Early Game Programming

Admin

- Questions from last time?
-

Game Programming

- Reminder from Day 1
  - There are popular engines that will hide some of the details we'll look at here from you
    - Unity/Unreal/Godot/etc
  - Those are great but…..
    - Just like you should see how data structures and searches/sorts work and implement them in Data Structures
      - But once you get into the working world you would never implement them yourself
      - You would use the versions in the standard library for the language you use
    - By the same token I want you to understand the underlying concepts for game (especially 2D games)
    - Even if you end up using these frameworks to hide some of that later.
  - Also - everything gets replaced

Movement

- How might we create movement
  - Or the illusion of movement
- For the player in our 2D games?

Movement

- How might we create movement
  - Or the illusion of movement
- For the player in our 2D games?
  - Move the player image in the window
    - Like we did last time
  - Move a background in the window
    - And have the player image on top of it.

Scroller Games

- Scroller games fairly straightforward to implement
  - side/top scroller same principle
  - create illusion of movement and continuity in direction of scrolling (side to side or top to bottom) by moving background.
  - allow player sprite to move in other direction.
- firing varies.

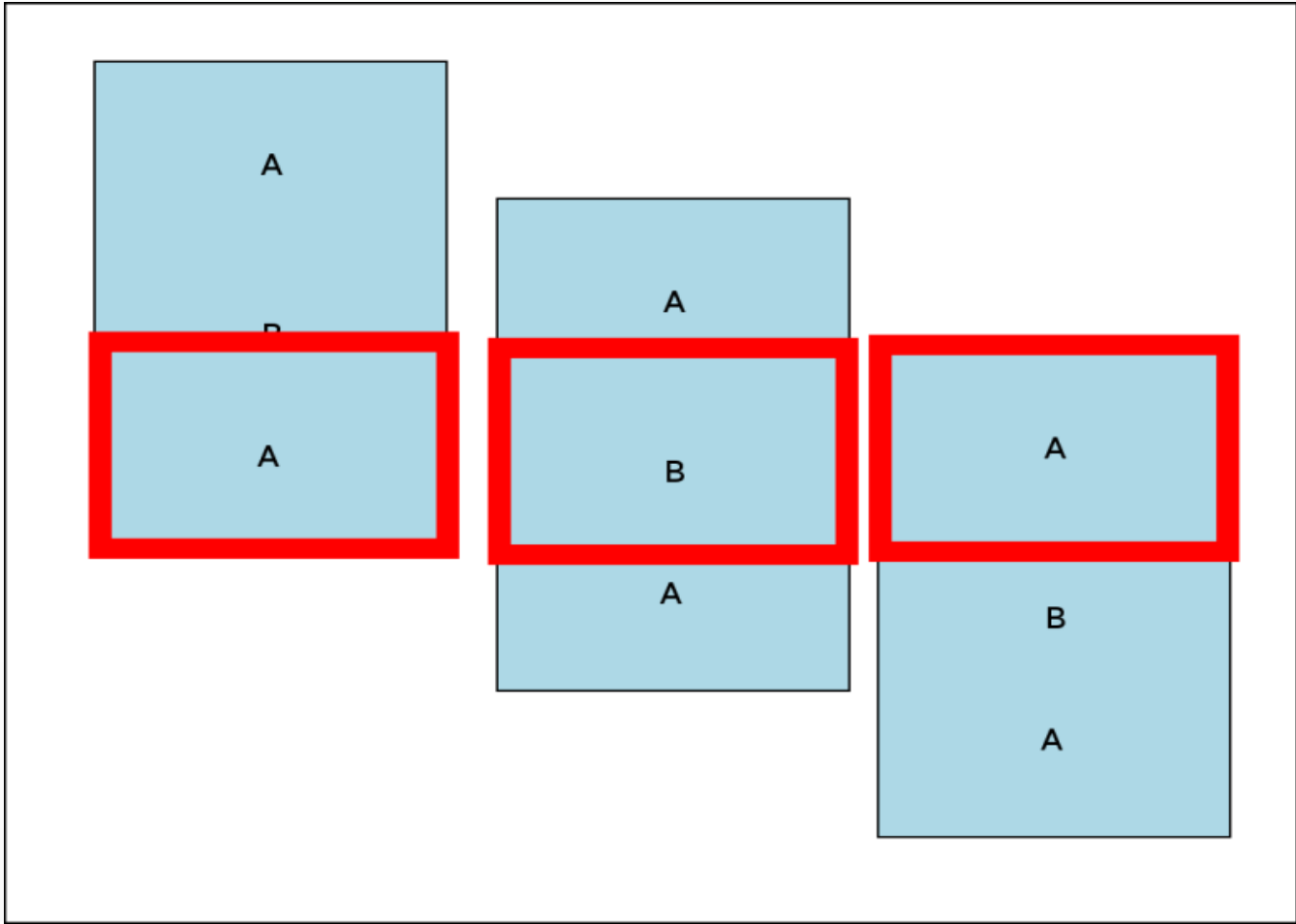
Two Scroller Techniques

- There are two easy scrolling background techniques
  - In both cases the background is a sprite
  - It is drawn before any of the foreground sprites.
- First use one big image
- Second use two identical images.

