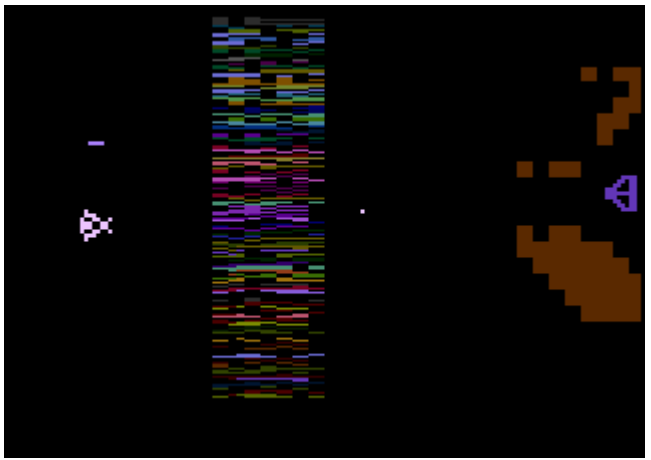
Camera and Game GUIs

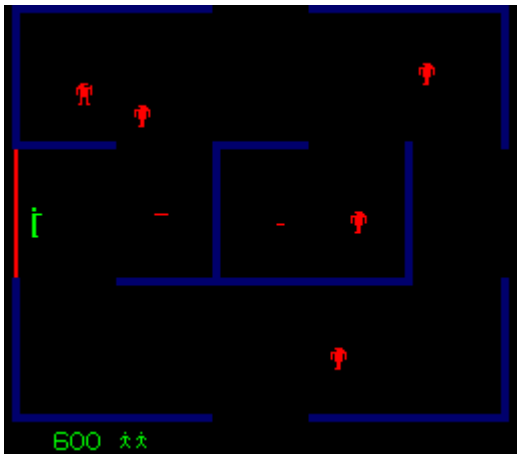
Admin

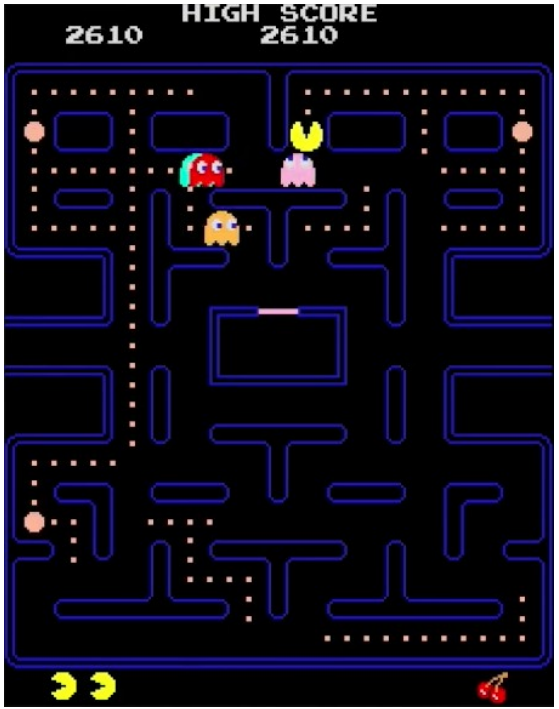
- Upcoming scheduling
  - The Nov 11 bridgewater twostep
  - The final exam
- Project questions?

Seeing part of the game ‘world’

- In early arcade video games we could see the whole ‘world’
  - Then player could see everything all at once

