**МІНІСТЕРТСВО ОСВІТИ І НАУКИ УКРАЇНИ**

Національний університет водного господарства та природокористування

Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки

Кафедра прикладної математики

Курсова робота

на тему:

«Програмна реалізація гри "Змійка"»

Виконав: студент групи

ПМ-21 ННІАКОТ

Ясковець І. О.

Перевірив: ст. викладач

Жуковський В. В.

Рівне 2015

### ЗМІСТ

1. Вступ
2. Постановка завдання
3. Опис алгоритму вирішення.
4. Вихідний код програми на С#
5. Тестування програми
6. Блок-схеми
7. Висновок
8. Список літератури

**1. Вступ**  
 C# є дуже близьким родичем мови [програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [Java](https://uk.wikipedia.org/wiki/Java). Мова Java була створена компанією Sun Microsystems, коли глобальний розвиток інтернету поставив задачу роззосереджених обчислень. Взявши за основу популярну мову C++, Java виключила з неї потенційно небезпечні речі (типу вказівників без контролю виходу за межі). Для роззосереджених обчислень була створена концепція віртуальної машини та машинно-незалежного байт-коду, свого роду посередника між вихідним текстом програм і апаратними інструкціями комп'ютера чи іншого інтелектуального пристрою.

Java набула чималої популярності, і була ліцензована також і компанією [Microsoft](https://uk.wikipedia.org/wiki/Microsoft). Але з плином часу Sun почала винуватити Microsoft, що та при створенні свого клону Java робить її сумісною виключно з платформою [Windows](https://uk.wikipedia.org/wiki/Windows), чим суперечить самій концепції машинно-незалежного середовища виконання і порушує ліцензійну угоду. Microsoft відмовилася піти назустріч вимогам Sun, і тому з'ясування стосунків набуло статусу [судового](https://uk.wikipedia.org/wiki/%D0%A1%D1%83%D0%B4) процесу. Суд визнав позицію Sun справедливою, і зобов'язав Microsoft відмовитися від позаліцензійного використання Java.

У цій ситуації в Microsoft вирішили, користуючись своєю вагою на ринку, створити свій власний аналог Java, мови, в якій корпорація стане повновладним господарем. Ця новостворена мова отримала назву C#. Вона успадкувала від Java концепції віртуальної машини (середовище .NET), байт-коду (MSIL) і більшої безпеки вихідного коду програм, плюс врахувала досвід використання програм на Java.

Нововведенням C# стала можливість легшої взаємодії, порівняно з мовами-попередниками, з кодом програм, написаних на інших мовах, що є важливим при створенні великих проектів. Якщо програми на різних мовах виконуються на платформі [.NET](https://uk.wikipedia.org/wiki/.NET), .NET бере на себе клопіт щодо сумісності програм (тобто типів даних, за кінцевим рахунком).

Станом на сьогодні C# визначено флагманською мовою корпорації Microsoft, бо вона найповніше використовує нові можливості .NET. Решта мов програмування, хоч і підтримуються, але визнані такими, що мають спадкові прогалини щодо використання .NET.

**2. Постановка завдання**

З розвитком цифрових технологій комп’ютери все більше входять в життя людини. Якщо раніше вони використовувалися виключно для складних математичних обчислень, то сьогодні сфера їх застосування значно розширилася. Комп'ютерні ігри - одне з найбільш масових застосувань комп'ютера.

Розвиток ігрової індустрії йшов стрімким темпом, і особливо користувався популярністю у підлітків. Перші ігри вирізнялися простотою інтерфейсу і логіки, але з часом вони ставали все складніше і складніше, над їх створенням працювала вже не одна людина, а ціла команда розробників.

Сучасні ігри вимагають досить великої продуктивності від комп'ютера, і не кожна офісна машина в силах відтворювати їх. Однак для відпочинку від монотонної роботи найчастіше досить простої, та не вимогливої до техніки, гри. Саме такій розробці присвячений даний курсовий проект - гра "Змійка".

**Графічний інтерфейс**

Користувацький інтерфейс-це способи користувача з вашою програмою, які можуть включати в себе зображення, звуки і текст. Орієнтуючись на середнього користувача інтерфейс повинен бути простим і зручним. Це знизить ймовірність помилок.

