Министерство транспорта Российской Федерации

Федеральное агентство железнодорожного транспорта

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Дальневосточный государственный университет путей сообщения»

Кафедра: «Информационные технологии и системы»

КУРСОВОЙ ПРОЕКТ

«Разработка программного приложения на языке C#»

(по дисциплине «Технологии и методы программирования»)

Выполнил: Журин Д.К

Группа: БО222ПИН

Проверил: Потапов И.И.

Хабаровск

2022

Аннотация

Данная работа посвящена созданию двухмерной игры с основной механикой аналогичной механике игры «Тетрис» на языке C# с использованием библиотеки OpenTK.

«Тетрис» – видеоигра, разработанная советским программистом Алексеем Пажитновым в 1984 году.

В оригинальной игре пользователю предлагается собрать в ряд несколько разновидных фигур. Сложность игры заключается в различной геометрии фигур, что затрудняет пользователю сбор целого ряда. В реализованной игре прямоугольной поле тетриса преобразовано в секторальный вид, соответственно, игроку следует собрать полный круг, вместо ряда.

Оглавление

[1. Введение 4](#_Toc122392982)

[2. Описание программы 5](#_Toc122392983)

[2.1 Класс «Block» 5](#_Toc122392984)

[2.2 Класс «BlockQueue» 5](#_Toc122392985)

[2.3 Класс «VisualFigure» 6](#_Toc122392986)

[2.4 Класс «Button» 6](#_Toc122392987)

[2.5 Класс «Cell» 6](#_Toc122392988)

[2.6 Класс «CircleCells» 6](#_Toc122392989)

[2.7 Класс «ContentPipe» 7](#_Toc122392990)

[2.8 Класс «Game» 7](#_Toc122392991)

[2.9 Класс «GameStatus» 7](#_Toc122392992)

[2.10 Класс «Position» 7](#_Toc122392993)

[2.11 Класс «TextRenderer» 7](#_Toc122392994)

[2.12 Класс «Program» 7](#_Toc122392995)

[3. Исходный код 8](#_Toc122392996)

[4. Руководство пользователя 9](#_Toc122392997)

[Заключение 11](#_Toc122392998)

[Список использованной литературы 12](#_Toc122392999)

# Введение

Одним из популярных направлений компьютерной деятельности является создание видеоигр., которые позволяют провести свободное время и отвлечь от повседневной рутины.

Целями данной работы является знакомство с библиотекой «OpenTK» и разработка игры по мотивам «Тетрис».

OpenTK (Open Toolkit) — это набор быстрых, переносимых, низкоуровневых привязок C# для OpenGL, OpenGL ES, OpenAL и OpenCL. Он работает на всех основных платформах и используется в сотнях приложений, игр и научно-исследовательских программ. OpenTK предоставляет несколько полезных библиотек, включая пакет математики/линейной алгебры, оконную систему и обработку ввода.

Задачами данной работы являются:

* изучение базового материала для работы с библиотекой OpenTK;
* планирование и создание концепции игры и всех её составляющих.

# Описание программы

Данная программа реализована на платформе .Net 6.0 с использованием библиотеки «OpenTK» на языке C#.

Приложение состоит из 11 классов, необходимых для описание поведениях всех объектов: «Block», «BlockQueue», «Button», «CircleCells», «Cell», «ContentPipe», «Game», «GameStatus», «Position», «Program», «TextRenderer», «VisualFigure».

## Класс «Block»

Данный класс предназначен для создания блока — фигуры. Класс в конструкторе принимает уникальный идентификатор блока (необходим для того, чтобы понять какая фигура создаётся), массив из координат составляющих «кубики» фигуры, стартовой позиции.

Метод «Move» позволяет передвигать блок изменяя его Offset. Метод Reset устанавливает значение по умолчанию для блока. Метод TilePosititons необходим для передачи в другие функции координат фигуры.

Методы RotateCW и RotateCCW необходимы для поворота блока.

## Класс «BlockQueue»

Данный класс необходим для создания очереди из падающих блоков. Класс имеет в себе закрытое поле массива элементов Block, в котором определены все фигуры, а так же методов RandomBlock, создающего случайный блок, и метода GetAndUpdate, который передаёт новый блок и создают сразу же следующий.

## Класс «VisualFigure»

Этот абстрактный класс необходим для заданий шаблонов для потенциальных фигур, с которыми пользователь может взаимодействовать мышью. Имеет делегат MouseDelegate и экземпляр этого делегата OnMouseDown, которые используются в наследуемых классах для реализации различных действий. А также 2 абстрактных метода Draw и IsPointinFigure.

## Класс «Button»

Данный класс наследуется от абстрактного класса «VisualFigure» и реализует его методы в контексте кнопки на экране. Для этого имеется соответствующих конструктор, принимающий координаты кнопки и её цвет.

## Класс «Cell»

Данный класс необходим для создания одной клетки-блока в общей таблице. В конструкторе принимает порядковый номер в таблице и необходимые данные для просчёта координат. Метод Draw отрисовывает блок.

## Класс «CircleCells»

Данных класс создаёт всю таблицу из экземпляров Cell. В лкассе присутствуют основные методы для получения информации о состоянии таблицы и её строк. Также имеются методы по очистке собранной строчки и сдвига других строк вниз.

## Класс «ContentPipe»

Данный класс загружает текстуру в память графического процесса и предоставляет уникальный id текстуры.

