СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программирование – процесс и искусство создания компьютерных программ с помощью языков программирования. Программирование сочетает в себе элементы искусства, науки, математики и инженерии. По выражению одного из основателей языков программирования Никлауса Вирта, «Программы = алгоритмы + структуры данных». Программирование основывается на использовании языков программирования, на которых записываются исходные тексты программ. Сегодня это возможность реализации своих идей и возможностей в различных сферах. Ранее количество сфер, в которых нужны были информационные технологии, было ограниченно. Сейчас же новые технологии приходят даже в те сферы жизни человека, о которых ранее нельзя было и подумать.

Данный курсовой проект предусматривает написание простейшего windows-приложения, компьютерная игра «Пошаговая стратегия».

Для решения поставленной задачи, разработка программы будет происходить в среде Microsoft Visual Studio 2022 с использованием языка программирования C# и спецификации Windows Forms.

Windows Forms – это платформа пользовательского интерфейса для создания классических приложений Windows. Она обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в Visual Studio. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений.

В Windows Forms можно разрабатывать графически сложные приложения, которые просто развертывать, обновлять, и с которыми удобно работать как в автономном режиме, так и в сети. Приложения Windows Forms могут получать доступ к локальному оборудованию и файловой системе компьютера, на котором работает приложение.

C# – объектно-ориентированный язык программирования общего назначения. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework и .NET Core. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ

Пошаговая стратегия (англ. Turn-Based Strategy, TBS) – поджанр стратегических игр, в которых игровой процесс состоит из последовательности фиксированных моментов времени, именуемых ходами (или шагами), во время которых игроки совершают свои действия.

Компьютерные пошаговые стратегии происходят от настольных стратегических игр, в которых игроки, как правило, совершали действия по очереди. До 1990 года почти все стратегические компьютерные игры были пошаговыми. Большинство первых пошаговых стратегий были либо вариациями существующих настольных игр, либо в той или иной степени были вдохновлены ими. Основной характеристикой пошаговых стратегических игр является дискретность игрового процесса. Игра состоит из фиксированных во времени моментов («шагов» или «ходов»), которые завершаются только по команде игрока. Во время этих ходов игрок совершает свои действия. Один ход может соответствовать промежутку во много лет в игровом мире, за которые игрок успевает управиться с событиями в каждом городе империи и отдать приказы сотням военных отрядов.

Правила игры: Персонаж по имени Боб двигается по круговому подземелью. Персонаж имеет следующие в характеристики: атака, здоровье. Во время хода персонажа случайным образом появляется противник, с которым нужно сыграть в камень-ножницы-бумага. При победе противнику наносится урон, равный показателю атаки персонажа. В случае поражения противник наносит персонажу урон, равный показателю атаки противника. Также при ходе может появиться случайное событие, повышающее или понижающее показатели персонажа (атака, здоровье). При движении на 1 клетку персонаж получает 1 очко.

Следовательно, в нашей системе четко прослеживается необходимость в реализации следующих функций:

- генерация карты;
- движение персонажа;
- появление случайных событий;
- подсчёт счёта;
- появление противника;
- игра в камень-ножницы-бумага;

2 ПРОЕКТИРОВАНИЕ

Проектирование – процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект – целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы. Наряду с анализом требований, является частью большой стадии жизненного цикла системы, называемой определением системы. Результаты этой стадии являются входной информацией для стадии реализации системы.

Главный этап на стадии проектирования программы – это определение ее ключевых составных частей, каждая из которых несет ответственность за определенные задачи, решаемые в процессе выполнения программы, и является относительно автономной либо частично зависимой от других частей[**Ошибка! Источник ссылки не найден.**].

При запуске программы на экране будет появляться меню. После введения имени игрока и нажатия на кнопку «Start» запускаться окно с картой и расположенным на ней персонажем.

Для выполнения поставленной задачи должны быть спроектированы следующие основные модули:

- Game – модуль отрисовки игры;
- Menu – модуль появления меню;
- BattleWindow – модуль отрисовки окна битвы;
- ALifeUnit – характеристики основных юнитов;
- GameObject – используемые игровые объекты;
- Bob – характеристики героя;
- Rat – характеристики противника;
- World – генерация карты;
- Tile – генерация плитки карты;
- DebufWindow – генерация диалогового окна случайного события;

3 РЕАЛИЗАЦИЯ ПРОГРАММЫ

3.1 Модуль Game

В модуле Game находится 4 метода:

- void Start(object sender, EventArgs e);
- void button1_Click(object sender, EventArgs e);
- void Draw(object sender, PaintEventArgs e);
- void CreateNewObject(GameObject gameObject);

Метод Start(object sender, EventArgs e), который представлен в листинге 3.1, регистрирует и создает окно игры, отображая всю необходимую о персонаже информацию в окне.

Листинг 3.1 – Метод Start(object sender, EventArgs e)

```
private void Start(object sender, EventArgs e)
{
var world = new World(3,3,0,0,this);
_bob = new Bob(4, 6, 32, 32, this,world, 10, 3, 0, "Bob");
label2.Text = $"Attack / {_bob.Attack}";
label3.Text = $"Health / {_bob.Health}";
label4.Text = $"Score / {_bob.Score}";
}
```

Второй метод button1_Click(object sender, EventArgs e) отвечает за передвижение персонажа по карте. При нажатии на кнопку программа проверяет здоровье персонажа, и, если оно больше 0, то персонаж перемещается на следующую клетку, вызывая случайное событие. Если здоровье персонажа равно 0, метод закрывает игровое окно и возвращает в меню. Метод представлен в листинге 3.2.

Листинг 3.2 – Метод button1_Click(object sender, EventArgs e).

```
private void button1_Click(object sender, EventArgs e)
{
if (_bob.Health > 0)
{
StepSound.Play();
for (var i = 0; i < _gameObjects.Count; i++)
_gameObjects[i]?.Update();
_bob.Score += 1;
pictureBox1.Invalidate();
label2.Text = $"Attack / {_bob.Attack}";
label3.Text = $"Health / {_bob.Health}";
label4.Text = $"Score / {_bob.Score}";
}
else
{
Menu.AddToScoreList(_bob.Score);
```

Продолжение листинга 3.2

```
Menu.Show();
Close();
}
}
```

Третий метод Draw(object sender, PaintEventArgs e) представлен в листинге 3.3. Метод отвечает за отрисовку игрового процесса в поле PictureBox.

