МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра САПР

Отчет

о выполнении производственной (эксплуатационной) практики

на тему: «Разработка структур данных и алгоритмов для игры «Тетрис»»

Выполнил студент: Ишенбеков Н.

Группа: 21ВА1

Руководитель: Гудков П.А.

2024

Оглавление

[Введение 3](#_Toc170222788)

[Цель работы: 4](#_Toc170222789)

[Ход работы: 4](#_Toc170222790)

[Структура и объявление структуры представляет в виде: 4](#_Toc170222791)

[Описание основных методов: 4](#_Toc170222792)

[Приложение 6](#_Toc170222793)

[Код программы: 8](#_Toc170222794)

**Введение**

**Структуры данных и алгоритмы для игры «Тетрис».**

Те́трис – компьютерная игра, первоначально изобретённая и разработанная советским программистом Алексеем Пажитновым. «Тетрис» представляет собой головоломку, построенную на использовании геометрических фигур «тетрамино» — разновидности полимино, состоящих из четырёх квадратов. В мире есть много разновидностей тетриса, в моем случае я реализовал вариант в котором есть 22 строки и 10 столбцов и есть 7 разновидностей падающих фигур.

# **Цель работы:**

Разработка структур данных и алгоритмов для игры «Тетрис».

# **Ход работы:**

## **Структура и объявление структуры представляет в виде:**

public class GameGrid – класс для основного поля игры

public class GameState – класс для отслеживания состояния игры

public class BlockQueue – класс содержащий массив блоков

public class Position – класс содержащий расположение блоков

public abstract class Block – абстрактный класс для всех видов блоков

public class BlockI : Block – класс для болка I

public class BlockJ : Block – класс для болка J

public class BlockL : Block – класс для болка L

public class BlockO : Block – класс для болка O

public class BlockS : Block – класс для болка S

public class BlockT : Block – класс для болка T

public class BlockZ : Block – класс для болка Z

private readonly ImageSource[] tileImages – массив содержащий изображения отдельных кирпичиков

private readonly ImageSource[] blockImages – массив содержащий изображения фигур для окна следующего блока и окна удержания блока

private readonly Image[,] imageControls – двумерный массив управления элементами изображения

## **Описание основных методов:**

public MainWindow() – точка входа в программу

private Image[,] SetUpGameCanvas(GameGrid grid) – метод настройки изображения элементов на холсте

private void DrawGrid(GameGrid grid) – отрисовка игровой сетки

private void DrawBlock(Block block) – отрисовка текущего блока

private void DrawNextBlock(BlockQueue blockQueue) - отрисовка следующего блока

private void DrawHeldBlock(Block heldBlock) – отрисовка удерживаемого блока

private void DrawGhostBlock(Block block) – отрисовка призрачного блока

private void Draw(GameState state) – отрисовка всего окна

private async Task GameLoop() – игровой цикл

private void Window\_KeyDown(object sender, KeyEventArgs e) – обработчик события нажатия кнопки

private async void GameCanvas\_Loaded(object sender, RoutedEventArgs e) – обработчик события загрузки поля с игрой

private async void PlayAgain\_Click(object sender, RoutedEventArgs e) – обработчик события нажатия кнопки PlayAgain

