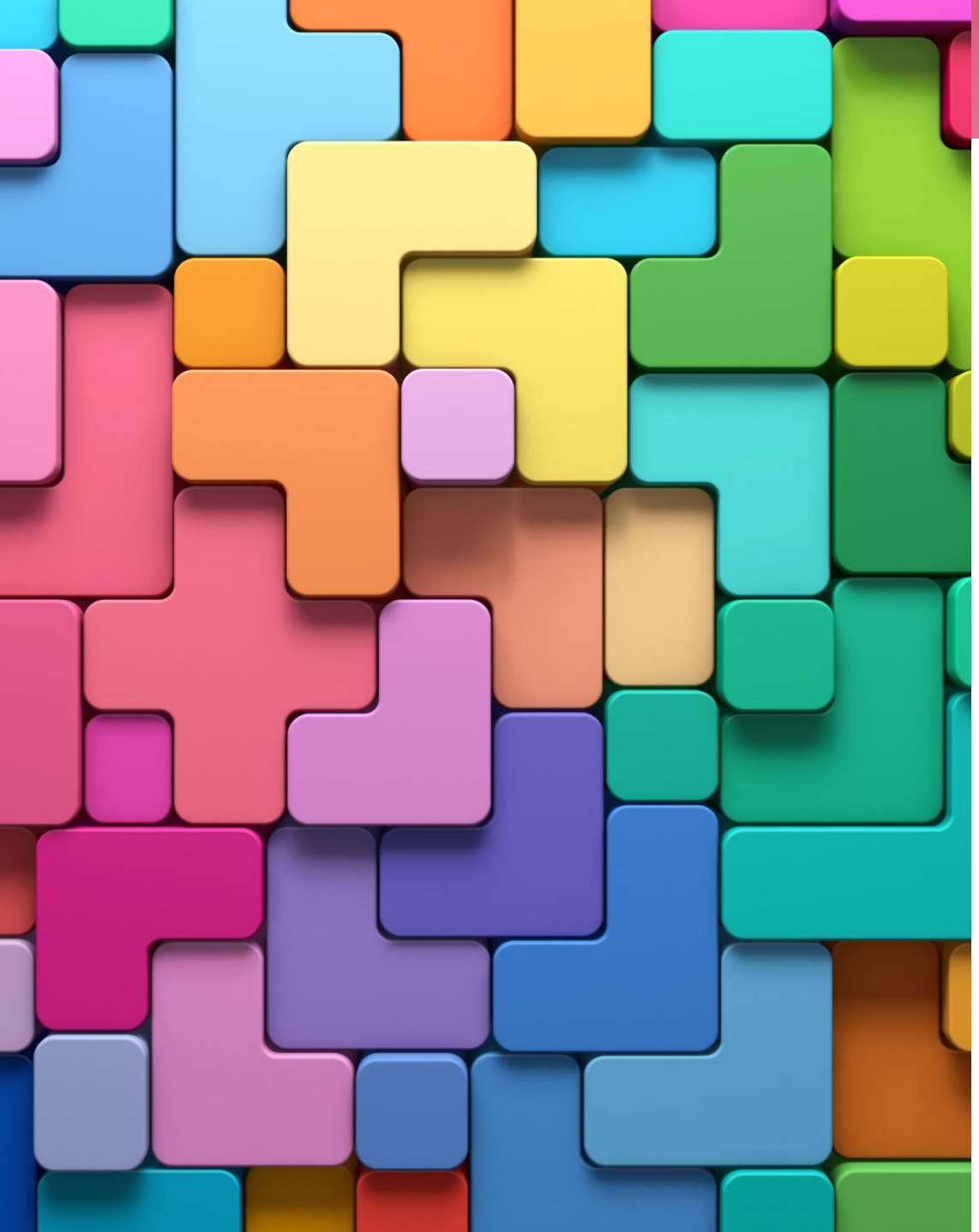
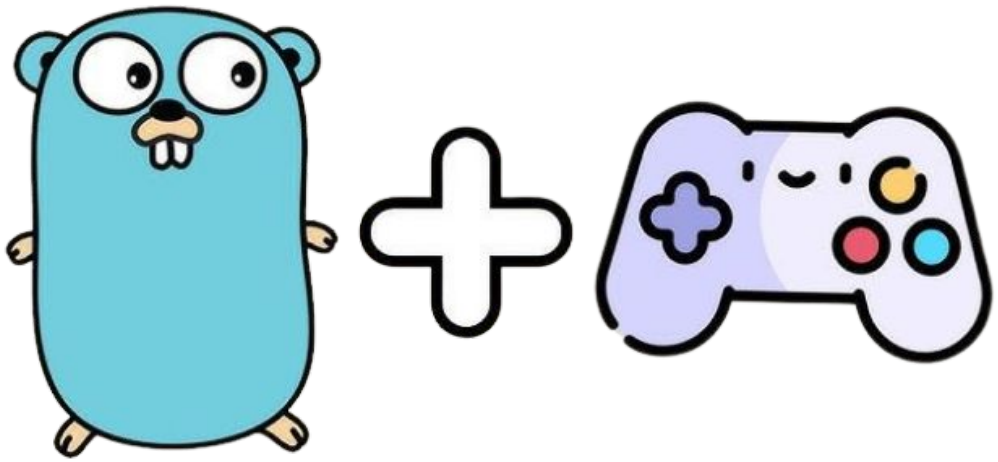


Construindo um game clássico da década de 70 em Go com suporte a multiplayer



Carlos Henrique

- Formado em Ciências da Computação
- Pós Graduado em Gestão de Projetos
- 18 anos inserindo bugs em produção
- 6 anos em Golang
- Atualmente Backend Developer na Coinbase
- Contribuidor em projetos Open Source

Renê Cardozo

- Especializado em transformar cafeína em código de qualidade duvidosa
- 5 anos desenvolvendo em Go
- Senior Software Engineer - TRACTIAN
- Formado em Ciência da Computação - USP

@reneepc

Agenda

- Um pouco de história
- Qual engine utilizar?
- Desafios
- Como detectar colisão?
- Criando uma IA simples
- Máquina de estados
- Comunicação client-server
- Goroutines, channels, context
- Demo!

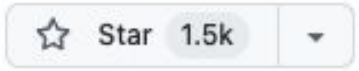
Um pouco de história

- Pong é um jogo eletrônico de esporte de arcade com temática de tênis de mesa, com gráficos bidimensionais, desenvolvido pela Atari e lançado originalmente em 1972. Foi
- 6502 Assembly foi a linguagem escolhida pelos desenvolvedores da Atari e foi originalmente lançada em 1947.
- Pong foi o primeiro jogo comercialmente bem-sucedido e ajudou a estabelecer a indústria de jogos eletrônicos junto com o Magnavox Odyssey.

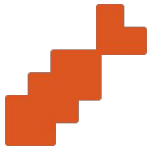
Um pouco de história

Qual engine utilizar?

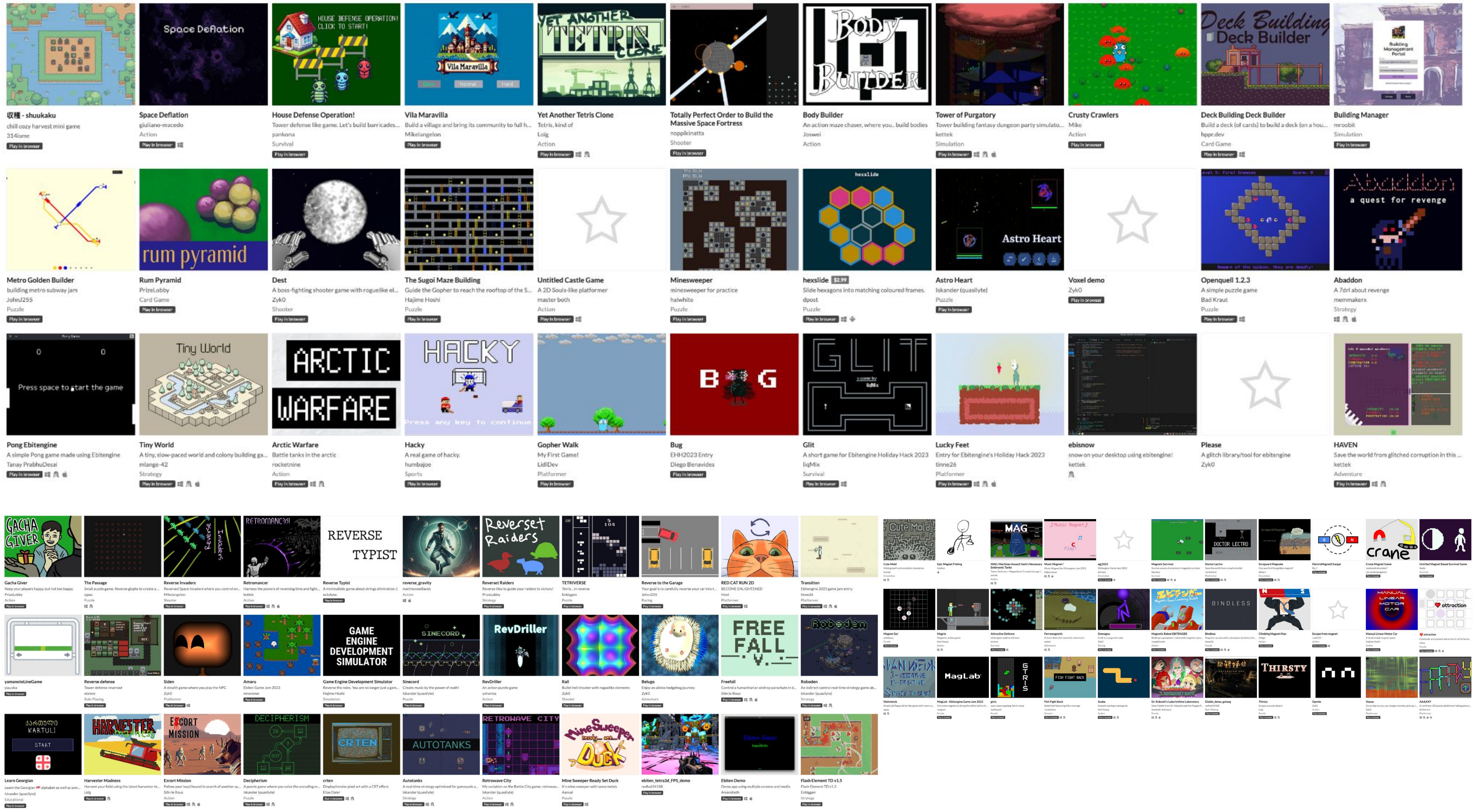
+ **gen2brain/raylib-go** 

+ **oakmound/oak** 

+ **hajimehoshi/ebiten** 



- **Ebitengine** é uma biblioteca open source para criação de jogos 2D. Permite criar jogos de forma rápida e simples utilizando sua API.
- Tudo é considerado uma imagem.
- Suporte a Gráficos 2D, Mouse, Teclado, Gamepad, Touch, Áudio.
- Cross-Platform: Windows, macOS, Linux, Web, Mobile e até para Nintendo Switch.