Мета: Створити гру на спритність "Змійка"

Закінчений варіант гри змійка повинен містити:

1) "Змійку", яка може рухатися в обмеженій площині. Маневрами "Змійки" повинен управляти гравець. "Змійка" повинна збільшуватися в розмірах при зіткненні з "яблуком" і гинути при зіткненні з "стіною", при перетині самої себе і при виході за межі площини заданої програмістом.

2) "Яблуко" або інший об'єкт, при зіткненні з яким "Змійка" "з'їдає" його (збільшується в довжину в залежності від умов заданих програмістом.

3) "стіна" об'єкт при зіткненні з яким змійка гине.

"Змійка" може управлятися за допомогою клавіш на формі, гра може мати лічильник набраних очок, таблицю кращих гравців, яка може збережуться і зчитуватися з файлу, а також інші елементи які збільшують цікавість гри.

**3. Опис алгоритму вирішення**

Алгоритм   
 1) Описуються змінні і константи

2) Вибір початкових координат заданим чином

3) Промальовування початкового положення

4) Генерація на полі яблука випадковим чином

5) Перевірка: чи не з'явилося яблуко на змійці.

6) Управління "змійкою" з клавіатури.

7) Перевірка: "не вкусила" чи "змійка сама себе", якщо вкусила виводиться інформаційне вікно (Інформаційна форма), в іншому випадку переходимо до наступного пункту.

8) Перевірка: чи "не з'їла" "змійка" "яблуко" якщо з'їла, то нарощуємо їй хвіст і переходимо до пункту 3), в іншому випадку див Наступний пункт.

9) Перемальовування "Змійки"

Гра на спритність "Змійка"

Закінчений варіант гри змійка повинен містити:

1) "Змійку", яка може рухатися в обмеженій площині. Маневрами "Змійки" повинен управляти гравець. "Змійка" повинна збільшуватися в розмірах при зіткненні з "яблуком" і гинути при зіткненні з "стіною", при перетині самої себе і при виході за межі площини заданої програмістом.

2) "Яблуко" або інший об'єкт, при зіткненні з яким "Змійка" "з'їдає" його (збільшується в довжину в залежності від умов заданих програмістом.

3) "стіна" об'єкт при зіткненні з яким змійка гине."Змійка" може управлятися за допомогою клавіш на формі, гра може мати лічильник набраних очок, таблицю кращих гравців, яка можуть зберігатися і зчитуватися з файлу, а також інші елементи які збільшують цікавість гри.   
Функції управління екранними об'єктами

Опис функцій   
/ / Функція розкидання яблук

private void PlaceFood()

{

var r = new Random();

var p = new Point();

do

{

p.X = r.Next(\_size.Width);

p.Y = r.Next(\_size.Height);

} while (\_snake.Contains(p));

Food = p;

}

1. Перевіряємо об'єкт-яблуко, чи потрапило воно на об'єкт-змійку

2. Якщо потрапило-> break;   
3. Присвоюємо колір об'кту-яблуко   
4. Малюєм об'єкт-яблуко  
  
/ / Функція старту змійки її координат і швидкості

1. Задаємо точки-розмір об’єкту змійка   
2. Коли створили то запам'ятовуємо позицію   
3. Задаємо швидкість пересування об'єкту змійці  
4. Малюємо об'єкт яблуко

/ / Функція яка рухає і малює   
public bool Move(EDirection direction, out List<Visualizer.Terrain> coordsToRepaint) //Рухаємо змію. Якщо вона себе кусає, повертаємо true;

{

coordsToRepaint = new List<Visualizer.Terrain> { new Visualizer.Terrain(Visualizer.TerrainInfo.Backcolor, \_snake.Peek()),

new Visualizer.Terrain(Visualizer.TerrainInfo.Snake, \_head) };

var s = \_head + \_dictionary[direction];

if (\_snake.Contains(s) || \_head.X <= 0 || \_head.X >= \_size.Width - 1 || \_head.Y<=0 || \_head.Y>=\_size.Height - 1)

return true;

\_head = s;

\_snake.Enqueue(\_head);

coordsToRepaint.Add(new Visualizer.Terrain(Visualizer.TerrainInfo.Head, \_head));

Кольори  
 Екранні об'єкти можна зафарбовувати різними кольорами. Можуть бути присутніми три кольори (колір фону, колір змії та її голови).  
  
