Introduction to Game Programming

Session 4 – Using arrays and structures

Nacho Iborra

IES San Vicente

Index of contents

1.Introduction....3
2.The brick structure....4
3.The bricks array....5
4.Collisions with bricks....7
4.1.Updating the score....7
5.To think a little....9

1. Introduction

In this session we are going to introduce structures and arrays into our video game project, to store the information of all the bricks. First of all, we will define the fields of the "brick" structure, and then we will learn how to place and print a brick in the screen according to its information. Then, we will create an array of bricks and explore it to print all the bricks of the scene.

Finally, we will check the collisions between the ball and the bricks, and learn how to destroy a brick from the scene so that the ball will never collide with it again. Besides, every time we destroy a brick, we will get some points for our score.

2. The brick structure

We need to store the following information about each brick in the scene:

- *x* and *y* coordinates (`byte`)
- The number of hits needed to destroy the brick;
- Brick *color* (`ConsoleColor`)
- A flag indicating if this brick has been destroyed or not (`bool`)
- The score that we get when destroying the brick (`byte`)

So we can define this structure at the beginning of our source code:

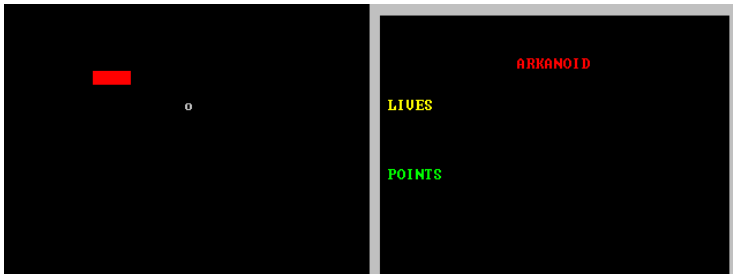
```
public class MyGame
{
    struct brick
    {
        public byte x;
        public byte y;
        public byte numberOfHits;
        public ConsoleColor color;
        public bool destroyed;
        public byte score;
    }
```

We can also try to draw a single brick in the scene. For instance:

```
// Print brick (test)
brick testBrick;
testBrick.x = 10;
testBrick.y = 5;
testBrick.numberOfHits = 1;
testBrick.color = ConsoleColor.Red;
testBrick.destroyed = false;
testBrick.score = 10;
Console.SetCursorPosition(testBrick.x, testBrick.y);
Console.BackgroundColor = testBrick.color;
Console.Write("    ");
Console.ResetColor();

// Print right panel
...
```

Note that we use 4 whitespaces to draw the brick, so each brick will be 4 spaces width.



Exercise 1

Define the *brick* structure shown above, and a test brick drawn wherever you want, with your preferred color.

3. The bricks array

Now that we have learnt how to define and draw a brick, we can remove the lines written before to draw a test brick:

```
// Print brick (test)

// Remove all lines in this block of code, until you reach next comment

// Print right panel
...
```

Instead of this, we are going to define an array of bricks, where each brick will have its own coordinates, color and score.

```
// Bricks
brick[] bricks = new brick[10];

bool exitGame = false;
...
```

We are going to distinguish between three different types of bricks:

- Red bricks, 1 hit, with a score of 10 points
- Blue bricks, 1 hit, with a score of 20 points
- Yellow bricks, 2 hits, with a score of 50 points

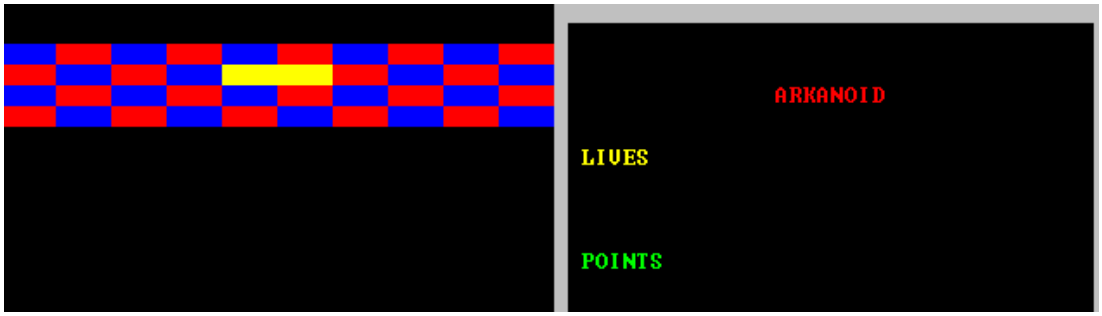
For instance, we can define an array of one yellow brick at each bound (left and right), followed by a blue brick and then 6 red bricks in the middle:

```
// Create bricks
for (int i = 0; i < bricks.Length; i++)
{
    bricks[i].x = (byte)(i*4);
    bricks[i].y = 2;
    bricks[i].destroyed = false;
    if (i == 0 || i == bricks.Length - 1)
    {
        bricks[i].numberOfHits = 2;
        bricks[i].color = ConsoleColor.Yellow;
        bricks[i].score = 50;
    } else if (i == 1 || i == bricks.Length - 2) {
        bricks[i].numberOfHits = 1;
        bricks[i].color = ConsoleColor.Blue;
        bricks[i].score = 20;
    } else {
        bricks[i].numberOfHits = 1;
        bricks[i].color = ConsoleColor.Red;
        bricks[i].score = 10;
    }
}

// Draw bricks
for (int i = 0; i < bricks.Length; i++)
{
    Console.SetCursorPosition(bricks[i].x, bricks[i].y);
    Console.BackgroundColor = bricks[i].color;
    Console.Write("    ");
}
```

