Министерство цифрового развития, связи и массовых

Коммуникаций Российской Федерации

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ПМ и К

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»

Тема: «Игра “Тетрис” в графическом режиме»

Выполнил: студент группы ИП-012

Николаев Алексей Дмитриевич

Проверил: ассистент кафедры ПМ и К

Бублей Д.А.

Новосибирск – 2021

Оглавление

Постановка задачи

Реализовать игровое приложение “Тетрис” в графическом режиме с использованием технологий объектно-ориентированного программирования. Реализовать в игровом приложении создание фигур тетрамино, их вращение и быстрое падение при нажатии определённых клавиш, а также удаление линий при полном их заполнении.

При удалении линий увеличивать счёт игрока, который отвечает за увеличение скорости падения фигур.

Реализовать при запуске игры установку начального значения скорости падения фигур и уровня сложности, который при инициализации игры создаёт почти заполненные линии в соответствии со своим числовым значением.

Концом игры считать момент, в который при создании фигуры будет недостаточно свободных ячеек для её отрисовки.

Технологии ООП:

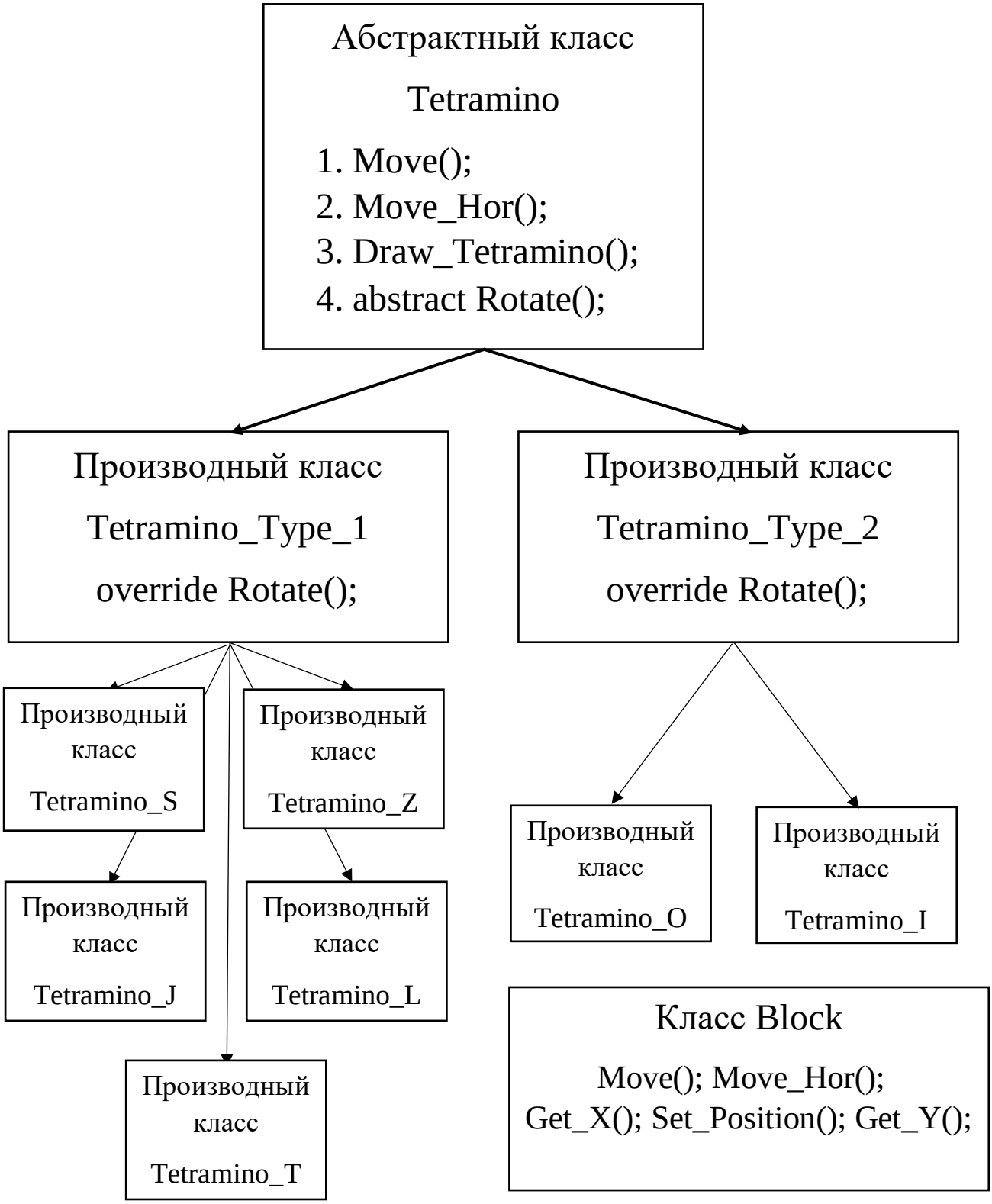
Необходимые технологии объектно-ориентированного программирования при создании игрового приложения:

