Improving our 2D game

Slides adapted from 4-week course at Cornell by Tom Roeder
Interactive Game loop
Interactive Game Loop

Theoretically this is done at a constant refresh rate
Problems With our Game Loop

- Varying power for CPU and GPU in games causes games to run at different speeds on different hardware.
- Solution is to pick a target frame rate and go to sleep if running too fast.
  - What if hardware can’t do that rate?
  - Wasting electricity (and computer power).
- For us, we will use the XNA game loop.
- For further information read http://www.nuclex.org/articles/xna-game-loop-basics
Game Loop: Things to consider

- Do all tasks need the same granularity? That is do they need to be updated at the same rate?
  - Maybe we want our physics to be updated at 120Hz.
  - But player input, will the user notice if the delay is 10ms or 20ms or 30ms? Maybe we can run this at 60Hz.
  - What about AI for our monsters? Maybe they only need to be updated at 10Hz.
  - What about networking?
  - Does drawing at 100Hz make sense?
- If the CPU is your bottleneck, consider running tasks at different rates.
XNA Game Loop: Fixed step

- `Game.IsFixedTimeStep = true;`  // Default
- XNA calls `Update()` TargetElapsedTime’s every second (default 60)
- XNA logic
  - If Update+Draw time < 1/60 will
    - `Update()`
    - `Draw()`
    - Hang out for rest of time.
  - If Update+Draw > 1/60
    - Set `GameTime.IsRunningSlowly = true;
    - Keep calling Update (without Draw) until caught up
    - If gets to far behind might punt


If you notice `GameTime.IsRunningSlowly` is true, then you can do less work to help out.
Debugging

- When you are debugging, the timing will get off, so just because Update() keeps getting called, that doesn’t necessarily mean that you are running too slow or what is running too slow.
XNA Variable Game Loop

- `Game.IsFixedTimeStep = false;`
  - `Update()`
  - `Draw()`
  - `Repeat`

- You can getting elapsed time information to control your physics.
Scenes

- Every game will most likely have several different screens (called scenes in game lingo)
  - Title
  - Options
  - Help
  - Level 1
  - Level 2
  - Score Board
  - You Lose!
  - You Win!
Scenes Have

- Background image
- Background music
- Actors that act in the scene
  - Enemies
  - Boundaries
  - Player avatar
  - Bullets
  - Etc.
XNA

- Game scenes are a GameComponent
  - Use DrawableGameComponent for visual components
  - Add them with Components.Add()
  - Can remove them with Components.Remove()
    - If you have objects such as bullets, is it better to remove then add a new one, or just change the state of the bullet back to the gun?

- We are going to create some scenes
  - Startscene
  - Actionscene
  - Helpscene
Lets Create Our Scenes

- Example from the book: *Beginning XNA 2.0 Game Programming From Novice to Professional*, Alexndre Lobao, Bruno Evangelista, and Jose Antonio Leal de Faria, Apress, 2008
public abstract class GameScene : DrawableGameComponent {
    private readonly List<GameComponent> components;
    public List<GameComponent> Components {
        get { return components; } // Expose for adding to it
    }
    public GameScene(Game game) : base(game) {
        components = new List<GameComponent>();
        Visible = false; Enabled = false;
    }
    public virtual void Show() { // Shows scene
        Visible = true;
        Enabled = true;
    }
}
GameScene class

```csharp
public virtual void Hide() { // Hide scene
    Visible = false;
    Enabled = false;
}

public override void Update(GameTime gameTime) {
    for (int i = 0; i < components.Count; i++)
        if (components[i].Enabled)
            components[i].Update(gameTime);
    base.Update(gameTime);
}
```

Update only those game components that are currently Enabled. If this scene is not Enabled then XNA won’t call Update()
public override void Draw(GameTime gameTime) {
    for (int i = 0; i < components.Count; i++) {
        GameComponent gc = components[i];
        if ((gc is DrawableGameComponent) &&
            ((DrawableGameComponent) gc).Visible)
            ((DrawableGameComponent) gc).Draw(gameTime);
    }
    base.Draw(gameTime);
}
GameScene class

```csharp
public override void Draw(GameTime gameTime) {
    foreach (GameComponent gc in components) {
        if ((gc is DrawableGameComponent) &&
            ((DrawableGameComponent) gc).Visible)
            ((DrawableGameComponent) gc).Draw(gameTime);
    }
    base.Draw(gameTime);
}
} // End class
```

