Department of Computer Science

08101 Programming 1

Week 7 Practical 2007/2008

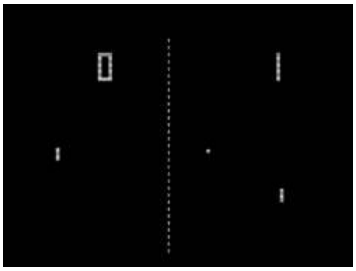
Pong

For the rest of this semester we are going to develop a version of the Pong game. The game will be developed in a number of phases. At each point we will get a different part of the game to work and at the end we should have a playable version of the game.

This work will be an assessed part of the course. You will be required to submit the program that you write and you will also be required to demonstrate the game working. You will be told exactly what behaviours are required to get a good mark in the work.

If you want to add to the specification, or even produce a totally different version of the game this is perfectly OK and may even earn you extra marks; but you will still be required to submit a character version of the game in addition to your improved version. You should also remember that it is not possible to earn more than 100% for the practical work, so you must not spend so much time on the game that work on other modules suffers.

The Game of Pong



Pong is a computer game which has been around for very many years, it was released by Atari in 1972 and the word Pong is actually an Atari trademark. You can argue that was the first ever video game to achieve widespread popularity.

The game is played by two players, each of which controls the position of their paddle. In the original game this was achieved by the use of rotary controls, but we will be using keys on the keyboard. Players must use their paddles to hit a ball which is continuously bouncing around the screen. If a player misses the ball, and it hits the wall behind their paddle their opponent scores a point and the game continues. The game can be played until one of the players reaches a particular score, or for a set period of time.

Drawing a Bouncing Ball

This week we are going to concentrate on creating a ball, making it move around the screen, and bounce off the edges. At the end of this practical you should have a ball which bounces around the screen.

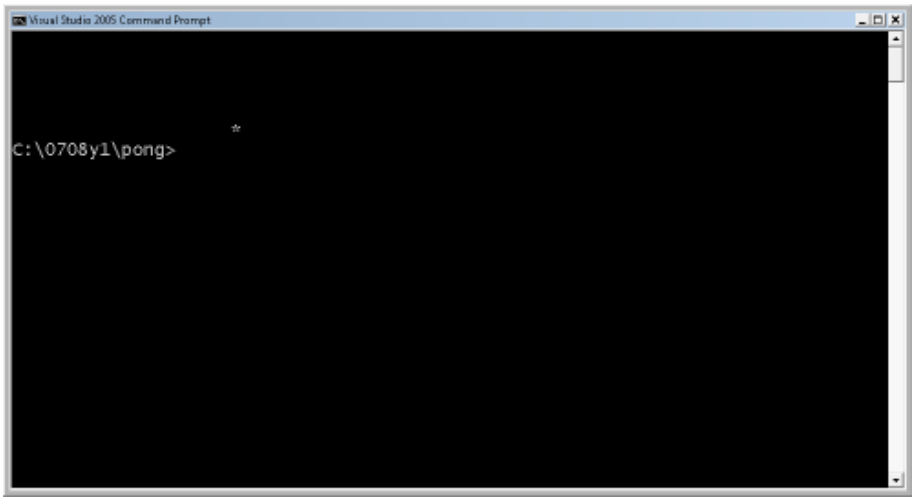
The Console Window

The game that we are going to create will run on the console window that we have used for all our programs. The window has a width of 80 characters and a height of 25 lines, which is adequate for a reasonable game. We can use methods in the Console class to allow us to control the drawing position (cursor) in this window so that we can draw the game elements at the required positions.

```
using System;
class PongGame
{
    static void Main ()
    {
        Console.Clear();
        Console.SetCursorPosition(20, 5);
        Console.Write("*");
    }
}
```

Pong01.cs

If you run the program above you will find that the screen is cleared and the * is drawn twenty characters across from the left margin and five lines down from the top of the screen:



The origin (i.e. the place corresponding to the cursor position 0,0) is the top left hand corner of the screen. If you increase X you move towards the right. If you increase Y you move down the screen. We will animate our ball by repeatedly drawing it at different positions on the screen, giving the appearance of movement.

Before you go any further; perform the following:

1. Run the above program.
2. Change the values in the **`SetCursorPosition`** method call and note the effect on the program. Try putting values outside the valid range (x 0-25, y 0-79) and note what happens.

Moving the Ball

When we are playing pong the ball must move over the surface of the screen in a particular direction.