  
Структура:

namespace Snake

{

class Snake

{

public enum EDirection //Перечислення напрямків, куди йти

{

N,S,E,W

}

private const int ScoreForFood = 10;

private readonly Dictionary<EDirection, Point> \_dictionary = new Dictionary<EDirection, Point> //Словник, переводить напрямок у відносні кординати

{

{EDirection.N, new Point(0, -1)}, //Північ

{EDirection.S, new Point(0, 1)}, //Південь

{EDirection.W, new Point(-1, 0)}, //Захід

{EDirection.E, new Point(1, 0)}, //Схід

};

private struct Segment //Структура сегментів. Фактично той же Point, тільки з перезавантаженням декількох параметрів, для того, щоб можна було легко складати відносні координати.

{

public readonly int X;

public readonly int Y;

public Segment(int x, int y) : this()

{

X = x;

Y = y;

}

public Segment(Point p) : this(p.X, p.Y) {}

public static Segment operator +(Segment lhs, Segment rhs)

{

return new Segment(lhs.X + rhs.X, lhs.Y + rhs.Y);

}

#region Point<->Segment implementation

public static implicit operator Point(Segment segment)

{

return new Point(segment.X, segment.Y);

}

public static implicit operator Segment(Point p)

{

return new Segment(p.X, p.Y);

}

#endregion

}

private readonly Queue<Segment> \_snake; //Сама змійка

private Segment \_head; //Голова

private readonly Size \_size; //Розміри карти

private static Point Food { get; set; } //Координати клітинки з їжею

public static uint Score { get; private set; }

public Snake(int n, Size mapSize) //Конструктор, визначає розміри карти, і кількість сегментів змії

{

\_snake = new Queue<Segment>();

\_size = mapSize;

var p = new Point(\_size.Width/2 - n/2, \_size.Height/2); //Ставимо змійку по центру карти

for (int i = 0; i < n; i++, p.X++)

\_snake.Enqueue(new Segment(p));

p.X--;

\_head = p;

PlaceFood();

}

public List<Point> GetCoords()

{

return \_snake.Select(segment => (Point) segment).ToList(); //Повертаємо список координат змії

}

public bool Move(EDirection direction, out List<Visualizer.Terrain> coordsToRepaint) //Рухаємо змію. Якщо вона себе кусає, повертаємо true;

{

coordsToRepaint = new List<Visualizer.Terrain> { new Visualizer.Terrain(Visualizer.TerrainInfo.Backcolor, \_snake.Peek()),

new Visualizer.Terrain(Visualizer.TerrainInfo.Snake, \_head) };

var s = \_head + \_dictionary[direction];

if (\_snake.Contains(s) || \_head.X <= 0 || \_head.X >= \_size.Width - 1 || \_head.Y<=0 || \_head.Y>=\_size.Height - 1)

return true;

\_head = s;

\_snake.Enqueue(\_head);

coordsToRepaint.Add(new Visualizer.Terrain(Visualizer.TerrainInfo.Head, \_head));

if (\_head == Food) //Якщо змійка потрапила на клітинку з їжею, ми заповнюємо випадкову клітинку їжею.

{

Score += ScoreForFood;

PlaceFood();

}

else \_snake.Dequeue(); //Інакше, щоб змія не збільшилась, прибираємо останній сегмент

coordsToRepaint.Add(new Visualizer.Terrain(Visualizer.TerrainInfo.Food, Food));

return false; //Ми себе не вкусили, значить false

}

private void PlaceFood()

{

var r = new Random();

var p = new Point();

do

{

p.X = r.Next(\_size.Width);

p.Y = r.Next(\_size.Height);

} while (\_snake.Contains(p));

Food = p;

}

}

}

**4. Вихідний текст програми С #**  
Розробка програми, що представляє собою комп'ютерну гру «Змійка»   
using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

namespace Snake

{

class Snake

{

public enum EDirection //Перечислення напрямків, куди йти

{

N,S,E,W

}

private const int ScoreForFood = 10;

private readonly Dictionary<EDirection, Point> \_dictionary = new Dictionary<EDirection, Point> //Словник, переводить напрямок у відносні координати

{

{EDirection.N, new Point(0, -1)}, //Північ

{EDirection.S, new Point(0, 1)}, //Південь

{EDirection.W, new Point(-1, 0)}, //Захід

{EDirection.E, new Point(1, 0)}, //Схід

};

private struct Segment //Структура сегментів. Фактично той же Point, тільки з перезавантаженням декількох параметрів, для того, щоб можна було легко складати відносні координати.

