Crystal Methods

Braun, Eric

Hart, Adam

Keller, Colton

Leatherman, Garett

Losinski, Tyler

Lab3 Audio

March 30, 2015

Animated Sprites class

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace AnimatedSprites

{

class AutomatedSprite: Sprite

{

// Sprite is automated. Direction is same as speed

public override Vector2 direction

{

get { return speed; }

}

//added collisioncues to the constructors

public AutomatedSprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed, string collisionCueName)

: base(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, collisionCueName)

{

}

public AutomatedSprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed,

int millisecondsPerFrame, string collisionCueName)

: base(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, millisecondsPerFrame, collisionCueName)

{

}

public override void Update(GameTime gameTime, Rectangle clientBounds)

{

// Move sprite based on direction

position += direction;

base.Update(gameTime, clientBounds);

}

}

}

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace AnimatedSprites

{

class BouncingSprite : AutomatedSprite

{

// Sprite is automated. Direction is same as speed

public override Vector2 direction

{

get { return speed; }

}

public BouncingSprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed, string collisionCueName)

: base(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed,collisionCueName)

{

}

public BouncingSprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed,

int millisecondsPerFrame, string collisionCueName)

: base(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, millisecondsPerFrame,collisionCueName)

{

}

public override void Update(GameTime gameTime, Rectangle clientBounds)

{

// Move sprite based on direction

position += direction;

//Check if it went off screen and reverses the direction if it goes off the screen

if (position.X < 0)

{

position.X = 0;

speed.X = speed.X \* -1;

}

if (position.Y < 0)

{

position.Y = 0;

speed.Y = speed.Y \* -1;

}

if (position.X > clientBounds.Width - frameSize.X)

{

position.X = clientBounds.Width - frameSize.X;

speed.X = speed.X \* -1;

//speed.Y = speed.Y \* -1;

}

if (position.Y > clientBounds.Height - frameSize.Y)

{

position.Y = clientBounds.Height - frameSize.Y;

//speed.X = speed.X \* -1;

speed.Y = speed.Y \* -1;

}

base.Update(gameTime, clientBounds);

}

}

}

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Audio;

using Microsoft.Xna.Framework.Content;

using Microsoft.Xna.Framework.GamerServices;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using Microsoft.Xna.Framework.Media;

using Microsoft.Xna.Framework.Net;

using Microsoft.Xna.Framework.Storage;

namespace AnimatedSprites

{

public class Game1 : Microsoft.Xna.Framework.Game

{

GraphicsDeviceManager graphics;

SpriteBatch spriteBatch;

SpriteManager spriteManager;

//Added sounds

SoundEffect soundEffect;

AudioEngine audioEngine;

WaveBank waveBank;

SoundBank soundBank;

Cue trackCue;

public Game1()

{

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

}

protected override void Initialize()

{

spriteManager = new SpriteManager(this);

Components.Add(spriteManager);

base.Initialize();

}

protected override void LoadContent()

{

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

//load/playing soundeffect code

//soundEffect = Content.Load<SoundEffect>(@"Audio\start");

//SoundEffectInstance soundEffectInstance = soundEffect.CreateInstance();

//soundEffectInstance.Play();

//initializing audioengine/soundbank

audioEngine = new AudioEngine(@"Content\Audio\GameAudio.xgs");

waveBank = new WaveBank(audioEngine, @"Content\Audio\Wave Bank.xwb");

soundBank = new SoundBank(audioEngine, @"Content\Audio\Sound Bank.xsb");

//starting looping track

trackCue = soundBank.GetCue("track");

trackCue.Play();

soundBank.PlayCue("start");

}

protected override void UnloadContent()

{

// TODO: Unload any non ContentManager content here

}

public void PlayCue(string cueName)

{

soundBank.PlayCue(cueName);

}

protected override void Update(GameTime gameTime)

{

// Allows the game to exit

if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==

ButtonState.Pressed)

this.Exit();

//updating sounds

audioEngine.Update();

base.Update(gameTime);

}

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.White);

base.Draw(gameTime);

}

}

}

//Cystal Methods

//Section 2

using System;

namespace AnimatedSprites

{

#if WINDOWS || XBOX

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

static void Main(string[] args)

{

using (Game1 game = new Game1())

{

game.Run();

}

}

}

#endif

}

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace AnimatedSprites

{

abstract class Sprite

{

// Stuff needed to draw the sprite

Texture2D textureImage;

protected Point frameSize;

Point currentFrame;

Point sheetSize;

//soundCue

public string CollisionCueName { get; private set; }

// Collision data

int collisionOffset;

// Framerate stuff

int timeSinceLastFrame = 0;

int millisecondsPerFrame;

const int defaultMillisecondsPerFrame = 16;

