ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Работу выполнил:

Студент 682 группы

Мочалов Н.

Работу проверил:

Павлов В.С.

Томск, 2020

Задание:

Нужно разработать игру «змейку»

Функциональные требования:

- управление змейкой должно происходить по клавишам (вверх, вниз, влево, вправо).

- Змейка должна неприрывно двигаться в сторону, куда нажал пользователь, по нажатии на другую клавишу, змейка должна поменять свое движение в сторону нажатой клавиши

- скорость и размер поля должен меняться в зависимости от выбранной пользователем сложности игры

легко[поле10х10, скорость медленная]

нормально[поле15х15,скорость нормальная]

сложно[после20х20,скорость быстрая]

Требования к оформлению:

- поле должно быть очерченым и понятным для пользователя

- змейка должна обладать головой и телом (символы ASCII)

- «фрукты» должны так же иметь свои собственные символы (ASCII)

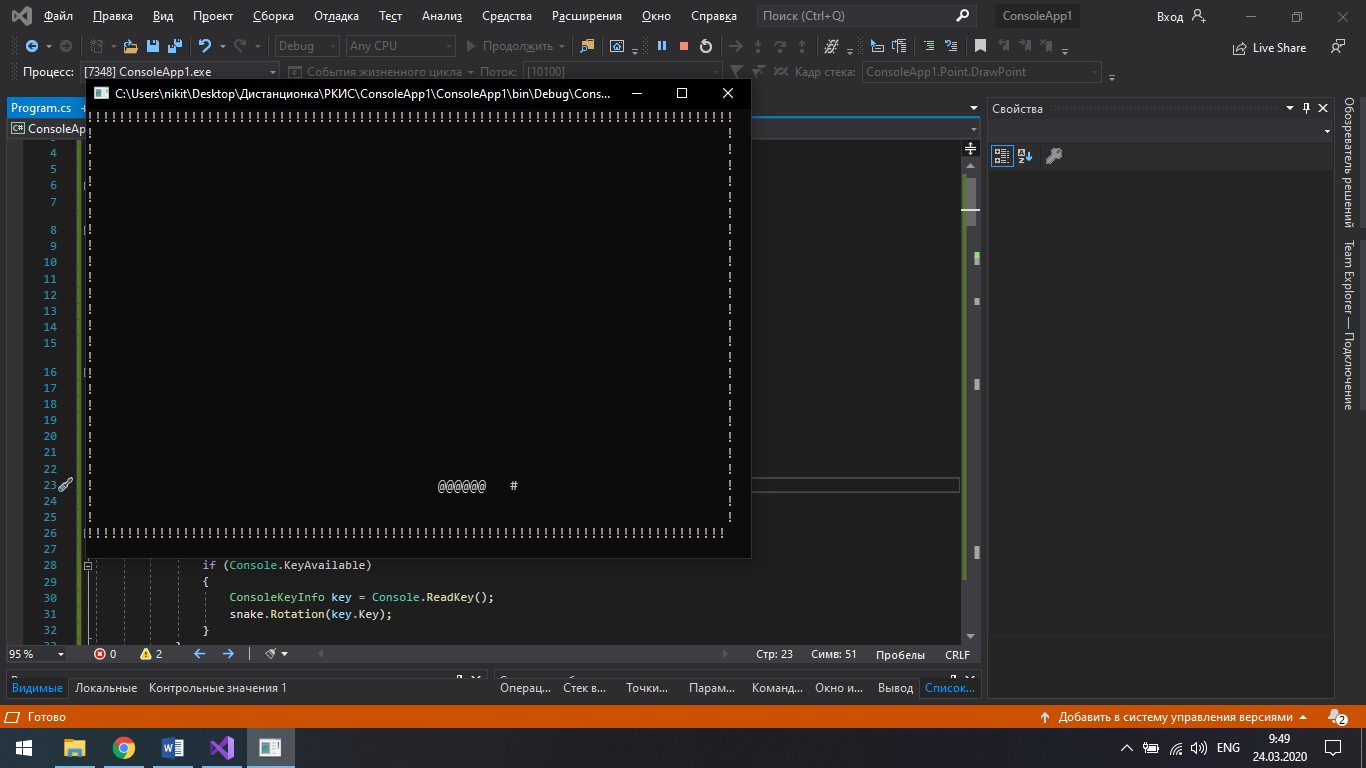
Требования к выполнению работы(общ)

