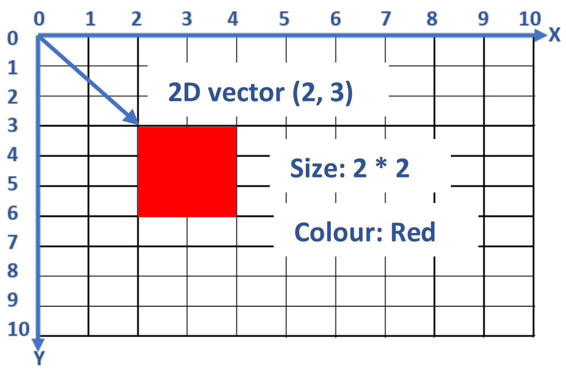
MONOGAME

# Making games with C#

## Preparing to add a shape (texture)

To prepare for adding a shape code needs to be added to three sections **Game1 : Game** and **Game1()**.

### Shape and position



Simply put, in games a shape is described as a texture, which has a size, colour and position.

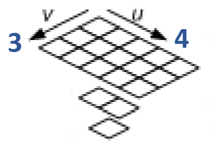
Collections of shapes can be used to make complex objects.

The 2D vector position of the shape, is the top left-hand corner.

In MonoGame, this is a **Texture2D**.

### Texture2D

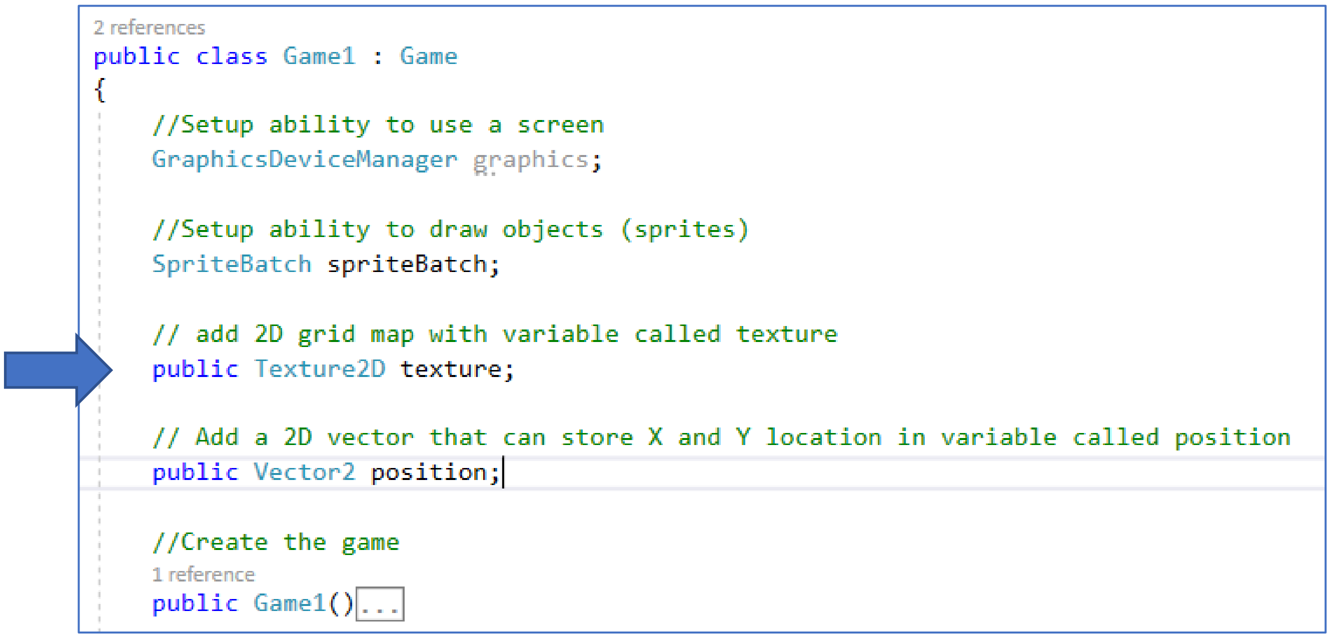
A Texture2D is a 2D grid of texels and is used to draw a shape. *Each texel* has an address coordinate by a u, v vector and can be given a colour.



A texel (pixel) is the smallest unit that the Graphics Processor Unit (GPU) can draw.

