At the heart of most games is checking for the collision of different objects in the game. These tutorials are about collision detection in 2D games. There are a variety of methods of performing collision detection. The simplest is bounding box, or rectangle, collision detection. To demonstrate this type of collision detection I will create a two player game of Pong. Pong was the first video game console. It played like ping pong. Each player controlled a paddle and tried to bounce a ball past the other player.

To get started, create a new XNA game called BoundingBoxCollision. You are going to need some graphics for the game. I suggest you use the ones I made. Download and unzip the graphics. Right click the Content folder in the game in the solution explorer. Select Add and then Existing item. Navigate to the folder that you unzipped the graphics and select the paddle.png, ball.png, and wall.png graphics.

Bounding Box Collision Detection Graphics

I will make a simple class to hold the objects in the game. This will reduce the amount of code that you write. Add a new class to your game called GameObject. This is the code for the GameObject class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

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

namespace BoundingBoxCollision
{
    public class GameObject
    {
        Texture2D texture;
        public Vector2 Position;
        public Vector2 Velocity;

        public Rectangle BoundingBox
        {
            get
            {
                return new Rectangle(
                    (int)Position.X,
                    (int)Position.Y,
                    texture.Width,
                    texture.Height);
            }
        }

        public GameObject(Texture2D texture, Vector2 position)
        {
            this.texture = texture;
            this.Position = position;
        }

        public GameObject(Texture2D texture, Vector2 position, Vector2 velocity)
        {
            this.texture = texture;
            this.Position = position;
            this.Velocity = velocity;
        }
```

```
        public void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(texture, Position, Color.White);
        }
    }
}
```

This class just has some things that are common to all game objects. This game is in 2D so I needed a Texture2D for the game object. All game objects have a position in the game. I use a Vector2 for the position. Vector2 gives you a little more control of the position of an object in your game world. For the same reason, I have a Vector2 for the velocity of the object in the game. What this means is that instead of moving an object 2 pixels or 3 pixels you can move an object 2.5 pixels. It is just a finer degree of control. The property BoundingBox returns a Rectangle that describes the game object. This will be used in detecting the collision between the different game objects. To find this rectangle you take the objects position and add the height and width of the object to its position. The width and height can be found using the Texture2D of the object.

To make creating game objects a little easier there are two constructors for the is class. The first one takes the Texture2D of the object and a Vector2 that describes its position. The second takes a Vector2 for the velocity of the object.

There is also a Draw method that can be called to draw the object. It takes as a parameter the current SpriteBatch object.

Now I will add two of objects to the game. I added images for the walls of the game. I did this so that I can do collision detection between the ball and the walls instead of checking to see if the ball will move off the screen. First thing to do is to add two fields for the walls. Near the SpriteBatch field add the following two fields to the Game1 class.

```
GameObject topWall;
GameObject bottomWall;
```

The next thing to do is to create the walls. I decided to make them in the LoadContent method. I will do that frequently because objects in the game need assets that are loaded through the Content Pipeline. Change the LoadContent method to the following.

```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    Texture2D wallTexture = Content.Load<Texture2D>("wall");
    topWall = new GameObject(
        wallTexture,
        Vector2.Zero);
    bottomWall = new GameObject(
        wallTexture,
        new Vector2(0, Window.ClientBounds.Height - wallTexture.Height));
}
```

The new code in the LoadContent method loads the image of the wall into the wallTexture variable. It then creates the two wall fields the first wall is positioned at the top of the screen and the second wall is

positioned at the bottom. To position the wall at the bottom I take the height of the window and subtract the height of the wall.

Now that we have the game objects we can draw them. That will be done in the Draw method of the Game1 class. Drawing in 2D is done with SpriteBatch objects. The drawing will take place between calls to Begin and End. Change the Draw method to the following.

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

    spriteBatch.Begin();

    topWall.Draw(spriteBatch);
    bottomWall.Draw(spriteBatch);

    spriteBatch.End();

    base.Draw(gameTime);
}
```