## Класс «Game»

Этот класс является основным классом игры, который реализует окно OpenGL. В этом классе созданы все методы для рендера кадра и изменения логики игры. В нём создаются все классы и используются для работы игры.

## Класс «GameStatus»

Данный класс реализует всю логику игры. Он собирает в себе классы Block, BlockQueue, CircleCells. Класс реализует методы для манипулирования падающим блоком, методы для установки блока и подсчёта числа очков. В этом классе создан метод, проверяющий положение блока на поле.

## Класс «Position»

Данных класс служит для хранение координат х и у.

## Класс «TextRenderer»

Данный класс хранит методы позволяющие отобразить текст на экране

## Класс «Program»

Данный класс содержит стартовый метод Main, запускающий игру.

# Исходный код

using OpenTK.Graphics.OpenGL;

using System.Text;

namespace Kursach

{

internal class TextRenderer : IDisposable

{

int choosenColumn, choosenRow, rows, columns, textureId;

float frameWidth, frameHeight, xx = 0.1f, yy = 0.1f;

List<int> vaoVboindex = new List<int>() { };

public TextRenderer(int rows, int columns, int textureId, float canvaWidth, float canvaHeight){

this.rows = rows;

this.columns = columns;

frameHeight = 1.0f / rows;

frameWidth = 1.0f / columns;

this.textureId = textureId;

xx \*= canvaWidth / 4;

yy \*= canvaHeight / 4;

}

public void PrepareText(float x, float y, float dx, string text, float scale = 1) {

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

Encoding encoding = Encoding.GetEncoding("windows-1251");

byte[] asciiCodes = encoding.GetBytes(text);

for (int i = 0; i < text.Length; i++){

LetterRender(x + i \* dx, y, asciiCodes[i], out int a, out int b, out int vao, scale);

vaoVboindex.Add(vao); vaoVboindex.Add(b); vaoVboindex.Add(a);

}

}

private void LetterRender(float x, float y, byte target, out int a, out int b, out int vao, float scale = 1){

choosenColumn = target % rows;

choosenRow = target / columns;

float[] texCords = { 0.0f + frameWidth \* (choosenColumn) , 0.0f + frameHeight \* (choosenRow + 1),

0.0f + frameWidth \* (choosenColumn),0.0f + frameHeight \* (choosenRow),

0.0f + frameWidth \* (choosenColumn + 1), 0.0f + frameHeight \* (choosenRow),

0.0f + frameWidth \* (choosenColumn + 1),0.0f + frameHeight \* (choosenRow + 1) };

float[] letCords = { x, y, x, y + (yy \* scale), x + (xx \* scale), y + (yy \* scale), x + (xx \* scale), y };

a = GL.GenBuffer();

GL.BindBuffer(BufferTarget.ArrayBuffer, a);

GL.BufferData(BufferTarget.ArrayBuffer, letCords.Length\*sizeof(float),letCords,BufferUsageHint.StaticDraw);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

b = GL.GenBuffer();

GL.BindBuffer(BufferTarget.ArrayBuffer,b);

GL.BufferData(BufferTarget.ArrayBuffer, texCords.Length \* sizeof(float), texCords, BufferUsageHint.StaticDraw);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

vao = GL.GenVertexArray();

GL.BindVertexArray(vao);

GL.EnableClientState(ArrayCap.VertexArray);

GL.EnableClientState(ArrayCap.TextureCoordArray);

GL.BindBuffer(BufferTarget.ArrayBuffer, b);

GL.TexCoordPointer(2, TexCoordPointerType.Float,0, 0);

GL.BindBuffer(BufferTarget.ArrayBuffer, a);

GL.VertexPointer(2, VertexPointerType.Float, 0, 0);

GL.BindVertexArray(0);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

GL.DisableClientState(ArrayCap.VertexArray);

GL.DisableClientState(ArrayCap.TextureCoordArray);

}

public void RenderText(){

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Blend);

GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);

for (int i = 0; i < vaoVboindex.Count; i += 3)

{

GL.BindVertexArray(vaoVboindex[i]);

GL.DrawArrays(PrimitiveType.Quads, 0, 4);

GL.BindVertexArray(0);

}

GL.Disable(EnableCap.Texture2D);

GL.Disable(EnableCap.Blend);

}

public void Dispose(){

GL.DeleteTexture(textureId);

for (int i = 0; i < vaoVboindex.Count; i += 3)

{

GL.BindVertexArray(0);

GL.DeleteVertexArray(vaoVboindex[i]);

GL.DeleteBuffer(vaoVboindex[i + 1]);

GL.DeleteBuffer(vaoVboindex[i + 2]);

}

}

public void Update(float x, float y, float dx, string text, float scale = 1){

for (int i = 0; i < vaoVboindex.Count; i+=3)

{

GL.BindVertexArray(0);

GL.DeleteVertexArray(vaoVboindex[i]);

GL.DeleteBuffer(vaoVboindex[i + 1]);

GL.DeleteBuffer(vaoVboindex[i + 2]);

}

vaoVboindex.Clear();

PrepareText(x, y, dx, text, scale = 1);

}

}

}

using OpenTK.Windowing.Common;

using OpenTK.Windowing.Desktop;

namespace Kursach{

internal class Program {

static void Main(string[] args){

StartGame();