Листинг 3.3 – метод Draw(object sender, PaintEventArgs e).

```
private void Draw(object sender, PaintEventArgs e)
{
var graphics = e.Graphics;
for (var i = 0; i < _gameObjects.Count; i++)
_gameObjects[i]?.Draw(graphics);
}
```

3.2 Модуль Menu

Данный модуль является стартовым модулем, с которого начинается запуск программы. Модуль содержит 4 метода:

- void button1_Click(object sender, EventArgs e);
- void button2_Click(object sender, EventArgs e);
- public void AddToScoreList(int score);
- void checkBox1_CheckedChanged(object sender, EventArgs e)

Метод button1_Click(object sender, EventArgs e) представлен в листинге 3.4 и является методом, вызываемым при нажатии на кнопку старт.

Листинг 3.4 – Метод button1_Click(object sender, EventArgs e)

```
private void button1_Click(object sender, EventArgs e)
{
new Game(this);
MainSound.Stop();
Hide();
}
```

Второй метод – button2_Click(object sender, EventArgs e)представлен в листинге 3.5.Метод отвечает за выход из программы при нажатии на кнопку.

Листинг 3.5 – Метод button2_Click(object sender, EventArgs e)

```
private void button2_Click(object sender, EventArgs e)
{
Application.Exit();
}
```

Третий метод – AddToScoreList(int score) предназначен для добавления счёта в таблицу лидеров, расположенную в форме Menu. Метод представлен в листинге 3.6.

Листинг 3.6 – Метод AddToScoreList(int score)

```
public void AddToScoreList(int score)
{
listBox1.Items.Clear();
if (_scores.ContainsKey(textBox1.Text == string.Empty ? "Unknow" :
textBox1.Text))
_scores[textBox1.Text == string.Empty ? "Unknow" : textBox1.Text] =
_scores[textBox1.Text == string.Empty ? "Unknow" : textBox1.Text] <
score ? score : _scores[textBox1.Text == string.Empty ? "Unknow" :
textBox1.Text];
Else
_scores.Add(textBox1.Text == string.Empty ? "Unknow" :
textBox1.Text, score);
foreach (var s in _scores.OrderByDescending(x => x.Value))
listBox1.Items.Add($"{s.Key}:{s.Value}\n");
}
```

Четвёртый метод – checkBox1_CheckedChanged(object sender, EventArgs e) отвечает за фоновую музыку меню.Метод представлен в листинге 3.7.

Листинг 3.7 – Метод checkBox1_CheckedChanged(object sender, EventArgs e)

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
if(checkBox1.Checked == true)
{
MainSound.PlayLooping();
}
else
{
MainSound.Stop();
}
}
```

3.3 Модуль Battle Window

Данный модуль содержит файл BattleWindow.cs. Модуль взаимодействует с модулем Game. В файле реализован класс Battle Window, в котором имеются следующие методы:

– void AddUnits(ALifeUnit player, ALifeUnit enemy) – метод добавления персонажа на форму, представлен в листинге 3.8;

– void Draw(object sender, PaintEventArgs e) – метод визуального отображения персонажей на форме в поле, представлен в листинге 3.9;

– void button1_Click(object sender, EventArgs e) – метод выбора камня, представлен в листинге 3.10;

– void button2_Click(object sender, EventArgs e)– метод выбора ножниц, представлен в листинге 3.11;

– void button3_Click(object sender, EventArgs e)– метод выбора бумаги, представлен в листинге 3.12;

Листинг 3.8 – метод AddUnits(ALifeUnit player, ALifeUnit enemy)

```
public void AddUnits(ALifeUnit player, ALifeUnit enemy)
{
_player = player;
_enemy = enemy;
button1.Show();
button2.Show();
button3.Show();
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
}
```

Листинг 3.9 – метод Draw(object sender, PaintEventArgs e)

```
private void Draw(object sender, PaintEventArgs e)
{
var graphics = e.Graphics;
float playerSize = 256;
float enemySize = 256;
graphics.DrawImage(_ratSprite, 400,0, (int)playerSize,
(int)playerSize);
graphics.DrawImage(_playerSprite, 24, 24, enemySize, enemySize);
}
```

Листинг 3.10 – метод button1_Click(object sender, EventArgs e)

```
private void button1_Click(object sender, EventArgs e)
{
var ratChoose = _random.Next(0,3);
switch (ratChoose) {
case 0:
DrawSound.Play();
new DebufWindow("Ничья!");
break;
case 1:
GoodSound.Play();
new DebufWindow($"Победа! Вы наносите {_enemy.Name}
{_player.Attack} ед. урона");
_enemy.Health -= _player.Attack;
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
break;
case 2:
BadSound.Play();
```

Продолжение листинга 3.10

```
new DebufWindow($"Поражение! {_enemy.Name} наносит вам
{_enemy.Attack} ед. урона");
_player.Health -= _enemy.Attack;
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
break;
}
if (_enemy.Health <= 0)
{
WinSound.Play();
new DebufWindow("You Win");
Close();
}
else if(_player.Health <=0)
{
DefeatSound.Play();
new DebufWindow("You Lose((:(");
Close();
}
pictureBox1.Invalidate();
}
```

Листинг 3.11 – метод button2_Click(object sender, EventArgs e)

```
private void button2_Click(object sender, EventArgs e)
{
var ratChoose = _random.Next(0, 3);
switch (ratChoose)
{
case 0:
BadSound.Play();
new DebufWindow($"Поражение! {_enemy.Name} наносит вам
{_enemy.Attack} ед. урона");
_player.Health -= _enemy.Attack;
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
break;
case 1:
DrawSound.Play();
new DebufWindow("Ничья!");
break;
case 2:
GoodSound.Play();
new DebufWindow($"Победа! Вы наносите {_enemy.Name}
{_player.Attack} ед. урона");
_enemy.Health -= _player.Attack;
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
break;
}
```

Продолжение листинга 3.11