// Movement data

protected Vector2 speed;

protected Vector2 position;

// Abstract definition of direction property

public abstract Vector2 direction

{

get;

}

//added collisioncues to constructors

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed, string collisionCueName)

: this(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, defaultMillisecondsPerFrame, collisionCueName)

{

}

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed,

int millisecondsPerFrame, string collisionCueName)

{

this.textureImage = textureImage;

this.position = position;

this.frameSize = frameSize;

this.collisionOffset = collisionOffset;

this.currentFrame = currentFrame;

this.sheetSize = sheetSize;

this.speed = speed;

this.millisecondsPerFrame = millisecondsPerFrame;

this.CollisionCueName = collisionCueName;

}

public virtual void Update(GameTime gameTime, Rectangle clientBounds)

{

// Update frame if time to do so based on framerate

timeSinceLastFrame += gameTime.ElapsedGameTime.Milliseconds;

if (timeSinceLastFrame > millisecondsPerFrame)

{

// Increment to next frame

timeSinceLastFrame = 0;

++currentFrame.X;

if (currentFrame.X >= sheetSize.X)

{

currentFrame.X = 0;

++currentFrame.Y;

if (currentFrame.Y >= sheetSize.Y)

currentFrame.Y = 0;

}

}

}

public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)

{

// Draw the sprite

spriteBatch.Draw(textureImage,

position,

new Rectangle(currentFrame.X \* frameSize.X,

currentFrame.Y \* frameSize.Y,

frameSize.X, frameSize.Y),

Color.White, 0, Vector2.Zero,

1f, SpriteEffects.None, 0);

}

// Gets the collision rect based on position, framesize and collision offset

public Rectangle collisionRect

{

get

{

return new Rectangle(

(int)position.X + collisionOffset,

(int)position.Y + collisionOffset,

frameSize.X - (collisionOffset \* 2),

frameSize.Y - (collisionOffset \* 2));

}

}

}

}

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace AnimatedSprites

{

abstract class Sprite

{

// Stuff needed to draw the sprite

Texture2D textureImage;

protected Point frameSize;

Point currentFrame;

Point sheetSize;

//soundCue

public string CollisionCueName { get; private set; }

// Collision data

int collisionOffset;

// Framerate stuff

int timeSinceLastFrame = 0;

int millisecondsPerFrame;

const int defaultMillisecondsPerFrame = 16;

// Movement data

protected Vector2 speed;

protected Vector2 position;

// Abstract definition of direction property

public abstract Vector2 direction

{

get;

}

//added collisioncues to constructors

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed, string collisionCueName)

: this(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, defaultMillisecondsPerFrame, collisionCueName)

{

}

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed,

int millisecondsPerFrame, string collisionCueName)

{

this.textureImage = textureImage;

this.position = position;

this.frameSize = frameSize;

this.collisionOffset = collisionOffset;

this.currentFrame = currentFrame;

this.sheetSize = sheetSize;

this.speed = speed;

this.millisecondsPerFrame = millisecondsPerFrame;

this.CollisionCueName = collisionCueName;

}

public virtual void Update(GameTime gameTime, Rectangle clientBounds)

{

// Update frame if time to do so based on framerate

timeSinceLastFrame += gameTime.ElapsedGameTime.Milliseconds;

if (timeSinceLastFrame > millisecondsPerFrame)

{

// Increment to next frame

timeSinceLastFrame = 0;

++currentFrame.X;

if (currentFrame.X >= sheetSize.X)

{

currentFrame.X = 0;

++currentFrame.Y;

if (currentFrame.Y >= sheetSize.Y)

currentFrame.Y = 0;

}

}

}

public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)

{

// Draw the sprite

spriteBatch.Draw(textureImage,

position,

new Rectangle(currentFrame.X \* frameSize.X,

currentFrame.Y \* frameSize.Y,

frameSize.X, frameSize.Y),

Color.White, 0, Vector2.Zero,

1f, SpriteEffects.None, 0);

}

// Gets the collision rect based on position, framesize and collision offset

public Rectangle collisionRect

{

get

{

return new Rectangle(

(int)position.X + collisionOffset,

(int)position.Y + collisionOffset,

frameSize.X - (collisionOffset \* 2),

frameSize.Y - (collisionOffset \* 2));

}

}

}

}

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace AnimatedSprites

{

abstract class Sprite

{

// Stuff needed to draw the sprite

Texture2D textureImage;

protected Point frameSize;

Point currentFrame;

Point sheetSize;

//soundCue

public string CollisionCueName { get; private set; }

// Collision data

int collisionOffset;

// Framerate stuff

int timeSinceLastFrame = 0;

int millisecondsPerFrame;

const int defaultMillisecondsPerFrame = 16;

// Movement data

protected Vector2 speed;

protected Vector2 position;