}

static void StartGame(){

GameWindowSettings gSettings = GameWindowSettings.Default;

NativeWindowSettings nSettings = new NativeWindowSettings(){

Title = "Game",

Size = (1280, 1024),

Flags = ContextFlags.Default,

Profile = ContextProfile.Compatability

};

Game game = new Game(gSettings, nSettings);

game.Run();

}

}

}

namespace Kursach{

internal class Position{

public Position(int row, int column){

Row = row;

Column = column;

}

public int Row { get; set; }

public int Column { get; set; }

}

}

using OpenTK.Mathematics;

namespace Kursach{

internal class GameStatus{

private Block currentBlock;

private int width, height;

public Block CurrentBlock{

get { return currentBlock; }

set{currentBlock = value;currentBlock.Reset();

}

}

public CircleCells CircleCell{ get; }

public BlockQueue BQ { get; }

public bool GameOver { get; set; }

public int Scores { get; set; }

public GameStatus(int height, int width, Vector2 centerPoint,int r, int dr ){

CircleCell = new CircleCells(height, width, centerPoint, r, dr);

this.width = width;

this.height = height;

BQ = new BlockQueue();

CurrentBlock = BQ.GetAndUpdate();

Block.Width = width;

}

private bool BlockFits(){

foreach (Position p in CurrentBlock.TilePositions())

{

if (!CircleCell.Empty(p.Row, p.Column))

return false;

if (CurrentBlock.Offset.Column + p.Column >= width)

CurrentBlock.Offset.Column -= width;

else if(p.Column + CurrentBlock.Offset.Column < 0)

CurrentBlock.Offset.Column += width;

}

return true;

}

public void RotateBlockCW(){

CurrentBlock.RotateCW();

if (!BlockFits())

CurrentBlock.RotateCCW();

}

public void MoveBlockLeft(){

CurrentBlock.Move(0, -1);

if (!BlockFits())

CurrentBlock.Move(0, 1);

}

public void MoveBlockRight(){

CurrentBlock.Move(0, 1);

if (!BlockFits())

CurrentBlock.Move(0, -1);

}

private bool IsGameOver(){

return !(CircleCell.RowEmptyChecker(0) && CircleCell.RowEmptyChecker(1));

}

private void PlaceBlock(){

foreach (Position p in CurrentBlock.TilePositions())

CircleCell[p.Row, p.Column] = CurrentBlock.Id;

if (currentBlock.Id == 8)

{

foreach (Position p in CurrentBlock.TilePositions())

{

for (int i = -1; i <= 1; i++)

{

for (int j = -1; j <= 1; j++)

{

if (p.Row+i>=0 && p.Row+i< height)

{

if (p.Column + j >= width)

CircleCell[p.Row + i, p.Column + j - width] = 0;

else if (p.Column + j < 0)

CircleCell[p.Row + i, p.Column + j + width] = 0;

else

CircleCell[p.Row + i, p.Column + j] = 0;

}

}

}

}

}

Scores = CircleCell.ClearAllRowsFull();

if (IsGameOver())

GameOver = true;

else

CurrentBlock = BQ.GetAndUpdate();

}

public void MoveBlockDown()

{

CurrentBlock.Move(1, 0);

if (!BlockFits())

{

CurrentBlock.Move(-1, 0);

PlaceBlock();

}

}

}

}

using OpenTK.Graphics.OpenGL;

using OpenTK.Mathematics;

using OpenTK.Windowing.Common;

using OpenTK.Windowing.Desktop;

using OpenTK.Windowing.GraphicsLibraryFramework; //795

namespace Kursach

{

internal class Game : GameWindow

{

int width, height, previouseScores, r, dr, textureId;

double fiX, fiY, ortoWidth, ortoHeight, lag = 0, TIME\_PER\_FRAME = 0.45;

bool pause=false;

GameStatus gameState;

Vector2 cursorPosition, centerPoint;

Button restart, close;

List<Button> buttons = new List<Button>() {};

TextRenderer tr1,tr2,tr3,tr4;

private Vector3[] ColorMass = new Vector3[]{

new Vector3(0,0,0), //black

new Vector3(0,255/255f,255/255f), //cyan

new Vector3(0f,0f,255/255f), //blue

new Vector3(255/255f,165/255f,0), //orange

new Vector3(255/255f,255/255f,0), //yellow

new Vector3(0,128/255f,0),//green

new Vector3(238/255f,130/255f,238/255f),//purpule

new Vector3(255/255f,0,0),//red

new Vector3(1,1,1),//white

};

public Game(GameWindowSettings gSettings, NativeWindowSettings nSettings) : base(gSettings, nSettings){

VSync = VSyncMode.On;

}

protected override void OnLoad(){

base.OnLoad();

width = 12;

height = 22;

ortoHeight = 1200;

ortoWidth = 1600;

r = 60;

dr = 20;

centerPoint = new Vector2(510, 510);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(0, ortoWidth, 0, ortoHeight, -1, 1);

