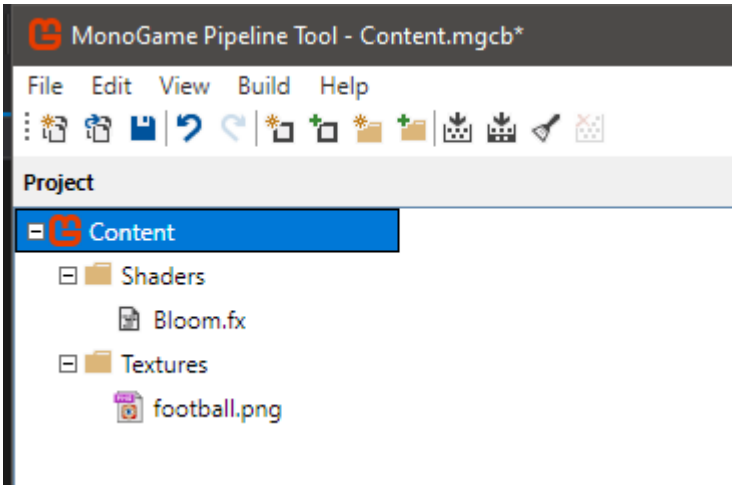
Monogame – Basics

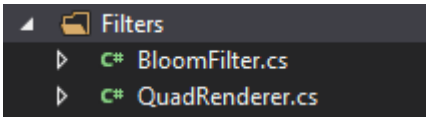
- 2D Basics
- Sprites (a moving ball)
- Position, Velocity & speed
- Basic controls (Keyboard input)
- Effects (Pixel Shader - Let's play with the Bloom effect)

Written by Brian Niels Bergh September 2019.

1. Add these elements to the content pipeline:



Add these two classes in a folder called Filters:



Remark: These classes uses a namespace for this example, if you grab'em, you might need to rename the namespaces:

```
namespace MonoBloom.Filters
{
```

Now, create these variables at the top of the Game1 class:

```
Texture2D txFootball;
RenderTarget2D renderTarget;
MonoBloom.Filters.BloomFilter bloomFilter;
```

In the LoadContent method of Game1, add these lines, just after the initialization of SpriteBatch:

```
txFootball = this.Content.Load<Texture2D>("Textures/football");
```

And initialize the bloom filter using this code:

```
if (graphics.GraphicsProfile == GraphicsProfile.HiDef)
{
  bloomFilter = new Filters.BloomFilter();
  bloomFilter.Load(GraphicsDevice, Content, graphics.PreferredBackBufferWidth, graphics.PreferredBackBufferHeight);
  bloomFilter.BloomPreset = Filters.BloomFilter.BloomPresets.SuperWide;
  bloomFilter.BloomStrengthMultiplier = 0.7f;
  bloomFilter.BloomThreshold = 0.05f;
}
```

Note, that the Load method of the bloomFilter takes the Content (Content manager) as argument, and assumes that the shader file is located in this structure: Shaders/BloomFilter/Bloom

If that's not the case, either change the path in the BloomFilter load method, or make sure that the structure in the content file match.

```
234     public void Load(GraphicsDevice graphicsDevice, ContentManager content, int
235     {
236       _graphicsDevice = graphicsDevice;
237       UpdateResolution(width, height);
238
239       //if quadRenderer == null -> new, otherwise not
240       _quadRenderer = quadRenderer ?? new QuadRenderer(graphicsDevice);
241
242       _renderTargetFormat = renderTargetFormat;
243
244       //Load the shader parameters and passes for cheap and easy access
245       _bloomEffect = content.Load<Effect>("Shaders/BloomFilter/Bloom");
```

You might play with the values of the bloom filter:

```
bloomFilter.BloomPreset = Filters.BloomFilter.BloomPresets.SuperWide;
bloomFilter.BloomThreshold = 0.4f;
bloomFilter.BloomStrengthMultiplier = .9f;
```

(you can even make some controls to handle adjustment of these runtime .-)

Now, setup the renderTarget to match our viewport and our current presentation properties. Remember to do this again (and call UpdateResolution on the bloomEffect) when the viewport size changes. (this can be a bit tricky as you need to dispose the old renderTarget)

```
alternativeRenderTarget = new RenderTarget2D(graphics.GraphicsDevice, graphics.GraphicsDevice.Viewport.Width,
  graphics.GraphicsDevice.Viewport.Height, false,
  GraphicsDevice.PresentationParameters.BackBufferFormat, DepthFormat.Depth24,
  graphics.GraphicsDevice.PresentationParameters.MultiSampleCount,RenderTargetUsage.PreserveContents, false, 1);
```