## Adding an option to make a shape

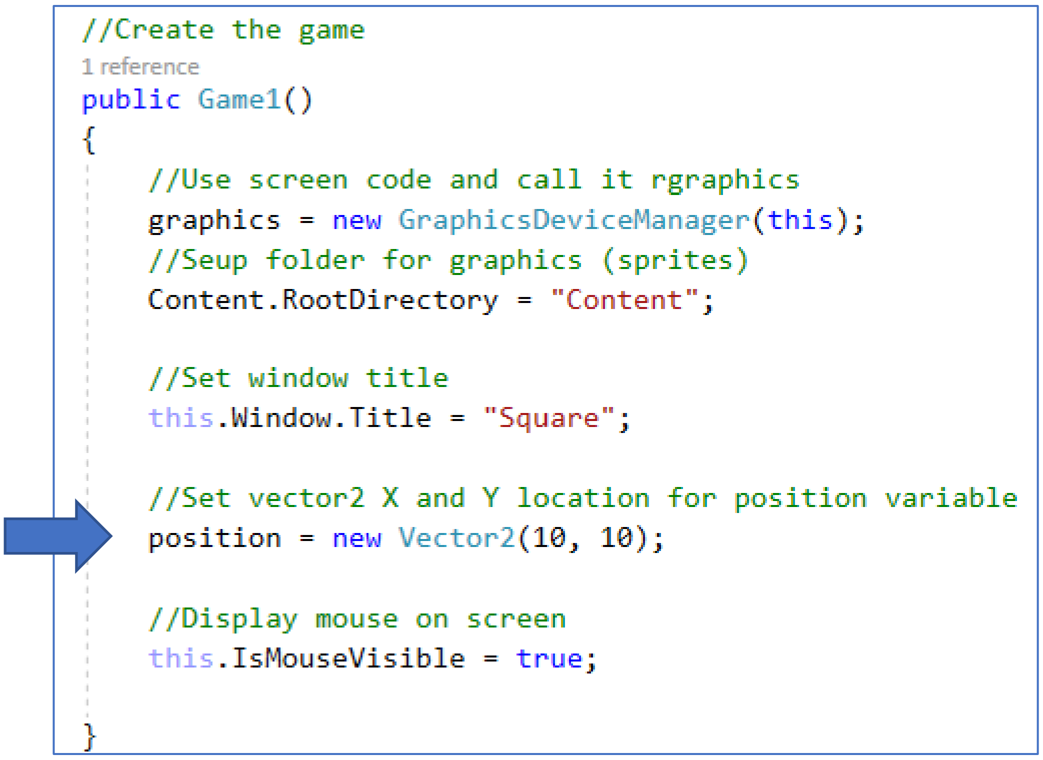
To allow a texture (shape) to be the following built-in objects that need adding at the highest level, so they are available to use later.



```
2 references
public class Game1 : Game
{
    //Setup ability to use a screen
    GraphicsDeviceManager graphics;

    //Setup ability to draw objects (sprites)
    SpriteBatch spriteBatch;

    // add 2D grid map with variable called texture
    public Texture2D texture;

    // Add a 2D vector that can store X and Y location in variable called position
    public Vector2 position;

    //Create the game
    1 reference
    public Game1()...
```

## Creating 2D Vector position

The next step is to create (instantiate) a position object variable called ***position***, to store the Vector2 data. This is done when the game is created using Game().



```
//Create the game
1 reference
public Game1()
{
    //Use screen code and call it rgraphics
    graphics = new GraphicsDeviceManager(this);
    //Seup folder for graphics (sprites)
    Content.RootDirectory = "Content";

    //Set window title
    this.Window.Title = "Square";

    //Set vector2 X and Y location for position variable
    position = new Vector2(10, 10);

    //Display mouse on screen
    this.IsMouseVisible = true;

}
```

## Creating a shape with colour

Now all of the basic steps are complete, the next step is to create a shape using the position and give it a colour, this is done in the **Initialize()** method

### Final Initialize code

Here is all the code together to add a shape and colour:

```
protected override void Initialize()
{
    // Create initialize empty texture (this is the square)
    texture = new Texture2D(this.GraphicsDevice, 50, 50);

    // create array to store colour of each texel in the square called texture
    Color[] colorData = new Color[50 * 50];

    //Loop through array to set the colour for each texel in texture
    for (int i = 0; i < (50*50); i++)
    {
        colorData[i] = Color.Red;
    }

    //Apply colour for the texture
    texture.SetData<Color>(colorData);

    //Always last in the sequence
    base.Initialize();

}
```