GL.MatrixMode(MatrixMode.Modelview);

gameState = new GameStatus(height, width, centerPoint, r, dr);

textureId = ContentPipe.LoadTexture(@"Content\Consolas\_Alpha\_W.png");

tr1 = new TextRenderer(16, 16, textureId, (float)ortoWidth, (float)ortoHeight);

tr2 = new TextRenderer(16, 16, textureId, (float)ortoWidth, (float)ortoHeight);

tr3 = new TextRenderer(16, 16, textureId, (float)ortoWidth, (float)ortoHeight);

tr4 = new TextRenderer(16, 16, textureId, (float)ortoWidth, (float)ortoHeight);

restart = new Button(1200, 800, 300, 200, Color4.Gray); close = new Button(1200, 500, 300, 200, Color4.Gray);

buttons.Add(restart);

buttons.Add(close);

close.OnMouseDown += CloseEvent;

restart.OnMouseDown += Restart;

tr1.PrepareText(1280, 880, 20, "RESTART", 0.9f);

tr1.PrepareText(1280, 580, 25, "CLOSE", 0.9f);

tr4.PrepareText(100, 1100, 25, "SCORE:" + gameState.Scores.ToString(), 0.9f);

tr2.PrepareText(900, 950, 37, "paused", 3f);

tr3.PrepareText(640, 530, 25, "GAME OVER", 0.9f);

}

protected override void OnUnload(){

gameState.CircleCell.Dispose();

tr1.Dispose(); tr2.Dispose(); tr3.Dispose(); tr4.Dispose();

base.OnUnload();

}

protected override void OnRenderFrame(FrameEventArgs args){

base.OnRenderFrame(args);

GL.ClearColor(Color4.Black);

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

DrawAll(gameState);

if (pause) { GL.Color4(Color4.White); tr2.RenderText(); }

if(gameState.GameOver)

{

GL.Color4(Color4.SlateGray);

GL.Begin(PrimitiveType.Quads);

GL.Vertex2(500, 300);

GL.Vertex2(500, 300 + 500);

GL.Vertex2(500 + 500, 300 + 500);

GL.Vertex2(500 + 500, 300);

GL.End();

GL.Color4(Color4.White);

tr3.RenderText();

}

SwapBuffers();

}

protected override void OnUpdateFrame(FrameEventArgs args){

if (!pause)

{

if (!gameState.GameOver)

{

lag += args.Time;

if (lag > TIME\_PER\_FRAME)

{

while (lag > TIME\_PER\_FRAME)

{

if (previouseScores < gameState.Scores && TIME\_PER\_FRAME - 0.03 > 0.03)

TIME\_PER\_FRAME -= 0.03;

previouseScores = gameState.Scores;

lag -= TIME\_PER\_FRAME;

gameState.MoveBlockDown();

this.Title = "Tetris Scores: " + gameState.Scores.ToString();

}

}

}

}

base.OnUpdateFrame(args);

}

protected override void OnResize(ResizeEventArgs e){

fiX = e.Width / ortoWidth; fiY = e.Height / ortoHeight;

base.OnResize(e);

GL.Viewport(0, 0, e.Width, e.Height);

}

protected override void OnKeyDown(KeyboardKeyEventArgs e){

base.OnKeyDown(e);

if (gameState.GameOver)

return;

else

{

switch (e.Key)

{

case Keys.Left: gameState.MoveBlockLeft(); break;

case Keys.Right: gameState.MoveBlockRight(); break;

case Keys.Up: gameState.RotateBlockCW(); break;

case Keys.Down: gameState.MoveBlockDown(); break;

case Keys.P:; pause = pause ? false : true; break;

}

}

}

protected override void OnMouseDown(MouseButtonEventArgs e){

base.OnMouseDown(e);

for (int i = 0; i < buttons.Count; i++)

if (buttons[i].IsPointInFigure(cursorPosition))

buttons[i].OnMouseDown?.Invoke();

}

protected override void OnMouseMove(MouseMoveEventArgs e){

base.OnMouseMove(e);

cursorPosition = new Vector2((float)(e.Position.X/fiX), (float)ortoHeight - (float)(e.Position.Y/fiY));

}

private void DrawAll(GameStatus gameState){

gameState.CircleCell.Draw(ColorMass);

foreach (Position p in gameState.CurrentBlock.TilePositions())

{

GL.Color3(ColorMass[gameState.CurrentBlock.Id]);

gameState.CircleCell.Cells[p.Row][p.Column].Draw();

}

foreach (Button butt in buttons)

butt.Draw();

GL.Color4(Color4.White);

tr1.RenderText();

tr4.RenderText();

if (previouseScores < gameState.Scores)

tr4.Update(100, 1100, 25, "SCORE:" + gameState.Scores.ToString(), 0.9f);

}

private void Restart(){

gameState = new GameStatus(height, width, centerPoint, r, dr);

tr4.Update(100, 1100, 25, "SCORE:" + gameState.Scores.ToString(), 0.9f);

GC.Collect();

}

private void CloseEvent(){

this.Close();

}

}

}

using System.Drawing.Imaging;

using System.Drawing;

using OpenTK.Graphics.OpenGL;

namespace Kursach

{

internal class ContentPipe{

static public int LoadTexture(string path){

if (!File.Exists(path))

throw new FileNotFoundException($"File not found at '{path}'");

int id = GL.GenTexture();

GL.BindTexture(TextureTarget.Texture2D, id);

Bitmap bmp = new Bitmap(path);

BitmapData data = bmp.LockBits( new Rectangle(0, 0, bmp.Width, bmp.Height),

ImageLockMode.ReadOnly, System.Drawing.Imaging.PixelFormat.Format32bppArgb);

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,data.Width, data.Height, 0,

