instance;
            }
        }
        public Grid grid {get; set;}
        public Player player {get; set;}
        public int level {get; set;}
        public int timer {get; set;}
        public bool GameOver {get; set;}
        public int w {get; set;}

        public DoorSwitch doorSwitch {get; set;}

        public bool CheckTimer() {
            if (_instance.timer <= 0) {
                _instance.timer = 10;
                _instance.player.health -= (byte)3;
                _instance.player.health = (_instance.player.health >= 200) ?
(byte)0 : _instance.player.health;
                return true;
            } else return false;
        }

        public BaseFactory BaseUnitFactory {get; set;}
        public MortalFactory MortalUnitFactory {get; set;}
    }
}
```

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Обозначения:

- Игрок – @
- Стена – #
- Пустые клетки –.
- Двери – H, после открытия - _
- Враг – I (отнимает 2 единицы здоровья)
- Враг 2 – V (отнимает 4 единицы здоровья)
- Ловушка –:
- Хилка – +
- Переключатель дверей – %

```
# # # # # #
H @ . I : H
# . I : . #
# I I I . #
H I . . % H
# # # # # #
Здоровье: 20 | Время: 10 | Уровень: 0
Перемещение:
w - Шаг вверх
d - Шаг вправо
s - Шаг вниз
a - Шаг влево
0 - Выйти
```

Рисунок 1 - Демонстрационный пример поля 1

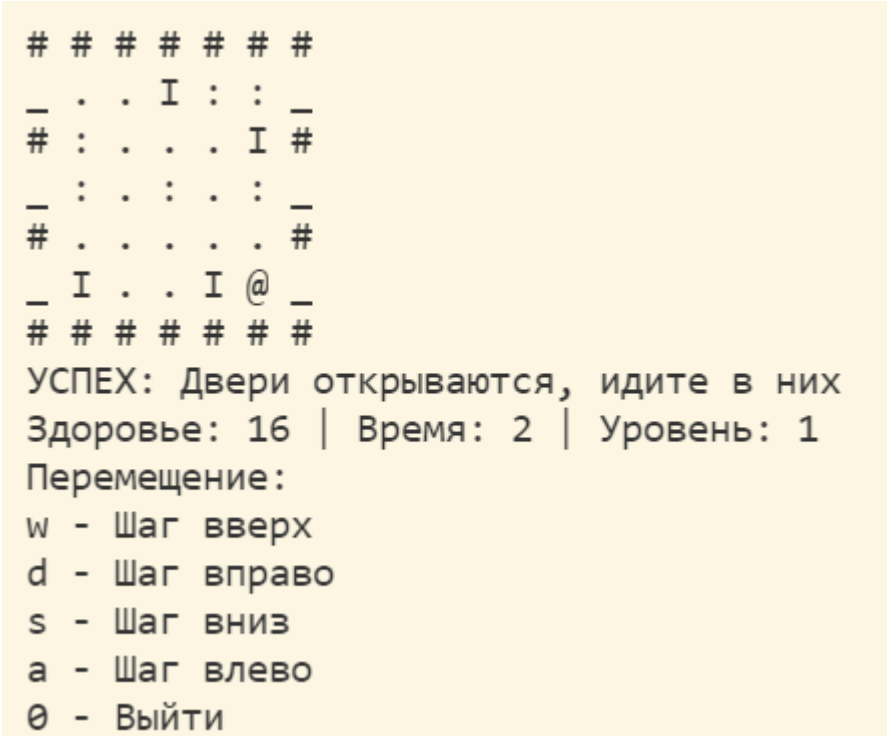

Рисунок 2 - Демонстрационный пример поля 2

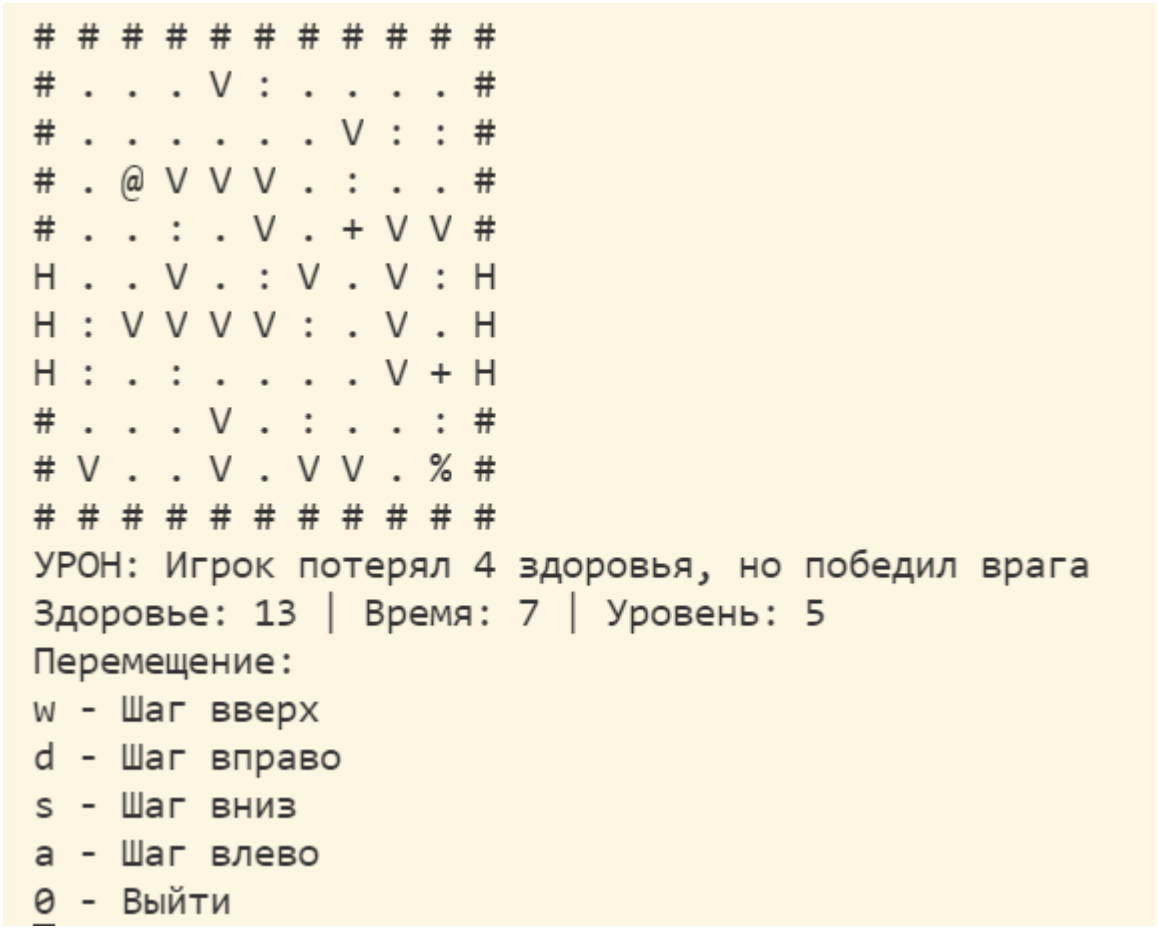

Рисунок 3 - Демонстрационный пример поля 3