СОДЕРЖАНИЕ

[Введение 4](#_Toc90435917)

[1 Анализ ИСХОДНЫХ ДАННЫХ и постановка задач 5](#_Toc90435918)

[2 Проектирование программы 6](#_Toc90435919)

[3 Реализация программы 7](#_Toc90435920)

[4 МЕТОДИКА И РЕЗУЛЬТАТЫ Тестирование 19](#_Toc90435921)

[Заключение 22](#_Toc90435922)

[Список использованных источников 23](#_Toc90435923)

[Приложение а (обязательное). Таблица тестирования программы 24](#_Toc90435924)

# Введение

Программирование – процесс и искусство создания компьютерных программ с помощью языков программирования [1].

Программирование сочетает в себе элементы искусства, науки, математики и инженерии [2].

В узком смысле слова, программирование рассматривается как кодирование – реализация одного или нескольких взаимосвязанных алгоритмов на некотором языке программирования. В более широком смысле, программирование – процесс создания программ, то есть разработка программного обеспечения. Большая часть работы программиста связана с написанием исходного кода на одном из языков программирования [2].

Различные языки программирования поддерживают различные стили программирования. Отчасти, искусство программирования состоит в том, чтобы выбрать один из языков, наиболее полно подходящий для решения имеющейся задачи. Разные языки требуют от программиста различного уровня внимания к деталям при реализации алгоритма, результатом чего часто бывает компромисс между простотой и производительностью.

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

Для решения поставленной задачи, разработка программы будет происходить в интигрированной среде разработки JetBrains Rider с использованием платформы .NET 6.0, языка програмирования C# и использованием библиотеки Windows Forms [11] для взаимодейтсвия с API Win32 [7][8].

API Win32 – это собственная платформа для Windows приложений. Этот API лучше всего подходит для классических приложений, которым требуется прямой доступ к системным функциям и оборудованию.[8] API Windows можно использовать во всех классических приложениях, и одни и те же функции обычно поддерживаются в 32-разрядной и 64-разрядной Windows.[9] Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, C++, так и на VB.Net, J# [10].

# 1 Анализ ИСХОДНЫХ ДАННЫХ и постановка задач

Flow– изначально флэш-игра, затем [видеоигра](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE%D0%B8%D0%B3%D1%80%D0%B0), разработанная [Дженовой Ченом](https://ru.wikipedia.org/wiki/%D0%94%D0%B6%D0%B5%D0%BD%D0%BE%D0%B2%D0%B0_%D0%A7%D0%B5%D0%BD) и Николасом Кларком и изданная компанией [Sony Computer Entertainment](https://ru.wikipedia.org/wiki/Sony) для [игровой приставки](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0) [Sony](https://ru.wikipedia.org/wiki/Sony) [PlayStation 3](https://ru.wikipedia.org/wiki/PlayStation_3) [3]. По жанру игра относится к симуляторам. Симулируется поведение [микроорганизма](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BE%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%BC).

Устройство игры и дизайн весьма просты и интуитивны. Изначально игрок управляет маленьким, белым, червеподобным организмом, который двигается по светлому двухмерному полю. Используя клавиатуру игрок управляет движением червяка.

Правила игры: В начале игры создается двухмерное поле на котором случайным образом генирируются наши опоненты и еда. Случайные зеленые квадраты процедурно генерируммые на игровом поле являются источником очков. Суть игры заключается в том, чтобы направлять червяка к другим микроорганизмам и поедать их, и тем самым развиваться. Также как и игрок, другие микроогранизмы могут развиваться и представлять опасность для игрокаю. Любой микроорганизм может расти до пределов экрана. Если микроорганизм зайдет за края мира, то он появится с противоположенной стороны. Если один организм сьест хвост другого, то хвост обрубится пополам в месте укуса.

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

* Перемещение микроорганизмов;
* Подсчет очков;
* Процедурное создание новых микроорганизмов;
* Исскусвенный интелект микроорганизмов;
* Обработка столкновений игровых обьектов.

# 2 Проектирование

Проектирование – процесс создания проекта программного обеспечения, а также дисциплина, изучающая методы проектирования. Проектирование ПО является частным случаем проектирования продуктов и процессов. 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.

