Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

## КУРСОВАЯ РАБОТА

### по дисциплине «Объектно-ориентированное программирование»

Тема: «Игра тетрис»

Выполнили: Студент 2-го курса,

группы ИВ-122

Гердележов Даниил Дмитриевич

Проверил преподаватель: Бублей Дмитрий Александрович

**Содержание:**

[1 Постановка задачи 3](#_Toc121418245)

[2 Структура классов 4](#_Toc121418246)

[3 Программная реализация 11](#_Toc121418247)

[4 Результаты работы 12](#_Toc121418248)

[5 Заключение 13](#_Toc121418251)

[6 Используемые источники 13](#_Toc121418252)

[7 Приложение. Листинг 14](#_Toc121418253)

[TetrDlg.h 14](#_Toc121418254)

[TetrDlg.cpp 15](#_Toc121418255)

[Сlass.h 20](#_Toc121418256)

[Сlass.cpp 23](#_Toc121418257)

[Tetr.h 35](#_Toc121418258)

[Tetr.cpp 35](#_Toc121418259)

[Resource.h 37](#_Toc121418260)

[Stdafx.h 37](#_Toc121418261)

[Targetver.h 38](#_Toc121418262)

[Stdafx.cpp 38](#_Toc121418263)

## 1 Постановка задачи

Необходимо написать программу, реализующую игру «Тетрис», используя объектно-ориентированный подход. Иерархия классов должна включать минимум три класса. Язык и среда программирования – С++.

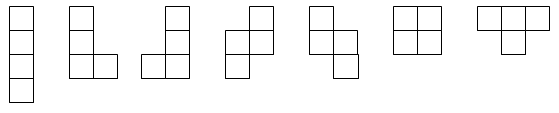
Фигуры, участвующие в игре, представлены на рисунке 1.

Рисунок 1 – Фигуры игры «Тетрис»

Игровое поле состоит из клеток (ширина – 10 клеток, высота – 20 клеток). Фигуры появляются в верхней части игрового поля. Через определенное время они смещаются вниз. Движение вниз прекращается, если фигура достигает нижней части игрового поля или если дальнейшее движение вниз невозможно, т.к. там находится другая фигура. После этого появляется следующая фигура. Пользователь может перемещать фигуры в стороны и вниз, поворачивать. Если ряд клеток заполнен полностью фигурами, то он удаляется и ряды, находящиеся выше данного ряда, сдвигаются вниз. Игра завершается, если следующую фигуру невозможно разместить в верхней части игрового поля.

При падении фигуры может удаляться от одной до четырех линий одновременно. Данные о количестве очков, начисляемом при удалении линий, приведены в таблице 1.

Таблица 1 Начисление очков при удалении линий

|  |  |
| --- | --- |
| Количество линий | Количество очков |
| 1 | 100 |
| 2 | 300 |
| 3 | 600 |
| 4 | 1000 |

Управление игрой можно осуществлять следующими клавишами:

- стрелка влево – перемещение фигуры влево;

- стрелка вправо – перемещение фигуры вправо;

- стрелка вниз – перемещение фигуры вниз;

- стрелка вверх – поворот фигуры.

## 2 Структура классов

Для решения поставленной задачи необходимо реализовать следующие классы:

- CTetrApp – приложение;

- СTetrDlg – диалог, в котором собственно выполняется игра;

- CRandom – генератор случайных чисел;

- CFigura – фигура;

- CKlet – клетка;

- CFgWindow – элемент, отображающий множество фигур;

- СNextwindow – элемент, отображающий следующую фигуру;

- CGameWindow – элемент, отображающий игровое поле;

- CColorText – окно, отображающее цветную надпись.

Иерархия классов представлена на рисунке 2.

CRandom

CWinApp

CTetrApp

CObject

CFigura

CKlet

CDialog

CAboutDlg

CTetrDlg

CWnd

CFgWindow

CColorText

CNextWindow

CGameWindow

Рисунок 2 – Иерархия классов

Класс CRandom – генератор псевдослучайных чисел. Генератор случайных чисел необходим при выборе следующей фигуры.

Методы класса CRandom представлены в таблице 2.

Таблица 2 – Методы класса CRandom

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Конструктор CRandom() | public | Конструктор, выполняющий инициализацию генератора случайных чисел |
| Функция random(int) | public | Генерация случайного числа 0 <= x <max |

Класс CFigura – фигура, состоящая из клеток. Базовым классом является класс CObject, т.к. класс CFigura используется в шаблоне CArray для создания динамического массива фигур. Класс включает в себя класс CKlet – клетки, из которых состоит фигура. Поля класса CFigura представлены в таблице 3, методы – в таблице 4.

Координаты клеток фигуры задаются относительно координаты фигуры. Таким образом для перемещения всей фигуры необходимо изменить координаты фигуры, а координаты клеток остаются без изменения.

Таблица 3 – Поля класса CFigura

|  |  |  |
| --- | --- | --- |
| Поле | Свойства | Описание |
| tcount | protected, static | Количество различных видов фигур |
| tcolor | protected, static | Массив значений цветов каждого вида фигур |
| Tdata | protected, static | Массив задает координаты клеток для создания каждого вида фигур |
| ifg,jfg | protected | Координаты фигуры на игровом поле. Координаты клеток, из которых состоит фигура, задаются относительно данных координат. |
| kl | protected | Динамический массив клеток. Клетки представляются объектами CKlet. Массив строится по шаблону класса CArray <CKlet\*,CKlet\*> |

Таблица 4 – Методы класса CFigura

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Конструктор  CFigura (int, int) | public | Конструктор, задающий по указанным координатам фигуру. Вид и поворот фигуры определяются случайным образом |
| Конструктор  CFigura (CFigura\*) | public | Конструктор создает фигуру с теми же свойствами, что и заданная (копирует фигуру) |
| Деструктор  ~CFigura () | public, virtual | Деструктор, удаляет все клетки, из которых состоит фигура |
| GetKletSize () | public, static | Получение размера клетки |
| Init (int, int) | public | Формирование заданного вида фигуры и заданного положения (заданное положение достигается за счет применения поворота) |
| InitRandom () | public | Формирование случайной фигуры |
| Paint (CDC\*) | public | Отображение фигуры. Отображение фигуры сводится к отображению клеток, из которых она состоит |
| Move (int,int) | public | Перемещение фигуры (изменение координат фигуры). Координаты клеток, входящих в фигуру, не изменяется |
| TestRect (int,int) | public | Проверка попадания фигуры в пределы игрового поля |
| FindKlet (int,int) | public | Поиск в фигуре клетки, которая располагается по заданным координатам |
| Оператор  && (CFigura&) | public | Оператор определяет наличие пересечения фигуры с заданной фигурой |
| Оператор  = (CFigura&) | public | Фигура копирует свойства заданной фигуры, включая количество и свойства клеток |
| Rotate () | public | Поворот фигуры на 90 градусов против часовой стрелки. Фигура изменяет координаты входящих в нее клеток. Координаты самой фигуры не изменяются |
| DeleteLine (int) | public | Удаление заданной линии. Клетки, находящиеся на заданной линии, удаляются, а находящиеся выше заданной линии сдвигаются вниз |
| Empty () | public | Проверка наличия в фигуре клеток. Используется после удаления клеток |

Класс CKlet – клетки, из которых составляются фигуры. Базовым классом является класс CObject, т.к. класс CKlet используется в шаблоне класса CArray для создания динамического массива клеток. Поля класса CKlet представлены в таблице 5, методы – в таблице 6.

