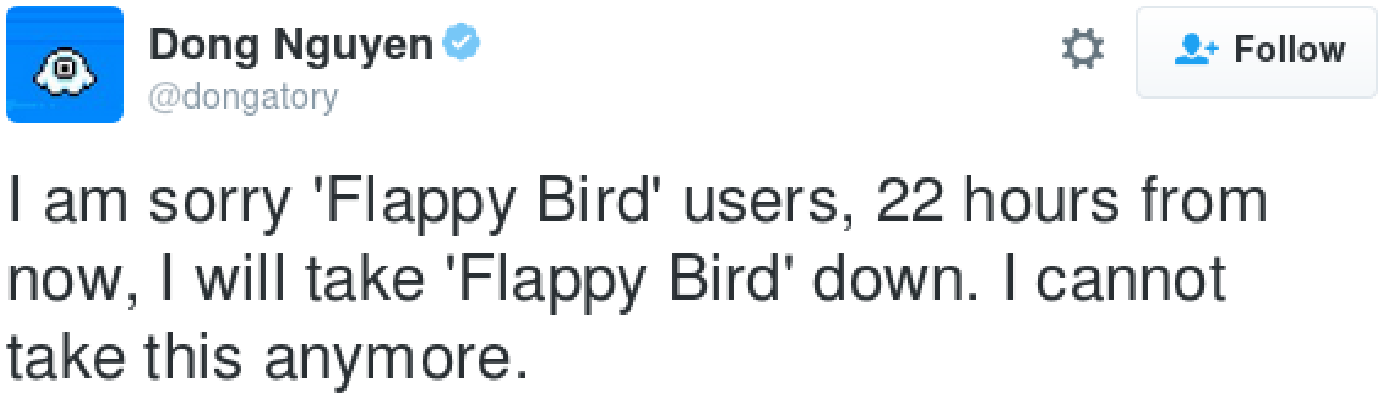
me?

Shabinesh Sivaraj (sab)
@shabinesh

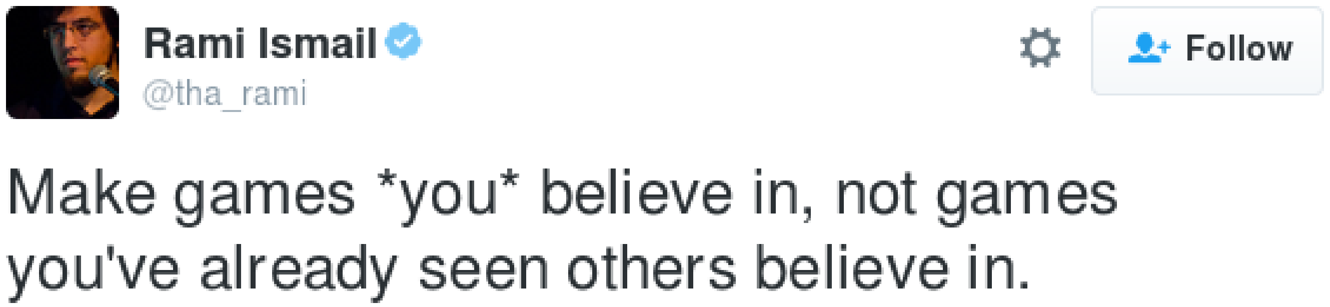
Before we being...

- FUD of game development
- Why Go?
- Run it everywhere.

Game for yourself


Dong Nguyen
@dongatory

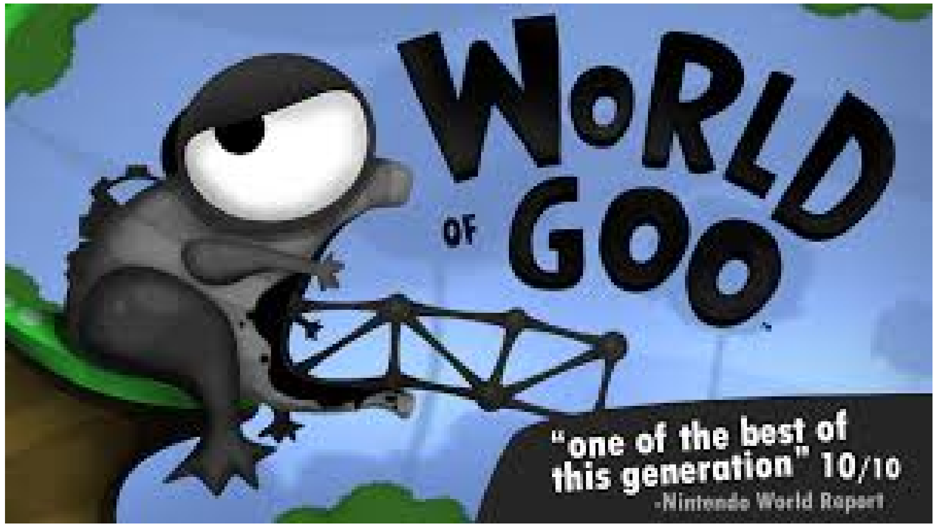
I am sorry 'Flappy Bird' users, 22 hours from now, I will take 'Flappy Bird' down. I cannot take this anymore.


