



INF1350 - Documentação Projeto Final

Integrantes:

- Paulo de Tarso Fernandes
- Jerônimo Augusto Soares

Estrutura de pasta

```
love
  config.lua
  main.lua
  botao.lua
  map.lua
  maquina.lua
  mqtt_library.lua
  utility.lua
  PixelOperator8.ttf
nodemcu
  credenciais.lua
```

Conteúdo

[Estrutura de pasta](#)
[Como executar](#)
 [Controles do jogo](#)
 [Regras do jogo](#)
[Estrutura dos códigos](#)
 [Protocolo MQTT](#)
 [Tipos de mensagens enviadas e recebidas](#)
 [NodeMCU](#)
 [LÖVE2D](#)
 [Servidor Lua](#)
Link para apresentação do projeto:
<https://www.canva.com>



Como executar

- Tendo todos os arquivos e pastas previamente mencionados, baixar e instalar Lua (<https://www.lua.org/download.html>), LÖVE2D (<https://love2d.org/>) e a IDE ESPlorer para NodeMCU (<https://github.com/4refr0nt/ESPlorer/releases/download/v0.2.0/ESPlorer-0.2.0.zip>).
- Conecte um NodeMCU ao seu computador.
- Abra a IDE do ESPlorer no seu computador.
- Conecte-se a porta do NodeMCU no ESPlorer (usando 115200 de baud rate).
- Envie para o NodeMCU todos os arquivos Lua da pasta nodemcu. AVISO: é preciso configurar as informações corretas em credenciais.lua para a rede na qual se está conectado e em config.lua para o ID específico do NodeMCU utilizado.
- Execute o compilador.lua no NodeMCU para gerar os arquivos maquina.lc, lig4.lc e matriz.lc. Caso apareça uma mensagem de erro durante essa etapa ou a anterior reclamando de falta de memória, aperte o botão reset do NodeMCU e tente novamente.
- Repita os mesmos procedimentos com outros NodeMCU.
- Selecione o arquivo main.lua da pasta server e execute o código. Certifique-se de que a conexão com o protocolo MQTT foi bem-sucedida. Para isso, pode ser usada a IDE ZeroBrane Studio (<https://studio.zerobrane.com/>). AVISO: se tiver outra instância desse código sendo executada, o servidor cairá. Apenas uma execução em um dispositivo é necessária.
- Execute o arquivo main.lua da pasta love e mantenha o main.lua da pasta server em execução. Será possível visualizar uma “sala” (existem 8 salas, permitindo 8 partidas simultâneas) pela janela LÖVE2D. Com o mouse, pode-se clicar nas setas e trocar de sala. AVISO: caso queira

visualizar mais janelas, por exemplo para acompanhar duas salas ao mesmo tempo, ou para cada jogador poder ver a partida distantes um do outro, é necessário trocar o ID em “config.lua” da mesma pasta em outro dispositivo.

- Quando ambas luzes LED do NodeMCU acenderem, clique no quarto botão para se conectar ao protocolo MQTT.
- Após confirmação de conexão no MQTT, clique mais uma vez no quarto botão. Usando os dois primeiros botões do NodeMCU, selecione uma sala. Cada uma emite um som para poder ser identificada. AVISO: caso não apareçam opções de sala (nenhum som emitido), verifique se o servidor está sendo executado corretamente e se a janela LÖVE2D está desenhando o jogo.
- Aperte o quarto botão novamente para confirmar a sala escolhida.
- Quando a mesma sala tiver dois NodeMCUs inscritos, fique atento para seus LEDs, pois quando o verde está aceso, significa que é o seu turno para jogar, e o outro fica com o LED vermelho aceso durante esse tempo, impossibilitado de fazer qualquer jogada. A cada jogada, um som é emitido simulando o som de uma peça caindo no tabuleiro. AVISO: se ambos jogadores ficarem inativos por mais de 60 segundos, serão retirados da sala e a partida será descartada.
- Um som de vitória é reproduzido para um jogador e o outro escuta um som de derrota para sinalizar o término da partida. Para começar um novo jogo, basta que ambos jogadores apertem o botão lateral de reset do NodeMCU ou esperem 10 segundos para retornarem ao estado de recebimento de salas e a sala da partida anterior seja liberada novamente.