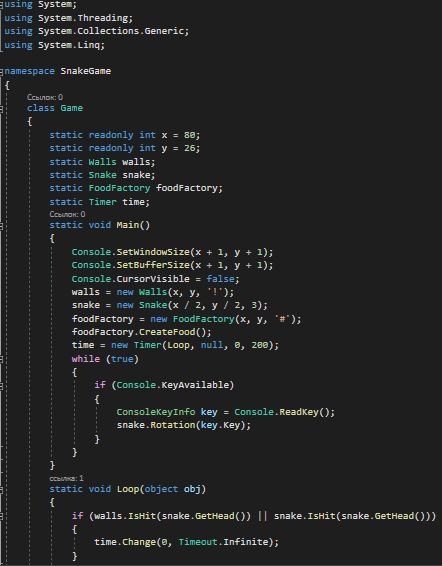
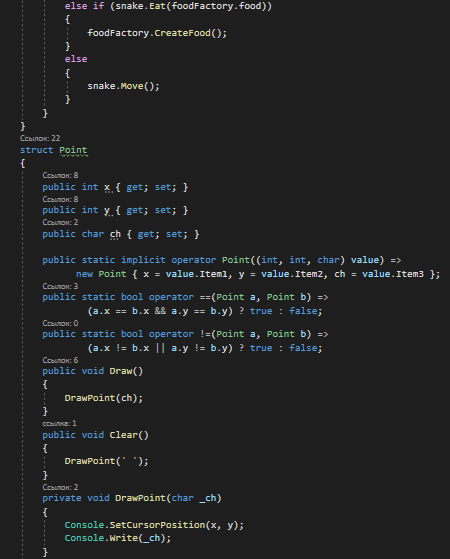
- работа должна быть выполнена на функциях и классах

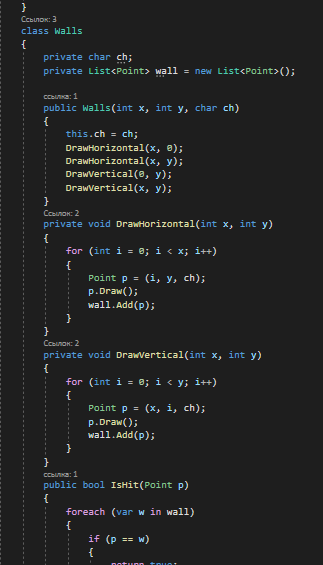
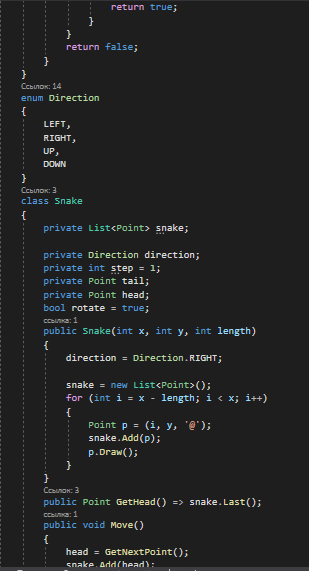
- поле и змейка не должны «мерцать» при движении

