Making Games in Go

(For Fun)

Miłosz Smółka

Three Dots Labs

threedots.tech

Why games?

I used to love coding, but now the job is constant meetings and status reports.
Did I lose my passion? :(

Code for fun again!

Why Go?

Why not a "real" engine?

Disclaimer #1

I'm a hobbyist

Disclaimer #2

It takes some fun out of playing games

Disclaimer #3

I focus on 2D games

All materials at the end!

Don't try to remember all of it.

What is a video game?

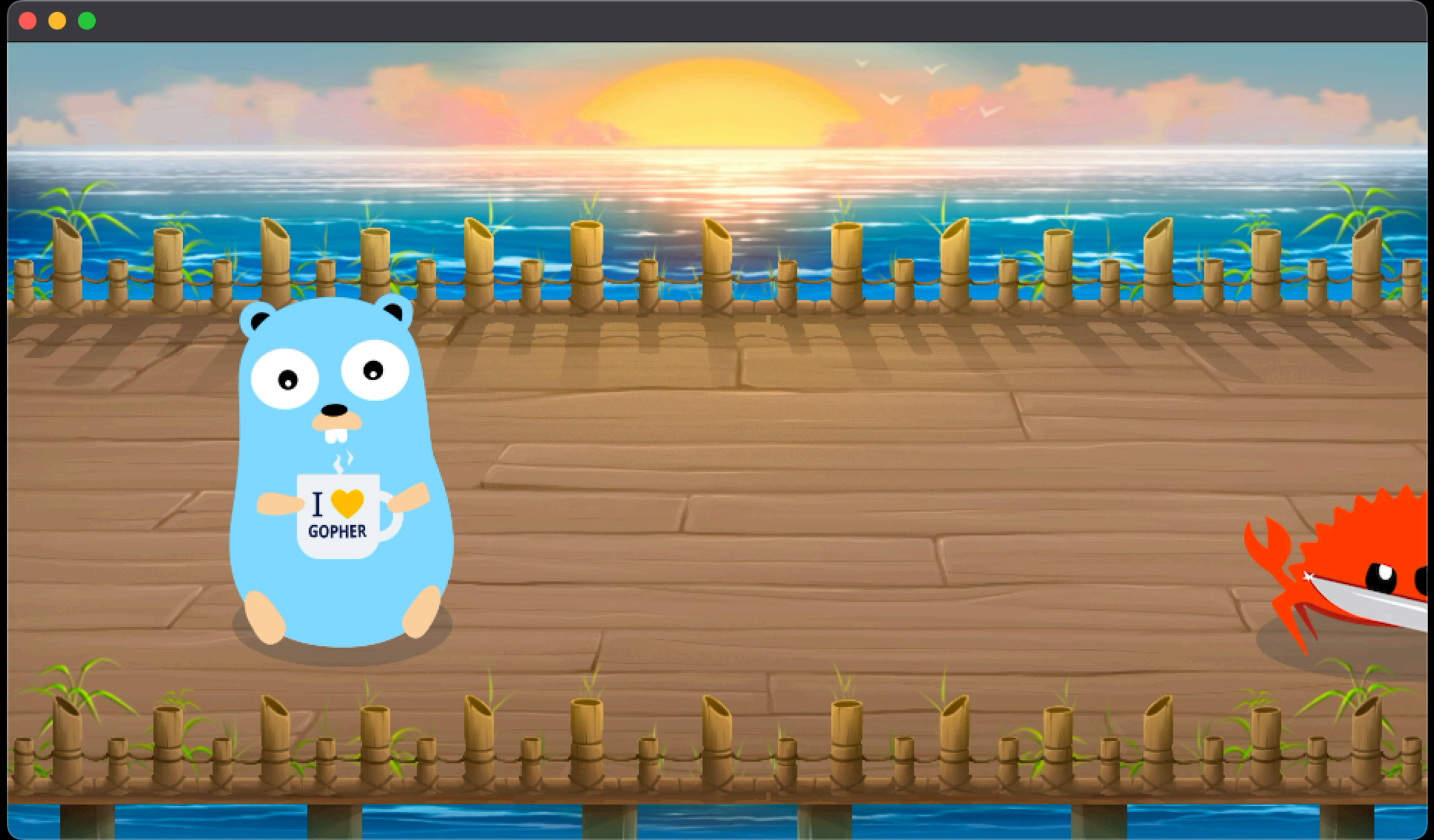

1/2
0 Points
Laps 1/2
Time 1:11.00
Best 0:00.00
Lamborghini
ND 4 SPD
1
2
3
4
5
6
7
8
9
136 MPH

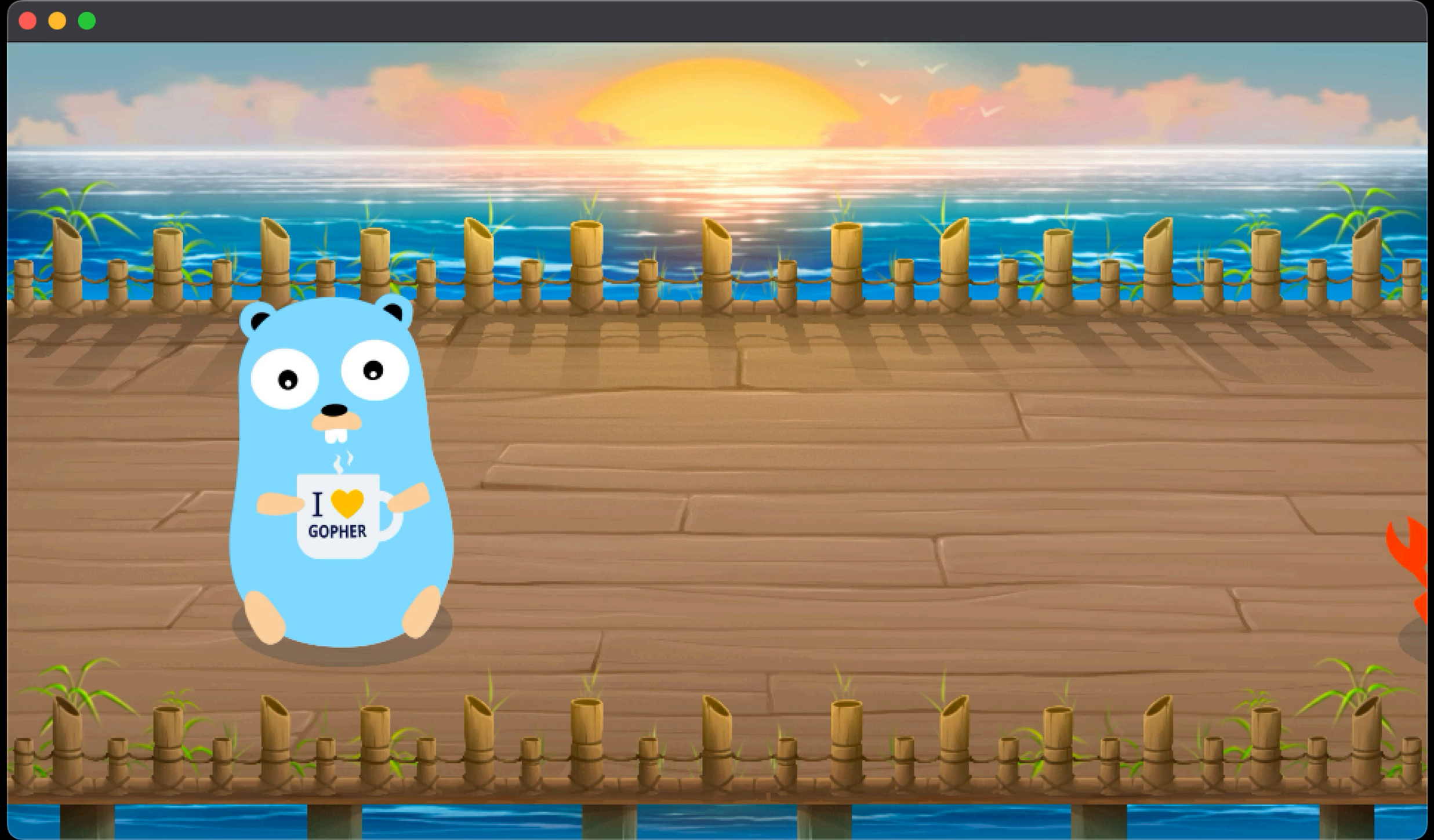
```
What's the number? 5
WRONG!!!!
What's the number? 9
YOU WIN!!!!!
What's the number?
```

```
What's the number? 5
WRONG!!!!
What's the number? 9
YOU WIN!!!!!
What's the number?
```

???

What I wish I knew at the beginning


I
GOPHER


I
GOPHER


I
GOPHER

1, 5, 60 FPS

(Frames Per Second)

An optical illusion

Just like a movie.


DUNE
PART TWO

But we don't have frames…

…so we need to generate them.