Таблица 5 – Поля класса CKlet

|  |  |  |
| --- | --- | --- |
| Поле | Свойства | Описание |
| ik, jk | protected | Координаты клетки относительно фигуры |
| cl | protected | Цвет клетки |
| size | public, static | Размер клетки |

Таблица 6 – Методы класса CKlet

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Конструктор  CKlet(int,int,COLORREF) | public | Конструктор создает клетку заданного цвета, с заданными координатами |
| Конструктор  CKlet(CKlet\*) | public | Конструктор создает клетку с теми же свойствами, что и заданная (копирует клетку) |
| Paint(CDC\*,int,int) | public | Отображение клетки по заданным базовым координатам (по заданным координатам фигуры) |
| TestRect(int,int,int,int) | public | Проверка попадания клетки в поле (по заданным размерам поля и координатам фигуры) |
| Include(int,int) | public | Проверка на соответствие координат клетки заданным координатам |
| GetCoord(int&,int&) | public | Получение координат клетки |
| SetCoord(int,int) | public | Установка координат клетки |
| Rotate() | public | Поворот клетки |

Класс CFgWindow – элемент, отображающий множество фигур. Базовым классом является класс CWnd. Поля класса CFgWindow представлены в таблице 7, методы – в таблице 8.

Данный класс, в свою очередь, является базовым для CNextWindow и CGameWindow, и в нем определяются два абстрактных виртуальных метода: GetSize и PaintBackground. Необходимость применения методов обусловлена тем, что данные методы необходимо реализовать по-разному в классах-потомках, но их использование нужно включить в базовый класс.

Таблица 7 – Поля класса CFgWindow

|  |  |  |
| --- | --- | --- |
| Поле | Свойства | Описание |
| fg | protected | Динамический массив фигур. Фигуры представляются объектами CFigura. Массив строится по шаблону класса CArray <CFigurat\*,CFigura\*> |

Таблица 8 – Методы класса CFgWindow

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Деструктор  ~CFgWindow() | public, virtual | Деструктор, удаляет все фигуры из массива фигур |
| Create(CWnd\*,int,  int,UINT) | public, virtual | Cоздание окна |
| GetSize(int&,int&) | public, virtual,  абстрактный | Получение необходимого размера окна |
| Clear() | public | Очистка окна (т.е. удаление всех фигур с игрового поля) |
| Paint(CDC\*) | protected, virtual | Отображение окна. Отображается фон с помощью функции PaintBackground, а затем все фигуры. |
| PaintBackground(CDC\*) | protected, virtual,  абстрактный | Отображение фона окна |
| OnPaint() | protected | Обработка события WM\_PAINT (Вызов функции отображения окна) |

Класс CNextWindow – элемент, отображающий следующую фигуру. Базовым классом является класс CFgWindow. Методы класса CNextWindow представлены в таблице 9.

Таблица 9 – Методы класса CNextWindow

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Next() | public | Выбор новой фигуры. При этом предыдущая фигура удаляется |
| GetNext() | public | Получение выбранной фигуры |
| GetSize(int&,int&) | protected, virtual | Получение необходимого размера окна. В окне должна поместится одна фигуры |
| PaintBackground(CDC\*) | protected, virtual | Отображение фона. Цвет фон совпадает с цветом диалогового окна |

Класс CGameWindow – элемент, отображающий игровое поле и положение всех фигур. Базовым классом является класс CFgWindow. Поля класса CGameWindow представлены в таблице 10, методы – в таблице 11.

При отображении данного окна необходимо применить двойную буферизации. Если этого не сделать, то изображение при перерисовывании будет мерцать, так как каждый раз необходимо перерисовывать фон, а затем и все фигуры. Альтернативным подходом является отслеживание изменения цвета отдельных клеток и перерисовки только изменяющихся элементов игрового поля. Но данных подход в нашем случае реализовать сложнее. Легче формировать изображение в битовой матрице, а затем уже готовое выводить на экран.

Таблица 10 – Поля класса CGameWindow

|  |  |  |
| --- | --- | --- |
| Поле | Свойства | Описание |
| nheight | protected | Высота поля (количество клеток) |
| nwidth | protected | Ширина поля (количество клеток) |
| pdcDisplayMemory | private | Контекст для двойной буферизации |
| pBitmap | private | Битовая матрица для двойной буферизации отображения |

Таблица 11 – Методы класса CGameWindow

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Конструктор  Create(CWnd\*,int,int, int,int,int) | public, virtual | Конструктор, инициализирует поля класса |
| NewFg(CFigura \* f) | public | Добавление новой фигуры с проверкой возможности добавления |
| Down() | public | Перемещение вниз с проверкой возможности перемещения |
| Left() | public | Перемещение влево с проверкой возможности перемещения |
| Right() | public | Перемещение вправо с проверкой возможности перемещения |
| End() | public | Проверка условия конца игры (когда в верхнюю часть игрового поля нельзя добавить новую фигуру) |
| Rotate() | public | Поворот фигуры на 90 градусов против часовой стрелки |
| Move(int,int) | public | Перемещение фигуры в заданном направлении с проверкой возможности перемещения |
| TestMove(int,int) | public | Проверка возможности перемещения фигуры |
| FindLine(int) | public | Проверка заполнения линии игрового поля клетками фигуры |
| DeleteLine(int) | public | Удаление линии |
| Lines() | public | Подсчет и удаление линий |
| PaintBackground(CDC\*) | protected, virtual | Отображение фона игрового поля |
| GetSize(int&,int&) | protected, virtual | Получение необходимого размера. Игровое поле должно вместить 10 клеток в ширину и 20 клеток в высоту. Кроме того поле обведено линией. |
| TestMove(CFigura\*) | protected | Проверка возможности перемещения текущей фигуры в заданное положение. Для проверки используется копия фигуры. |
| OnPaint() | protected | Обработка сообщения WM\_PAINT. Отображение игрового поля с использованием двойной буферизации, которая исключает мерцание при перерисовке |

Класс CColorText – элемент, отображающий цветную надпись. Базовым классом является класс CWnd. Поля класса CColorText представлены в таблице 12, методы – в таблице 13.

Таблица 12 – Поля класса CColorText

|  |  |  |
| --- | --- | --- |
| Поле | Свойства | Описание |
| fontname | protected | Название шрифта |
| fontsize | protected | Размер шрифта |
| color | protected | Цвет надписи |

Таблица 13 – Методы класса CColorText

|  |  |  |
| --- | --- | --- |
| Метод | Свойства | Описание |
| Конструктор  CColorText() | public | Начальная инициализация данных |
| Create(const RECT&,CWnd\*,UINT) | public, virtual | Создание окна с заданными координатами |
| Create(char\*,const RECT&,CWnd\*,UINT, char\*,int,COLORREF) | public, virtual | Создание окна с заданными координатами, текстом и цветом |
| SetNumber(int) | public | отображение числа |
| OnPaint() | protected | Обработка сообщения WM\_PAINT. Отображение цветной надписи. |

## 3 Программная реализация

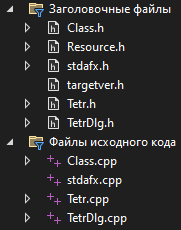


Рисунок 3 – Структура проекта.