OpenTK.Graphics.OpenGL.PixelFormat.Rgba,

PixelType.UnsignedByte,

data.Scan0);

bmp.UnlockBits(data);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS, (int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT, (int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);

return id;

}

}

}

using OpenTK.Graphics.OpenGL;

using OpenTK.Mathematics;

namespace Kursach{

internal class CircleCells : IDisposable{

private Cell[][] cells;

internal Cell[][] Cells { get => cells; set => cells = value; }

public CircleCells(int height, int width, Vector2 Center, int r, int dr){

Cells = new Cell[height][];

for (int i = height - 1; i >= 0; i--)

{

Cells[i] = new Cell[width];

}

for (int i = height - 1; i >= 0; i--)

{

for (int j = 0; j < width; j++)

{

Cells[i][j] = new Cell(Center, j, i, r, dr, width);

}

}

this.rows = height;

this.columns = width;

}

private int rows, columns;

public int Cleared { get; set; }

public int Rows { get { return rows; } }

public int Columns { get { return columns; } }

public int this[int r, int c] { get => this.cells[r][c].Id; set => this.cells[r][c].Id = value; }

public bool Insider(int r, int c){

return (r >= 0 && r < Rows) && (c >= 0 && c < Columns);

}

public bool Empty(int r, int c){

return Insider(r, c) && cells[r][c].Id == 0;

}

public bool RowFullChecker(int r){

for (int c = 0; c < Columns; c++)

if (cells[r][c].Id == 0)

return false;

return true;

}

public bool RowEmptyChecker(int r){

for (int c = 0; c < Columns; c++)

if (cells[r][c].Id != 0)

return false;

return true;

}

private void ClearRow(int r){

for (int c = 0; c < Columns; c++)

cells[r][c].Id = 0;

}

private void MoveRowDown(int r, int numRows){

for (int c = 0; c < Columns; c++){

if (r == Rows - 1)

cells[r][c].Id = 0;

else

{

cells[r + numRows][c].Id = cells[r][c].Id;

cells[r][c].Id = 0;

}

}

}

public int ClearRow(){

int cleared = 0;

for (int r = Rows - 1; r > 0; r--)

{

if (RowFullChecker(r))

{

ClearRow(r);

cleared++;

}

if (cleared > 0)

MoveRowDown(r - 1, cleared);

}

return cleared;

}

public int ClearAllRowsFull(){

int temp = 0;

while (RowFullCheckerAll())

temp += ClearRow();

if (temp >= 4)

{

Cleared = 2 \* temp + Cleared;

return Cleared;

}

else

Cleared += temp;

return Cleared;

}

private bool RowFullCheckerAll(){

bool temp = false;

for (int r = Rows - 1; r > 0; r--){

temp = RowFullChecker(r);

if (temp)

break;

}

return temp;

}

public void Draw(Vector3[] ColorMass){

for (int i = 0; i < rows; i++){

for (int j = 0; j < columns; j++){

GL.Color3(ColorMass[cells[i][j].Id]);

cells[i][j].Draw();

}

}

}

public void Dispose(){

for (int i = 0; i < rows; i++){

for (int j = 0; j < columns; j++)

Cells[i][j].Dispose();

}

}

}

internal class Cell : IDisposable

{

float x2, y2, x3, y3, x4, y4, x5, y5;

int id, bufferId; float[] temp;

public int Id { get { return id; } set { id = value; } }

public Cell(Vector2 Center, int j, int i, int r, int dr, int width){

float dAlpha = 2 \* (float)Math.PI / width;

x2 = Center.X + (r + i \* dr) \* (float)Math.Cos(j \* dAlpha);

y2 = Center.Y + (r + i \* dr) \* (float)Math.Sin(j \* dAlpha);

x3 = Center.X + (r + i \* dr + dr) \* (float)Math.Cos(j \* dAlpha);

y3 = Center.Y + (r + i \* dr + dr) \* (float)Math.Sin(j \* dAlpha);

x4 = Center.X + (r + i \* dr + dr) \* (float)Math.Cos((j + 1) \* dAlpha);

y4 = Center.Y + (r + i \* dr + dr) \* (float)Math.Sin((j + 1) \* dAlpha);

x5 = Center.X + (r + i \* dr) \* (float)Math.Cos((j + 1) \* dAlpha);

y5 = Center.Y + (r + i \* dr) \* (float)Math.Sin((j + 1) \* dAlpha);

temp = new float[] { x2, y2, x3, y3, x4, y4, x5, y5 };

bufferId = GL.GenBuffer();

GL.BindBuffer(BufferTarget.ArrayBuffer, bufferId);

GL.BufferData(BufferTarget.ArrayBuffer, temp.Length \* sizeof(float), temp, BufferUsageHint.StaticDraw);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

}

public void Draw(){

GL.EnableClientState(ArrayCap.VertexArray);

GL.BindBuffer(BufferTarget.ArrayBuffer, bufferId);

GL.VertexPointer(2, VertexPointerType.Float, 0, 0);

GL.DrawArrays(PrimitiveType.Polygon, 0, 4);

GL.Color4(Color4.Red);

GL.DrawArrays(PrimitiveType.LineLoop, 0, 4);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

GL.DisableClientState(ArrayCap.VertexArray);

}

public void Dispose(){

GL.DeleteBuffer(bufferId);