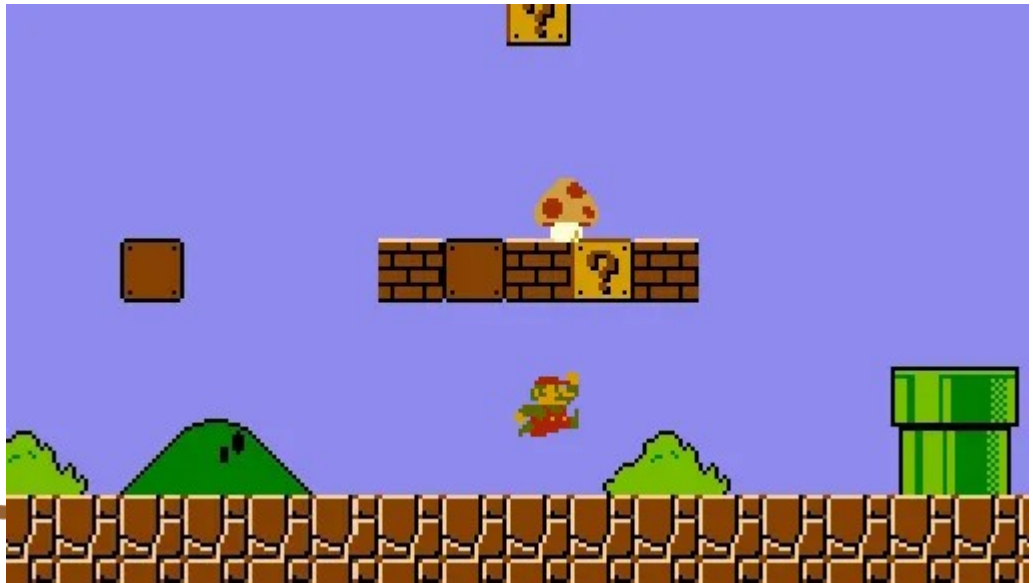






Seeing part of the game ‘world’

- In early arcade video games we could see the whole ‘world’
  - Then player could see everything all at once
  - Next we had the scrolling background to give the impression of a larger world
  -



Camera

- Eventually most games settled on the notion of a ‘camera’ which would show just the relevant part of the world on screen.
  - For 2d games
    - We might render the entire scene
    - Then only draw part of it to the screen in draw
    - Either using something like sub-image, or more likely just translating the large image so we see the relevant part.
  - Built in to all of the engines
  - We will use a library to help us out.

Camera Libarary

- https://github.com/tducasse/ebiten-camera
- Like most of the ebiten and ebiten adjacent projects
  - Simple, extensible.
  - Official readme ‘tutorial’

```
import camera "github.com/tducasse/ebiten-camera"


myCamera := camera.Init(screenWidth, screenHeight)
world := ebiten.NewImage(worldWidth, worldHeight)

ebiten.SetWindowSize(screenWidth*windowScale,
screenHeight*windowScale)
// in your Draw call
world.Clear()
// your draw calls go here, but target world instead of screen
player.Draw(world)
// then draw the world onto the screen
myCamera.Draw(world, screen)
```

My Comments

- Annotated
  - You want variables in your game for the camera and ‘world’ that you will init in main
  - Init your camera with window size
  - Then the rest is done in draw
  - Let’s look as a slightly more complete demo
  - https://github.com/jsantore/SimpleCamera

```
import camera "github.com/tducasse/ebiten-camera"


myCamera := camera.Init(screenWidth, screenHeight)
world := ebiten.NewImage(worldWidth, worldHeight)
ebiten.SetWindowSize(screenWidth*windowScale,
screenHeight*windowScale)

// in your Draw call
world.Clear()
// your draw calls go here, but target world
instead of screen
player.Draw(world)
// then draw the world onto the screen
myCamera.Draw(world, screen)
```

Example

```
//go:embed assets
var EmbeddedAssets embed.FS

type cameraDemoGame struct {
	background    *ebiten.Image //the background image on disk
	displayedLevel *ebiten.Image //world image: background + player
	cameraView    *camera.Camera
	player        player
	drawOps       ebiten.DrawImageOptions
}

type player struct {
	pict *ebiten.Image
	x, y int
}
```

- Pretty standard update with bounds

```
func (demo *cameraDemoGame) Update() error {
	if ebiten.IsKeyPressed(ebiten.KeyLeft) && demo.player.x > 100
{
		demo.player.x -= 5
	} else if ebiten.IsKeyPressed(ebiten.KeyRight) &&
demo.player.x < 1800 {
		demo.player.x += 5
		log.Println("x is now ", demo.player.x)
	}
	if ebiten.IsKeyPressed(ebiten.KeyUp) && demo.player.y > 100 {
		demo.player.y -= 5
	} else if ebiten.IsKeyPressed(ebiten.KeyDown) &&
demo.player.y < 900 {
		demo.player.y += 5
	}
	return nil
}
```

Draw is where the difference is

- Draw to world, set follow, then have camera draw world sub image to screen

```
func (demo *cameraDemoGame) Draw(screen *ebiten.Image) {
	//draw to the world at first
	//first draw background
	demo.drawOps.GeoM.Reset()
	demo.displayedLevel.DrawImage(demo.background, &demo.drawOps)
	//next draw player
	demo.drawOps.GeoM.Reset()
	demo.drawOps.GeoM.Translate(float64(demo.player.x), float64(demo.player.y))
	demo.displayedLevel.DrawImage(demo.player.pict, &demo.drawOps)

	//now move the camera to be over the player
	demo.cameraView.Follow.H = demo.player.y * 2
	demo.cameraView.Follow.W = demo.player.x * 2
	//finally draw to the screen
	demo.cameraView.Draw(demo.displayedLevel, screen)
}
```

Let’s see it run

- Let’s download this and run it.