收穫 - shuukaku
chill cozy harvest mini game
314isme
Space Deflation
giuliano-macedo
Action
House Defense Operation!
Tower defense like game. Let's build barricades...
pankona
Survival
Vila Maravilla
Build a village and bring its community to full h...
Mikelangelon
Yet Another Tetris Clone
Tetris, kind of
Loïg
Action
Totally Perfect Order to Build the Massive Space Fortress
noppikinatta
Shooter
Body Builder
An action maze chaser, where you.. build bodies
Joswei
Action
Tower of Purgatory
Tower building fantasy dungeon party simulato...
kettek
Simulation
Crusty Crawlers
Mike
Action
Deck Building Deck Builder
Build a deck (of cards) to build a deck (on a hou...
hppr.dev
Card Game
Building Manager
mroobit
Simulation
Metro Golden Builder
building metro subway jam
JohnJ255
Puzzle
Rum Pyramid
PrizeLobby
Card Game
Dest
A boss-fighting shooter game with roguelike el...
Zyk0
Shooter
The Sugoi Maze Building
Guide the Gopher to reach the rooftop of the S...
Hajime Hoshi
Puzzle
Untitled Castle Game
A 2D Souls-like platformer
master both
Action
Minesweeper
minesweeper for practice
halwhite
Puzzle
hexslide $2.99
Slide hexagons into matching coloured frames.
dpost
Puzzle
Astro Heart
Iskander (quasilyte)
Puzzle
Voxel demo
Zyk0
Openquell 1.2.3
A simple puzzle game
Bad Kraut
Puzzle
Abaddon
A 7drl about revenge
memmakerx
Strategy
Pong Ebitengine
A simple Pong game made using Ebitengine
Tanay PrabhuDesai
Tiny World
A tiny, slow-paced world and colony building ga...
mlange-42
Strategy
Arctic Warfare
Battle tanks in the arctic
rocketnine
Action
Hacky
A real game of hacky.
humbajoe
Sports
Gopher Walk
My First Game!
LidIDev
Platformer
Bug
EHH2023 Entry
Diego Benavides
Glit
A short game for Ebitengine Holiday Hack 2023
liqMix
Survival
Lucky Feet
Entry for Ebitengine's Holiday Hack 2023
tinne26
Platformer
ebisnow
snow on your desktop using ebitengine!
kettek
Please
A glitch library/tool for ebitengine
Zyk0
HAVEN
Save the world from glitched corruption in this ...
kettek
Adventure
Gacha Giver
The Passage
Reverse Invaders
Retromancer
Reverse Typist
reverse_gravity
Reverset Raiders
TETRIVERSE
Reverse to the Garage
RED CAT RUN 2D
Transition
yamanoteLineGame
Reverse defense
Siden
Amaru
Game Engine Development Simulator
Sinecord
RevDriller
Rail
Belugo
Freefall
Roboden
Learn Georgian
Harvester Madness
Ezcort Mission
Decipherism
crten
Autotanks
Retrowave City
Mine Sweeper Ready Set Duck
ebiten_tetra3d_FPS_demo
Ebiten Demo
Flash Element TD v1.5

+ Lista de jogos https://itch.io/c/2030581/made-with-ebitengine

Ebitengine

```go
package main

import (
    "github.com/hajimehoshi/ebiten/v2"
    "github.com/hajimehoshi/ebiten/v2/ebitenutil"
)

type Game struct{}

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

func (g *Game) Draw(screen *ebiten.Image) {
    ebitenutil.DebugPrint(screen, "Hello, World!")
}

func (g *Game) Layout(outsideWidth, outsideHeight int) (screenWidth, screenHeight int) {
    return 320, 240
}

func main() {
    ebiten.SetWindowSize(640, 480)
    ebiten.SetWindowTitle("Hello, World!")

    if err := ebiten.RunGame(&Game{}); err != nil {
        // handle error
    }
}
```

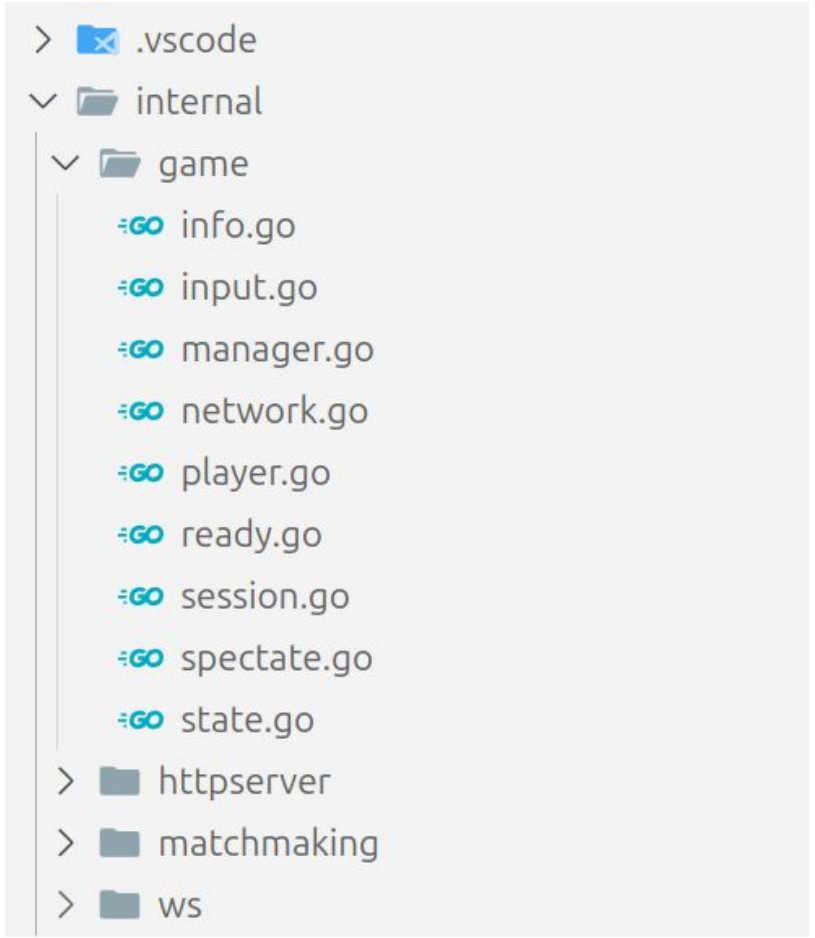
- 60 FPS por padrão
- Draw
- Update
- Layout

Desafios

+ Aprender a mecânica e as funções da Ebitengine
+ Criar um mecanismo de colisão
+ Exibir estatísticas tais como FPS, Ping, …
+ Interagir com o usuário
+ Adicionar suporte ao modo multiplayer

Estrutura do Projeto

Lógica da engine compartilhada



Como detectar colisão?

```go
// MaxX returns the maximum X value of the rectangle.
func (r Rect) MaxX() float64 {
    return r.X + r.Width
}

// MaxY returns the maximum Y value of the rectangle.
func (r Rect) MaxY() float64 {
    return r.Y + r.Height
}

// Intersects returns true if the rectangle intersects with another rectangle.
func (r Rect) Intersects(other Rect) bool {
    return r.X <= other.MaxX() &&
        other.X <= r.MaxX() &&
        r.Y <= other.MaxY() &&
        other.Y <= r.MaxY()
}
```

Como detectar colisão?

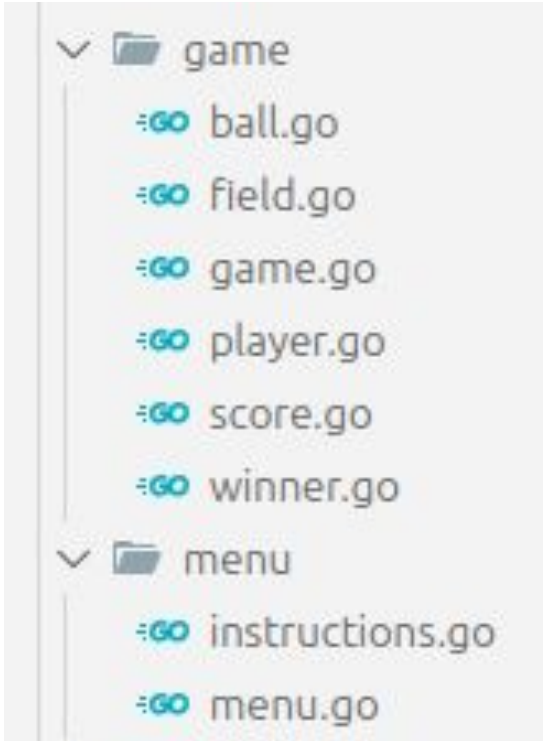
```go
func (p *player) Bounds() geometry.Rect {
    return geometry.Rect{
        X:      p.position.X,
        Y:      p.position.Y,
        Width:  p.bouncerWidth,
        Height: p.bouncerHeight,
    }
}
```

```go
func (b *Local) checkPaddleBounce(p1Bounds, p2Bounds geometry.Rect) {
    if b.ball.Bounds().Intersects(p1Bounds) {
        b.position.X = p1Bounds.X + p1Bounds.Width + width
        b.bounceOffPaddle()
    }

    if b.ball.Bounds().Intersects(p2Bounds) {
        b.position.X = p2Bounds.X - b.width
        b.bounceOffPaddle()
    }
}
```

Criando uma IA simples

```go
func GuessBallPosition(ballY, enemyY, enemyHeight, screenHeight, fieldBorderWidth float64) float64 {
    delta := float64(rand.IntN(15))

    if enemyY < ballY-delta {
        enemyY += cpuSpeed // Move down
    }

    if enemyY > ballY+delta {
        enemyY -= cpuSpeed // Move up
    }

    return keepInBounds(enemyY, enemyHeight, screenHeight, fieldBorderWidth)
}
```

