using Microsoft.VisualBasic;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading.Tasks.Dataflow;

using static System.Runtime.InteropServices.JavaScript.JSType;

namespace ConsoleMinigame

{

internal class Program

{

ConsoleKeyInfo keyInfo = Console.ReadKey(false);

//class Colectables

//{

// //string[] fruits = { "lemon", "orange", "apple" };

//}

static void Main(string[] args)

{

Random random = new Random();

Fruit berry = new Fruit();

player player1 = new player();

Level lv1 = new Level();

Queue<int> X\_cord = player1.X\_cord;

Queue<int> Y\_cord = player1.Y\_cord;

lv1.size = 25;

int score = lv1.score\_points;

string[] status = { score.ToString() };

//int[] length = player1.snake\_length;

int n = lv1.size;

int[,] grid = new int[n, n];

int x = 3, y = 4;

var input = ConsoleKey.G;

bool fruit\_eaten = true;

BuildBorder(grid, n);

bool alive = true;

while (true)

{

Thread.Sleep(100 - (lv1.score\_points \* 2));

//

fruit\_eaten = lv1.fruit\_eaten;

if (lv1.fruit\_eaten == true)

{

int rate = 2;

int chance = random.Next(1, rate);

//fruit\_spawn(grid, berry.x, berry.x);

fruit\_spawn(grid, n, 1);

lv1.score\_points += 1;

status[0] = lv1.score\_points.ToString();

lv1.fruit\_eaten = false;

if (chance == rate)

{

fruit\_spawn(grid, n, 2);

}

}

Movement\_input(player1, grid, lv1);

//status[0] = player1.score\_points.ToString();

//TailSpawn(grid, player1);

Refresh(grid, n, status, player1, lv1);

//Console.WriteLine(status[0]);

}

}

static void Refresh(int[,] grid, int size, string[] status, object instance, object level)

{

player player = (player)instance;

Level Level = (Level)level;

Console.Clear();

Console.WriteLine($" points :{status[0]} ");

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

{

Console.WriteLine();

for (int j = 0; j < size; j++)

{

//if (grid[i, j] == 1 || grid[i, j] == 2)

//{

// Console.Write("██");

//}

/\*else\*/

if (grid[i, j] == -1)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.Write("██");

Console.ForegroundColor = ConsoleColor.White;

}

else if (grid[i, j] == 0)

{

Console.Write(" ");

}

else if (grid[i, j] < Level.score\_points && grid[i, j] > 0)

{

Console.Write("██");

}

else if (grid[i, j] == Level.score\_points)

{

//Console.ForegroundColor = ConsoleColor.Red;

Console.Write("██");

//Console.ForegroundColor = ConsoleColor.White;

}

else if (grid[i, j] == -2)

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.Write("██");

Console.ForegroundColor = ConsoleColor.White;

}

else if (grid[i, j] == 1)

{

Console.ForegroundColor = ConsoleColor.Yellow;

Console.Write("██");

Console.ForegroundColor = ConsoleColor.White;

}

}

}

}

static void Movement\_input(object instance, int[,] grid, object level)

{

player player = (player)instance;

Level Level = (Level)level;

Queue<int> X\_cord = player.X\_cord;

Queue<int> Y\_cord = player.Y\_cord;

int x = player.x;

int y = player.y;

bool fruit\_eaten = Level.fruit\_eaten;

//Console.WriteLine(x);

var input = ConsoleKey.P;

if (Console.KeyAvailable == true)

{

input = Console.ReadKey().Key;

}

string key = Level.LastKey;

switch (input)

{

case ConsoleKey.W:

key = "W";

break;

case ConsoleKey.D:

key = "D";

break;

case ConsoleKey.A:

key = "A";

break;

case ConsoleKey.S:

key = "S";

break;

default: break;

}

grid[player.y, player.x] = Level.score\_points;

switch (key)

{

case "W":

y--;

break;

case "D":

x++;

break;

case "A":

x--;

break;

case "S":

y++;

break;

default: break;

}

Length\_reduce(grid, player);

Level.fruit\_eaten = Check\_val(grid, fruit\_eaten, x, y, -1);

if (Check\_val(grid, fruit\_eaten, x, y, -2) == true)

{

Level.score\_points += 5;

}

Level.LastKey = key;

player.x = x;

player.y = y;

}

static void BuildBorder(int[,] grid, int size)

{

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

{

if (i == 0 || i == size - 1)

{

for (int j = 0; j < size; j++)

{

grid[i, j] = -1;

}

}

for (int j = 0; j < size; j++)

{

if (j == 0 || j == size - 1)

{

Console.WriteLine();

grid[i, j] = -1;

}

}

}

}

static bool Check\_val(int[,] grid, bool i, int cordinateX, int cordinateY, int value)

{

i = false;

if (grid[cordinateY, cordinateX] == value)

{ i = true; }

return i;

}

static void fruit\_spawn(int[,] grid, int n, int id)

{

Random random = new Random();

int x = random.Next(1, n - 2);

int y = random.Next(1, n - 2);

grid[x, y] = -1 \* id;

}

static void Length\_reduce(int[,] grid, object instance)

{

int size = grid.GetLength(0);

player player = (player)instance;

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

{

for (int j = 0; j < size; j++)

{

if (grid[i, j] > 0)

{

grid[i, j]--;

}

}

}

}

static void TailSpawn(int[,] grid,object instance)

{

player player = (player)instance;

foreach (int x in player.X\_cord) ;

{

int x = 0;

//player.X\_cord.ToArray().ToList().IndexOf(x);

x = player.X\_cord.Dequeue();

int y = 0;

y = player.Y\_cord.Dequeue();

grid[x, y] = 2;

}

//player1.X\_cord.Enqueue(player1.x);

//player1.Y\_cord.Enqueue(player1.y);

//foreach (int i in player1.X\_cord)

//{

// grid[player1.Y\_cord.Peek(), player1.X\_cord.Peek()] = 2;

//}

//player1.X\_cord.Dequeue();

//player1.Y\_cord.Dequeue();

}

}

}