Exercise 2

Define an array of 40 bricks, and use a "for" structure to fill it so that, when you draw them in the screen, you get something like this:



4. Collisions with bricks

To finish with this session, let's implement a collision detection algorithm to check if the ball collides with any brick. We must take into account four possible collisions, depending on whether the ball collides the upper, lower, left or right border of the brick.

- If the ball collides the upper or lower border of the brick, we need to change the ball's Y direction.
- If the ball collides the left or right border of the brick, then we change the ball's X direction.

This would be, for instance, the code to check a collision with the lower border of a brick:

```
if (ballY == bricks[i].y + 1 && ballYDirection < 0 &&
    ballX >= bricks[i].x && ballX < bricks[i].x + 4)
{
    bricks[i].numberOfHits--;
    ballYDirection = -ballYDirection;
}
```

Note that we check if:

- Ball Y coordinate is just below the brick
- Ball Y direction is upwards (negative)
- Ball X coordinate is between brick's X and brick length

If the ball collides with a brick, we need to check if the brick must be destroyed, and if so, destroy it:

```
// Check if brick must be destroyed, then destroy the brick
if (bricks[i].numberOfHits == 0)
{
    bricks[i].destroyed = true;
    // Remove brick from the scene (erase it)
    ...
}
```

Exercise 3

Implement the collision detection algorithm for all the sides of each brick of the array, so that the bricks will be destroyed properly and the ball will bounce against them when it collides.

4.1. Updating the score

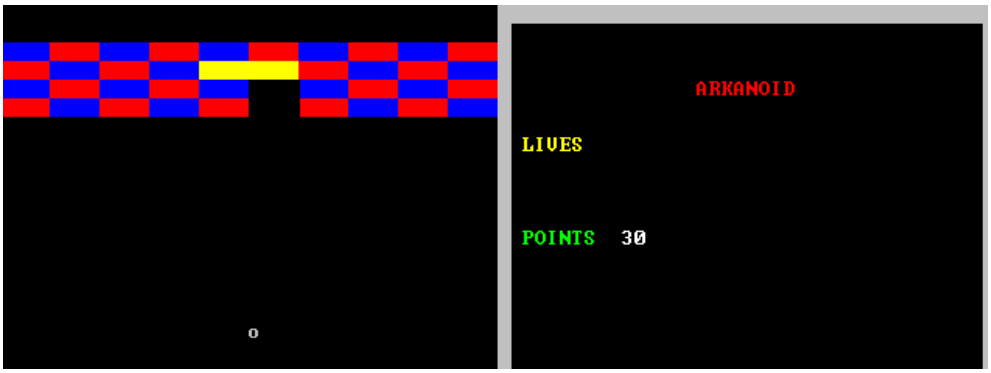
Every time we destroy a brick, we must add a score to our total score. To do this, we need to define a variable to store this total score:

```
bool exitGame = false;
int totalScore = 0;
```

Then, whenever we destroy a brick, we update this value and print it in the corresponding coordinates:

```
if (bricks[i].numberOfHits == 0)
{
    bricks[i].destroyed = true;
```

```
totalScore += bricks[i].score;
Console.SetCursorPosition(50, 12);
Console.ForegroundColor = ConsoleColor.White;
Console.Write(totalScore);
Console.ResetColor();
...
```



Exercise 4

Add this score to your game so that it will be increased every time we destroy a brick.

5. To think a little...

To finish with this session, think about the advantages of defining another structure for the ball features: X and Y coordinates, X and Y directions and X angle. If you store all this information inside a *struct*, you don't need to manage all these variables separately.

Exercise 5

Add a new *struct* to the project to store all the information about the ball. Call this structure *ball*, and replace every single variable within the code by this structure and its corresponding fields.