# **Приложение**

Стартовый вид игры при запуске

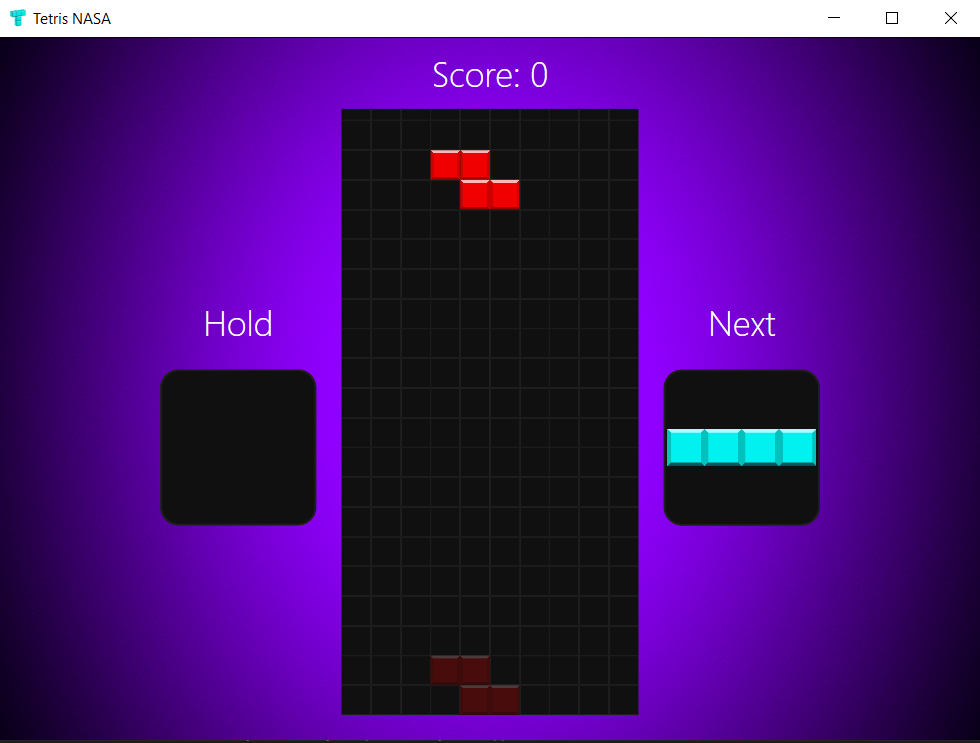


Рисунок 1 Начальное окно при запуске игры

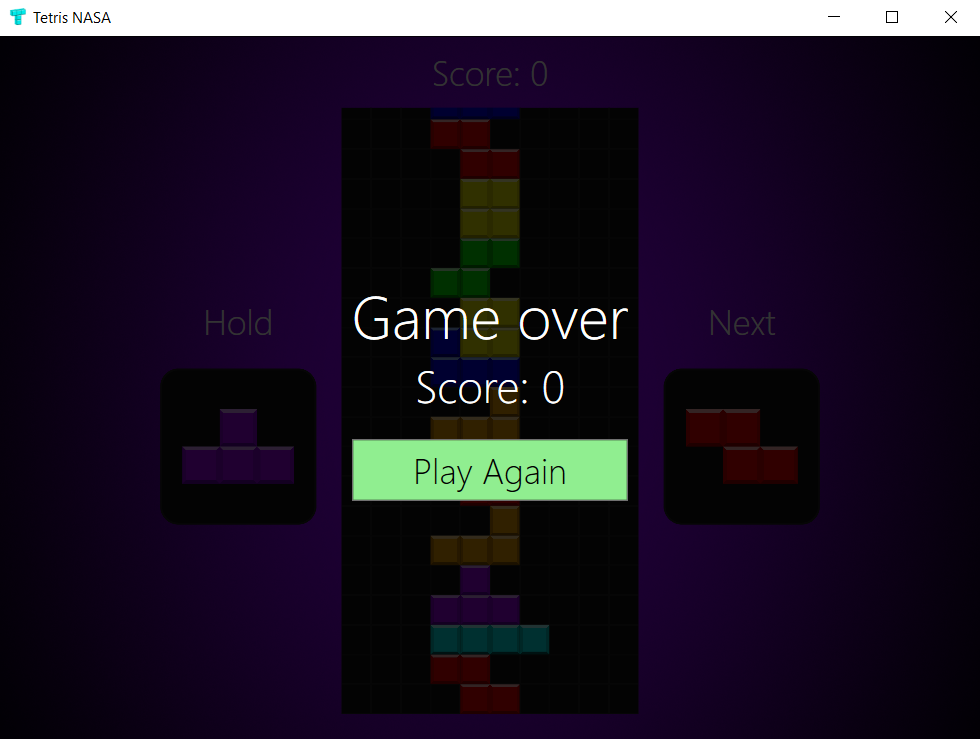


Рисунок 2 Окно при окончании игры

**Заключение**

были разработаны структуры данных и алгоритмы для игры «Тетрис». В ходе разработки были изучены методы создания оконных приложений WPF (Windows Presentation Foundation) на языке C#. В игру были добавлены следующие механики: введение игрового счета текущей сессии, увеличения сложности игры во время геймплея, предпросмотр следующей фигуры, удержание текущей фигуры и отображение конечной позиции фигуры (призрачная фигура).

# **Код программы:**

MainWindow.xaml.cs

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace Tetris\_WPFApp

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private readonly ImageSource[] tileImages = new ImageSource[]

{

new BitmapImage(new Uri("Assets/TileEmpty.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TileCyan.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TileBlue.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TileOrange.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TileYellow.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TileGreen.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TilePurple.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/TileRed.png", UriKind.Relative))

};

private readonly ImageSource[] blockImages = new ImageSource[]

{

new BitmapImage(new Uri("Assets/Block-Empty.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-I.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-J.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-L.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-O.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-S.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-T.png", UriKind.Relative)),

new BitmapImage(new Uri("Assets/Block-Z.png", UriKind.Relative))

};

// Двумерный массив управления элементами изображения

private readonly Image[,] imageControls;

// Переменные для контроля сложности игры