Interfaces

- When you have interface components,
  - Use the camera for the game as seen earlier,
  - Then draw the interface on the screen itself, after the camera has adjusted
  - For this first pass, we will just draw a bit of text on the screen as the camera moves around
    - This is the simple camera demo with text added
    - https://github.com/jsantore/CameraWithInterface
  -

Changes from Simple Camera

- Added a LoadFontEmbedded
  - Same as the old load font from the font demo,
  - but uses EmbbeddedFS.Open instead of os.Open
- Added to game struct

```
textOps       text.DrawOptions
drawFont      *text.GoXFace
```

- In main

```
fontFace := LoadFontEmbedded("Square-Black.ttf", 18)
drawFace := text.NewGoXFace(fontFace)
```

- Then used drawFace to initialize the drawfont in the game struct

- Draw added several new lines at the end
- Now lets see it work

```
//Newly adding a first pass at an interface
demo.textOps.GeoM.Reset()
demo.textOps.GeoM.Translate(50, 50) //let's start the text in the upper left of the window
demo.textOps.ColorScale.Reset()
demo.textOps.ColorScale.ScaleWithColor(colornames.Wheat)
info := fmt.Sprintf("Player is at %d, %d", demo.player.x, demo.player.y)
text.Draw(screen, info, demo.drawFont, &demo.textOps)
```

GUI

- What is a gui?
  - The lucky volunteer program keeps on trucking.
  -

GUI

- What is a gui?
  - The lucky volunteer program keeps on trucking.
  - Potential Answer:
    - A set of controls for allowing the user to more easily interact with the program

GUI

- What is a gui from the point of view of the application developer?
  - lucky volunteer?
  -

GUI

- What is a gui from the point of view of the application developer?
  - A set of components that respond to events, and you provide those components your functions to call when the events happen
  - Or something like this right?

GUI

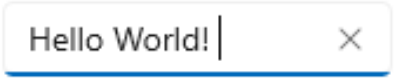
- What is a gui from the point of view of the GUI library developer?
  - Lucky Volunteer?
  - This one is harder – since most of you were the users of gui libraries

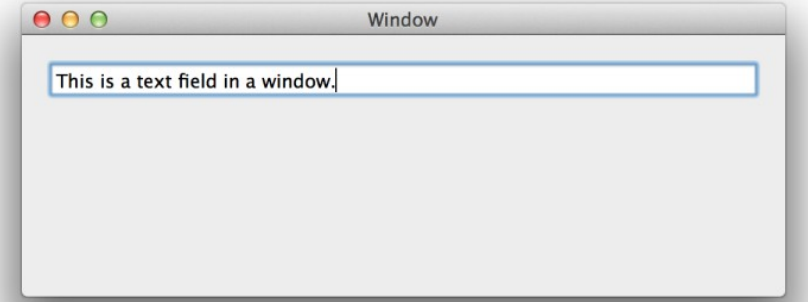
GUI

- What is a gui from the point of view of the GUI library developer?
  - A library that draws an image at a particular point in the screen/window
  - When the component represented by the image gains ‘focus’ then it will ask the operating system to send all events to it.
  - Then the component will call your handler functions when the right events from the OS arrive.

GUIs

- Windowed operating systems all provide a set of images for their GUI controls (system native) eg Windows 11 (left) and MacOS (right)
  - Image credit to Microsoft and Apple developer docs





- Other libraries provide their own images
- But system native libraries are often desired so that users can use hard won knowledge of how a program ‘should’ work.

GUIs for Games

- Often Games want a unique look
  - Don’t want our game to look like the spreadsheet we use for work.
  - So often want our programs to use custom images for components.
  - So we might use text entry and two sets of images for a checkbox, in the checked and unchecked states.
  - 
  - 
  -
  - Depending on our game 

GUIs for games

- So how do we get these unique GUIs?
  - Lucky volunteer?

GUIs for games

- So how do we get these unique GUIs?
  - We could build our own engine with custom GUI components
  - Or?