{

public readonly int X;

public readonly int Y;

public Segment(int x, int y) : this()

{

X = x;

Y = y;

}

public Segment(Point p) : this(p.X, p.Y) {}

public static Segment operator +(Segment lhs, Segment rhs)

{

return new Segment(lhs.X + rhs.X, lhs.Y + rhs.Y);

}

#region Point<->Segment implementation

public static implicit operator Point(Segment segment)

{

return new Point(segment.X, segment.Y);

}

public static implicit operator Segment(Point p)

{

return new Segment(p.X, p.Y);

}

#endregion

}

private readonly Queue<Segment> \_snake; //Сама змійка

private Segment \_head; //Голова

private readonly Size \_size; //Розміри карти

private static Point Food { get; set; } //Координати клітинки з їжею

public static uint Score { get; private set; }

public Snake(int n, Size mapSize) //Конструктор, визначає розміри карти, і кількість сегментів змії

{

\_snake = new Queue<Segment>();

\_size = mapSize;

var p = new Point(\_size.Width/2 - n/2, \_size.Height/2); //Ставимо змійку по центру карти

for (int i = 0; i < n; i++, p.X++)

\_snake.Enqueue(new Segment(p));

p.X--;

\_head = p;

PlaceFood();

}

public List<Point> GetCoords()

{

return \_snake.Select(segment => (Point) segment).ToList(); //Повертаємо список координат змії

}

public bool Move(EDirection direction, out List<Visualizer.Terrain> coordsToRepaint) //Рухаємо змію. Якщо вона себе кусає, повертаємо true;

{

coordsToRepaint = new List<Visualizer.Terrain> { new Visualizer.Terrain(Visualizer.TerrainInfo.Backcolor, \_snake.Peek()),

new Visualizer.Terrain(Visualizer.TerrainInfo.Snake, \_head) };

var s = \_head + \_dictionary[direction];

if (\_snake.Contains(s) || \_head.X <= 0 || \_head.X >= \_size.Width - 1 || \_head.Y<=0 || \_head.Y>=\_size.Height - 1)

return true;

\_head = s;

\_snake.Enqueue(\_head);

coordsToRepaint.Add(new Visualizer.Terrain(Visualizer.TerrainInfo.Head, \_head));

if (\_head == Food) //Якщо змійка потрапила на клітинку з їжею, ми заповнюємо випадкову клітинку їжею.

{

Score += ScoreForFood;

PlaceFood();

}

else \_snake.Dequeue(); //Інакше, щоб змія не збільшилась, прибираємо останній сегмент

coordsToRepaint.Add(new Visualizer.Terrain(Visualizer.TerrainInfo.Food, Food));

return false; //Ми себе не вкусили, значить false

}

private void PlaceFood()

{

var r = new Random();

var p = new Point();

do

{

p.X = r.Next(\_size.Width);

p.Y = r.Next(\_size.Height);

} while (\_snake.Contains(p));

Food = p;