Lógica de Menu

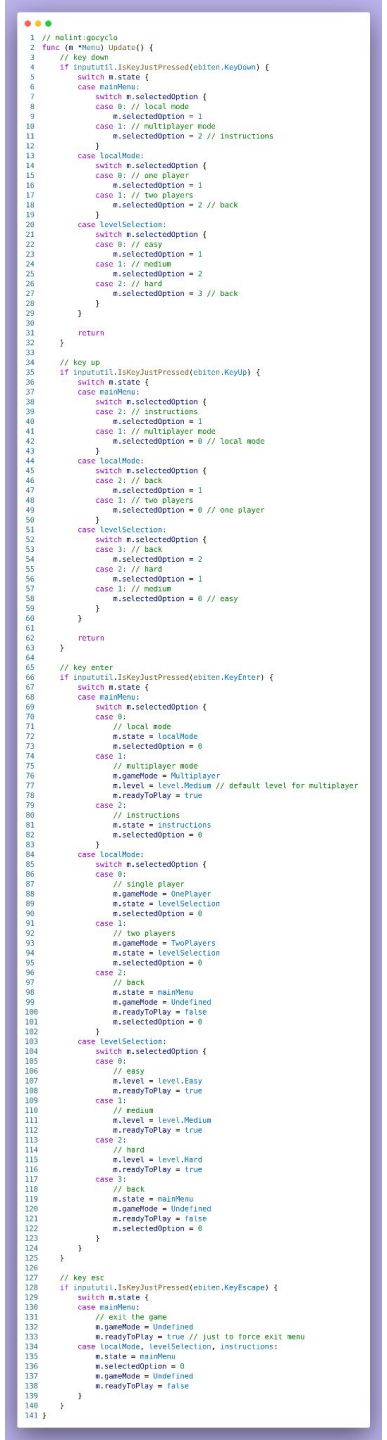



```go
// nolint:gocyclo
func (m *Menu) Update() {
    // key down
    if inpututil.IsKeyJustPressed(ebiten.KeyDown) {
        switch m.state {
        case mainMenu:
            switch m.selectedOption {
            case 0: // local mode
                m.selectedOption = 1
            case 1: // multiplayer mode
                m.selectedOption = 2 // instructions
            }
        case localMode:
            switch m.selectedOption {
            case 0: // one player
                m.selectedOption = 1
            case 1: // two players
                m.selectedOption = 2 // back
            }
        case levelSelection:
            switch m.selectedOption {
            case 0: // easy
                m.selectedOption = 1
            case 1: // medium
                m.selectedOption = 2
            case 2: // hard
                m.selectedOption = 3 // back
            }
        }

        return
    }

    // ...
```

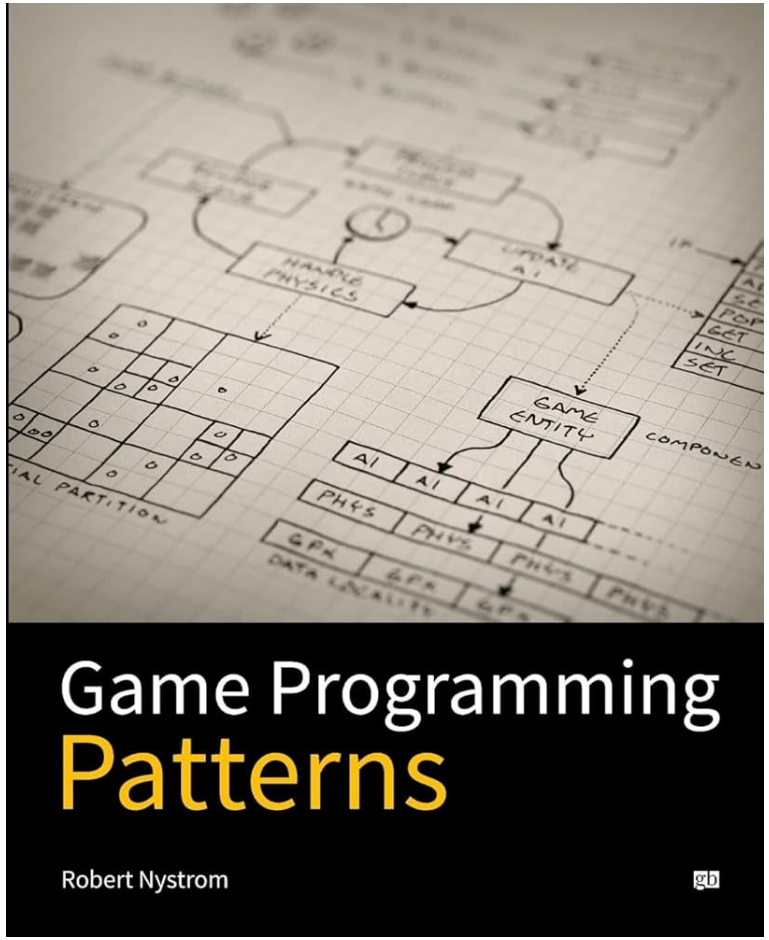
Spaghetti

- Apenas 3 menus já tornam a lógica de seleção enorme
- Alto acoplamento entre componentes

```go
// nolint:gocyclo
func (m *Menu) Update() {
    // key down
    if inpututil.IsKeyJustPressed(ebiten.KeyDown) {
        switch m.state {
        case mainMenu:
            switch m.selectedOption {
            case 0: // local mode
                m.selectedOption = 1
            case 1: // multiplayer mode
                m.selectedOption = 2 // instructions
            }
        case localMode:
            switch m.selectedOption {
            case 0: // one player
                m.selectedOption = 1
            case 1: // two players
                m.selectedOption = 2 // back
            }
        case levelSelection:
            switch m.selectedOption {
            case 0: // easy
                m.selectedOption = 1
            case 1: // medium
                m.selectedOption = 2
            case 2: // hard
                m.selectedOption = 3 // back
            }
        }

        return
    }

    // ...
```