What I will add next are the two paddles. They will also be GameObjects. The one difference is they need input from the players. The player on the left side of the field will use the W and S keys to move their paddle up and down. The player on the right side of the field will use the Up and Down keys to move the their paddle up and down. To gather input for the paddles you will also need a KeyboardState field to read the state of the keyboard. Add the following fields near the wall fields in the Game1 class.

```
GameObject playerOne;
GameObject playerTwo;
KeyboardState keyboardState;
```

The next thing to do is to create the game objects for the players. That will be done in the LoadContent method, the same as the walls. Change the LoadContent method to the following.

```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    Texture2D wallTexture = Content.Load<Texture2D>("wall");
    topWall = new GameObject(
        wallTexture,
        Vector2.Zero);
    bottomWall = new GameObject(
        wallTexture,
        new Vector2(0, Window.ClientBounds.Height - wallTexture.Height));

    Texture2D paddleTexture = Content.Load<Texture2D>("paddle");
    Vector2 position;

    position = new Vector2(
        0,
        (Window.ClientBounds.Height - paddleTexture.Height) / 2);
    playerOne = new GameObject(paddleTexture, position);
```

```
    position = new Vector2(
        (Window.ClientBounds.Width - paddleTexture.Width),
        (Window.ClientBounds.Height - paddleTexture.Height) / 2);
    playerTwo = new GameObject(paddleTexture, position);
}
```

The first new code is field that holds the Texture2D for paddle. There is also a Vector2 to hold the position of the paddles. The initial position of the paddles will be in the left and right side of the screen centered vertically. The playerOne game object will be on the left side, its X position will be 0. Its Y position is found taking the height of the window, subtracting the height of the paddle and dividing the result by 2. To position the other paddle on the right side of the screen you take the width of the screen and subtract the width of the paddle. To center it vertically you use the same procedure as the other paddle.

Before getting to updating the paddles I will get to drawing them. That will be done in the Draw method. You just call the draw method of the playerOne and playerTwo objects like before. Change the Draw method to the following.

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

    spriteBatch.Begin();

    topWall.Draw(spriteBatch);
    bottomWall.Draw(spriteBatch);

    playerOne.Draw(spriteBatch);
    playerTwo.Draw(spriteBatch);

    spriteBatch.End();

    base.Draw(gameTime);
}
```

The next step is getting the paddles so that they can move up and down the screen. In the Update method I will check to see if the keys for moving the paddles are down. If they are down I will try moving the paddle up or down. I will then call a method, CheckPaddleWallCollision, that will detect collision between the paddles and the walls. If there is a collision between them the paddle will be moved so that there isn't. Change the Update method to the following. I will give you the code for the CheckPaddleWallCollision method shortly.

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    keyboardState = Keyboard.GetState();

    if (keyboardState.IsKeyDown(Keys.W))
        playerOne.Position.Y -= 10f;

    if (keyboardState.IsKeyDown(Keys.S))
        playerOne.Position.Y += 10f;
```

```
    if (keyboardState.IsKeyDown(Keys.Up))
        playerTwo.Position.Y -= 10f;

    if (keyboardState.IsKeyDown(Keys.Down))
        playerTwo.Position.Y += 10f;

    CheckPaddleWallCollision();
    base.Update(gameTime);
}
```

The Update method first checks to see if the W key is currently down. If it is, you want to move the paddle up and subtract 10 pixels from the Y value of its position. If the S key is down you want to move the paddle down so you add 10 pixels to the Y value of its position. You do the same thing for the second player if the Up key is currently down or the Down key is currently down. I then call the CheckPaddleWallCollision method to see if the paddles collide with the walls. This is the code for the CheckPaddleWallCollision method.

```
private void CheckPaddleWallCollision()
{
    if (playerOne.BoundingBox.Intersects(topWall.BoundingBox))
        playerOne.Position.Y = topWall.BoundingBox.Bottom;

    if (playerOne.BoundingBox.Intersects(bottomWall.BoundingBox))
        playerOne.Position.Y = bottomWall.BoundingBox.Y
            - playerOne.BoundingBox.Height;

    if (playerTwo.BoundingBox.Intersects(topWall.BoundingBox))
        playerTwo.Position.Y = topWall.BoundingBox.Bottom;

    if (playerTwo.BoundingBox.Intersects(bottomWall.BoundingBox))
        playerTwo.Position.Y = bottomWall.BoundingBox.Y
            - playerTwo.BoundingBox.Height;
}
```

