МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**Лабораторная работа № 5**

по курсу «Объектно-ориентированное программирование»

**«Классы, объекты, наследование в C#»**

Выполнили:

студенты гр. КТбо2-7

Миронченко П.Д.

Проверил:

Тарасов С. А.

**Таганрог 2020**

1. **Спецификации классов  
   Спецификация класса Program** Класс является точкой входа в программу и имеет лишь один статический метод Main

**Спецификация класса Game**

Класс Game представляет собой консольный интерактор. Содержит в себе объекты классов наследованных от Figure, Food Creator и Point. Осуществляет взаимодействие с ними

**Спецификация класса Figure**

Содержит в себе список объектов класса Point и имеет методы отрисовки объектов в консоли и проверки пересечения с объектами классов Point и Figure и абстрактные методы движения и добавления новый точек в список

**Спецификация класса Point** Класс содержит в себе координаты какой-либо точки, её символ и цвет и методы получения следующей точки по направлению, отрисовки точки на консоль и проверки пересечения с другим объектом класса Point

**Спецификация класса Snake**

Является наследником класса Figure. Реализует все абстрактные методы Figure.

**Спецификация класса Field**

Является наследником класса Figure, содержит в себе только поля и методы родительского класса

**Спецификация класса Borderline**

Является наследником класса Figure, содержит в себе только поля и методы родительского класса

**Спецификация класса FoodCreator** класс содержит в себе поля типа Random и Point, а также метод генерации объектов, класса Point  
**Спецификация класса GameOverException**

Класс является пользовательским исключением, которое выкидывается в классе Snake, если координаты головы змейки равны каким-либо другим координатам змейки

**Спецификация класса FigureFactory**

Класс является фабрикой, которая имеет 3 метода: создать объект класса Snake, Field и Borderline  
  
**2. Используемые математические зависимости и алгоритмы**

В программе не использовались какие-либо математические зависимости и алгоритмы общего назначения

1. **Диаграмма класса**

Diagram, schematic

Description automatically generated

1. **Листинг программы**

**Program.cs**

using System;

namespace lab5

{

class Program

{

static void Main()

{

Game game = new Game(30, 30);

game.Play();

}

}

}

**Game.cs**

using System;

using System.Threading;

namespace lab5

{

public class Game

{

private bool \_gameOver;

private Figure \_borderline;

private Figure \_field;

private Figure \_snake;

private FoodCreator \_foodCreator;

private Point \_food;

private Direction \_direction;

public Game(int rows, int columns)

{

\_gameOver = false;

FigureFactory factory = new FigureFactory(rows, columns);

\_field = factory.CreateField();

\_snake = factory.CreateSnake();

\_borderline = factory.CreateBorderline();

\_foodCreator = new FoodCreator(rows, columns);

\_foodCreator.GenerateNewFood();

\_food = \_foodCreator.Food;

\_direction = Direction.LEFT;

Console.SetWindowSize(rows+2, columns+2);

}

private void \_UpdateField()

{

if (\_snake.isCross(\_food))

{

\_foodCreator.GenerateNewFood();

\_food = \_foodCreator.Food;

\_snake.AddPoint();

}

if (\_snake.isCross(\_borderline))

{

throw new GameOverException();

}

\_borderline.Draw();

\_field.Draw();

\_food.Draw();

\_snake.Draw();

}

private void \_ReadDirection()

{

if (Console.KeyAvailable)

{

ConsoleKeyInfo key = Console.ReadKey();

switch (key.Key)

{

case ConsoleKey.LeftArrow:

\_direction = Direction.LEFT;

break;

case ConsoleKey.RightArrow:

\_direction = Direction.RIGHT;

break;

case ConsoleKey.DownArrow:

\_direction = Direction.DOWN;

break;

case ConsoleKey.UpArrow:

\_direction = Direction.UP;

break;

}

}

}

public void Play()

{

int levelOfSpeed = -1;

while (!(levelOfSpeed >= 1 && levelOfSpeed <= 10))

{

Console.WriteLine("Choose level of dificult from 1 to 10");

levelOfSpeed = Convert.ToInt32(Console.ReadLine());

}

Console.Clear();

while (!\_gameOver)

{

try

{

\_ReadDirection();

\_snake.Move(\_direction);

\_UpdateField();

Thread.Sleep(500/levelOfSpeed);

}

catch (GameOverException)

{

\_gameOver = true;

Console.Write("Game over!");

}

}

}

}

}

**Figure.cs**

using System;

using System.Collections.Generic;

namespace lab5

{

public abstract class Figure

{

protected List<Point> \_figure;

protected Figure()

{

\_figure = new List<Point>();

}

public void Draw()

{

foreach (Point p in \_figure)

{

p.Draw();

}

}

public bool isCross(Figure other)

{

foreach (Point p in \_figure)

{

if (other.isCross(p))

{

return true;

}

}

return false;

}

public bool isCross(Point point)

{

foreach(Point p in \_figure)