private readonly int maxDelay = 1000;

private readonly int minDelay = 75;

private readonly int delayDecreace = 25;

// Состояние игры

private GameState gameState = new GameState();

public MainWindow()

{

InitializeComponent();

imageControls = SetUpGameCanvas(gameState.GameGrid);

}

// Метод настройки изображения элементов на холсте

private Image[,] SetUpGameCanvas(GameGrid grid)

{

Image[,] imageControls = new Image[grid.Rows, grid.Columns];

int cellSize = 25;

for (int r = 0; r < grid.Rows; r++)

{

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

{

Image imageControl = new Image

{

Width = cellSize,

Height = cellSize

};

Canvas.SetTop(imageControl, (r - 2) \* cellSize + 10);

Canvas.SetLeft(imageControl, c \* cellSize);

GameCanvas.Children.Add(imageControl);

imageControls[r, c] = imageControl;

}

}

return imageControls;

}

// Отрисовка игровой сетки

private void DrawGrid(GameGrid grid)

{

for (int r = 0; r < grid.Rows; r++)

{

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

{

int id = grid[r, c];

imageControls[r, c].Opacity = 1;

imageControls[r, c].Source = tileImages[id];

}

}

}

// Отрисовка текущего блока

private void DrawBlock(Block block)

{

foreach (Position p in block.TilePositions())

{

imageControls[p.Row, p.Column].Opacity = 1;

imageControls[p.Row, p.Column].Source = tileImages[block.Id];

}

}

// Отрисовка следующего блока

private void DrawNextBlock(BlockQueue blockQueue)

{

Block next = blockQueue.NextBlock;

NextImage.Source = blockImages[next.Id];

}

// Отрисовка удерживаемого блока

private void DrawHeldBlock(Block heldBlock)

{

if (heldBlock == null)

{

HoldImage.Source = blockImages[0];

}

else

{

HoldImage.Source = blockImages[heldBlock.Id];

}

}

// Отрисовка призрачного блока

private void DrawGhostBlock(Block block)

{

int dropDistance = gameState.BlockDropDistance();

foreach (Position p in block.TilePositions())

{

imageControls[p.Row + dropDistance, p.Column].Opacity = 0.25;

imageControls[p.Row + dropDistance, p.Column].Source = tileImages[block.Id];