При запуске программы на экране будет появляться игровое поле 60x36. Начинают появляться микроорганизмы. Игровое поле можно увидеть на .

Для выполнения поставленной задачи должны быть спроектированы следующие основные модули:

* Flow – главный модуль программы который отвечает за работу с другими модулями и взаимодействие с ними, обновление окна игры и обработку столкновений;
* GameObject – абстрактный модуль являющийся объектом который может находится в игре;
* Square – примитив квадрата;
* SquareShadow – модуль отвечающий за рисование тени под примитивами квадрата;
* RandomColor – статический модуль помогающий создавать случайные цвета в цветовой модели RGB;
* Snake – модуль объекта микроорганизма;
* PlayerSnake – микроорганизм которым управляет игрок;
* ComputerSnake – микроорганизм которым управляет компьютер;
* Tail – модуль хвоста микроорганизмов;
* Bio – модуль объекта еды;
* MainMenu – модуль окна главного меню.

# 3 Реализация программы

**3.1 Модуль Flow**

У модуля Flow есть 10 методов.

Метод Flow\_Load(object sender, EventArgs e) создает обьекты на старте игры в случайных позициях, представлен в листинге 3.1.

Листинг 3.1 – Метод Flow\_Load(object sender, EventArgs e)

private void Flow\_Load(object sender, EventArgs e)

{

spawnBios.Interval = FlowRandom.Next(1000, 10000);

AddNewBios();

CreateGameObject(new PlayerSnake(3, 3, 16, 16, this));

CreateGameObject(new ComputerSnake(4, 40, 16, 16, this));

CreateGameObject(new ComputerSnake(32, 40, 16, 16, this));

CreateGameObject(new ComputerSnake(24, 40, 16, 16, this));

gameTimer.Start();

}

Второй Метод AddNewBios(int min = 0, int max = 0) представлен в листинге 3.2 и связан с модулем Bio. Этот метод создает случайное количество обьектов еды на поле в случайном месте.

Листинг 3.2 – Метод AddNewBios(int min = 0, int max = 0)

public void AddNewBios(int min = 0, int max = 20)

{

for (int i = 0; i < FlowRandom.Next(min, max); i++)

CreateGameObject(

new Bio(FlowRandom.Next(0, 60),

FlowRandom.Next(0, 36), 16, 16,

this));

}

Метод AddNewComp(int min = 0, int max = 0) представлен в листинге 3.3 и связан с модулем ComputerSnake. Этот метод создает случайное количество микроорганизмов управляемых компьютером еды на поле в случайном месте.

Листинг 3.3 – Метод AddNewBios(int min = 0, int max = 0)

public void AddCompBios(int min = 0, int max = 20){

for (int i = 0; i < FlowRandom.Next(min, max); i++)

CreateGameObject(

new ComputerSnake(FlowRandom.Next(0, 60),

FlowRandom.Next(0, 36), 16, 16, this)); }

Метод CreateGameObject(GameObject gameObject) добавляет в список игровых обьектов новый обьект. Представлен в листинге 3.4.

Листинг 3.4 – Метод CreateGameObject(GameObject gameObject)

public GameObject CreateGameObject(GameObject gameObject){

Продолжение листинга 3.3

gameObjects.Add(gameObject);

return gameObject;

}

Метод KeyIsDown(object sender, KeyEventArgs e) получает с клавиатуры нажатые клавиши и сохраняет в поле соотвествующей клавиши положительное значение. Представлен в листинге 3.5.

Листинг 3.5 – Метод CreateGameObject(GameObject gameObject)

private void KeyIsDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Left && directions != "right")

goLeft = true;

if (e.KeyCode == Keys.Right && directions != "left")

goRight = true;

if (e.KeyCode == Keys.Up && directions != "down")

goUp = true;

if (e.KeyCode == Keys.Down && directions != "up")

goDown = true;

if (e.KeyCode == Keys.K)

goCreate = true;

}

Метод DrawCanvas(object sender, PaintEventArgs) каждое обновление кадра вызывает методы Draw у обьектов в списке gameObjects. Представлен в листинге 3.6.

