Министерство цифрового развития, связи и массовых коммуникаций РФ

Сибирский Государственный Университет Телекоммуникаций и Информатики СибГУТИ

Кафедра САПР

Отчет по курсовой работе

По дисциплине: «ПиОГИ»

**Выполнил(а):** Чебаевский Н.А

**Группа:** ТТМ-22

**Проверил(а):** \_\_\_\_\_

\

Новосибирск 2022 г

стр.

**1.Вариант**

Вариант 3 - Игровое приложение

**2.Задание**

Разработать игру "Тетрис" с использованием языка программирования C и технологии WPF. В работе предполагается реализовать основные функции игры, включая управление блоками, их вращение и падение, а также механизм очистки заполненных рядов.

**3.Описание проекта**

Проект реализован на языке C с использованием технологии WPF (Windows Presentation Foundation). Он представляет собой игру "Тетрис", в которой игрок управляет падающими тетромино и старается заполнить горизонтальные линии для их удаления.

Создание игры началось с настройки главного окна, где будет проходить геймплей. В качестве заднего фона установлено заранее подготовленное изображение. В центре установлен Canvas с черным цветом в качестве заднего фона, это нужно, чтобы кубики были хорошо видны. Сверху и справа от Canvas расположены текстовые блоки(TextBlock) «Score:» и «Next», которые будут указывать на счет игрока и следующий набор кубиков.

В игровом процессе будет задействован ряд обьектов, а также необходимых для их правильного взаимодействия механизмов, для каждого из которых созданы классы. Классы также будут созданы для хранения некоторых данных. Полный список созданных классов представлен ниже.

Список классов:

**- MainWindow.xaml** и **MainWindow.xaml.cs**: Определяют главное окно игры, интерфейс и его логику.

- **Block.cs**: Абстрактный класс, представляющий общие свойства и методы для всех блоков (тетромино).

- **IBlock.cs, JBlock.cs, LBlock.cs, OBlock.cs, SBlock.cs, TBlock.cs**, **ZBlock.cs** - Классы, представляющие конкретные типы блоков (тетромино) с их уникальными свойствами.

- **BlockQueue.cs**: Класс, управляющий очередью следующих блоков.

- **GameGrid.cs**: Класс, представляющий игровую сетку и обеспечивающий ее функциональность.

- **GameState.cs**: Класс, отвечающий за текущее состояние игры, обработку действий игрока и логику игрового процесса.

**- Position.cs**: Класс, представляющий позицию в игровой сетке.

**Геймплей**

Геймплей разделен на несколько этапов: появление нового блока, управление блоком с клавиатуры (влево, вправо, вниз, вращение), удержание блока, падение блока, проверка заполненных линий и обновление счета. Игра завершается, когда блоки достигают верхней части игровой сетки.

**Используемые инструменты и алгоритмы**

Используется язык программирования C# с использованием технологии WPF для создания графического интерфейса. Для представления каждого типа блока (тетромино) используется отдельный класс, унаследованный от абстрактного класса Block.

Код класса Block:

using System.Collections.Generic;

using System;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Tetris

{

public abstract class Block

{

protected abstract Position[][] Tiles { get; }//двумерный массив позиций который содержит позиции плитки в четрех состояниях вращения

protected abstract Position StartOffset { get; }// начальное смещение где блок появляется в сетке

public abstract int Id { get; }// целочисленный идентификатор для различия блоков

private int rotationState;//текущее состояние вращения

private Position offset;//текущее смещение

public Block()//устонавливаем сещение равное начальному смещению

{

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

}

public IEnumerable<Position> TilePositions()// возвращает позиции сетки, занемаемые блоком с учетом текущего поворота и смещения

{

foreach (Position p in Tiles[rotationState])

{

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

}

}

public void RotateCW()// поворот блока на 90 градусов по часовой

{

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

}

public void RotateCCW()// поворот блока против часовой

{

if (rotationState == 0)

{

rotationState = Tiles.Length - 1;

}

else

{

rotationState--;

}

}

public void Move(int rows, int columns)// перемещение блока с заданым кол-вом строк и столбцов

{

offset.Row += rows;

offset.Column += columns;

}

public void Reset()// сбрасывание вращения и положения

{

rotationState = 0;

offset.Row = StartOffset.Row;

offset.Column = StartOffset.Column;

}

}

}

Игровая логика реализована в классе GameState, включая обработку пользовательского ввода, управление блоками, проверку столкновений и завершение игры.