(60 per second)

The Game Loop

```
for {
    DrawFrame(screen)
}
```

Actually not trivial

```
func DrawFrame(screen) {
    // RENDER PIXELS
    // SOMETHING SOMETHING SHADERS
    // ✨ MAGIC GPU CALLS ✨
    // ???
    // FRAME ON THE MONITOR
}
```

(I don't care)

Ebitengine™

A dead simple 2D game engine for Go

A lightweight library with a nice API

(not a framework)

```
type Game interface {
    Update() error
    Draw(screen *ebiten.Image)
    Layout(width, height int) (int, int)
}
```

Let's focus on

`Draw`

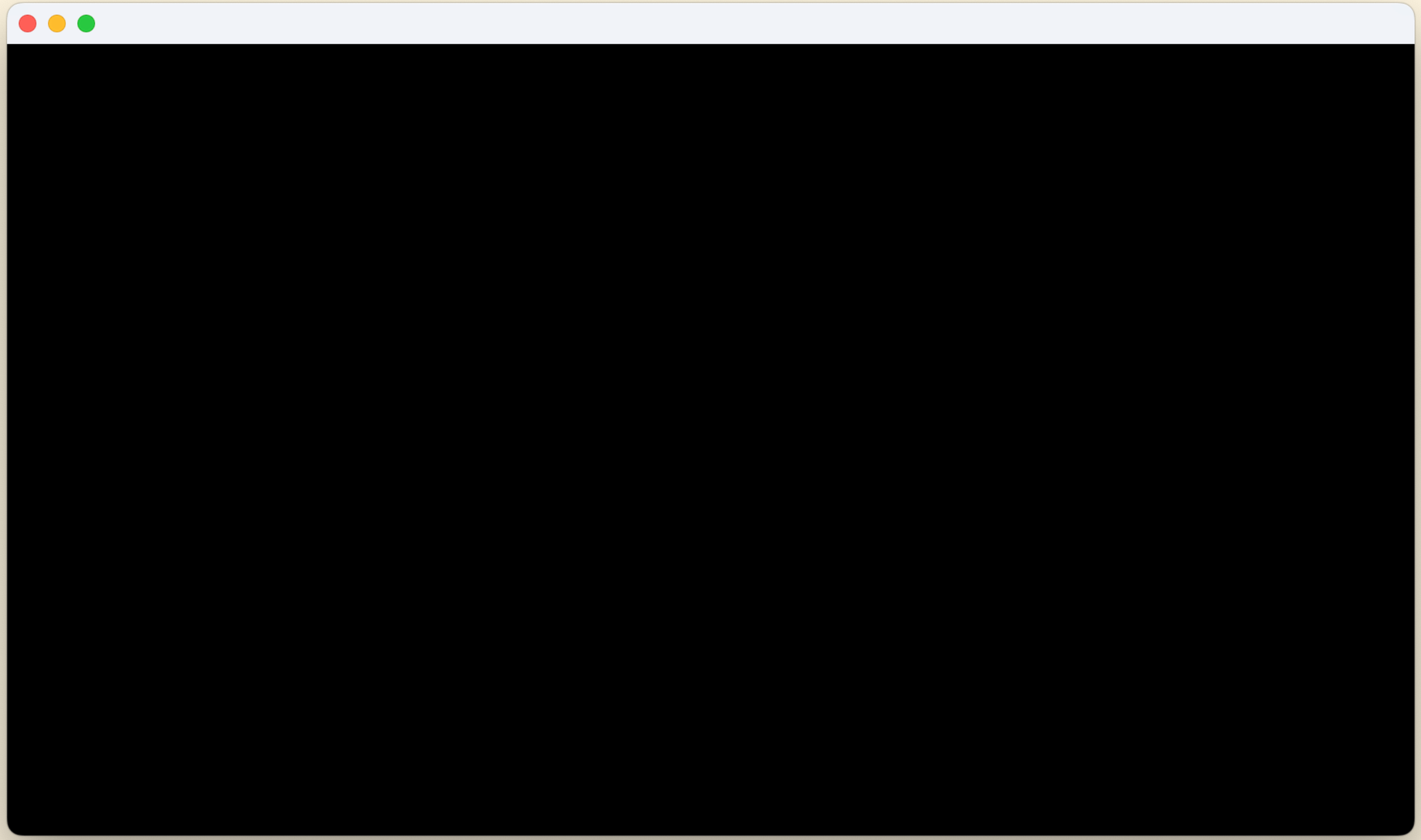
```
type Game struct {}

func (g *Game) Draw(screen *ebiten.Image) {

}
```

`ebiten.Image`

A collection of pixels.

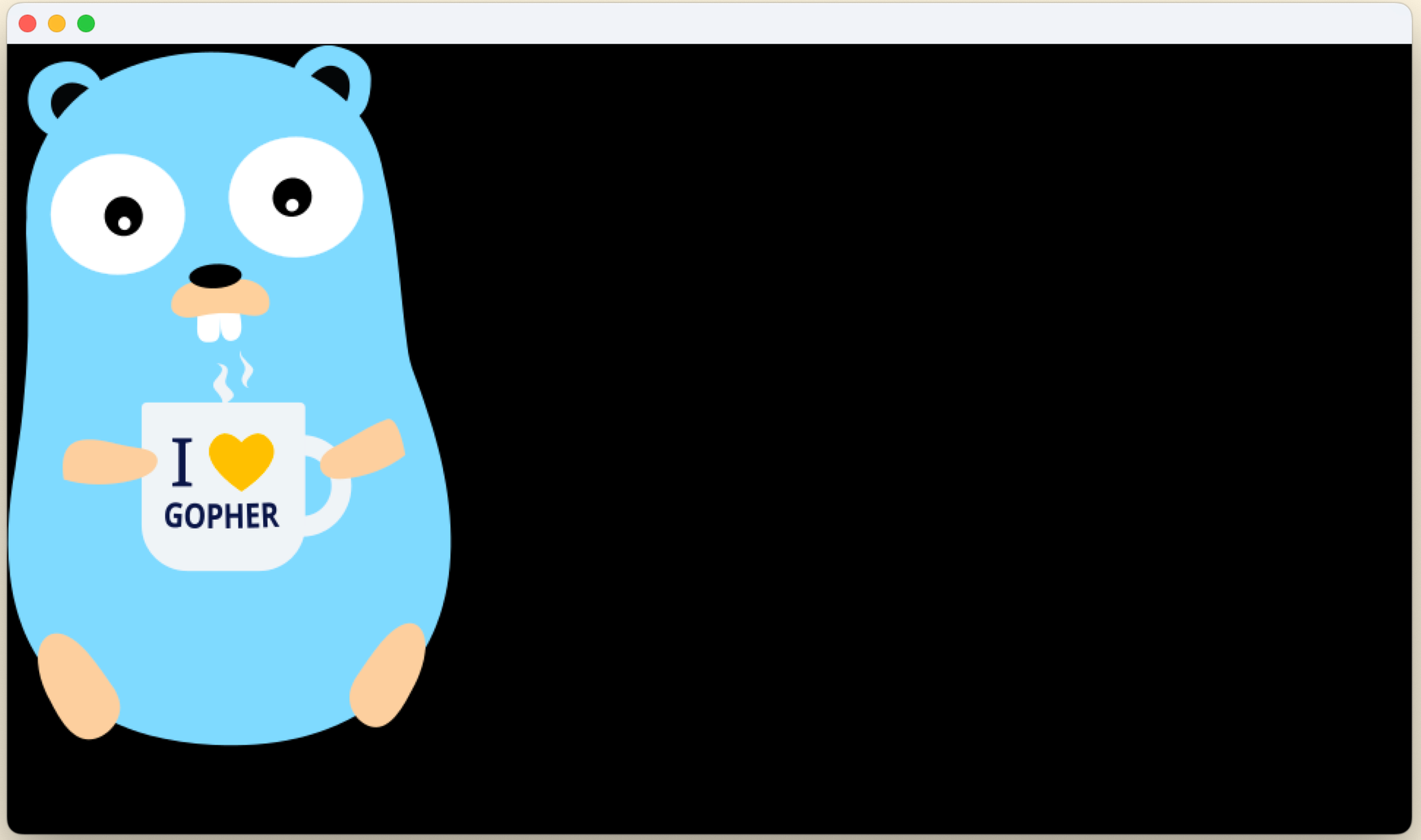


DrawImage

```
screen.DrawImage(image, options)
```

```
var Gopher = LoadImage("gopher.png")

func (g *Game) Draw(screen *ebiten.Image) {
    screen.DrawImage(Gopher, nil)
}
```


I
GOPHER

The `screen` starts empty every frame

Executed 60+ times per second!