Tetr.h

Это основной файл заголовка для приложения. В него включены другие определенные для проекта заголовки (в том числе Resource.h) и объявлен класс приложения CTetrApp.

Tetr.cpp

Это основной исходный файл приложения, содержащий класс приложения CTetrApp.

Мастер приложений создает один класс диалоговых окон:

TetrDlg.h, TetrDlg.cpp – диалоговое окно

Эти файлы содержат класс CTetrDlg. Этот класс определяет поведение основного диалогового окна приложения. Шаблон диалогового окна находится в файле Tetr.rc, который можно редактировать в Microsoft Visual C++.

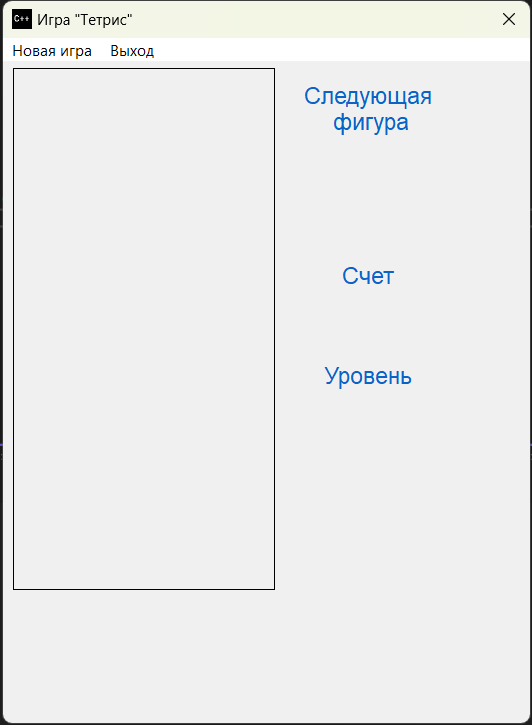
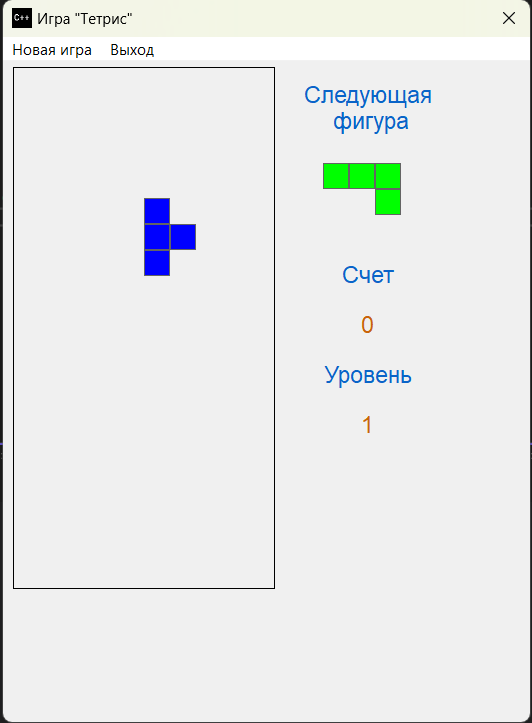
StdAfx.h, StdAfx.cpp

Эти файлы используются для построения файла предкомпилированного заголовка (PCH) с именем Tetr.pch и файла предкомпилированных типов с именем StdAfx.obj.

Resource.h

Это стандартный файл заголовка, определяющий новые идентификаторы ресурсов. Microsoft Visual C++ прочитывает и обновляет этот файл.

## 4 Результаты работы



## 

Рисунок 4 – Состояние программы после запуска.

Рисунок 5 – Состояние программы во время игры.

## 

Рисунок 6 – Состояние программы после проигрыша.

## 5 Заключение

В результате выполнения работы была создана программа, реализующая игру «Тетрис». Программа была протестирована и удовлетворяет всем требованиям, перечисленным при анализе задачи. При написании программы использовались язык и среда программирования – С++.

## 6 Используемые источники

1. Приложения MFC для рабочего стола (<https://learn.microsoft.com/ru-ru/cpp/mfc/mfc-desktop-applications?view=msvc-170>)
2. Руководство по языку программирования C++ (<https://metanit.com/cpp/tutorial/>)
3. *Ситняковская Е.И.* Краткая теория по курсу «Объектно-ориентированное программирование» [Электронный ресурс] : учеб.-метод. пособие / Сиб. гос. ун-т телекоммуникаций и информатики. Новосибирск : СибГУТИ, 2021
4. *Бьерн Страуструп* Язык программирования С++ (<http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>)

## 7 Приложение. Листинг

### TetrDlg.h

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | // TetrDlg.h : файл заголовка  //  #pragma once  #include "Class.h"  // диалоговое окно CTetrDlg  **class** **CTetrDlg** : **public** CDialogEx  {  // Создание  **public:**  CTetrDlg(CWnd\* pParent = NULL); // стандартный конструктор  // Данные диалогового окна  **enum**  {  IDD = IDD\_TETR\_DIALOG  };  **protected:**  **virtual** **void** **DoDataExchange**(CDataExchange\* pDX); // поддержка DDX/DDV  **public:**  **void** **Timer**(); // установка таймера (по уровню)  // Реализация  **protected:**  HICON m\_hIcon;  **int** level; // уровень  **int** lines; // количество линий  **int** score; // счет  **int** end; // конец игры  // элементы "цветные надписи"  CColorText m\_TextNext, m\_TextScore, m\_TextLever, m\_Level, m\_Score;  CGameWindow m\_GameWind; // игровое поле  CNextWindow m\_NextWind; // следующая фигура  **virtual** BOOL **PreTranslateMessage**(MSG\* pMsg);  // Созданные функции схемы сообщений  **virtual** BOOL **OnInitDialog**();  afx\_msg **void** **OnSysCommand**(UINT nID, LPARAM lParam);  afx\_msg **void** **OnPaint**();  afx\_msg HCURSOR **OnQueryDragIcon**();  DECLARE\_MESSAGE\_MAP()  **public:**  afx\_msg **void** OnNewGame();  afx\_msg **void** **OnTimer**(UINT\_PTR nIDEvent);  afx\_msg **void** **OnKeyDown**(UINT nChar, UINT nRepCnt, UINT nFlags);  }; |

### TetrDlg.cpp

// TetrDlg.cpp : файл реализации

//

#include "stdafx.h"

#include "Tetr.h"

#include "TetrDlg.h"

#include "afxdialogex.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// Диалоговое окно CAboutDlg используется для описания сведений о приложении

**class** **CAboutDlg** : **public** CDialogEx

{

**public:**

CAboutDlg();

// Данные диалогового окна

**enum**

{

IDD = IDD\_ABOUTBOX

};

**protected:**

**virtual** **void** **DoDataExchange**(CDataExchange\* pDX); // поддержка DDX/DDV

// Реализация

**protected:**

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)

{

}

**void** CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// диалоговое окно CTetrDlg

CTetrDlg::CTetrDlg(CWnd\* pParent /\*=NULL\*/)

: CDialogEx(CTetrDlg::IDD, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

**void** CTetrDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CTetrDlg, CDialogEx)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_COMMAND(ID\_NEWGAME, &CTetrDlg::OnNewGame)

ON\_WM\_TIMER()

ON\_WM\_KEYDOWN()

END\_MESSAGE\_MAP()

// обработчики сообщений CTetrDlg

BOOL CTetrDlg::OnInitDialog()

{

CDialogEx::OnInitDialog();

// Добавление пункта "О программе..." в системное меню.