}

}

}

using OpenTK.Graphics.OpenGL;

using OpenTK.Mathematics;

using OpenTK.Windowing.Common;

namespace Kursach{

delegate void MouseDelegate();

internal class Button{

float x, y, width, height;

Color4 color; float[] temp;

int bufferId;

public MouseDelegate OnMouseDown;

public Button(float x, float y, float width, float height, Color4 color){

this.x = x;

this.y = y;

this.width = width;

this.height = height;

this.color = color;

temp = new float[] {x,y,x,y+height,x+width,y+height,x+width,y };

bufferId = GL.GenBuffer();

GL.BindBuffer(BufferTarget.ArrayBuffer, bufferId);

GL.BufferData(BufferTarget.ArrayBuffer, temp.Length \* sizeof(float), temp, BufferUsageHint.StaticDraw);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

}

public void Draw(){

GL.Color4(color);

GL.EnableClientState(ArrayCap.VertexArray);

GL.BindBuffer(BufferTarget.ArrayBuffer, bufferId);

GL.VertexPointer(2, VertexPointerType.Float, 0, 0);

GL.DrawArrays(PrimitiveType.Polygon, 0, 4);

GL.BindBuffer(BufferTarget.ArrayBuffer, 0);

GL.DisableClientState(ArrayCap.VertexArray);

}

public bool IsPointInFigure(float pointX, float pointY){

if (pointX < x) return false;

if (pointY < y) return false;

if (pointX > x + width) return false;

if (pointY > y + height) return false;

return true;

}

public bool IsPointInFigure(Vector2 point){

return IsPointInFigure(point.X, point.Y);

}

}

}

namespace Kursach{

internal class BlockQueue{

private Block[] blocks = new Block[]{

new Block( // BlockI

new Position[][]{

new Position[] { new(1,0), new(1,1), new(1,2), new(1,3) },

new Position[] { new(0,2), new(1,2), new(2,2), new(3,2) },

new Position[] { new(2,0), new(2,1), new(2,2), new(2,3) },

new Position[] { new(0,1), new(1,1), new(2,1), new(3,1) }

}, new Position(-1, 3), 1

),

new Block( //J

new Position[][]{

new Position[] {new(0, 0), new(1, 0), new(1, 1), new(1, 2)},

new Position[] {new(0, 1), new(0, 2), new(1, 1), new(2, 1)},

new Position[] {new(1, 0), new(1, 1), new(1, 2), new(2, 2)},

new Position[] {new(0, 1), new(1, 1), new(2, 1), new(2, 0)}

},new Position(0, 3),2

),

new Block(//L

new Position[][]{

new Position[] {new(0,2), new(1,0), new(1,1), new(1,2)},

new Position[] {new(0,1), new(1,1), new(2,1), new(2,2)},

new Position[] {new(1,0), new(1,1), new(1,2), new(2,0)},

new Position[] {new(0,0), new(0,1), new(1,1), new(2,1)}

},new Position(0, 3),3

),

new Block(//O

new Position[][]{

new Position[] { new(0,0), new(0,1), new(1,0), new(1,1) },

new Position[] { new(0,0), new(0,1), new(1,0), new(1,1) },

new Position[] { new(0,0), new(0,1), new(1,0), new(1,1) },

new Position[] { new(0,0), new(0,1), new(1,0), new(1,1) }

}, new Position(0,4),4

),

new Block( //S

new Position[][]{

new Position[] { new(0,1), new(0,2), new(1,0), new(1,1) },

new Position[] { new(0,1), new(1,1), new(1,2), new(2,2) },

new Position[] { new(1,1), new(1,2), new(2,0), new(2,1) },

new Position[] { new(0,0), new(1,0), new(1,1), new(2,1) }

},new Position(0,3),5

),

new Block( // T

new Position[][]{

new Position[] {new(0,1), new(1,0), new(1,1), new(1,2)},

new Position[] {new(0,1), new(1,1), new(1,2), new(2,1)},

new Position[] {new(1,0), new(1,1), new(1,2), new(2,1)},

new Position[] {new(0,1), new(1,0), new(1,1), new(2,1)}

}, new Position(0,3),6

),

new Block( // Z

new Position[][]{

new Position[] {new(0,0), new(0,1), new(1,1), new(1,2)},

new Position[] {new(0,2), new(1,1), new(1,2), new(2,1)},

new Position[] {new(1,0), new(1,1), new(2,1), new(2,2)},

new Position[] {new(0,1), new(1,0), new(1,1), new(2,0)}

}, new Position(0,3),7

),

new Block(//Bomb

new Position[][]{

new Position[] { new(0,0) },

new Position[] { new(0,0) },

new Position[] { new(0,0) },

new Position[] { new(0,0) }

}, new Position(0,4),8

),

};

private Random random = new Random();

public Block NextBlock { get; set; }

private Block RandomBlock(){

return blocks[random.Next(blocks.Length)];

}

public BlockQueue(){

NextBlock = RandomBlock();

}

public Block GetAndUpdate(){

Block block = NextBlock;

do

{

NextBlock = RandomBlock();

} while (block.Id == NextBlock.Id);

return block;

}

}

}

namespace Kursach

{

internal class Block

{

internal Position[][] Tiles { get; set; }

internal Position StartOffset { get; set; }

public int Id { get; set; }

internal Position Offset { get => offset; set => offset = value; }

private int rotationState;

private Position offset;

static internal int Width { get; set; }

static internal int Height { get; set; }

public Block()

