UNIVERSITY OF Hull

Mad Game Development

Rob Miles

Department of Computer Science

Introduction

- XNA Overview
- Creating a projectile
  - Texture and position
  - Aiming and firing
  - Updating during movement
  - Detecting collisions
- Panning the screen
  - Using a Transformation Matrix
  - Panning when aiming
  - Following the ball
- Game State

Projectile

- The ball is a texture and a rectangle
- We give the ball a position using floating point values
  - These are converted to integers to position the draw rectangle
- Velocity is the change in position each time the ball is updated
- Acceleration is the change in velocity each time the ball is updated

Ball Data

```
Texture2D ballTexture;
Rectangle ballRectangle;
float ballX, ballY;
float ballXSpeed, ballYSpeed;
float ballXMaxSpeed, ballYMaxSpeed;
float ballXAccel, ballYAccel;
```

- This is the information stored about the ball
- The texture gives the ball appearance
- The rectangle gives the ball position

Ball Update

```
// Update the X and Y position of the ball based on its
// present speed
ballX += ballXSpeed;
ballY += ballYSpeed;

// Apply acceleration to the ball speed
ballXSpeed += ballXAccel;
ballYSpeed += ballYAccel;
```

- This is the code that does the physics for the ball movement
- The acceleration acts on the ball once it has been fired
- This code lives in the Update method which is called 60 times a second by XNA to update game objects

Ball Position

```
// Position the draw rectangle to the nearest integer
ballRectangle.X = (int)(ballX + 0.5f);
ballRectangle.Y = (int)(ballY + 0.5f);
```

- This is the code that positions the ball rectangle based on the floating point values we calculate
- We need to do this because the rectangle is used to draw the ball, and that is positioned using integers
- However, we need to do floating point calculations when using the speed and acceleration

Ball Drawing

```
spriteBatch.Draw(ballTexture, ballRectangle, Color.White);
```

- This draws the ball on the screen
- We have a similar draw operation for the other items in the game
- Drawing is performed in the Draw method which is called automatically by XNA
- This must appear in the middle of a `spriteBatch.Begin()` – `spriteBatch.End()` sequence

Ball Launching

```
// Press A to fire the ball
if (gamePad.Buttons.A == ButtonState.Pressed)
{
    // Set the speed of the ball from the left thumbstick
    // position
    ballXSpeed = gamePad.ThumbSticks.Left.X *
                                        ballXMaxSpeed;
    ballYSpeed = -gamePad.ThumbSticks.Left.Y *
                                        ballYMaxSpeed;
    aimingShot = false;
}
```

- This sets the initial speed of the ball when fire is pressed
- We use the left thumbstick to set the speed values

The Target

```
/// <summary>
/// Texture of the single target
/// </summary>
Texture2D targetTexture;
/// <summary>
/// Used to draw the target and detect collisions
/// </summary>
Rectangle targetRectangle;
```

- The target is another sprite
- It just has a texture and a position rectangle

Detecting Collisions

```
// If we have hit a target play a sound and reset the ball
if (ballRectangle.Intersects(targetRectangle))
{
    bangSound.Play();
    resetBall();
}
```

- We can detect collisions by using rectangle intersection
- This code tests to see if the ball has hit the target
- If it has it plays an explosion sound and resets the ball

Scrolling Background

```
Texture2D backTexture;
Rectangle backRectangle;
float backScrollSpeed;
```

- The background is another texture and rectangle
- But this rectangle is much bigger than the display
- We want to be able to scroll the display around and view different parts of the playfield
- We can do this by using a transformation matrix on the SpriteBatch draw

Transformation Matrix

```
Matrix transformation;

transformation = Matrix.CreateTranslation(xOffset, 0, 0);
```

- A transformation matrix is a lump of maths that can be applied to a drawing operation to change the way it looks
  - Scaling
  - Rotation
  - Translation
- We just want to translate our view when we draw it
- XNA will create a translation matrix for us

Drawing with a Transformation Matrix

```
spriteBatch.Begin(SpriteBlendMode.AlphaBlend,
                  SpriteSortMode.Immediate,
                  SaveStateMode.None, transformation);

// transformed drawing

spriteBatch.End();
```

- We can ask SpriteBatch to use this matrix to transform the drawing operation
- All the drawing performed will be moved according to the matrix supplied
- We can follow this with an “untransformed” draw if we wish

Panning the Screen with the ball

```
// Check if the ball has reached the right hand edge
if (ballRectangle.Right + scrollMargin > width)
{
  // need to scroll the screen
  xOffset = width - (ballRectangle.Right + scrollMargin) ;
}
```

- We set the offset to pan the screen when the ball reaches the right hand edge
- This is the offset value used to create the translation matrix

Panning the Screen when aiming

```
xOffset = xOffset - gamePad.ThumbSticks.Right.X *
                    backScrollSpeed;
```

- When aiming we use the X value of the right thumbstic to change the xOffset value
- This lets the user pan left and right
- This version of the code does not stop the user panning right off the background image

Game State

```
if (aimingShot)
{
    // Aiming the shot
}
else
{
    // firing the shot
}
```

- I'm using a simple boolean to control the game state
- You might want to add other game states as well

Stuff you Need to Do

- Add some more assets
  - Targets to avoid
  - Targets to hit
- Sort out the edges
  - Stop the player from panning off the screen
- Add more game states
- Have fun

Sample Code

- All the code you have seen

Questions?

www.destructiongolf.com