Департамент профессионального образования Томской области

Областное государственное бюджетное профессиональное образовательное учреждение среднего профессионального образования

**«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»**

Специальность «Информационные системы и программирование»

Отчёт о практической работе по дисциплине «Разработка кода информационных систем»

Тема: Создание консольного приложения “Snake”

Студент

«\_\_» \_\_\_\_\_\_\_\_ 2020 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Буторина Д.Л.

Преподаватель

«\_\_» \_\_\_\_\_\_\_\_ 2020 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Павлов В.С.

Томск 2020

Код:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ConsoleApp7

{

class Game

{

static readonly int x = 80;

static readonly int y = 26;

static Walls walls;

static Snake snake;

static FoodFactory foodFactory;

static Timer time;

static void Main()

{

Console.WriteLine("Нажмите 1, чтобы начать игру");

int n = 0;

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

if (n == 1)

{

int x1 = 20;

int y1 = 20;

walls = new Walls(x1, y1, '#');

snake = new Snake(x1 / 2, y1 / 2, 3);

foodFactory = new FoodFactory(x1, y1, '@');

}

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

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

Console.CursorVisible = false;

foodFactory.CreateFood();

switch (n)

{

case 1:

{

time = new Timer(Loop, null, 0, 300);

break;

}

case 2:

{

time = new Timer(Loop, null, 0, 250);

break;

}

case 3:

{

time = new Timer(Loop, null, 0, 200);

break;

}

}

while (true)

{

if (Console.KeyAvailable)

{

ConsoleKeyInfo key = Console.ReadKey();

snake.Rotation(key.Key);

}

}

}

static void Loop(object obj)

{

if (walls.IsHit(snake.GetHead()) || snake.IsHit(snake.GetHead()))

{

time.Change(0, Timeout.Infinite);

}

else if (snake.Eat(foodFactory.food))

{

foodFactory.CreateFood();

}

else

{

snake.Move();

}

}

}

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 static bool operator ==(Point a, Point b) =>

(a.x == b.x && a.y == b.y) ? true : false;

public static bool operator !=(Point a, Point b) =>

(a.x != b.x || a.y != b.y) ? true : false;

public void Draw()

{

DrawPoint(ch);

}

public void Clear()

{

DrawPoint(' ');

}

private void DrawPoint(char \_ch)

{

Console.SetCursorPosition(x, y);

Console.Write(\_ch);

}

}

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);

}

}

public bool IsHit(Point p)

{

foreach (var w in wall)

{

if (p == w)

{

return true;

}

}

return false;

}

}

enum Direction

{

LEFT,

RIGHT,

UP,

DOWN

}

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 x1, int y1, int length)

{

direction = Direction.RIGHT;

snake = new List<Point>();

for (int i = x1 - length; i < x1; i++)

{

Point p = (i, y1, '\*');

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 bool Eat(Point p)

{

head = GetNextPoint();

if (head == p)

{

snake.Add(head);

head.Draw();

return true;

}

return false;

}

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;

}

}

public bool IsHit(Point p)

{

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

{

if (snake[i] == p)

{

return true;

}

}

return false;

}

}

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();

}

}

}