MonoGame Blocks Tutorial Part 4

Contents

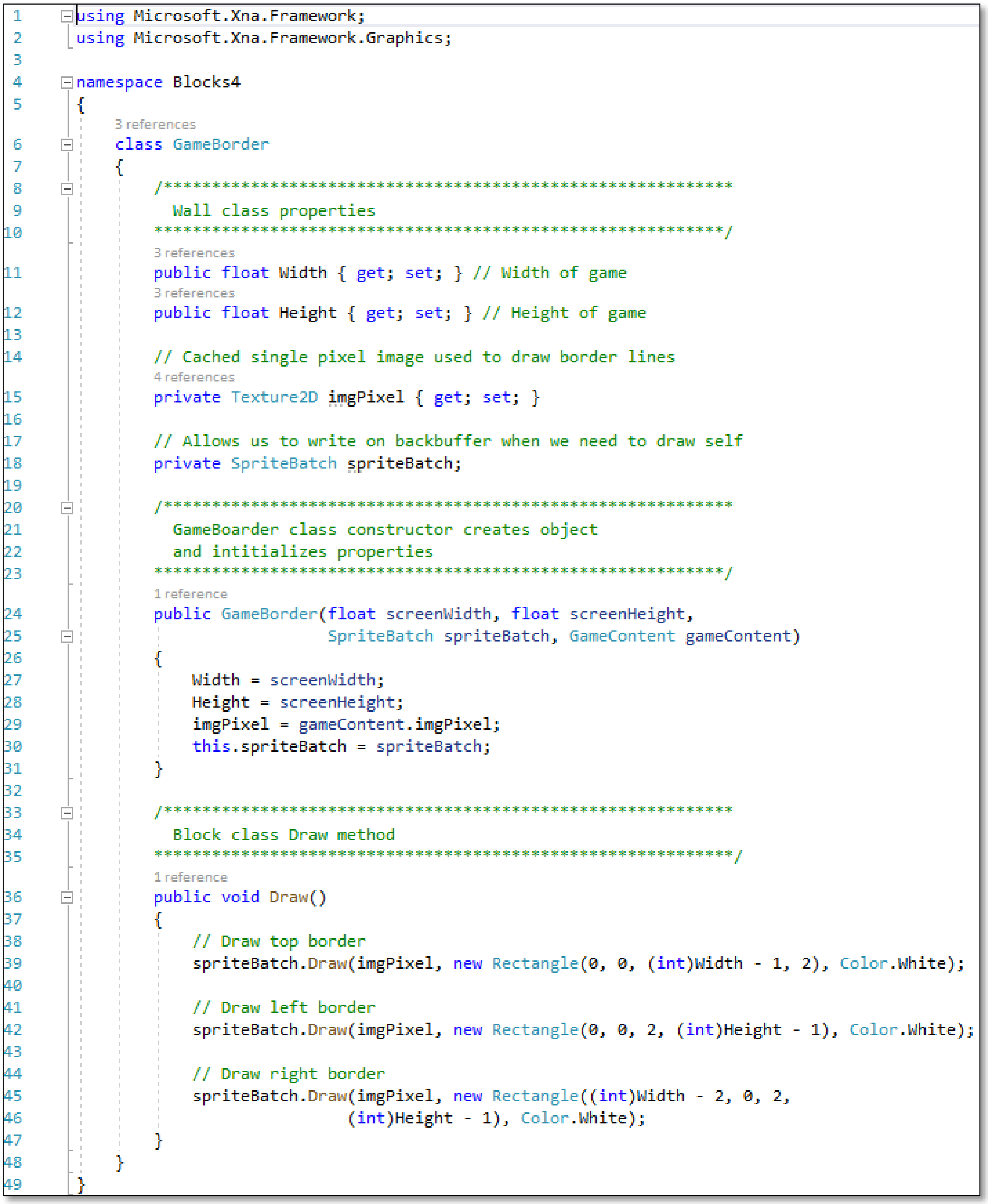
Part 4 – The Game Border

Time required: 30 minutes

In the last section, we began drawing the game playing field by adding the paddle and block wall. Now we are ready to draw the borders on the edge of the playing field.

We could just create images of our game borders, but I wanted to show how you can draw basic shapes in MonoGame. So, we'll draw the borders as 1 pixel-wide rectangles. We'll add a new `GameBorder` class for this.

1. Right-click on the project **Blocks** in the solution explorer. Select **Add → Class** from the drop-down menus.
2. In the **Add New Item** Dialog, enter "GameBorder.cs" for the class name. Click **Add**.
3. Enter the following code in "GameBorder.cs".