}

}

}

**5. Тестування програми**

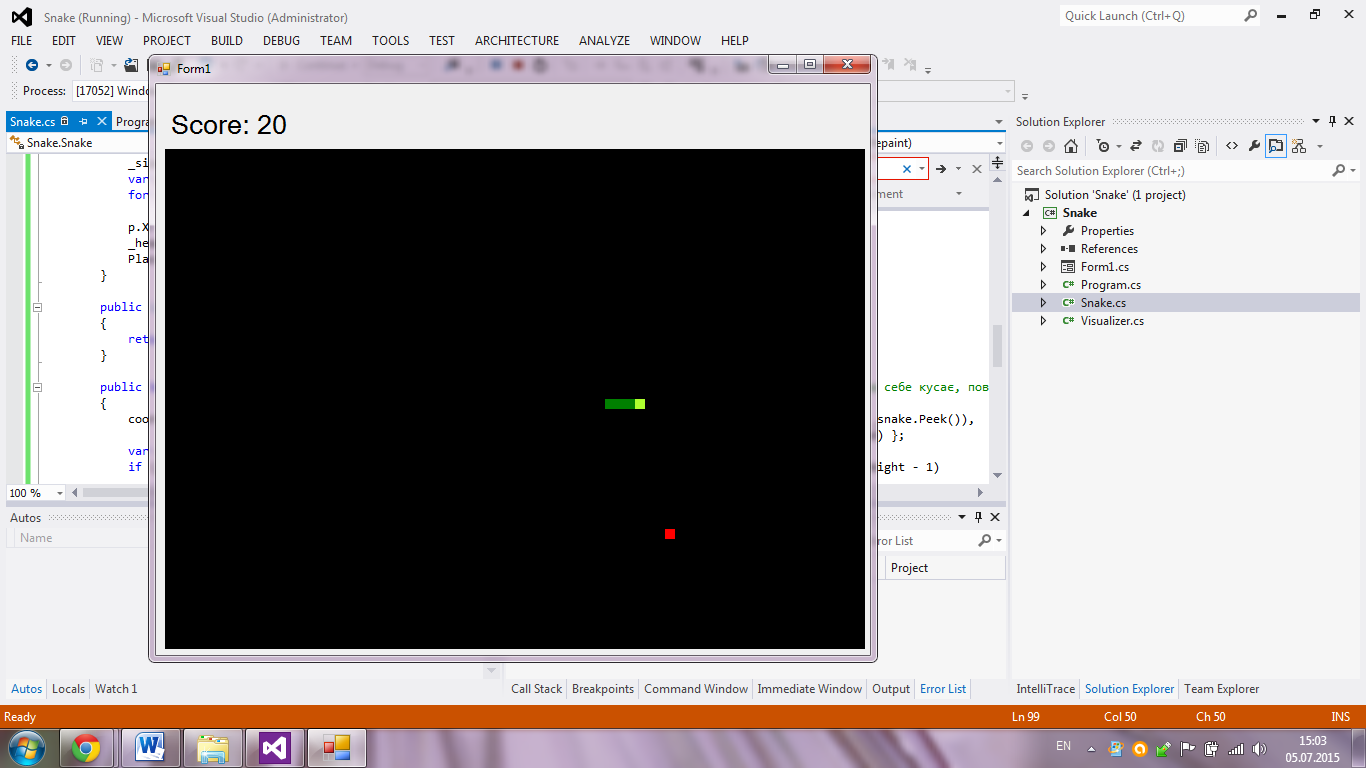
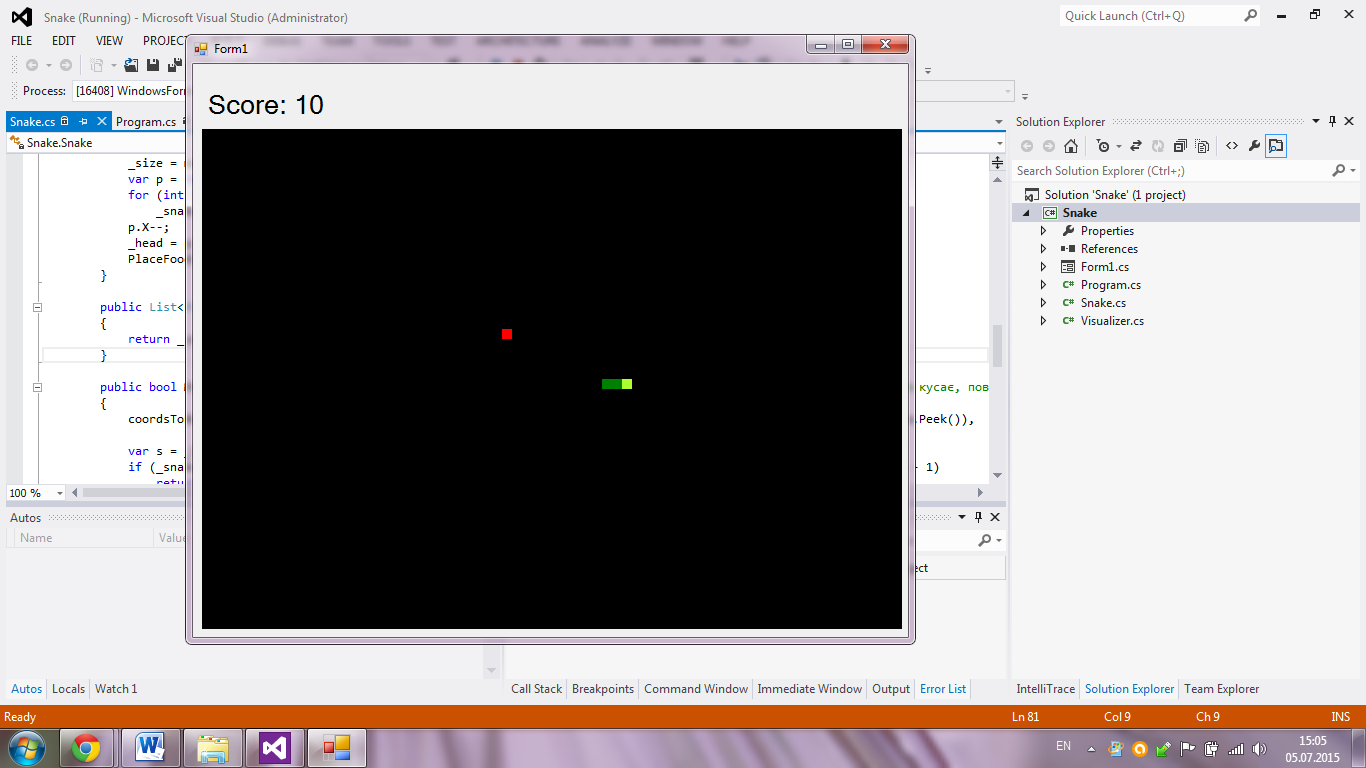
Програма тестувалася, на Windows 7 (32 та 64 розрядній версіях).   
**Вигляд гри**  