- Инкапсуляция (все поля данных не доступны из внешних функций)
- Наследование (минимум 3 класса, один из которых абстрактный)
- Полиморфизм
- Конструкторы, Перегрузка конструкторов
- Списки инициализации

Также для реализации проекта будут дополнительно использованы виртуальные методы и параметры по умолчанию.

Структура классов

Схема наследования классов для реализации фигур тетрамино выглядит следующим образом:



В абстрактном классе для создания фигуры описан массив “указателей” на объекты класса Block, которые хранят в себе поля pos_x и pos_y, указывающие координаты блоков тетрамино, а также поля pos_x0 и pos_y0 для указания точки вращения фигур. (Все переменные в C#, которые хранят объект класса имеют ссылочный тип, так что они по умолчанию являются указателями).

Как видно из рисунка 1 в оригинальном Тетрисе точки вращения для ‘I’ (палочка) и ‘O’(квадрат) тетрамино отличаются от точек вращения всех остальных фигур. Для ‘I’ и ‘O’ тетрамино они находятся между блоками, тогда как для остальных фигур находятся внутри одного блока.

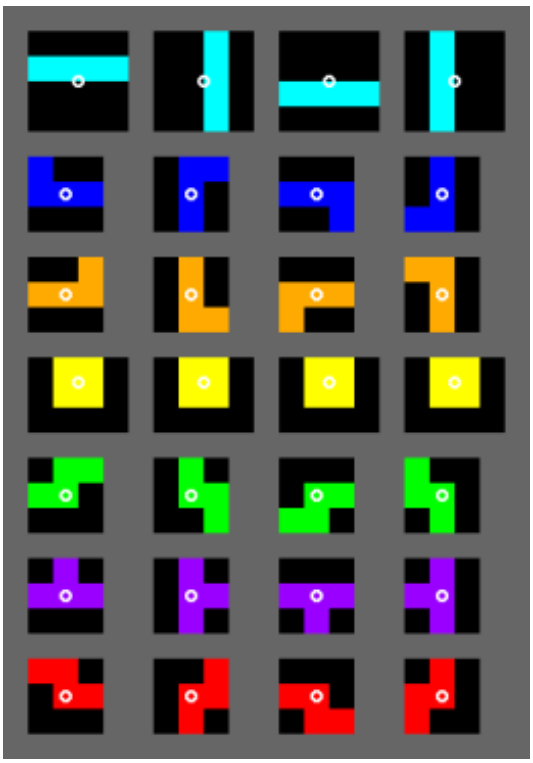


Рисунок 1. Иллюстрация вращения фигур тетрамино (“I”, “J”, ”L”, “O”, ”S”, ”T”, ”Z” сверху вниз)

Для того, чтобы не прописывать функцию поворота для каждой фигуры отдельно, я создал два производных класса Tetramino_Type_1 и Tetramino_Type_2, в которых перегрузил абстрактную функцию Rotate() родительского класса Tetramino, и уже от этих классов создал производные классы для каждой из фигур тетрамино со своими конструкторами.

Также для реализации проекта были созданы класс Field, производные классы `Text_Label` и `Info_Label` от `Label`, `Change_Panel` от `Panel`, `Form_Settings` и `MyForm` от `Form`. Но большинство из них выполняют вспомогательные функции так что о них поподробнее в следующем разделе.

Программная реализация

Перед тем как рассказывать, как программно-реализовано игровое приложение “Тетрис” стоит уточнить, что код программы написан на языке C# с использованием уже встроенных в него пространств имён `System`, `System.Windows.Forms` и `System.Drawings`.

Для реализации игрового приложения с использованием графического интерфейса я создал производный класс `MyForm`, который унаследовал все переменные и делегаты от класса `Form`.

Класс `MyForm` хранит в себе все глобальные данные, которые отвечают за игровой процесс. К ним относятся размер блоков тетрамино (`Size`), количество строк (`Rows`) и колонок (`Cols`) для игрового поля, кадры для отрисовки игрового поля (`frame`) и следующей фигуры (`bmp_Next`), начальные уровень сложность (`Level`) и скорость игры (`Speed`), игровой таймер для движения фигур через определённые интервалы времени (`game_timer`) и начальное значение самого интервала (`interval`).