Листинг 3.6 – Метод DrawCanvas(object sender, PaintEventArgs e)

private void DrawCanvas(object sender, PaintEventArgs e)

{

var graphics = e.Graphics;

foreach(var obj in gameObjects)

obj.Draw(graphics);

}

Метод gameTimer\_Tick(object sender, EventArgs e) каждое обновление кадра вызывает методы Update у обьектов в списке gameObjects, проверяет на на наличие столкновений и удаленяет обьекты которые уже не используются. Представлен в листинге 3.7.

Листинг 3.7 – Метод gameTimer\_Tick(object sender, EventArgs e)

private void gameTimer\_Tick(object sender, EventArgs e)

{

CheckForCollisions();

DeleteDiedObjects();

}

Метод CheckForCollisions() проверяет на на наличие столкновений на всём поле. Если столкновения найдены, то метод вызывает метод OnHit(GameObject hit) класса GameObject. Представлен в листинге 3.8.

Листинг 3.8 – Метод CheckForCollisions ()

private void CheckForCollisions()

{

for(var i = 0; i < gameObjects.Count; i++)

{

for(var k = 0; k < gameObjects.Count; k++)

{

if (i == k) continue;

if (gameObjects[i].X == gameObjects[k].X && gameObjects[i].Y == gameObjects[k].Y)

{

if (!\_collidedDetectedQuene.Contains(gameObjects[k]) || !\_collidedDetectedQuene.Contains(gameObjects[i]))

{

gameObjects[i].OnHit(gameObjects[k]);

gameObjects[k].OnHit(gameObjects[i]);

\_collidedDetectedQuene.Add(gameObjects[i]);

\_collidedDetectedQuene.Add(gameObjects[k]);

}

}

}

}

}

Метод GetRandomTarget<T>() добваляет в отфильтрованный список объекты из списка gameObjects которые являются классами T и возвращает случайный объект из отфильтрованного списка. Если обьектов класса T нет, то метод создает на поле новых микроэлементов и повторяет поиск. Представлен в листинге 3.9.

Листинг 3.9 – Метод GetRandomTarget<T> ()

public T GetRandomTarget<T>() where T : GameObject

{

var filter = gameObjects.Select(

x => {

return new {

succses = x is T, val = x };

})

.Where(a => a.succses)

.Select(v => (T)v.val).ToList();

AddNewBios();

return GetRandomTarget<T>();

}

var randomIndex = FlowRandom.Next(0, filter.Count - 1); var randomGameObject = filter[randomIndex]; if (randomGameObject is Snake)

{

var snake = randomGameObject as Snake; randomGameObject = snake.GetRandomTailTile() as T;

}

Продолжение листинга 3.9

return randomGameObject;

}

Метод GetGameObjects<T>() возвращает в списке все обьекты класса T которые находятся в списке gameObjects. Представлен в листинге 3.10.

Листинг 3.10 – Метод GetGameObjects<T> ()

public List<T> GetGameObjects<T>() where T : GameObject

{

var filter = gameObjects

.Select(

x => { return new { succses = x is T, val = x }; })

.Where(a => a.succses)

.Select(v => (T)v.val).ToList();

return filter;

}

**3.2 Модуль GameObject**

Данный модуль является абстрактным классом от которого должны наследоваться все классы которые могут находиться в игре. У класса GameObject есть поля координат, размеры объекта, ссылка на окно которому объект принадлежит и объект тени из модуля SquareShadow. При вызове конструктора данного класса все поля заполняются данными из аргументов. В классе GameObject реализованы следующие функции:

* void Draw() – метод отрисовки объекта, представлен в листинге 3.11;
* void Update() – метод обновления объекта, представлен в листинге 3.12;
* void OnHit(GameObject gameObject) – метод который вызывается при столкновении c другим объектом, представлен в листинге 3.13.