### *Creating a shape code Explanation*

### Creating a shape

This is the point where the shape is created in computer memory and given a size.

```
protected override void Initialize()
{
    // Initialise (create) new texture
    texture = new Texture2D(this.GraphicsDevice, 50, 50); //instantiate empt texture 50x50 texels
```

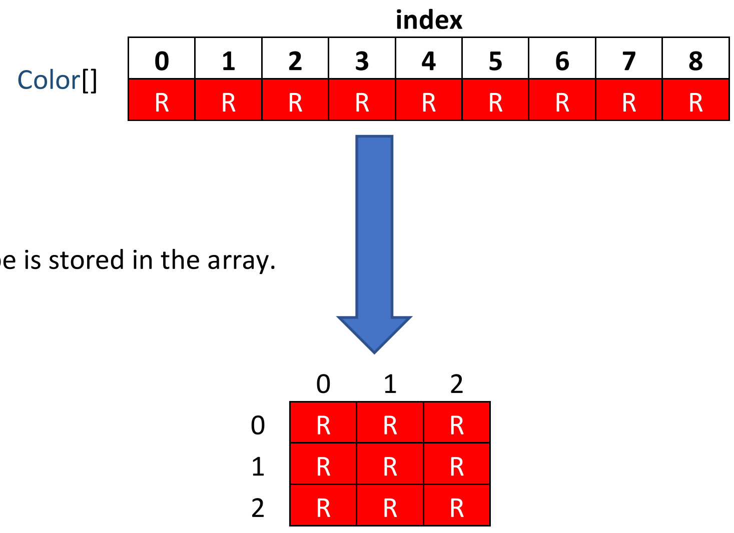
### Making a colour

The colour of each texel needs to be defined. This is done in a special array (list) called **Color[]**

```
    Color[] colorData = new Color[50 * 50]; //create array to store colour of each texel
```

The MonoGame Color[] object is used to create an array called ***colorData*** to store a colour for each texel grid position, the are 2500 texels for a 50x50 sized shape.

*Example:*
This is a simple example, much more texels can be used to create much more complex pixel art.



The colour of the shape is stored in the array.

In this example each position in the (50x50) array is looped through and made red, like the array above.

The loop variable i (index) is changed by the for loop and tells the code where to save the next colour in the array (list) using the index number.

```
//Loop through array to set the colour
for (int i = 0; i < (50*50); i++)
{
    colorData[i] = Color.Red;
}
```

### Giving the shape a colour

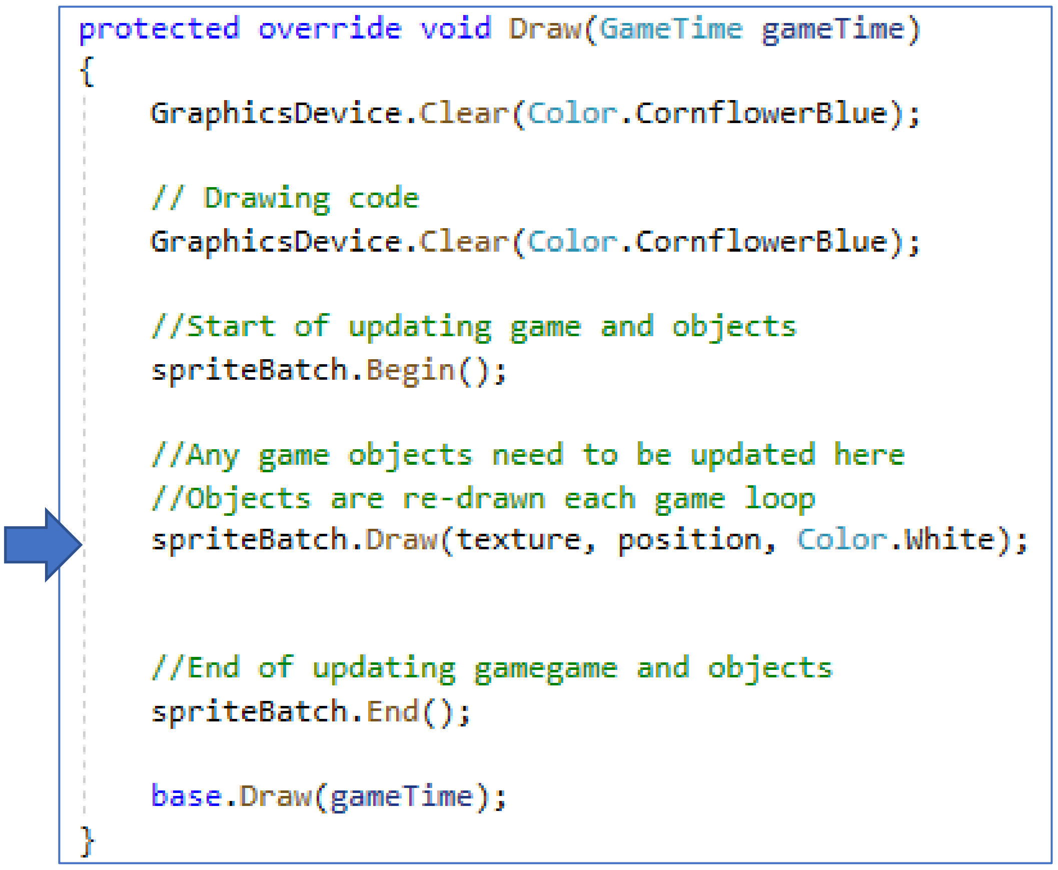
The final step is to join the two together, texture and colour, using texture.SetData method, this tells the texture shape to use the colourData array.

```
texture.SetData<Color>(colorData); //Set texture colour using the array
```

