Animating Textures

Slides built from Carter Chapter 10
Animating Sprites (textures)

Images from wikipedia.org
Animating Sprites (textures)

Images from wikipedia.org
Lets Add to Our XELibrary

- Going to add a CelAnimationManager class
  - Want to pass in a sheet of animated sprites
  - Tell it where the animations are
  - Have it play them for us at the frame rate we wish
- Sounds like a GameService
- This could also be a content processor if we wished
- What do we need to do?
How do we need to process this?
How about more than 1 anim?

- Suppose we had 4 animations in a single file
- How would we do it?
public sealed partial class CelAnimationManager :
    Microsoft.Xna.Framework.GameComponent
{
    private Dictionary<string, CelAnimation> animations =
        new Dictionary<string, CelAnimation>();
    private Dictionary<string, Texture2D> textures = new
        Dictionary<string, Texture2D>();
    private ContentManager content;

    private string contentPath;

    Use a Dictionary <key, value> to store the animations and the textures
Constructor

public CelAnimationManager(Game game, string contentPath)
    : base(game)
{
    this.contentPath = contentPath;

    if (this.contentPath.LastIndexOf('\\') < this.contentPath.Length - 1)
        this.contentPath += "\\";

    content = new ContentManager(game.Services);
}
public override void Initialize()
{
    base.Initialize();
}

Initialize()
AddingAnimations

- Suppose we have a texture with just a single animation.
  - Name of animation (its key)
  - Name of texture file
  - Number of rows and columns in the texture (cel count)
  - Frames per second.
public void AddAnimation(string animationKey, string textureName, CelCount celCount, int framesPerSecond) {
    if (!textures.ContainsKey(textureName))
        textures.Add(textureName, content.Load<Texture2D>(contentPath + textureName));
    int celWidth = (int)(textures[textureName].Width / celCount.NumberOfColumns);
    int celHeight = (int)(textures[textureName].Height / celCount.NumberOfRows);
    int numberOfCels = celCount.NumberOfColumns * celCount.NumberOfRows;

    // we create a cel range by passing in start location of 1,1
    // and end with number of column and rows
    // 2,1 = 1,1,2,1 ; 4,2 = 1,1,4,2
    AddAnimation(animationKey, textureName, new CelRange(1, 1, celCount.NumberOfColumns, celCount.NumberOfRows),
                  celWidth, celHeight, numberOfCels, framesPerSecond);
}
public void AddAnimation(string animationKey, string textureName, CelRange celRange, int celWidth, int celHeight, int numberOfCels, int framesPerSecond) {
    CelAnimation ca = new CelAnimation(textureName, celRange, framesPerSecond);

    if (!textures.ContainsKey(textureName))
        textures.Add(textureName, content.Load<Texture2D>(contentPath + textureName));

    ca.CelWidth = celWidth;
    ca.CelHeight = celHeight;
    ca.NumberOfCels = numberOfCels;
    ca.CelsPerRow = textures[textureName].Width / celWidth;

    if (animations.ContainsKey(animationKey))
        animations[animationKey] = ca;
    else
        animations.Add(animationKey, ca);
}
public void ToggleAnimation(string animationKey)
{
    if (animations.ContainsKey(animationKey))
    {
        animations[animationKey].Paused = !animations[animationKey].Paused;
    }
}
public override void Update(GameTime gameTime) {
    foreach (KeyValuePair<string, CelAnimation> animation in animations) {
        CelAnimation ca = animation.Value;

        if (ca.Paused)
            continue; //no need to update this animation, check next one
        ca.TotalElapsedTime += (float)gameTime.ElapsedGameTime.TotalSeconds;
        if (ca.TotalElapsedTime > ca.TimePerFrame)
            { 
                ca.Frame++;
                ca.Frame = ca.Frame % (ca.NumberOfCels); //min: 0, max: total cels
                ca.TotalElapsedTime -= ca.TimePerFrame; //reset our timer
            }
    }
    base.Update(gameTime);
}
public void Draw(GameTime gameTime, string animationKey, SpriteBatch batch, Vector2 position) {
    Draw(gameTime, animationKey, batch, animations[animationKey].Frame, position);
}

public void Draw(GameTime gameTime, string animationKey, SpriteBatch batch, int frame, Vector2 position) {
    Draw(gameTime, animationKey, batch, animations[animationKey].Frame, position, Color.White);
}
public void Draw(GameTime gameTime, string animationKey, SpriteBatch batch, int frame, Vector2 position, Color color)
{
    if (!animations.ContainsKey(animationKey))
        return;