- If the ball is moving to the right the X value will be increasing
- If the ball is moving to the left the X value will be decreasing
- If the ball is moving up the Y value will be decreasing
- If the ball is moving down the Y value will be increasing

To make a ball move around the screen we need to keep track of its position and update this each time the game loop goes around:

```
using System;

class PongGame
{
    static void Main ()
    {
        int x=0;
        int y=0;
        Console.Clear();
        while(true)
        {
            Console.SetCursorPosition(x, y);
            Console.Write("*");
            x = x + 1;
            y = y + 1;
            System.Threading.Thread.Sleep(500);
        }
    }
}
```

Pong02.cs

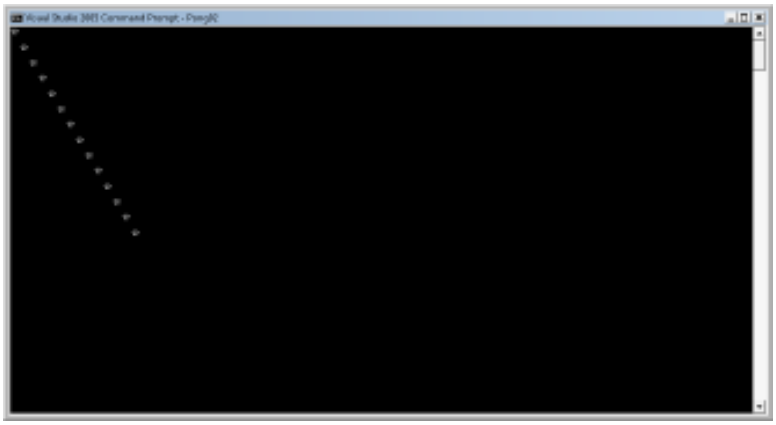
This program loops around drawing stars. Each time round the loop the x and y positions are increased by 1, causing the star to be drawn at a different position each time.

The program runs forever, which means that when you run it you will need to use the CTR+C (hold down the control key and press C) combination to stop the program running.

Note also that I have added a delay statement so that we can actually see the program results. If the program ran at full speed the * would quickly vanish off the screen. The delay is provided by the line:

```
System.Threading.Thread.Sleep(500);
```

This calls a "sleep" method on the program which causes it to pause for 500 milliseconds. A millisecond is a thousandth of a second, which means that the line above makes the program pause for half a second. This is probably a bit slow for the game, but it allows us to see what is happening. If you run the program a diagonal line of * characters are drawn down the screen:



Before you go any further; perform the following:

3. Run the above program. Note how the * characters are displayed.
4. Change the line **`y = y+1`** to **`y = y+0`**. Note the effect on the movement.
5. Change the **`500`** in the line **`Thread.Sleep(500)`** to the value 100. Note the effect on the movement.

Animating the Ball

To get the effect of movement we need to erase the old ball position before we draw the new one. If we do this rapidly enough it will appear to the player that it is moving down the screen. Many video games do this by clearing the entire screen before redrawing everything. We are not going to do this, instead we are going to erase the old ball and then redraw the ball in its new position:

```
using System;

class PongGame
{
    static void Main ()
    {
        int x=0;
        int y=0;
        int xSpeed = 1;
        int ySpeed = 1;
        Console.Clear();
        while(true) {
            Console.SetCursorPosition(x, y);
            Console.Write(" ");
            x = x + xSpeed;
            y = y + ySpeed;
            Console.SetCursorPosition(x, y);
            Console.Write("*");
            System.Threading.Thread.Sleep(500);
        }
    }
}
```

Pong03.cs

This program displays a "*" character moving down the screen. It does this by performing a sequence of erase old version, update position and redraw. This is the fundamental method by which games work. When you are playing a game it is repeatedly updating the display and the position of the components in the game environment.

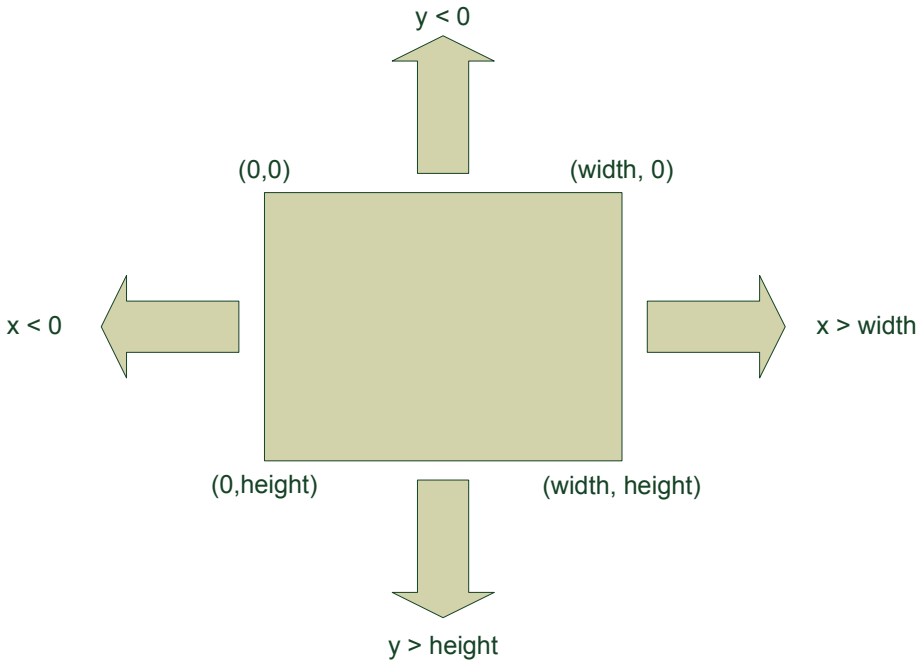
Note that it also adds **`xSpeed`** and **`ySpeed`** variables to keep track of the direction of movement of the ball. By changing these we can control how the ball moves.