{

if (p.isCross(point))

{

return true;

}

}

return false;

}

public abstract void Move(Direction direction);

public abstract void AddPoint();

}

}

**Borderline.cs**

using System;

namespace lab5

{

public class Borderline : Figure

{

public Borderline(int rows, int columns)

{

for (int i = 0; i < columns+2; ++i)

{

\_figure.Add(new Point(0, i, '#', ConsoleColor.Black));

}

for (int i = 1; i <= rows; ++i)

{

\_figure.Add(new Point(i, 0, '#', ConsoleColor.Black));

\_figure.Add(new Point(i, columns+1, '#', ConsoleColor.Black));

}

for (int i = 0; i < columns+2; ++i)

{

\_figure.Add(new Point(rows+1, i, '#', ConsoleColor.Black));

}

}

public override void AddPoint()

{

throw new NotImplementedException();

}

public override void Move(Direction direction)

{

throw new NotImplementedException();

}

}

}

**Field.cs**

using System;

using System.Collections.Generic;

namespace lab5

{

public class Field : Figure

{

public Field(int rows, int columns)

{

for (int i = 1; i <= rows; ++i)

{

for (int j = 1; j <= columns; ++j)

{

\_figure.Add(new Point(i, j, '0', ConsoleColor.Gray));

}

}

}

public override void AddPoint()

{

throw new NotImplementedException();

}

public override void Move(Direction direction)

{

throw new NotImplementedException();

}

}

}

**Snake.cs**

using System;

using System.Collections.Generic;

namespace lab5

{

public class Snake : Figure

{

public Snake(int rows, int columns)

{

\_figure.Add(new Point(rows / 2, columns / 2, 'S', ConsoleColor.Red));

}

public override void AddPoint()

{

int idx = \_figure.Count - 1;

int xTail = \_figure[idx].x;

int yTail = \_figure[idx].y;

\_figure.Add(new Point(yTail, xTail,'S', ConsoleColor.Red));

}

public override void Move(Direction direction)

{

\_figure.Insert(0, \_figure[0].GetNextPoint(direction));

\_figure.Remove(\_figure[\_figure.Count-1]);

if (\_figure.Count > 2 &&\_figure[0].isCross(\_figure[2]))

{

\_figure.Reverse();

}

for (int i = 3; i < \_figure.Count; ++i)

{

if (\_figure[0].isCross(\_figure[i]))

{

throw new GameOverException("Game over!\n");

}

}

}

}

}

**Direction.cs**

using System;

namespace lab5

{

public enum Direction

{

LEFT,

RIGHT,

UP,

DOWN

}

}

**FoodCreator.cs**

using System;

namespace lab5

{

public class FoodCreator

{

private Point \_food;

public Point Food

{

get { return \_food; }

}

private int \_columns;

private int \_rows;

private Random \_generator;

public FoodCreator(int rows, int columns)

{

\_generator = new Random();

\_rows = rows;

\_columns = columns;

\_food = new Point(rows / 2, columns / 2, '$', ConsoleColor.Yellow);

}

public void GenerateNewFood()

{

\_food.x = \_generator.Next(1, \_columns+1);

\_food.y = \_generator.Next(1, \_rows+1);

}

}

}

**Point.cs**

using System;

namespace lab5

{

public class Point

{

public int x;

public int y;

public char symbol;

public ConsoleColor color;

public Point(int \_y, int \_x, char \_symbol, ConsoleColor \_color)

{

y = \_y;

x = \_x;

symbol = \_symbol;

color = \_color;

}

public Point GetNextPoint(Direction direction)

{

if (direction == Direction.UP)

{

return new Point(y - 1, x, symbol, color);

}

else if (direction == Direction.DOWN)

{

return new Point(y + 1, x, symbol, color);

}

else if (direction == Direction.LEFT)

{

return new Point(y, x-1, symbol, color);

}

else if (direction == Direction.RIGHT)

{

return new Point(y, x+1, symbol, color);

}

return new Point(y, x, symbol, color);

}

public bool isCross(Point other)

{

return x == other.x && y == other.y;

}

public void Draw()

{

Console.BackgroundColor = color;

Console.SetCursorPosition(x, y);

Console.Write(symbol);

}

}

}

**GameOverException.cs**

using System;

namespace lab5

{

public class GameOverException : Exception

{

public GameOverException()

{

}

public GameOverException(string message) : base(message)

{

}

public GameOverException(string message, Exception inner)

: base(message, inner)

{

}

}

}

**FigureFactory.cs**

using System;

namespace lab5

{

public class FigureFactory

{

private int \_rows;

private int \_columns;

public FigureFactory(int rows, int columns)

{

\_rows = rows;

\_columns = columns;

}

public Figure CreateSnake()

{

return new Snake(\_rows, \_columns);

}

public Figure CreateField()

{

return new Field(\_rows, \_columns);

}

public Figure CreateBorderline()

{

return new Borderline(\_rows, \_columns);

}

}

}

**s**