}

}

// Отрисовка сетки и текущего блока

private void Draw(GameState state)

{

DrawGrid(state.GameGrid);

DrawGhostBlock(gameState.CurrentBlock);

DrawBlock(state.CurrentBlock);

DrawNextBlock(gameState.BlockQueue);

DrawHeldBlock(gameState.HeldBlock);

ScoreText.Text = $"Score: {gameState.Score}";

}

// Игровой цикл

private async Task GameLoop()

{

Draw(gameState);

while (!gameState.GameOver)

{

int delay = Math.Max(minDelay, maxDelay - (gameState.Score \* delayDecreace));

await Task.Delay(delay);

gameState.MoveBlockDown();

Draw(gameState);

}

GameOverMenu.Visibility = Visibility.Visible;

FinalScoreText.Text = $"Score: {gameState.Score}";

}

// Обработчик события нажатия кнопки

private void Window\_KeyDown(object sender, KeyEventArgs e)

{

if (gameState.GameOver) { return; }

switch (e.Key)

{

case Key.Left:

gameState.MoveBlockLeft();

break;

case Key.Right:

gameState.MoveBlockRight();

break;

case Key.Down:

gameState.MoveBlockDown();

break;

case Key.Up:

gameState.RotateBlockCW();

break;

case Key.Z:

gameState.RotateBlockCCW();

break;

case Key.C:

gameState.HoldBlock();

break;

case Key.Space:

gameState.DropBlock();

break;

default:

return;

}

Draw(gameState);

}

// Обработчик события загрузки поля с игрой

private async void GameCanvas\_Loaded(object sender, RoutedEventArgs e)

{

await GameLoop();

}

// Обработчик нажатия кнопки PlayAgain

private async void PlayAgain\_Click(object sender, RoutedEventArgs e)

{

gameState = new GameState();

GameOverMenu.Visibility = Visibility.Hidden;

await GameLoop();

}

}

}

MainWindow.xaml

<Window x:Class="Tetris\_WPFApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Tetris\_WPFApp"

mc:Ignorable="d"

Title="Tetris NASA" Height="600" Width="800"

MinWidth="600" MinHeight="600"

Foreground="White"

FontFamily="Segoe UI Light" FontSize="28"

KeyDown="Window\_KeyDown">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.Background>

<ImageBrush ImageSource="Assets/Background.png"/>

</Grid.Background>

<Viewbox Grid.Row="1"

Grid.Column="1"

Margin="0, 0, 0, 20">

<Canvas x:Name="GameCanvas"

Grid.Row="1"

Grid.Column="1"

Background="#101010"

Width="250"

Height="510"

ClipToBounds="True"

Loaded="GameCanvas\_Loaded"/>

</Viewbox>

<TextBlock x:Name="ScoreText"

Grid.Row="0"

Grid.Column="1"

Text="Score"

Margin="0, 10"

TextAlignment="Center"/>

<StackPanel Grid.Row="1"

Grid.Column="0"

VerticalAlignment="Center"

HorizontalAlignment="Right">

<TextBlock Text="Hold"

TextAlignment="Center"/>

<Image x:Name="HoldImage"

Margin="20"

Width="125"/>

</StackPanel>

<StackPanel Grid.Row="1"

Grid.Column="2"

VerticalAlignment="Center"

HorizontalAlignment="Left">

<TextBlock Text="Next"

TextAlignment="Center"/>

<Image x:Name="NextImage"

Margin="20"

Width="125"/>

</StackPanel>

<Grid x:Name="GameOverMenu"

Background="#CC000000"

Grid.RowSpan="2"

Grid.ColumnSpan="3"

Visibility="Hidden">

<StackPanel HorizontalAlignment="Center"

VerticalAlignment="Center">

<TextBlock Text="Game over"

FontSize="48"