(computers are fast) 👉

Moving around

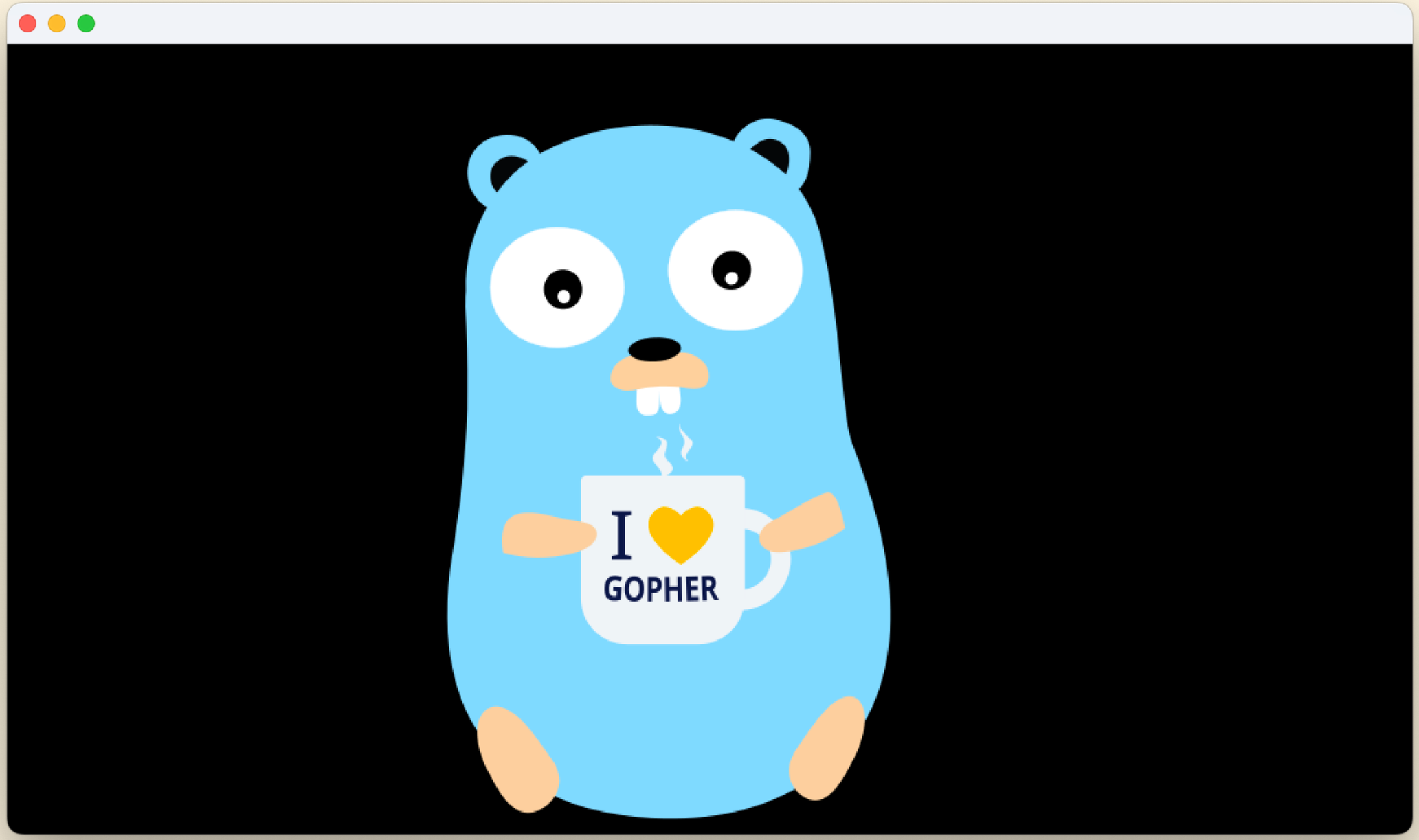

(0, 0)
X
Y

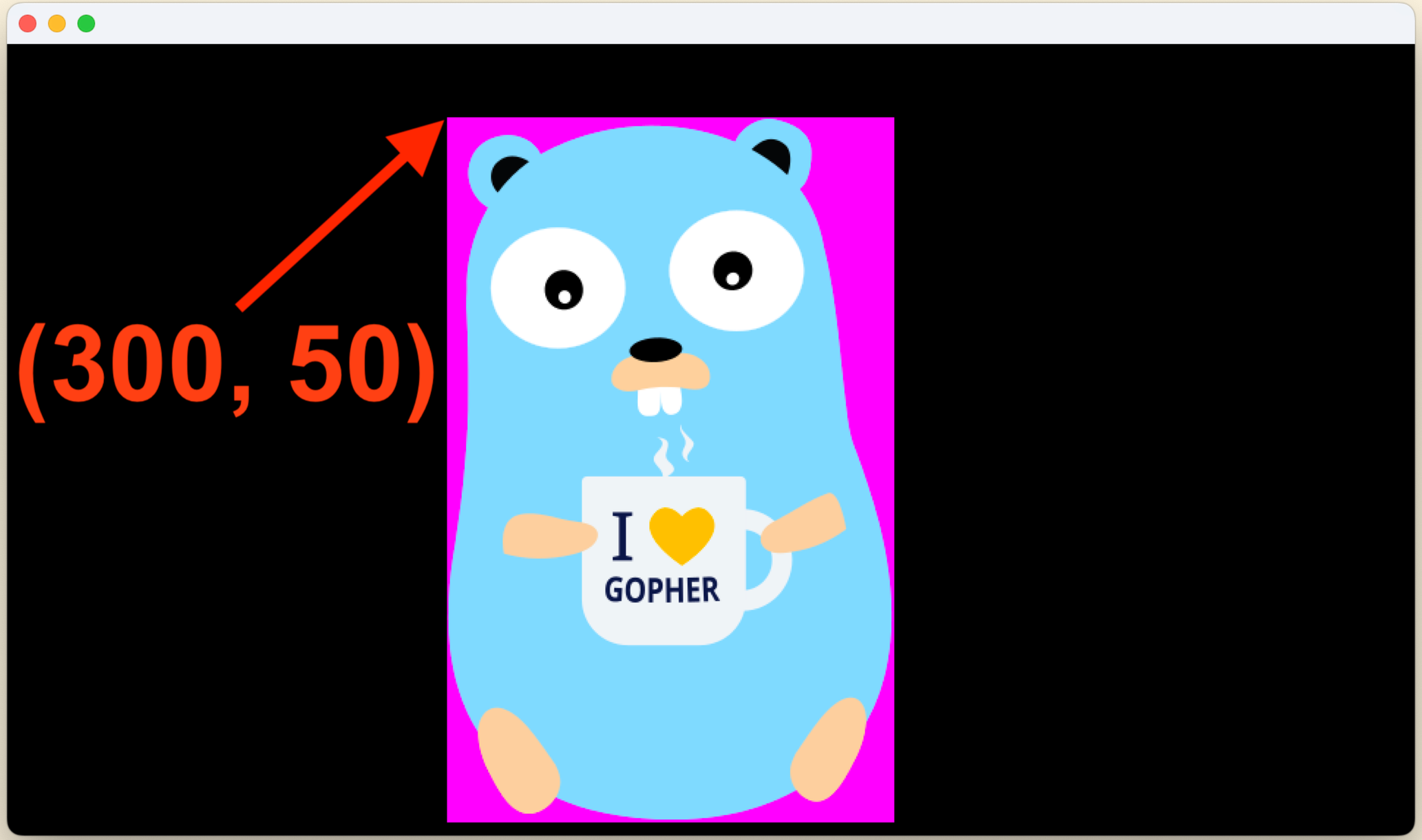
```
op := &ebiten.DrawImageOptions{}
```

```
op.GeoM.Translate(x, y)
```

```
op := &ebiten.DrawImageOptions{}
op.GeoM.Translate(300, 50)

screen.DrawImage(Gopher, op)
```


I
GOPHER


(300, 50)
I
GOPHER

```
op.GeoM.Translate()
```

Geo… what?

A *geometry matrix*, of course!

Matrix (mathematics)

From Wikipedia, the free encyclopedia

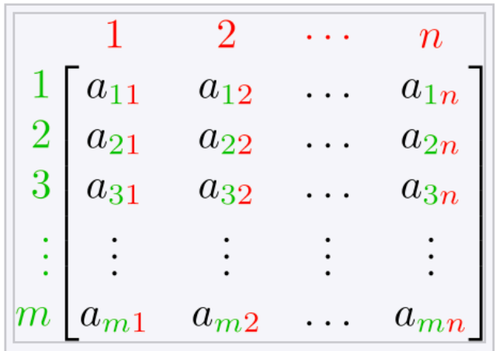
"Matrix theory" redirects here. For the physics topic, see Matrix theory (physics). For other uses of "Matrix", see Matrix (disambiguation).

In mathematics, a **matrix** (pl.: **matrices**) is a rectangular array or table of numbers, symbols, or expressions, with elements or entries arranged in rows and columns, which is used to represent a mathematical object or property of such an object.

For example,

$$\begin{bmatrix} 1 & 9 & -13 \\ 20 & 5 & -6 \end{bmatrix}$$

is a matrix with two rows and three columns. This is often referred to as a "two-by-three matrix", a "2×3 matrix", or a matrix of dimension 2×3.



An $m \times n$ matrix: the m rows are horizontal and the n columns are vertical. Each element of a matrix is often denoted by a variable with two subscripts. For example, $a_{2,1}$ represents the element at the second row and first column of the matrix.

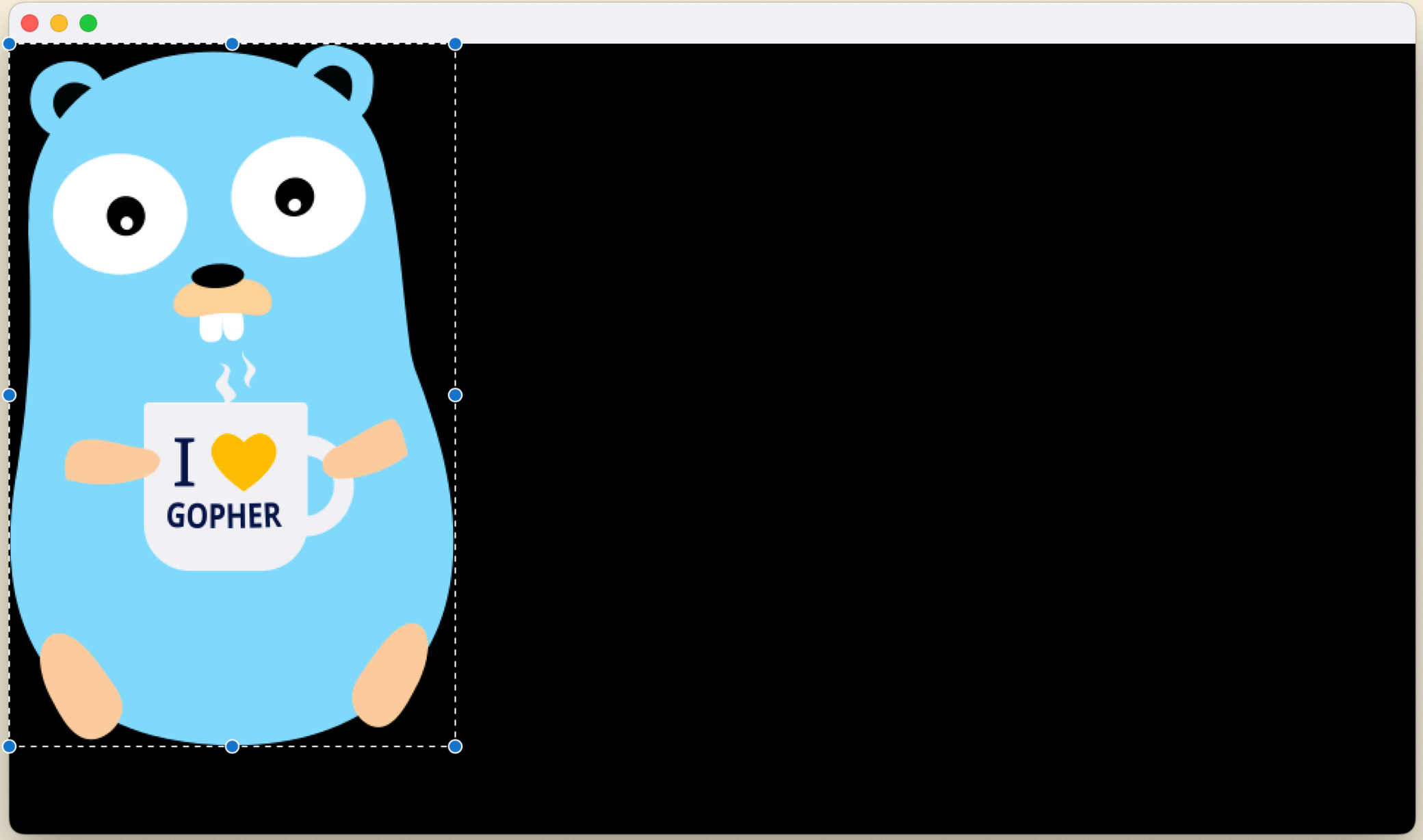
Matrices are commonly related to linear algebra. Notable exceptions include incidence matrices and adjacency matrices in graph theory.[1] This article focuses on matrices related to linear algebra, and, unless otherwise specified, all matrices represent linear maps or may be viewed as such.