Now, somewhere in the Game1 class, add these tree lines

```
Vector2 ballPosition = Vector2.Zero;
Vector2 ballVelocity = new Vector2(1, 1);
float ballSpeed = 100f;
```

Enable VSYNC (use Vertical Synchronization = FPS = Monitor Refresh Rate)
Add these two lines to the Game1 constructor, just after the graphics variable is populated:

```
// enable VSYNC
IsFixedTimeStep = false; // explicit no fixed timestep
graphics.SynchronizeWithVerticalRetrace = true; // VSYNC on
```

Now, to the fun part…..Using the stuff…..

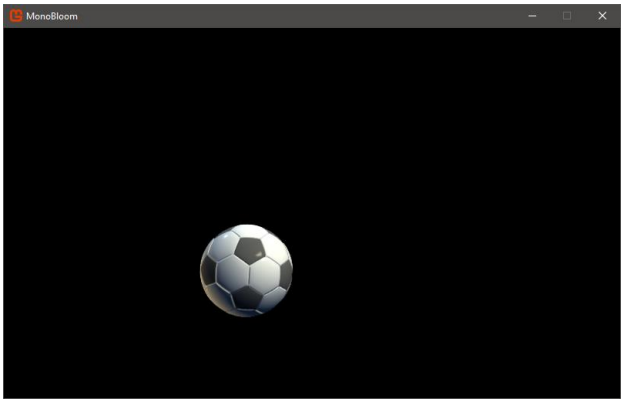
In the Update method of Game1 add this region (Update Ball Postion)

```
protected override void Update(GameTime gameTime)
{
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
    Exit();

  #region Update Ball Position

  // update ball position
  ballPosition += ballVelocity * (float)(gameTime.ElapsedGameTime.TotalSeconds *  (double)ballSpeed);

  // see if we need to flip from 1 to -1 (flip directions)
  ballVelocity.X *= ((ballPosition.X + txFootball.Width > graphics.GraphicsDevice.Viewport.Width || ballPosition.X < 0) ? -1 : 1);
  ballVelocity.Y *= ((ballPosition.Y + txFootball.Height > graphics.GraphicsDevice.Viewport.Height || ballPosition.Y < 0) ? -1 : 1);

  #endregion


  base.Update(gameTime);
}
```

In the Draw method of Game1, add this region (Draw Ball)

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

  #region Draw Ball

  spriteBatch.Begin(SpriteSortMode.Deferred, blendState: BlendState.NonPremultiplied);

  spriteBatch.Draw(txFootball, ballPosition, Color.White);

  spriteBatch.End();

  #endregion

  base.Draw(gameTime);
}
```

Now, press F5 to run the project…..You should see a ball bouncing on the walls:



Wooooooooooooooouhhhhhh…….Borring, right? 😊

Now, to the REALLY fun part…..using the bloom effect…

So far we have been drawing everything (the ball!) to the default renderTarget.

Consider the renderTarget our “Sufface we draw to” (its juts a texture we draw on!)

We now need to draw the scene to a texture (renderTarget) we can then do some post processing to, and the draw that texture back to our default renderTarget.

We could of cause just apply our bloom effect to the default renderTarget, but then EVERYTHING in our scene will have the bloom effect applied to it, which isn’t always what we want.

In order to use shaders, we need to instruct our graphics device to use the HI-RES profile.
(There are two, one calle Reach and one called HiDef) – This is because the monogame is a portable format, also used on phones and other more restricted platforms, where shaders aren’t guaranteed to be supported. In Windows (almost any gfx card) they are .-) – So set the profile to HiDef like this:

```
public Game1()
{
  graphics = new GraphicsDeviceManager(this);

  graphics.GraphicsProfile = GraphicsProfile.HiDef;
```

In the Game1 constructor.

Now we beed to render to a different render target than the build-in one. As we need to extract what we have drawn so far (in the bloom filter) and apply some post processing to it.

In the draw method, add this at the top:

```
GraphicsDevice.SetRenderTarget(renderTarget);
GraphicsDevice.Clear(Color.Black);
```

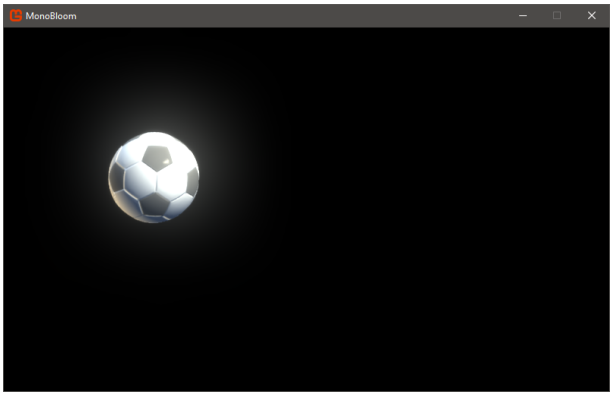
Then add this section, just after the section where we draw the ball (not to the renderTarget)

```
#region Apply Bloom

GraphicsDevice.SetRenderTarget(null);
GraphicsDevice.Clear(Color.Black);
if (bloomFilter != null)
{


  Texture2D bloom = null;
  bloom = bloomFilter.Draw(renderTarget, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

  GraphicsDevice.SetRenderTarget(renderTarget);
  spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.Additive);
  spriteBatch.Draw(bloom, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
  spriteBatch.End();
  GraphicsDevice.SetRenderTarget(null);


  // now draw our renderTarget to our default Render Target

  spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
  spriteBatch.Draw(renderTarget, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
  spriteBatch.End();

}

#endregion
```

The complete draw code now looks like this:

```
protected override void Draw(GameTime gameTime)
{
  // set our render target to our special one (renderTarget)
  GraphicsDevice.SetRenderTarget(renderTarget);
  // clear its background
  GraphicsDevice.Clear(Color.Black);

  #region Draw Ball
  // draw the ball to our rendertarget (renderTarget)
  spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
  spriteBatch.Draw(txFootball, ballPosition, Color.White);
  spriteBatch.End();
  #endregion

  #region Apply Bloom
  if (bloomFilter != null)
  {
    Texture2D bloom = null;
    // draw our post processing (bloom) to the new texture "bloom"; using our renderTarget (containing our ball-image so far)
    bloom = bloomFilter.Draw(renderTarget, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

    // Remark: bloomFilter.Draw will flip our render target back to null (default)
    // flip it back to renderTarget, as we aren't done with it.
    GraphicsDevice.SetRenderTarget(renderTarget);
    // draw our bloom texture to the renderTarget (overwrite the old non-bloomed ball image)
    spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.Additive);
    spriteBatch.Draw(bloom, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
    spriteBatch.End();

    // now, set our render target back null (to draw on the default render target, the one we actually see from our backbuffer)
    GraphicsDevice.SetRenderTarget(null);

    // now, draw our renderTarget texture to our default render target (null), so we can see the final result
    spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
    spriteBatch.Draw(renderTarget, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
    spriteBatch.End();
    // done :-)
  }

  #endregion

  base.Draw(gameTime);
}
```

Now, press F5, and let’s see if there are differences?



Ohh yeah! Bloooooooming it is 😊

Now, we can make some controls to easily adjust the effect, and also add a few lines of extra code to visualize the properties.

Add these variables anywhere in the root of the class Game1

```
bool useBloom = true;
float BloomStrengthMultiplier = 0.7f;
float BloomThreshold = 0.05f;
float numberOfBalls = 15;
float bloomStrengthMultiplierSpeed = 0.1f;
float bloomThresholdSpeed = 0.1f;
KeyboardState oldKeyboardState;
List<Ball> balls;
Random random = new Random(DateTime.Now.Millisecond);
SpriteFont font;
```

In the draw method “Apply Bloom” region, extend the line if(bloomFilter != null) to:

```
#region Apply Bloom
if (useBloom && bloomFilter != null)
{
```

In the region Apply Bloom, move these 3 lines outside the If statement mentioned above, and outside the region:

```
// now, draw our renderTarget texture to our default render target (null), so we can see the final result
spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
spriteBatch.Draw(renderTarget, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
spriteBatch.End();
```

Now it looks like this:

```
#region Apply Bloom
if (useBloom && bloomFilter != null)
{
  Texture2D bloom = null;
  // draw our post processing (bloom) to the new texture "bloom"; using our renderTarget (containing our ball-image so far)
  bloom = bloomFilter.Draw(renderTarget, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

  // Remark: bloomFilter.Draw will flip our render target back to null (default)
  // flip it back to renderTarget, as we aren't done with it.
  GraphicsDevice.SetRenderTarget(renderTarget);
  // draw our bloom texture to the renderTarget (overwrite the old non-bloomed ball image)
  spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.Additive);
  spriteBatch.Draw(bloom, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
  spriteBatch.End();

  // now, set our render target back null (to draw on the default render target, the one we actually see from our backbuffer)
  GraphicsDevice.SetRenderTarget(null);
  // done :-)
}
#endregion

// now, draw our renderTarget texture to our default render target (null), so we can see the final result
spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
spriteBatch.Draw(renderTarget, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
spriteBatch.End();

base.Draw(gameTime);
```

Now, in the update method, add the new section “Check Keyboard”

```
protected override void Update(GameTime gameTime)
{
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
    Exit();

  #region Check Keyboard

  KeyboardState keyboardState = Keyboard.GetState();

  if (keyboardState.IsKeyDown(Keys.B) && oldKeyboardState.IsKeyUp(Keys.B))
    useBloom = !useBloom;

  if (BloomStrengthMultiplier < 2 && keyboardState.IsKeyDown(Keys.Add) && keyboardState.IsKeyUp(Keys.LeftAlt))
    BloomStrengthMultiplier = MathHelper.Clamp(BloomStrengthMultiplier + ((float)gameTime.ElapsedGameTime.TotalSeconds * bloomStrengthMultiplierSpeed), 0, 2f);

  if (BloomStrengthMultiplier >0 && keyboardState.IsKeyDown(Keys.Subtract) && keyboardState.IsKeyUp(Keys.LeftAlt))
    BloomStrengthMultiplier = MathHelper.Clamp(BloomStrengthMultiplier - ((float)gameTime.ElapsedGameTime.TotalSeconds * bloomStrengthMultiplierSpeed), 0, 2f);

  if (BloomThreshold < 1 && keyboardState.IsKeyDown(Keys.Add) && keyboardState.IsKeyDown(Keys.LeftAlt))
    BloomThreshold = MathHelper.Clamp(BloomThreshold + ((float)gameTime.ElapsedGameTime.TotalSeconds * bloomThresholdSpeed), 0, 1f);

  if (BloomThreshold > 0 && keyboardState.IsKeyDown(Keys.Subtract) && keyboardState.IsKeyDown(Keys.LeftAlt))
    BloomThreshold = MathHelper.Clamp(BloomThreshold - ((float)gameTime.ElapsedGameTime.TotalSeconds * bloomThresholdSpeed), 0, 1f);

  #endregion
```

Add this line at the end of the update (but before base.update)

```
oldKeyboardState = keyboardState;
```

In the Draw method, add these lines:

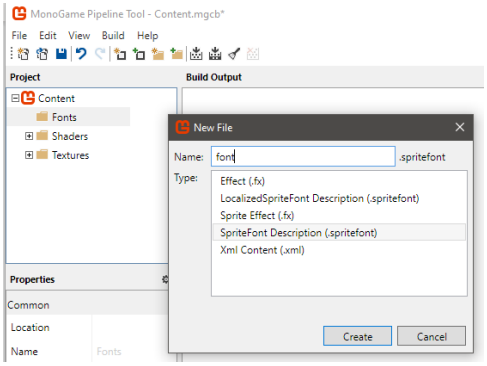
```
#region Apply Bloom
if (useBloom && bloomFilter != null)
{
  bloomFilter.BloomStrengthMultiplier = BloomStrengthMultiplier;
  bloomFilter.BloomThreshold = BloomThreshold;
```

Move these lines OUTSIDE the if(useBloom)

```
// now, set our render target back null (to draw on the default render target, the one we actually see from our backbuffer)
GraphicsDevice.SetRenderTarget(null);
```

You can now adjust BloomStrengthMultiplier by + and -
And BloomThreshold by ALT+ + and –

In the Content project, add a new folder and name it “Fonts”, right click on the new folder, and click Add new Item…



Select SpriteFont Description (.spritefont) – And name it “font”

In the method LoadContent, add this line:

```
font = this.Content.Load<SpriteFont>("Fonts/font");
```

After the line

```
txFootball = this.Content.Load<Texture2D>("Textures/football");
```

In the draw method, add the region Draw Texts last (but before base.Draw), to ensure we draw the text on top of everything)

```
#region Draw POST processed texture back to our default target
// now, draw our renderTarget texture to our default render target (null), so we can see the final result
spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
spriteBatch.Draw(renderTarget, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height), Color.White);
spriteBatch.End();

#endregion

#region Draw Texts
// draw texts
spriteBatch.Begin(SpriteSortMode.BackToFront, blendState: BlendState.NonPremultiplied);

float yOffset = 10;
spriteBatch.DrawString(font, $"Bloom Enabled:{(useBloom ? "Yes" : "No")}", new Vector2(10, yOffset), Color.White);
yOffset += font.MeasureString("X").Y;
spriteBatch.DrawString(font, $"Bloom Threshold:{BloomThreshold.ToString("0.00")}", new Vector2(10, yOffset), Color.White);
yOffset += font.MeasureString("X").Y;
spriteBatch.DrawString(font, $"Bloom Multiplier:{BloomStrengthMultiplier.ToString("0.00")}", new Vector2(10, yOffset), Color.White);

spriteBatch.End();

#endregion

base.Draw(gameTime);
```

Try it for yourself…..F5!

And now for the final fun….lets add some more balls……as they say over there!

At this point, its easier for us to maintain each ball in a separate class, add this class to the project:

```
namespace MonoBloom
{
    0 references
    public class Ball
    {
        public Texture2D texture;
        public Vector2 position;
        public Vector2 velocity;
        public float speed;
    }
}
```

You can now remove these lines from Game1

```
Vector2 ballPosition = Vector2.Zero;
Vector2 ballVelocity = new Vector2(1, 1);
float ballSpeed = 100f;
```

Now, add the following region to the bottom of the LoadContent method in Game1

```
#region Generate balls

// generate a list of balls, all starting at different locations, and all with different speed and velocity
balls = new List<Ball>();
for (int i = 0; i < numberOfBalls; i++)
{
    balls.Add(new Ball()
    {
        position = new Vector2(
            (float)((double)(graphics.GraphicsDevice.Viewport.Width - txFootball.Width) * random.NextDouble()),
            (float)((double)(graphics.GraphicsDevice.Viewport.Height - txFootball.Height) * random.NextDouble())),
        speed = (float)random.Next(50, 250),
        texture = txFootball,
        velocity = new Vector2((random.NextDouble() < 0.5d ? 1 : -1), (random.NextDouble() < 0.5d ? 1 : -1))
    });
}

#endregion
```

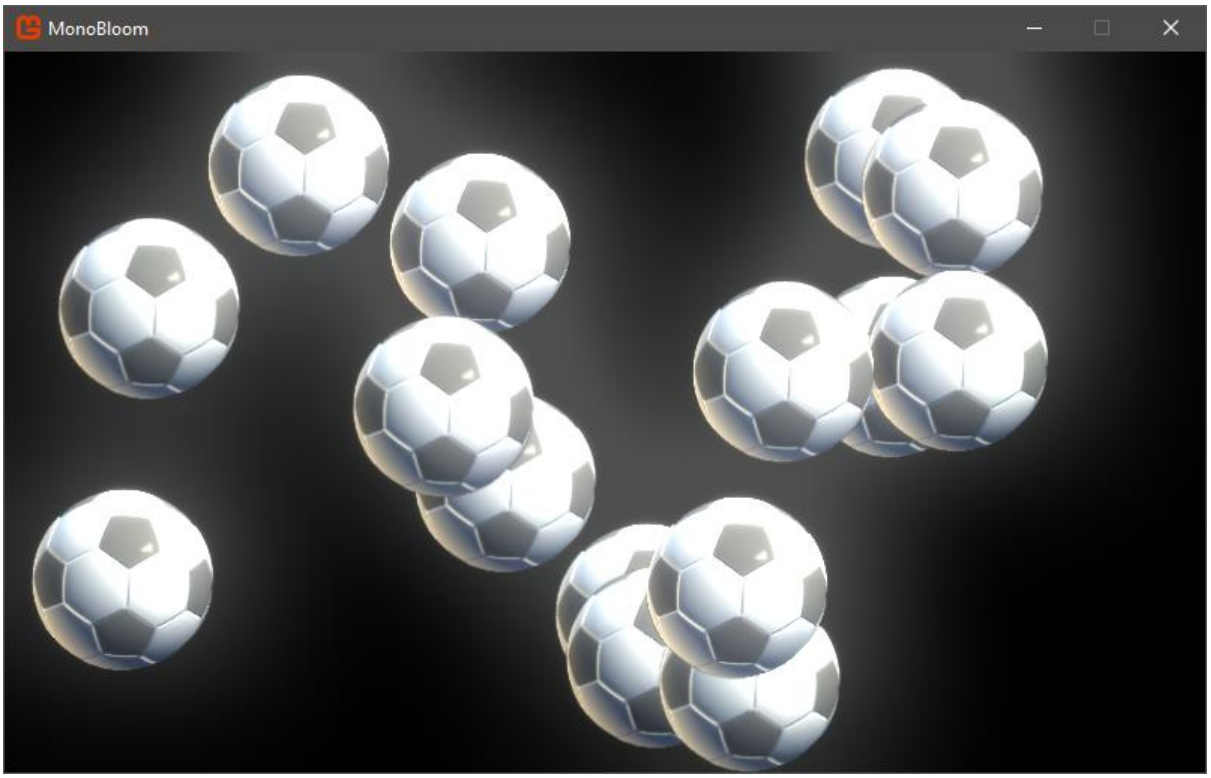
Add this to the bottom of the Update method of Game1 (but before the Base.Update!)

