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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Программирование»
Тема: Интерфейсы классов, взаимодействие классов, перегрузка операций

Студентка гр. 0324 _____ Жигалова Д. А.

Преподаватель _____ Глущенко А. Г.

Санкт-Петербург
2021

Цель работы.

Знакомство с интерфейсами классов, взаимодействием классов и перегрузкой операций.

Основные теоретические положения.

C++ позволяет переопределить действие большинства операций так, чтобы при использовании с объектами конкретного класса они выполняли заданные функции. Эта дает возможность использовать собственные типы данных точно так же, как стандартные. Обозначения собственных операций вводить нельзя. Можно перегружать любые операции, существующие в C++, за исключением: . , * ?: # ## sizeof

Перегрузка операций осуществляется с помощью методов специального вида {функций-операций) и подчиняется следующим правилам:

- при перегрузке операций сохраняются количество аргументов, приоритеты операций и правила ассоциации (справа налево или слева направо), используемые в стандартных типах данных;
- для стандартных типов данных переопределять операции нельзя;
- функции-операции не могут иметь аргументов по умолчанию;
- функции-операции наследуются (за исключением =);
- функции-операции не могут определяться как static.

Функцию-операцию можно определить тремя способами: она должна быть либо методом класса, либо дружественной функцией класса, либо обычной функцией. В двух последних случаях функция должна принимать хотя бы один аргумент, имеющий тип класса, указателя или ссылки на класс.

Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Шаги реализации

- Разбейте вашу функциональность на две части: независимое ядро и опциональные зависимые части. Независимое ядро станет издателем. Зависимые части станут подписчиками.

- Создайте интерфейс подписчиков. Обычно в нём достаточно определить единственный метод оповещения.

- Создайте интерфейс издателей и опишите в нём операции управления подпиской. Помните, что издатель должен работать только с общим интерфейсом подписчиков.

- Вам нужно решить, куда поместить код ведения подписки, ведь он обычно бывает одинаков для всех типов издателей. Самый очевидный способ — вынести этот код в промежуточный абстрактный класс, от которого будут наследоваться все издатели.

- Но если вы интегрируете паттерн в существующие классы, то создать новый базовый класс может быть затруднительно. В этом случае вы можете поместить логику подписки во вспомогательный объект и делегировать ему работу из издателей.

- Создайте классы конкретных издателей. Реализуйте их так, чтобы после каждого изменения состояния они отправляли оповещения всем своим подписчикам.

- Реализуйте метод оповещения в конкретных подписчиках. Не забудьте предусмотреть параметры, через которые издатель мог бы отправлять какие-то данные, связанные с происшедшим событием.

- Возможен и другой вариант, когда подписчик, получив оповещение, сам возьмёт из объекта издателя нужные данные. Но в этом случае вы будете вынуждены привязать класс подписчика к конкретному классу издателя.

- Клиент должен создавать необходимое количество объектов подписчиков и подписывать их у издателей.

Стратегия — это поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и помещает каждый из них в

собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

Паттерн Стратегия предлагает определить семейство схожих алгоритмов, которые часто изменяются или расширяются, и вынести их в собственные классы, называемые стратегиями.

Вместо того, чтобы изначальный класс сам выполнял тот или иной алгоритм, он будет играть роль контекста, ссылаясь на одну из стратегий и делегируя ей выполнение работы. Чтобы сменить алгоритм, вам будет достаточно подставить в контекст другой объект-стратегию.

Важно, чтобы все стратегии имели общий интерфейс. Используя этот интерфейс, контекст будет независимым от конкретных классов стратегий. С другой стороны, вы сможете изменять и добавлять новые виды алгоритмов, не трогая код контекста.

Постановка задачи.

Разработать и реализовать набор классов:

- Класс базы
- Набор классов ландшафта карты
- Набор классов нейтральных объектов поля

Класс базы должен отвечать за создание юнитов, а также учитывать юнитов, относящихся к текущей базе. Основные требования к классу база:

- База должна размещаться на поле
- Методы для создания юнитов
- Учет юнитов, и реакция на их уничтожение и создание
- База должна обладать характеристиками такими, как здоровье, максимальное количество юнитов, которые могут быть одновременно созданы на базе, и.т.д.

Набор классов ландшафта определяют вид поля. Основные требования к классам ландшафта:

Должно быть создано минимум 3 типа ландшафта

- Все классы ландшафта должны иметь как минимум один интерфейс
- Ландшафт должен влиять на юнитов (например, возможно пройти по клетке с определенным ландшафтом или запрет для атаки определенного типа юнитов)
- На каждой клетке поля должен быть определенный тип ландшафта

Набор классов нейтральных объектов представляют объекты, располагаемые на поле и с которыми могут взаимодействие юнитов. Основные требования к классам нейтральных объектов поля:

- Создано не менее 4 типов нейтральных объектов
- Взаимодействие юнитов с нейтральными объектами, должно быть реализовано в виде перегрузки операций
- Классы нейтральных объектов должны иметь как минимум один общий интерфейс

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.

Ландшафт создан с помощью классов Wall, Floor и Door. По клеткам Floor игрок проходит свободно, по клеткам Wall игрок не может ходить, а клетки Door открыты для игрока после взаимодействия им с определенным типом объекта.

К нейтральным объектам в созданной программе относятся Warrior, Trap, Heal и DoorSwitch. Все взаимодействия в игре реализованы с помощью паттерна Команда.