TextAlignment="Center"/>

<TextBlock x:Name="FinalScoreText"

Text="Score: "

FontSize="36"

TextAlignment="Center"/>

<Button Content="Play Again"

Background="LightGreen"

Margin="0, 20, 0, 0"

Padding="5"

Click="PlayAgain\_Click"/>

</StackPanel>

</Grid>

</Grid>

</Window>

Position.cs

namespace Tetris\_WPFApp

{

public class Position

{

public int Row { get; set; }

public int Column { get; set; }

public Position(int row, int column)

{

Row = row;

Column = column;

}

}

}

GameGrid.cs

namespace Tetris\_WPFApp

{

public class GameGrid

{

// Двумерный массив и размеры массива

private readonly int[,] grid;

public int Rows { get; }

public int Columns { get; }

// Индексация для двумерного массива

public int this[int r, int c]

{

get => grid[r, c];

set => grid[r, c] = value;

}

// Конструктор

public GameGrid(int rows, int columns)

{

Rows = rows;

Columns = columns;

grid = new int[Rows, Columns];

}

// Проверка находится ли внутри сетки

public bool IsInside(int r, int c)

{

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

}

// Проверка пуста ли ячейка

public bool IsEmpty(int r, int c)

{

return IsInside(r, c) && grid[r, c] == 0;

}

// Заполенена ли строка целиком

public bool IsRowFull(int r)

{

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

{

if (grid[r, c] == 0)

{

return false;

}

}

return true;

}

// Пуста ли строка

public bool IsRowEmpty(int r)

{

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

{

if (grid[r, c] != 0)

{

return false;

}

}

return true;

}

// Очистка строки

private void ClearRow(int r)

{

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

{

grid[r, c] = 0;

}

}

// Перемещение вниз всей строки на опред кол-во строк

private void MoveRowDown(int r, int numRows)

{

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

{

grid[r + numRows, c] = grid[r, c];

grid[r, c] = 0;

}

}

// Метод четкого заполнения строк

public int ClearFullRows()

{

int cleared = 0;

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

{

if (IsRowFull(r))

{

ClearRow(r);

cleared++;

}

else if (cleared > 0)

{

MoveRowDown(r, cleared);

}

}

return cleared;

}

}

}

GameState.cs

namespace Tetris\_WPFApp

{

public class GameState

{

// Резервное поле для следующего блока

private Block currentBlock;

public Block CurrentBlock

{

get => currentBlock;

private set

{

currentBlock = value;

currentBlock.Reset();

for (int i = 0; i < 2; i++)

{

currentBlock.Move(1, 0);

if (!BlockFits())

{

currentBlock.Move(-1, 0);

}

}

}

}

// Игровая сетка

public GameGrid GameGrid { get; }

// Очередь блоков

public BlockQueue BlockQueue { get; }

// Логическое значение "Конец игры"

public bool GameOver { get; private set; }

// Счет игры

public int Score { get; private set; }

// Удерживаемый блок

public Block HeldBlock { get; private set; }

// Можно ли удержать

public bool CanHold { get; private set; }

// Конструктор с 22 строками и 10 столбцами

public GameState()

{

GameGrid = new GameGrid(22, 10);

BlockQueue = new BlockQueue();

CurrentBlock = BlockQueue.GetAndUpdate();

CanHold = true;

}

// Метод проверки является ли блок легальным

private bool BlockFits()

{

foreach (Position p in CurrentBlock.TilePositions())

{

if (!GameGrid.IsEmpty(p.Row, p.Column))

{

return false;

}

}

return true;

}

// Удержание блока

public void HoldBlock()

{

if (!CanHold)

{

return;

}

if (HeldBlock == null)

{

HeldBlock = currentBlock;

CurrentBlock = BlockQueue.GetAndUpdate();

}

else

{

Block temp = currentBlock;

CurrentBlock = HeldBlock;

HeldBlock = temp;

}

CanHold = false;