Также в этом классе хранится указатель на объект класса `Field` (`game`), который отвечает за хранение информации об игровом поле и правильном движении фигур тетрамино.

В конструкторе класса `MyForm` в первую очередь создаётся объект класса `Form_Settings` (`settings`), которое также является производным от класса Form, с помощью которого игрок может настроить начальные значения уровня сложности и скорости игры, после чего в любой момент нажав на кнопку с подписью “Продолжить” начать игру. (Рисунок 2)

Для настройки значений уровня и скорости были созданы два объекта специального класса `Change_Panel` от класса `Panel`, в конструкторе которого создаётся объект класса `Text_Label`: `Label`, который отвечает за написание названия параметра, объект класса `Info_Label`: `Label`, который показывает численное значение этого параметра, а также две кнопки класса Button со стрелками влево, вправо, которые за уменьшение и увеличения значения параметра соответственно в пределах заданных значений максимума и минимума.

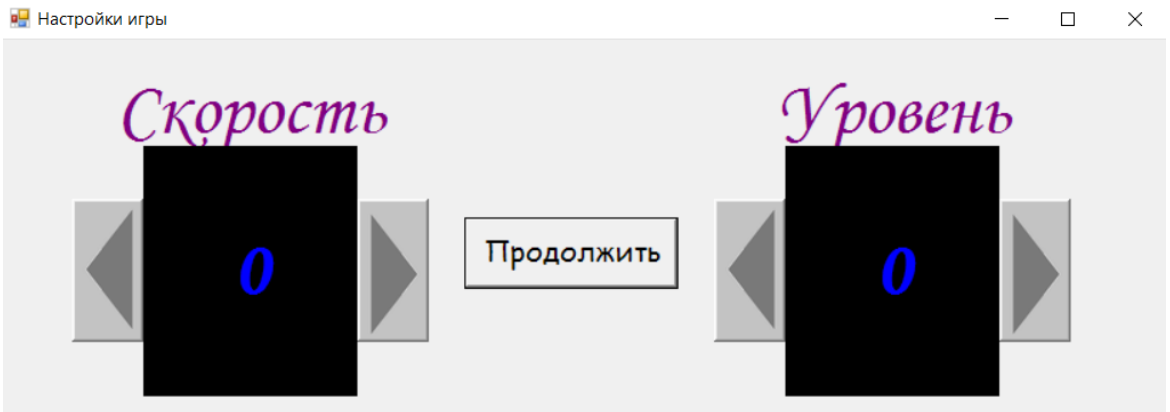


Рисунок 2. Меню настроек игры

После выхода из окна “Настройки игры”, в конструкторе класса `MyForm`, происходит создание объектов классов `Text_Label` и `Info_Label` для отображения внутриигровой информации, инициализация переменных и присваивание значений скорости и уровня из настроек и самое главное, создание объекта класса `Field` (`game`), при создании которого конструктору передаётся количество колонок, строк и уровней. (Рисунок 3)

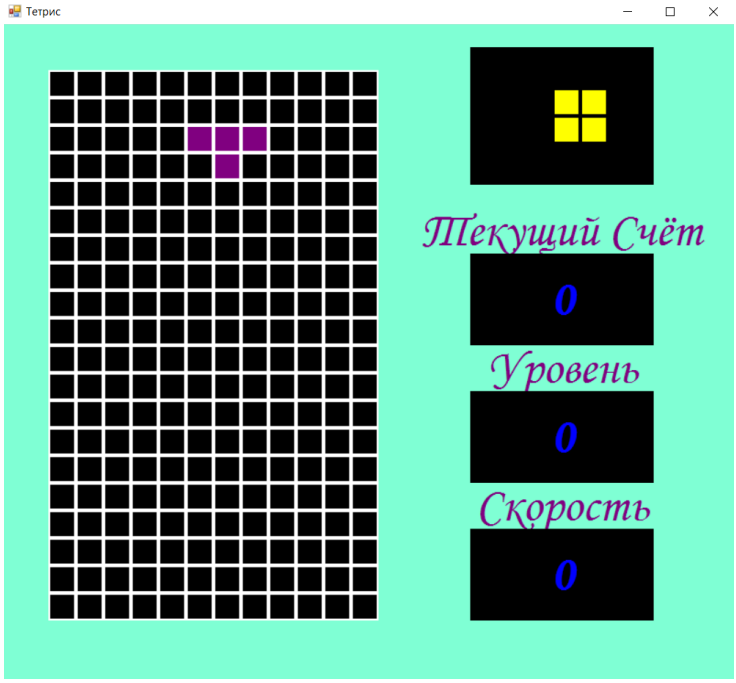


Рисунок 3. Окно, созданное объектом класса `MyForm`

Класс `Field` хранит в себе переменные ширины и высоты игрового поля в клетках, текущий счёт игры (`score`) в открытом поле класса, два указателя класса `Tetramino`, один из которых указывает на текущую фигуру (`Figure`), а другой указатель на следующую (`Figure_Next`).