Листинг 3.11 – Метод Draw(Graphics graphics)

public virtual void Draw(Graphics graphics)

{

\_shadow.Draw(graphics);

}

Листинг 3.12 – Метод Update()

public virtual void Update()

{

\_shadow?.Update();

}

Листинг 3.13 – Метод OnHit (GameObject gameObject)

public virtual void OnHit(GameObject gameObject)

{

Продолжение листинга 3.13

Console.WriteLine("Collision detected");

}

**3.3 Модуль Square**

Данный модуль является объектом квадрата. Его задача быть удобным модулем для создания, рисования и обновления квадратов на поле игры. В этом модуле реализован класс Square который наследуется от класса GameObject. В классе Square содержится метод Draw(), в котором находится логика телепортация объекта на противоположную сторону поля при захождении объекта на границу поля. Представлен в листинге 3.14.

Листинг 3.14 – Метод Draw(Graphics graphics).

public override void Draw(Graphics graphics)

{

base.Draw(graphics);

if (X < 0)

X = 60;

if (X > 60)

X = 0;

if (Y < 1)

Y = 36;

if (Y > 36)

Y = 1;

graphics.FillRectangle(

GameObjectSolidBrush,

new Rectangle(X \* Width, Y \* Height, Width, Height));

}

**3.4 Модуль SquareShadow**

Данный модуль является объектом тени для объектов класса GameObject. Его задача рисоваться под обьектом и следить за его положением. В этом модуле реализован класс SquareShadow который наследуется от класса Square. В классе SquareShadow содержится метод Draw(), в котором находится логика рисования объекта под другим объектом класса GameObject. Представлен в листинге 3.15.

Листинг 3.15 – Метод Draw(Graphics graphics).

public override void Draw(Graphics graphics)

{

graphics.FillRectangle(

Продолжение листинга 3.15

GameObjectSolidBrush,

new Rectangle(

(parentGameObject.X \* Width) + \_shadowOffset.X , (parentGameObject.Y \* Height) + \_shadowOffset.Y,

Width,

Height));

}

**3.5 Модуль Snake**

Модуль содержит в себе класс Snake который наследуется от Square. В классе Snake описывается логика поведения микроорганизма у которого есть хвост, он может двигаться в влево, вправо, вверх и вниз. Так же класс Snake следит за своими хвостами и обновляет их вслед за собой, что создает иллюзию того, что микроорганизм ползет по полю игры. Еще одна из ответственностей класса это поедания объектов еды, объектов хвостов других микроорганизмов и самих микроорганизмов.

В классе Snake реализованы следующие методы:

* void MoveAllTails() – метод двигающий все хвосты которые принадлежат Snake, представлен в листинге 3.16;
* void OnHit(GameObject gameObject) – метод отвечающий за поедание других объектов во время столкновения с ними, представлен в листинге 3.17;
* GameObject GetRandomTailTile() – метод который возвращает случайный объект класса Tail из списка \_snake в котором находится хвост микроорганизма, представлен в листинге 3.18;
* int DeleteTail(Tail tail) – метод который удаляет элементы из хвоста до обьекта tail из аргумента метода, представлен в листинге 3.19;

Листинг 3.16 – Метод OnHit(GameObject gameObject)

public override void OnHit(GameObject gameObject)

{

if(gameObject is Bio)

{

var tail = new Tail(

X,

Y,

Width,

Height,

BelongForm,\_tailColor,

this);

\_snake.Add((Tail)BelongForm.CreateGameObject(tail));

ToDoOnEatBio();

if(BelongForm.FlowRandom.Next(0,5) == 2)

Продолжение листинга 3.16

BelongForm.AddNewBios(1,2);

}

if(gameObject is Tail)

{

var tail = gameObject as Tail;

if(tail.ParentSnake != this)

{

var hittedSnake = tail.ParentSnake;

if(hittedSnake.\_snake.Count > 4)

{

var tempLength = hittedSnake.\_snake.Count / 2;

for (var i = hittedSnake.\_snake.Count - 1; i > tempLength; i--)

{

var snakeTail = hittedSnake.\_snake[i];

hittedSnake.\_snake.Remove(snakeTail);

snakeTail.GoingToDie = true;

}

}

else

{

var snake = tail.ParentSnake;

if (snake.\_snake.Count > \_snake.Count)

{

foreach (var tailr in \_snake)

tailr.GoingToDie = true;

GoingToDie = true;

}

if (snake.\_snake.Count < \_snake.Count)