{

Offset = new Position(StartOffset.Row, StartOffset.Column);

}

public Block(Position[][] Tiles, Position StartOffset, int Id)

{

this.Id = Id;

this.Tiles = Tiles;

this.StartOffset = StartOffset;

Offset = new Position(this.StartOffset.Row, this.StartOffset.Column);

}

public void Move(int rows, int columns)

{

Offset.Row += rows;

Offset.Column += columns;

}

public void Reset()

{

rotationState = 0;

Offset.Row = StartOffset.Row;

Offset.Column = StartOffset.Column;

}

public IEnumerable<Position> TilePositions()

{

foreach(Position p in Tiles[rotationState])

{

if(p.Column + Offset.Column >= Width)

{

yield return new Position(p.Row + Offset.Row, p.Column + Offset.Column - Width);

}

else if(p.Column + Offset.Column < 0)

{

yield return new Position(p.Row + Offset.Row, p.Column + Offset.Column + Width);

}

else

yield return new Position(p.Row + Offset.Row, p.Column + Offset.Column);

}

}

public void RotateCW()

{

if(rotationState !=3)

{

rotationState++;

}

else { rotationState=0; }

}

public void RotateCCW()

{

if(rotationState == 0)

rotationState = Tiles.Length - 1;

else

rotationState--;

}

}

}

# Руководство пользователя

При запуске программы пользователи видит перед собой стартовую игру, представленную на рисунке Рисунок 3.1.

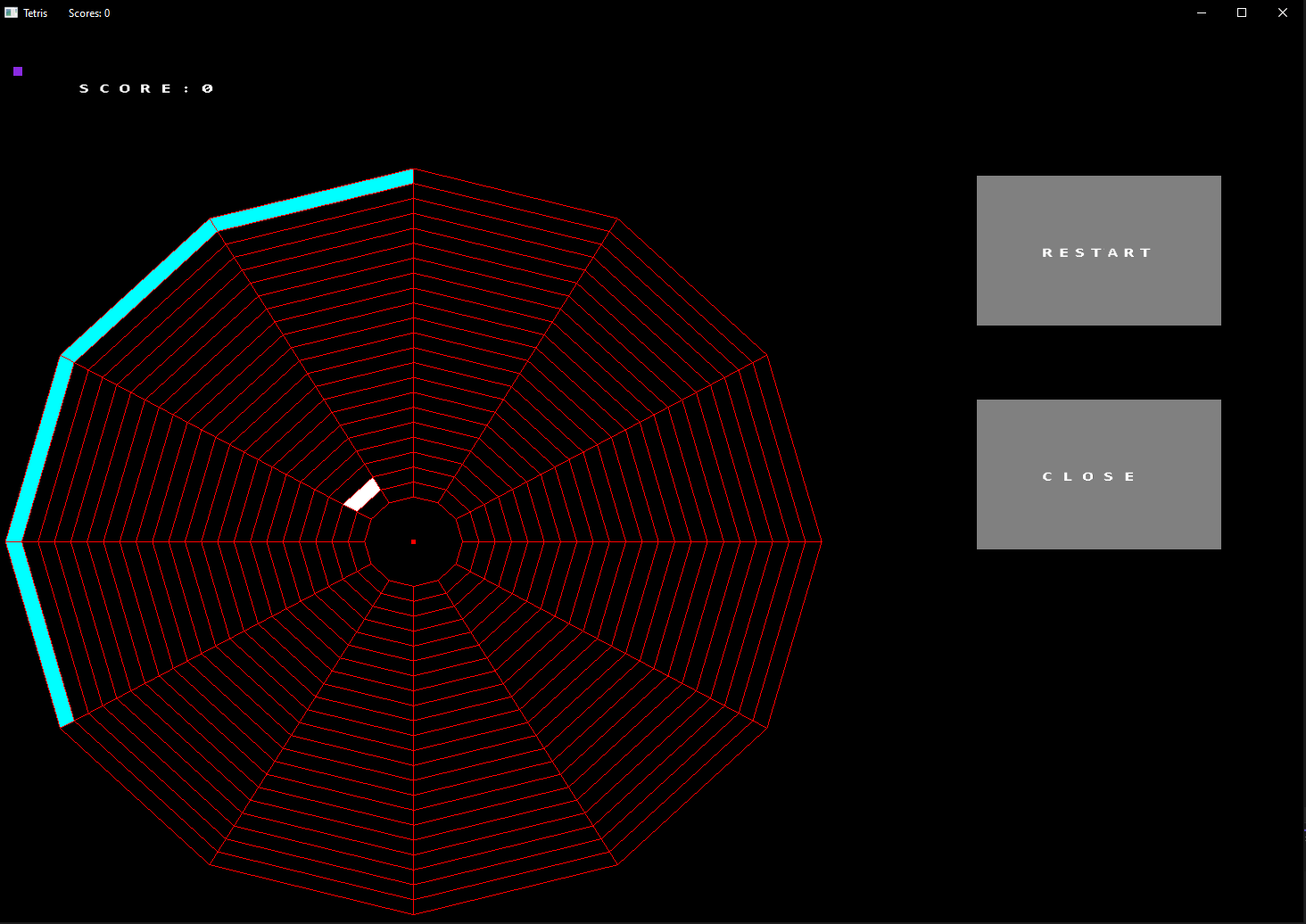


Рисунок 3.1 — Игровое окно

Всё управление игрой ориентированно на использование стрелочек на клавиатуру, где стрелочка вниз отвечает за перенос блока к дальним радиусам, стрелочка вправо вращает блок против часовой стрелки, стрелочка влево вращает блок по часовой стрелочке, стрелочка вверх вращает блок относительно своей оси.