Rami Ismail
@tha_rami

Make games *you* believe in, not games you've already seen others believe in.

Introducing SDL2

- Cross platform library to access to audio, keyboard, mouse, joystick & graphics.
- Officially supported for Windows, Mac, Linux, iOS & Android.
- Library is written in C.
- There are bindings for Python, C#, Ada, Rust etc.,
- And Golang.
- SDL 2.0 is distributed under zlib license.



What are we using SDL 2.0 for today?

- Create windows.
- Handle events from keyboard & mouse.
- Accelerated 2D hardware rendering.
- Timers
- Load images.

Game today...

- Write a game.
- Each stage is one jump closer to game
- Get the Go development environment ready

```
flappy/
    src/
    bin/
    pkg/
```

- Stages are organized into branches in git.
  - 6 Levels to finish in 3 hours
  - `git clone https://github.com/shabinesh/flappy`
  - `export GOPATH=/path/to/dir/`

Installing go, sdl2 & go-sdl2

- https://godoc.org/github.com/veandco/go-sdl2
- Install the SDL libraries
  - `Ubuntu:  apt-get install libsdl2{,-mixer,-image,-ttf}-dev`
  - `Fedora: dnf install SDL2{,_mixer,_image,_ttf}-devel`
  - `Osx: brew install sdl2{,_image,_ttf}`
  - `go get -v github.com/veandco/go-sdl2/sdl{,_image_ttf}`

fundamentals: Window, Events & All

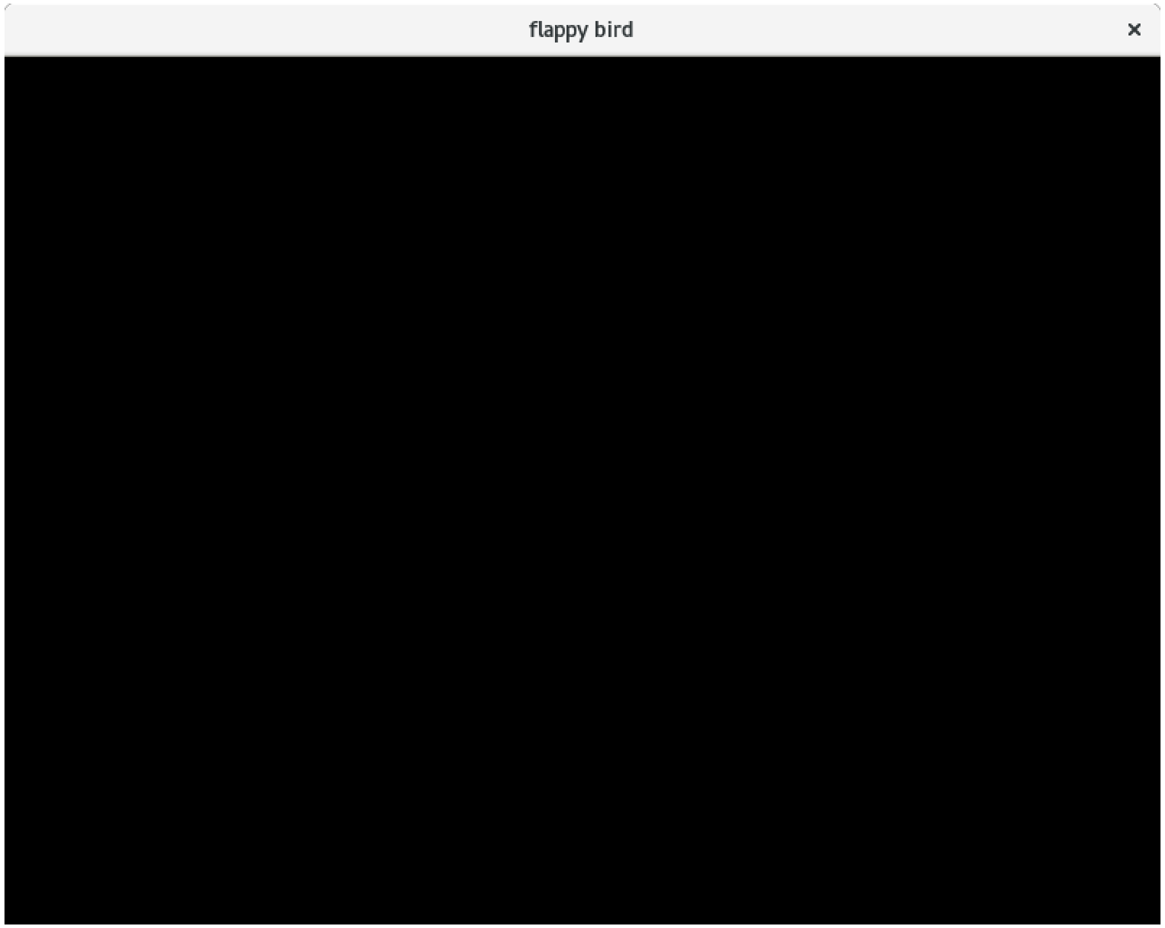
- Pieces to put together
  - Window
  - Renderer
  - Event handling
  - Scene

```
func main() {
   for {
      handleEvents()
      updateStates()
      display()
   }
}
```