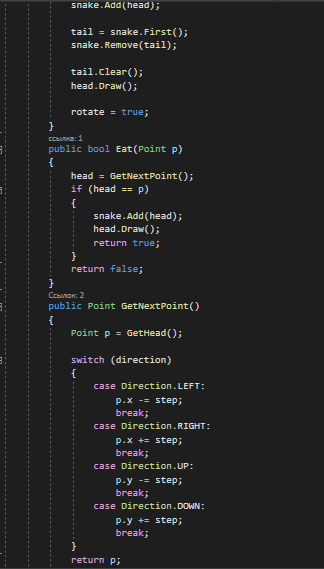
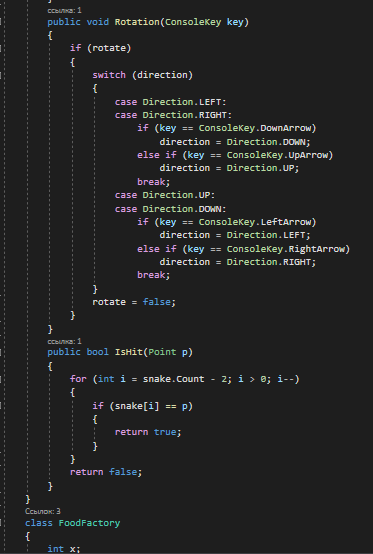
Требования к отчету:

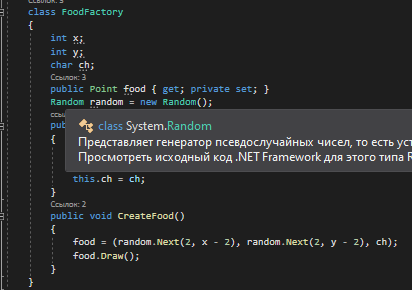
- файл с отчетом должен содержать: титульный лист, задание, результат работы (скриншоты), листинг программы с подробными комментариями.





**Делаю поле игры. Задал переменные X и Y, размер и буфер окна.**

class Game

{

static readonly int x = 80;

static readonly int y = 26;

static void Main()

{

Console.SetWindowSize(x + 1, y + 1);

Console.SetBufferSize(x + 1, y + 1);

}

}

**Point содержит координаты и символ, который будет выводится на экран. Тут же методы для вывода на экран точки и ее «стирания».**

struct Point

{

public int x

{ get; set; }

public int y { get; set; }

public char ch { get; set; }

public static implicit operator Point((int, int, char) value) =>

new Point {x = value.Item1, y = value.Item2, ch = value.Item3};

public void Draw(){ DrawPoint(ch); }

public void Clear(){ DrawPoint(' '); }

private void DrawPoint(char \_ch)

{

Console.SetCursorPosition(x, y);

Console.Write(\_ch);

}

}

**Тут стены, границы игрового поля и отрисовка.**

class Walls

{

private char ch;

private List<Point> wall = new List<Point>();

public Walls(int x, int y, char ch)

{

this.ch = ch; DrawHorizontal(x, 0);

DrawHorizontal(x, y);

DrawVertical(0, y);

DrawVertical(x, y);

}

private void DrawHorizontal(int x, int y)

{

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

{

Point p = (i, y, ch);

p.Draw();

wall.Add(p);

}

}

private void DrawVertical(int x, int y)

{

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

{

Point p = (x, i, ch);

p.Draw();

wall.Add(p); } } }

**Объявил поле walls в классе Game, а в Main инициализировал.**

class Game

{

static Walls walls;

static void Main()

{

walls = new Walls(x, y, '#');

**Генерим еду, класс FoodFactory, который и будет заниматься созданием еды внутри стен.**

class FoodFactory

{

int x;

int y;

char ch;

public Point food { get; private set; }

Random random = new Random();

public FoodFactory(int x, int y, char ch)

{

this.x = x;

this.y = y;

this.ch = ch;

}

public void CreateFood()

{

food = (random.Next(2, x - 2), random.Next(2, y - 2), ch);

food.Draw(); } }

**Движение змейки.**

class Snake

{

private List<Point> snake;

private Direction direction;

private int step = 1;

private Point tail;

private Point head;

bool rotate = true;

public Snake(int x, int y, int length)

{

direction = Direction.RIGHT;

snake = new List<Point>();

for (int i = x - length; i < x; i++)