```
if (_enemy.Health <= 0)
{
WinSound.Play();
new DebufWindow("You Win");
Close();
}
else if(_player.Health <=0)
{
DefeatSound.Play();
new DebufWindow("You Lose((:(");
Close();
}
pictureBox1.Invalidate();
}
```

Листинг 3.12 – метод button3_Click(object sender, EventArgs e)

```
private void button3_Click(object sender, EventArgs e)
{
var ratChoose = _random.Next(0, 3);
switch (ratChoose)
{
case 0:
GoodSound.Play();
new DebufWindow($"Победа! Вы наносите {_enemy.Name}
{_player.Attack} ед. урона");
_enemy.Health -= _player.Attack;
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
break;
case 1:
BadSound.Play();
new DebufWindow($"Поражение! {_enemy.Name} наносит вам
{_enemy.Attack} ед. урона");
_player.Health -= _enemy.Attack;
label1.Text = $"Здоровье: {_player.Health}";
label2.Text = $"Здоровье: {_enemy.Health}";
break;
case 2:
DrawSound.Play();
new DebufWindow("Ничья!");
break;
}
if (_enemy.Health <= 0)
{
WinSound.Play();
new DebufWindow("You Win");
Close();
}
else if(_player.Health <=0)
```

Продолжение листинга 3.12

```
{
DefeatSound.Play();
new DebufWindow("You Lose((:(");
Close();
}
pictureBox1.Invalidate();
}
```

3.4 Модуль ALifeUnit

Данный модуль отвечает за хранение информации об используемых в игре юнитах. В ALifeUnit реализован абстрактный класс ALifeUnit, который связан с модулем GameObject и содержит конструктор ALifeUnit, инициализирующий все характеристики игровых юнитов. Модуль представлен в листинге 3.13.

Листинг 3.13 – Файл ALifeUnit.cs

```
using System.Media;
namespace BobAdv
{
public abstract class ALifeUnit : GameObject
{
public string Name { get; set; }
internal int Health { get; set; }
internal int Attack { get; set; }
internal int Score { get; set; }
public ALifeUnit(int x, int y, int width, int height, Game form,
int health, int attack,int score, string name) : base(x, y, width,
height, form)
{
Health = health;
Attack = attack;
Name = name;
Score = score;
}
      public abstract void BattleDraw(Graphics graphics);
      }
      }
```

3.5 Модуль GameObject

Данный модуль является абстрактным классом, от которого наследуются все классы, которые могут находиться в игре. Модуль отвечает за все игровые

объекты, создаваемые в окне игры (персонаж, карта, противник, клетка поля). Представлен в листинге 3.14

Листинг 3.14 – Абстрактный класс Game Object

```
      public abstract class GameObject
      {
      protected Game Form;
      public int X;
      public int Y;
      public int Width;
      public int Height;
      public GameObject(int x, int y, int width, int height, Game
form)
     {
      X = x;
      Y = y;
      Width = width;
      Height = height;
      Form = form;
      Form.CreateNewObject(this);
      }
      public abstract void Draw(Graphics graphics);
      public abstract void Update();
      public static Image RotateImage(Image img, float
rotationAngle)
{
Bitmap bmp = new Bitmap(img.Width, img.Height);
Graphics gfx = Graphics.FromImage(bmp);
gfx.TranslateTransform((float)bmp.Width / 2, (float)bmp.Height /
2);
gfx.RotateTransform(rotationAngle);
gfx.TranslateTransform(-(float)bmp.Width / 2, -(float)bmp.Height /
2);
gfx.InterpolationMode = InterpolationMode.HighQualityBicubic;
gfx.DrawImage(img, new Point(0, 0));
gfx.Dispose();
return bmp;
}
}
```

3.6 Модуль Bob

Модуль содержит файл Bob.cs. В файле содержится класс Bob. Этот класс используется только Game.cs.

Файл содержит информацию о герое и необходимые для создания персонажа характеристики (атака, здоровье, счёт, имя). Также в Bob содержится

механика передвижения персонажа по карте и механика генерации случайного сообщения (уменьшение и увеличение здоровья, уменьшение и увеличение атаки, режим ярости при пониженном здоровье и генерация окна нападения противника).

В файле реализованы следующие методы:

– void Draw(Graphics graphics) – метод отрисовки персонажа на карте, представлен в листинге 3.15;

– void BattleDraw(Graphics graphics) – метод отрисовки противника на карте , представлен в листинге 3.16;

– void Update() – метод обновления формы и генерации диалоговых окон баффов или нападения противника, представлен в листинге 3.17;

Листинг 3.15 – метод void Draw(Graphics graphics)

```
public override void Draw(Graphics graphics)
{
graphics.DrawImage(Sprite, new Point(X * Width, Y * Height));
}
```

Листинг 3.16 – метод void BattleDraw(Graphics graphics)

```
public override void BattleDraw(Graphics graphics)
{
graphics.DrawImage(Sprite, X * Width, Y * Height);
}
```

Листинг 3.17 – метод void Update()

```
public override void Update()
{
_world.MoveToNextTile(this);
var random = new Random();
var cube = random.Next(1, 10);
switch (cube)
{
case 1:
if (Attack > 0)
{
Attack -= 1;
DebufSound.Play();
new DebufWindow("О нет... Вы утомились. Ваша атака понижается на 1
ед.");
}
break;
case 2:
Attack += 1;
BufSound.Play();
new DebufWindow("Вы нашли странное зелье. Ваша атака повышается на
1 ед.");
break;
case 3:
Health += 1;
```

Продолжение листинга 3.17