Window & Events

Create a window

- `CreateWindow() / CreateWindowAndRenderer()`
  - https://wiki.libsdl.org/SDL_CreateWindow



Create a window cont..

- `sdl.Init(flags)`
  - Initialize all subsystems: Audio, video, joystick etc.
  - First function to be called from SDL.
  - Flags specifies what subsystem to initialize which are ORed, we will use the `INIT_EVERYTHING`.
- `sdl.Delay(ms)`
  - Sdl version of sleep, delays for a specified milliseconds.
- `window.Destroy()`
  - free the memory before quitting.
- `sdl.Quit()`
  - Quit all subsystems.

Continued..

- `PollEvent()/ WaitEvent()`
  - Returns next event from the queue
  - Returns nil if no event.

```
running := true
for running {
    for event := PollEvent(); event != nil; event = PollEvent() {
        // look for events here
    }
}
```

Rendering

SDL2 fundamentals: Rendering

- Software rendering
  - GPU is not used.
  - `sdl.Surface`
  - Image is blitted on screen
  - Stored in RAM

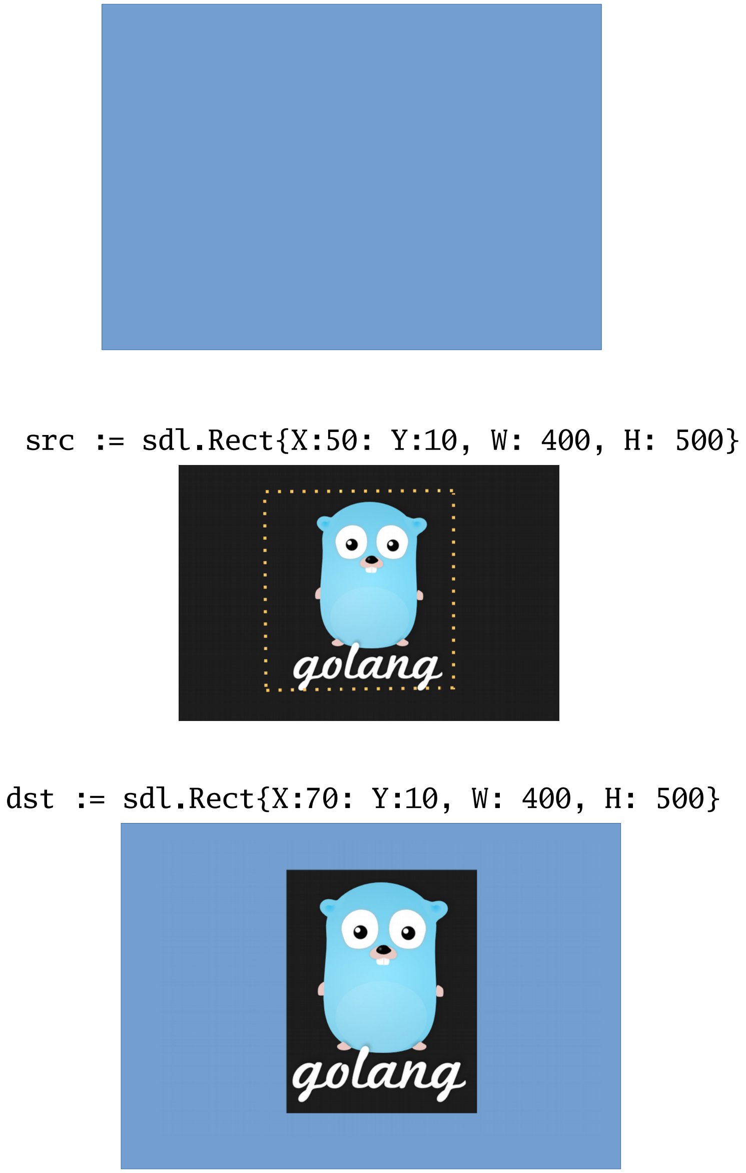
- Hardware rendering
  - GPU used to render images
  - `sdl.Texture`
  - `sdl.Renderer` is used to render
  - Stored in VRAM