// IDM\_ABOUTBOX должен быть в пределах системной команды.

ASSERT((IDM\_ABOUTBOX & **0xFFF0**) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < **0xF000**);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

**if** (pSysMenu != NULL)

{

BOOL bNameValid;

CString strAboutMenu;

bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);

ASSERT(bNameValid);

**if** (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

// Задает значок для этого диалогового окна. Среда делает это автоматически,

// если главное окно приложения не является диалоговым

SetIcon(m\_hIcon, TRUE); // Крупный значок

SetIcon(m\_hIcon, FALSE); // Мелкий значок

// элемент для отображение игрового поля

m\_GameWind.Create(**this**, **10**, **5**, IDC\_GAMEWIND, **10**, **20**);

// элемент для отображения следующей фигуры

m\_NextWind.Create(**this**, **320**, **75**, IDC\_NEXTWIND);

// надписи:

m\_TextNext.Create("Следующая**\n** фигура", CRect(**280**, **20**, **450**, **74**), **this**, IDC\_STATIC,

"Arial", **14**, RGB(**10**, **100**, **200**));

m\_TextScore.Create("Счет", CRect(**280**, **200**, **450**, **230**), **this**, IDC\_STATIC,

"Arial", **14**, RGB(**10**, **100**, **200**));

// счет

m\_Score.Create("", CRect(**280**, **250**, **450**, **280**), **this**, IDC\_STATIC,

"Arial", **14**, RGB(**200**, **100**, **10**));

m\_TextLever.Create("Уровень", CRect(**280**, **300**, **450**, **330**), **this**, IDC\_STATIC,

"Arial", **14**, RGB(**10**, **100**, **200**));

// уровень

m\_Level.Create("", CRect(**280**, **350**, **450**, **380**), **this**, IDC\_STATIC,

"Arial", **14**, RGB(**200**, **100**, **10**));

**return** TRUE; // возврат значения TRUE, если фокус не передан элементу управления

}

**void** CTetrDlg::OnSysCommand(UINT nID, LPARAM lParam)

{

**if** ((nID & **0xFFF0**) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

**else**

{

CDialogEx::OnSysCommand(nID, lParam);

}

}

// При добавлении кнопки свертывания в диалоговое окно нужно воспользоваться приведенным ниже кодом,

// чтобы нарисовать значок. Для приложений MFC, использующих модель документов или представлений,

// это автоматически выполняется рабочей областью.

**void** CTetrDlg::OnPaint()

{

**if** (IsIconic())

{

CPaintDC dc(**this**); // контекст устройства для рисования

SendMessage(WM\_ICONERASEBKGND, **reinterpret\_cast**<WPARAM>(dc.GetSafeHdc()), **0**);

// Выравнивание значка по центру клиентского прямоугольника

**int** cxIcon = GetSystemMetrics(SM\_CXICON);

**int** cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

**int** x = (rect.Width() - cxIcon + **1**) / **2**;

**int** y = (rect.Height() - cyIcon + **1**) / **2**;

// Нарисуйте значок

dc.DrawIcon(x, y, m\_hIcon);

}

**else**

{

CDialogEx::OnPaint();

}

}

// Система вызывает эту функцию для получения отображения курсора при перемещении

// свернутого окна.

HCURSOR CTetrDlg::OnQueryDragIcon()

{

**return** **static\_cast**<HCURSOR>(m\_hIcon);

}

// инициализация игры

**void** CTetrDlg::OnNewGame()

{

m\_GameWind.Clear(); // очистка поля

m\_NextWind.Next(); // выбор фигуры

m\_GameWind.NewFg(m\_NextWind.GetNext()); // отображение фигуры

m\_NextWind.Next(); // выбор следующей фигуры

m\_NextWind.RedrawWindow(); // перерисовка

m\_GameWind.RedrawWindow();

level = **1**; // начальный уровень

lines = **0**; // количество убранных линий

score = **0**; // счет

Timer();

m\_Score.SetNumber(score); // отображение счета

m\_Level.SetNumber(level); // отображение уровня

end = false;

}

// установка таймера (скорости) в соответствии со скоростью

**void** CTetrDlg::Timer()

{

**int** t;

**switch** (level) // получение времени между перемещением фигур

{

**case** **1**:

t = **1000**;

**break**;

**case** **2**:

t = **800**;

**break**;

**case** **3**:

t = **600**;

**break**;

**case** **4**:

t = **500**;

**break**;

**case** **5**:

t = **450**;

**break**;

**case** **6**:

t = **400**;

**break**;

**case** **7**:

t = **350**;

**break**;

**case** **8**:

t = **300**;

**break**;

**case** **9**:

t = **250**;

**break**;

**default:**

t = **200**;

**break**;

}

SetTimer(**1**, t, NULL);

}

**void** CTetrDlg::OnTimer(UINT\_PTR nIDEvent)

{

**if** (nIDEvent == **1** && !end)

{

**if** (!m\_GameWind.Down()) // дальше падать некуда

{

**int** k = m\_GameWind.Lines(); // подсчет и удаление полных линий

**switch** (k) // увеличение счета

{

**case** **1**:

score += **100**;

**break**;

**case** **2**:

score += **300**;

**break**;

**case** **3**:

score += **600**;

**break**;

**case** **4**:

score += **1000**;

**break**;

}

m\_Score.SetNumber(score); // отображение счета

lines += k;

**if** (lines >= **10**) // после удаления 10 линий переход на следующий уровень

{

lines = **0**;

level++;

m\_Level.SetNumber(level); // отображение уровня

Timer();

}

m\_GameWind.NewFg(m\_NextWind.GetNext()); // новая фигура

m\_NextWind.Next(); // выбор следующей фигуры

m\_NextWind.RedrawWindow();

**if** (m\_GameWind.End())

end = true; // проверка условия окончания игры

}

m\_GameWind.RedrawWindow();

}

CDialogEx::OnTimer(nIDEvent);

}

// функция позволяет обрабатывать события нажатия клавиш курсора в OnKeyDown

BOOL CTetrDlg::PreTranslateMessage(MSG\* pMsg)

{

**if** (pMsg->message == WM\_KEYDOWN)

{

**if** (pMsg->wParam == VK\_LEFT || pMsg->wParam == VK\_RIGHT || pMsg->wParam == VK\_DOWN || pMsg->wParam == VK\_UP)

**return** false;

}

**return** CDialogEx::PreTranslateMessage(pMsg);

}

**void** CTetrDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

**if** (!end) // игра не закончена

{

**switch** (nChar)

{

**case** VK\_LEFT: // влево

m\_GameWind.Left();

m\_GameWind.RedrawWindow();

**break**;

**case** VK\_RIGHT: // вправо

m\_GameWind.Right();

m\_GameWind.RedrawWindow();

**break**;

**case** VK\_DOWN: // вниз

m\_GameWind.Down();

m\_GameWind.RedrawWindow();

**break**;

**case** VK\_UP: // вверх - поворот

m\_GameWind.Rotate();

m\_GameWind.RedrawWindow();