Square matrices, matrices with the same number of rows and columns, play a major role in matrix theory. Square matrices of a given dimension form a noncommutative ring, which is one of the most common examples of a noncommutative ring. The determinant of a square matrix is a number associated with the matrix, which is fundamental for the study of a square matrix; for example, a square matrix is invertible if and only if it has a nonzero determinant and the eigenvalues of a square matrix are the roots of a polynomial determinant.

(no one cares)

Instead…

Think of copy-pasting an image


I
GOPHER

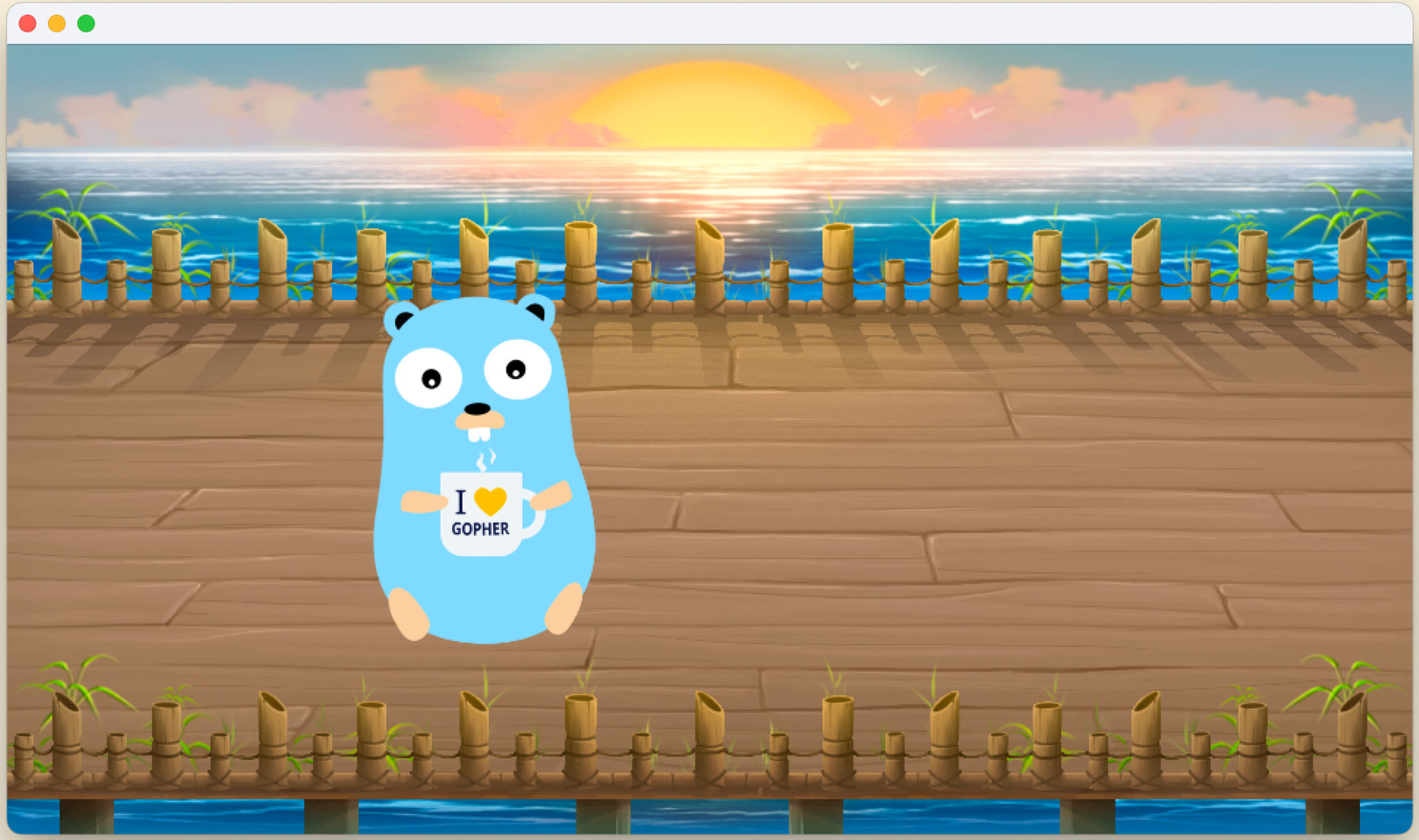
Calls are relative

```
op.GeoM.Translate(10, 0)
op.GeoM.Translate(-3, 0)
// Moved by (7, 0)
```

The order of drawing matters!

Draw the background first

```
screen.DrawImage(Background, op)
// ...
screen.DrawImage(Gopher, op)
```


I
GOPHER

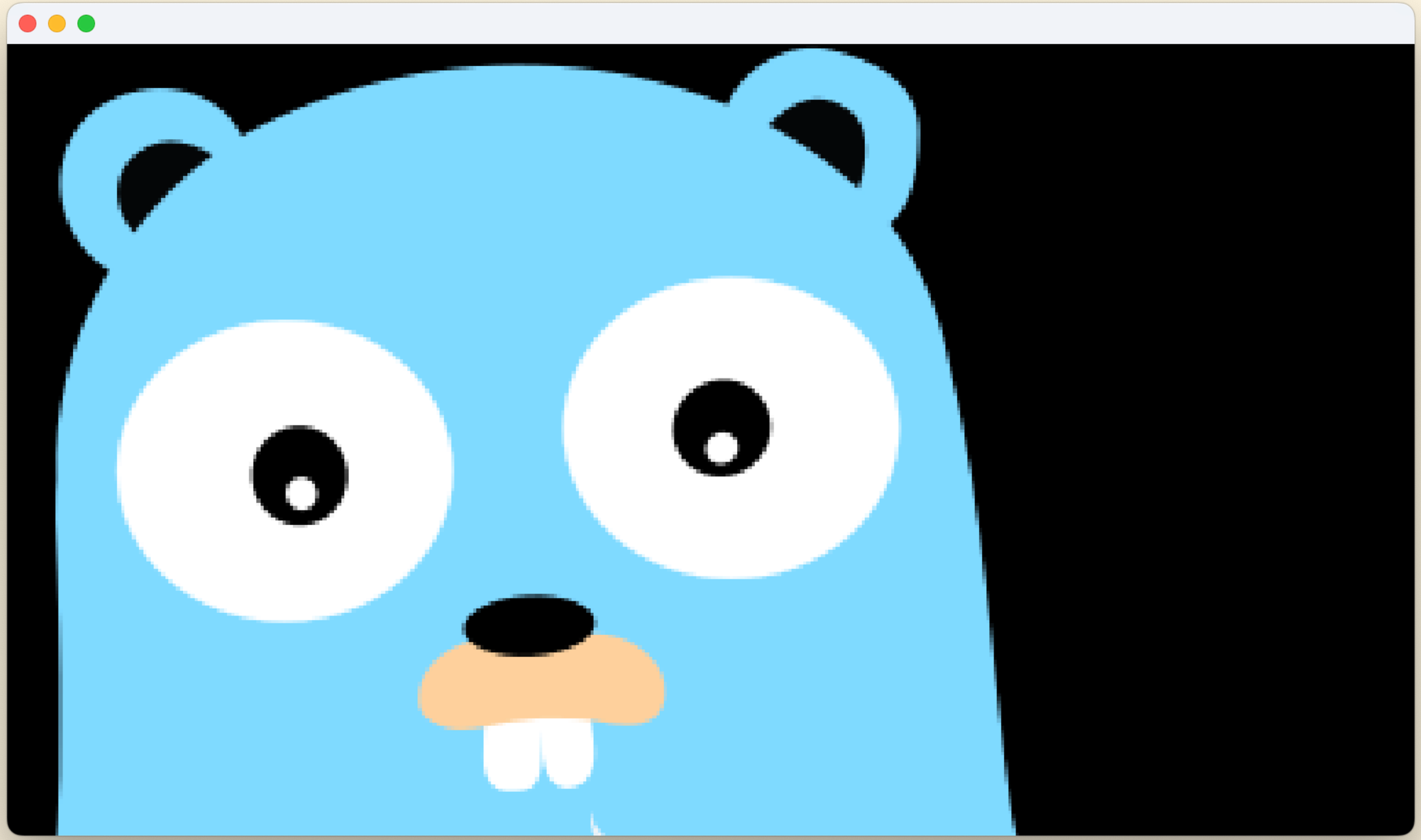
Don't re-use `op`

Unless you want to.

```
op := &ebiten.DrawImageOptions{}
screen.DrawImage(Background, op)
// ...
op = &ebiten.DrawImageOptions{}
op.GeoM.Translate(300, 50)
screen.DrawImage(Gopher, op)
```