Код класса GameState:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Tetris

{

public class GameState// класс состояния игры

{

private Block currentBlock;// свойство с резервным полем для текущего поля

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);

}

}

}

}

public GameGrid GameGrid { get; }

public BlockQueue BlockQueue { get; }

public bool GameOver { get; private set; }

public int Score { get; private set; }// очки в виде кол-ва очищеных строк

public Block HeldBlock { get; private set; }//свойство для задержания

public bool CanHold { get; private set; }

public GameState()

{

GameGrid = new GameGrid(22, 10);// инициализируем игровую сетку в 22 строки и 10 столбиков

BlockQueue = new BlockQueue();// инициализируем очередь блоков

CurrentBlock = BlockQueue.GetAndUpdate();// получение случайного блока для текущего свойства

CanHold = true;

}

private bool BlockFits()// проверка на нахождение текущего блока в допустимой позиции или нет

{

foreach (Position p in CurrentBlock.TilePositions())

{

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

{

return false;

}

}

return true;

}

public void HoldBlock()// метод удержания блока

{

if (!CanHold)

{

return;

}

if (HeldBlock == null)

{

HeldBlock = CurrentBlock;

CurrentBlock = BlockQueue.GetAndUpdate();

}

else

{

Block tmp = CurrentBlock;

CurrentBlock = HeldBlock;

HeldBlock = tmp;

}

CanHold = false;

}

public void RotateBlockCW()// поврот блока по часовой если это возможно

{

CurrentBlock.RotateCW();

if (!BlockFits())

{

CurrentBlock.RotateCCW();

}

}

public void RotateBlockCCW()// поворот против часовой елси это возможно

{

CurrentBlock.RotateCCW();

if (!BlockFits())

{

CurrentBlock.RotateCW();

}

}

public void MoveBlockLeft()// перемещение блока влево

{

CurrentBlock.Move(0, -1);

if (!BlockFits())

{

CurrentBlock.Move(0, 1);

}

}

public void MoveBlockRight()// перемещение блока вправо

{

CurrentBlock.Move(0, 1);

if (!BlockFits())

{

CurrentBlock.Move(0, -1);

}

}

private bool IsGameOver()// проверка закончилась ли игра

{

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

}

private void PlaceBlock()// вызываем когда блок не может быть перемещен вниз

{

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;

}

}

public void MoveBlockDown()// перемещение вниз

{

CurrentBlock.Move(1, 0);

if (!BlockFits())

{

CurrentBlock.Move(-1, 0);

PlaceBlock();

}

}

private int TileDropDistance(Position p)

{

int drop = 0;

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

{

drop++;

}

return drop;

}

public int BlockDropDistance()

{

int drop = GameGrid.Rows;

foreach (Position p in CurrentBlock.TilePositions())

{

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

}

return drop;

}

public void DropBlock()// метод отбрасывния блоков

{

CurrentBlock.Move(BlockDropDistance(), 0);

PlaceBlock();

}

}

}

Посредством класса BlockQueue осуществляется управление очередью блоков.

Код класса BlockQueue:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace Tetris

{

public class BlockQueue// класс очереди блоков, отвечающий за выбор следующего блока

{

private readonly Block[] blocks = new Block[]// массив содержащий 7 классов каждого блока

{

new IBlock(),

new JBlock(),

new LBlock(),

new OBlock(),

new SBlock(),

new TBlock(),

new ZBlock()

};

private readonly Random random = new Random();//случайный обьект

public Block NextBlock { get; private set; }//свойство для следующего блока в очереди

public BlockQueue()//инициализируем следующий блок случайным блоком

{

NextBlock = RandomBlock();

}

private Block RandomBlock()// возвращаем случайный блок

{

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

}

public Block GetAndUpdate()//возвращает следующий блок и обновляет свойство

{

Block block = NextBlock;

do

{

NextBlock = RandomBlock();

}

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

return block;

}

}

}

Код для всех задействованных тетромино примерно одинаков.

Код для IBlock выглядит вот так:

public class IBlock : Block// для плитки вида I

{

private readonly Position[][] tiles = new Position[][]// позиции плитки для четырех состояний вращения

{

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;//идентефикатор

protected override Position StartOffset => new Position(-1, 3);//начальное смещение

protected override Position[][] Tiles => tiles;//для свойства плитки мы возвращаем массив плиток выше