{

Point p = (i, y, '\*');

snake.Add(p); p.Draw(); } }

public Point GetHead() => snake.Last();

public void Move()

{

head = GetNextPoint();

snake.Add(head);

tail = snake.First();

snake.Remove(tail);

tail.Clear();

head.Draw();

rotate = true;

}

public Point GetNextPoint()

{

Point p = GetHead();

switch (direction)

{

case Direction.LEFT: p.x -= step;

break;

case Direction.RIGHT: p.x += step;

break;

case Direction.UP: p.y -= step;

break;

case Direction.DOWN: p.y += step;

break;

}

return p;

}

public void Rotation(ConsoleKey key)

{

if (rotate)

{

switch (direction)

{

case Direction.LEFT: case Direction.RIGHT: if (key == ConsoleKey.DownArrow) direction = Direction.DOWN;

else if (key == ConsoleKey.UpArrow) direction = Direction.UP;

break;

case Direction.UP: case Direction.DOWN: if (key == ConsoleKey.LeftArrow) direction = Direction.LEFT;

else if (key == ConsoleKey.RightArrow) direction = Direction.RIGHT;

break;

}

rotate = false; } } }

**Вывод на экран**

class Game

{

static Snake snake;

static void Main()

{

snake = new Snake(x / 2, y / 2, 3);

Змейку двигается, считывает что нажимаем и передаёт в программу и происходит движение.

class Game

{

static void Main ()

{

while (true)

{

if (Console.KeyAvailable)

{

ConsoleKeyInfo key = Console.ReadKey ();

snake.Rotation(key.Key);

}

**Проверка врезалась ли змея в стену либо в саму себя**

class Walls

{

public bool IsHit (Point p)

{

foreach (var w in wall)

{

if (p == w)

{

return true;

} }

return false; }

class Snake

{

public bool IsHit (Point p)

{

for (int i = snake.Count - 2; i > 0; i--)

{

if (snake[i] == p)

{

return true;

} }

return false; }

**Если съела, удлиняем**

class Snake

{

public bool Eat (Point p)

{

head = GetNextPoint ();

if (head == p)

{

snake.Add (head);

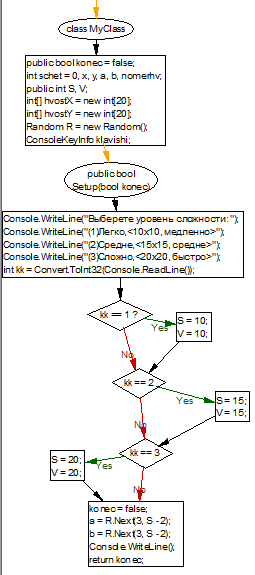
head.Draw ();

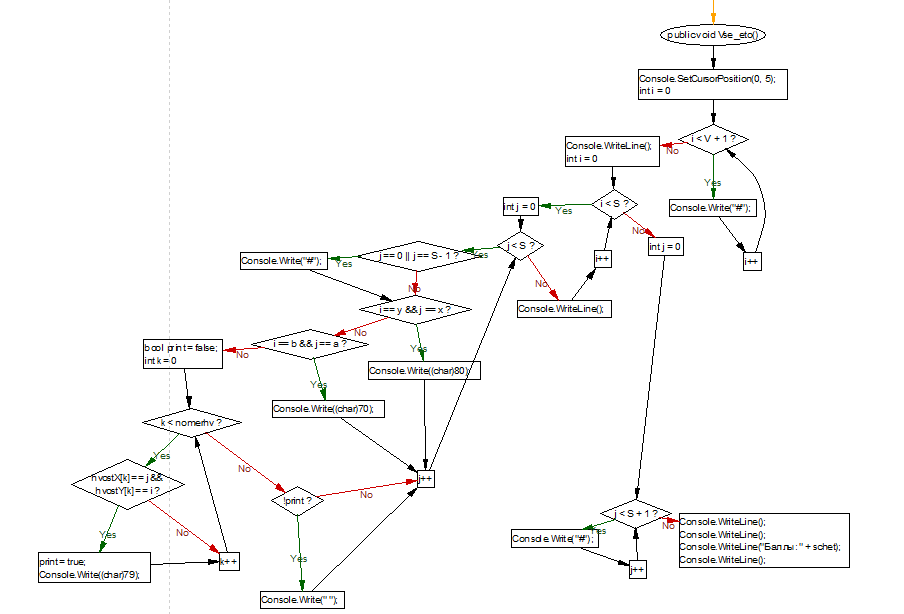
return true;