- GameScene class allows us to tell XNA when and when not to display scene.
- Calls Update() and Draw() for actors that need to update or draw
ImageComponent Class

// Draw a texture either centered or stretched
public class ImageComponent : DrawableGameComponent
{
    public enum DrawMode
    {
        Center = 1,
        Stretch,
    };

    protected readonly Texture2D texture;
    protected readonly DrawMode drawMode;
    protected SpriteBatch spriteBatch = null;
    protected Rectangle imageRect;
public ImageComponent(Game game, Texture2D texture, DrawMode drawMode) : base(game) {
    this.texture = texture;
    this.drawMode = drawMode;
    spriteBatch = (SpriteBatch) Game.Services.GetService(typeof(SpriteBatch));

    switch (drawMode) {
    case DrawMode.Center:
        break;
    case DrawMode.Stretch:
        imageRect = new Rectangle(0, 0, Game.Window.ClientBounds.Width, Game.Window.ClientBounds.Height);
        break;
    }
}
public override void Draw(GameTime gameTime)
{
    spriteBatch.Draw(texture, imageRect, Color.White);
    base.Draw(gameTime);
}

Need to draw ourselves. Do that with the SpriteBatch

Will this work on all displays? Widescreen, TV 4:3 vs 16:9
Let’s do the Help Scene

namespace RockRainEnhanced
{

public class HelpScene : GameScene
{
    public HelpScene(Game game, Texture2D textureBack, Texture2D textureFront): base(game)
    {
        Components.Add(new ImageComponent(game, textureFront, ImageComponent.DrawMode.Center));
    }
}
}
Declare Scene in Game class

protected HelpScene helpScene;
protected Texture2D helpBackgroundTexture,
    helpForegroundTexture;

// In LoadContent()
helpBackgroundTexture = 
    Content.Load<Texture2D>("helpbackground");
helpForegroundTexture = 
    Content.Load<Texture2D>("helpForeground");
helpScene = new HelpScene(this,
    helpBackgroundTexture, helpForegroundTexture);
Components.Add(helpScene);
public class StartScene : GameScene {
    protected TextMenuComponent menu; // Misc
    protected readonly Texture2D elements;
    protected AudioComponent audioComponent; // Audio
    protected Cue backMusic;
    protected SpriteBatch spriteBatch = null; // Spritebatch
    protected Rectangle rockRect = new Rectangle(0, 0, 536, 131);
    protected Vector2 rockPosition; // GUI
    protected Rectangle rainRect = new Rectangle(120, 165, 517, 130);
    protected Vector2 rainPosition;
    protected Rectangle enhancedRect = new Rectangle(8, 304, 375, 144);
    protected Vector2 enhancedPosition;
    protected bool showEnhanced;
    protected TimeSpan elapsedTime = TimeSpan.Zero;
Constructor

public StartScene(Game game, SpriteFont smallFont, SpriteFont largeFont, Texture2D background, Texture2D elements) : base(game) {
    this.elements = elements;
    Components.Add(new ImageComponent(game, background, ImageComponent.DrawMode.Center));
    string[] items = {"One Player", "Two Players", "Help", "Quit"};
    menu = new TextMenuComponent(game, smallFont, largeFont);// Menu
    menu.SetMenuItems(items);
    Components.Add(menu);

    spriteBatch = (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
    // Get the current audiocomponent and play the background music
    audioComponent = (AudioComponent)Game.Services.GetService(typeof(AudioComponent));
}
Show() method

public override void Show()   {
    audioComponent.PlayCue("newmeteor");
    backMusic = audioComponent.GetCue("startmusic");
    rockPosition.X = -1*rockRect.Width;
    rockPosition.Y = 40;
    rainPosition.X = Game.Window.ClientBounds.Width;
    rainPosition.Y = 180;
    // Center menu
    menu.Position = new Vector2((Game.Window.ClientBounds.Width-menu.Width)/2, 330);