drawing on paper ≈ rendering on screen

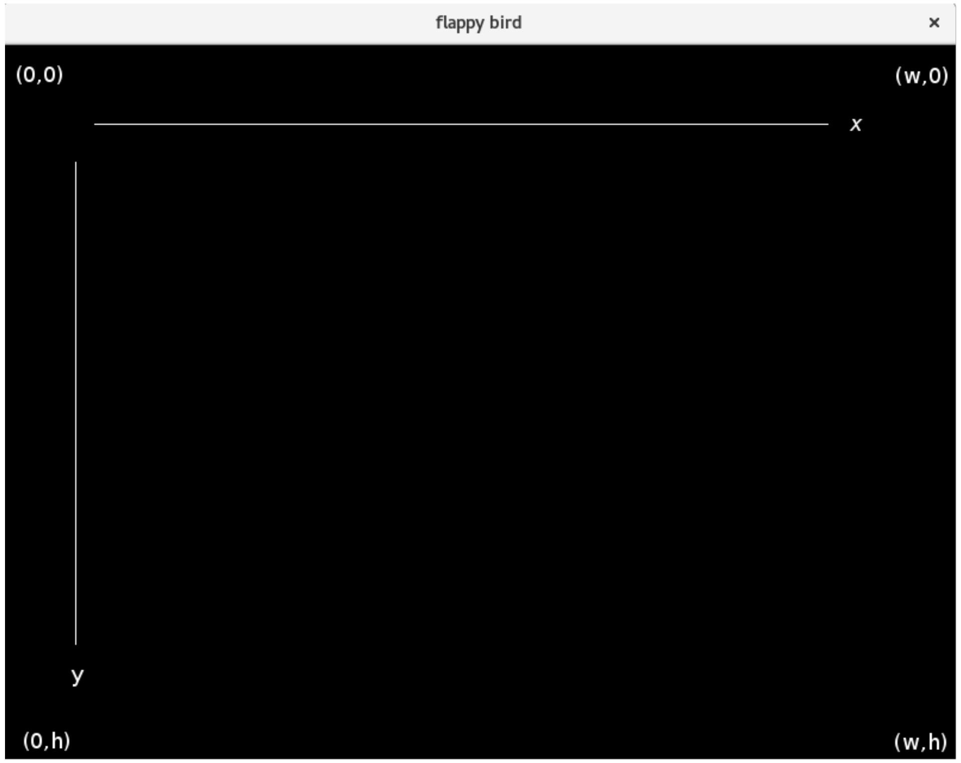
Rendering…

- Load images from files as sdl.Surface
- Create texture from surface
- Copy texture to renderer
- Present the renderer

- `renderer.CreateTextureFromSurface()`
- `renderer.Copy()`
- `renderer.Present()`


src := sdl.Rect{X:50: Y:10, W: 400, H: 500}
golang
dst := sdl.Rect{X:70: Y:10, W: 400, H: 500}
golang

Render title - `git checkout level1`

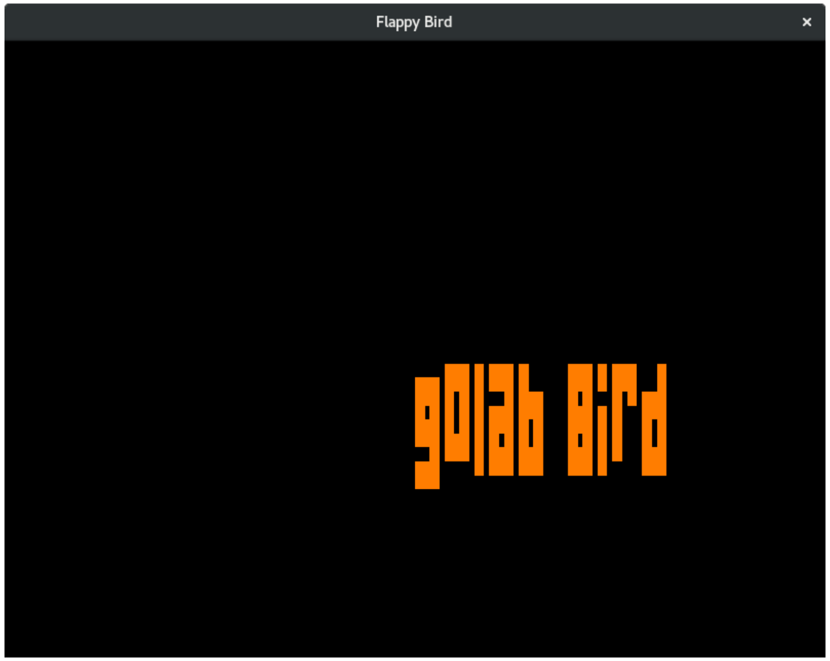


- `sdl_ttf` library used to render TrueType fonts.
- Used to create a texture from the font.
- `sdl_ttf.Init()` is the first function to be called before any other function from this library.

sdl_ttf.