Scrolling Background I

- Use one large image (specially crafted)
  - image is three times as large as the screen/window
  - Beginning and end thirds of the image are identical.
  - Move the image across the window
  - When the image is about to show window pixels, move it back to start position.

- Image credit:
- Game Programming
  - By Harris
- Published by Wiley



A second approach

- Another approach that works with nearly any image:
  - Another approach to scrolling background
    - have two background instances and show them one after another
    - no need to have identical parts of the image any more
    - just need beginning and end of image to match
  - sample:

Lets try

- We will combine the two
  - We'll use a background image repeated three times
  - Draw on board if/as needed
- This will be our background image
  - Note the green bits at the bottom
  - And the while clouds
  - Are all about the same height on left and right side of image.
    - We'll see the clouds look a little funky

Lets look at the code

```
package main

import (
	"fmt"
	"github.com/hajimehoshi/ebiten/v2"
	"github.com/hajimehoshi/ebiten/v2/ebitenutil"
	_ "image/png"
)

type scrollDemo struct {
	player          *ebiten.Image
	background      *ebiten.Image
	backgroundXView int
}
```

- First imports and our game struct
- Mostly similar to last time
  - We won't use the player in the first pass.
- Any questions?

The Main function

```
func main() {
	ebiten.SetWindowSize(1000, 1000)
	ebiten.SetWindowTitle("Scroller Example")
	//New image from file returns image as image.Image (_) and ebiten.Image
	backgroundPict, _, err :=
ebitenutil.NewImageFromFile("background.png")
	if err != nil {
		fmt.Println("Unable to load background image:", err)
	}

	demo := scrollDemo{
		player:     nil,
		background: backgroundPict,
	}
	err = ebiten.RunGame(&demo)
	if err != nil {
		fmt.Println("Failed to run game", err)
	}
}
```

- Our main is also fairly similar to last time,
- but let's look at it.
- Then any questions?

Update and Layout

```
func (demo *scrollDemo) Update() error {
        backgroundWidth := demo.background.Bounds().Dx()
        maxX := backgroundWidth * 2
        demo.backgroundXView -= 4
        demo.backgroundXView %= maxX
        return nil
}


func (s scrollDemo) Layout(outsideWidth, outsideHeight int)
(screenWidth, screenHeight int) {
        return outsideWidth, outsideHeight
}
```

- Layout same as before
- Update
  - The max we want to scroll is 2 times the size
    - (that will leave one copy on the screen)
  - Move the image 4 pixels left
  - If we have moved more than 2 copies of the background over, then move it back to the beginning

Draw

```
func (demo *scrollDemo) Draw(screen *ebiten.Image) {
	drawOps := ebiten.DrawImageOptions{}
	const repeat = 3
	backgroundWidth := demo.background.Bounds().Dx()
	for count := 0; count < repeat; count += 1 {
		drawOps.GeoM.Reset()
		drawOps.GeoM.Translate(float64(backgroundWidth*count),
			float64(-1000))
		drawOps.GeoM.Translate(float64(demo.backgroundXView), 0)
		screen.DrawImage(demo.background, &drawOps)
	}
}
```

- Draw the background 3 times
  - Move it off the top of the screen (image is 2k pixels tall)
  - Move it horizontally first by its position in the three image roll
  - Then by the amount calculated in update

Lets take a look

- Let's put it all together and run it.
- https://github.com/jsantore/BareBonesScroll
- I’ve updated it from the slides for full screen.

Input

- It is all well and good to move an image on the screen
  - And it is needed for games
- But without user input, it isn't really a game.
- So lets get some input
  - Start with traditional laptop/desktop rather than controllers and touch
- So how will we get input?

Input

- It is all well and good to move an image on the screen
  - And it is needed for games
- But without user input, it isn't really a game.
- So lets get some input
  - Start with traditional laptop/desktop rather than controllers and touch
- So how will we get input?
  - Mouse and keyboard first, let's start with mouse

Ebitengine InputUtil Module

- The functions you want are in the InputUtil package
- Mouse
  - func IsMouseButtonJustPressed(button ebiten.MouseButton) bool
    - returns a boolean value indicating whether the given mouse button is pressed just in the current tick.
    - IsMouseButtonJustPressed must be called in a game's Update, not Draw.
  - func IsMouseButtonJustReleased(button ebiten.MouseButton) bool
    - IsMouseButtonJustReleased returns a boolean value indicating whether the given mouse button is released just in the current tick.
    - IsMouseButtonJustReleased must be called in a game's Update, not Draw.
  - func MouseButtonPressDuration(button ebiten.MouseButton) int
    - MouseButtonPressDuration returns how long the mouse button is pressed in ticks (Update).
    - MouseButtonPressDuration must be called in a game's Update, not Draw.