}

**Результаты работы в виде скриншотов**

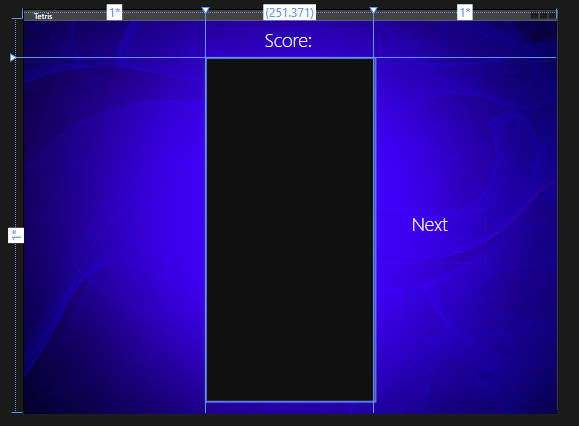


Рисунок 1 – игровое поле в проекте

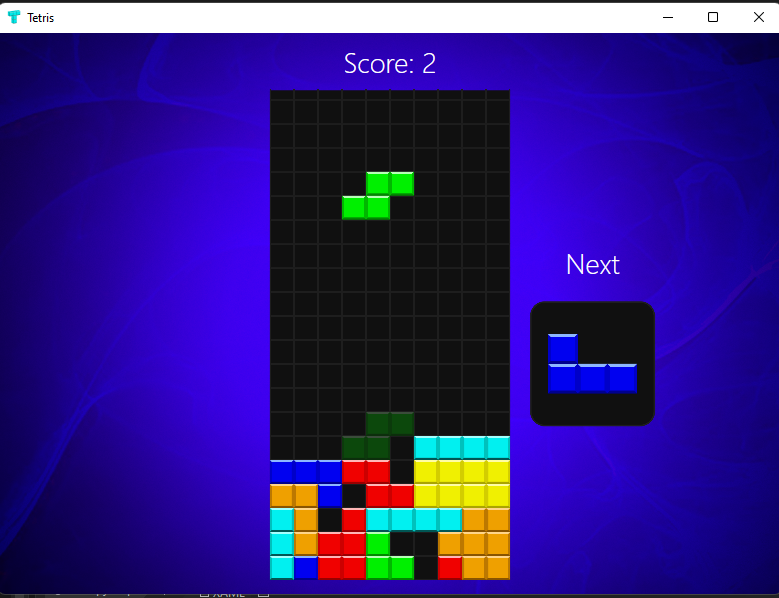


Рисунок 2 – запущенная игра

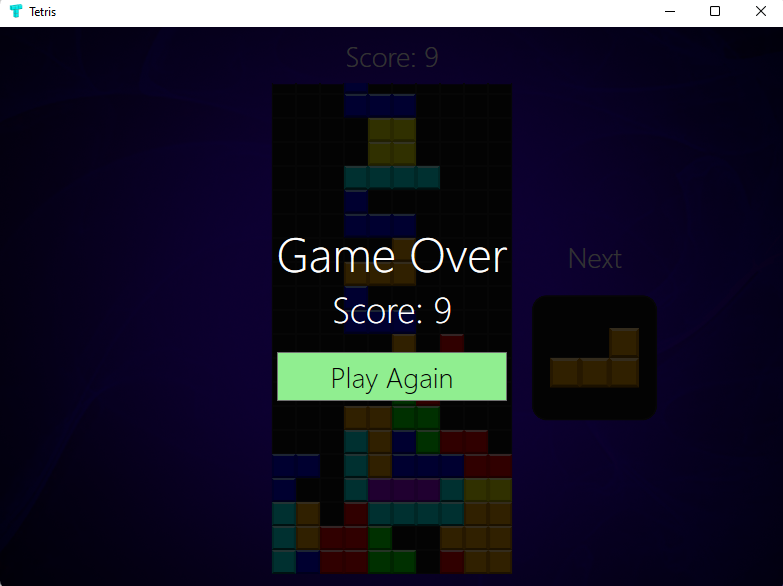


Рисунок 2 – экран поражения

**Вывод**

В ходе работы была успешно разработана игра "Тетрис" на языке программирования C с использованием WPF. Проект включает в себя основные элементы геймплея, архитектуру классов, обработку пользовательского ввода и графический интерфейс. Реализованный код обладает чистой структурой и легко поддается расширению.

Перспективы развития

Для дальнейшего развития проекта можно рассмотреть следующие направления:

1. Улучшение графического интерфейса: Добавление анимаций, звукового сопровождения и других элементов для улучшения визуального восприятия.

2. Расширение функциональности: Внедрение новых типов блоков, режимов игры или уровней сложности.

3. Многопользовательский режим: Разработка многопользовательского режима для соревновательного взаимодействия.

4. Оптимизация кода и рефакторинг: Повышение эффективности и читаемости кода, улучшение архитектуры.

**Заключение**

Разработка игры "Тетрис" позволила применить и углубить знания по языку C и технологии WPF. Полученный опыт в разработке игровых приложений может быть полезен для дальнейшего профессионального роста в области программирования.