Controles do jogo

- primeiro botão do NodeMCU (canto superior esquerdo): move ficha para próxima coluna da esquerda
- segundo botão do NodeMCU (canto superior direito): move ficha para próxima coluna da direita
- quarto botão do NodeMCU (canto inferior direito): preenche uma ficha na coluna marcada
- botão lateral de reset do NodeMCU: recomeça a execução dos arquivos que contém e o retorna para o estado inicial da aplicação
- com o mouse, na janela LÖVE2D, é possível clicar nas setas para escolher a sala que se deseja observar

Regras do jogo

- Ambos jogadores devem preencher os buracos do mapa com suas fichas.
- Cada jogador só pode utilizar uma ficha no seu turno. Não é permitido jogar durante o turno do seu oponente.
- O objetivo é completar uma linha de 4 fichas, seja verticalmente, horizontalmente ou em diagonal.
- O jogo termina quando um dos jogadores atingir esse objetivo primeiro.
- Caso queira, pode assistir partidas ocorrendo em outras salas como espectador.
- ATENÇÃO: 60 segundos de inatividade durante uma partida resulta no cancelamento da partida e regressão ao estado pré-jogo. 10 segundos após o fim de uma partida, ela é apagada.

Estrutura dos códigos

Protocolo MQTT

Usando o broker fornecido para essa matéria:

```
topic = "INF1350_LIG4"  
broker = "139.82.100.100"  
port = 7981
```

Tipos de mensagens enviadas e recebidas

Mensagem	Remetente	Destinatário	Descrição
love_id,BROADCAST,NIL,ALL	LÖVE2D	Todos inscritos no tópico	<p>Essa mensagem indica um estado de espera. É a primeira mensagem enviada pela aplicação LÖVE2D ao entrar em qualquer sala. Enviada como broadcast para o servidor receber e saber que essa aplicação LÖVE2D quer receber a lista de salas. Apesar do servidor ter seu próprio ID na comunicação MQTT, pensamos que faria sentido imaginá-lo como um mediador, do qual não seria necessariamente possível acessar os dados.</p>
love_id,BROADCAST,salax,GET	LÖVE2D	Todos inscritos no tópico	A aplicação LÖVE2D está pedindo

Mensagem	Remetente	Destinatário	Descrição
			atualizações de uma das salas ao servidor. Acontece toda vez que tentamos visualizar uma sala específica na janela logo em seguida a mensagem anterior.
node_id,BROADCAST,NIL	NodeMCU	Todos inscritos no tópico	Assim como a primeira mensagem, indica um estado de espera, mas nesse caso com o NodeMCU. É enviada para o servidor saber que esse NodeMCU está esperando receber a lista de salas disponíveis. A mensagem possui broadcast pelo mesmo motivo daquela mensagem.
node_id,BROADCAST,salax,SUB	NodeMCU	Todos inscritos no tópico	Mensagem que indica inscrição do NodeMCU em uma sala para o servidor alocá-lo na partida.
node_id,BROADCAST,salax,MOV,valor	NodeMCU	Todos inscritos no tópico	Mensagem que indica que o NodeMCU moveu sua peça para outra coluna no tabuleiro. O