- `sdl_ttf.OpenFont()`
  - open a TrueType font file with point/pixel size
- `font.RenderUTF8_{Solid,Shaded,Blended}()`
  - Returns a surface instance of the title

Got this?



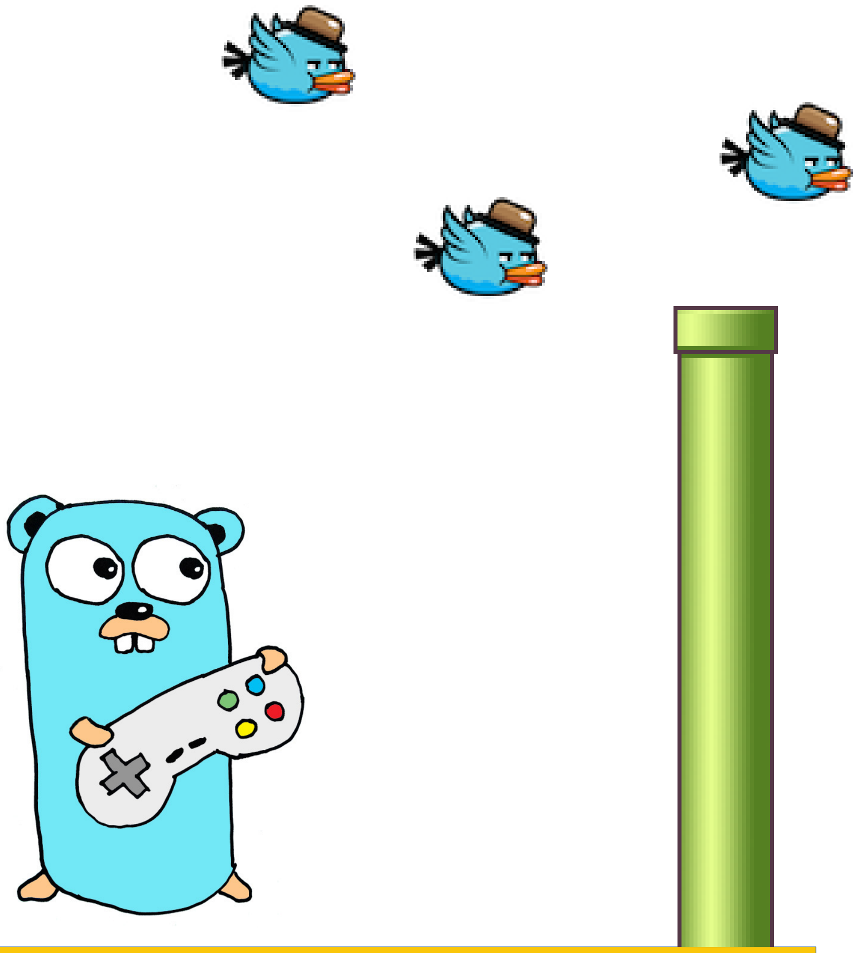
Scenes

- Scene has many scene objects which maintain their states

```
type Scene struct {
    bird Bird
    Cam Camera
}
```

- Creating a scene

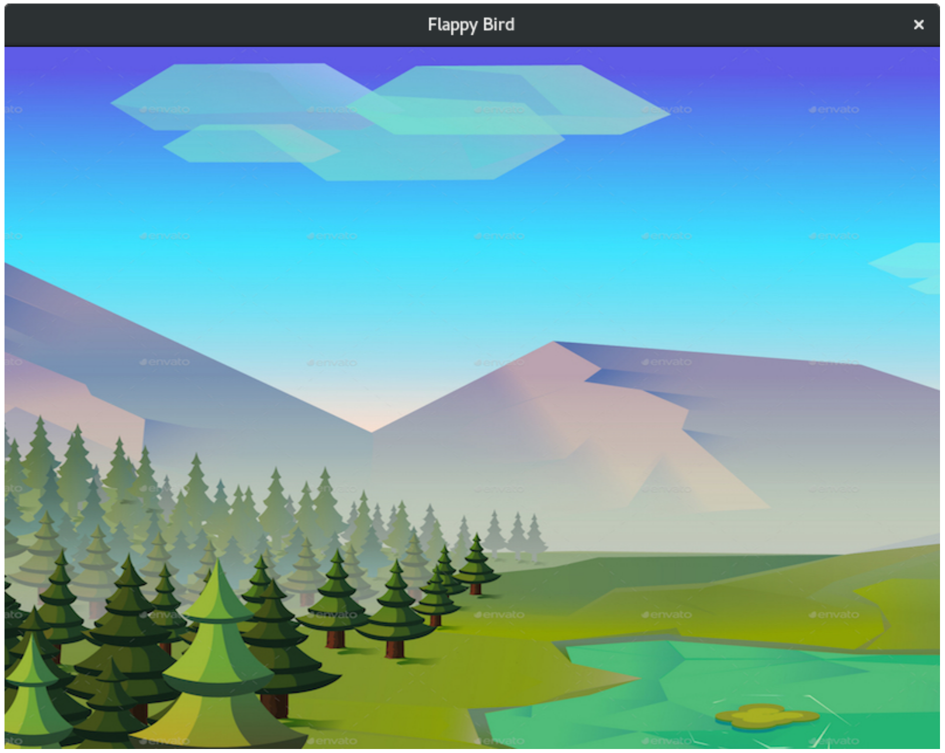
```
func (s Scene) drawFrame() {
    // copy all object to the
    //renderer, draw each object
}
```