Before you go any further; perform the following:

6. Run the above program. Note how the * characters are displayed.
7. Change the line **`y = y+1`** to **`y = y+0`**. Note the effect on the movement.
8. Change the **`500`** in the line **`Thread.Sleep(500)`** to the value 100. Note the effect on the movement.

Making the Ball Bounce

At the moment the ball doesn't react when it reaches the edge of the screen, instead it just keeps travelling. We need to fix this, so that the ball will bounce around.



The diagram shows the ways in which the ball can leave the screen. If the y value becomes larger than the height of the screen the ball will go off the bottom of the screen (remember that when y is 0 the ball is at the top of the screen).

When the ball reaches the edge of the screen we must reverse the appropriate direction of movement, if the ball gets to the left or right hand edges we need to reverse the speed in x, if the ball gets to the top or bottom we need to reverse the speed in y.

The standard console window is 80 characters wide and 25 characters high. So if the x value ever reaches 79 (remember that we start counting at 0) we need to reverse the speed in the x direction.

To reverse the speed we just need to multiply the speed by minus one.

```
if ( ( x == 79 ) || ( x == 0 ) )
{
    xSpeed = xSpeed * -1;
}
```

Remember that the || operator is logical or. If either of the conditions is true the program must reverse the direction of movement in

Before you go any further; perform the following:

9. Add code to your pong program to cause the ball to bounce in X and Y.

Detecting an Exit key

At the moment we have to exit the game by stopping the program. This is not good. We need to add some extra code so that the player can end the game by pressing a key on the keyboard. We will need to detect key presses later when we move paddles around, but we are going to start by exiting the game loop when the user presses the Escape key.

Detecting a keypress

To control the game we need to know when the user has pressed one of the keys on the keyboard. The **`Console`** class provides us with a property which returns **`true`** if a key has been pressed:

```
if (Console.KeyAvailable)
{
     // if we get here a key has been pressed
}
```

In the code above the statements in the block will be obeyed if a key has been pressed. If no key has been pressed the program will just continue.

Reading the keyboard.

If we have a key available we can read it from the keyboard buffer. The information about a key is provided in the form of a structure which contains information including the key which was pressed.

There is a special enumerated type, **`ConsoleKey`**, which contains a range of values which represent printable keys (like A, B or full stop) and also non-printable keys (like Down Arrow or Escape). We can feed a switch construction the value of the key pressed and use this to select appropriate statements which will update the direction of movement. Our game loop is now:

1. Erase the old game display
2. Check the keyboard to get the player moves
3. Update the position of the game elements
4. Check for the end of the game
5. Draw the game elements
6. Pause for a while
7. Go to step 1

```
using System;

class PongGame
{
    static void Main ()
    {
        int x=0;
        int y=0;
        int xSpeed = 1;
        int ySpeed = 1;
        bool gameRunning = true;
        Console.Clear();
        while(gameRunning) {
            // Erase the old game elements
            Console.SetCursorPosition(x, y);
            Console.Write(" ");
            x = x + xSpeed;
            y = y + ySpeed;

            // Get the player move
            if (Console.KeyAvailable)
            {
                ConsoleKeyInfo keyInfo =
                    Console.ReadKey(true);
                switch (keyInfo.Key)
                {
                  case ConsoleKey.Escape:
                  gameRunning = false;
                  break;
                }
            }

            // Draw the new game elements
            Console.SetCursorPosition(x, y);
            Console.Write("*");
            if ( ( x == 79 ) || ( x == 0 ) )
            {
                xSpeed = xSpeed * -1;
            }

            System.Threading.Thread.Sleep(100);
```

```
            }
        }
}
```

Pong04.cs

The code above implements part of the game, in that the player can exit the game by pressing the Escape key. Note that the loop is now controlled by a bool variable, gameRunning, which is set to false to stop the loop and end the game. However, it does not properly bounce the ball.

Before you go any further; perform the following:

10. Improve your version of the program so that a player can exit the game by pressing the Escape key.

Adding Colour

At the moment the background to our game is black and the ball is just a white character. Fortunately the **`Console`** class provides us with a way we can change these colours and make our game more visually interesting:

```
Console.ForegroundColor = ConsoleColor.Yellow;
Console.BackgroundColor = ConsoleColor.Green;
```

The **`ForegroundColor`** (note the American spelling) property of the console sets the colour of text that is printed. **`ConsoleColor`** is an enumerated type with a range of values for the colours which are available.

The **`BackGroundColor`** property sets the colour of the background for any text that is written on the screen. It also sets the colour which is used to fill the screen when it is cleared.

These properties set the colour for all subsequent character write operations. This means that you can draw multiple colours (perhaps green for one bat and blue for the another) by selecting the required colour before each draw operation. This also works for the background of characters.

Before you go any further; perform the following:

11. Add a bit of colour to your game. But please don't spend too much time on this. In a real game development you would have some graphic designers who would decide on these aspects of the game.

Pong Phase 1

At the end of this session you should have game in which a ball bounces around the screen. Next time we will be adding the bats.

Rob Miles November 2007