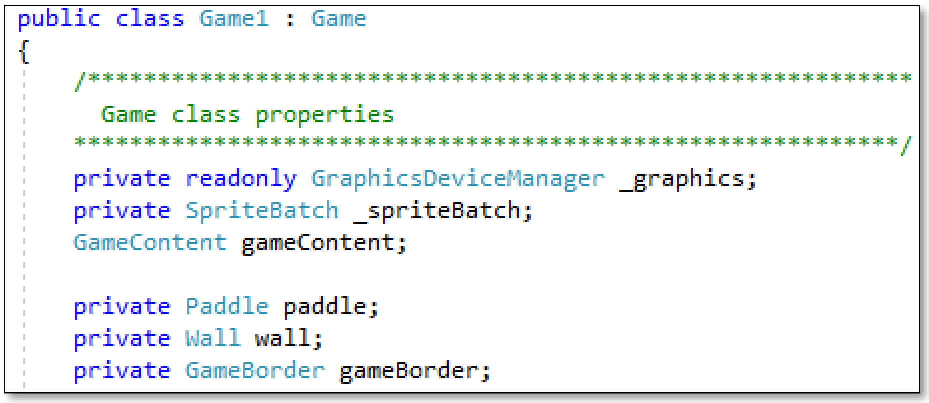
```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace Blocks4
{
    3 references
    class GameBorder
    {
        /************************************************************
          Wall class properties
        ************************************************************/
        3 references
        public float Width { get; set; } // Width of game
        3 references
        public float Height { get; set; } // Height of game

        // Cached single pixel image used to draw border lines
        4 references
        private Texture2D imgPixel { get; set; }

        // Allows us to write on backbuffer when we need to draw self
        private SpriteBatch spriteBatch;

        /************************************************************
          GameBoarder class constructor creates object
          and intitializes properties
        ************************************************************/
        1 reference
        public GameBorder(float screenWidth, float screenHeight,
                          SpriteBatch spriteBatch, GameContent gameContent)
        {
            Width = screenWidth;
            Height = screenHeight;
            imgPixel = gameContent.imgPixel;
            this.spriteBatch = spriteBatch;
        }

        /************************************************************
          Block class Draw method
        *************************************************************/
        1 reference
        public void Draw()
        {
            // Draw top border
            spriteBatch.Draw(imgPixel, new Rectangle(0, 0, (int)Width - 1, 2), Color.White);

            // Draw left border
            spriteBatch.Draw(imgPixel, new Rectangle(0, 0, 2, (int)Height - 1), Color.White);

            // Draw right border
            spriteBatch.Draw(imgPixel, new Rectangle((int)Width - 2, 0, 2,
                                (int)Height - 1), Color.White);
        }
    }
}
```

Our game border will be a single-pixel wide white line on the left, top and right side of the screen. A ball hitting a border will bounce. Since the bottom of the screen has no border, a ball passing the bottom of the screen will fall out of play. We'll just pass the screen height and width to the constructor, so we know where to draw the lines. We'll also get a reference to an image with a single white pixel, that we'll save in `imgPixel`.

In our *Draw* method, we'll call `spriteBatch.Draw`, and use the Rectangle class to create a rectangle of white pixels to the screen. But, we'll set either the height or width to "1" in each call, so we are effectively drawing single lines. We can use this approach whenever we want to draw simple lines or rectangles of arbitrary shapes at run-time.

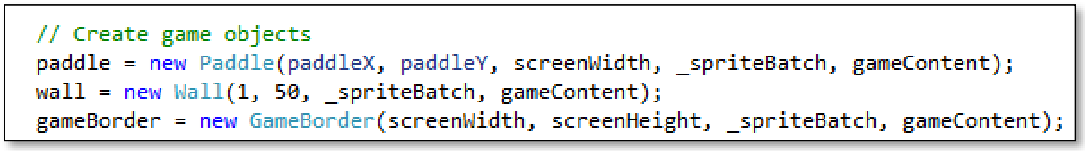
Game Class

We add a new private variable for our `GameBorder` instance to "Game1.cs", as shown below:



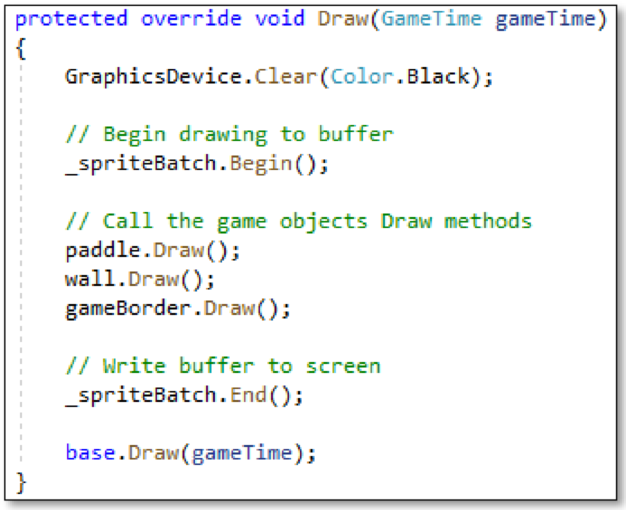
```
public class Game1 : Game
{
    /*************************************************************
      Game class properties
    *************************************************************/
    private readonly GraphicsDeviceManager _graphics;
    private SpriteBatch _spriteBatch;
    GameContent gameContent;

    private Paddle paddle;
    private Wall wall;
    private GameBorder gameBorder;
```

We'll also need to instantiate it. We can do that in the **LoadContent** method of "Game1.cs". Add the line as show below:



```
// Create game objects
paddle = new Paddle(paddleX, paddleY, screenWidth, _spriteBatch, gameContent);
wall = new Wall(1, 50, _spriteBatch, gameContent);
gameBorder = new GameBorder(screenWidth, screenHeight, _spriteBatch, gameContent);
```

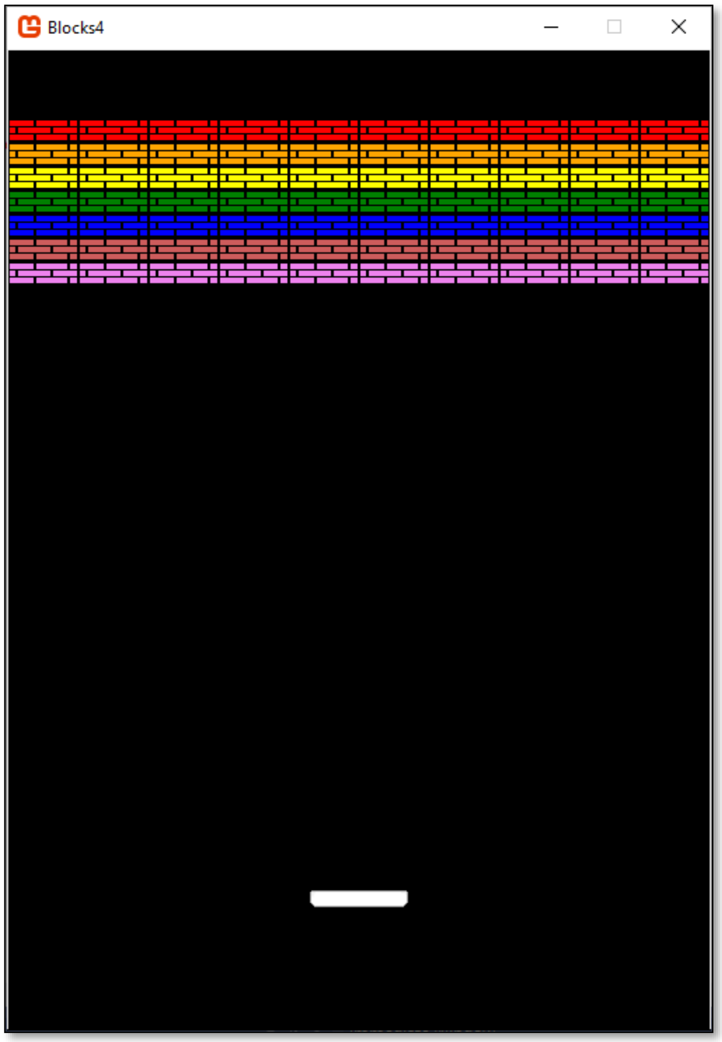
We need to tell the border to draw itself. Add a call to **gameBorder.Draw()** in our "Game1.cs" file **Draw** method as shown below:



```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.Black);

    // Begin drawing to buffer
    _spriteBatch.Begin();

    // Call the game objects Draw methods
    paddle.Draw();
    wall.Draw();
    gameBorder.Draw();

    // Write buffer to screen
    _spriteBatch.End();

    base.Draw(gameTime);
}
```

Run the Game

Go ahead and run the game, by pressing **F5**, just to make sure it draws your game borders. It should look like this:



We have the playing field, but nothing is moving. Boring . . . .

Let’s add some action to our game in the next episode.

Assignment Submission

Zip up the pseudocode and the project folder. Submit in Blackboard.