Рис. 1. Вікно програми

Рис. 2. Вікно програми

**6. Блок-схеми**

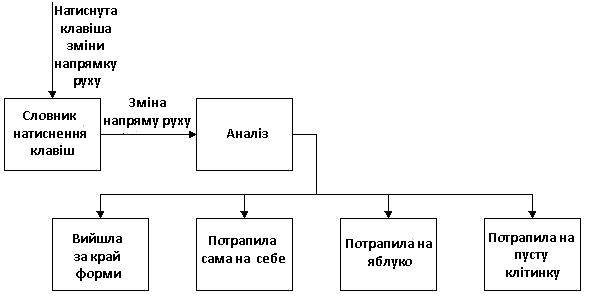
****

Рис. 3. UML діаграма станів

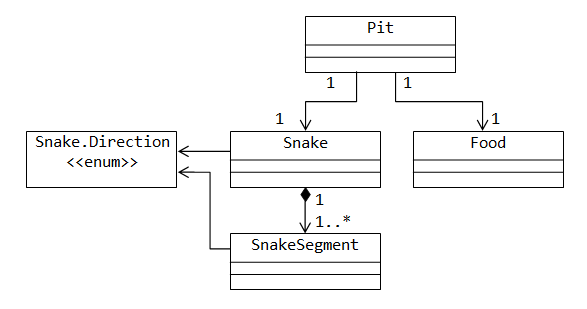
****

Рис. 4. UML діаграма класів/об’єктів

**7. Висновок**  
 У коді завдання зазначено максимально допустиму кількість яблук, тому в грі передбачено дане обмеження.   
 Можна додати озвучування подій гри, рівень складності гри і різні налаштування параметрів гри в коді програми. Це викликало б більший інтерес до гри.   
 У цілому додаток являє собою майже найпростіший варіант гри «Змійка» з набором найнеобхідніших функцій.   
**8.Список літератури**

1. В.А. Жарков. “Компьютерная графика, мультимедия и игры на Visual C#”. – Жарков Пресс, 2005. – с.31-38
2. Герберт Шилдт. “C# 4.0: полное руководство”. – Вільямс, 2011. – с.155-170.
3. Гради Буч, Джеймс Рамбо, Івар Якобсон “Язык UML. Руководство

пользователя” 2-ге вид. – ДМК Пресс, 2006. – с.120-148, 252-261