```
#region Update Ball Positions

foreach (var ball in balls)
{
    ball.position += ball.velocity * (float)(gameTime.ElapsedGameTime.TotalSeconds * (double)ball.speed);
    ball.velocity.X *= ((ball.position.X + ball.texture.Width > graphics.GraphicsDevice.Viewport.Width || ball.position.X < 0) ? -1 : 1);
    ball.velocity.Y *= ((ball.position.Y + ball.texture.Height > graphics.GraphicsDevice.Viewport.Height || ball.position.Y < 0) ? -1 : 1);
}

#endregion
```

Replace the draw region in the Draw method of Game1 with this:

```
#region Draw Ball
// draw the ball to our rendertarget (renderTarget)
spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
foreach (var ball in balls)
{
    spriteBatch.Draw(ball.texture, ball.position, Color.White);
}
spriteBatch.End();
#endregion
```

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5 !!!!!!!!! 😊



You may play around with the initialization code (number of balls, speed etc)

You find the complete source code on the next pages.

Game1.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;

namespace MonoBloom
{
  /// <summary>
  /// This is the main type for your game.
  /// </summary>
  public class Game1 : Game
  {
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    Texture2D txFootball;
    RenderTarget2D renderTarget;
    MonoBloom.Filters.BloomFilter bloomFilter;

    bool useBloom = true;
    float BloomStrengthMultiplier = 0.7f;
    float BloomThreshold = 0.05f;
    float numberOfBalls = 15;
    float bloomStrengthMultiplierSpeed = 0.1f;
    float bloomThresholdSpeed = 0.1f;
    KeyboardState oldKeyboardState;
    List<Ball> balls;
    Random random = new Random(DateTime.Now.Millisecond);
    SpriteFont font;
    public Game1()
    {
      graphics = new GraphicsDeviceManager(this);

      graphics.GraphicsProfile = GraphicsProfile.HiDef;


      // enable VSYNC
      IsFixedTimeStep = false; // explicit no fixed timestep
      graphics.SynchronizeWithVerticalRetrace = true; // VSYNC on

      Content.RootDirectory = "Content";
    }

    protected override void Initialize()
    {
      base.Initialize();
    }

    protected override void LoadContent()
    {
      spriteBatch = new SpriteBatch(GraphicsDevice);

      txFootball = this.Content.Load<Texture2D>("Textures/football");
      font = this.Content.Load<SpriteFont>("Fonts/font");

      if (graphics.GraphicsProfile == GraphicsProfile.HiDef)
      {
        bloomFilter = new Filters.BloomFilter();
        bloomFilter.Load(GraphicsDevice, Content, graphics.PreferredBackBufferWidth,
graphics.PreferredBackBufferHeight);
        bloomFilter.BloomPreset = Filters.BloomFilter.BloomPresets.SuperWide;
        bloomFilter.BloomStrengthMultiplier = 0.7f;
        bloomFilter.BloomThreshold = 0.05f;
      }
```

```
      renderTarget = new RenderTarget2D(graphics.GraphicsDevice,
graphics.GraphicsDevice.Viewport.Width,
        graphics.GraphicsDevice.Viewport.Height, false,
        GraphicsDevice.PresentationParameters.BackBufferFormat, DepthFormat.Depth24,
        graphics.GraphicsDevice.PresentationParameters.MultiSampleCount,
RenderTargetUsage.PreserveContents, false, 1);

      #region Generate balls

      // generate a list of balls, all starting at different locations, and all with different speed
and velocity
      balls = new List<Ball>();
      for (int i = 0; i < numberOfBalls; i++)
      {
        balls.Add(new Ball()
        {
          position = new Vector2(
            (float)((double)(graphics.GraphicsDevice.Viewport.Width - txFootball.Width) *
random.NextDouble()),
            (float)((double)(graphics.GraphicsDevice.Viewport.Height - txFootball.Height) *
random.NextDouble())),
          speed = (float)random.Next(50, 250),
          texture = txFootball,
          velocity = new Vector2((random.NextDouble() < 0.5d ? 1 : -1), (random.NextDouble() < 0.5d
? 1 : -1))
        });
      }

      #endregion
    }
    protected override void UnloadContent()
    {
    }

    protected override void Update(GameTime gameTime)
    {
      if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

      #region Check Keyboard

      KeyboardState keyboardState = Keyboard.GetState();

      if (keyboardState.IsKeyDown(Keys.B) && oldKeyboardState.IsKeyUp(Keys.B))
        useBloom = !useBloom;

      if (BloomStrengthMultiplier < 2 && keyboardState.IsKeyDown(Keys.Add) &&
keyboardState.IsKeyUp(Keys.LeftAlt))