    CelAnimation ca = animations[animationKey];

    //first get our x increase amount (add our offset-1 to our current frame)
    int xincrease = (ca.Frame + ca.CelRange.FirstCelX - 1);
    //now we need to wrap the value so it will loop to the next row
    int xwrapped = xincrease % ca.CelsPerRow;
    //finally we need to take the product of our wrapped value and a cel's width
    int x = xwrapped * ca.CelWidth;
Draw()

// to determine how much we should increase y, we need to look at how
// much we increased x and do an integer divide
    int yincrease = xincrease / ca.CelsPerRow;
// now we can take this increase and add it to our Y offset-1 and multiply
// the sum by our cel height
    int y = (yincrease + ca.CelRange.FirstCelY - 1) * ca.CelHeight;

    Rectangle cel = new Rectangle(x, y, ca.CelWidth, ca.CelHeight);

    batch.Draw(textures[ca.TextureName], position, cel, color);
}
CellAnimation class

public class CelAnimation
{
    private string textureName;
    private CelRange celRange;
    private int framesPerSecond;
    private float timePerFrame;

    public float TotalElapsedTime = 0.0f;
    public int CelWidth;
    public int CelHeight;
    public int NumberOfCels;
    public int CelsPerRow;
    public int Frame;
    public bool Paused = false;
}
Constructor

```csharp
public CelAnimation(string textureName, CelRange celRange, int framesPerSecond)
{
    this.textureName = textureName;
    this.celRange = celRange;
    this.framesPerSecond = framesPerSecond;
    this.timePerFrame = 1.0f / (float)framesPerSecond;
    this.Frame = 0;
}
```
attributes

public string TextureName {
    get { return (textureName); } }

public CelRange CelRange {
    get { return (celRange); }}

public int FramesPerSecond {
    get { return (framesPerSecond); }}

public float TimePerFrame {
    get { return (timePerFrame); }}
CelCount struct

public struct CelCount
{
    public int NumberOfColumns;
    public int NumberOfRows;

    public CelCount(int numberOfColumns, int numberOfRows)
    {
        NumberOfColumns = numberOfColumns;
        NumberOfRows = numberOfRows;
    }
}
CelRange struct

public struct CelRange
{
    public int FirstCelX;
    public int FirstCelY;
    public int LastCelX;
    public int LastCelY;

    public CelRange(int firstCelX, int firstCelY, int lastCelX, int lastCelY)
    {
        FirstCelX = firstCelX;
        FirstCelY = firstCelY;
        LastCelX = lastCelX;
        LastCelY = lastCelY;
    }
}
Using CelAnimationManager

private CelAnimationManager cam;

• Initialize()
cam = new CelAnimationManager(this, @"Content\Textures\");
Components.Add(cam);

• LoadContent()
cam.AddAnimation("enemy1", "MrEye", new CelCount(4, 2), 8);
cam.AddAnimation("enemy2", "MrEye", new CelCount(4, 2), 12);
cam.AddAnimation("enemy3", "MrEye", new CelCount(4, 2), 6);
cam.AddAnimation("complex1", "complex", new CelRange(1, 1, 2, 1), 64, 64, 2, 2);
cam.AddAnimation("complex2", "complex", new CelRange(3, 1, 1, 3), 64, 64, 7, 8);
cam.AddAnimation("complex3", "complex", new CelRange(2, 3, 1, 4), 64, 64, 4, 2);
cam.AddAnimation("complex4", "complex", new CelRange(2, 4, 4, 4), 64, 64, 3, 5);
Draw()

spriteBatch.Begin();
cam.Draw(gameTime, "enemy1", spriteBatch, new Vector2(50, 50));
cam.Draw(gameTime, "enemy2", spriteBatch, new Vector2(150, 75));
cam.Draw(gameTime, "enemy3", spriteBatch, new Vector2(70, 130));
cam.Draw(gameTime, "complex1", spriteBatch, new Vector2(400, 50));
cam.Draw(gameTime, "complex2", spriteBatch, new Vector2(400, 150));
cam.Draw(gameTime, "complex3", spriteBatch, new Vector2(400, 250));
cam.Draw(gameTime, "complex4", spriteBatch, new Vector2(400, 350));
spriteBatch.End();