Check Mouse Click

- The checkIfTargetClicked is called from Update
- The CursorPosition is in window coordinates

```
type Target struct {
	pict  *ebiten.Image
	dx    int
	dy    int
	x     int
	y     int
	count int
}
```

```
func checkIfTargetClicked(target Target) bool {
	if inpututil.IsMouseButtonJustPressed(ebiten.MouseButton0) {
		mouseX, mouseY := ebiten.CursorPosition()
		goalWidth := target.pict.Bounds().Dx()
		goalHeight := target.pict.Bounds().Dy()
		if mouseX > target.x && mouseX < target.x+goalWidth &&
			mouseY < target.y+goalHeight && mouseY > target.y
{
			return true
		}
	}
	return false
}
```

Keyboard Input

- Keyboard input is in two different modules

ebiten.IsKeyPressed(<key>)

  - Returns true if <key> is pressed, false if it is not pressed

inpututil.IsKeyJustPressed(key)

  - Returns true if <key> was pressed in this update cycle.

- Both must be called from update, not draw

- For example

```
func getPlayerInput(game *scrollerGame) {
        if ebiten.IsKeyPressed(ebiten.KeyArrowUp) &&
game.Player.yLoc > 0 {
                game.Player.yLoc -= 3
        } else if ebiten.IsKeyPressed(ebiten.KeyArrowDown) &&
                game.Player.yLoc < WINDOW_HEIGHT-
game.Player.pict.Bounds().Dy() {
                game.Player.yLoc += 3
        }
        if inpututil.IsKeyJustPressed(ebiten.KeySpace) {
                firePlayerShot(game)
        }
}
```

Sounds

- While we **could** have a game with just images,
  - Sound is really vital
  - Ebitengine has methods for playing sounds
  - At first let's look at the simplest possible setup
    - Playing a wav file that is in the same folder as the main project.
  - Full demo
    - https://github.com/jsantore/SimpleEbitenSound
  - But we'll look at the vital/new pieces in the following slides.

Sounds II

- In Ebitengine
  - Playing sounds requires both
    - an audio.Context
  - And
    - And audio.Player
  - All of the audio players can share a context if they need/want to.
  - You need one player per sound.

- Imports and 'game' struct

```
import (
	"fmt"
	"github.com/hajimehoshi/ebiten/v2"
	"github.com/hajimehoshi/ebiten/v2/audio"
	"github.com/hajimehoshi/ebiten/v2/audio/wav"
	"github.com/hajimehoshi/ebiten/v2/inpututil"
	"golang.org/x/image/colornames"
	"os"
)

type soundDemo struct {
	audioContext *audio.Context
	soundPlayer  *audio.Player
	counter      int
}
```

Sounds III

- Create the context and sound player for the struct in main

```
func main() {
	soundContext := audio.NewContext(SOUND_SAMPLE_RATE)
	soundGame := soundDemo{
		audioContext: soundContext,
		soundPlayer:  LoadWav("Thunder1.wav", soundContext),
		counter:      20,
	}
	ebiten.SetWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT)
	ebiten.SetWindowTitle("Demo Simple Soundr")
	err := ebiten.RunGame(&soundGame)
	if err != nil {

	}

}
```

- Load the sound file

```
func LoadWav(name string, context *audio.Context) *audio.Player {
	thunderFile, err := os.Open(name)
	if err != nil {
		fmt.Println("Error Loading sound: ", err)
	}
	thunderSound, err :=
wav.DecodeWithoutResampling(thunderFile)
	if err != nil {
		fmt.Println("Error interpreting sound file: ", err)
	}
	soundPlayer, err := context.NewPlayer(thunderSound)
	if err != nil {
		fmt.Println("Couldn't create sound player: ", err)
	}
	return soundPlayer
}
```

Using the sounds

- Let's use the sound now

```
func (demo *soundDemo) Update() error {
	demo.counter += 1
	if inpututil.IsKeyJustPressed(ebiten.KeySpace) {
		demo.soundPlayer.Rewind()
		demo.soundPlayer.Play()
		demo.counter = 0
	}
	return nil
}
func (s soundDemo) Draw(screen *ebiten.Image) {
	if s.counter >= 20 {
		screen.Fill(colornames.Crimson)
	} else {
		screen.Fill(colornames.Deepskyblue)
	}
}
```

- Let's go over it
- Ask any questions
- We can take a quick look at it running
- https://github.com/jsantore/SimpleEbitenSound
-