## Drawing the sprites to screen

This method is used to draw and update all the sprites on screen with their positions.

**Final Draw code**

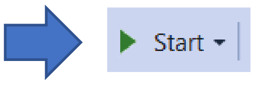


```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // Drawing code
    GraphicsDevice.Clear(Color.CornflowerBlue);

    //Start of updating game and objects
    spriteBatch.Begin();

    //Any game objects need to be updated here
    //Objects are re-drawn each game loop
    spriteBatch.Draw(texture, position, Color.White);


    //End of updating gamegame and objects
    spriteBatch.End();

    base.Draw(gameTime);
}
```

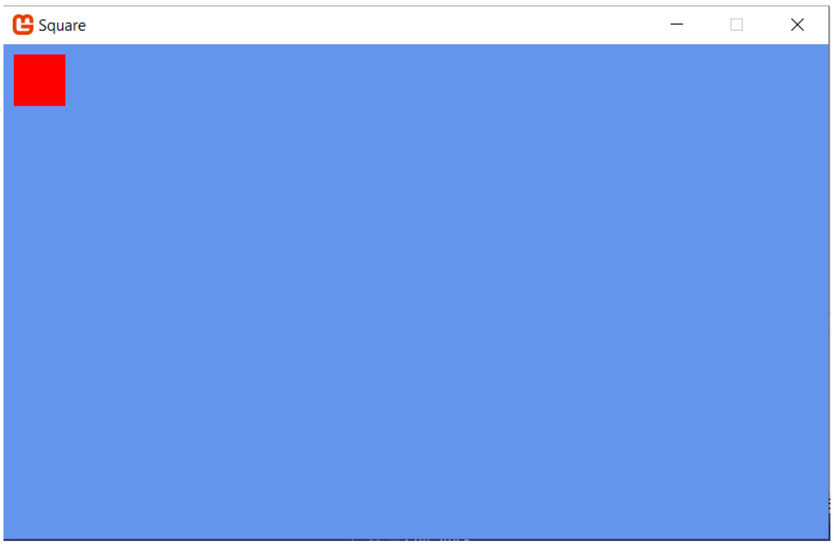
## Test, test and check

It is really important to check your code to see if it does what you expect. Testing is vital!