        BloomStrengthMultiplier = MathHelper.Clamp(BloomStrengthMultiplier +
((float)gameTime.ElapsedGameTime.TotalSeconds * bloomStrengthMultiplierSpeed), 0, 2f);

      if (BloomStrengthMultiplier >0 && keyboardState.IsKeyDown(Keys.Subtract) &&
keyboardState.IsKeyUp(Keys.LeftAlt))
        BloomStrengthMultiplier = MathHelper.Clamp(BloomStrengthMultiplier -
((float)gameTime.ElapsedGameTime.TotalSeconds * bloomStrengthMultiplierSpeed), 0, 2f);

      if (BloomThreshold < 1 && keyboardState.IsKeyDown(Keys.Add) &&
keyboardState.IsKeyDown(Keys.LeftAlt))
        BloomThreshold = MathHelper.Clamp(BloomThreshold +
((float)gameTime.ElapsedGameTime.TotalSeconds * bloomThresholdSpeed), 0, 1f);

      if (BloomThreshold > 0 && keyboardState.IsKeyDown(Keys.Subtract) &&
keyboardState.IsKeyDown(Keys.LeftAlt))
        BloomThreshold = MathHelper.Clamp(BloomThreshold -
((float)gameTime.ElapsedGameTime.TotalSeconds * bloomThresholdSpeed), 0, 1f);

      #endregion
```

```
      #region Update Ball Positions

      foreach (var ball in balls)
      {
        ball.position += ball.velocity * (float)(gameTime.ElapsedGameTime.TotalSeconds *
(double)ball.speed);
        ball.velocity.X *= ((ball.position.X + ball.texture.Width >
graphics.GraphicsDevice.Viewport.Width || ball.position.X < 0) ? -1 : 1);
        ball.velocity.Y *= ((ball.position.Y + ball.texture.Height >
graphics.GraphicsDevice.Viewport.Height || ball.position.Y < 0) ? -1 : 1);
      }

      #endregion


      base.Update(gameTime);
    }



    protected override void Draw(GameTime gameTime)
    {
      // set our render target to our special one (renderTarget)
      GraphicsDevice.SetRenderTarget(renderTarget);
      // clear its background
      GraphicsDevice.Clear(Color.Black);

      #region Draw Ball
      // draw the ball to our rendertarget (renderTarget)
      spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
      foreach (var ball in balls)
      {
        spriteBatch.Draw(ball.texture, ball.position, Color.White);
      }
      spriteBatch.End();
      #endregion

      #region Apply Bloom
      if (useBloom && bloomFilter != null)
      {
        bloomFilter.BloomStrengthMultiplier = BloomStrengthMultiplier;
        bloomFilter.BloomThreshold = BloomThreshold;

        Texture2D bloom = null;
        // draw our post processing (bloom) to the new texture "bloom"; using our renderTarget
(containing our ball-image so far)
        bloom = bloomFilter.Draw(renderTarget, graphics.GraphicsDevice.Viewport.Width,
graphics.GraphicsDevice.Viewport.Height);

        // Remark: bloomFilter.Draw will flip our render target back to null (default)
        // flip it back to renderTarget, as we aren't done with it.
        GraphicsDevice.SetRenderTarget(renderTarget);
        // draw our bloom texture to the renderTarget (overwrite the old non-bloomed ball image)
        spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.Additive);
        spriteBatch.Draw(bloom, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height),
Color.White);
        spriteBatch.End();

        // now, set our render target back null (to draw on the default render target, the one we
actually see from our backbuffer)
        GraphicsDevice.SetRenderTarget(null);
        // done :-)
      }
      #endregion

      #region Draw POST processed texture back to our default target
      // now, draw our renderTarget texture to our default render target (null), so we can see the
final result
```

```
      spriteBatch.Begin(SpriteSortMode.Immediate, blendState: BlendState.NonPremultiplied);
      spriteBatch.Draw(renderTarget, new Rectangle(0, 0, renderTarget.Width, renderTarget.Height),
Color.White);
      spriteBatch.End();

      #endregion

      #region Draw Texts
      // draw texts
      spriteBatch.Begin(SpriteSortMode.BackToFront, blendState: BlendState.NonPremultiplied);

      float yOffset = 10;
      spriteBatch.DrawString(font, $"Bloom Enabled:{(useBloom ? "Yes" : "No")}", new Vector2(10,
yOffset), Color.White);
      yOffset += font.MeasureString("X").Y;
      spriteBatch.DrawString(font, $"Bloom Threshold:{BloomThreshold.ToString("0.00")}", new
Vector2(10, yOffset), Color.White);
      yOffset += font.MeasureString("X").Y;
      spriteBatch.DrawString(font, $"Bloom Multiplier:{BloomStrengthMultiplier.ToString("0.00")}",
new Vector2(10, yOffset), Color.White);

      spriteBatch.End();

      #endregion

      base.Draw(gameTime);
    }
  }
}
```

