

# Coding for Game Development

## Section Two

### XNA Game Programming

#### Basic Sprite Handling

1/20

# What is a Sprite?

- Sprites are a basic part of 2D games.
- A sprite is a small self contained graphical element which we can move independently around the game area.
- In an XNA game, we update the position of each sprite in the **Update()** method and then draw their new positions in the **Draw()** method.
- XNA provides several support classes and methods which makes sprite drawing very simple.

2/20

# Sprite Graphics

- We usually create the sprite graphics using some graphical design program and add the file to the content pipeline for the project.
- We normally choose a file format which supports transparency (preferably *alpha* transparency).
- The images are loaded as **Texture2D** objects, which we declare as fields of the **Game1** class. The boilerplate code creates some standard objects which we need:

```
GraphicsDeviceManager graphics; // Boilerplate
SpriteBatch spriteBatch; // Boilerplate

Texture2D balloon; // Added sprite
Texture2D dart; // Added sprite
```

3/20

# The SpriteBatch Class

- XNA provides the **SpriteBatch** class which looks after all sprite drawing and re-drawing operations.
- We perform all these operations within the **Draw()** method.
- Typically, we make calls like this:

```
spriteBatch.Begin(); // Compulsory call to start

spriteBatch.Draw(... the background image ...);
spriteBatch.Draw(... first sprite ...);
spriteBatch.Draw(... second sprite ...);

spriteBatch.End(); // Compulsory call to finish
```

4/20

# Loading Sprite Files

- We load all image files in the **LoadContent()** method, following the **TODO** flag:

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

    // TODO: use this.Content to load your game content here

    balloon = this.Content.Load<Texture2D>("Balloon");
    dart = this.Content.Load<Texture2D>("Dart");
}

```

Graphic file names (no file extensions)

5/20

# The SpriteBatch Draw() method

- There are several versions of **spriteBatch.Draw()**, but the simplest takes this form:

```
Vector2 position = new Vector2(100, 400); // screen x and y
spriteBatch.Draw(theSprite, position, Color.White);
```

- Note that we use a **Vector2** object to store the x-y position of the sprite on the screen.
- This version draws the entire sprite image onto the screen. The top left sprite pixel is drawn at the x-y position specified in the **Vector2**.
- We can specify a colour *tint* for the sprite, or **White** to leave its colour unchanged.

6/20

## A Simple Sprite Game

- Over the next few slides we will look at a complete program which uses sprites.
- Of course, much of the code is automatically generated boilerplate – we simply modify this boilerplate to get the appropriate functionality for our game.

7/20

## Game Fields

- We must define a **Texture2D** as a global field of the game class.
- We also add **double** fields for the sprite's screen position and velocity.

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
}
Texture2D balloon;
double xPos, yPos;
double xVel = 2.0, yVel = 2.0;
:::
}
```

Note: **double** recommended rather than **int**

8/20

## Updating the Sprite Position

- In the **Update()** method we move the sprite a few pixels at a time:

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (.....)
        this.Exit();

    // TODO: Add your update logic here

    xPos += xVel;
    yPos += yVel;
}
base.Update(gameTime);
```

9/20

## The Draw() Method

- Inside the **Draw()** method, we draw the sprite at its new position:

```
protected override void Draw(GameTime gameTime)
{
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here

    spriteBatch.Begin();
    Vector2 position = new Vector2((float) xPos, (float) yPos);
    spriteBatch.Draw(theSprite, position, Color.White);
    spriteBatch.End();

    base.Draw(gameTime);
}
```

Note: type casts required

10/20

## Game Window Size

- We can obtain the size of the game window using a supplied object called **Window.ClientBounds**:

```
int screenWidth = Window.ClientBounds.Width;
int screenHeight = Window.ClientBounds.Height;
```

- We can use this to detect whether a sprite is going off the edge of the screen by putting this code in the **Update()** method:

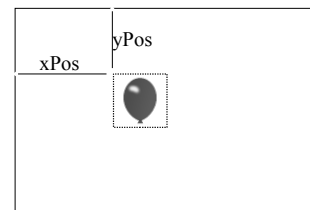
```
if (xPos < 10 || xPos > screenWidth - 50)
{
    xVel = -xVel;
}
```

- Now the sprite bounces off the edge!

11/20

## Window and Sprite Size Issues

- The sprite **Draw()** method places the top left corner of the sprite image at the specified x-y locations:



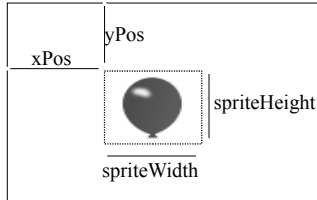
- We need to take the sprite's width and height into account when deciding if the sprite has hit the edge of the screen.

12/20

## Scaling a Sprite

- We can change (scale) the width and height of a sprite by using an alternative form of **Draw()**. In this version we replace the **Vector2** by a **Rectangle**:

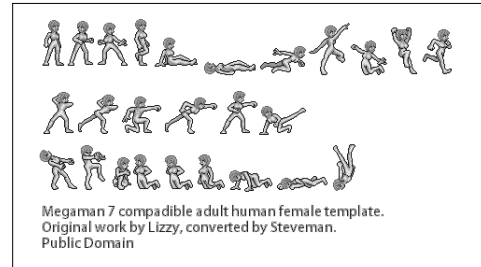
```
Rectangle screenRect =
    new Rectangle((float) xPos, (float) yPos,
        spriteWidth, spriteHeight);
spriteBatch.Draw(theSprite, screenRect, Color.White);
```



13 / 20

## Sprite Animation

- Sprite animation is usually performed by taking images from a sprite sheet, such as this example:



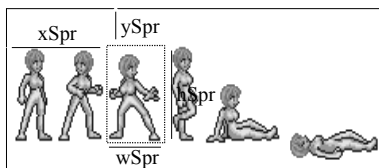
- The whole sheet is loaded as one **Texture2D** graphic.

14 / 20

## Drawing Images from a Sprite Sheet

- We can use an alternative version of **Draw()** to select a rectangle from a sprite sheet. The **spriteSource** defines the area of the sprite texture to be drawn.

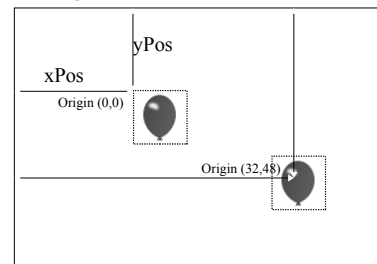
```
Vector2 position = new Vector2((float) xPos, (float) yPos);
Rectangle spriteSource =
    new Rectangle(xSpr, ySpr, wSpr, hSpr);
spriteBatch.Draw(theSprite, position,
    spriteSource, Color.White);
```



15 / 20

## Sprite Origin

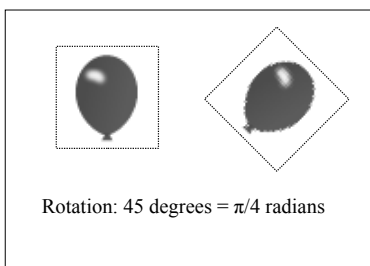
- By default, the sprite origin is the top left hand corner, but this can be changed if we wish.
- The sprite is positioned on the screen, so that its origin is placed at the x-y location on the screen.



16 / 20

## Sprite Rotation

- We can rotate the sprite about its origin point.
- The amount of rotation is specified in *radians* not degrees.



17 / 20

## Sprite Scaling and Rotating

- If we're really ambitious we can draw a sprite with optional size scaling and rotation using a more complex version of **Draw()**:

```
public void Draw (
    Texture2D texture,
    Vector2 screenPosition,
    Rectangle spriteSourceRectangle,
    Color color, // colour tint
    float rotation, // rotation (radians)
    Vector2 spriteOrigin, // 0.0 is top left corner of sprite
    float scalingFactor, // Make the sprite bigger or smaller
    SpriteEffects effects, // Optional horizontal/vertical flip
    float layerDepth // 0.0 (front) .. 1.0 (back)
)
```

18 / 20

## Layers of Sprites

- When we have multiple sprites, we should draw them in order starting with the background image first, and then sprites going from back to front.
- Generally, the mouse cursor and scores are drawn last.
- If several sprites overlap on the same area of the screen, then the sprites which are drawn later will obscure the sprites which are drawn earlier.
- Some versions of the sprite **Draw()** method include a **layerDepth** parameter, which *sometimes* can be used to change the default drawing order; however, it's probably best if you don't use this particular feature.

19/20

20/20