Draw Scene

- Draw background scene
- Load image from file to texture
- Blit texture to renderer
- Present renderer

Got this?



Animate

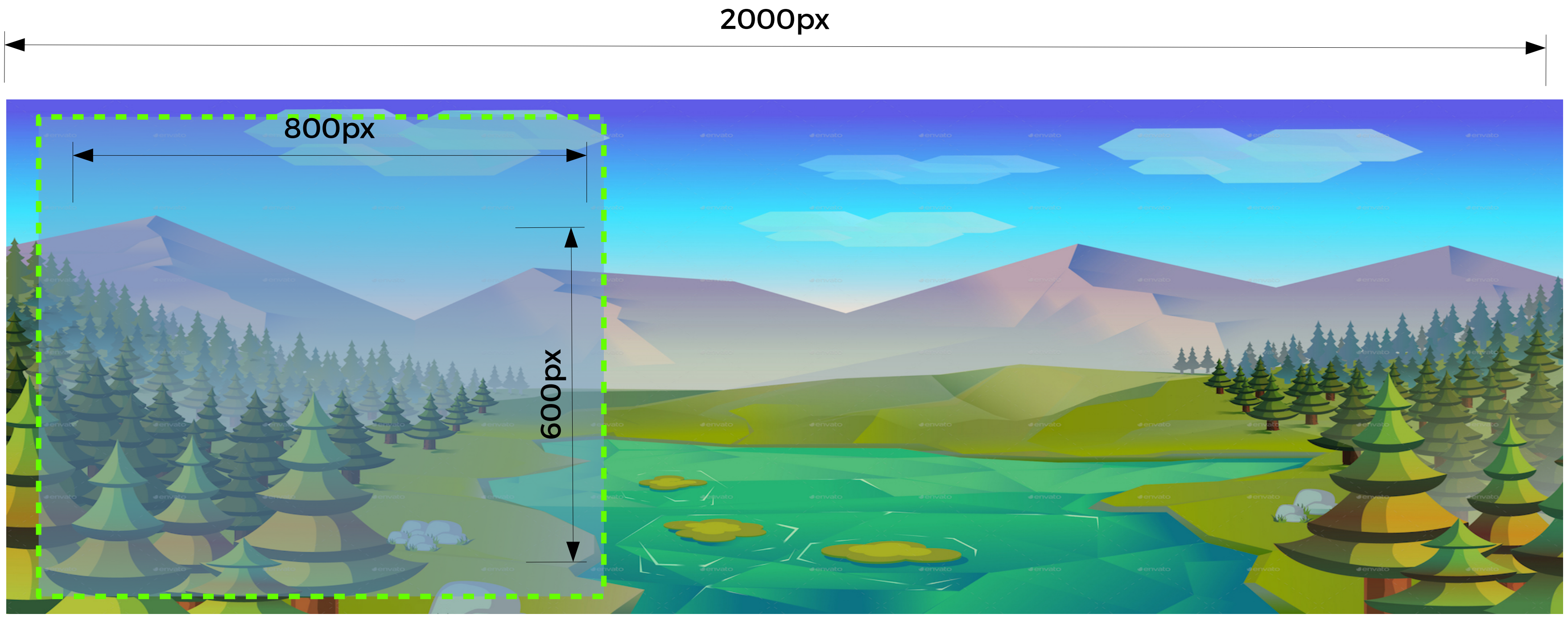
fps

- Render scene to screen `n` times per second to keep things animated.
- Know your fps.

```
go func(fps int) {
    for {
        drawFrame()
        renderer.Present()
        sdl.Delay(1000 / fps)
    }
}(10)
```