Ball.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MonoBloom
{
  public class Ball
  {
    public Texture2D texture;
    public Vector2 position;
    public Vector2 velocity;
    public float speed;
  }
}
```

Please note that the Bloom filter was not written by me, here is the info:

```
  /// Version 1.1, 16. Dez. 2016
  ///
  /// Bloom / Blur, 2016 TheKosmonaut
  ///
  /// High-Quality Bloom filter for high-performance applications
  ///
  /// Based largely on the implementations in Unreal Engine 4 and Call of Duty AW
  /// For more information look for
  /// "Next Generation Post Processing in Call of Duty Advanced Warfare" by Jorge Jimenez
  /// http://www.iryoku.com/downloads/Next-Generation-Post-Processing-in-Call-of-Duty-Advanced-
Warfare-v18.pptx
  ///
  /// The idea is to have several rendertargets or one rendertarget with several mip maps
  /// so each mip has half resolution (1/2 width and 1/2 height) of the previous one.
  ///
  /// 32, 16, 8, 4, 2
  ///
  /// In the first step we extract the bright spots from the original image. If not specified
otherwise thsi happens in full resolution.
  /// We can do that based on the average RGB value or Luminance and check whether this value is
higher than our Threshold.
  ///     BloomUseLuminance = true / false (default is true)
  ///     BloomThreshold = 0.8f;
  ///
  /// Then we downscale this extraction layer to the next mip map.
  /// While doing that we sample several pixels around the origin.
  /// We continue to downsample a few more times, defined in
  ///     BloomDownsamplePasses = 5 ( default is 5)
  ///
  /// Afterwards we upsample again, but blur in this step, too.
  /// The final output should be a blur with a very large kernel and smooth gradient.
  ///
  /// The output in the draw is only the blurred extracted texture.
  /// It can be drawn on top of / merged with the original image with an additive operation for
example.
  ///
  /// If you use ToneMapping you should apply Bloom before that step.
```