Mensagem	Remetente	Destinatário	Descrição
			servidor lê essa mensagem para resetar o timer de inatividade.
node_id,BROADCAST,salax,OK,valor	NodeMCU	Todos inscritos no tópico	Mensagem que indica que o NodeMCU colocou sua peça na coluna especificada no tabuleiro. O servidor também reseta o timer de inatividade com essa mensagem.
server_id,node_id,sala1;sala2;...	Servidor	NodeMCU	O servidor envia a lista de salas para o NodeMCU que requisitou.
server_id,node_id,salax,JOG1	Servidor	NodeMCU	O servidor aloca o primeiro NodeMCU que se inscreveu nessa sala como o jogador nº 1.
server_id,node_id,salax,JOG2	Servidor	NodeMCU	O servidor aloca o segundo NodeMCU que se inscreveu nessa sala como o jogador nº 2.
server_id,node_id,salax,NEG	Servidor	NodeMCU	O servidor nega alocar o NodeMCU para a sala que tentou se inscrever por estar cheia.
server_id,love_id,sala1;sala2;...	Servidor	LÖVE2D	O servidor envia a lista de salas para a aplicação LÖVE2D que requisitou.

Mensagem	Remetente	Destinatário	Descrição
server_id,love_id,salax,MATRIZ,0102...,valor	Servidor	LÖVE2D	Servidor envia para a aplicação LÖVE2D o estado atual do jogo, com a sala, a matriz com as peças jogadas e a posição atual da próxima peça.
server_id,OBS,salax,QTDJOG,valor	Servidor	LÖVE2D	Servidor passa quantidade de jogadores numa sala para a aplicação LÖVE2D atualizar seu indicador de quantidade. Nesse caso, é "OBS" e não "love_id" porque qualquer aplicação LÖVE2D deve ler essa mensagem.
server_id,OBS,salax,MOV,valor	Servidor	LÖVE2D	Após receber a mensagem com MOV do NodeMCU, o servidor repassa a mesma informação para o LÖVE2D desenhar.
server_id,OBS,salax,OK,valor	Servidor	LÖVE2D	Após receber a mensagem com confirmação do posicionamento da peça mais recente do NodeMCU, o servidor repassa a mesma informação para o LÖVE2D desenhar.

Mensagem	Remetente	Destinatário	Descrição
server_id,BROADCAST,salax,RESET	Servidor	Todos inscritos no tópico	O servidor envia essa mensagem para todos a fim de notificá-los que essa sala foi resetada e qualquer partida ou NodeMCU inscrito foi removido.

NodeMCU

- credenciais.lua e config.lua, como mencionados acima, são arquivos de configuração de informações como rede wi-fi e o ID do NodeMCU utilizado.
- station.lua depende das informações de credenciais.lua. Esse arquivo realiza a conexão com a rede wi-fi, afinal. Além disso, executa o lig4.lc. Chegaremos nesse arquivo nos próximos tópicos.
- init.lua prepara os LEDs do NodeMCU e executa station.lua.
- beep.lua declara a função recursiva beep, que recebe uma lista de notas musicais para tocar no NodeMCU. Ela vai tocando nota por nota até a lista ficar vazia. Usa funções do módulo pwm (para reprodução de sons no buzzer) e tmr (para timer) do NodeMCU.

```

local function beep(notas)
    local nota = table.remove(notas,1)
    local freq,duration = nota[1],nota[2]
    freq = frequencias[freq] or freq
    pwm.stop(buzzer)
    pwm.setup(buzzer, freq, 512)
    pwm.start(buzzer)
    tmr.create():alarm(duration, tmr.ALARM_SINGLE, function()
        pwm.stop(buzzer)
        if #notas > 0 then
            beep(notas)
        end
    end)
end

```

- O beep é utilizado pela matriz.lua, que é responsável pela criação da matriz que representa as peças no jogo. Possui três funções relacionadas a essa matriz: imprime (imprime a matriz do jogo no seu estado atual), dropPiece (atualiza a matriz colocando a peça atual na coluna escolhida pelo jogador e reproduz um som quando isso acontece) e verifica (que confere se algum jogador já venceu a partida ou se o jogo continua normalmente).

```

function matriz.dropPiece(peca, x)
    local dt = 160
    local notas = {}
    notas[2] = { 1000, 200 }
    if matriz[6][x] ~= 0 then
        return false
    elseif matriz[1][x] == 0 then
        matriz[1][x] = peca
        dt = dt * 6
        notas[1] = { 1, dt }
        beep(notas)
        return true
    end
    for i = 5, 1, -1 do
        if matriz[i][x] ~= 0 then
            matriz[i + 1][x] = peca
            dt = dt * (6-i)
            break
        end
    end
    notas[1] = { 1, dt }
    beep(notas)
    return true
end