Также класс хранит в себе указатели на двухмерные массивы объектов классов `bool` (`values`) (в C# каждый тип является классом) и `Color` (`Colors`). Массив values предназначен для хранения информации о том, есть ли на этой клетки блок упавшего тетрамино: `false` – ячейка пустая, `true` – ячейка не пустая. Массив `Colors` отвечает за графическое отображение всех ячеек поля: если цвет элемента массива чёрный, то ячейка пуста.

В конструкторе этого класса происходит инициализация переменных, а также заполнение двухмерных массивов с учётом уровня сложности с помощью метода `FillMatrix()`. Переменным указателям класса `Tetramino` передаются объект 1 из 7 классов, которые описывают отдельные фигуры.

Игровое поле рисуется методом `Draw_Frame()`, в котором в переданном как аргумент объекте класса Bitmap рисуются белые линии и все ячейки поля.

Метод `Change_Figure()` отвечает за остановку фигуры при падении, проверяет заполненность линий и в зависимости от количества заполненных линий увеличивает текущий счёт (100 очков за линию и за каждую удалённую линию до этого от одной фигуры). После этого создаётся новая фигура и проверяется что ни один из блоков тетрамино не лежит на заполненном блоке. В противном случае функция вернёт 1, что означает, что игра окончена.

Проверка выполняется в методе `Check_Line()` для линий, которые лежат между верхней и нижней границей упавшей фигуры, начиная сверху. Если проверка прошла успешно, вызывается метод `Down_Values()`, который опускает все значения и цвета из матрицы (двумерного массива) на одну строку вниз, пока не будет найдена полностью пустая строка.

Как было оговорено выше, абстрактный класс `Tetramino` хранит в себе массив указателей на объекты класса `Block`, цвет фигуры и координаты точки вращения для всех типов тетрамино.

Движение фигуры по горизонтали реализовано с помощью метода `Move_Hor()`, на вход которому ссылка на объект класса `Field` и направление движения по горизонтали `dir` (1 или -1). В этом методе просчитывается будущее положение каждого из блоков тетрамино и с помощью метода `CheckValue()` проверяется занятость этих новых координат для блоков. Если проверка прошла успешно, то каждый из блоков двигается в этом направлении с помощью метода `Move_Hor()` класса `Block`.

Движение фигуры по вертикали реализовано с помощью метода `Move()` аналогично как по горизонтали. Но если проверка прошла неудачно, функция вернёт 1, что будет означать то, что фигура остановилась.

В методе `Draw_Tetramino()` аналогично методу `Draw_Frame()` из класса `Field` передаётся объект класса Bitmap, в котором в зависимости от координат блока, рисуется фигура.

Метод `Rotate()`, который отвечает за поворот фигуры, в классе `Tetramino` указан как абстрактный, так что он был описан в производных классах `Tetramino_Type_1` и `Tetramino_Type_2`.

В классе `Tetramino_Type_1` метод `Rotate()` реализован следующим образом: сначала для каждого блока тетрамино вычисляется разность между его координатами и координатами точки вращения. После

этого новая координата по оси абсцисс приравнивается сумме соответствующей координаты центра и разности по оси ординат. Новая координата по оси ординат вычисляется аналогично, но вместо суммы стоит разность.

В классе `Tetramino_Type_2` метод `Rotate()` реализован аналогичным образом как в классе `Tetramino_Type_1`. Единственное чем эти методы отличаются, это тем что в классе `Tetramino_Type_2` новые координаты вычисляются только как суммы.

В конструкторах производных классов, унаследовавших поля и методы классов `Tetramino_Type_1` и `Tetramino_Type_2`, создаются четыре объекта класса `Block`, передавая их координаты относительно центра в конструкторе, и присваиваются личные цвет данных фигур.

Движение фигур на игровом поле реализовано с помощью игрового таймера (`game_timer` класса `Timer`), который через определённые промежутки времени сдвигает фигуру по вертикали, проверяет закончилась ли игра, отрисовывает следующую фигуру в отдельном окне и игровое поле.

Интервал для таймера по умолчанию равен 400 мс, но каждый уровень сложности уменьшает значение интервала на 20 мс. Уровень не может превышает значения “15” и он увеличивается на 1 за каждую 1000 очков в счёте. При изменении скорости в меню настроек изменится начальное значение скорости, а прибавка от количества очков в счёте останется неизменной.