This is the first collision detection code and it uses bounding boxes, or Rectangles. The rectangle class has a method called Intersects that will tell if two Rectangles overlap. If the Rectangles overlap, there is a collision between the two objects. This is illustrated in the diagram below. The black rectangle around the paddle on the left intersects with the black rectangle of the wall so the paddle collides with the wall. The black rectangle of the paddle on the right doesn't intersect with the black rectangle of the wall so there is no collision.

If the paddle collides with the top wall I set the position of the paddle to be the bottom of the wall. If the paddle collides with the bottom wall I set the position of the paddle to be the Y position of the wall

minus the height of the paddle. Setting the paddle to these values keeps them from going past the wall. The top is easier than the bottom because for the bottom you need to take the height of the paddle into account when you are moving the paddle so it no longer collides with the wall. That is why I had to subtract the height of the paddle.

Now it is time to add in the ball. The ball is a little more complicated than the paddles. It is not detecting if there is a collision but the ball should deflect at an appropriate angle when it collides with an object. If the ball gets past player one's or player two's paddle it needs to be reset.

To start with you need to add a field for the ball and create the ball. The ball will be created like the other objects in the game, in the LoadContent method. You will also want to draw the ball like all of the other game objects. Add the following field to the Game1 class with the other GameObject fields and change the LoadContent and Draw methods to the following.

```
GameObject ball;

protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    Texture2D wallTexture = Content.Load<Texture2D>("wall");
    topWall = new GameObject(
        wallTexture,
        Vector2.Zero);
    bottomWall = new GameObject(
        wallTexture,
        new Vector2(0, Window.ClientBounds.Height - wallTexture.Height));

    Texture2D paddleTexture = Content.Load<Texture2D>("paddle");
    Vector2 position;

    position = new Vector2(
        0,
        (Window.ClientBounds.Height - paddleTexture.Height) / 2);
    playerOne = new GameObject(paddleTexture, position);

    position = new Vector2(
        (Window.ClientBounds.Width - paddleTexture.Width),
        (Window.ClientBounds.Height - paddleTexture.Height) / 2);
    playerTwo = new GameObject(paddleTexture, position);

    Texture2D ballTexture = Content.Load<Texture2D>("ball");

    position = new Vector2(
        playerOne.BoundingBox.Right + 1,
        (Window.ClientBounds.Height - ballTexture.Height) / 2);

    ball = new GameObject(
        ballTexture,
        position,
        new Vector2(8f, -8f));
}


protected override void Draw(GameTime gameTime)
```

```
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();

    topWall.Draw(spriteBatch);
    bottomWall.Draw(spriteBatch);

    playerOne.Draw(spriteBatch);
    playerTwo.Draw(spriteBatch);

    ball.Draw(spriteBatch);

    spriteBatch.End();

    base.Draw(gameTime);
}
```

The ball uses the second constructor of the GameObject class so it needs a position and a velocity. I set the initial position to be just to the right of the paddle, using the Right property of paddle's bounding box. I also add 1 to X value to make sure in the first frame there will not be a collision between the paddle and the ball. I found the Y position the same way that I found the Y value of the paddle's position. I set the ball to move right 8 pixels and up 8 pixels.

Now it is time to add the logic for the ball into the game. I will do that in the Update method. Each frame of the game you want to move the ball by its velocity. You will then want to check for collision between the ball and the other objects in the game. It would be a good idea to check for collision between the ball and the paddles after the players have moved their paddles. I will write a method CheckBallCollision that will check for the collision of the ball and other game objects. Change the Update method to the following.

