Пензенский государственный технологический университет

Факультет автоматизированных информационных технологий

Кафедра «Прикладная информатика»

ЕН.Ф.02

**СОЗДАНИЕ КЛАССА «Змейки»**

Отчет о лабораторной работе № 7

по дисциплине «Информатика и программирование»

Выполнили: ст-ты гр.17СН1с

Тюняев А.В., Богданов М.О.

Проверил: ст. преп. каф. ПИ

Юранов В.С.

2018

**1 Формулировка задачи**

1. В приложение из 4 лабораторной работы в класс Point необходимо добавить методы для перемещения точки и очистки точки.
2. Добавить новый класс Snake описывающий змейку.
3. Добавить в класс Snake методы для движения змейки в заданном направлении.
4. В основной программе создать экземпляр класса змейки и бесконечный цикл, в котором проверить движение объекта змейки.

**2 Требования к программе**

Программа должна вывести границы экрана и “Змейку”

**3 Описание программы**

**3.1 Общие сведения**

Программа разработана в среде Visual Studio 2017

**3.2 Функциональное назначение**

Программа предназначена для вывода границ экрана и “Змейки”

**3.3 Описание логической структуры**

*Класс Point:*

В классе объявляются три поля, значение оси x, значение оси y, символ, который будет выводится.

Перегрузка конструктора Point.

Конструктор Point с тремя параметрами.

Конструктор Point с одним параметром.

Метод Draw служит для вывода символов на экран.

Метод IsHit используется для проверки точек на пересечения.

Метод Move используется для изменения положения змейки.

Метод Clear используется для удаления последней точки змейки.

*Класс Figure:*

Создается список pList для записи координат точек

Метод Draw служит для вывода символов на экран используя список pList.

Метод IsHit используется для проверки фигур на пересечение.

Метод IsHit используется для проверки точек на пересечения.

*Класс VerticalLine:*

Этот класс является наследником класса Figure.

Метод VerticalLine принимает значения смещения по оси y, координату по оси x и символ.

Создается экземпляр класса List<Point> (списка).

В цикл передаются значения начальной и конечной точки по оси y.

В цикле создается экземпляр класса Point и в список добавляются координаты точек по осям x и y, и необходимый символ.

*Класс HorizontalLine:*

Этот класс является наследником класса Figure.

Метод HorizontalLine принимает значения смещения по оси x, координату по оси y и символ.

Создается экземпляр класса List<Point> (списка).

В цикл передаются значения начальной и конечной точки по оси x.

В цикле создается экземпляр класса Point и в список добавляются координаты точек по осям x и y, и необходимый символ.

*Класс Walls:*

Создается список wallList.

В методе Walls

Создаются экземпляры классов HorizontalLine, VerticalLine и передаются параметры.

Происходит добавление прямых в список wallList.

С помощью метода Draw происходит вывод фигур на экран.

*Перечисление Direction:*

Добавляем перечисление направлений.

*Класс Snake:*

Этот класс является наследником класса Figure.

В конструктор класса Snake будут передаваться следующие параметры: Точку с которой будет начинаться хвост, текущую длину змейки, направление движения змейки.

В цикле несколько раз будут создаваться точки являющиеся точной копией хвостовой точки, которая передана в конструкторе, затем эта точка будет сдвинута на заданной число позиций по направлению движения змейки и эта точка будет добавлена в список.

Метод Move.

В переменную tail класса Point записывается первый элемента списка pList с помощью метода First().

Удаление хвостовой точки из списка pList методом Remove().

Вызывается метод GetNextPoint для определения координат перемещения головной точки.

Добавление точки, полученной из метода GetNextPoint, в список pList.

Удаление хвостовой точки змейки из консоли.

Метод GetNextPoint:

Получение текущего положения головы змейки из списка pList с помощью метода Last.

Создается новая точка, которая является копией предыдущего положения головной точки.

Перемещение точки по направлению движения змейки.

Получение точки, которая является новым положением головы змейки.

*Класс Program:*

Метод Main (основной метод) в нем создается экземпляр класса Walls, и передаются параметры. Вызывается метод Draw класса Walls, для вывода границ экрана.

Создается переменная класса Point в нее передаются координаты начальной точки змейки и символ.

Создается переменная класса Snake и в нее передаются параметры начальной точки, длину змейки и направление движения.

Вызывается метод Draw для отрисовки змейки.

В бесконечном цикле вызывается метод Move для движения змейки.

Создается задержка.

**3 Описание применения**

После запуска программы выводятся 4 линии, которые образуют границы экрана и выводится перемещающаяся фигура “Змейка”.

**Выводы**

Познакомились со способами анимации в консольном приложении.

**Контрольные вопросы**

1. **Для чего используется тип-перечисление?**

Перечисляемый тип— в программировании тип данных, чьё множество значений представляет собой ограниченный список идентификаторов

1. **Что такое конструктор класса?**

Конструктор класса — специальный блок инструкций, вызываемый при создании объекта

1. **Что такое метод класса?**

Класс— это сложный (структурированный, составной) тип данных, объединяющий переменные, которые называют полями класса, и функции для работы с этими полями, которые называют *методами* класса.

1. **В чем отличие модификатором доступа private и public?**

* public: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
* private: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.

