Introduction to Game Programming

Console video game

Session 1 – Screen limits, colors and random numbers

Nacho Iborra

IES San Vicente

Index of contents

1.Introduction....3
2.Positioning text in console....4
3.Using colors....6
3.1.The ConsoleColor element....6
3.2.Using the color palette....6
4.Dealing with randomness....8

1. Introduction

In this first session we will focus on some concepts learnt during the first classes. Specially, we will practice with the *Console.WriteLine* instruction to print text in the console, and see how to control where to print this text. Besides, we are going to learn how to print text using different foreground and background colors.

To finish this session, we will also deal with an essential concept of game programming, which is randomness. Without it, video games would always behave in the same way, enemies will always move following the same path, objects will always appear in the same positions, and playing the same video game more than once would become a *déjà vu*. You will see later in this module some specific methods of getting random numbers, but here we will just have a simple approach.

2. Positioning text in console

Let's create a simple program that prints a 'C' character in the middle of the screen/console. Taking into account that the console has 80 columns and 24 rows, we can do this by just printing 11 blank lines, and then printing 40 whitespaces and the desired character:

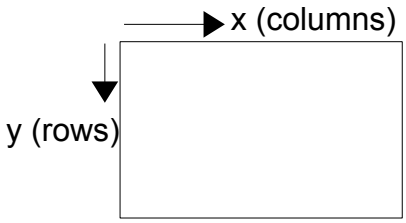
```
using System;


public class MyGame
{
      public static void Main()
      {
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine();
            Console.Write("                                        ");
            Console.WriteLine("C");
      }
}
```

However, we can do this code easier if we use an instruction called *SetCursorPosition*. For instance, if we put this instruction in a program:

```
Console.SetCursorPosition(5, 10);
```

It will place the cursor at row number 10 and column number 5 (or x = 5, y = 10). Take into account that the coordinate system in console works like this:

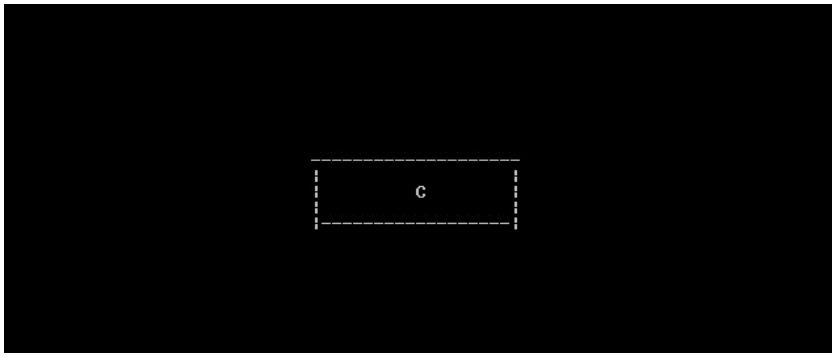


Exercise 1

Modify the initial exercise, and replace all the *Console.WriteLine* instructions with a *Console.SetCursorPosition* that places the cursor in the appropriate position (column 40, row 12), and then just one *Console.WriteLine* instruction that prints "C" in that place.

Exercise 2

Extend previous exercise by printing a rectangle in the middle of the screen. This rectangle must go from x = 30, y = 10 to x = 49, y = 14 (inclusive). You must have something like this when running the program:



3. Using colors

3.1. The ConsoleColor element

If you take a look at the official C# documentation, you can find an element called *ConsoleColor*. It is an enumeration of available colors for the console. It lets you work with a reduced color set, that we can use to either print text or highlight cells in the console. The complete color list is as follows:

3.2. Using the color palette

In order to use this color palette, we can use two instructions from our *Console* element:

```
Console.ForegroundColor = ConsoleColor.Red
```

sets the color for writing text (in this case, it is set to *Red* color)

```
Console.BackgroundColor = ConsoleColor.DarkBlue
```

sets the color for the background (in this case, it is set to *DarkBlue*)