{

foreach (var tailr in snake.\_snake)

tailr.GoingToDie = true;

snake.GoingToDie = true;

}

}

}

}

if(gameObject is Snake)

{

var snake = gameObject as Snake;

if(snake.\_snake.Count > \_snake.Count)

{

foreach(var tail in \_snake)

tail.GoingToDie = true;

GoingToDie = true;

}

if (snake.\_snake.Count < \_snake.Count)

{

foreach (var tail in snake.\_snake)

Продолжение листинга 3.16

tail.GoingToDie = true;

snake.GoingToDie = true;

}

}

}

Листинг 3.17 – Метод MoveAllTails()

private void MoveAllTails()

{

if(\_snake.Count > 0)

{

\_snake[0].X = X;

\_snake[0].Y = Y;

}

for(var i = \_snake.Count - 1; i >= 1; i--)

{

\_snake[i].X = \_snake[i - 1].X;

\_snake[i].Y = \_snake[i - 1].Y;

}

}

Листинг 3.18 – Метод GetRandomTailTile()

public GameObject GetRandomTailTile()

{

if(\_snake.Count > 0)

{

var randomIndex = BelongForm.FlowRandom.Next(0, \_snake.Count - 1);

return \_snake[randomIndex];

}

return this;

}

Листинг 3.19 – Метод DeleteTail(Tail tile)

public int DeleteTails(Tail tail)

{

var index = \_snake.IndexOf(tail);

if(index > -1)

\_snake = \_snake.GetRange(0, index);

return index;

}

**3.6 Модуль PlayerSnake**

Данный модуль отвечает за механизм управления игроком микроорганизма. Каждый кадр микроорганизм двигается на одну клетку в сторону последней нажатой клавиши, которая устанавливается в методе KeyIsDown() в классе Flow. В PlayerSnake реализован класс PlayerSnake и содержит следующие функции:

* void Update() – метод обновления объекта, в котором устанавливается движение персонажа в зависимости от нажатой клавиши, представлен в листинге 3.20;
* void ToDoOnEatBio() – метод который вызывается при поедании еды, представлен в листинге 3.21;

Листинг 3.20 –Метод Update()

public override void Update()

{

base.Update();

switch (BelongForm.directions)

{

case "left":

X--;

break;

case "right":

X++;

break;

case "up":

Y--;

break;

case "down":

Y++;

break;

}

}

Листинг 3.21 –Метод Update()

public override void ToDoOnEatBio()

{

base.ToDoOnEatBio();

BelongForm.AddScore();

}

**3.7 Модуль ComputerSnake**

Данный модуль отвечает за механизм управления компьютером микроорганизма. Каждый кадр микроорганизм двигается на одну клетку в сторону случайно выбранной цели. В ComputerSnake реализован класс ComputerSnake и содержит следующие функции:

* void Update() – метод обновления объекта, представлен в листинге 3.22;
* void MoveTargetByX() – метод который двигает микроорганизм в сторону еды по оси координат X, представлен в листинге в 3.23;
* void MoveTargetByY() – метод который двигает микроорганизм в сторону еды по оси координат Y, представлен в листинге в 3.24;

Листинг 3.22 – метод Update()

public override void Update()

{

base.Update();

if(Target.X == X && Target.Y == Y || BelongForm.FlowRandom.Next(0,50) == 30)

Target = BelongForm.GetRandomTarget<GameObject>();

var randomChoice = BelongForm.FlowRandom.Next(1, 3);

if (randomChoice == 1)

MoveToTargetByX();

if(randomChoice == 2)

MoveToTargetByY();

}

Листинг 3.23 – метод MoveToTargetByX()

private void MoveToTargetByX(){

if (Target.X > X)

X++;

else

X--;

}

Листинг 3.24 – метод MoveToTargetByY()

private void MoveToTargetByY()

{

if (Target.Y > Y)

Y++;

else

Y--;

}

**3.8 Модуль Bio**

Данный модуль отвечает класс за Bio, который является объектом еды. Каждый кадр микроорганизм двигается на одну клетку в случайную сторону. Класс Bio содержит следующие функции:

* void Update() – метод обновления объекта, в котором еда в случайном случае двигается на одну клетку в случайную сторону, представлен в листинге 3.25;
* void OnHit(GameObject gameObject) – метод уничтожения объекта если было столкновение с классом Snake, представленв листинге, представлен в листинге 3.26;

Листинг 3.25 –метод Update()

public override void Update()

{

base.Update();

if(BelongForm.FlowRandom.Next(0,20) == 9)

{

X += BelongForm.FlowRandom.Next(-1, 1);

Y += BelongForm.FlowRandom.Next(-1, 1);

}

if (GoingToDie)

{

GameObjectColor = Color.Red;

GameObjectSolidBrush = new SolidBrush(GameObjectColor);

}

}

Листинг 3.26 –метод OnHit(GameObject gameObject)

public override void OnHit(GameObject gameObject) {

if (gameObject is Snake)

GoingToDie = true;

}

**3.9 Модуль MainMenu**

Данный модуль отвечает класс за MainMenu, который наследуется от класса Form и является главным меню данного игрового приложения. Задача этого модуля создавать окно при запуске приложение, дать пользователю выходить и запускать игру и просматривать таблицу рекордов. Класс MainMenu содержит следующие функции:

* void button1\_click() – метод который вызывается при нажатии на кнопку «Play». Он запускает игровое приложение. Представлен в листинге 3.27;
* void AddToScoreList(int score) – метод который добавляет или обновляет новый рекорд в таблице лидеров. Представлен в листинге 3.28;

Листинг 3.27 –метод button1\_Click()

private void button1\_Click(object sender, EventArgs e)

{

PressSound.Play();

new Flow(this);

Hide();

}

Листинг 3.28 –метод AddToScoreList (int score)

public void AddToScoreList(int score)

{

listView1.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))

listView1.Items.Add($"{s.Key}:{s.Value}\n");

}

# 4 Тестирование программы

Тестирование программы – процесс проверки программы на соответствие ожидаемого результата и полученного, путём подбора тестов выбранных определенным образом [4][5].

Существует несколько видов классификации и методов тестирования. В данном курсовом проекте для тестирования программы использовался метод «Функционального тестирования» типа черного ящика.

Функциональное тестирование типа черного ящика – метод тестирования функционального поведения программы без доступа к исходному коду программы и проверки реализуемости функциональных требований. Тестирование черного ящика – углубление во внутренне устройство системы, за пределы ее внешних интерфейсов

**Функциональные требования:**

– функциональная пригодность;

– Точность;

– Способность к взаимодействию;

– Соответствие стандартам и правилам;

– Защищённость.

Для выполнения тестирования черного ящика не требуется большое количество специальных знаний и при использовании автоматизации тестирования на этом уровне, поддержка тестовых сценариев может оказаться достаточно полезной, если программа часто изменяется.

При запуске программы открывается игровое меню. Пользователь видит три опции на белом фоне ().



Рисунок 4.1 –Меню игры

При нажатии на опцию «Exit» приложение закрывается. При нажатии на опцию «Play» открывается новое окно с самой игрой. При нажатии на клавиши стрелок игрок начинает двигаться в соответствующие стороны ().

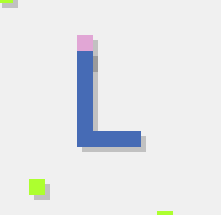


Рисунок 4.2 – Управление игроком

В процессе игры пользователь набирает очки путем поедания еды (рисунок 4.3).

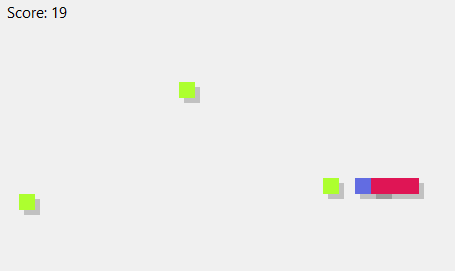


Рисунок 4.3 – Набирание очков.

Если игрок столкнется с другим микроорганизмом и при этом будет меньше его, то игра закончится и откроется следующее окно (рисунок 4.4).



Рисунок 4.4 – Окно проигрыша.

После смерти игра проверяет, побил ли игрок свой рекорд и обновляет его при необходимости (рисунок 4.5).