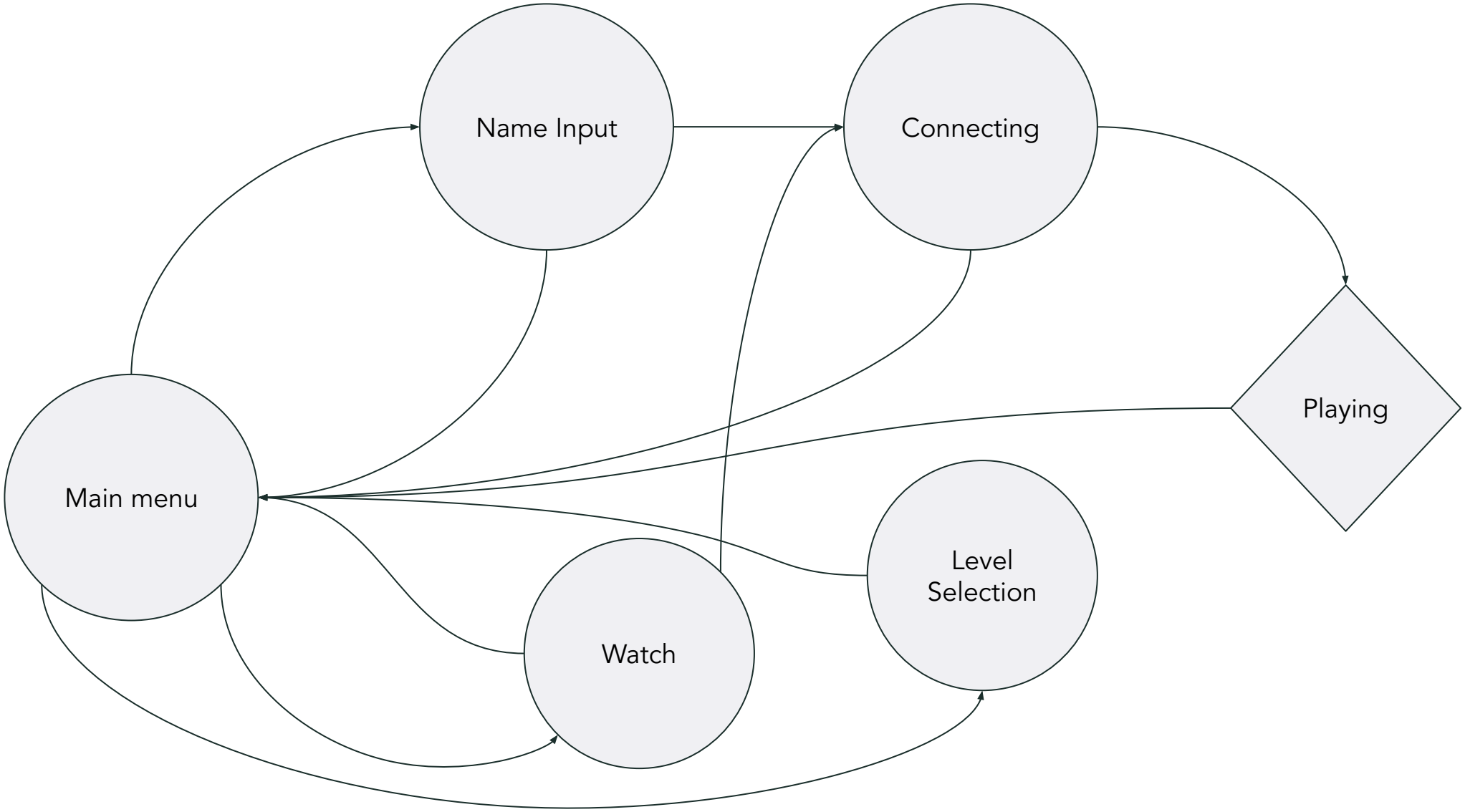


State Pattern

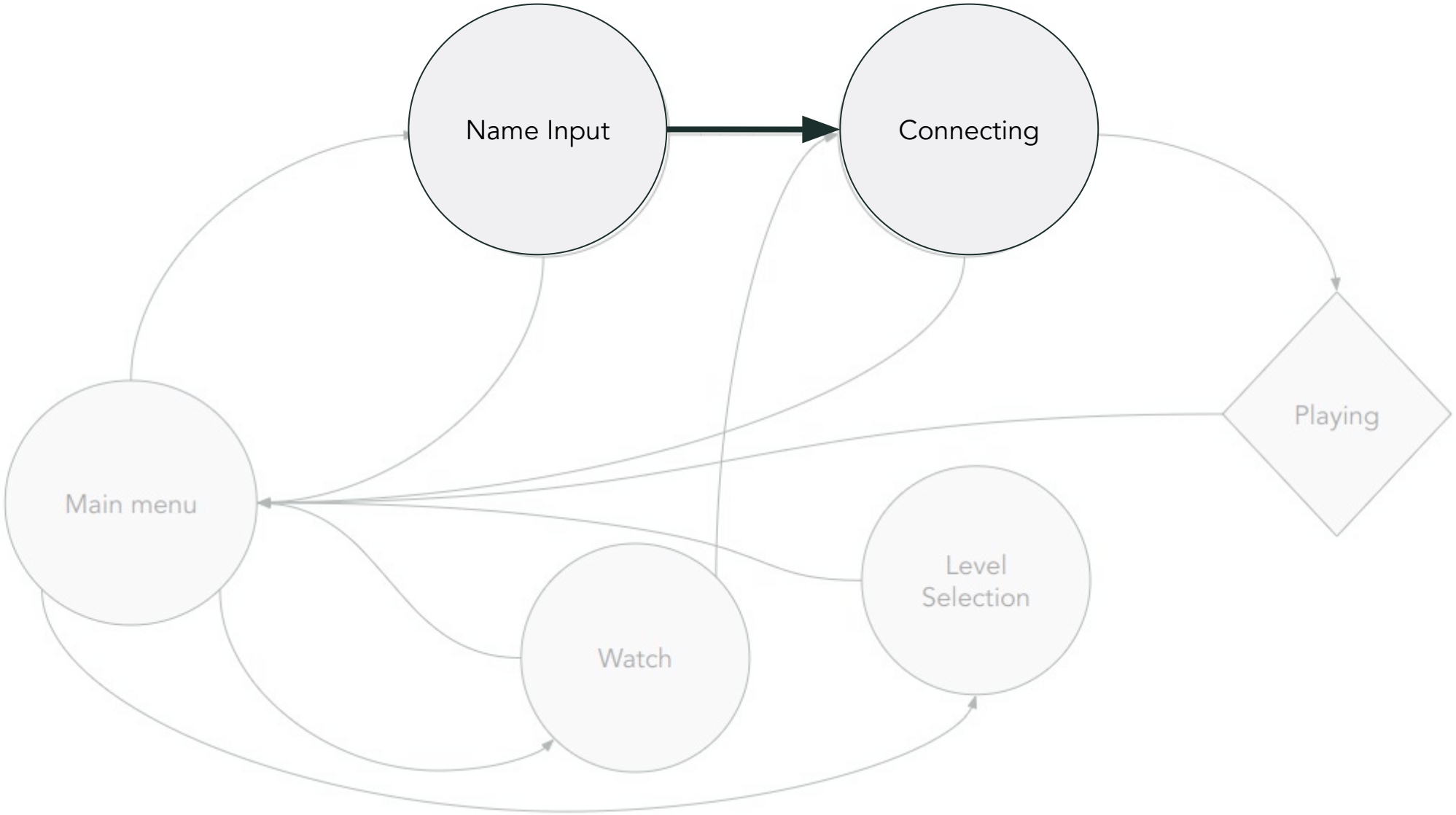


Robert Nystrom

Menu State


Name Input
Connecting
Playing
Main menu
Level
Selection
Watch

Menu State


Name Input
Connecting
Playing
Main menu
Level Selection
Watch

State Pattern

```go
// New creates a new game menu.
func New(font *font.Font, screenWidth, screenHeight int) *Menu {
    menu := &Menu{
        font:         font,
        gameMode:     Undefined,
        screenWidth:  screenWidth,
        screenHeight: screenHeight,
        states:       make(map[string]state),
    }

    menu.ChangeState(newMainMenuState(menu))

    return menu
}

// ChangeState changes the current state of the menu.
func (m *Menu) ChangeState(state state) {
    if st, ok := m.states[state.String()]; ok {
        m.currentState = st
        return
    }

    m.states[state.String()] = state
    m.currentState = state
}
```

State Pattern

```go
// New creates a new game menu.
func New(font *font.Font, screenWidth, screenHeight int) *Menu {
    menu := &Menu{
        font:         font,
        gameMode:     Undefined,
        screenWidth:  screenWidth,
        screenHeight: screenHeight,
        states:       make(map[string]state),
    }

    menu.ChangeState(newMainMenuState(menu))

    return menu
}

// ChangeState changes the current state of the menu.
func (m *Menu) ChangeState(state state) {
    if st, ok := m.states[state.String()]; ok {
        m.currentState = st
        return
    }

    m.states[state.String()] = state
    m.currentState = state
}
```

```go
// Update updates the menu.
func (m *Menu) Update() {
    m.currentState.Update()
}

// Draw draws the menu.
func (m *Menu) Draw(screen *ebiten.Image) {
    m.currentState.Draw(screen)
}
```

Base menu

```go
type baseState struct {
    selectedOption int
    options        []string
    menu           *Menu
}

func (s *baseState) navigateOptions(maxOptions int) {
    if inpututil.IsKeyJustPressed(ebiten.KeyDown) {
        if s.selectedOption < maxOptions-1 {
            s.selectedOption++
        }
    }

    if inpututil.IsKeyJustPressed(ebiten.KeyUp) {
        if s.selectedOption > 0 {
            s.selectedOption--
        }
    }
}
```

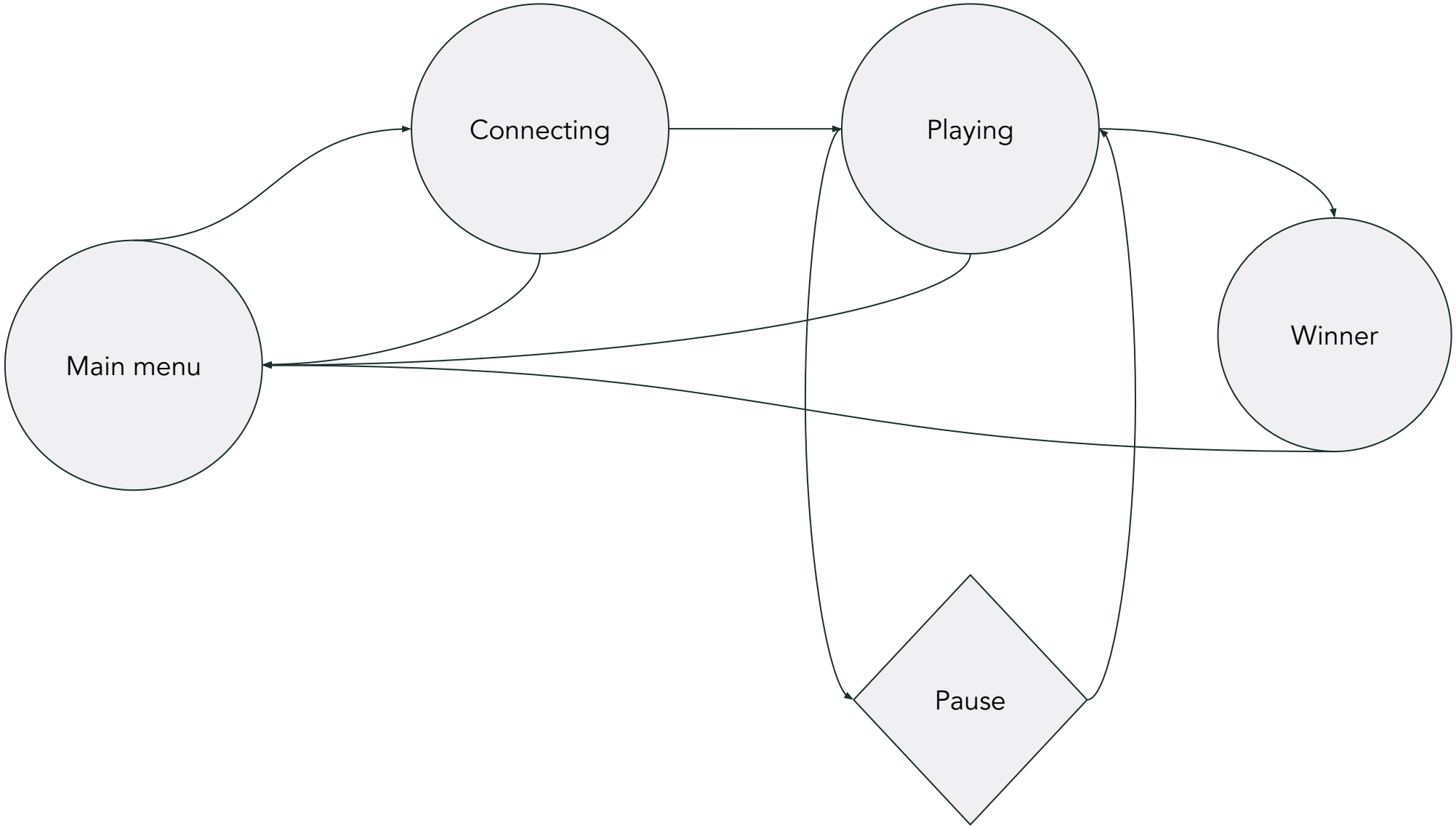
Base menu

```
type mainMenuState struct {
    *baseState
}

func newMainMenuState(menu *Menu) *mainMenuState {
    return &mainMenuState{
        baseState: &baseState{
            menu: menu,
            options: []string{
                localModeStr,
                multiplayerStr,
                spectateStr,
                instructionsStr},
        },
    }
}
```