}

// Поворот текущего блока по часовой стрелке, если возможно

public void RotateBlockCW()

{

CurrentBlock.RotateCW();

// Если нельзя повернуть, то возвращаем в исходное положение

if (!BlockFits()) { CurrentBlock.RotateCCW(); }

}

// Поворот текущего блока против часовой стрелки, если возможно

public void RotateBlockCCW()

{

CurrentBlock.RotateCCW();

// Если нельзя повернуть, то возвращаем в исходное положение

if (!BlockFits()) { CurrentBlock.RotateCW(); }

}

// Перемещение текущего блока влево

public void MoveBlockLeft()

{

CurrentBlock.Move(0, -1);

// Если нельзя переместить, то возвращаем в исходное положение

if (!BlockFits()) { CurrentBlock.Move(0, 1); }

}

// Перемещение текущего блока вправо

public void MoveBlockRight()

{

CurrentBlock.Move(0, 1);

// Если нельзя переместить, то возвращаем в исходное положение

if (!BlockFits()) { CurrentBlock.Move(0, -1); }

}

// Проверка окончена ли игра

private bool IsGameOver()

{

return !(GameGrid.IsRowEmpty(0) && GameGrid.IsRowEmpty(1));

}

// Метод который будет вызван, если фигура не может быть перемещен вниз

private void PlaceBlock()

{

// Перебор позиций текущего блока и установка этих позиций

foreach (Position p in CurrentBlock.TilePositions())

{

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

}

Score += GameGrid.ClearFullRows();

if (IsGameOver())

{

GameOver = true;

}

else

{

CurrentBlock = BlockQueue.GetAndUpdate();

CanHold = true;

}

}

// Метод перемещения текущего блока вниз

public void MoveBlockDown()

{

CurrentBlock.Move(1, 0);

// Если нельзя переместить, то возвращаем в исходное положение

if (!BlockFits())

{

CurrentBlock.Move(-1, 0);

PlaceBlock();

}

}

// Количество пустых ячеек под блоком

private int TileDropDistance(Position p)

{

int drop = 0;

while (GameGrid.IsEmpty(p.Row + drop + 1, p.Column))

{

drop++;

}

return drop;

}

// На сколько блоков можно переместить вниз

public int BlockDropDistance()

{

int drop = GameGrid.Rows;

foreach (Position p in CurrentBlock.TilePositions())

{

drop = System.Math.Min(drop, TileDropDistance(p));

}

return drop;

}

// Перемещение блока вниз на сколько возможно

public void DropBlock()

{

CurrentBlock.Move(BlockDropDistance(), 0);

PlaceBlock();

}

}

}

Block.cs

namespace Tetris\_WPFApp

{

public abstract class Block

{

protected abstract Position[][] Tiles { get; } // Позиция клеток в 4 состояния вращения

protected abstract Position StartOfSet { get; } // Начальное смещение

public abstract int Id { get; } // Код фигуры

private int rotationState; // Состояние вращения

private Position offset; // Текущее смещение

public Block()

{

offset = new Position(StartOfSet.Row, StartOfSet.Column);

}

// Метод который возвращает позицию сетки занятую блоком с учетом поворота и смещения

public IEnumerable<Position> TilePositions()

{

foreach (Position p in Tiles[rotationState])

{

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

}

}

// Метод поворота фигуры на 90 градусов по часовой стрелке

public void RotateCW()

{

rotationState = (rotationState + 1) % Tiles.Length;

}

// Метод поворота фигуры на 90 градусов против часовой стрелки

public void RotateCCW()

{

if (rotationState == 0) { rotationState = Tiles.Length - 1; }

else { rotationState--; }

}

// Метод который перемещает блок на заданное количество строк и столбцов

public void Move(int rows, int columns)

{

offset.Row += rows;

offset.Column += columns;

}

// Метод сброса позиции и вращения

public void Reset()