С объектами типа Warrior игрок может сражаться и теряет при этом свое здоровье. Объекты Trap – это ловушки, которые могут тратить ходы игрока с определённой вероятностью. Heal – объекты хиллок, которые возвращают первоначальное здоровье игроку.

DoorSwitch позволяет открыть двери для прохода на новый уровень. Открытие дверей реализуется через паттерн «Наблюдатель».

Выводы.

При разработке программы были изучены интерфейсы и взаимодействие классов, а также паттерны Наблюдатель и 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();
                if (message != "") { Console.WriteLine(message); message = ""; }
                gameManager.Instance.grid.draw();
                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");
                if (gameManager.Instance.GameOver) {
                    break;
                }
                string input = Console.ReadLine();
                int key = 0;
                if (input == "0") break;
                switch (input)
                {
                    case "w" or "a" or "s" or "d":
                        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 = "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,
        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:
                    return new Warrior();
                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 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 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;}
    }
}
```

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:
                    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");
        }
    }
}
```

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Обозначения:

- Игрок – @
- Стена – #
- Пустые клетки –.
- Двери – H, после открытия - _
- Враг – I
- Ловушка –:
- Хилка – +
- Переключатель дверей – %

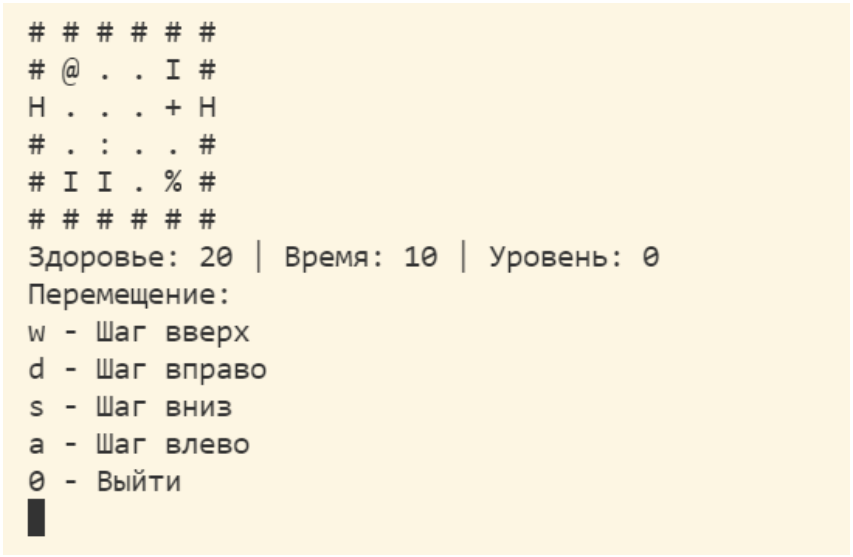

Рисунок 1 - Демонстрационный пример поля 1

```
УРОН: Игрок потерял 2 здоровья, но победил врага
# # # # # # # #
H . I . : : + H
H . . I . : I H
H . . @ . I . H
# I I . . . . #
# I . . : . : #
# . . . I I % #
# # # # # # # #
Здоровье: 18 | Время: 6 | Уровень: 2
Перемещение:
w - Шаг вверх
d - Шаг вправо
s - Шаг вниз
a - Шаг влево
0 - Выйти
█
```

Рисунок 2 - Демонстрационный пример поля 2

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # #
# . . . . . . : I : I I . . : . . + I . . . . . : I . #
# I . . I I . I I : . I . . I . I + I I + I . . I : I #
_ + : . + . I . I . : : : I : . I I I : I I : I . . . _
_ I I . : : . . I . + : : . I I . . : . . I + : . . : _
# . I . . I : I I . . . I I . . I . . . . : I I . . . #
_ . I . I . . . . . . I I . I I . : : . . : I I . : . _
_ + . . . . I I . . . : . + I : I I I . : . . . . . . _
_ . I I . . . . . . I I I . I I + : : : . : . I : . + _
_ + I I I I I . I I . . I . . . I : I I I : . I . : I _
# : I : I . . . I . . . : . . : . + I : : . . . I I . #
# : . . I . I . . . : . + + . I : I I . I . I . . . I #
# : . : . : I I I . I . : . . . . I + + I I . I : . . #
_ . I : I I + : . . . . I I . + . I I . . : I : . I . _
# . . . . . I I . . . . : + . I I . . I I . : I I . + #
# : . I : . . I : : . . + I : . I . I I . I I . I I . #
# . I . I . . I . . : . I I I I . I : : . : . I . . . #
_ + . . I I . I + + + . I . I . : . . : . I I . . . I _
# . : : + . . . . . . . : I . I I I : . I : I I + . . #
_ : . . . I I : I I . . I I . : I : I : : : . . + I I _
_ . . . . I : I I . . . . . . . . . : : I I I . : . + _
# : . I . : . . . : . : : : I + . . : + . . . : . . I #
# . I I . . : . : + : I . . I I : . . . : . : I : I : #
# : : I . I . . I I . . . + I . I I . . I . . I : I I #
_ . . I I . . . I . : I I I : I : @ . . . . . . . I . _
# . . . . I . . I . I + I . I I . I I I I : . I . . . #
_ . I . I . : I : I . . I : I : I . . : . . . . . . . _
# # # # # # # # # # # # # # # # # # # # # # # # # # # #
Здоровье: 20 | Время: 1 | Уровень: 22
Перемещение:
w - Шаг вверх
d - Шаг вправо
s - Шаг вниз
a - Шаг влево
0 - Выйти
█
```

Рисунок 3 - Демонстрационный пример поля 3