```
BufSound.Play();
new DebufWindow("Вы нашли исцеляющий свиток. Ваше здоровье
повышается на 1 ед.");
break;
case 4:
if (Health > 1)
{
Health -= 1;
DebufSound.Play();
new DebufWindow("О нет... Вас укусил паук. Ваше здоровье падает на
1 ед.");
}
break;
case 5:
if ((Health == 1 && this.Attack < 2) || (this.Health < 4 &&
this.Attack == 0))
{
Health = 1;
Attack = 10;
new DebufWindow("Вы в ярости.Атака увеличена");
}
break;
case 6:
AttentionSound.Play();
new DebufWindow("На вас напала крыса");
var battleWindow = new BattleWindow();
battleWindow.AddUnits(this, new Rat(0, 0, 0, 0, Form, 0, 0, 0,
"Nafanya"));
break;
}
if (Health <= 0)
{
DefeatSound.Play();
new DebufWindow("You died");
Form.Menu.AddToScoreList(Score);
Form.Menu.Show();
Form.Close();
}
}
```

3.7 Модуль Rat

Данный модуль отвечает за генерацию и хранение информации о противнике. Характеристики противника генерируются случайным образом

(здоровье в диапазоне от 6 до 8, атака в диапазоне от 2 до 4). Содержит в себе конструктор Rat, представленный в листинге 3.18.

Листинг 3.18 – конструктор Rat

```
public Rat(int x, int y, int width, int height, Game form, int
health, int attack,int score, string name) : base(x, y, width,
height, form, health, attack,score, name)
{
var random = new Random();
Health = random.Next(6, 8);
Attack = random.Next(2, 4);
}
```

3.8 Модуль World

Модуль World используется программой для генерации карты, по которой будет передвигаться персонаж. Модуль содержит в себе файл World.cs, который имеет методы и классы, наследуемые от GameObject. Содержит 3 метода

Первый метод – SetToTile(GameObject gameObject) для подключения клетки хода к карте. Представлен в листинге 3.19.

Второй метод – MoveToNextTile(GameObject gameObject) используется программой для перемещения персонажа на следующую клетку. Представлен на листинге 3.20.

Третий метод – InitMap() используется для генерации карты. Карта представляет собой симметричную фигуру, в которой значения длины и ширины пути генерируются случайным образом. Метод представлен в листинге 3.21.

Листинг 3.19 – метод SetToTile(GameObject gameObject)

```
public void SetToTile(GameObject gameObject) {
var index = _random.Next(_tiles.Count);
_gameObjectInWorld.Add(new Tuple<GameObject, int>(gameObject,
index));
_gameObjectInWorld.Last().Item1.X = _tiles[index].X;
_gameObjectInWorld.Last().Item1.Y = _tiles[index].Y;
}
```

Листинг 3.20 – метод MoveToNextTile(GameObject gameObject)

```
public void MoveToNextTile(GameObject gameObject)
{
var findedObject = _gameObjectInWorld.FindIndex(x => x.Item1 ==
gameObject);
if (_gameObjectInWorld[findedObject].Item2 == _tiles.Count - 1)
{
_gameObjectInWorld[findedObject].Item1.X = _tiles[0].X;
_gameObjectInWorld[findedObject].Item1.Y = _tiles[0].Y;
_gameObjectInWorld[findedObject] = new Tuple<GameObject,
```

Продолжение листинга 3.20

```
int>(_gameObjectInWorld[findedObject].Item1, 0);
return;
}
_gameObjectInWorld[findedObject].Item1.X =
_tiles[_gameObjectInWorld[findedObject].Item2 + 1].X;
_gameObjectInWorld[findedObject].Item1.Y =
_tiles[_gameObjectInWorld[findedObject].Item2 + 1].Y;
_gameObjectInWorld[findedObject] = new Tuple<GameObject,
int>(_gameObjectInWorld[findedObject].Item1,
_gameObjectInWorld[findedObject].Item2 + 1);
}
```

Листинг 3.21 – метод InitMap()

```
private void InitMap()
{
Random x1r = new Random();
var x1Count = x1r.Next(10,25);
Random y1r = new Random();
var y1Count = y1r.Next(3,14);
Random x2r = new Random();
var x2Count = x2r.Next(10,25);
Random y2r = new Random();
var y2Count = y2r.Next(3, 14);
for (int x = X; x <= x1Count + X; x++)
_tiles.Add(new Tile(x, Y, 32, 32, Form, 0));
for (int y = Y ; y <= y1Count + Y; y++)
_tiles.Add(new Tile(X + x1Count, y, 32, 32, Form, 0));
for (int x = X+x1Count; x <=x1Count + x2Count + X; x++)
_tiles.Add(new Tile(x, Y+y1Count, 32, 32, Form, 0));
for (int y = Y + y1Count; y <= y1Count + y2Count + Y; y++)
_tiles.Add(new Tile(X + x1Count+x2Count, y, 32, 32, Form, 0));
for (int x = X + x1Count +x2Count; x >=x2Count + X; x--)
_tiles.Add(new Tile(x, Y + y1Count+ y2Count , 32, 32, Form, 0));
for (int y = Y + y1Count+y2Count; y >= y2Count + Y; y--)
_tiles.Add(new Tile(X+x2Count, y, 32, 32, Form, 0));
for (int x = X + x2Count; x >= X; x--)
_tiles.Add(new Tile(x, Y + y2Count, 32, 32, Form, 0));
for (int y = Y + y2Count; y >= Y; y--)
_tiles.Add(new Tile(X, y, 32, 32, Form, 0));
}
```

3.9 Модуль Tile

Модуль является наследуемым классом от Game Object. В классе Tile, представленном в листинге 3.22 содержатся все характеристики, необходимые программе для объявления и последующем расположении клетки на карте.

Листинг 3.22– класс Tile

