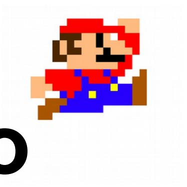
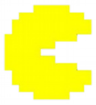
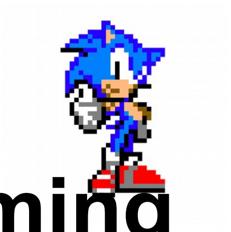
Introduction to Game Programming







Console video game

Session 2 – The game loop and user input

Nacho Iborra

IES San Vicente

Index of contents

1.Introduction.. 3
2.The game loop.. 4
3.Getting user input.. 6
  3.1.Improving the user input..7
4.Partial redrawings.. 9
  4.1.Erasing old positions.. 10

1. Introduction

In this session we will cover two different concepts. Firstly, we will see the general structure of (almost) every game loop, and the different operations that must be performed during each iteration. We will apply this loop to our video game in its current state.

Then, we will see how to take information from the user (this is, movement orders), and apply them to an object of the screen. Finally, we will see how to improve these concepts so that the behavior of the video game will be much more fluid.

2. The game loop

The main part of a game is a loop that repeats a list of actions. These actions are (normally):

1. Draw elements on the screen. In some games, you will need to "delete" previous screen to draw the new content. In C# console applications, this can be done with the instruction:

   ```
   Console.Clear();
   ```

2. Check if the user has introduced any input (key pressed, gamepad moved, mouse clicked...)
3. Move enemies and any other object of the scene that needs to be moved
4. Check collisions between objects and characters, to check if we have collided with an enemy, a shot, a wall... or if we have collected an item, for instance. With these tests, we can update points, lives and any other information about the game state.
5. Apply a pause after each iteration, so that the videogame does not run very fast on newer computers

These steps repeat while the game is running (i.e. the number of lives is greater than 0, we have not reached the end of the game yet and we have not exited the game in any way).

Let's implement a first approach to this game loop. To simplify it, we will repeat the loop indefinitely, and we will leave most of the tasks empty, to be completed later when we get more knowledge about C# programming.

```
using System;

public class MyGame
{
    public static void Main()
    {
        // Declare variables here

        // Loop repeats indefinitely
        while (1 == 1)
        {
            // 1. Draw elements. In this place we will put the code
            // written in previous session for now.
            Console.Clear();
            Console.SetCursorPosition(...);
            …

            // 2. Read input and calculate player's new position
            // TO DO
```

```
                    // 3. Move enemies and other objects
                    // TO DO

                    // 4. Check collisions and update game state
                    // TO DO

                    // 5. Pause
                    // TO DO
              }
       }
}
```

Exercise 1

Update the class *MyGame* from previous sessions with a loop like the one seen in previous example. Put the code that you have already written previously in the step 1 of the loop (declare all the variables needed before the game loop).

NOTE: if you run the game with these changes, it will never end. You will need to close the console manually to finish it. We will see later how to set another end conditions.

NOTE 2: you will also note a blinking when running the program. It is normal, since we clear and redraw the whole console after each iteration. We will see later in this session how to avoid this blinking in console games.

3. Getting user input

Now, we are going to get the user input and move the main character according to what user has just typed. We will use the keys A and D to move the character to the left and right respectively.

First of all, we need to specify the coordinates of our main character, and draw it in the console. To do this, we are going to define two "int" variables called "characterX" and "characterY", and place the bar in X = 18, Y = 23:

```
int characterX = 19, characterY = 23;
```

Then, we print the main character in the specified coordinates. As it is going to be a bar, we need to print some whitespaces (five, for instance). Do this in step #1 of the game loop.

```
            while (1 == 1)
            {
                  // 1. Draw elements. In this place we will put the code
                  // written in previous session for now.

                  …

                  // Print main character
                  Console.SetCursorPosition(characterX, characterY);
                  Console.BackgroundColor = ConsoleColor.White;
                  Console.Write("     ");
                  Console.ResetColor();
```