За движение фигур влево отвечают клавиши A и стрелка влево, за движение вправо – D и стрелка вправо, за поворот фигуры – Q, W и стрелка вверх, за быстрое опускание фигуры – E,S и стрелка вниз (при нажатии этой клавиши интервал таймера уменьшится до 50 мс, а при отпускании увеличится до прошлого значения), за перезапуск игры – R, а за паузу – P. При нажатии клавиши Escape приложение закроется.

Результаты работы

Рисунок 4. Меню настроек игры с ненулевыми значениями скорости и уровня сложности

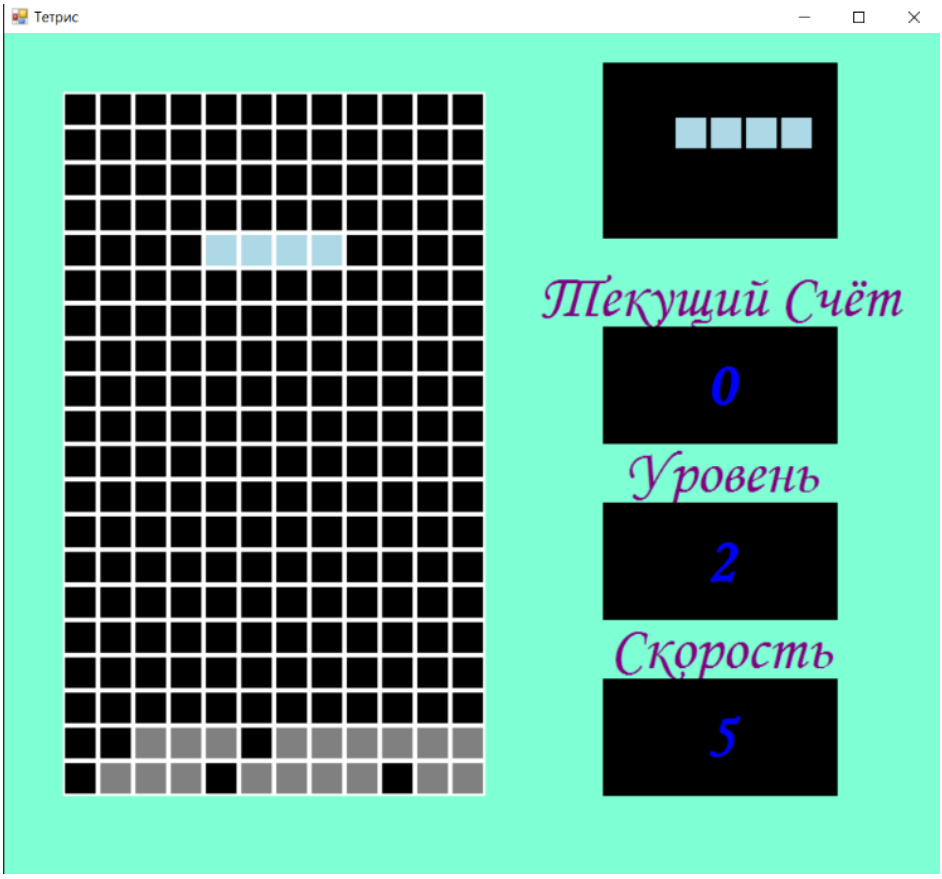


Рисунок 5. Начало игры с не нулевыми скоростью и уровнем

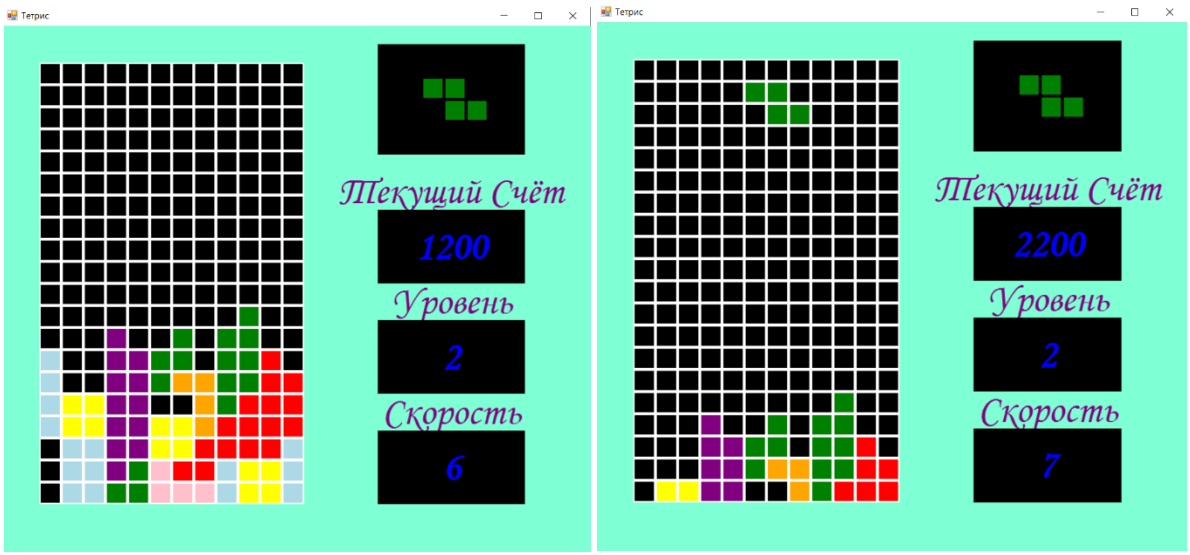


Рисунок 6. Экран до и после заполнения 4 линий

(+1000 очков => +1 скорость)