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    ball.Position += ball.Velocity;

    keyboardState = Keyboard.GetState();

    if (keyboardState.IsKeyDown(Keys.W))
        playerOne.Position.Y -= 10f;

    if (keyboardState.IsKeyDown(Keys.S))
        playerOne.Position.Y += 10f;

    if (keyboardState.IsKeyDown(Keys.Up))
        playerTwo.Position.Y -= 10f;

    if (keyboardState.IsKeyDown(Keys.Down))
        playerTwo.Position.Y += 10f;

    CheckPaddleWallCollision();
    CheckBallCollision();
```

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

The changes to the Update method are at the start of the method update the ball's position using its velocity. The second is after the call to CheckPaddleWallCollision there is a call to CheckBallCollision. The code for the CheckBallCollision method follows.

```
private void CheckBallCollision()
{
    if (ball.BoundingBox.Intersects(topWall.BoundingBox))
    {
        ball.Velocity.Y *= -1;
        ball.Position += ball.Velocity;
    }

    if (ball.BoundingBox.Intersects(bottomWall.BoundingBox))
    {
        ball.Velocity.Y *= -1;
        ball.Position += ball.Velocity;
    }

    if (ball.BoundingBox.Intersects(playerOne.BoundingBox))
    {
        ball.Velocity.X *= -1;
        ball.Position += ball.Velocity;
    }

    if (ball.BoundingBox.Intersects(playerTwo.BoundingBox))
    {
        ball.Velocity.X *= -1;
        ball.Position += ball.Velocity;
    }
}
```

This method just checks if the bounding box of the ball intersects with the bounding boxes of the walls and the paddles. If the ball collides with a wall the direction the ball is traveling vertically should be reflected. This can be done simply by multiplying the Y value of its velocity by -1. Similarly, if the ball collides with the paddle you want to change the direction it is traveling horizontally. This can be done by multiplying the X value of its velocity by -1. It is also a good idea after changing the velocity to update the position of the ball using the new velocity.

The game works fine except for one thing. If the ball gets past one of the paddles it should reset the game to its starting point. To tell if the ball has gone off the left side of the screen you should check if the X value of the position is less than zero minus the width of the ball. To tell if the ball has gone off the right side of the screen you should check if the X value of the balls position is greater than the width of the window. I will do that in the CheckBallCollision method. If the ball does go off the screen I will call a method SetInStartPosition. Change the CheckBallCollision method to the following and add the SetInStartPosition method.

```
private void CheckBallCollision()
{
    if (ball.BoundingBox.Intersects(topWall.BoundingBox))
    {
        ball.Velocity.Y *= -1;
        ball.Position += ball.Velocity;
```

```
    }

    if (ball.BoundingBox.Intersects(bottomWall.BoundingBox))
    {
        ball.Velocity.Y *= -1;
        ball.Position += ball.Velocity;
    }

    if (ball.BoundingBox.Intersects(playerOne.BoundingBox))
    {
        ball.Velocity.X *= -1;
        ball.Position += ball.Velocity;
    }

    if (ball.BoundingBox.Intersects(playerTwo.BoundingBox))
    {
        ball.Velocity.X *= -1;
        ball.Position += ball.Velocity;
    }

    if ((ball.Position.X < -ball.BoundingBox.Width)
        || (ball.Position.X > Window.ClientBounds.Width))
        SetInStartPostion();

}

private void SetInStartPostion()
{
    playerOne.Position.Y = (
        Window.ClientBounds.Height -
        playerOne.BoundingBox.Height) / 2;

    playerTwo.Position.Y = (
        Window.ClientBounds.Height -
        playerTwo.BoundingBox.Height) / 2;

    ball.Position.X = playerOne.BoundingBox.Right + 1;

    ball.Position.Y = (
        Window.ClientBounds.Height -
        ball.BoundingBox.Height) / 2;

    ball.Velocity = new Vector2(8f, -8f);
}
```

The SetInStartPosition method just sets values back to their starting position. The paddles are centered vertically on the screen. The ball is positioned just to the right of player one's paddle. The velocity of the ball is set back to its starting value as well.

That is it for the Pong and bounding box collision tutorial. I will be writing more basic collision detection tutorials. Things you can do to improve the game is allow the player to play against the computer. You can also improve the code for deflecting the ball upon collision. Another thing is you can have the speed of the ball increase every so often. You can find the code for the Game1 and GameObject classes below.

Bounding Box Collision Detection Source Code