    // These elements will be visible when the 'Rock Rain' title is done.
    menu.Visible = false;
    menu.Enabled = false;
    showEnhanced = false;

    base.Show();
}
public override void Hide()
{
    backMusic.Stop(AudioStopOptions.Immediate);
    base.Hide();
}

public int SelectedMenuIndex
{
    get { return menu.SelectedIndex; }
}
public override void Update(GameTime gameTime) {
    if (!menu.Visible) {
        if (rainPosition.X >= (Game.Window.ClientBounds.Width - 595)/2)
            rainPosition.X -= 15;
        if (rockPosition.X <= (Game.Window.ClientBounds.Width - 715)/2)
            rockPosition.X += 15;
        else {
            menu.Visible = true;
            menu.Enabled = true;
            backMusic.Play();
        }
    }
}
Update()

#if XBOX360
    enhancedPosition = new Vector2((rainPosition.X + 
                                     rainRect.Width - enhancedRect.Width / 2),
                                     rainPosition.Y);
#else
    enhancedPosition = 
        new Vector2((rainPosition.X + rainRect.Width - 
                       enhancedRect.Width/2) - 80, rainPosition.Y);
#endif

    showEnhanced = true;

}  // If Menu visible
else {
    elapsedTime += gameTime.ElapsedGameTime;

    if (elapsedTime > TimeSpan.FromSeconds(1)) {
        elapsedTime -= TimeSpan.FromSeconds(1);
        showEnhanced = !showEnhanced;
    }
}

base.Update(gameTime);
Draw()

/// <summary>
/// Allows the game component to draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
public override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
    spriteBatch.Draw(elements, rockPosition, rockRect, Color.White);
    spriteBatch.Draw(elements, rainPosition, rainRect, Color.White);
    if (showEnhanced)
    {
        spriteBatch.Draw(elements, enhancedPosition, enhancedRect, Color.White);
    }
}
Now What

- Define the font handling
- Develop an actionscene (similar to startscene, but it implements our game.
- Lets look at the game class to see how we move between scenes.
public class Game1 : Game {
    private readonly GraphicsDeviceManager graphics;
    private SpriteBatch spriteBatch;

    protected Texture2D helpBackgroundTexture, helpForegroundTexture;
    protected Texture2D startBackgroundTexture, startElementsTexture;
    protected Texture2D actionElementsTexture, actionBackgroundTexture;
    protected HelpScene helpScene;
    protected StartScene startScene;       // Scenes
    protected ActionScene actionScene;
    protected GameScene activeScene;
    private AudioComponent audioComponent; // Audio
    private SpriteFont smallFont, largeFont, scoreFont;   // Fonts
    protected KeyboardState oldKeyboardState; // input
    protected GamePadState oldGamePadState;
public Game1() {
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    oldKeyboardState = Keyboard.GetState();
    oldGamePadState = GamePad.GetState(PlayerIndex.One);   // input
#if XBOX360
    // On the 360 always fullscreen using user's prefered resolution
    graphics.PreferredBackBufferWidth = this.Window.ClientBounds.Width;
    graphics.PreferredBackBufferHeight = this.Window.ClientBounds.Height;
    // Get multisampling essentially for free on the 360, so turn it on
    graphics.PreferMultiSampling = true;
#endif
}
Game Class: Initialize()

protected override void Initialize()
{
  // Create the basics game objects
  audioComponent = new AudioComponent(this);
  Components.Add(audioComponent);
  Services.AddService(typeof (AudioComponent), audioComponent);
  base.Initialize();
}
Game Class: LoadContent()

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

    // Create the Credits / Instruction Scene
    helpBackgroundTexture = Content.Load<Texture2D>("helpbackground");
    helpForegroundTexture = Content.Load<Texture2D>("helpforeground");
    helpScene = new HelpScene(this, helpBackgroundTexture, helpForegroundTexture);
    Components.Add(helpScene);
}
Game Class: LoadContent()

// Create the Start Scene
smallFont = Content.Load<SpriteFont>("menuSmall");
largeFont = Content.Load<SpriteFont>("menuLarge");
startBackgroundTexture = Content.Load<Texture2D>("startbackground");
startElementsTexture = Content.Load<Texture2D>("startSceneElements");
startScene = new StartScene(this, smallFont, largeFont,
startBackgroundTexture, startElementsTexture);
Components.Add(startScene);