**break**;

}

}

CDialogEx::OnKeyDown(nChar, nRepCnt, nFlags);

}

### Сlass.h

#include <afxtempl.h>

#define kSize 10

// класс - генератор случайных чисел

**class** **CRandom**

{

**public:**

CRandom();

**int** **random**(**int** max);

};

//-----------------------------------

// класс - фигура (из клеток)

**class** **CFigura** : **public** CObject

{

**protected:**

// класс - клетка

**class** **CKlet** : **public** CObject

{

**protected:**

**int** ik, jk; // координаты клетки

COLORREF cl; // цвет

**public:**

**static** **int** size; // размер клетки

CKlet(**int** i, **int** j, COLORREF c);

CKlet(CKlet\* p);

**void** **Paint**(CDC\* pdc, **int** ifg, **int** jfg); // отображение клетки

**bool** **TestRect**(**int** ifg, **int** jfg, **int** w, **int** h); // клетка в поле?

**bool** **Include**(**int** x, **int** y); // клетка имеет координаты (x,y)?

**void** **GetCoord**(**int**& x, **int**& y); // получение координат клетки

**void** **SetCoord**(**int** x, **int** y); // установка координат клетки

**void** **Rotate**(); // поворот

};

**typedef** CArray<CKlet\*, CKlet\*> CKletArray; // тип - массив клеток

**protected:**

**static** **int** tcount; // количество различных фигур

**static** **int** tcolor[]; // цвета фигур

**static** **int** tdata[][**4**][**2**]; // координаты клеток для фигур

**int** ifg, jfg; // координаты фигур

CKletArray kl; // клетки фигуры

**public:**

CFigura(**int** i, **int** j);

CFigura(CFigura\* f);

**virtual** ~CFigura();

**static** **int** **GetKletSize**(); // получение размера клетки

**void** **Init**(**int** t, **int** p); // установка фигуры (t - фигура, p - поворот)

**void** **InitRandom**(); // случайная фигура

**void** **Paint**(CDC\* pdc); // отображение фигуры

**bool** **Move**(**int** x, **int** y); // перемещение фигуры

**bool** **TestRect**(**int** w, **int** h); // фигура попадает в поле?

**bool** **FindKlet**(**int** x, **int** y); // одна из клеток фигуры имеет координаты (x,y)?

**bool** **operator**&&(CFigura& f); // фигуры пересекаются?

**void** **operator**=(CFigura& f); // копирование фигуры

**void** **Rotate**(); // вращение

**void** **DeleteLine**(**int** k); // удаление линии k

**bool** **Empty**(); // фигура не содержит клеток?

};

**typedef** CArray<CFigura\*, CFigura\*> CFgArray; // тип - набор фигур

//-----------------------------------

// класс - окно, отображающее фигуры

**class** **CFgWindow** : **public** CWnd

{

**protected:**

CFgArray fg; // список фигур

**public:**

**virtual** ~CFgWindow();

**virtual** BOOL **Create**(CWnd\* pParentWnd, **int** x, **int** y, UINT nID); // создание окна

**virtual** **void** GetSize(**int**& w, **int**& h) = **0**; // получение нужного размера окна

**void** **Clear**(); // очистка окна

**protected:**

**virtual** **void** **Paint**(CDC\* pdc); // отображение окна

**virtual** **void** PaintBackground(CDC\* pdc) = **0**; // отображение фона

// Implementation

**protected:**

//{{AFX\_MSG(CFgWindow)

afx\_msg **void** **OnPaint**();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

//-----------------------------------

// класс - следующая фигура

**class** **CNextWindow** : **public** CFgWindow

{

**public:**

**void** Next(); // выбор новой фигуры

CFigura\* **GetNext**(); // получение выбранной следующей фигуры

**protected:**

**virtual** **void** **GetSize**(**int**& w, **int**& h); // получение необходимого размера

**virtual** **void** **PaintBackground**(CDC\* pdc); // отображение фона

};

//-----------------------------------

// класс - игровое поле

**class** **CGameWindow** : **public** CFgWindow

{

**protected:**

**int** nheight; // высота поля

**int** nwidth; // ширина поля

**private:**

CDC\* pdcDisplayMemory; // контекст для двойной буферизации

CBitmap\* pBitmap; // битовая матрица для двойной буферизации отображения

**public:**

CGameWindow();

**virtual** ~CGameWindow();

**virtual** BOOL **Create**(CWnd\* pParentWnd, **int** x, **int** y, UINT nID, **int** nw, **int** nh);

**bool** **NewFg**(CFigura\* f); // добавление новой фигуры

**bool** **Down**(); // перемещение вниз

**void** **Left**(); // перемещение влево

**void** **Right**(); // перемещение вправо

**bool** **End**(); // проверка условия конца игры

**void** **Rotate**(); // поворот фигуры

**bool** **Move**(**int** x, **int** y); // перемещение фигуры

**bool** **TestMove**(**int** x, **int** y); // проверка возможности перемещения фигуры

**bool** **FindLine**(**int** k); // проверка заполенности линии

**void** **DeleteLine**(**int** k); // удаление линии

**int** **Lines**(); // подсчет и удаление линий

**protected:**

**virtual** **void** **PaintBackground**(CDC\* pdc); // отображение фона

**virtual** **void** **GetSize**(**int**& w, **int**& h); // получение необходимого размера

**bool** **TestMove**(CFigura\* f); // проверка возможности перемещния фигуры

// Implementation

**protected:**

//{{AFX\_MSG(CFgWindow)

afx\_msg **void** **OnPaint**();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

//-----------------------------------

// класс - цветная надпись

**class** **CColorText** : **public** CWnd

{

**protected:**

CString fontname; // имя шрифта

**int** fontsize; // размер шрифта

COLORREF color; // цвет

**public:**

CColorText();

**virtual** BOOL **Create**(**const** RECT& rect, CWnd\* pParentWnd, UINT nID);

**virtual** BOOL **Create**(**char**\* Text, **const** RECT& rect, CWnd\* pParentWnd, UINT nID,

**char**\* font, **int** fsize, COLORREF c);

**void** **SetNumber**(**int** Number); // отображение числа

// Implementation

**protected:**

//{{AFX\_MSG(CColorText)

afx\_msg **void** **OnPaint**();

//}}AFX\_MSG

DECLARE\_MESSAGE\_MAP()

};

### Сlass.cpp

#include "stdafx.h"

#include "Class.h"

CRandom Rnd;

//////////////////////////////////////////////////////////////////////

// инициализация генератора случайных чисел

CRandom::CRandom()

{

srand((**unsigned**)time(NULL));

}

// получение случайного числа 0 <= x < max

**int** CRandom::random(**int** max)

{

**return** rand() % max;

}

//////////////////////////////////////////////////////////////////////

**int** CFigura::CKlet::size = **26**; // размер клетки

CFigura::CKlet::CKlet(**int** i, **int** j, COLORREF c)

{

ik = i;

jk = j;

cl = c;

}

CFigura::CKlet::CKlet(CKlet\* p)

{

ik = p->ik;

jk = p->jk;

cl = p->cl;

}

// оторажение клетки

**void** CFigura::CKlet::Paint(CDC\* pdc, **int** ifg, **int** jfg)

{

**int** i, j;

i = (ifg + ik) \* size;

j = (jfg + jk) \* size;

CPen **pen**(PS\_SOLID, **1**, RGB(**100**, **100**, **100**));

CPen\* oldpen = pdc->SelectObject(&pen);

pdc->Rectangle(i, j, i + size, j + size);

pdc->FillSolidRect(i + **1**, j + **1**, size - **2**, size - **2**, cl);

pdc->SelectObject(oldpen);

}

// клетка в поле?