Infinite scrolling - `git checkout level2`

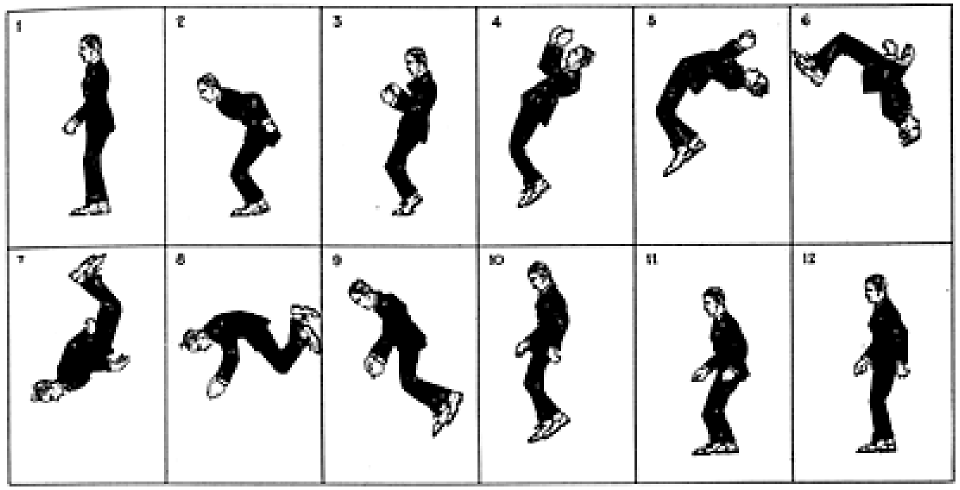
- Camera? Speed?



```
func (c Camera) Draw() {
   c.current += speed
   renderer.Copy(tex, &sdl.Rect{X: x.current, Y: 0, W: windowWidth , H: windowHeight}, nil)
   // reset c.current if c.current + windowWidth > 2000
}
```