```
public class Tile : GameObject
{
public Image Sprite =
Image.FromFile("../../../resources/tile.png");
public Tile(int x, int y, int width, int height, Game form, float
rotation) : base(x, y, width, height, form)
{Sprite = RotateImage(Sprite, rotation);}
public override void Draw(Graphics graphics)
{graphics.DrawImage(Sprite, X * Width, Y * Height);}
public override void Update() {}}
}
```

3.10 Модуль DebufWindow

Модуль DebufWindow отвечает за появление диалоговых окон, содержащих информацию о случайных событиях, появляющихся по ходу игры. DebufWindow содержит в себе один метод void button1_Click(object sender, EventArgs e), используемый программой для выхода из диалогового окна события. Метод представлен в листинге 3.23.

Листинг 3.23 – метод void button1_Click(object sender, EventArgs e)

```
private void button1_Click(object sender, EventArgs e)
{
Close();
}
```

Также модуль содержит в себе конструктор DebufWindow(string message), который содержит информацию о сообщении, передаваемом при появлении случайного события в модуле Bob. После нажатия на клавишу «ОК» игра продолжается, диалоговое окно пропадает. Конструктор DebufWindow представлен в листинге 3.24.

Листинг 3.24 – конструктор public DebufWindow(string message)

```
public DebufWindow(string message)
{
InitializeComponent();
label1.Text = message;
ShowDialog();
}
```

4 ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование программы – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом.

Тестирование программы проводится с помощью метода черного ящика.

Тестирование методом черного ящика – метод тестирования программного обеспечения, который предполагает, что внутренняя структура системы неизвестны тестировщику. Выбираются входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же известен, каким должен быть результат этой обработки. Тестирование черного ящика – углубление во внутренне устройство системы, за пределы ее внешних интерфейсов

Недостатки:

– тестирование не может производиться на ранних этапах: имеется необходимость ожидание создания пользовательского интерфейса;

– нельзя провести более тщательное тестирование, с покрытием большого количества путей выполнения программы.

Преимущества:

– для выполнения тестирования черного ящика не требуется большое количество специальных знаний;

– при использовании автоматизации тестирования на этом уровне, поддержка тестовых скриптов может оказаться достаточно полезной, если программа часто изменяется.

При запуске программы появляется меню. Пользователь видит фон с кнопками старта игры и выхода из игры, таблицу лидеров и поле для ввода имени (рисунок 4.1)

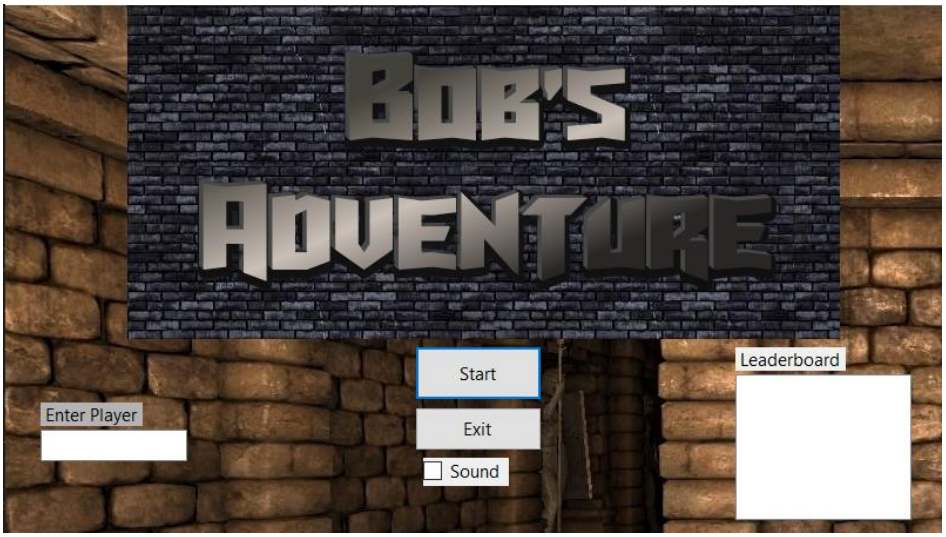

Рисунок 4.1 – Меню игры

При нажатии на кнопку «Start» появляется окно, содержащее игровое поле (рисунок 4.2), показатели персонажа, счёт и самого персонажа (рисунок 4.3)

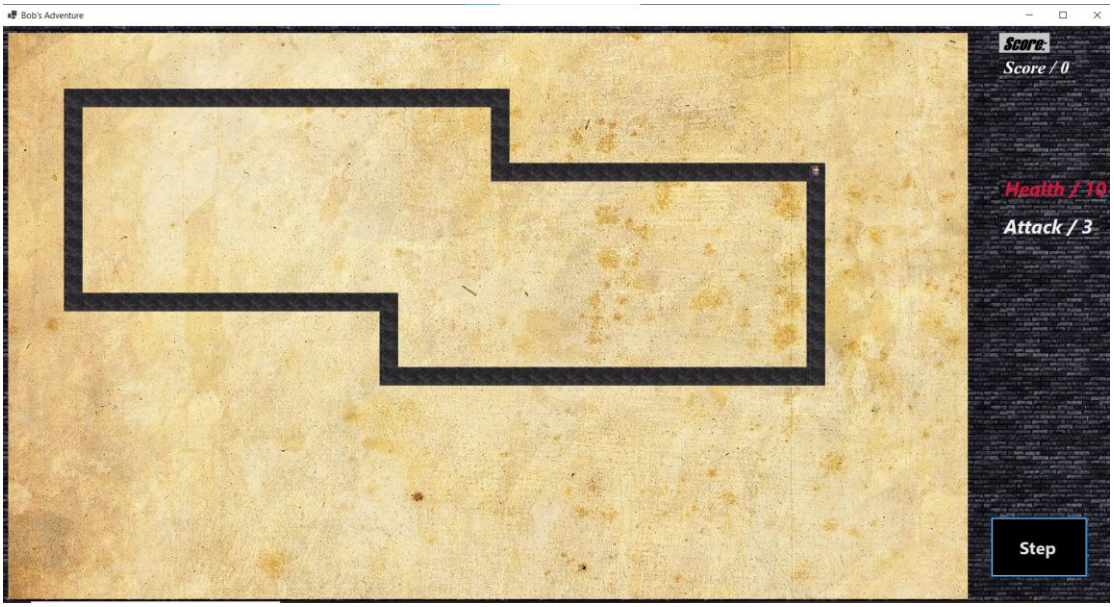

Рисунок 4.2 – Игровое поле

Рисунок 4.3 – Персонаж на поле

При нажатии на кнопку «Step» игрок перемещается на 1 клетку по часовой стрелке и ему случайным образом выпадает событие (рисунок 4.4)

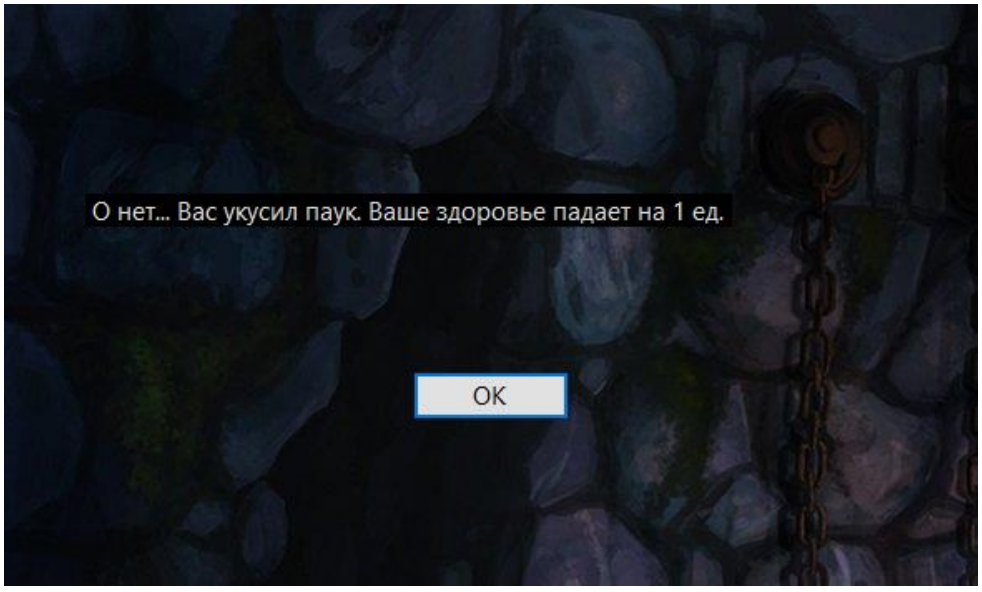

Рисунок 4.4 – Диалоговое окно случайного события

В процессе игры появляется случайное событие – нападение крысы (рисунок 4.5)

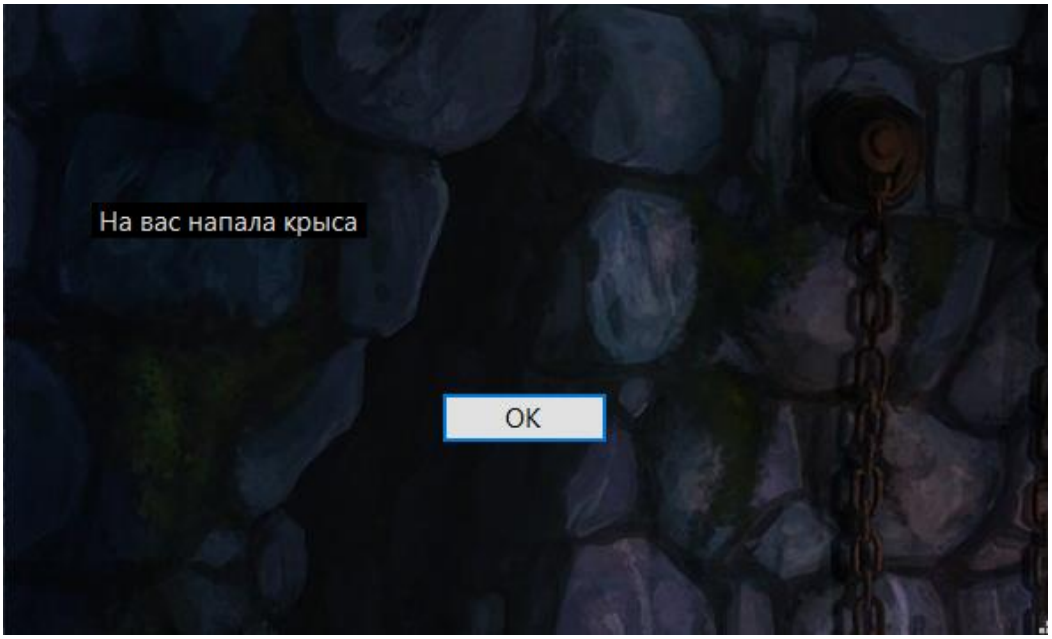

Рисунок 4.5 – Нападение крысы

После нажатия на кнопку «ОК» появляется окно игры в камень-ножницы-бумага (рисунок 4.6). Игрок выбирает один из предметов, и в случае победы наносит урон противнику (рисунок 4.7),в случае поражения противник наносит урон герою(рисунок 4.8). В случае выбора одинаковых предметов (Ничья) урон не наносится ни противнику, ни игроку (рисунок 4.9). Каждое из действий сопровождается звуковым сигналом.

Рисунок 4.6 – Окно битвы

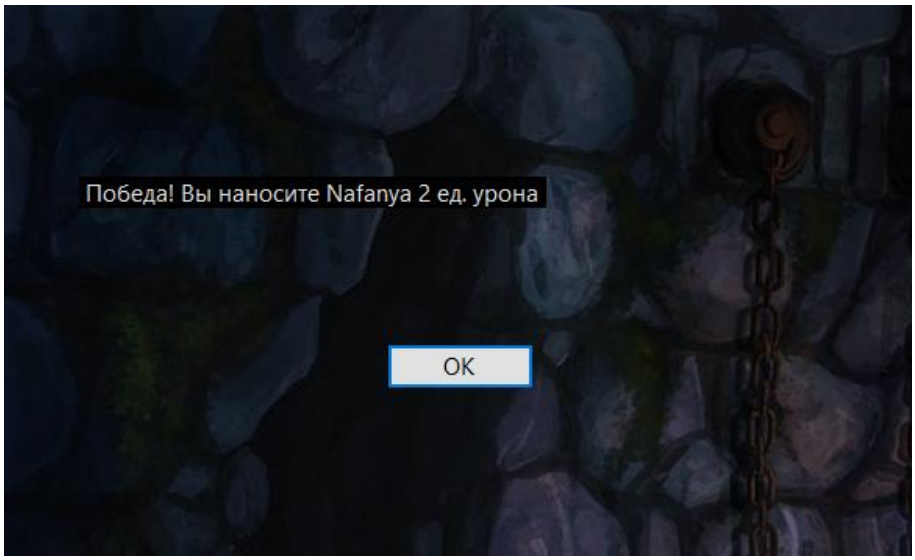

Рисунок 4.7 – Победа

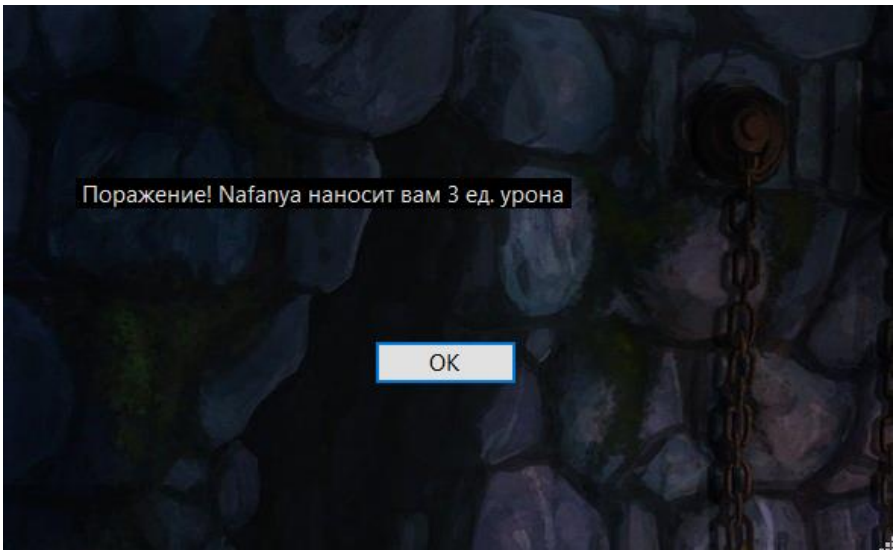

Рисунок 4.8 – Поражение

Рисунок 4.9 – Ничья

В случае, когда здоровье противника становится равным 0 появляется окно победы и игра продолжается. В случае, когда игрок теряет все очки здоровья игра заканчивается (рисунок 4.10) и отправляет игрока в Главное меню, где в поле LeaderBoard появляется информация об игроке и набранных им очках (рисунок 4.11).

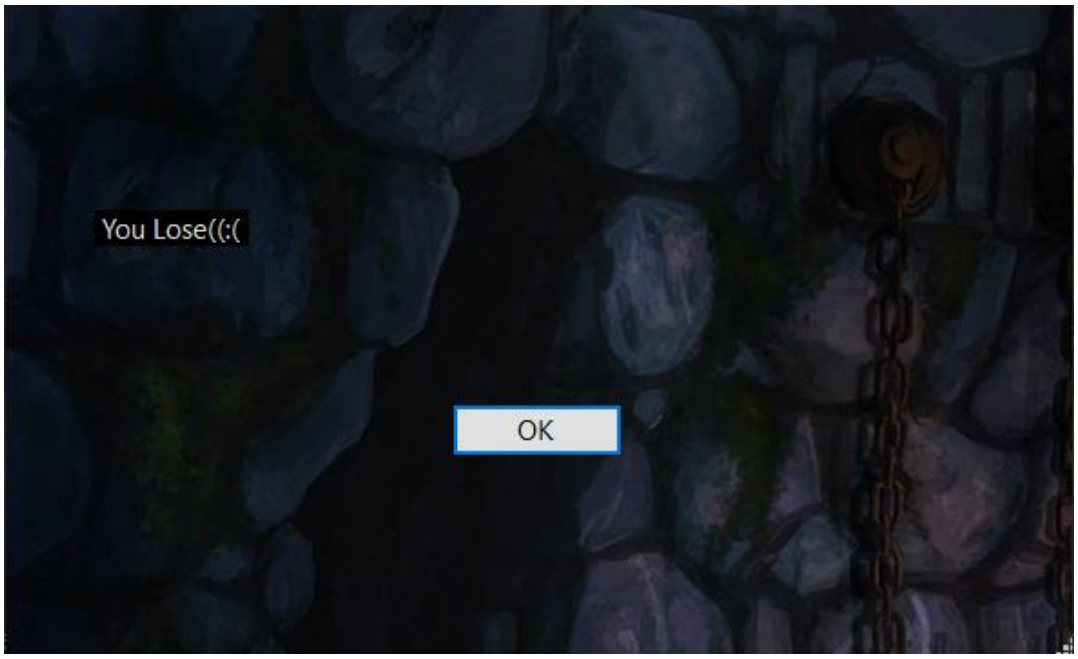

Рисунок 4.10 – Окно поражения

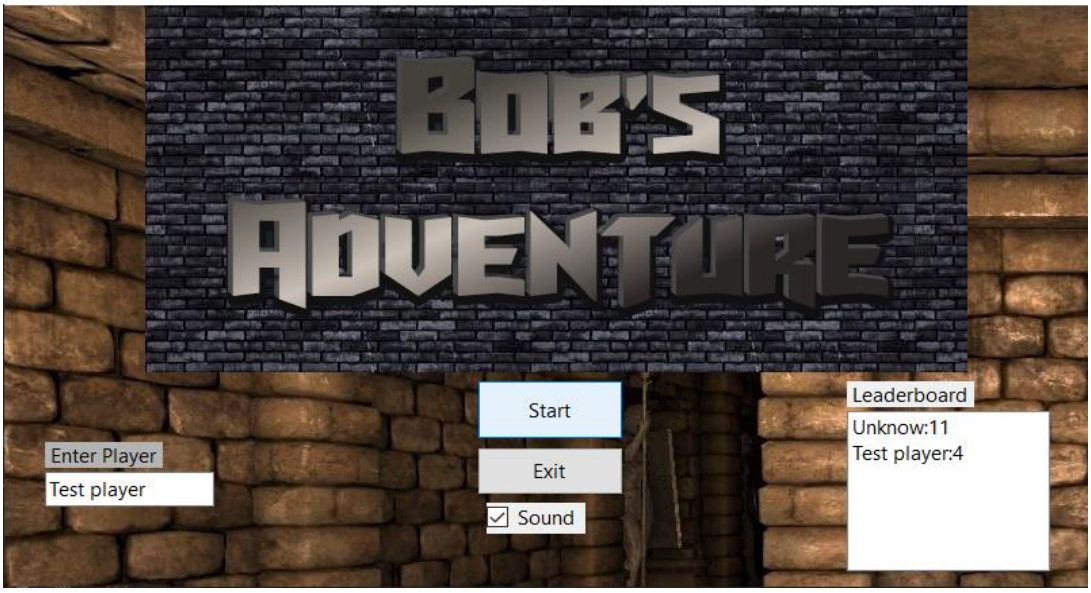

Рисунок 4.11 – Обновленное окно меню

Результаты тестирования показаны в таблице А.1 (приложение А).