Now let's move to step number 2 of the game loop (get user input), and fill it with something like this:

```
using System;

public class MyGame
{
      public static void Main()
      {
            int characterX = 19, characterY = 21;
            string key;
            … // Other variables declared previously

            // Loop repeats indefinitely
            while (1 == 1)
            {
                  // 1. Draw elements. In this place we will put the code
                  // written in previous session for now.
                  …

                  // 2. Read input and calculate player's new position

                  key = Console.ReadLine();
                  if (key == "a")
                        characterX--;
                  if (key == "d")
                        characterX++;

                  // 3. Move enemies and other objects
                  // TO DO

                  // 4. Check collisions and update game state
                  // TO DO

                  // 5. Pause
                  // TO DO
            }
      }
}
```

We have also defined a variable called *key* to store the text written by the user, and two integer variables (*characterX* and *characterY*) to store the current position of the main character.

Exercise 2

Update your game program with these instructions to move the main character. Remember to link the character position with the *characterX* and *characterY* variables, so that, if you change the value of these variables, the position of the character will be automatically updated.

NOTE: to read the user input in every iteration, set the cursor position to X = 0, Y = 24 so that you will not type anything within the game board.

3.1. Improving the user input

Having to press Enter after each input is not very friendly for the user. We can improve it if we access the keyboard directly, with *ConsoleKeyInfo* object. Just replace the *key* variable with a new variable of type *ConsoleKeyInfo*, and the new lines written in step 2 in previous example with these lines below. Besides, we will use the arrow keys instead of A-D keys to move the character:

```
using System;

public class MyGame
{
      public static void Main()
      {
            int x, y;
            ~~string key~~;              // Delete this variable
            **ConsoleKeyInfo key;**
            … // Other variables declared previously

            // Loop repeats indefinitely
            while (1 == 1)
            {
                  // 1. Draw elements. In this place we will put the code
                  // written in session 01 for now.

                  …

                  // 2. Read input and calculate player's new position

                  ~~key = Console.ReadLine()~~;            // Delete this line
                  **key = Console.ReadKey(false);**
                  ~~if (key == "a")   characterX--~~;     // Delete these lines
                  ~~if (key == "d")   characterX++~~;
                  **if (key.Key == ConsoleKey.RightArrow)**
                        **characterX++;**
                  **if (key.Key == ConsoleKey.LeftArrow)**
                        **characterX--;**
                  // 3. Move enemies and other objects
                  // TO DO

                  // 4. Check collisions and update game state
                  // TO DO

                  // 5. Pause
                  // TO DO
            }
      }
}
```

Exercise 3

Update the game by changing the code to move the character typed in exercise 2 with the new code shown in previous example.

4. Partial redrawings

In graphical environments, the graphics engine automatically clears the screen and redraws the components in it, but we don't see any blinking because of the graphics capabilities.

Unfortunately, we can't say the same when working with console video games, since console is not designed for quick redrawing, and everytime we clean it to redraw the elements, we see how it blinks.

But we have a solution to avoid this blinking. We must pay attention at which elements of our video game are changing at each iteration, and redraw only these elements.

In our case, only the main character (and the ball, later) change their position after each iteration, so we can take the rest of the code outside the game loop. We will get something like this:

```
using System;

public class MyGame
{
      public static void Main()
      {
            // Variable declarations
            // Draw right half and texts

            while (1 == 1)
            {
                  // 1. Draw movable elements (character and ball)

                  // 2. Read input and calculate player's new position

                  // 3. Move enemies and other objects
                  // TO DO

                  // 4. Check collisions and update game state
                  // TO DO

                  // 5. Pause
                  // TO DO
            }
      }
}
```

Exercise 4

Update the code with these changes and test it. You will see that the screen no longer blinks. However, old positions of the main character are not erased.

4.1. Erasing old positions

The final step is to erase the character's old position before moving it to the new one. To do this, we have to enter these instructions BEFORE changing the coordinates of the character:

1. Set cursor position to character's current X and Y
2. Print as many white spaces as character´s length (with black background, in our case)

Then, we can change the corresponding coordinates, and in the next iteration the character or enemy will be placed in its new position.

Exercise 5

Add these instructions before updating the position of the character, and see how it moves without leaving a trail behind it.