GUIs for games

- So how do we get these unique GUIs?
  - We could build our own engine with custom GUI components
  - Or we could use an existing engine/library which will take custom images and use them.
    - Godot and Unity both have gui components that will take a custom image.
    - For ebitengine there is a related library ebitenui
    - https://github.com/ebitenui/ebitenui
    - This library has seen a major overhaul and a lot of improvements in the first half of the year. (if you find my old stuff on ebitenui – it will now all be wrong/outdated).
    - Docs at https://ebitenui.github.io/ for more
    - Ebitenui comes with a default set of images, but allows you to use your own.

Arraigning GUIs

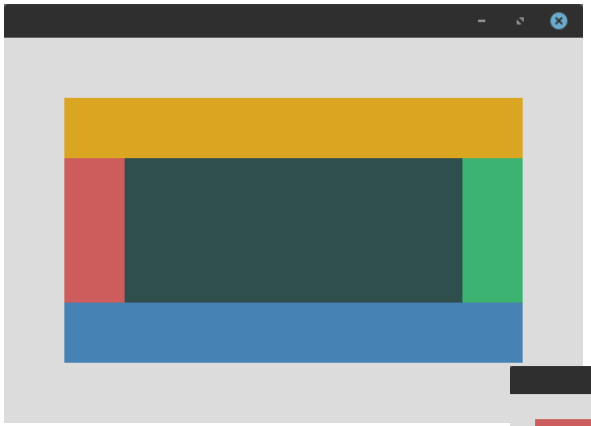
- Game guis work a lot more like old desktop UIs than modern reactive/mobile Uis
  - Usually don’t change game resolution or arraignment
  - You all had to take comp152 to get here
    - And Comp152 has GUI programming as a mandatory outcome
    - So how are (desktop first) GUIs usually arraigned?
      - How do you specify where everything goes?

Arraigning GUIs

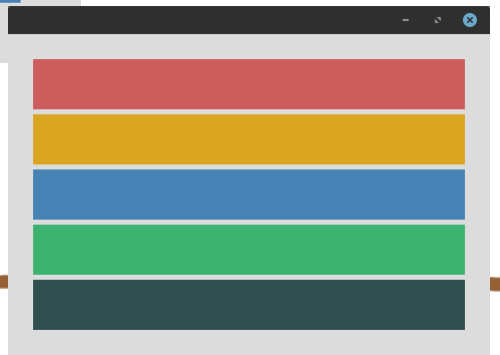
- Game guis work a lot more like old desktop UIs than modern reactive/mobile Uis
  - Usually don’t change game resolution or arraignment
  - You all had to take comp152 to get here
    - And Comp152 has GUI programming as a mandatory outcome
    - So how are (desktop first) GUIs usually arraigned?
      - How do you specify where everything goes?
    - Usually there are some sort of Container with a Layout object that determines this.
    - Give me some examples of Layouts you have run into?
      - Lets put them on the board.

Arraigning GUIs

- EbitenUI supports subset of these layouts.
  - AnchorLayout



  - RowLayout
    - Vertical
    - Or horzontal



- GridLayout

- StackedLayout
  - Not yet well documented
- Images all from official docs

Arraigning GUIs

- To make arbitrary Uis
  - Put a container with one Layout inside of another container with another layout.
  - Putting a rowLayout inside of the center pane of an anchor layout for example

Control over look, feel, behavior

- In most GUIs
  - You *can* have a great deal of control over things like fonts and colors
  - But there are defaults that are automatically selected to make it look like every other gui on that OS
- For games
  - We want it to look unique
  - So ebitenui tends to make you set some values yourself.
    - Eg, no default font or font color.

ebitenui

- Since this is ebitenui, it is design to work with the ebiten library
  - There is a draw and update in the UI object
  - If you are using a camera
    - Call ui.Draw after you do all of your camera drawing to the screen
      - Or your UI will end up being drawn offscreen.
    -

Ebitenui

- The struct you have to build

ebitenui.UI

- It has one field that you have to fill
  - A container that will be the root container of the GUI
  - Remind us again what is a container?
    - Lucky volunteer

Ebitenui

- The struct you have to build

ebitenui.UI

- It has one field that you have to fill
  - A container that will be the root container of the GUI
  - Remind us again what is a container?
    - Invisible UI element that holds the visible components/widgets
    - You apply layouts to containers.