If we want to recover the original (default) background and foreground colors, we can use this instruction:

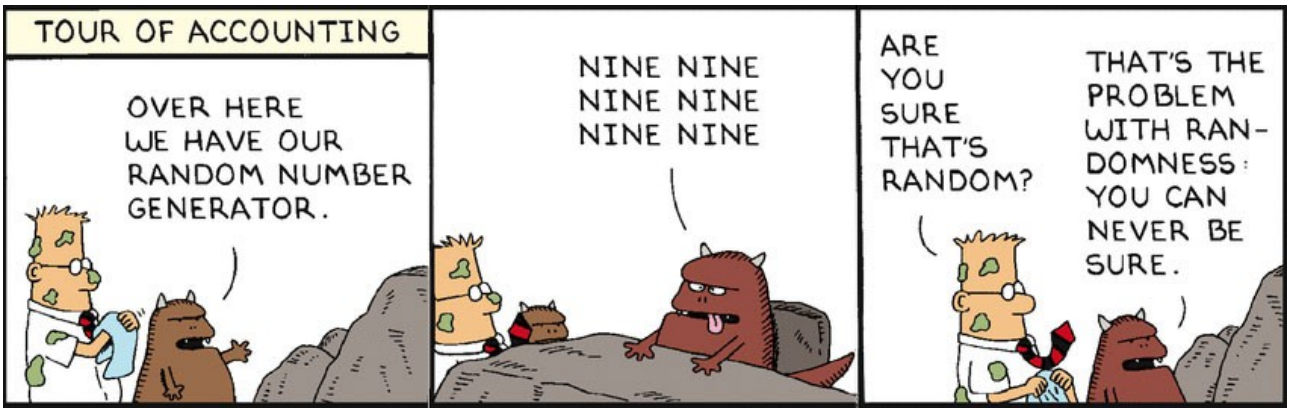
```
Console.ResetColor();
```

Exercise 3

Now, we are going to update previous exercise. In this case, we are going to draw the basic structure of our video game, with a right panel with some information, and a left

panel to play the game. Use different colors and try to adjust the appropriate text positions to get something like this:

4. Dealing with randomness



As said before, randomness is a very important concept in video game programming, since it lets us generate different situations, and different movement paths or evolutions, each time we run the game.

We will learn later that there are some special instructions to generate random elements, but for now, we will use a very simple instruction from C# language to generate random numbers:

```
int millisec = DateTime.Now.Millisecond;
```

Everytime this instruction is executed, it will take the current number of milliseconds. As this is a number that changes continuously, it will be different in every new execution of the instruction.

Let's go back to our video game. Comment the lines written in previous exercises, and check out this code:

```
using System;


public class MyGame
{
      public static void Main()
      {
            // … Previous, commented code


            int millisec = DateTime.Now.Millisecond;
            Console.WriteLine(millisec);
      }
}
```

Note that, everytime you run the program, you get a different number. That can help us introduce some randomness in our applications.

Let's do it. Let's write a ball (character 'o') in a random place of the console. How could we do this? The instruction *DateTime.Now.Millisecond* gets a number between 0 and 999, and our console just lets us move from 0 to 79 (for X or columns) and from 0 to 23 (for Y or

rows). Even more, we are going to play in just half of the screen (X from 0 to 39). We need a way to limit the randomness to these ranges. To do this, we can use the % operator. For instance, if we want to generate a number from 0 to 4 (inclusive), we can do it by getting the module 5 of the generated number:

```
int randomNumber = DateTime.Now.Millisecond % 5;
```

And what if we want to generate a number from 1 to 5 instead of getting it from 0 to 4? We can just add 1 to previous instruction:

```
int randomNumber = 1 + DateTime.Now.Millisecond % 5;
```

Exercise 4

Uncomment the previously commented lines in the program before, and add code to write a ball 'o' in a random position within the left half of the screen. At the end, you must have something like this (the ball position may be different):