Game Development

Practical 3
Breaking Out Game-Part 2

Year 4

Load up your game from the previous practical. You need to add the brick to the content project. Instead of having many bricks of different colors, you can have one brick that tint for different colors.

It would be a good idea to have a class for bricks. Right click your **BreakingOut** project, select **Add** and then **Class**. Name this new class **Brick**. This is the code for the brick class.

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

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

namespace BreakingOut
{
   class Brick
   {
      Texture2D texture;
      Rectangle location;
      Color tint;
      bool alive;

      public Rectangle Location
      {
         get { return location; }
      }

      public Brick(Texture2D texture, Rectangle location, Color tint)
      {
         this.texture = texture;
         this.location = location;
         this.tint = tint;
         this.alive = true;
      }

      public void CheckCollision(Ball ball)
      {
      }
```

```
        public void Draw(SpriteBatch spriteBatch)
        {
            if (alive)
                spriteBatch.Draw(texture, location, tint);
        }
    }
}
```

The next thing to do is to draw some bricks. For that you need to add few fields. You need to know how many columns and rows of bricks you have. It would be a good idea to store the texture for the bricks. Add these four fields just below the **screenRectangle** field.

```
int bricksWide = 10;
int bricksHigh = 5;
Texture2D brickImage;
Brick[,] bricks;
```

In the constructor for your game the first thing that happens is that a **GraphicsDeviceManager** is created. This **GraphicsDeviceManager** helps your game talk with the video card. The class has 2 properties **PreferredBackBufferWidth** and **PrefferedBackBufferHeight**. You can set them to be the width and height of the window your game plays in. Change the constructor for the **Game1** class to the following.

```
public Game1()
{
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            graphics.PreferredBackBufferWidth = 750;
            graphics.PreferredBackBufferHeight = 600;
            screenRectangle = new Rectangle(
                                0,
                               0,
                              graphics.PreferredBackBufferWidth,
                              graphics.PreferredBackBufferHeight);
}
```

Set the width to 750, because 10 times the width of a brick works out to be 750. 600 for the height of the back buffer was a bit of an arbitrary choice. Many monitors have a height of 728 pixels. 600 pixels, plus the title bar and height of the window looked “good” on the screen.

Now would be a good time to explain some things about game programming. You will often hear the terms frame, frame rate, back buffer, and many others when people talk about games. The first term frame, what does that mean exactly? A frame of a game is where you update game objects and draw

them, in simplest terms. Updating objects means moving them, checking for collisions, and moving objects relative to the player's input, applying physics, and many more things. You then draw the objects in your game.

The frame rate is how many frames your game processes per second. It is best to try and have your game run at a constant frame rate across all computers because computers differ in processing power. If a game runs perfect on your computer but your friend has a fast computer and you don't have your game try and run at a constant frame rate it will run much faster on your friend's computer.

If you look at the image for the bricks on the next page it is a grey scale image. It is made up of various shades of grey. In XNA you can tint 2D objects drawn with the **Draw** method of the **SpriteBatch** class. Change the **LoadContent** method to the following.

```
brickImage = Content.Load<Texture2D>("brick");
```

Now you need to actually create the bricks. Add the relevant statements in the **StartGame** method. You can change the **StartGame** method to the following.

```
private void StartGame()
{
        paddle.SetInStartPosition();
        ball.SetInStartPosition(paddle.GetBounds());

        bricks = new Brick[bricksWide, bricksHigh];

        for (int y = 0; y < bricksHigh; y++)
        {
           Color tint = Color.White;

           switch (y)
           {
              case 0:
                 tint = Color.Blue;
                 break;
              case 1:
                 tint = Color.Red;
                 break;
              case 2:
                 tint = Color.Green;
                 break;
              case 3:
                 tint = Color.Yellow;
                 break;
              case 4:
```

```
                tint = Color.Purple;
                break;
            }

            for (int x = 0; x < bricksWide; x++)
            {
                bricks[x, y] = new Brick(
                    brickImage,
                    new Rectangle(
                        x * brickImage.Width,
                        y * brickImage.Height,
                        brickImage.Width,
                        brickImage.Height),
                    tint);
            }
        }
}
```

Now it is time to actually draw them. You will draw them in the same place that you drew the paddle and the ball, in the **Draw** method. Change the **Draw** method of your game class to the following.

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

        spriteBatch.Begin();

        paddle.Draw(spriteBatch);
        ball.Draw(spriteBatch);

        foreach (Brick brick in bricks)
            brick.Draw(spriteBatch);

          spriteBatch.End();

        base.Draw(gameTime);
    }
```

If you build and run the game now you will have the paddle, ball, and bricks all being drawn. You will see that the ball disappears when it gets to the bricks. That is because of the nature of drawing things in 2D. The order in which you draw things is important. Think of it like this. If you take a piece of paper and put it on the floor and then you put another piece of paper on top of part of it the second piece of paper hides part of the first piece. If you place another on top of them it covers, or hides, the parts of the first two, and so on. Try swapping the order in which you draw items to the following.

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

        foreach (Brick brick in bricks)
           brick.Draw(spriteBatch);

        paddle.Draw(spriteBatch);
        ball.Draw(spriteBatch);

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

Now if you build and run your game the ball will be drawn over top of the bricks. You can see that the order in which you draw things is important.

Colors in XNA are composed of four values. There are red, blue, green, and alpha components. Now ignore the alpha component and focus on the red, blue, and green components. The values for these components can be represented using numbers between 0 and 255, or a byte. So, a color is made up of four bytes, including the alpha channel. As you move down from 255 to 0 you are using less of that component. To tint an object you use various levels of red, blue, and green. White is represented using full red, blue, and green. Using it to tint an object returns the original object. If you were to lower red toward 0 you would be removing red from that object. If you had 0 red and 255 green and blue the object would be tinted cyan. By tinting a grey scale image using different colors you get varying shades of that color. So, if you tint the grey scale brick using blue the grey scale brick will be made up of shades of blue.