В ходе тестирования программы не было замечено сбоев или аварийного завершения работы игрового приложения, что свидетельствует о его полной работоспособности.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была поставлена задача – на основе полученных в ходе изучения курса «Конструирование программного обеспечения» знаний создать работоспособное программное обеспечение, представляющее собой игровое приложение «Пошаговая стратегия». Для реализации поставленной задачи были использованы изученный в курсе язык программирования – C#, и изученная в курсе технология пользовательского интерфейса – Windows Forms. В ходе разработки игрового приложения проблем в использовании выбранных инструментов не было выявлено. Была написана собственная логика и сюжет игрового приложения, придуманы правила игры и игровые механики, которые удалось реализовать в полном объёме.

В игровом приложении существует возможность:

- генерации карты;
- движения персонажа;
- появления случайных событий;
- набора очков;
- появления противника;
- игры в камень-ножницы-бумага;

В ходе тестирования не было обнаружено сбоев в программе или аварийного завершения работы ПО. Все тесты были пройдены успешно, все функции и возможности приложения были реализованы. Игровая логика полностью соответствует описанию. Это говорит о полной функциональности и работоспособности игрового приложения, что свидетельствует об успешном выполнении главной задачи курсовой работы.

В итоге, можно сказать, что программа реализована в соответствии со всеми требованиями, протестирована надлежащим образом, работает стабильно и может использоваться для решения поставленной задачи. Поставленная задача выполнена в полной мере.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вирт Н. Алгоритмы + структуры данных = программы. – М.: Мир, 1985;
2. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона + CD. М.: ДМК Пресс, 2010. ISBN 978-5-94074-584-6, 0-13-022005-9;
3. Wikipedia: Программирование [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Программирование. Дата доступа: 24.11.2022;
4. Wikipedia: Тестирование [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Тестирование. Дата доступа: 26.11.2022;
5. Wikipedia: Пошаговая стратегия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Пошаговая_стратегия. Дата доступа: 28.11.2022;