// Abstract definition of direction property

public abstract Vector2 direction

{

get;

}

//added collisioncues to constructors

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed, string collisionCueName)

: this(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, defaultMillisecondsPerFrame, collisionCueName)

{

}

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed,

int millisecondsPerFrame, string collisionCueName)

{

this.textureImage = textureImage;

this.position = position;

this.frameSize = frameSize;

this.collisionOffset = collisionOffset;

this.currentFrame = currentFrame;

this.sheetSize = sheetSize;

this.speed = speed;

this.millisecondsPerFrame = millisecondsPerFrame;

this.CollisionCueName = collisionCueName;

}

public virtual void Update(GameTime gameTime, Rectangle clientBounds)

{

// Update frame if time to do so based on framerate

timeSinceLastFrame += gameTime.ElapsedGameTime.Milliseconds;

if (timeSinceLastFrame > millisecondsPerFrame)

{

// Increment to next frame

timeSinceLastFrame = 0;

++currentFrame.X;

if (currentFrame.X >= sheetSize.X)

{

currentFrame.X = 0;

++currentFrame.Y;

if (currentFrame.Y >= sheetSize.Y)

currentFrame.Y = 0;

}

}

}

public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)

{

// Draw the sprite

spriteBatch.Draw(textureImage,

position,

new Rectangle(currentFrame.X \* frameSize.X,

currentFrame.Y \* frameSize.Y,

frameSize.X, frameSize.Y),

Color.White, 0, Vector2.Zero,

1f, SpriteEffects.None, 0);

}

// Gets the collision rect based on position, framesize and collision offset

public Rectangle collisionRect

{

get

{

return new Rectangle(

(int)position.X + collisionOffset,

(int)position.Y + collisionOffset,

frameSize.X - (collisionOffset \* 2),

frameSize.Y - (collisionOffset \* 2));

}

}

}

}

Simple Sounds project

//Cystal Methods

//Section 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace AnimatedSprites

{

abstract class Sprite

{

// Stuff needed to draw the sprite

Texture2D textureImage;

protected Point frameSize;

Point currentFrame;

Point sheetSize;

//soundCue

public string CollisionCueName { get; private set; }

// Collision data

int collisionOffset;

// Framerate stuff

int timeSinceLastFrame = 0;

int millisecondsPerFrame;

const int defaultMillisecondsPerFrame = 16;

// Movement data

protected Vector2 speed;

protected Vector2 position;

// Abstract definition of direction property

public abstract Vector2 direction

{

get;

}

//added collisioncues to constructors

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed, string collisionCueName)

: this(textureImage, position, frameSize, collisionOffset, currentFrame,

sheetSize, speed, defaultMillisecondsPerFrame, collisionCueName)

{

}

public Sprite(Texture2D textureImage, Vector2 position, Point frameSize,

int collisionOffset, Point currentFrame, Point sheetSize, Vector2 speed,

int millisecondsPerFrame, string collisionCueName)

{

this.textureImage = textureImage;

this.position = position;

this.frameSize = frameSize;

this.collisionOffset = collisionOffset;

this.currentFrame = currentFrame;

this.sheetSize = sheetSize;

this.speed = speed;

this.millisecondsPerFrame = millisecondsPerFrame;

this.CollisionCueName = collisionCueName;

}

public virtual void Update(GameTime gameTime, Rectangle clientBounds)

{

// Update frame if time to do so based on framerate

timeSinceLastFrame += gameTime.ElapsedGameTime.Milliseconds;

if (timeSinceLastFrame > millisecondsPerFrame)

{

// Increment to next frame

timeSinceLastFrame = 0;

++currentFrame.X;

if (currentFrame.X >= sheetSize.X)

{

currentFrame.X = 0;

++currentFrame.Y;

if (currentFrame.Y >= sheetSize.Y)

currentFrame.Y = 0;

}

}

}

public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch)

{

// Draw the sprite

spriteBatch.Draw(textureImage,

position,

new Rectangle(currentFrame.X \* frameSize.X,

currentFrame.Y \* frameSize.Y,

frameSize.X, frameSize.Y),

Color.White, 0, Vector2.Zero,

1f, SpriteEffects.None, 0);

}

// Gets the collision rect based on position, framesize and collision offset

public Rectangle collisionRect

{

get

{

return new Rectangle(

(int)position.X + collisionOffset,

(int)position.Y + collisionOffset,

frameSize.X - (collisionOffset \* 2),

frameSize.Y - (collisionOffset \* 2));

}

}

}

}

//Crystal Methods

//Section 2

using System;

namespace simpleSounds

{

#if WINDOWS || XBOX

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

static void Main(string[] args)

{

using (Game1 game = new Game1())

{

game.Run();

}

}

}

#endif

}