Test by pressing start



It should look like this:



## Experiment

Have a go at changing the following and test what happens:

**The position: Game1( )**

```
//Set vector2 X and Y location for position variable
position = new Vector2(10, 10);
```

**The shape size: Initalise( )**

```
protected override void Initialize()
{
    // Initialise (create) new texture
    texture = new Texture2D(this.GraphicsDevice, 50, 50); //instantiate empt texture 50x50 texels
```

**The colour: Initalise( )**

*Note*: If you change the size, don’t forget to change the size (50 *50) to match.

```
    //Loop through array to set the colour
    for (int i = 0; i < (50*50); i++)
    {
        colorData[i] = Color.Red;
    }
```

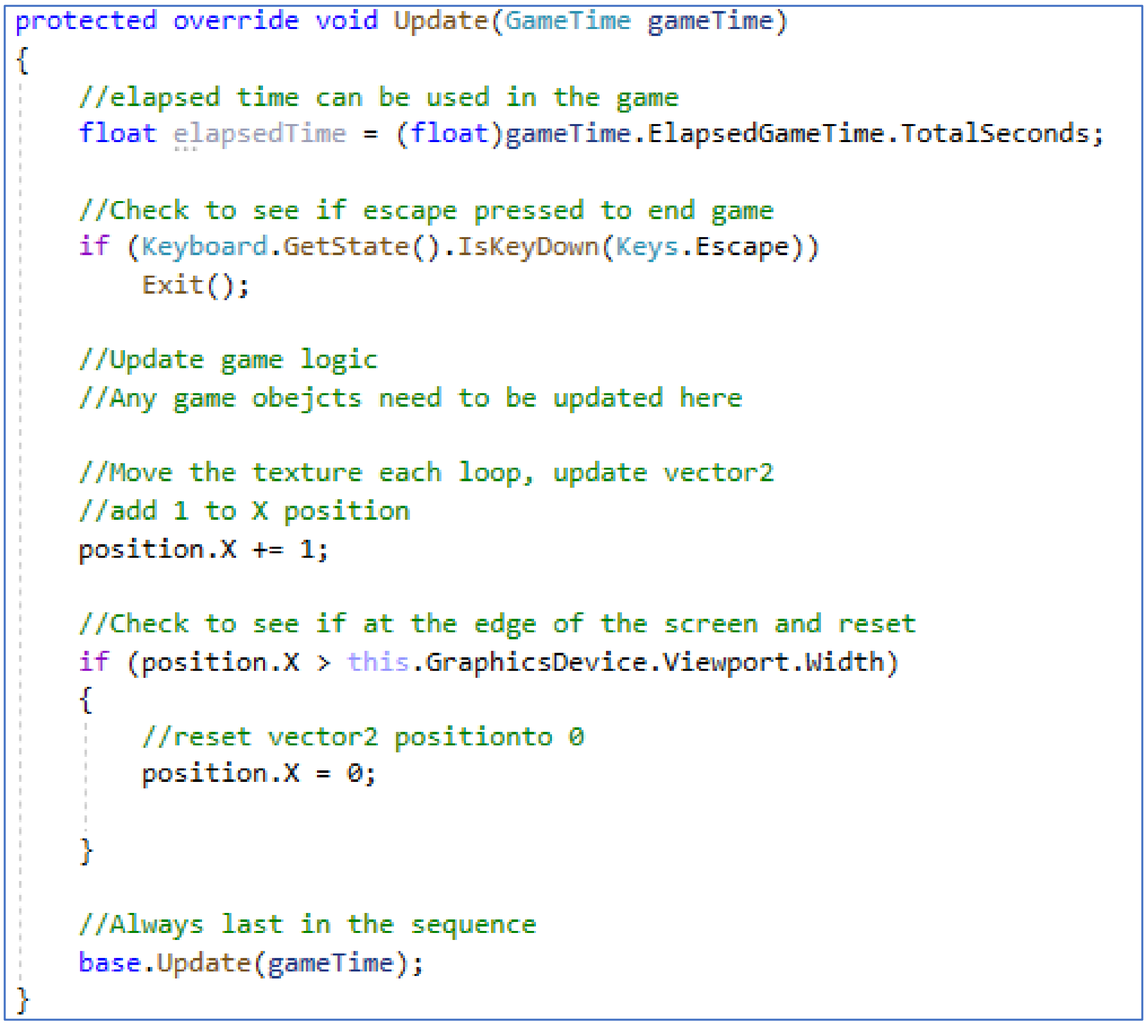
```
    Color[] colorData = new Color[50 * 50]; //create array to store colour of each texel
```

## Adding simple movement

Movement is controlled by changing (updating) the Vector2 position for each object on the screen. In this simple example the rectangle will move across the screen. This is done in the **Update()** method

### Final Update code

Here is all the code together to change the position and check the game update logic:

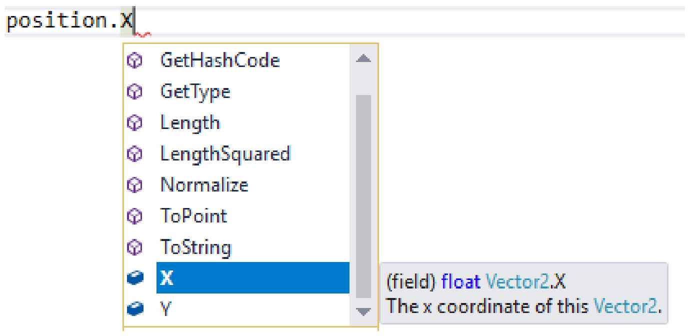


```
protected override void Update(GameTime gameTime)
{
    //elapsed time can be used in the game
    float elapsedTime = (float)gameTime.ElapsedGameTime.TotalSeconds;

    //Check to see if escape pressed to end game
    if (Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    //Update game logic
    //Any game obejcts need to be updated here

    //Move the texture each loop, update vector2
    //add 1 to X position
    position.X += 1;

    //Check to see if at the edge of the screen and reset
    if (position.X > this.GraphicsDevice.Viewport.Width)
    {
        //reset vector2 positionto 0
        position.X = 0;

    }

    //Always last in the sequence
    base.Update(gameTime);
}
```

***Creating movement code Explanation***

## Adding movement

In this example, the Vector2 object called ***position*** setup earlier has an X value.



```
position.X
```

(field) float Vector2.X
The x coordinate of this Vector2.

The **position.X** value can be changed each game loop and the texture re-drawn, giving the texture movement.

```
protected override void Update(GameTime gameTime)
{
    //Update game logic
    //Update vector2 position, add 1 to X each game loop
    position.X += 1;
```

## Adding simple logic

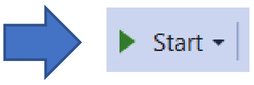
The **Update()** method is also where game logic is added. In this example the position is check to see if it is at the edge of the screen and set to zero.

```
//Check to see if at the screen edge and reset
if (position.X > this.GraphicsDevice.Viewport.Width)
{
    position.X = 0;
}
```

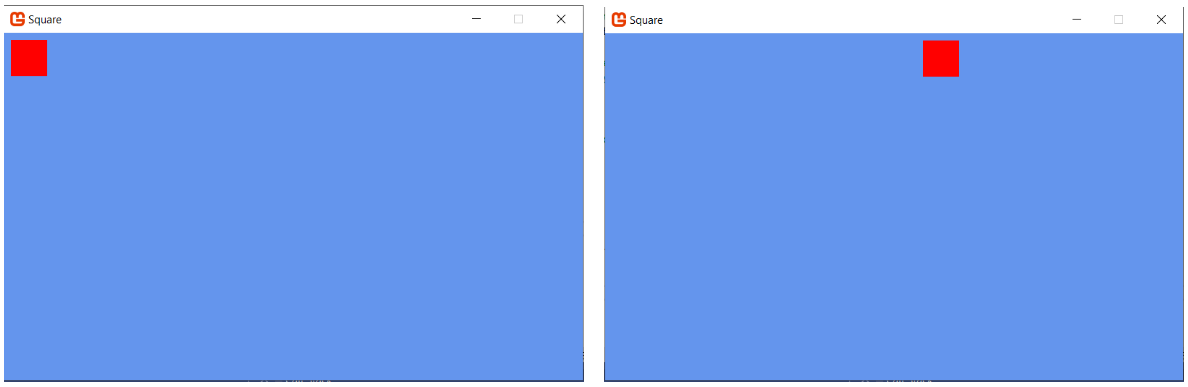
## Test, test and check

It is really important to check your code to see if it does what you expect. Testing is vital!

Test by pressing start



The shape should move across the screen and jump back when it goes off the edge:



## Experiment

Have a go at changing the following and test what happens:

**The Y position: Update ( )**

***Add a change to Y***

```
//Move the texture each loop, update vector2
//add 1 to X position
position.X += 1;

//add 1 to X position
position.Y += 1;
```

***Add a bottom of screen check***

Check to see if the texture moves off the bottom, otherwise it will move forever downwards!

```
if (position.Y > this.GraphicsDevice.Viewport.Width)
{
    //reset vector2 position to 0
    position.Y = 0;

}
```