6. Wikipedia: C# [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/C_Sharp. Дата доступа: 30.11.2022;
7. Learn Microsoft: Руководство по классическим приложениям (Windows Forms .NET) [Электронный ресурс]. – Режим доступа: learn.microsoft.com/dotnet/desktop/winforms/overview/?view=netdesktop-6.0. Дата доступа: 15.11.2022;
8. Systems Engineering Thinking Wiki: Проектирование [Электронный ресурс]. – Режим достпупа: http://sewiki.ru/Проектирование. Дата доступа: 28.11.2022;
9. Gamedev.ru: Распределение ресурсов для стратегии [Электронный ресурс]. – Режим доступа:_https://gamedev.ru/code/articles/resource_distribution. Дата доступа: 15.10.2022;
10. Wikipedia: Тестирование чёрного ящика или поведенческое тестирование [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Тестирование_по_стратегии_чёрного_ящика. Дата доступа: 30.11.2022;

ПРИЛОЖЕНИЕ А
(обязательное)
Таблица тестирования программы

Таблица А.1 – Способы проверок с указанием ожидаемых результатов испытаний

Тестовый вариант	Входные данные	Ожидаемый результат	Результат тестирования
Запуск клиентского модуля	Вход в игровое приложение	Открытие главного меню	Тест пройден успешно
Загрузка игрового процесса	Нажатие на кнопку «Start»	Открытие окна игры	Тест пройден успешно
Выход игрока из игрового приложения	Нажатие на кнопку «Exit»	Закрытие игрового приложения	Тест пройден успешно
Управление игроком	Нажатие на кнопку Step	Движение игрока	Тест пройден успешно
Набор очков	Нажатие на кнопку Step	Счетчик очков повышается на одну единицу	Тест пройден успешно
Появление событий	Нажатие на кнопку Step	Открытие случайных диалоговых окон	Тест пройден успешно
Появление окна битвы	Выпадение события «Rat want’s to kill you» и нажатие на клавишу OK	Открытие окна битвы с противником	Тест пройден успешно
Игра в камень-ножницы-бумага	Нажатие на кнопки Камень, Ножницы, Бумага.	Появление диалоговых окон и нанесение урона противнику/персонажу	Тест пройден успешно
Включение/ выключение звукового сопровождения	Нажатие на галочку «Sound»	Воспроизведение/ остановка фоновой музыки	Тест пройден успешно
Сохранение результата в таблицу лидеров	При смерти игрока его счет должен быть больше чем его предыдущий	Имя игрока и его счёт появляются в таблице лидеров в главном меню	Тест пройден успешно