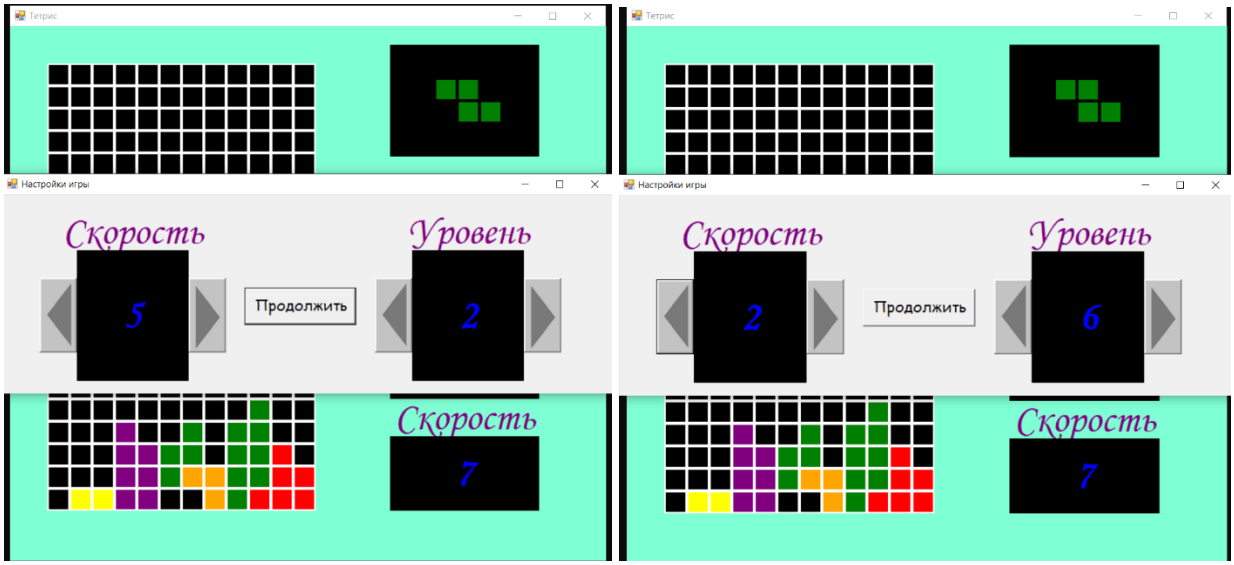


Рисунок 7. Изменение скорости и уровня в меню паузы

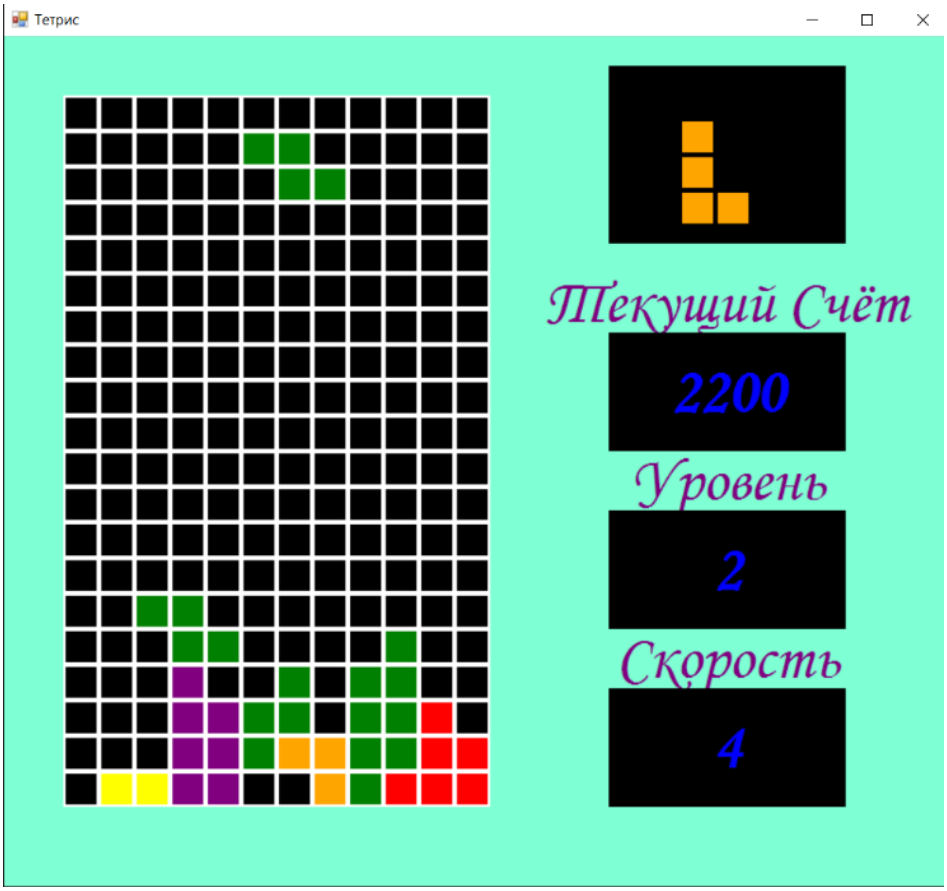


Рисунок 8. Игра после снятия паузы (Скорость была 5+2, стала 2+2)

Рисунок 9. Конец игры

Рисунок 10. Начало игры после перезапуска

Заключение

В данном курсовом проекте было реализовано игровое приложение «Тетрис» с использованием технологий объектно-ориентированного программирования, знания о которых были получены на лекциях по III семестре. Были реализованы создание фигур тетрамино, их вращение и быстрое падение при нажатии определённых клавиш, а также удаление линий при полном их заполнении.

Для создания были использованы принципы:

- Инкапсуляции, которое проявляется в закрытом доступе ко многим полям классов;
- Наследования, которое использовалось для создания разных типов фигур тетрамино;
- Полиморфизма, которое проявляется в использовании абстрактных (виртуальных) методов для вращения фигур.

Также были использованы конструкторы и их перегрузки, списки инициализации и параметры по умолчанию.

Используемые источники

Из источников я могу выделить литературу, по которой я самостоятельно изучал программирование на языке C#:

- Васильев. А.Н. Программирование на C# для начинающих. Основные сведения [Электронный ресурс] //ЛитРес. – URL: https://www.litres.ru/aleksey-nikolaevich-/programmirovanie-na-c-dlya-nachinauschih-osn-34338191/
- Васильев. А.Н. Программирование на C# для начинающих. Особенности языка [Электронный ресурс] //ЛитРес. – URL: https://www.litres.ru/aleksey-nikolaevich-/programmirovanie-na-s-dlya-nachinauschih-oso-39451764/