Scale

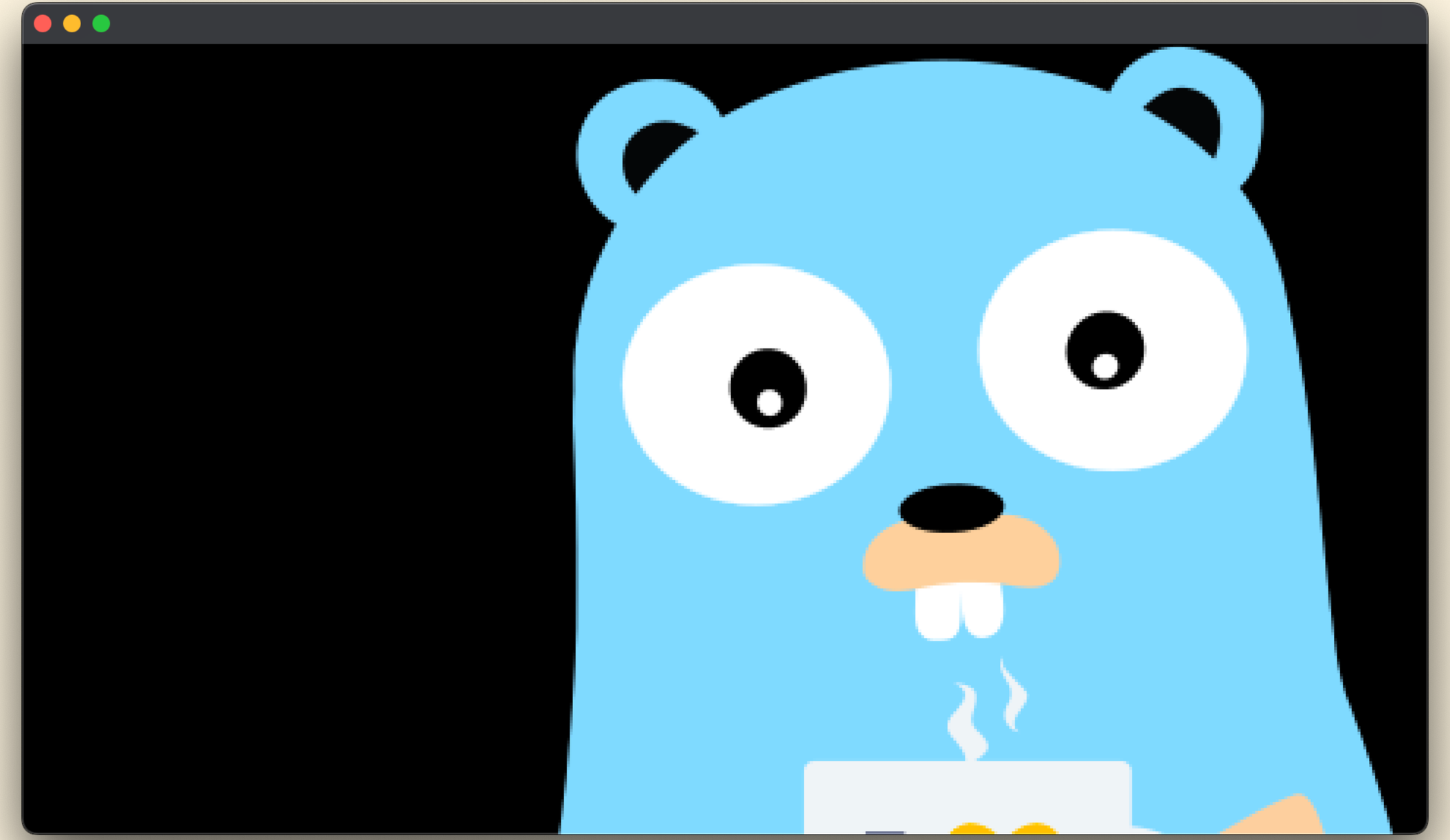
```
op.GeoM.Scale(2.5, 2.5)
```



The order matters!

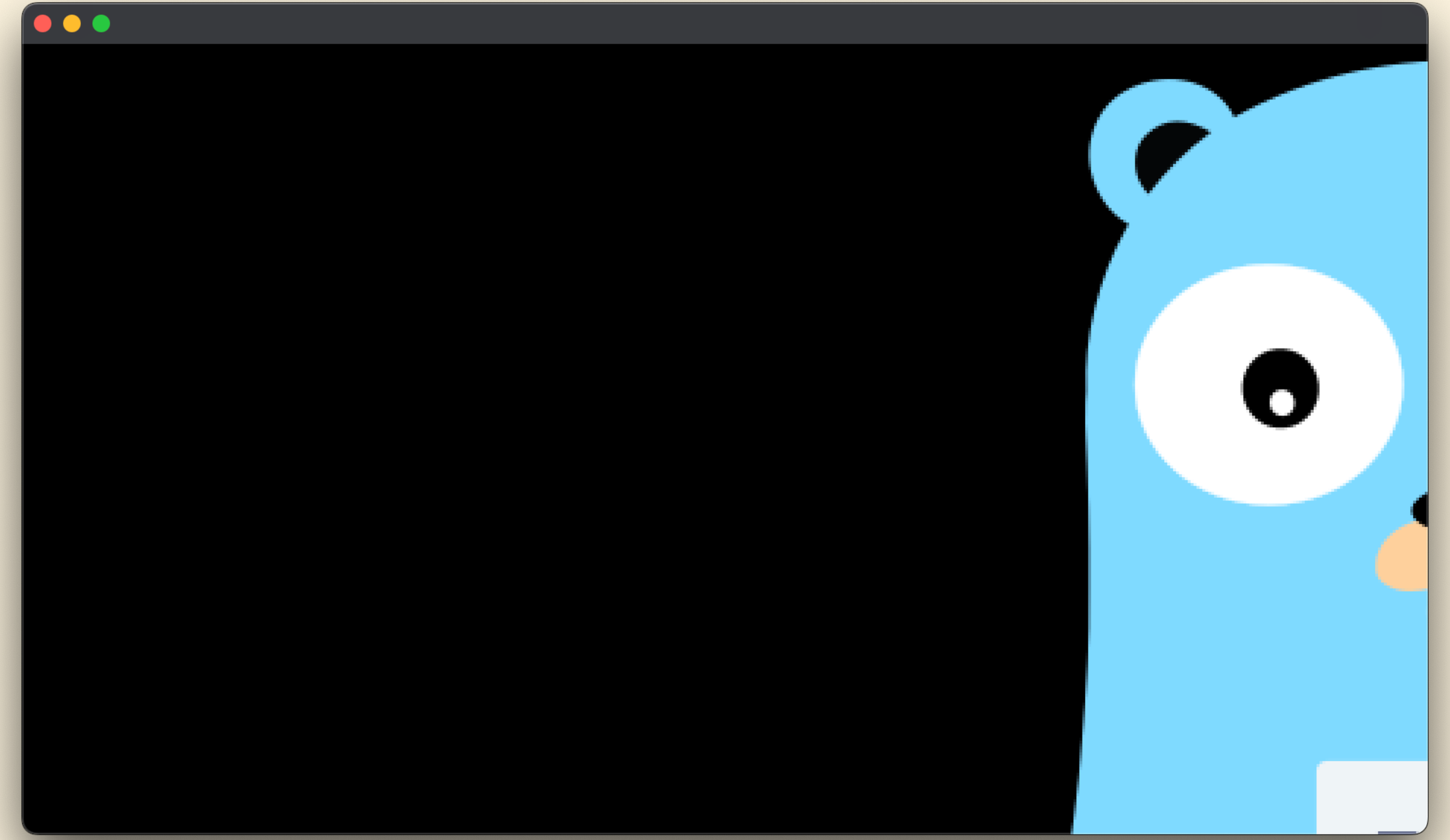
Scale, then move

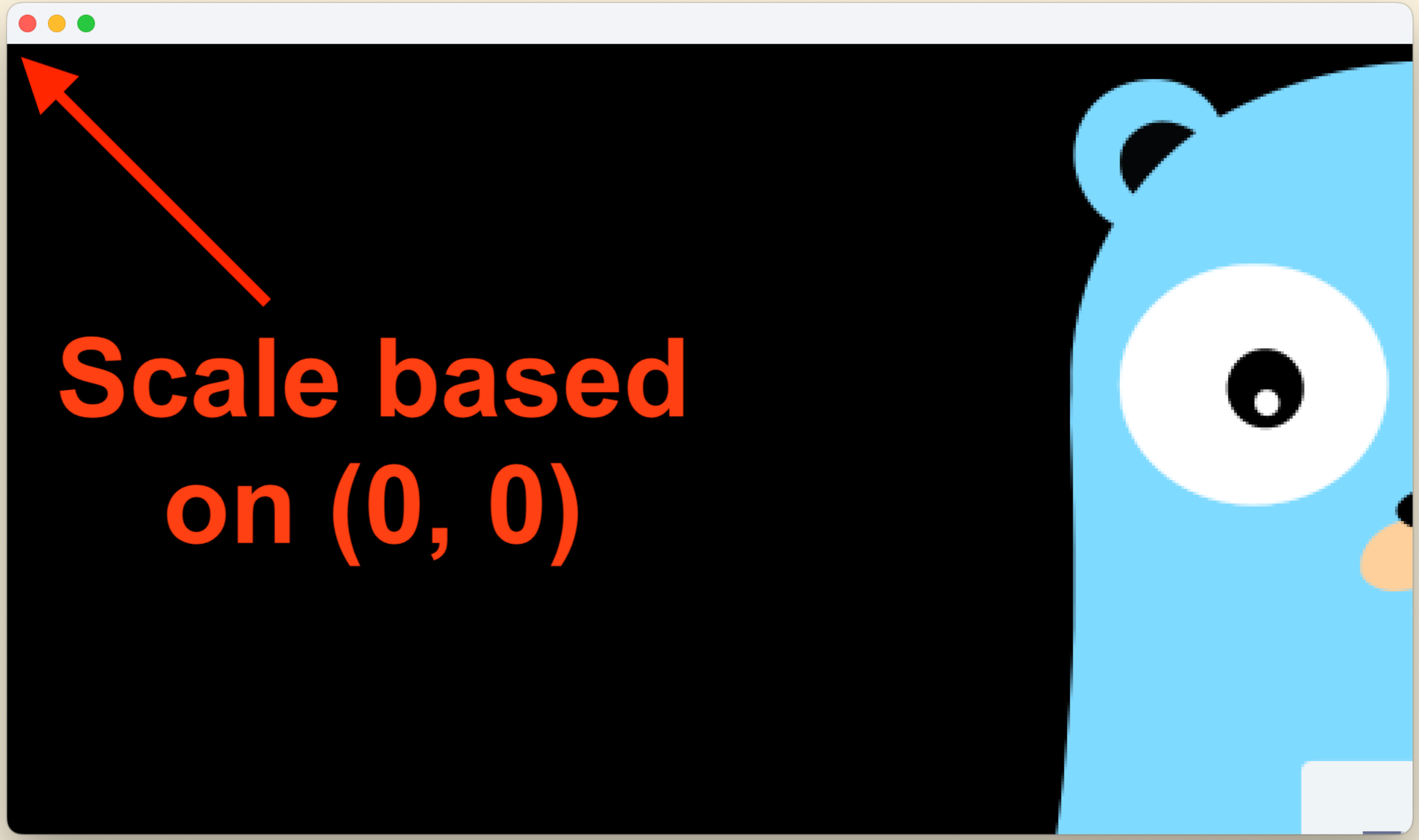
```
op.GeoM.Scale(2, 2)
op.GeoM.Translate(300, 0)
```