Desacoplamento

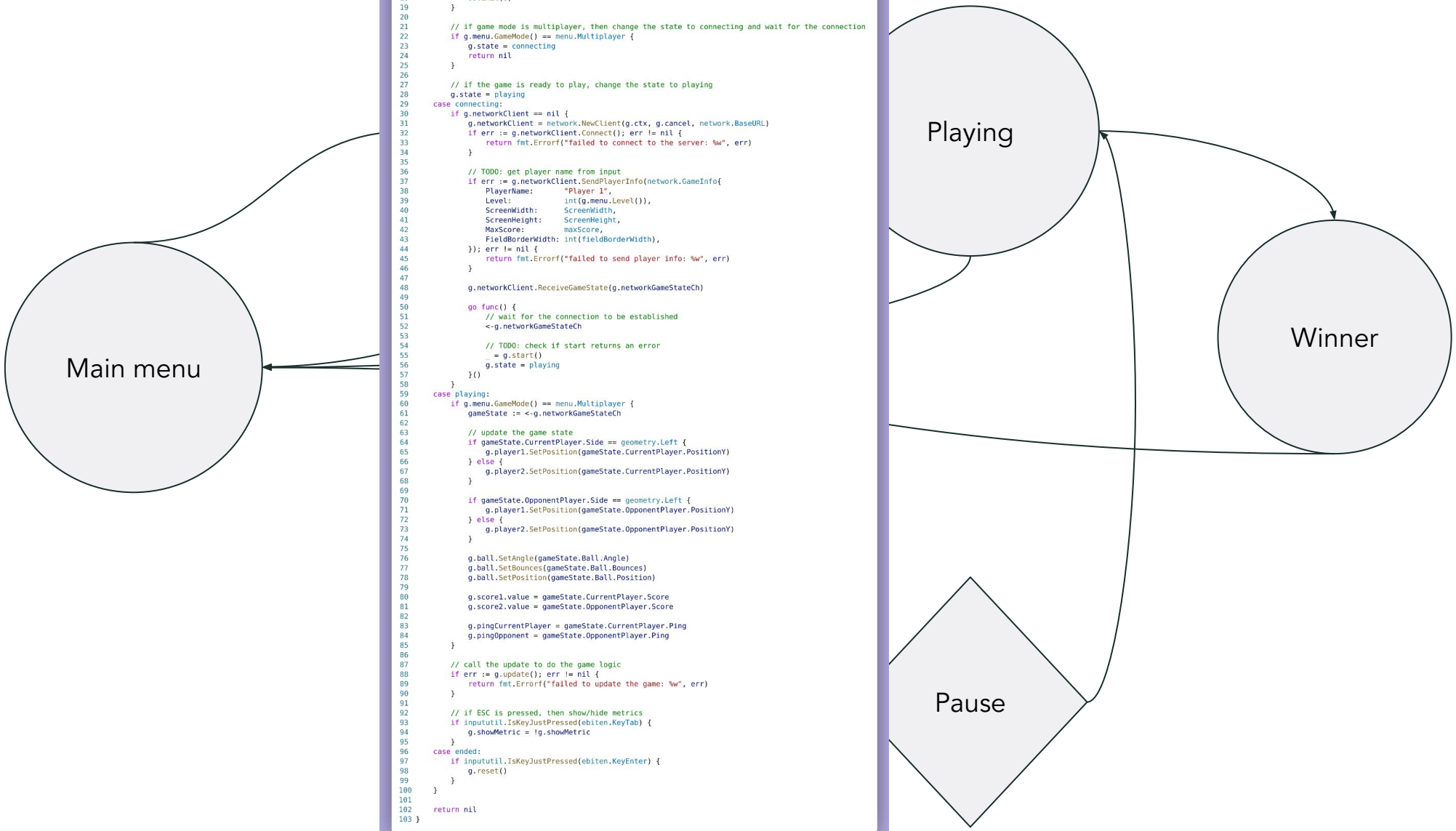
```go
func (s *mainMenuState) Update() {
    s.navigateOptions(len(s.options))

    if inpututil.IsKeyJustPressed(ebiten.KeyEnter) {
        switch s.selectedOption {
        case 0:
            s.menu.ChangeState(newLocalModeState(s.menu))
        case 1:
            s.menu.ChangeState(newInputNameState(s.menu))
        case 2:
            s.menu.ChangeState(newSpectateSessionsState(s.menu))
        case 3:
            s.menu.ChangeState(newInstructionsState(s.menu))
        }
    }

    // return to quit the game
    if inpututil.IsKeyJustPressed(ebiten.KeyEscape) {
        s.menu.gameMode = Undefined
        s.menu.readyToPlay = true
    }
}
```

Game State

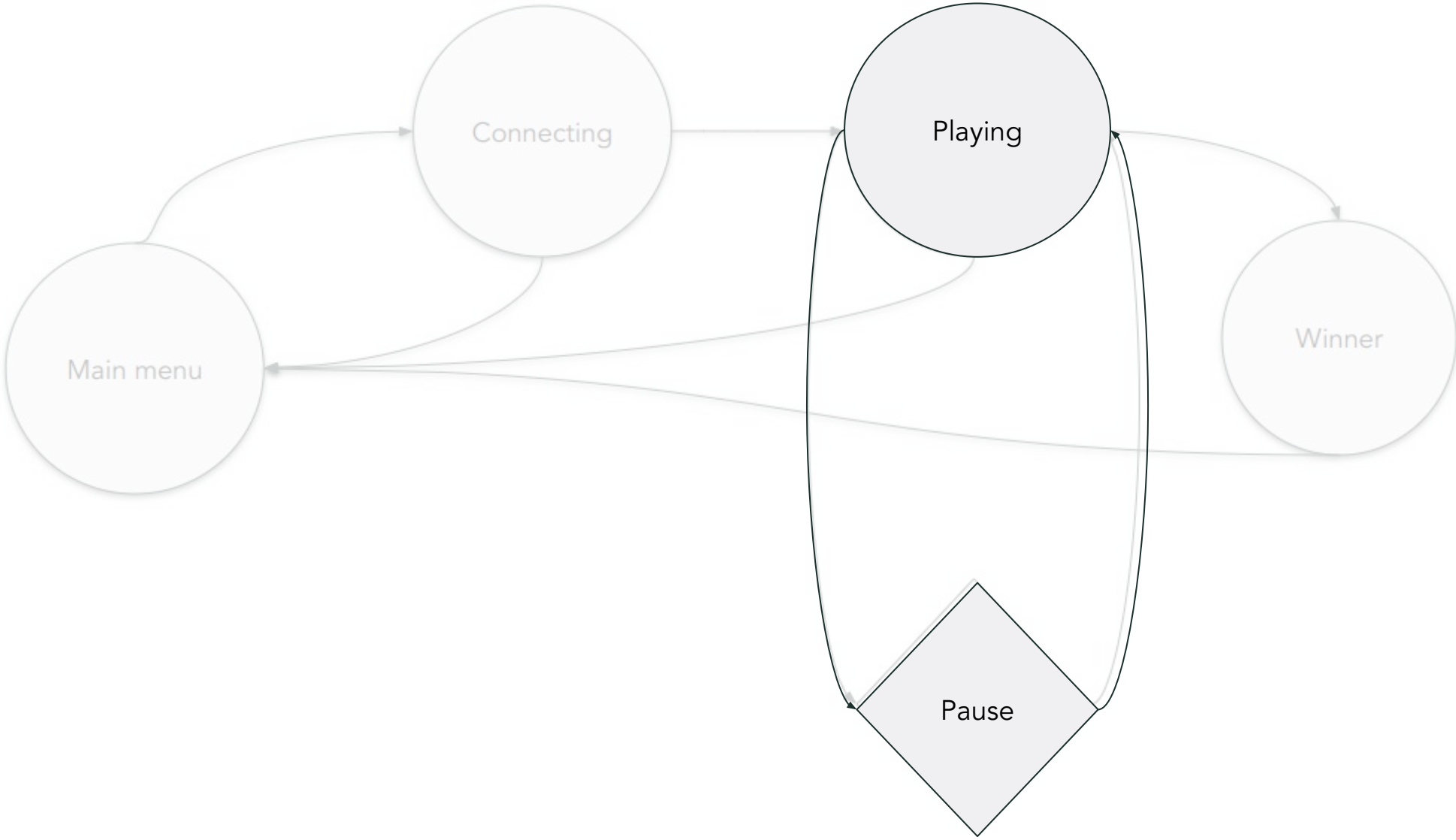


Game State

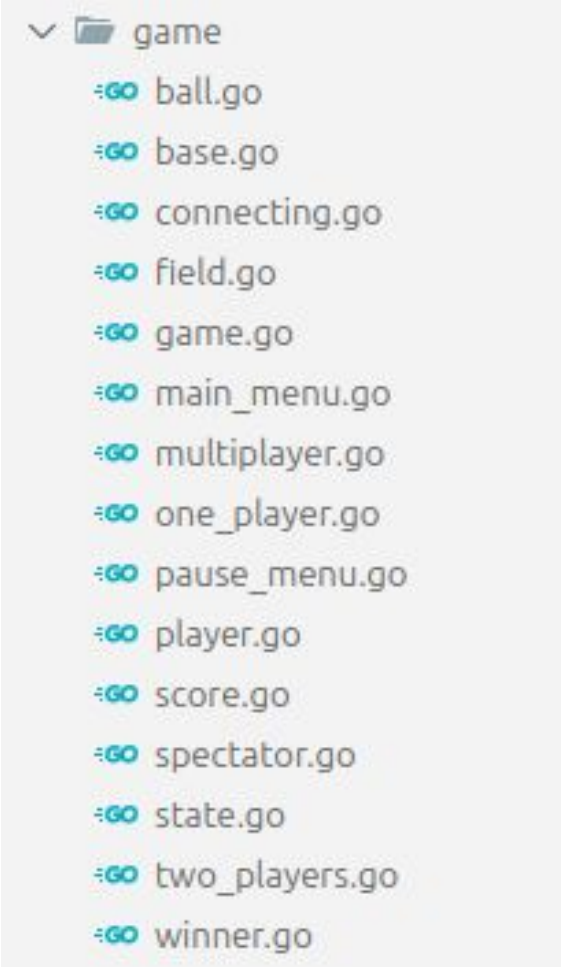
```go
// Update updates the game.
func (g *Game) Update() error {
    switch g.state {
    case notReady:
        // if the game is not ready to play, udpate the menu
        if !g.menu.IsReadyToPlay() {
            g.menu.Update()

            return nil
        }

        // it means to exit the game
        if g.menu.GameMode() == menu.Undefined {
            if runtime.GOOS != "js" {
                return ebiten.Termination
            }

            os.Exit(0)
        }

        // if game mode is multiplayer, then change the state to connecting and wait for the connection
        if g.menu.GameMode() == menu.Multiplayer {
            g.state = connecting
            return nil
        }

        // if the game is ready to play, change the state to playing
        g.state = playing
    case connecting:
        if g.networkClient == nil {
            g.networkClient = network.NewClient(g.ctx, g.cancel, network.BaseURL)
            if err := g.networkClient.Connect(); err != nil {
                return fmt.Errorf("failed to connect to the server: %w", err)
            }

            // TODO: get player name from input
            if err := g.networkClient.SendPlayerInfo(network.GameInfo{
                PlayerName:     "Player 1",
                Level:          int(g.menu.Level()),
                ScreenWidth:    ScreenWidth,
                ScreenHeight:   ScreenHeight,
                MaxScore:       maxScore,
                FieldBorderWidth: int(fieldBorderWidth),
            }); err != nil {
                return fmt.Errorf("failed to send player info: %w", err)
            }

            g.networkClient.ReceiveGameState(g.networkGameStateCh)

            go func() {
                // wait for the connection to be established
                <-g.networkGameStateCh

                // TODO: check if start returns an error
                _ = g.start()
                g.state = playing
            }()
        }
    case playing:
        if g.menu.GameMode() == menu.Multiplayer {
            gameState := <-g.networkGameStateCh

            // update the game state
            if gameState.CurrentPlayer.Side == geometry.Left {
                g.player1.SetPosition(gameState.CurrentPlayer.PositionY)
            } else {
                g.player2.SetPosition(gameState.CurrentPlayer.PositionY)
            }

            if gameState.OpponentPlayer.Side == geometry.Left {
                g.player1.SetPosition(gameState.OpponentPlayer.PositionY)
            } else {
                g.player2.SetPosition(gameState.OpponentPlayer.PositionY)
            }

            g.ball.SetAngle(gameState.Ball.Angle)
            g.ball.SetBounces(gameState.Ball.Bounces)
            g.ball.SetPosition(gameState.Ball.Position)

            g.score1.value = gameState.CurrentPlayer.Score
            g.score2.value = gameState.OpponentPlayer.Score

            g.pingCurrentPlayer = gameState.CurrentPlayer.Ping
            g.pingOpponent = gameState.OpponentPlayer.Ping
        }

        // call the update to do the game logic
        if err := g.update(); err != nil {
            return fmt.Errorf("failed to update the game: %w", err)
        }

        // if ESC is pressed, then show/hide metrics
        if inpututil.IsKeyJustPressed(ebiten.KeyTab) {
            g.showMetric = !g.showMetric
        }
    case ended:
        if inpututil.IsKeyJustPressed(ebiten.KeyEnter) {
            g.reset()
        }
    }

    return nil
}
```