{

rotationState = 0;

offset.Row = StartOfSet.Row;

offset.Column = StartOfSet.Column;

}

}

}

BlockI.cs

namespace Tetris\_WPFApp

{

public class BlockI : Block

{

// Позиции для 4 состояний вращения

private readonly Position[][] tiles = 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) }

};

public override int Id => 1; // Код состояния вращения фигуры

protected override Position StartOfSet => new Position(-1, 3); // Начальное смещение (середина верхнего ряда)

protected override Position[][] Tiles => tiles;

}

}

BlockJ.cs

namespace Tetris\_WPFApp

{

public class BlockJ : Block

{

// Позиции для 4 состояний вращения

private readonly Position[][] tiles = 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, 0), new(2, 1) }

};

public override int Id => 2; // Код состояния вращения фигуры

protected override Position StartOfSet => new Position(0, 3); // Начальное смещение (середина верхнего ряда)

protected override Position[][] Tiles => tiles;

}

}

BlockL

namespace Tetris\_WPFApp

{

public class BlockL : Block

{

// Позиции для 4 состояний вращения

private readonly Position[][] tiles = 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) }

};

public override int Id => 3; // Код состояния вращения фигуры

protected override Position StartOfSet => new Position(0, 3); // Начальное смещение (середина верхнего ряда)

protected override Position[][] Tiles => tiles;

}

}

BlockO.cs

namespace Tetris\_WPFApp

{

public class BlockO : Block

{

private readonly Position[][] tiles = new Position[][]

{

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

};

public override int Id => 4;

protected override Position StartOfSet => new Position(0, 4);

protected override Position[][] Tiles => tiles;

}

}

BlockS.cs

namespace Tetris\_WPFApp

{

internal class BlockS : Block

{

// Позиции для 4 состояний вращения

private readonly Position[][] tiles = 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) }

};

public override int Id => 5; // Код состояния вращения фигуры

protected override Position StartOfSet => new Position(0, 3); // Начальное смещение (середина верхнего ряда)

protected override Position[][] Tiles => tiles;

}

}

BlockT.cs

namespace Tetris\_WPFApp

{

public class BlockT : Block

{

// Позиции для 4 состояний вращения

private readonly Position[][] tiles = 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) }

};

public override int Id => 6; // Код состояния вращения фигуры

protected override Position StartOfSet => new Position(0, 3); // Начальное смещение (середина верхнего ряда)

protected override Position[][] Tiles => tiles;

}

}

BlockZ.cs

namespace Tetris\_WPFApp

{

public class BlockZ : Block

{

// Позиции для 4 состояний вращения

private readonly Position[][] tiles = 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) }

};

public override int Id => 7; // Код состояния вращения фигуры

protected override Position StartOfSet => new Position(0, 3); // Начальное смещение (середина верхнего ряда)

protected override Position[][] Tiles => tiles;

}

}

BlockQueue.cs

namespace Tetris\_WPFApp

{

public class BlockQueue

{

// Массив блоков

private readonly Block[] blocks = new Block[]

{

new BlockI(),

new BlockJ(),

new BlockL(),

new BlockO(),

new BlockS(),

new BlockT(),

new BlockZ()

};

// Случайный объект

private readonly Random random = new Random();

// Следующий блок в очереди

public Block NextBlock { get; private set; }

// Конструктор

public BlockQueue()

{

NextBlock = RandomBlock();

}

// Метод возвращает случайный блок

private Block RandomBlock()

{

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

}

// Метод возвращает следующий блок и обновляет свойство NextBlock

public Block GetAndUpdate()

{

Block block = NextBlock;

do { NextBlock = RandomBlock(); }

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

return block;

}

}

}

Ассеты можно скачать здесь:

<https://1drv.ms/f/s!AgoL24NcaWizgZJGBOD62f3jRmu8Gg?e=DhbPI2>

[дата обращения 25.06.2024]