// Create the Action Scene
actionElementsTexture = Content.Load<Texture2D>("rockrainenhanced");
actionBackgroundTexture = Content.Load<Texture2D>("SpaceBackground");
scoreFont = Content.Load<SpriteFont>("score");
actionScene = new ActionScene(this, actionElementsTexture,
actionBackgroundTexture, scoreFont);
Components.Add(actionScene);

startScene.Show(); // Start game in the start Scene
activeScene = startScene;
Game Class: ShowScene()

/// <summary>
/// Open a new scene
/// </summary>
/// <param name="scene">Scene to be opened</param>
protected void ShowScene(GameScene scene)
{
    activeScene.Hide();
    activeScene = scene;
    scene.Show();
}
Game Class: Update()

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
  // Handle Game Inputs
  HandleScenesInput();

  base.Update(gameTime);
}
Game Class: HandleScenesInput()

/// Handle input of all game scenes
private void HandleScenesInput() {
    if (activeScene == startScene)
        HandleStartSceneInput();
    else if (activeScene == helpScene)
        if (CheckEnterA())
            ShowScene(startScene);
    else if (activeScene == actionScene)
        HandleActionInput();
}

private bool CheckEnterA()
{
    // Get the Keyboard and GamePad state
    GamePadState gamepadState = GamePad.GetState(PlayerIndex.One);
    KeyboardState keyboardState = Keyboard.GetState();

    bool result = (oldKeyboardState.IsKeyDown(Keys.Enter) &&
                   (keyboardState.IsKeyUp(Keys.Enter)));
    result |= (oldGamePadState.Buttons.A == ButtonState.Pressed) &&
              (gamepadState.Buttons.A == ButtonState.Released);

    oldKeyboardState = keyboardState;
    oldGamePadState = gamepadState;

    return result;
}
private void HandleActionInput()
{
    // Get the Keyboard and GamePad state
    GamePadState gamepadState = GamePad.GetState(PlayerIndex.One);
    KeyboardState keyboardState = Keyboard.GetState();

    bool backKey = (oldKeyboardState.IsKeyDown(Keys.Escape) &&
                    (keyboardState.IsKeyUp(Keys.Escape)));
    backKey |= (oldGamePadState.Buttons.Back == ButtonState.Pressed) &&
               (gamepadState.Buttons.Back == ButtonState.Released);

    bool enterKey = (oldKeyboardState.IsKeyDown(Keys.Enter) &&
                     (keyboardState.IsKeyUp(Keys.Enter)));
    enterKey |= (oldGamePadState.Buttons.A == ButtonState.Pressed) &&
                (gamepadState.Buttons.A == ButtonState.Released);

    oldKeyboardState = keyboardState;
    oldGamePadState = gamepadState;
}

Verify the key was pressed in this window
if (enterKey)
    if (actionScene.GameOver)
        ShowScene(startScene);
else
    {
        audioComponent.PlayCue("menu_back");
        actionScene.Paused = !actionScene.Paused;
    }
if (backKey)
    ShowScene(startScene);
} // End HandleActionInput
private void HandleStartSceneInput() {
    if (CheckEnterA()) {
        audioComponent.PlayCue("menu_select3");
        switch (startScene.SelectedMenuIndex) {
            case 0: actionScene.TwoPlayers = false;
                ShowScene(actionScene);
                break;
            case 1: actionScene.TwoPlayers = true;
                ShowScene(actionScene);
                break;
            case 2: ShowScene(helpScene);
                break;
            case 3: Exit(); break;
        }
    }
}
Game Class: Draw()

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// 
protected override void Draw(GameTime gameTime)
{
    // Begin..
    spriteBatch.Begin();

    // Draw all Game Components..
    base.Draw(gameTime);

    // End.
    spriteBatch.End();
}

Could have used Services.GetService to get the spriteBatch service that was added in LoadContent()
Pausing the game

```csharp
if (!pause &&
    keyboard.IsKeyDown(Keys.Tab))
    pause = true;
else if (pause &&
    keyboard.IsKeyDown(Keys.LeftControl))
    pause = false;
If (pause == false){
    DoGameLogic();
    base.Update(gameTime);
}
```
Collision Detection

- Can do an intersect() from
  - Rectangle
  - BoundingSphere
  - BoundingBox
  - BoundingFrustum

- If you are doing pixel level collision with sprites, first test for collision of bounding objects, then do pixel by pixel comparison. One bounding object may not be the most efficient
Game Over

- Remove all Components
- Replace background with game over scene
- Pause the music