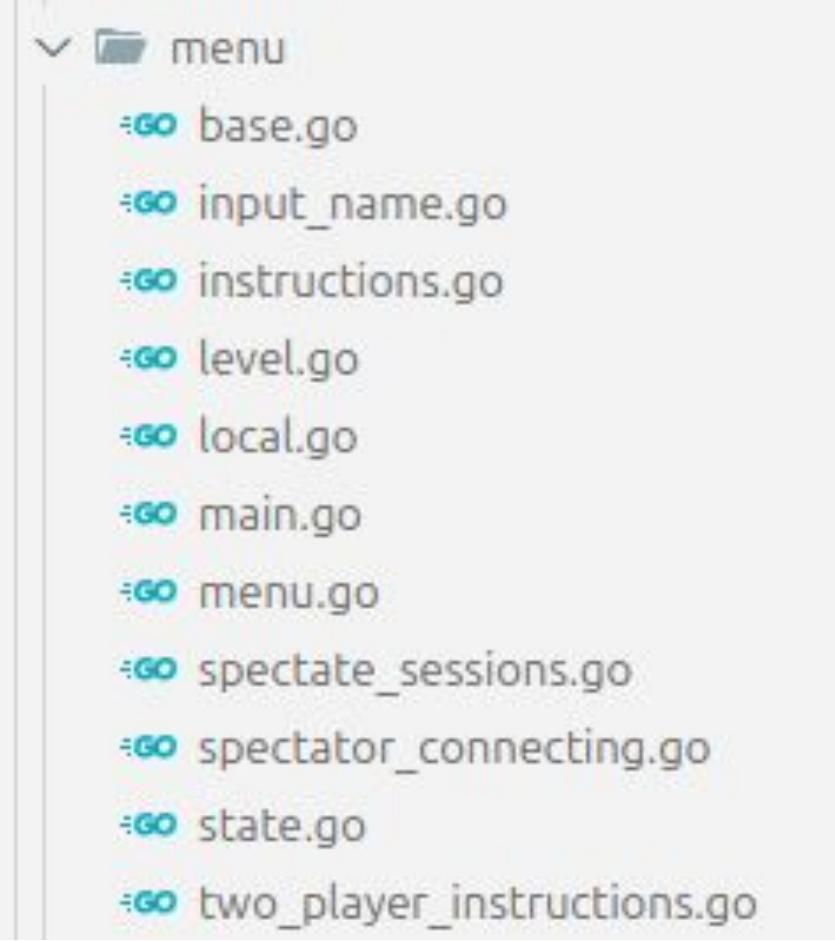


Game State

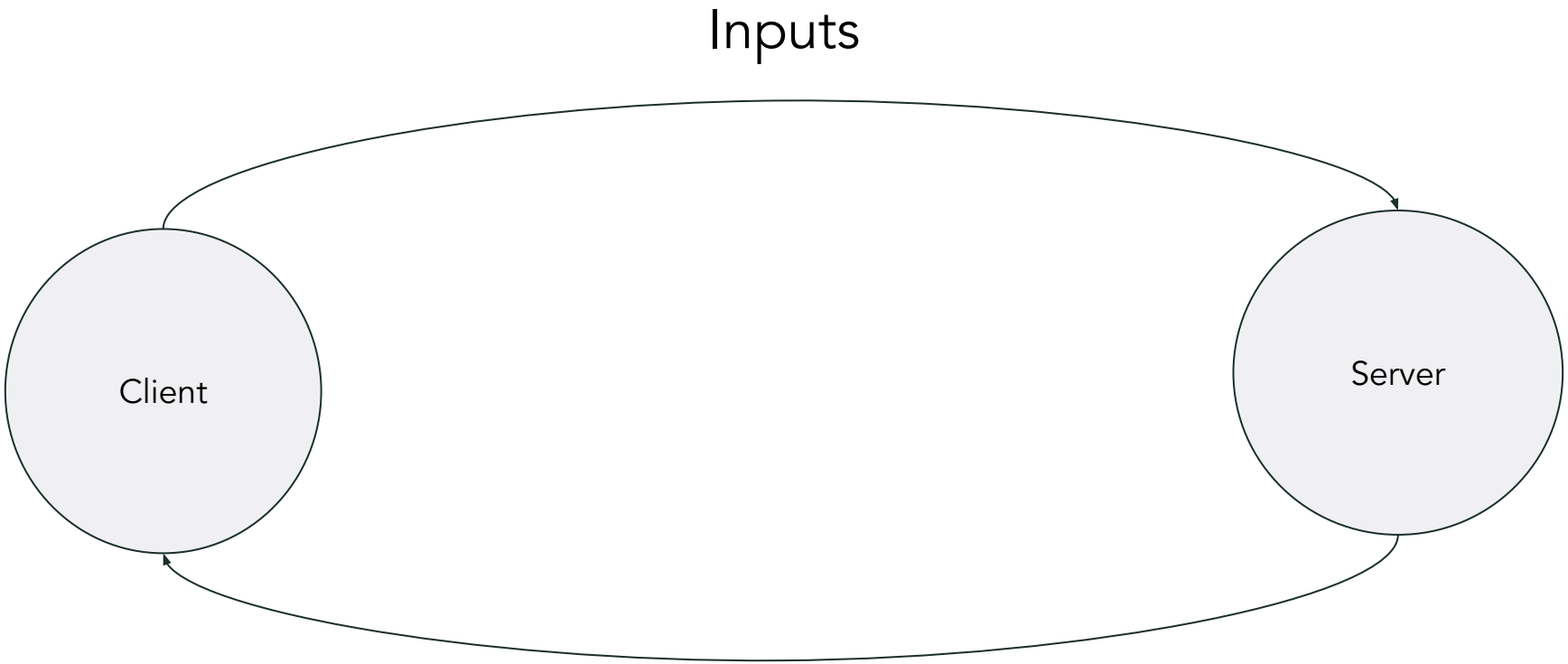


Desacoplamento



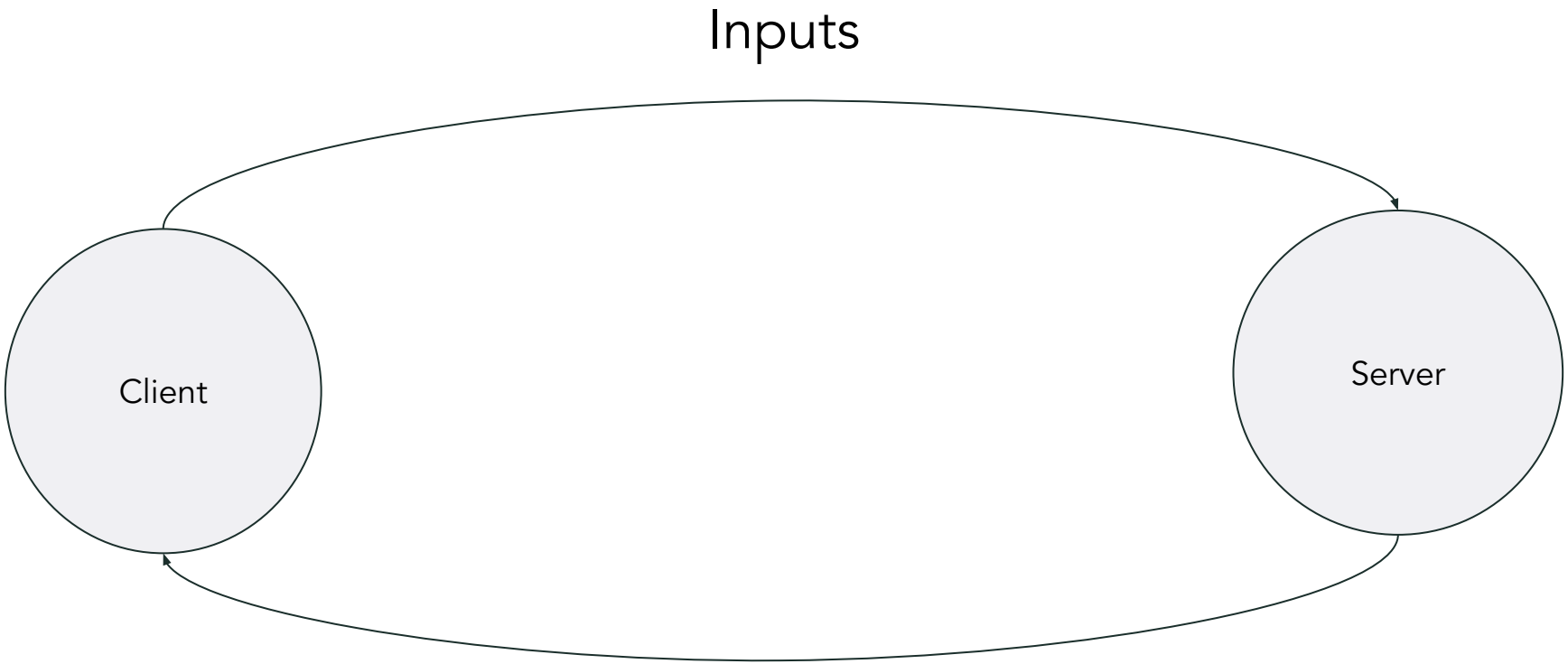


Multiplayer



- Overhead mínimo para envio de mensagens

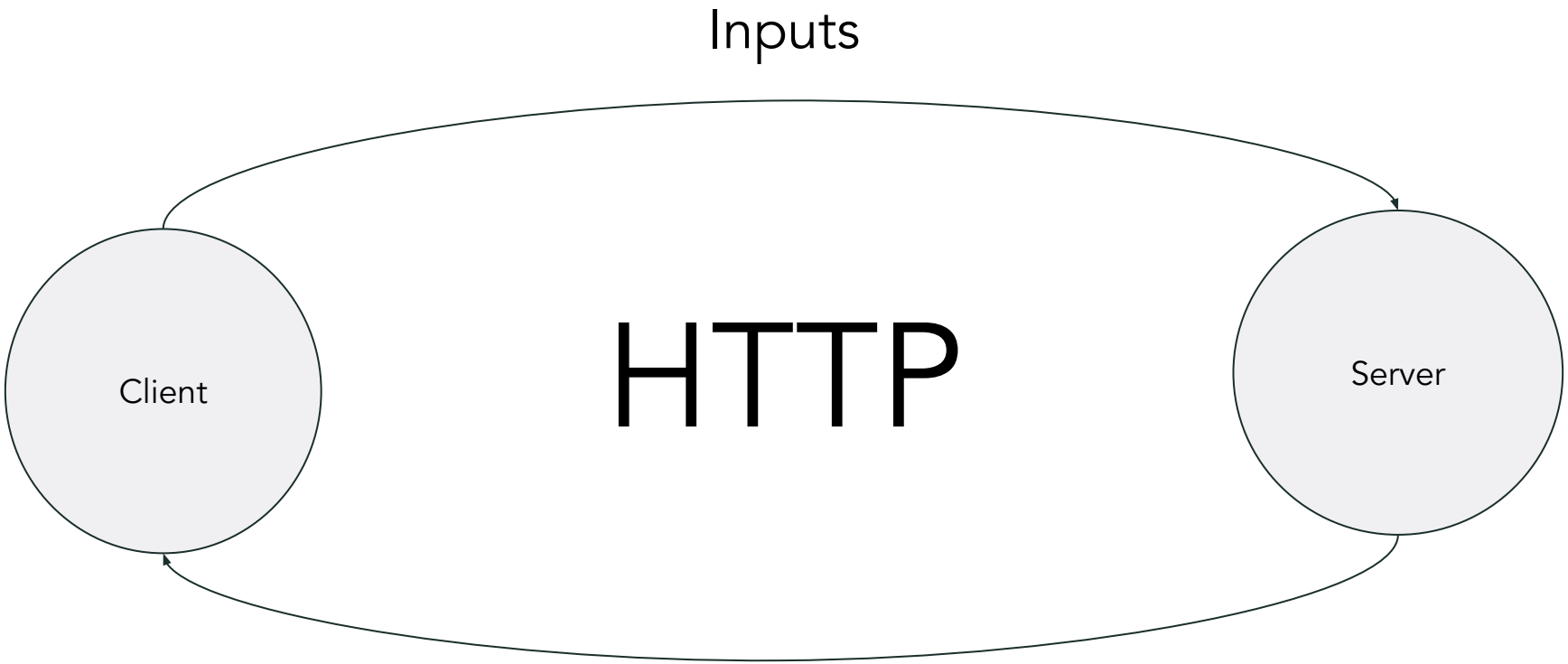
Multiplayer