**bool** CFigura::CKlet::TestRect(**int** ifg, **int** jfg, **int** w, **int** h)

{

**int** i, j;

i = ifg + ik;

j = jfg + jk;

**return** i >= **0** && i < w&& j >= **0** && j < h;

}

// клетка имеет координаты (x,y)?

**bool** CFigura::CKlet::Include(**int** x, **int** y)

{

**return** (ik == x && jk == y);

}

// получение координат клетки

**void** CFigura::CKlet::GetCoord(**int**& x, **int**& y)

{

x = ik;

y = jk;

}

// установка координат клетки

**void** CFigura::CKlet::SetCoord(**int** x, **int** y)

{

ik = x;

jk = y;

}

// поворот

**void** CFigura::CKlet::Rotate()

{

**int** i, j;

i = jk;

j = **3** - ik;

ik = i;

jk = j;

}

//////////////////////////////////////////////////////////////////////

// координаты клеток для фигур

**int** CFigura::tdata[][**4**][**2**] =

{

{{**1**, **0**}, {**1**, **1**}, {**1**, **2**}, {**2**, **2**}},

{{**2**, **0**}, {**2**, **1**}, {**2**, **2**}, {**1**, **2**}},

{{**0**, **1**}, {**1**, **1**}, {**1**, **2**}, {**2**, **1**}},

{{**0**, **1**}, {**1**, **1**}, {**2**, **1**}, {**3**, **1**}},

{{**1**, **0**}, {**1**, **1**}, {**2**, **1**}, {**2**, **2**}},

{{**2**, **0**}, {**2**, **1**}, {**1**, **1**}, {**1**, **2**}},

{{**1**, **1**}, {**1**, **2**}, {**2**, **1**}, {**2**, **2**}} };

// цвета фигур

**int** CFigura::tcolor[] =

{

RGB(**255**, **0**, **0**),

RGB(**0**, **255**, **0**),

RGB(**0**, **0**, **255**),

RGB(**255**, **255**, **0**),

RGB(**255**, **0**, **255**),

RGB(**0**, **255**, **255**),

RGB(**255**, **255**, **255**) };

// количество различных фигур

**int** CFigura::tcount = **7**;

CFigura::CFigura(**int** i, **int** j)

{

ifg = i;

jfg = j;

InitRandom();

}

CFigura::CFigura(CFigura\* f)

{

ifg = f->ifg;

jfg = f->jfg;

**int** n = f->kl.GetSize();

**for** (**int** i = **0**; i < n; i++)

kl.Add(**new** CKlet(f->kl[i]));

}

CFigura::~CFigura()

{

CKlet\* p;

**while** (kl.GetSize() > **0**)

{

p = kl[**0**];

kl.RemoveAt(**0**);

**delete** p;

}

}

**int** CFigura::GetKletSize() // получение размера клетки

{

**return** CKlet::size;

}

// установка фигуры (t - фигура, p - поворот)

**void** CFigura::Init(**int** t, **int** p)

{

**if** (t >= tcount)

**return**;

**for** (**int** i = **0**; i < **4**; i++)

kl.Add(**new** CKlet(tdata[t][i][**0**], tdata[t][i][**1**], tcolor[t]));

**for** (**int** i = **0**; i < p; i++)

{

Rotate();

}

}

// случайная фигура

**void** CFigura::InitRandom()

{

**int** t, p;

t = Rnd.random(tcount);

p = Rnd.random(**4**);

Init(t, p);

}

// отображение фигуры

**void** CFigura::Paint(CDC\* pdc)

{

**int** n = kl.GetSize();

**for** (**int** i = **0**; i < n; i++)

kl[i]->Paint(pdc, ifg, jfg);

}

// перемещение фигуры

**bool** CFigura::Move(**int** x, **int** y)

{

ifg += x;

jfg += y;

**return** true;

}

// фигура попадает в поле?

**bool** CFigura::TestRect(**int** w, **int** h)

{

**int** n = kl.GetSize();

**for** (**int** i = **0**; i < n; i++)

**if** (!kl[i]->TestRect(ifg, jfg, w, h))

**return** false;

**return** true;

}

// одна из клеток фигуры имеет координаты (x,y)?

**bool** CFigura::FindKlet(**int** x, **int** y)

{

**int** n = kl.GetSize();

**for** (**int** i = **0**; i < n; i++)

**if** (kl[i]->Include(x - ifg, y - jfg))

**return** true;

**return** false;

}

// фигуры пересекаются?

**bool** CFigura::**operator**&&(CFigura& f)

{

**int** x1, y1, x2, y2, n1, n2, i1, i2;

n1 = kl.GetSize();

n2 = f.kl.GetSize();

**for** (i1 = **0**; i1 < n1; i1++)

**for** (i2 = **0**; i2 < n2; i2++)

{

kl[i1]->GetCoord(x1, y1);

f.kl[i2]->GetCoord(x2, y2);

**if** (ifg + x1 == f.ifg + x2 && jfg + y1 == f.jfg + y2)

**return** true;

}

**return** false;

}

// копирование фигуры

**void** CFigura::**operator**=(CFigura& f)

{

ifg = f.ifg;

jfg = f.jfg;

CKlet\* p;

**while** (kl.GetSize() > **0**)

{

p = kl[**0**];

kl.RemoveAt(**0**);

**delete** p;

}

**while** (f.kl.GetSize() > **0**)

{

p = f.kl[**0**];

f.kl.RemoveAt(**0**);

kl.Add(p);

}

}

// вращение

**void** CFigura::Rotate()

{

**int** n = kl.GetSize();

**for** (**int** i = **0**; i < n; i++)

kl[i]->Rotate();

}

// удаление линии k

**void** CFigura::DeleteLine(**int** k)

{

CKlet\* p;

**int** x, y;

**int** n = kl.GetSize();

**int** j = **0**;

**for** (**int** i = **0**; i < n; i++)

{

p = kl[j];

p->GetCoord(x, y);

**if** (jfg + y == k)

{

kl.RemoveAt(j);

**delete** p;

}

**else**

{

j++;

**if** (jfg + y < k)

{

y++;

p->SetCoord(x, y);

}

}

}

}

// фигура не содержит клеток?