Other Sound Options

- Example just now is sound built into ebitengine, but for more control there are other good options
  - https://github.com/ebitengine/oto
    - Newer player associated with (but not part of) ebitengine
  - https://github.com/SolarLune/resound
    - Give you more control over things like reverb and sound delays

Collisions

- Collisions are vital for games
  - Basically check to see if two images overlap each other.
  - We will start with a very simple approach,
    - Does any part of the first image overlap any part of the second image
    - We will use resolv library this semester "github.com/solarlune/resolv"
    - Library supports two approches, we'll use simple one for this first pass
  - Use intersection method on shape passing another shape
    - Return value is a resolv.IntersectionSet object
    - Could be empty
    - Could contain the sub polygon of the intersection.
  -

Using this collision Library

- The two structs
  - Let's look at this and understand it.

```
type Enemy struct {
        pict          *ebiten.Image
        collisionRect *resolv.ConvexPolygon
        deltaX        int
}

type PlayerData struct {
        pict          *ebiten.Image
        collisionRect *resolv.ConvexPolygon
}
```

```
func checkPlayerCollision(game *scrollerGame) {
        for _, baddie := range game.Enemies {
                if hit :=
game.Player.collisionRect.Intersection(

baddie.collisionRect); !hit.IsEmpty() {
                        game.state = endState
                }
        }
}
```

Drawing Text

- Ebitengine has a text v2 package for drawing text
  - Be sure to use it and not the old v1 package that many AIs are generating.
- Lets talk fonts and faces
  - What is what?

Drawing Text

- Ebitengine has a text v2 package for drawing text
  - Be sure to use it and not the old v1 package that many AIs are generating.
- Lets talk fonts and faces
  - What is what?
- There are two font faces shipped in the go standard library
  - Both are fairly small – lots of tutorials out there referencing them.
- We'll look at using a ttf to create a face.
  - Can find the demo at https://github.com/jsantore/MinimalEbitenFont

To draw text – first load font

- Lets look at loading a font and creating a face of the correct font size
  - Notice io.ReadAll instead of depricated ioutil version
  - Let's walk through it.

```
func LoadFont(fontFile string, size float64) font.Face {
	fileHandle, err := os.Open(fontFile)
	if err != nil {
		log.Fatal(err)
	}
	fontData, err := io.ReadAll(fileHandle)
	if err != nil {
		log.Fatal(err)
	}
	ttFont, err := opentype.Parse(fontData)
	if err != nil {
		log.Fatal(err)
	}
	fontFace, err := opentype.NewFace(ttFont,
&opentype.FaceOptions{
		Size:    size,
		DPI:     72,
		Hinting: font.HintingFull,
	})
	return fontFace
}
```

The ‘game’ struct and two of the methods

- The game struct just has
  - The text
  - The font to draw the text

```
type textDemo struct {
        text string
        font font.Face
}

func (demo textDemo) Update() error {
        return nil
}


func (demo textDemo) Layout(outsideWidth,
outsideHeight int)

(screenWidth, screenHeight int) {
        return outsideWidth, outsideHeight
}
```

main

- Here is the main
  - I think it is pretty straightforward after that last couple of weeks
  - But I've been doing this longer than you – so ask questions if you have them.

```
func main() {
	ebiten.SetWindowSize(1000, 1000)
	ebiten.SetWindowTitle("Text Display Demo")
	textFont := LoadFont("Square-Black.ttf", 100)
	demo := textDemo{
		text: "Hello World",
		font: textFont,
	}
	err := ebiten.RunGame(&demo)
	if err != nil {
		log.Fatal(err)
	}
}
```

Finally: Draw

- Draw is where the interest is
  - Ebitengine text vs requires
    - GoXface
  - text.DrawOptions
    - Combines standard draw and layout (which contains the ColorScale that we see here)
- Finally text.Draw needs
  - An ebiten image
  - A string
  - A goXFace
  - The text.DrawOptions

```
func (demo textDemo) Draw(screen *ebiten.Image) {
        drawFace := text.NewGoXFace(demo.font)
        textOpts := &text.DrawOptions{
                DrawImageOptions: ebiten.DrawImageOptions{},
                LayoutOptions:    text.LayoutOptions{},
        }
        textOpts.GeoM.Reset()
        textOpts.GeoM.Translate(350, 450)
        textOpts.ColorScale.ScaleWithColor(colornames.Red)
        text.Draw(screen, demo.text, drawFace, textOpts)
}
```

Now lets try

- Let’s put it all together
  - Grab the BareBonesScroll from much earlier
  - Let’s adjust it to so it puts up some start text until the user hits the space bar
  - Then it shows the scrolling background

There is the basics

- We have now seen all of the very basics that we need to build a simple game (or at least a proto-game).
- Any questions?
- Let's look at our first game programming project