Весь перечень блоков аналогичен перечню блоков тетриса, за исключением особого белого блока – бомбы. Этот блок, при невозможности продолжить движении «вниз», взрывается, уничтожая всё в радиусе 1 клетки относительно себя.

При нажатии кнопки «Restart» пользователь полностью перезапускает игру. При нажатии кнопки «Close» игра закрывается. При нажатии клавиши «p» игра останавливается, как показано на рисунке Рисунок 3.2.

Набранные очки отображаются в левом верхнем углу в виде надписи «Score», а также в названии игрового окна.

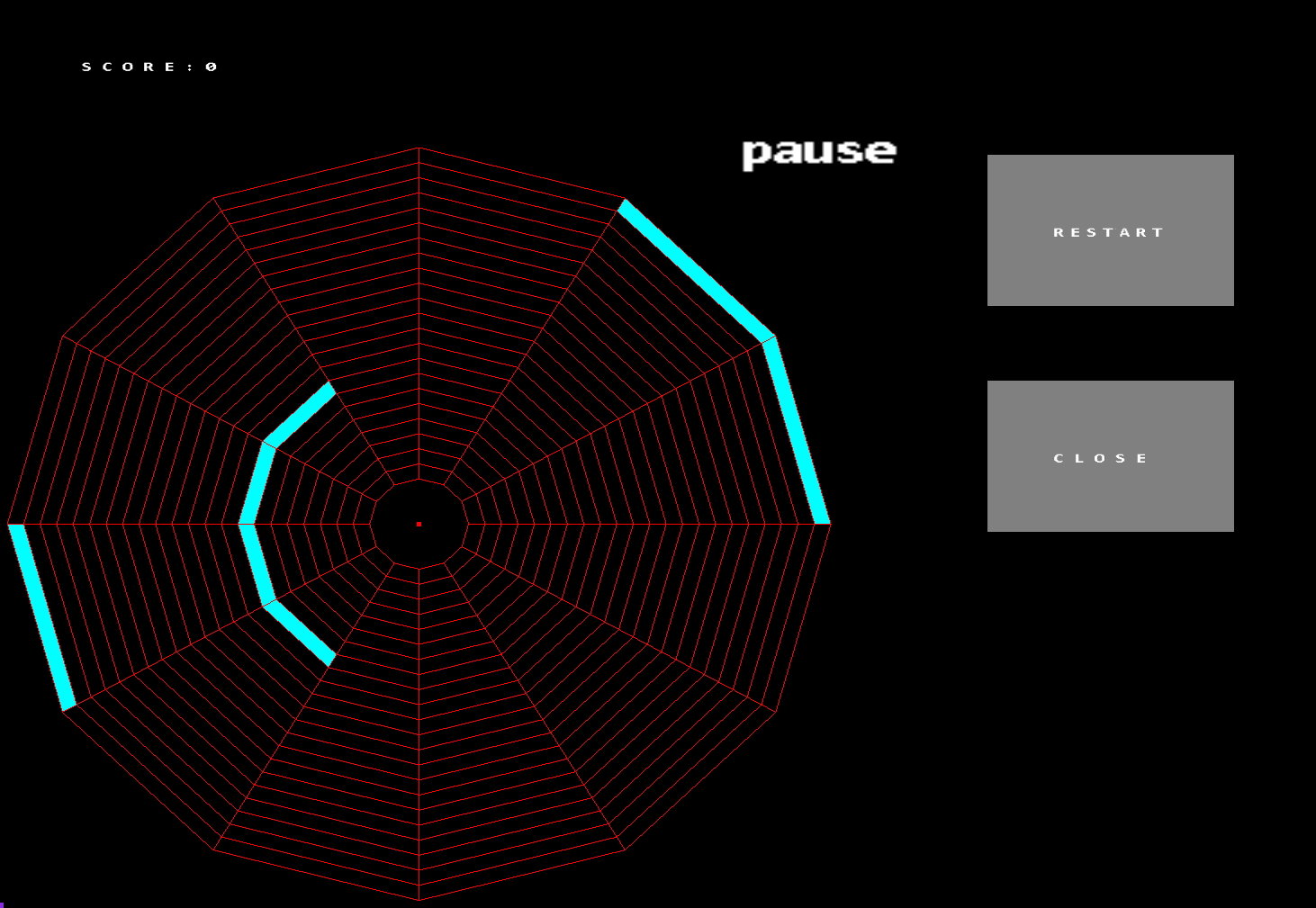


Рисунок 3.2 — Игровое окно на паузе

# Заключение

В ходе данной работы была создана игра, подобная игре Tetris, на языке программирования C# с применением технологии OpenTK.

# Список использованной литературы

1. Либерти Д. Программирование на C#. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 588 с., ил. ISBN 5-93286-038-3
2. Рихтер Дж. Р55 CLR via С#. Программирование на платформе Microsoft .NET Framework 4.0 на языке С#. 3-е изд. - СПб.: Питер, 201 2.-928 с.: ил. ISBN 978-5-459-00297-3