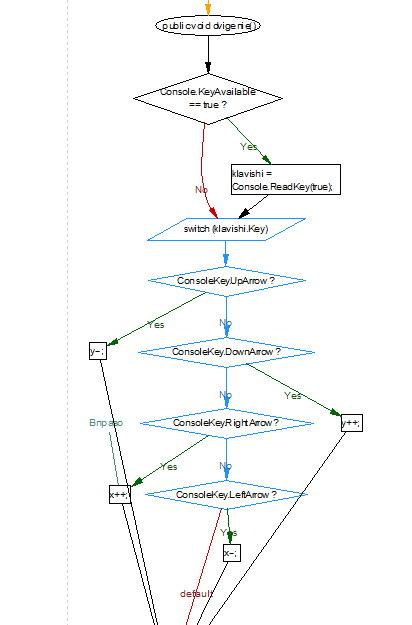
}

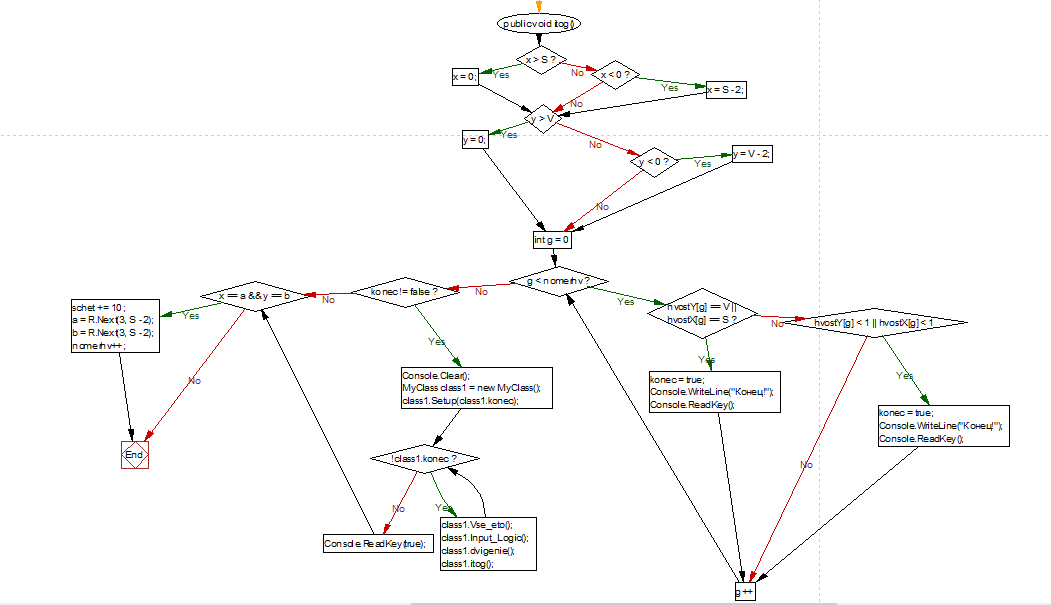
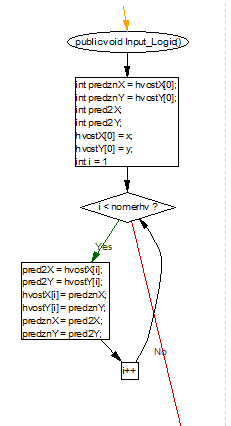
return false;

}





****

****