Приложение. Листинг

1) Программная реализация классов для создания тетрамино:

```
class Block
{
    int pos_x;
    int pos_y;
    public Block(int x=0,int y=0)
    {
        pos_x=x;
        pos_y=y;
    }
    public void Move()
    {
        pos_y++;
    }
    public void Move_Hor(int dir)
    {
        pos_x+=dir;
    }
    public void Set_Position(int x, int y)
    {
        pos_x=x;
        pos_y=y;
    }
    public int Get_X()
    {
        return pos_x;
    }
    public int Get_Y()
    {
        return pos_y;
    }
}
```

```
abstract class Tetramino
{
    public Block[] blocks;
    protected int pos_x0;
    protected int pos_y0;
    protected Color Block_Color;
    public Tetramino(int pos_x0, int pos_y0)
    {
        blocks=new Block[4];
        this.pos_x0=pos_x0;
        this.pos_y0=pos_y0;
    }
    public Color Get_Color()
    {
        return Block_Color;
    }
    public int Move(Field game)
    {
        for(int i=0;i<blocks.Length;i++)
        {
            if(game.Check-
Value(blocks[i].Get_X(),blocks[i].Get_Y()+1)) return 1;
        }
        for(int i=0;i<blocks.Length;i++)
        {
            blocks[i].Move();
        }
        pos_y0++;
        return 0;
    }

    public void Move_Hor(Field game, int dir)
    {
        for(int i=0;i<blocks.Length;i++)
        {
            if(game.Check-
Value(blocks[i].Get_X()+dir,blocks[i].Get_Y())) return;
        }
        for(int i=0;i<blocks.Length;i++)
        {
            blocks[i].Move_Hor(dir);
```

```
        }
        pos_x0+=dir;
    }

    public void Draw_Tetramino(Graphics g, SolidBrush
br, int Size)
    {
        br.Color=Block_Color;
        for(int i=0;i<blocks.Length;i++)
        {
            g.FillRectangle(br,
blocks[i].Get_X()*Size+2,blocks[i].Get_Y()*Size+2,Size-
4,Size-4);
        }
    }
    abstract public void Rotate(Field game);
}

class Tetramino_Type_1: Tetramino
{
    public Tetramino_Type_1(int pos_x0, int pos_y0):
base(pos_x0,pos_y0){}
    public override void Rotate(Field game)
    {
        int dx, dy;
        int[] x_new, y_new;
        x_new=new int[4];
        y_new=new int[4];
        for(int i=0;i<4;i++)
        {
            dx=blocks[i].Get_X()-pos_x0;
            dy=blocks[i].Get_Y()-pos_y0;
            x_new[i]=pos_x0+dy;
            y_new[i]=pos_y0-dx;
            if(game.CheckValue(x_new[i],y_new[i])) re-
turn;
        }
        for(int i=0;i<4;i++)
        {
            blocks[i].Set_Position(x_new[i],y_new[i]);
        }
    }
```

```
}

class Tetramino_Type_2: Tetramino
{
    public Tetramino_Type_2(int pos_x0, int pos_y0):
base(pos_x0,pos_y0){}

    public override void Rotate(Field game)
    {
        int dx, dy;
        int[] x_new, y_new;
        x_new=new int[4];
        y_new=new int[4];
        for(int i=0;i<4;i++)
        {
            dx=blocks[i].Get_X()-pos_x0;
            dy=blocks[i].Get_Y()-pos_y0;
            x_new[i]=pos_x0+dy;
            y_new[i]=pos_y0+dx;
            if(game.CheckValue(x_new[i],y_new[i])) re-
turn;
        }
        for(int i=0;i<4;i++)
        {
            blocks[i].Set_Position(x_new[i],y_new[i]);
        }
    }
}
```

2) Основные поля и методы класса Field:

```
class Field
{
    public int score;
    int width;
    int height;
    bool[,] values;
    Color[,] Colors;
```

```
public Tetramino Figure;
public Tetramino Figure_Next;
public Field(int Cols, int Rows, int Level)
{
    score=0;
    width=Cols;
    height=Rows;
    values=new bool[width,height];
    Colors=new Color[width,height];
    Fill_Matrix(Level);
    Figure=Create_Tetramino();
    Figure_Next=Create_Tetramino();
}
public bool CheckValue(int x, int y)
{
    if(x<0 || x>=width) return true;
    if(y<0 ||  y>=height) return true;
    if(values[x,y]) return true;
    return false;
}
public int Change_Figure()
{
    Color temp=Figure.Get_Color();
    int i,j;
    for(int k=0;k<Figure.blocks.Length;k++)
    {
        i=Figure.blocks[k].Get_X();
        j=Figure.blocks[k].Get_Y();
        Colors[i,j]=temp;
```

```
            values[i,j]=true;
        }
        Check_Line();
        Figure=Figure_Next;
        Figure_Next=Create_Tetramino();
        for(int k=0;k<Figure.blocks.Length;k++)
        {
            i=Figure.blocks[k].Get_X();
            j=Figure.blocks[k].Get_Y();
            if(CheckValue(i,j)) return 1;
        }
        return 0;
    }

    void Down_Values(int y_line)
    {
        bool empty;
        for(int j=y_line;j>0;j--)
        {
            empty=true;
            for(int i=0;i<width;i++)
            {
                if(values[i,j-1]) empty=false;
                values[i,j]=values[i,j-1];
                Colors[i,j]=Colors[i,j-1];
            }
            if(empty) break;
        }
    }
```

```
    void Check_Line()
    {
        int factor=1;
        int y_min=height, y_max=0;
        int y_temp;
        for(int k=0;k<Figure.blocks.Length;k++)
        {
            y_temp=Figure.blocks[k].Get_Y();
            if(y_temp>y_max) y_max=y_temp;
            if(y_temp<y_min) y_min=y_temp;
        }
        bool full;
        for(int j=y_min;j<=y_max;j++)
        {
            full=true;
            for(int i=0;i<width;i++)
            {
                if(values[i,j]==false) full=false;
            }
            if(full)
            {
                Down_Values(j);
                score+=100*factor;
                factor++;
            }
        }
    }
}
```