**bool** CFigura::Empty()

{

**return** kl.GetSize() == **0**;

}

//////////////////////////////////////////////////////////////////////

CFgWindow::~CFgWindow()

{

Clear();

}

// создание окна

BOOL CFgWindow::Create(CWnd\* pParentWnd, **int** x, **int** y, UINT nID)

{

CRect rect;

**int** w, h;

GetSize(w, h); // вычисление размера

rect.SetRect(x, y, x + w, y + h);

**return** CWnd::Create(NULL, NULL, WS\_CHILD | WS\_VISIBLE | WS\_DISABLED,

rect, pParentWnd, nID, NULL);

}

// очистка окна

**void** CFgWindow::Clear()

{

CFigura\* f;

**while** (fg.GetSize() > **0**)

{

f = fg[**0**];

fg.RemoveAt(**0**);

**delete** f;

}

}

// отображение окна

**void** CFgWindow::Paint(CDC\* pdc)

{

PaintBackground(pdc);

**int** n = fg.GetSize();

**for** (**int** i = **0**; i < n; i++)

fg[i]->Paint(pdc);

}

BEGIN\_MESSAGE\_MAP(CFgWindow, CWnd)

//{{AFX\_MSG\_MAP(CFgWindow)

ON\_WM\_PAINT()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

**void** CFgWindow::OnPaint()

{

CPaintDC dc(**this**);

Paint(&dc);

}

//////////////////////////////////////////////////////////////////////

// получение необходимого размера

**void** CNextWindow::GetSize(**int**& w, **int**& h)

{

h = w = CFigura::GetKletSize() \* **4**;

}

// отображение фона

**void** CNextWindow::PaintBackground(CDC\* pdc)

{

CRect rect;

GetClientRect(rect);

pdc->FillSolidRect(rect, GetSysColor(COLOR\_BTNFACE));

}

// выбор новой фигуры

**void** CNextWindow::Next()

{

CFigura\* f;

**while** (fg.GetSize() > **0**)

{

f = fg[**0**];

fg.RemoveAt(**0**);

**delete** f;

}

fg.Add(**new** CFigura(**0**, **0**));

}

// получение выбранной следующей фигуры

CFigura\* CNextWindow::GetNext()

{

CFigura\* f;

f = fg[**0**];

fg.RemoveAt(**0**);

**return** f;

}

//////////////////////////////////////////////////////////////////////

CGameWindow::CGameWindow()

{

pdcDisplayMemory = **new** CDC;

pBitmap = **new** CBitmap;

}

CGameWindow::~CGameWindow()

{

CFigura\* f;

**while** (fg.GetSize() > **0**)

{

f = fg[**0**];

fg.RemoveAt(**0**);

**delete** f;

}

**delete** pBitmap;

**delete** pdcDisplayMemory;

}

// получение необходимого размера

**void** CGameWindow::GetSize(**int**& w, **int**& h)

{

h = CFigura::GetKletSize() \* nheight + **2**;

w = CFigura::GetKletSize() \* nwidth + **2**;

}

// отображение фона

**void** CGameWindow::PaintBackground(CDC\* pdc)

{

CRect rect;

GetClientRect(rect);

CPen **pen**(PS\_SOLID, **1**, RGB(**0**, **0**, **0**));

CPen\* oldpen = pdc->SelectObject(&pen);

pdc->Rectangle(rect.left, rect.top, rect.Width(), rect.Height());

pdc->FillSolidRect(rect.left + **1**, rect.top + **1**, rect.Width() - **2**, rect.Height() - **2**,

GetSysColor(COLOR\_BTNFACE));

pdc->SelectObject(oldpen);

pdc->SetViewportOrg(**1**, **1**);

}

BOOL CGameWindow::Create(CWnd\* pParentWnd, **int** x, **int** y, UINT nID, **int** nw, **int** nh)

{

BOOL r;

nwidth = nw;

nheight = nh;

r = CFgWindow::Create(pParentWnd, x, y, nID);

**if** (pdcDisplayMemory->GetSafeHdc() == NULL)

{

CClientDC dc(**this**);

CRect rect;

GetClientRect(rect);

pdcDisplayMemory->CreateCompatibleDC(&dc);

pBitmap->CreateCompatibleBitmap(&dc, rect.right, rect.bottom);

}

**return** r;

}

// добавление новой фигуры

**bool** CGameWindow::NewFg(CFigura\* f)

{

fg.InsertAt(**0**, f, **1**);

f->Move(**4**, **0**);

**return** true;

}

// проверка возможности перемещения фигуры

**bool** CGameWindow::TestMove(CFigura\* f)

{

**if** (!f->TestRect(nwidth, nheight))

**return** false;

**int** n = fg.GetSize();

**for** (**int** i = **1**; i < n; i++)

**if** (\*f && \*fg[i])

**return** false; // фигура пересекается с другой

**return** true;

}

// поворот фигуры

**void** CGameWindow::Rotate()

{

CFigura cur(fg[**0**]); // копия фигуры

cur.Rotate(); // поворот фигруы

**if** (TestMove(&cur))

\*fg[**0**] = cur; // изменение текущей фигуры

}

// перемещение фигуры

**bool** CGameWindow::Move(**int** x, **int** y)

{

CFigura cur(fg[**0**]); // копия фигуры

cur.Move(x, y); // поворот фигруы

**if** (TestMove(&cur))

{

\*fg[**0**] = cur; // изменение текущей фигуры

**return** true;

}

**return** false;

}

// перемещение вниз

**bool** CGameWindow::Down()

{

**return** Move(**0**, **1**);

}

// перемещение влево

**void** CGameWindow::Left()

{

Move(-**1**, **0**);

}

// перемещение вправо

**void** CGameWindow::Right()

{

Move(**1**, **0**);

}

// проверка условия конца игры

**bool** CGameWindow::End()

{

**return** !Move(**0**, **0**);

}

BEGIN\_MESSAGE\_MAP(CGameWindow, CWnd)

//{{AFX\_MSG\_MAP(CGameWindow)

ON\_WM\_PAINT()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

**void** CGameWindow::OnPaint()

{

CBitmap\* pOldBitmap = pdcDisplayMemory->SelectObject(pBitmap);

pdcDisplayMemory->SelectClipRgn(NULL);

Paint(pdcDisplayMemory);

CPaintDC **dc**(**this**);

CRect rect;

GetClientRect(rect);

pdcDisplayMemory->SetViewportOrg(**0**, **0**);

dc.BitBlt(rect.left, rect.top, rect.Width(), rect.Height(),

pdcDisplayMemory, rect.left, rect.top, SRCCOPY);

}

// проверка заполенности линии

**bool** CGameWindow::FindLine(**int** k)

{

**int** i, j, n;

**bool** find;

**for** (i = **0**; i < nwidth; i++)

{

n = fg.GetSize();

find = false;

**for** (j = **0**; j < n; j++)

{

**if** (fg[j]->FindKlet(i, k))

{

find = true;

**break**;

}

}

**if** (!find)

**return** false;

}

**return** true;

}

// удаление линии

**void** CGameWindow::DeleteLine(**int** k)

{

CFigura\* f;

**int** n = fg.GetSize();

**int** j = **0**;

**for** (**int** i = **0**; i < n; i++)

{

f = fg[j];

f->DeleteLine(k);

**if** (f->Empty())

{

fg.RemoveAt(j);

**delete** f;

}

**else**

j++;

}

}

// подсчет и удаление линий

**int** CGameWindow::Lines()

{

**int** i;

**int** k = **0**;

**for** (i = **0**; i < nheight; i++)

**if** (FindLine(i))

{

DeleteLine(i);

k++;

}

**return** k;

}

//////////////////////////////////////////////////////////////////////

CColorText::CColorText()

{

fontname = "Arial";

color = RGB(**0**, **0**, **0**);

fontsize = **10**;

}

BOOL CColorText::Create(**const** RECT& rect, CWnd\* pParentWnd, UINT nID)