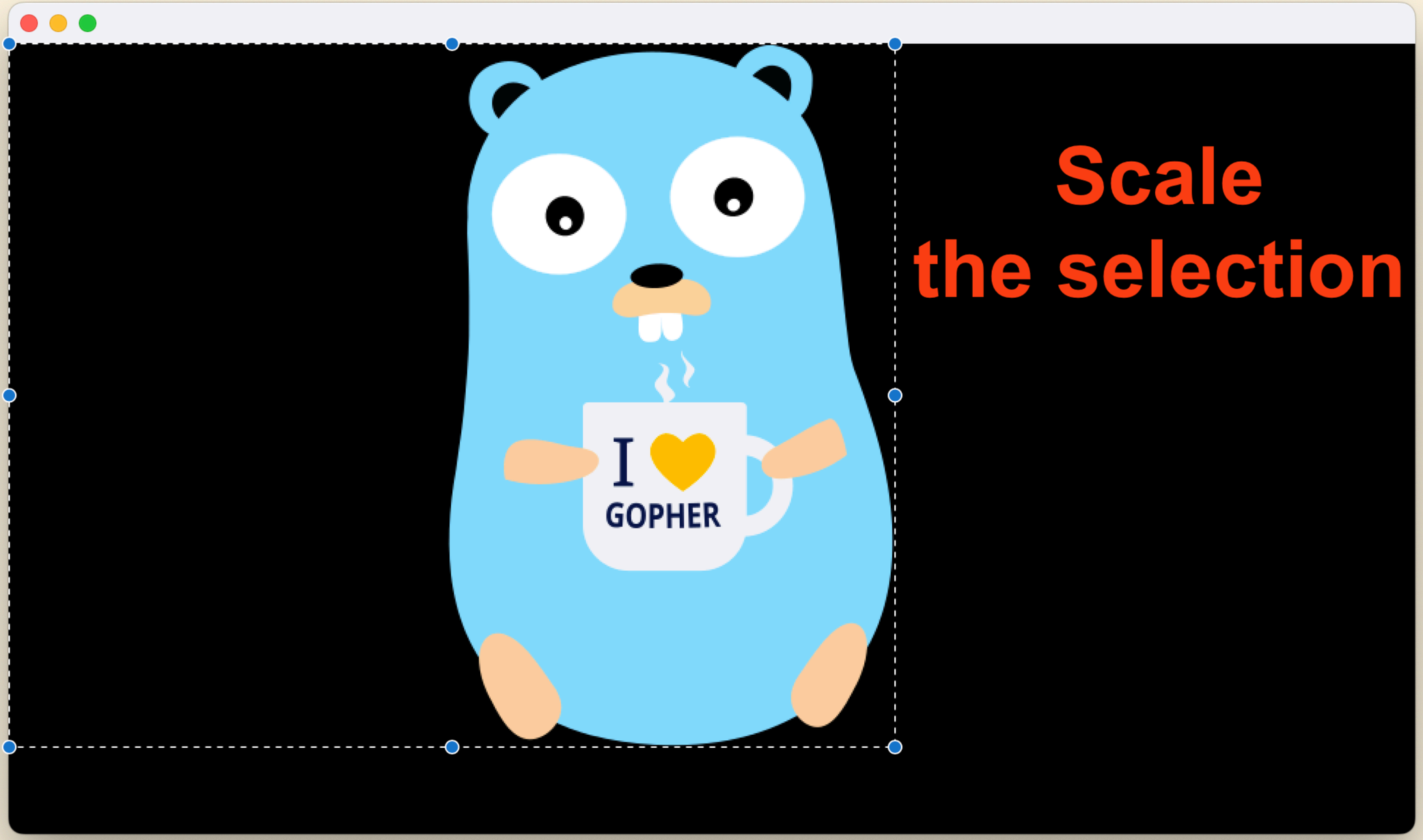


Move, then scale

```
op.GeoM.Translate(300, 0)
op.GeoM.Scale(2, 2)
```




Scale based
on (0, 0)

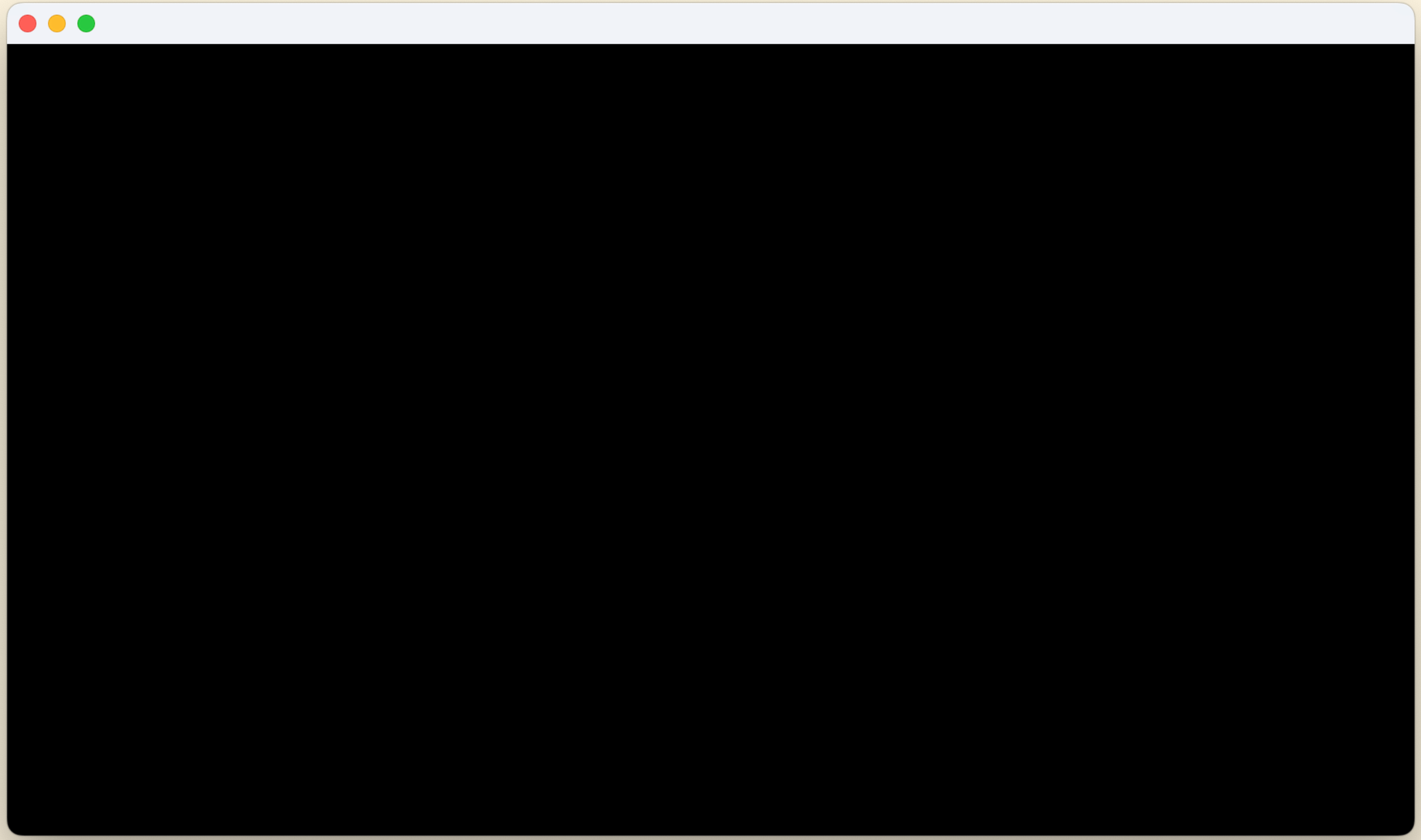

I
GOPHER
Scale
the selection

A rule of thumb:

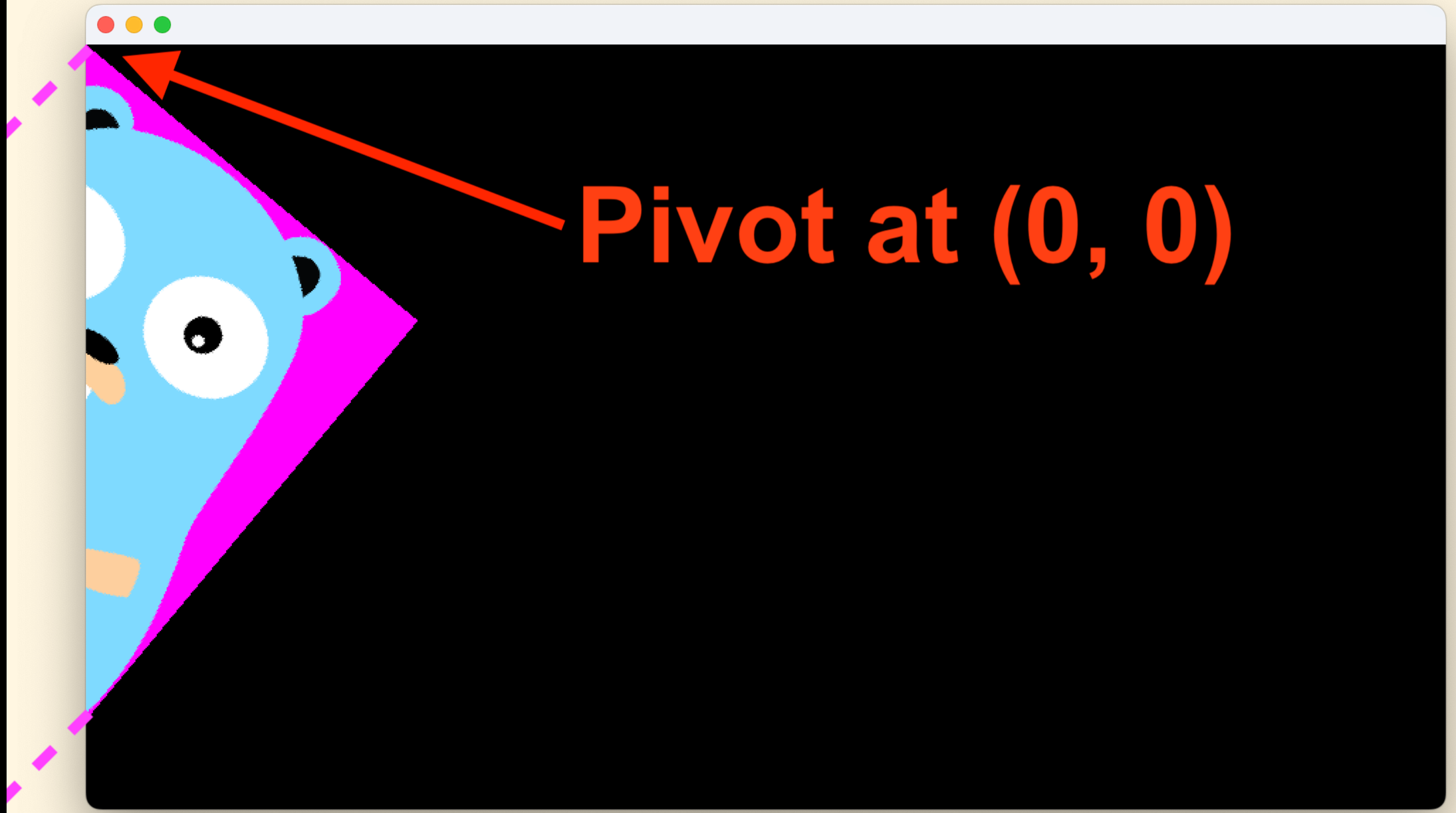
`Translate` last

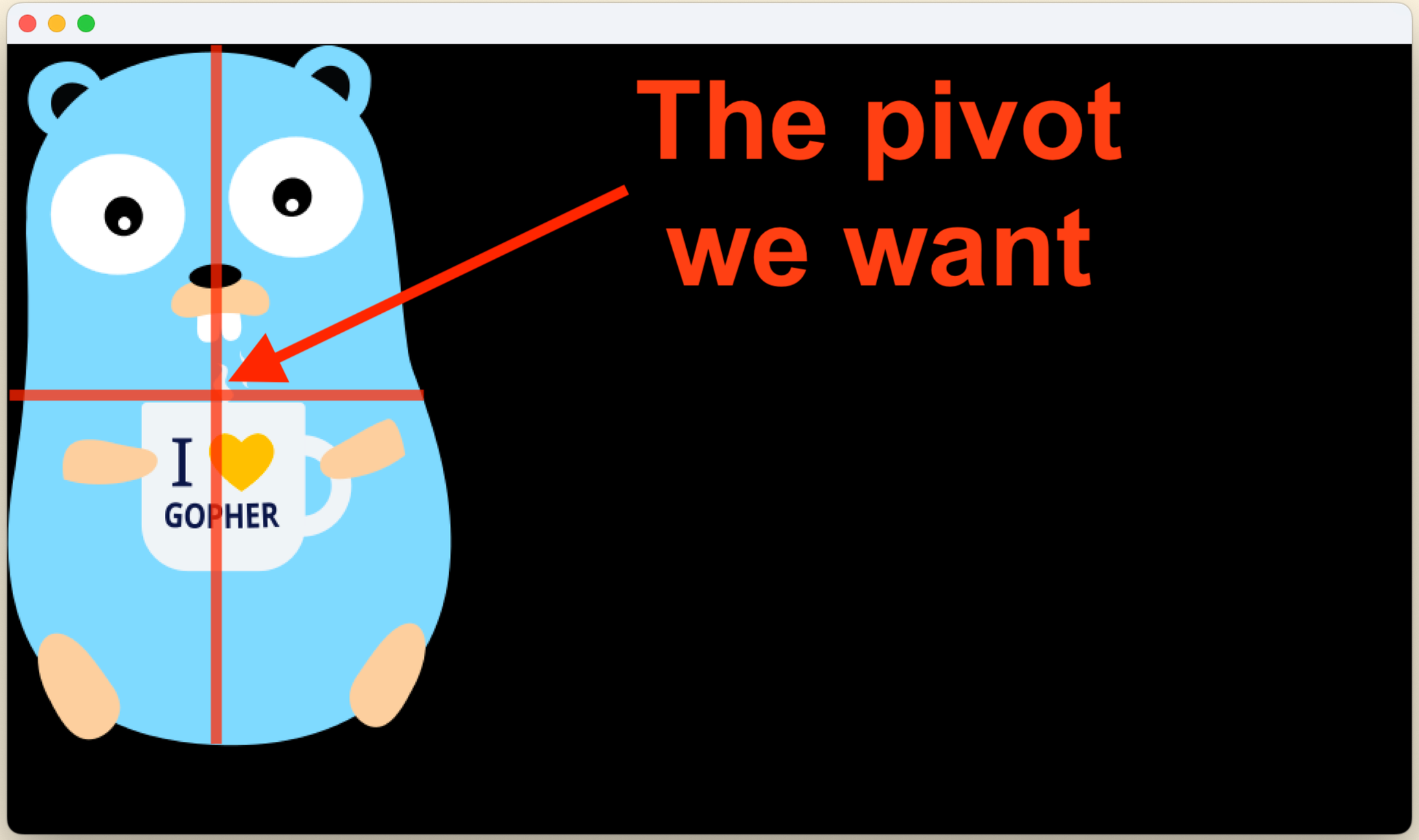
Rotating

```
op.GeoM.Rotate(toRadians(90))
```



What happened?


Pivot at (0, 0)

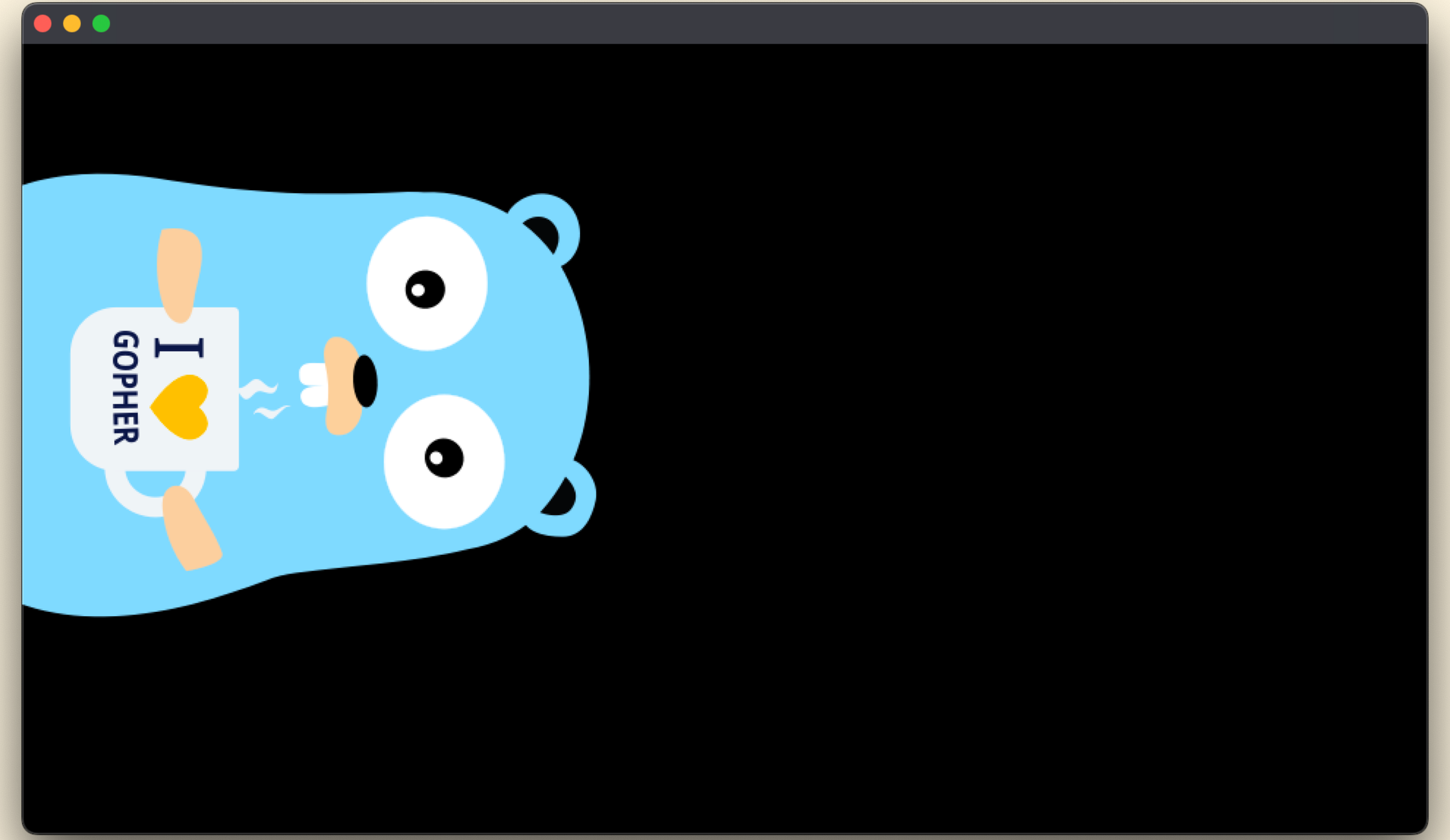

The pivot
we want
I
GOPHER

First, get bounds

```
bounds := Gopher.Bounds()
halfWidth := float64(bounds.Dx() / 2)
halfHeight := float64(bounds.Dy() / 2)
```

Move back and forth

```
op.GeoM.Translate(
    -halfWidth,
    -halfHeight,
)
op.GeoM.Rotate(toRadians(90))
op.GeoM.Translate(
    halfWidth,
    halfHeight,
)
```