- Overhead mínimo para envio de mensagens

- Full duplex
- Real-time

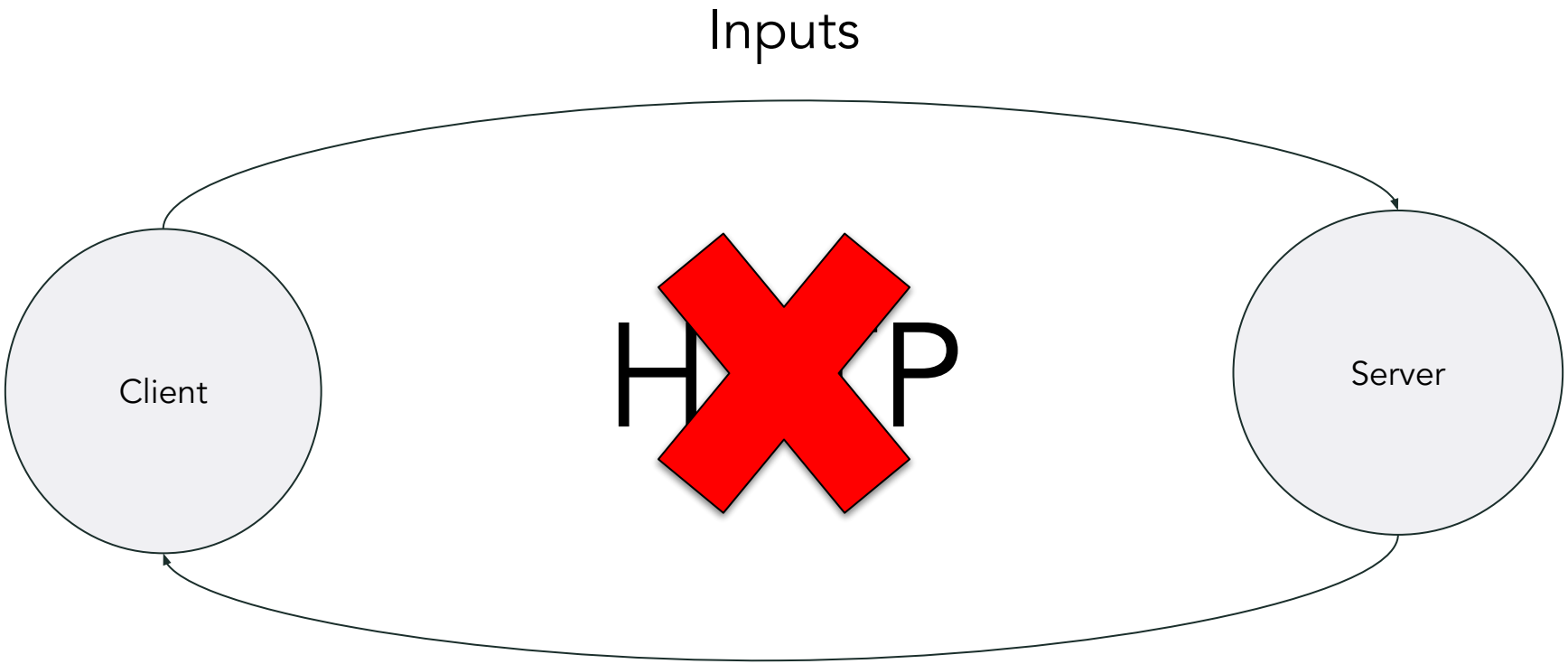
Multiplayer (HTTP)



- Overhead mínimo para envio de mensagens

- Full duplex
- Real-time

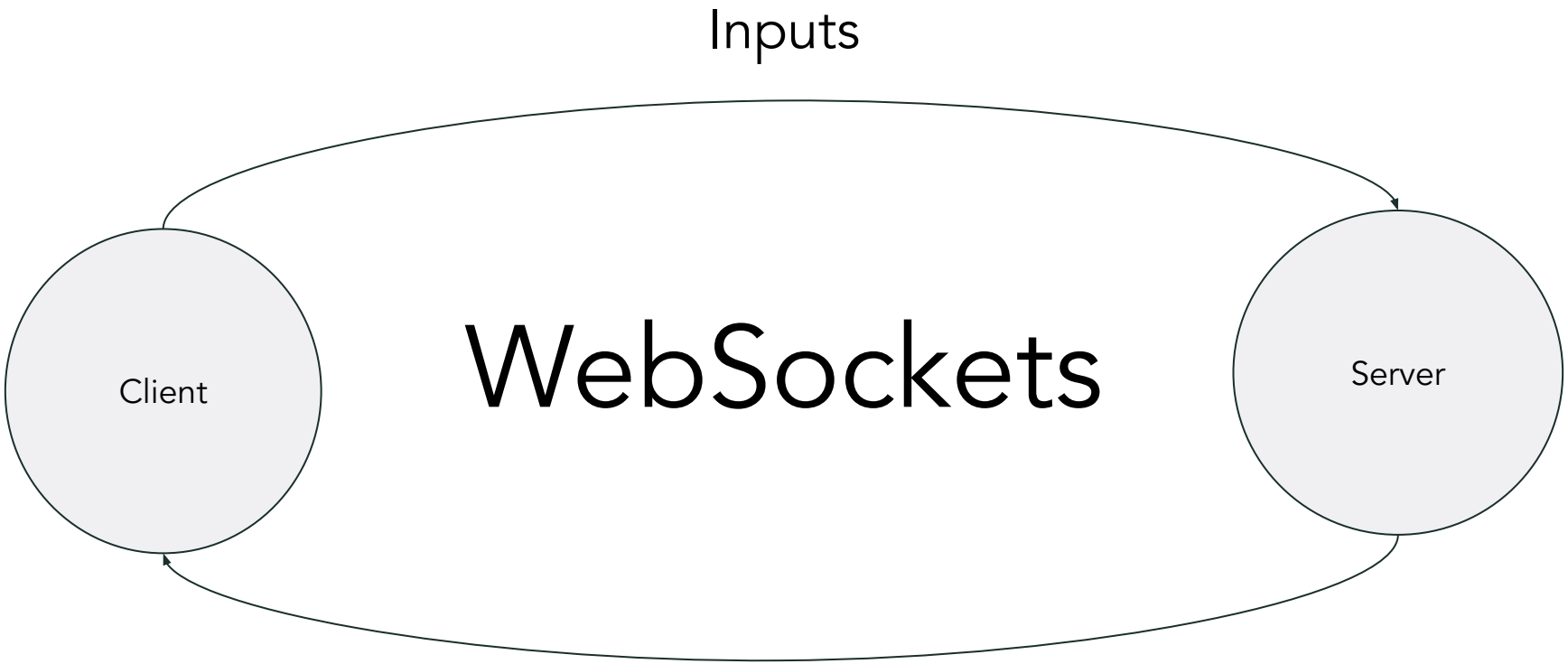
Multiplayer (HTTP)