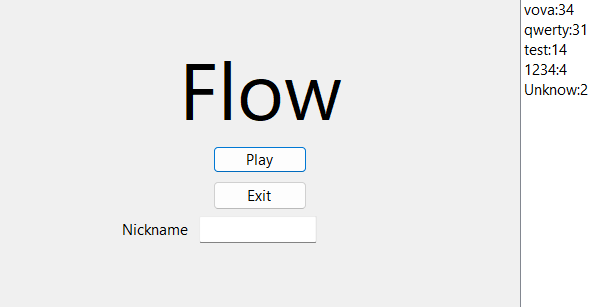


Рисунок 4.5 – Таблица рекордов.

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

# Заключение

Идея программирования заключается в том, чтобы составить алгоритм и перевести его на язык программирования. Фактически, язык программирования, это набор правил описания команд и заранее созданных функций. Каждый язык программирования ограничивает разработчика строго предопределенным набором ключевых слов и команд, которые могут быть использованы при разработке.

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 «Flow».

В игровом приложении существует возможность:

* паузы игры;
* передвижения фигур;
* счета и отображения очков;
* перезагрузка игры после ее окончания;
* проигрывание звуков и музыки;
* увеличивать размер окна приложения;
* обработки столкновений с объектами.

В ходе тестирования не было обнаружено сбоев в программе или аварийного завершения работы ПО, что говорит о полной функциональности игрового приложения и выполнении главной задачи курсовой работы.

В итоге, можно сказать, что программа реализована в соответствии всем требованиям, 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 и структуры данных. Новая версия для Оберона + CD. М.: ДМК Пресс, 2010. [ISBN 978-5-94074-584-6](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785940745846), 0-13-022005-9;
2. Wikipedia : Программирование [Электронный ресурс].  – Режим доступа: <https://ru.wikipedia.org/wiki/Программирование>. Дата доступа: 30.09.2022;
3. Wikipedia: Flow(video game) [Электронный ресурс].  – Режим доступа: https://en.wikipedia.org/wiki/Flow(videogame). Дата доступа: 30.07.2022;
4. Wikipedia: Тестирование программного обеспечения [Электронный ресурс]. – ­Режим доступа: https://en.wikipedia.org/wiki/Software\_testing. Дата доступа: 10.10.2022;
5. Wikipedia: Функциональное тестирование [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Functional\_testing. Дата доступа: 10.10.2022;
6. Microsoft: Windows Forms documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/. Дата доступа: 11.10.2022;
7. Microsoft: .Net documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/. Дата доступа: 11.10.2022;
8. Microsoft: Programming reference for the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/ Дата доступа: 11.10.2022;
9. Wikipedia: Windows API [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Windows\_API. Дата доступа: 13.11.2022;
10. Microsoft: C# documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/csharp/. Дата доступа: 11.10.2022;
11. Wikipedia: C Sharp (programming language) [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/C\_Sharp\_(programming\_language). Дата доступа: 10.11.2022.

# 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 опцию главной страницы «Play» | Открытие окна игры | Тест пройден успешно |
| Выход из игрового приложения | Нажатие на опцию главной страницы «Exit» | Закрытие игрового приложения | Тест пройден успешно |
| Управление игроком | Нажатие на направляющие клавиши | Изменение направления игрока | Тест пройден успешно |
| Набор очков | Игрок должен соприкоснуться с объектом еды | Счетчик очков повышается на одну единицу | Тест пройден успешно |
| Удаление объектов с игрового поля | Игрок должен соприкоснуться с объектом еды | Объект еды пропадает с игрового поля | Тест пройден успешно |
| Взаимодействие с другими микроорганизмами | Игрок должен соприкоснуться с объектом меньшего микроорганизма | У микроорганизма отнимается его часть до места соприкосновения | Тест пройден успешно |
| Сохранение результата в таблицу лидеров | При смерти игрока его счет должен быть больше чем его предыдущий | Имя игрока и его появляются в таблице лидеров в главном меню | Тест пройден успешно |
| Проигрыш | Игрок должен соприкоснуться с объектом большего микроорганизма | Открытия окна проигрыша | Тест пройден успешно |