I
GOPHER

Flip

Scale with -1

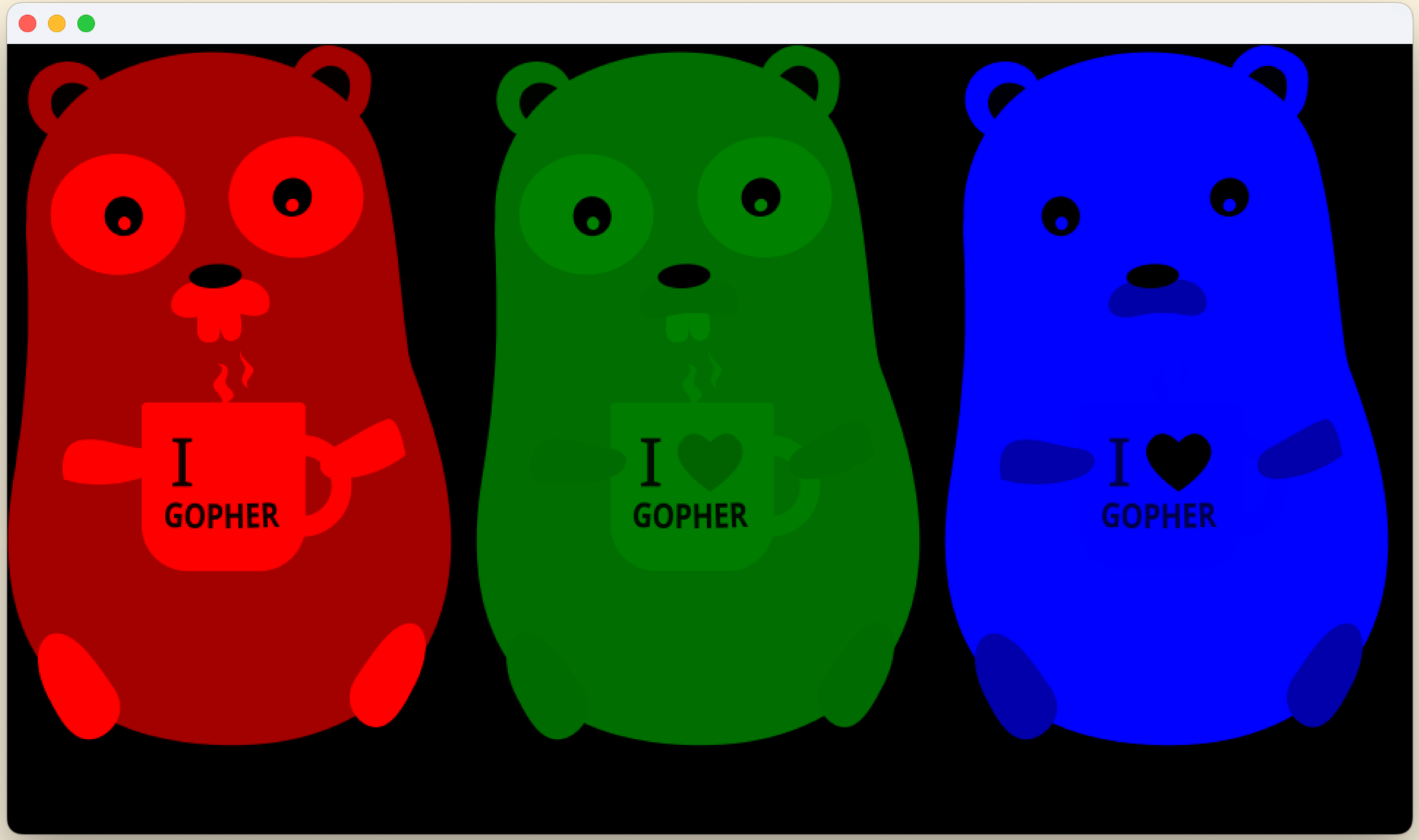
The same trick

```
op.GeoM.Translate(
    -halfWidth,
    -halfHeight,
)
op.GeoM.Scale(1, -1)
op.GeoM.Translate(
    halfWidth,
    halfHeight,
)
```

I

Color

```
op.ColorScale.ScaleWithColor(colornames.Red)
```


I
GOPHER
I
GOPHER
I
GOPHER

```
op.ColorScale.ScaleAlpha(0.5)
```


I
GOPHER
I
GOPHER
I
GOPHER

Drawing Essentials

- `Draw()`
- `image.DrawImage()`
- Position
- Scale
- Rotation
- Color & Transparency

Combine them!


I ♥ GOPHER

We need more than one frame.

The Game Logic

Working with *the state*.

State?

```
type Game struct {
    // This is your state
}
```

```
type Vector struct {
    X float64
    Y float64
}

type Game struct {
    playerPosition *Vector
}
```

Hardcoded values are boring

```
op := &ebiten.DrawImageOptions{}
op.GeoM.Translate(300, 50)

screen.DrawImage(Gopher, op)
```

Use values from the state

```
op := &ebiten.DrawImageOptions{}
op.GeoM.Translate(
    g.playerPosition.X,
    g.playerPosition.Y,
)

screen.DrawImage(Gopher, op)
```

The state changes over time

The Game Loop

```
for {
    DrawFrame(screen)
}
```

The Game Loop

```
for {
    Update()
    DrawFrame(screen)
}
```

```
func (g *Game) Update() error {
    return nil
}
```

One update

=

One *Tick*

Running at 60 ticks per second (TPS)

All in a single loop

Forget goroutines for now

```
func (g *Game) Update() error {
    g.playerPosition.X += 1
    return nil
}
```

Working with time

Counting ticks

```
type Game struct {
    ticks int
}
```

```
g.ticks++

if g.ticks == 120 {
    g.movingLeft = !g.movingLeft
    g.ticks = 0
}
```

Not the best API

Calculate with

```
ebiten.TPS()
```

Timers

```
timer := NewTimer(2*time.Second)

// ...

g.timer.Update()
if g.timer.IsDone() {
    // ...
    g.timer.Reset()
}
```

Use "% done" to scale values over time

```
op.ColorScale.ScaleAlpha(
    g.timer.PercentDone(),
)
```

Input

No *events*!

Just `if` conditions in the `Update`.

IsKeyPressed()

```
if ebiten.IsKeyPressed(ebiten.KeyD) {
    g.playerPosition.X += 1
}
```

```
inpututil.
IsKeyJustPressed()
IsKeyJustReleased()
```

```
if inpututil.IsKeyJustPressed(
    ebiten.KeySpace) {
    g.jumping = true
}
```

Similar for mouse, gamepad, and touch.

Game Objects

Grouping data.

Structs

```
type Object struct {
    Position *Vector
    Image    *ebiten.Image
}
```

State

```
type Game struct {
    objects []*Object
}
```

Drawing

```
for _, obj := range g.objects {
    op := &ebiten.DrawImageOptions{}
    op.GeoM.Translate(
        obj.Position.X,
        obj.Position.Y,
    )
    screen.DrawImage(obj.Image, op)
}
```

```
for _, obj := range g.objects {
    obj.Position.X -= 2
}
```

Logic Essentials

- `Update`
- State
- Timers
- Input
- Game Objects

Layout

The last method.

(check the docs)

```
func (g *Game) Layout(
    width, height int) (int, int) {
    return width, height
}
```

Remember:

It's an optical illusion


I
GOPHER

Just keep on drawing images.

It's all there is.

Beyond Essentials

Very briefly.

`embed` for portable assets

```
//go:embed assets/gopher.png
var gopher []byte

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

UI

```
text.Draw()
```

Just like `DrawImage()`

```
op := &text.DrawOptions{}
op.GeoM.Translate(250, 180)

text.Draw(
    screen,
    "GAME OVER",
    Font,
    op,
)
```


Game Over

Animations

Change the sprite over time.

```
var Zombie []*ebiten.Image

type Game struct {
    timer *Timer
    index int
}
```

Update

```
g.timer.Update()
if g.timer.IsDone() {
    g.index++
    if g.index >= len(Sprites)-1 {
        g.index = 0
    }
    g.timer.Reset()
}
```

Draw

```
screen.DrawImage(
    Zombie[g.index],
    op,
)
```

Or simply animate drawing options


I
GOPHER

Cameras

Different points of view.


Camera
Position

Draw the game on a separate image

Keep the offscreen

```
type Game struct {
    cameraPosition *Vector
    offscreen      *ebiten.Image
}
```

Draw

```
g.offscreen.Clear()
g.offscreen.Draw(...)

op.GeoM.Translate(
    -g.cameraPosition.X,
    -g.cameraPosition.Y,
)
screen.DrawImage(g.offscreen, op)
```

Update the position

```
g.cameraPosition.X += 1
```

Hierarchy

Parent + children.

```
type Object struct {
    Position *Vector
    Image    *ebiten.Image
    Children []*Object
}
```

Debug Mode

Cheats on!

`vector`

Drawing primitive shapes

Audio

Music & Sound Effects ✅

Platforms

Runs on web!

Runs on

mobile!


Inventory (0)
- Zadok Allan -
You recognize this man as Zadok, the drunk the boy told you about.
"What do you want?"
-> "Here, have a sip." - hand him the whiskey bottle
Zadok takes a sip.
-> Wait.
"Look at the ocean..."
-> Take a Look
"GOD HELP US! RUN!"
What...?

Runs on

Nintendo Switch!

But...

You're good to go!

Check the materials!


Examples:
1. Static
2. Moved
3. Scale
4. FirstScaleThenMove
5. FirstMoveThenScale
6. NaiveRotate
7. Rotate
8. RotateStepByStep
9. Flip
10. RGB
11. Alpha
12. Party
13. ConstantVelocity
14. TicksCounting
15. WithTimer
16. Objects
17. Layers
18. Camera
19. StaticAnimation
20. Animation
21. Hierarchy
22. CrabsAttack
Controls:
– Arrow keys or WASD to change the level
– Esc to show/hide this screen
– Slash (/) to toggle debug mode
– R to restart current scene

Thanks!

Go make a game!

tdl.is/mg25