- Overhead mínimo para envio de mensagens

- Full duplex
- Real-time

Multiplayer (WebSockets)



- Overhead mínimo para envio de mensagens

- Full duplex
- Real-time

gorilla/websocket

```
1 http.HandleFunc("/multiplayer", wsServer.HandleConnections)
2 http.HandleFunc("/spectate", wsServer.HandleSpectatorConnections)
3 http.HandleFunc("/sessions", s.handleSessions)
```

gorilla/websocket

```go
func (s *Server) HandleConnections(w http.ResponseWriter, r *http.Request) {
    // Upgrade HTTP connection to WebSocket
    conn, err := s.upgrader.Upgrade(w, r, nil)
    if err != nil {
        slog.Error("Failed to upgrade connection", slog.Any("error", err))
        return
    }

    // Wait for initial player info
    var info game.GameInfo
    if err := conn.ReadJSON(&info); err != nil {
        //...
    }
```

coder/websocket

- WASM support
- O acesso de rede em WebAssembly é restrito
  - Erro de DNS, dial, etc.
- Implementa build tags para detectar quando está rodando no browser e utiliza APIs específicas.

Goroutines (latência)

```
1  func sendPingMessages(player *game.Network) {
2      for {
3          select {
4          case <-player.Ctx.Done():
5              // Cleanup ...
6              return
7
8          case <-time.After(5 * time.Second):
9              player.Ping()
10         }
11     }
12 }
13
14 go sendPingMessage(...)
```

Goroutines (input)

```
for {
            select {
            case <-player.Network.Ctx.Done():
                return
            default:
                var input PlayerInput
                if err := player.Network.Conn.ReadJSON(&input); err != nil {
                    slog.Error("Error reading player input", slog.Any("error", err))
                    player.Terminate()

                    return
                }

                if !input.Up && !input.Down {
                    continue
                }

                slog.Info("Received input", slog.Any("input", input))

                player.inputQueue <- input
            }
        }
```

User Network Context

```
1  func sendPingMessages(player *game.Network) {
2      for {
3          select {
4          case <-player.Ctx.Done():
5              // Cleanup ...
6              return
7
8          case <-time.After(5 * time.Second):
9              player.Ping()
10         }
11     }
12 }
13
14 go sendPingMessage(...)
```

Game loop

- Ticker (60hz)

```go
func (session *GameSession) Start() {
    session.ticker = time.NewTicker(time.Second / 60)
    defer session.ticker.Stop()

    session.ready()

    for {
        select {
        case <-session.Player1.Network.Ctx.Done():
            session.handleDisconnection(session.Player1)
            return
        case <-session.Player2.Network.Ctx.Done():
            session.handleDisconnection(session.Player2)
        case <-session.ticker.C:
            session.update()

            session.broadcastGameState()

            session.broadcastToSpectators(
                session.currentGameState()
            )

            if session.gameEnded() {
                session.ticker.Stop()
                session.endGame()
            }
        }
    }
}
```

Frame update

```go
func (session *GameSession) update() {
    session.Player1.ProcessInputs()
    session.Player2.ProcessInputs()

    session.ball.Update(
        session.Player1.basePlayer.Bounds(),
        session.Player2.basePlayer.Bounds()
    )

    if scored, goalSide := session.ball.CheckGoal(); scored {
        session.handleScore(goalSide)
    }
}
```

Read Input

```go
func (p *Player) ProcessInputs() {
    for {
        select {
        case input := <-p.inputQueue:
            p.basePlayer.Update(player.Input{
                Up:   input.Up,
                Down: input.Down,
            })
        default:
            return
        }
    }
}
```

State Broadcast

```
case <-session.ticker.C:
    session.update()

    session.broadcastGameState()
```

Game State

```go
type GameState struct {
    Ball     BallState   `json:"ball"`
    Current  PlayerState `json:"current"`
    Opponent PlayerState `json:"opponent"`
}

type BallState struct {
    Angle    float64         `json:"angle"`
    Bounces  int             `json:"bounces"`
    Position geometry.Vector `json:"position"`
}

type PlayerState struct {
    Name      string        `json:"name"`
    PositionY float64       `json:"position_y"`
    Side      geometry.Side `json:"side"`
    Score     int8          `json:"score"`
    Ping      int64         `json:"ping"`
    Winner    bool          `json:"winner,omitempty"`
}
```

Live streaming

```
1  session.broadcastToSpectators(session.currentGameState())
```

Spectator (Live streaming)

```
type GameSession struct {
    ID       string
    Player1 *Player
    Player2 *Player
    ball     ball.Ball
    level    level.Level
    ticker   *time.Ticker

    // Spectate
    spectators     []Network
    spectatorMutex sync.Mutex
}
```

Melhorias

+ Command Pattern (permite rebind)
+ Streaming das sessões
+ Replay das sessões
+ Utilizar custom UDP ao invés de TCP
+ Melhorias de performance
  - channels vs Iterators
  - generics vs interface
+ Não enviar mudança de estado quando não houver necessidade
+ Tocar som ao interagir

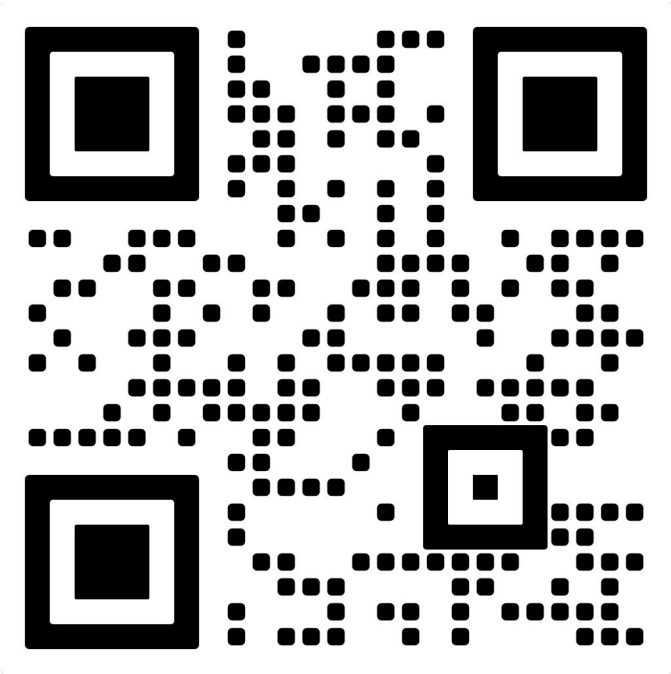
PONGO

One Player

Two Players

Back

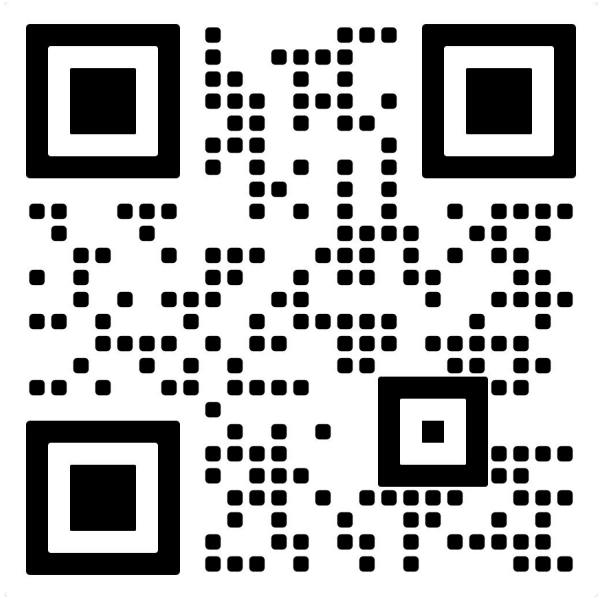
Demo!!



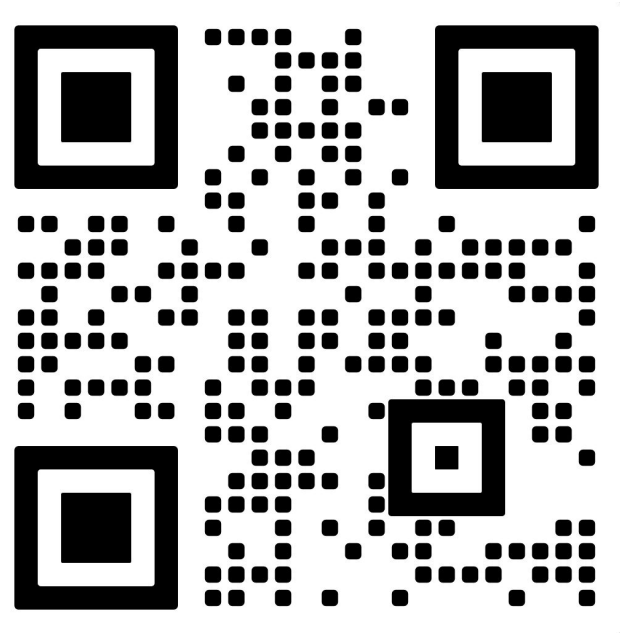
pong.go-go.dev

It’s only a demo sometime shit happens

Repositórios



pong-multiplayer-go
(game)



pongo-server
(server)

Obrigado

Carlos

Renê