ТЕКСТ ПРОГРАММЫ

Приложение А

(обязательное)

namespace Lab7

{

class Point

{

int x;

int y;

char sym;

public Point(int x, int y, char sym)

{

this.x = x;

this.y = y;

this.sym = sym;

}

public Point(Point p)

{

x = p.x;

y = p.y;

sym = p.sym;

}

public void Draw()

{

Console.SetCursorPosition(x, y);

Console.Write(sym);

}

public bool IsHit(Point p)

{

return p.x == this.x && p.y == this.y;

}

public void Move(int offset, Direction direction)

{

if (direction == Direction.RIGHT)

{

x = x + offset;

}

else if (direction == Direction.LEFT)

{

x = x - offset;

}

else if (direction == Direction.UP)

{

y = y - offset;

}

else if (direction == Direction.DOWN)

{

y = y + offset;

}

}

public void Clear()

{

sym = ' ';

Draw();

}

}

}

namespace Lab7

{

class VerticalLine : Figure

{

public VerticalLine(int yUp, int yDown, int x, char sym)

{

pList = new List<Point>();

for (int y = yUp; y <= yDown; y++)

{

Point p = new Point(x, y, sym);

pList.Add(p);

}

}

}

}

namespace Lab7

{

class HorizontalLine : Figure

{

public HorizontalLine(int xLeft, int xRight, int y, char sym)

{

pList = new List<Point>();

for (int x = xLeft; x <= xRight; x++)

{

Point p = new Point(x, y, sym);

pList.Add(p);

}

}

}

}

namespace Lab7

{

class Walls

{

List<Figure> wallList;

public Walls(int mapWidth, int mapHeight)

{

wallList = new List<Figure>();

HorizontalLine upLine = new HorizontalLine(0, mapWidth - 2, 0, '+');

HorizontalLine downLine = new HorizontalLine(0, mapWidth - 2, mapHeight - 1, '+');

VerticalLine leftLine = new VerticalLine(0, mapHeight - 1, 0, '+');

VerticalLine rightLine = new VerticalLine(0, mapHeight - 1, mapWidth - 2, '+');

wallList.Add(upLine);

wallList.Add(downLine);

wallList.Add(leftLine);

wallList.Add(rightLine);

}

public bool IsHit(Figure figure)

{

foreach (var wall in wallList)

{

if (wall.IsHit(figure))

{

return true;

}

}

return false;

}

public void Draw()

{

foreach (var wall in wallList)

{

wall.Draw();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab7

{

class Figure

{

protected List<Point> pList;

public void Draw()

{

foreach (Point p in pList)

{

p.Draw();

}

}

public bool IsHit(Figure figure)

{

foreach (var p in pList)

{

if (figure.IsHit(p))

return true;

}

return false;

}

private bool IsHit(Point point)

{

foreach (var p in pList)

{

if (p.IsHit(point))

return true;

}

return false;

}

}

}

namespace Lab7

{

enum Direction

{

LEFT,

RIGHT,

UP,

DOWN

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab7

{

class Snake : Figure

{

Direction direction;

public Snake(Point tail, int length, Direction \_direction)

{

direction = \_direction;

pList = new List<Point>();

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

{

Point p = new Point(tail);

p.Move(i, direction);

pList.Add(p);

}

}

public void Move()

{

Point tail = pList.First();

pList.Remove(tail);

Point head = GetNextPoint();

pList.Add(head);

tail.Clear();

head.Draw();

}

private Point GetNextPoint()

{

Point head = pList.Last();

Point nextPoint = new Point(head);

nextPoint.Move(1, direction);

return nextPoint;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

namespace Lab7

{

class Program

{

static void Main(string[] args)

{

Walls walls = new Walls(80, 30);

walls.Draw();

Point p = new Point(4, 5, '\*');

Snake snake = new Snake(p, 4, Direction.RIGHT);

snake.Draw();

while (true)

{

snake.Move();

Thread.Sleep(100);

}

}

}

}

РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

Приложение Б

(обязательное)

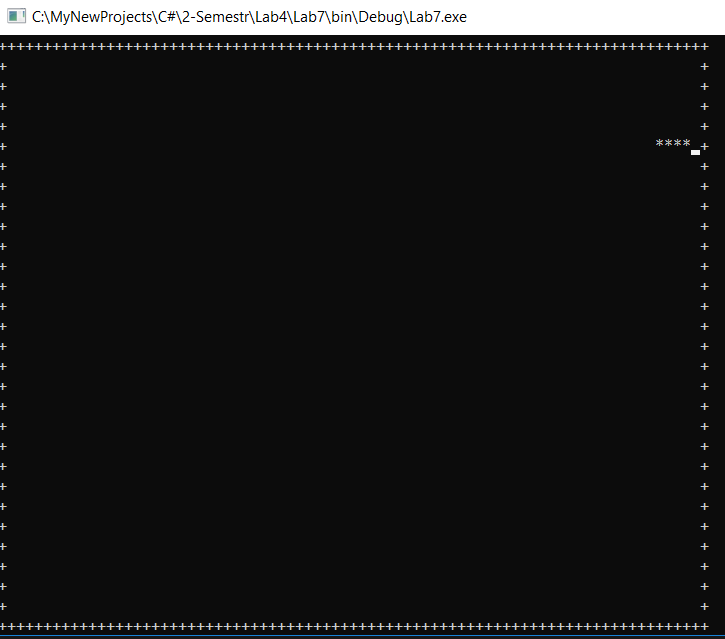


Рисунок 1 – Результат выполнения программы