- Set properties of a container with ContainerOpts
  - Like
    - Layout
    - WidgetOpts (to style the widgets in the container

Button

- Very solid but longer tutorial on docs site
  - https://ebitenui.github.io/widgets/button/index.html
  - Create one with

widget.NewButton

  - Parameters should be a list of ButtonOpts objects

widget.ButtonOpts

- Common ButtonOpts

widget.ButtonOpts.ClickedHandler

  - Pass a function to be called when button pressed.

widget.ButtonOpts.TextLabel

widget.ButtonOpts.TextFace

widget.ButtonOpts.TextColor

  - If your button has text, set it with TextLabel, and then you have to have a font and TextColor (struct)

widget.ButtonOpts.Image

  - This will take a struct with the custom images for the button's various states
    - Normal, mousehover, pressed etc

Label

- What is a label control/widget in GUIs?

Label

- What is a label control/widget in GUIs?
  - Bit of text to explain nearby elements to user
  - Create one with

widget.NewLabel

  - Pass a set of LabelOpts as params
  - A more simple widget so fewer opts typically

- Two opts that likely matter

widget.LabelOpts.LabelColor

- Takes a LabelColor struct

widget.LabelOpts.LabelFace

- Takes a font fact

Text Input

- Create a TextInput (text box) using

widget.NewTextInput

- And as parameters pass several TextInputOpts objects like

widget.TextInputOpts.Face

- To pass general WidgetOpts

widget.TextInputOpts.WidgetOpts

- To adjust the pixel offset of text use

widget.TextInputOpts.Padding

- To set the image for the widget (see nineslice soon)

widget.TextInputOpts.Image

  - Takes a TextInputImage struct

- And for font color

widget.TextInputOpts.Color

  - Takes a TextInputColor struct

Color Structs

- All of these color structs contain several colors each
  - Why do you suppose?
  - Lucky volunteer?

Color Structs

- All of these color structs contain several colors each
  - Can use different colors in different situation
    - Selected
    - Normal ect.

- TextInputColor example

```
widget.TextInputColor{
        Idle:         colornames.Bisque,
        Disabled:     colornames.Gray,
        Caret:        colornames.Black,
        DisabledCaret: colornames.Gray,
}
```

- ButtonColor

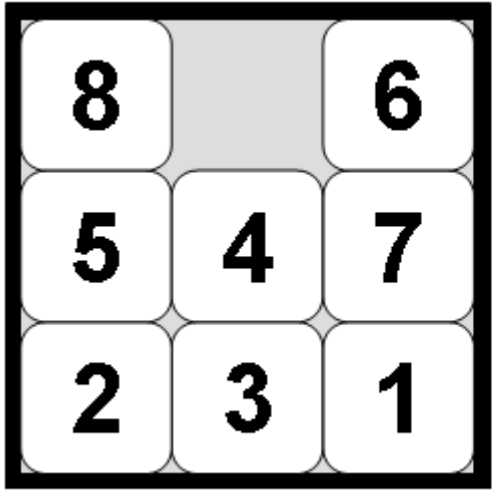
```
widget.ButtonTextColor{
        Idle:     colornames.Azure,
        Disabled: colornames.Gray,
        Hover:    colornames.Aquamarine,
        Pressed:  colornames.Aquamarine,
}
```

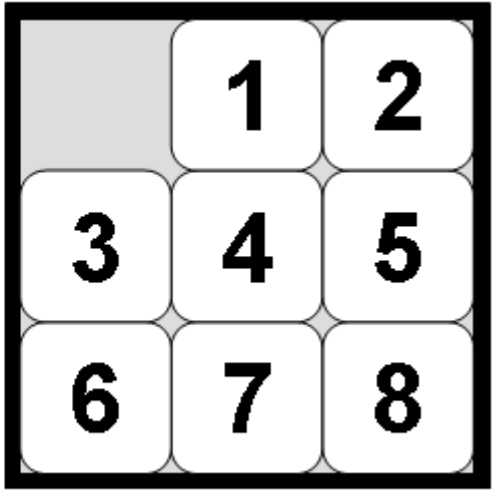
Nine Slice

- What is an 8-Puzzle?
  - Lucky volunteer?

Nine Slice

- What is an 8-Puzzle?
  - Puzzle divided into 9 locations
    - 8 sliding tiles
    - And space
- In the case of this puzzle
  - All 9 parts are equal sizec





Nine Slice

- Because in game UI,
  - you need to use arbitrary images for widgets
  - And those widgets need to be a variety of sizes
- Solution: chop the image into nine parts and stretch the middle as much as we need to.