Flying bird - `git checkout level3`

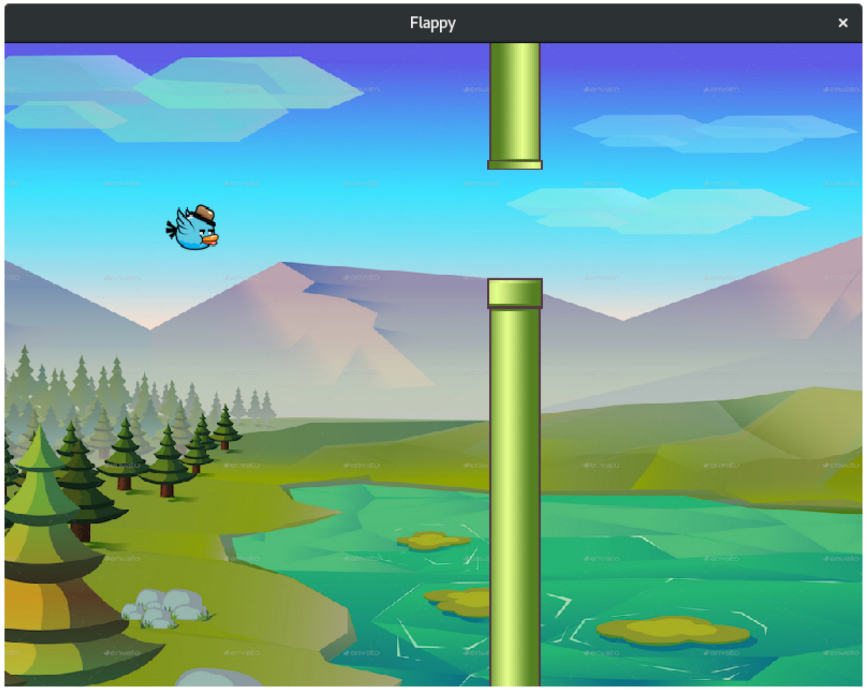
- Fly the birds.
- The birds flap

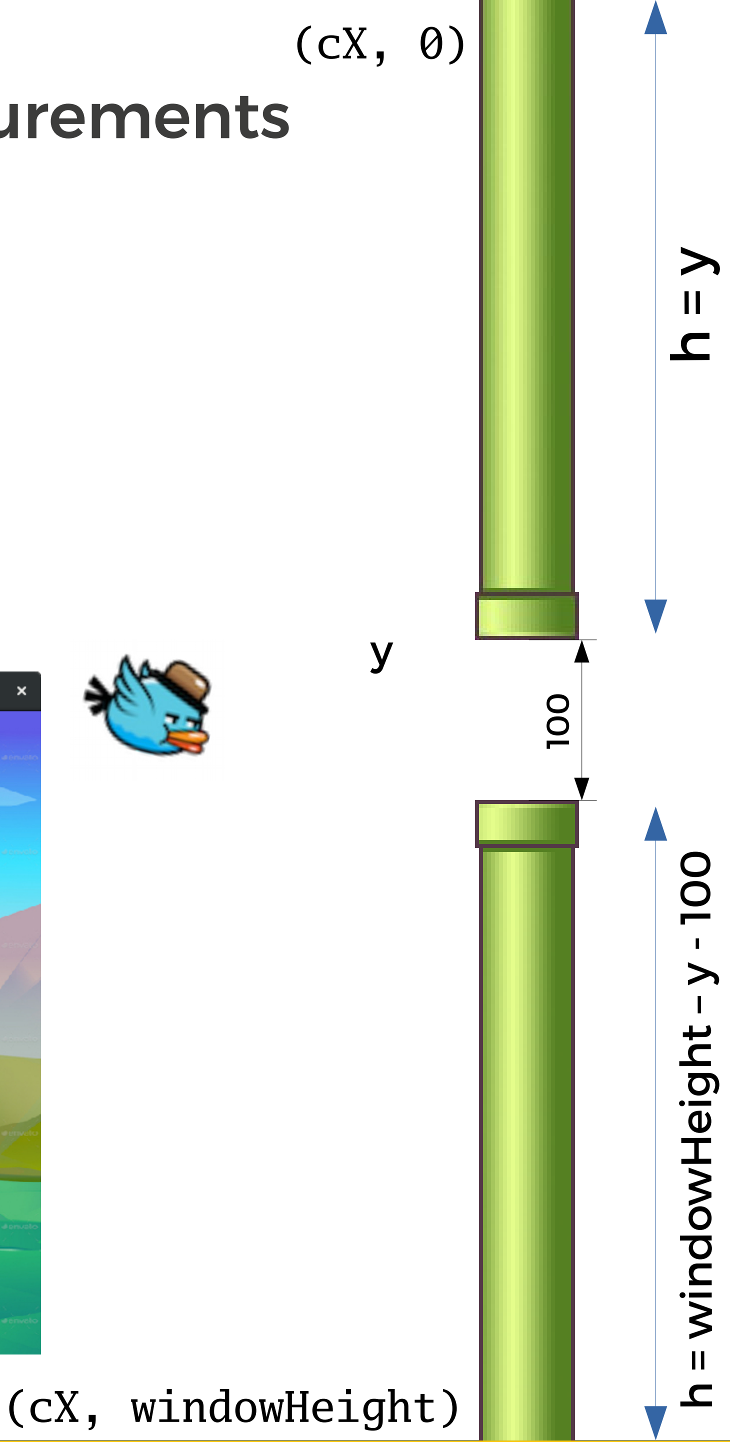


Some more pepper..

- Add gravity for the bird to fall
- Lift up the bird if the key event occur
- Let gravity take its course.
- Limit them flying away from boundaries

Pipe Measurements





Pipes

- Randomly generate pipes

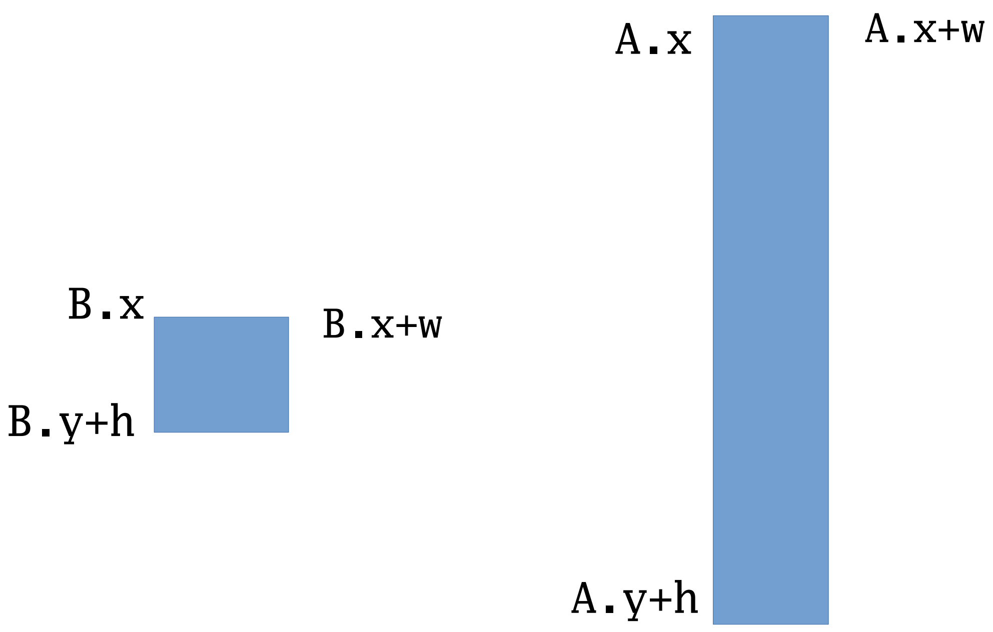
```
type PipePair struct {
    tex         *sdl.Texture
    pipes       []Pipe
    currentPipe int
    speed       int32
}

type Pipe struct {
    cX          int32
    topPipeH    int32
    bottomPipeH int32
}
```

```
func (p *PipePair) draw() {
    /* draw current top & bottom
    pipe */
    renderer.Copy(topPipe)
    renderer.CopyEx(bottomPipe)
}
```

Collision Detection

- Make sure the bird dies when it hits the pipe.



Questions?

Grazie!

@shabinesh
sivarajshabinesh@gmail.com