{

**return** CWnd::Create(NULL, NULL, WS\_CHILD | WS\_VISIBLE | WS\_DISABLED,

rect, pParentWnd, nID, NULL);

}

BOOL CColorText::Create(**char**\* Text, **const** RECT& rect, CWnd\* pParentWnd, UINT nID,

**char**\* font, **int** fsize, COLORREF c)

{

BOOL r;

r = Create(rect, pParentWnd, nID);

SetWindowText(Text);

fontname = font;

fontsize = fsize;

color = c;

**return** r;

}

BEGIN\_MESSAGE\_MAP(CColorText, CWnd)

//{{AFX\_MSG\_MAP(CColorText)

ON\_WM\_PAINT()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

// отображение надписи

**void** CColorText::OnPaint()

{

CString s;

CRect rect;

CFont font;

CFont\* oldfont;

CBrush **brush**(GetSysColor(COLOR\_BTNFACE));

CPaintDC **dc**(**this**);

font.CreatePointFont(fontsize \* **10**, fontname);

oldfont = dc.SelectObject(&font);

GetWindowText(s);

GetClientRect(rect);

dc.SetTextColor(color);

dc.SetBkColor(GetSysColor(COLOR\_BTNFACE));

dc.FillRect(rect, &brush);

dc.DrawText(s, rect, DT\_CENTER | DT\_VCENTER);

dc.SelectObject(oldfont);

}

// отображение числа

**void** CColorText::SetNumber(**int** Number)

{

CString s;

s.Format("%d", Number);

SetWindowText(s);

RedrawWindow();

}

### Tetr.h

// Tetr.h : главный файл заголовка для приложения PROJECT\_NAME

//

#pragma once

#ifndef \_\_AFXWIN\_H\_\_

#error "включить stdafx.h до включения этого файла в PCH"

#endif

#include "resource.h" // основные символы

// CTetrApp:

// О реализации данного класса см. Tetr.cpp

//

**class** **CTetrApp** : **public** CWinApp

{

**public:**

CTetrApp();

// Переопределение

**public:**

**virtual** BOOL **InitInstance**();

// Реализация

DECLARE\_MESSAGE\_MAP()

};

**extern** CTetrApp theApp;

### Tetr.cpp

// Tetr.cpp : Определяет поведение классов для приложения.

#include "stdafx.h"

#include "Tetr.h"

#include "TetrDlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CTetrApp

BEGIN\_MESSAGE\_MAP(CTetrApp, CWinApp)

ON\_COMMAND(ID\_HELP, &CWinApp::OnHelp)

END\_MESSAGE\_MAP()

// создание CTetrApp

CTetrApp::CTetrApp()

{

// поддержка диспетчера перезагрузки

m\_dwRestartManagerSupportFlags = AFX\_RESTART\_MANAGER\_SUPPORT\_RESTART;

// TODO: добавьте код создания,

// Размещает весь важный код инициализации в InitInstance

}

// Единственный объект CTetrApp

CTetrApp theApp;

// инициализация CTetrApp

BOOL CTetrApp::InitInstance()

{

// InitCommonControlsEx() требуется для Windows XP, если манифест

// приложения использует ComCtl32.dll версии 6 или более поздней версии для включения

// стилей отображения. В противном случае будет возникать сбой при создании любого окна.

INITCOMMONCONTROLSEX InitCtrls;

InitCtrls.dwSize = **sizeof**(InitCtrls);

// Выберите этот параметр для включения всех общих классов управления, которые необходимо использовать

// в вашем приложении.

InitCtrls.dwICC = ICC\_WIN95\_CLASSES;

InitCommonControlsEx(&InitCtrls);

CWinApp::InitInstance();

AfxEnableControlContainer();

// Создать диспетчер оболочки, в случае, если диалоговое окно содержит

// представление дерева оболочки или какие-либо его элементы управления.

CShellManager\* pShellManager = **new** CShellManager;

CTetrDlg dlg;

m\_pMainWnd = &dlg;

INT\_PTR nResponse = dlg.DoModal();

**if** (nResponse == IDOK)

{

// TODO: Введите код для обработки закрытия диалогового окна

// с помощью кнопки "ОК"

}

**else** **if** (nResponse == IDCANCEL)

{

// TODO: Введите код для обработки закрытия диалогового окна

// с помощью кнопки "Отмена"

}

// Удалить диспетчер оболочки, созданный выше.

**if** (pShellManager != NULL)

{

**delete** pShellManager;

}

// Поскольку диалоговое окно закрыто, возвратите значение FALSE, чтобы можно было выйти из

// приложения вместо запуска генератора сообщений приложения.

**return** FALSE;

}

### Resource.h

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by Tetr.rc

//

#define IDM\_ABOUTBOX 0x0010

#define IDD\_ABOUTBOX 100

#define IDS\_ABOUTBOX 101

#define IDD\_TETR\_DIALOG 102

#define IDR\_MAINFRAME 128

#define IDR\_MENU 129

#define IDC\_GAMEWIND 1001

#define IDC\_NEXTWIND 1002

#define ID 32771

#define ID\_NEWGAME 32772

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NEXT\_RESOURCE\_VALUE 130

#define \_APS\_NEXT\_COMMAND\_VALUE 32773

#define \_APS\_NEXT\_CONTROL\_VALUE 1000

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

### Stdafx.h

// stdafx.h: включите файл для добавления стандартных системных файлов

// или конкретных файлов проектов, часто используемых,

// но редко изменяемых

#pragma once

#ifndef \_SECURE\_ATL

#define \_SECURE\_ATL 1

#endif

#ifndef VC\_EXTRALEAN

#define VC\_EXTRALEAN // Исключите редко используемые компоненты из заголовков Windows

#endif

#include "targetver.h"

#define \_ATL\_CSTRING\_EXPLICIT\_CONSTRUCTORS // некоторые конструкторы CString будут явными

// отключает функцию скрытия некоторых общих и часто пропускаемых предупреждений MFC

#define \_AFX\_ALL\_WARNINGS

#include <afxwin.h> // основные и стандартные компоненты MFC

#include <afxext.h> // расширения MFC

#include <afxdisp.h> // классы автоматизации MFC

#ifndef \_AFX\_NO\_OLE\_SUPPORT

#include <afxdtctl.h> // поддержка MFC для типовых элементов управления Internet Explorer 4

#endif

#ifndef \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcmn.h> // поддержка MFC для типовых элементов управления Windows

#endif // \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcontrolbars.h> // поддержка MFC для лент и панелей управления

#ifdef \_UNICODE

#if defined \_M\_IX86

#pragma comment(linker, "/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#elif defined \_M\_X64

#pragma comment(linker, "/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#else

#pragma comment(linker, "/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#endif

#endif

### Targetver.h

#pragma once

// Включение SDKDDKVer.h обеспечивает определение самой последней доступной платформы Windows.

// Если требуется выполнить построение приложения для предыдущей версии Windows, включите WinSDKVer.h и

// задайте для макроопределения \_WIN32\_WINNT значение поддерживаемой платформы перед вхождением SDKDDKVer.h.

#include <SDKDDKVer.h>

### Stdafx.cpp

// stdafx.cpp: исходный файл, содержащий стандарт

// будет предварительно откомпилированным заголовком

// stdafx.obj будет содержать предварительно откомпилированные сведения о типе

#include "stdafx.h"