```

- Usando tanto matriz.lc e beep.lua, a maquina.lua implementa a máquina de estados do NodeMCU. A função criaMaquina engloba os estados da máquina e retorna a máquina em si. Temos os seguintes estados: inicio (conecta NodeMCU ao MQTT), conectando (estado intermediário entre o anterior e o próximo apenas para confirmar a conexão bem-sucedida), espera (espera receber mensagem com a lista de salas), escolhendo (com a lista de salas, pode-se passar por cada e escolher uma), entrando (aloça NodeMCU a posição de jogador nº1, 2, ou retorna ao estado de espera se a sala estiver lotada), jogo1 (turno do jogador nº1) e jogo2 (turno do jogador nº2). Durante a partida, ficamos transitando entre estados jogo1 e jogo2 até que haja um reset para o estado de espera, seja pelo fim do jogo ou só pura inatividade mesmo.
- lig4.lua depende de config.lua e maquina.lc, pois ela centraliza as informações de todos esses códigos para usá-los no jogo. Além de declarar um conjunto de variáveis constantes para serem usadas por maquina.lua, ele trata o debounce dos botões e cuida da conexão com o MQTT pela função conecta_cliente abaixo.

```

function conecta_cliente()
    print("Conectando cliente")

    consts.m = mqtt.Client("INF1350_" .. config.meuid, 120)

    local function conexao_sucesso(client)
        print("Connected to MQTT broker")
        client:subscribe(config.topic, 0, function(client)
            f = maquina[consts.estado] and maquina[consts.estado]["subscribe"]
            if f ~= nil then
                f(client)
            end
        end)
    end

```

```

    end

    local function conexao_falha(client)
        f = maquina[consts.estado] and maquina[consts.estado]["confail"]
        if f == nil then
            f(client)
        end
    end

    consts.m:on("message", function(client, topic, message)
        f = maquina[consts.estado] and maquina[consts.estado]["message"]
        if f == nil then
            f(client, topic, message)
        end
    end)

    consts.m:on("offline", function(client)
        node.restart()
    end)

    consts.m:connect(config.broker, config.port, 0, conexao_sucesso, conexao_falha)
end

```

- compilador.lua apenas compila matriz.lua, maquina.lua e lig4.lua, os arquivos de maior tamanho da pasta. Sua execução de dentro do NodeMCU resulta nos arquivos de mesmo nome, mas com terminações .lc, a versão compilada do código original, e necessário por questões de espaço de memória. Os arquivos .lc são menores em tamanho de memória.

LÖVE2D

- utility.lua e mqtt_library.lua são arquivos fornecidos pelo professor da disciplina, que possibilitam o uso de funções para protocolo MQTT.
- config.lua configura as informações de MQTT da mesma forma que o do NodeMCU, com um ID próprio.
- map.lua é responsável pelo desenho do tabuleiro e das peças, definindo a função dropPiece. Diferente da mesma função na pasta nodemcu, essa não apenas atualiza uma matriz com a posição das peças, também realiza a animação da peça caindo até seu local escolhido.
- botao.lua cria a classe botão, que é usada para criar os botões que trocam de sala na janela LÖVE2D. É repleta de métodos, como funções para definir o texto e a cor do botão.
- A máquina de estados do LÖVE2D é implementada pelo maquina.lua. Ela possui menos estados que a do NodeMCU, com apenas esses três: “listando” (que recebe a lista de salas e manda mensagem para pegar informações de uma específica), “carregando” (na qual o tabuleiro é desenhado e o número de jogadores indicado) e “jogo” (quando dois jogadores se inscrevem e a partida começa).
- Na main.lua, temos botões sendo criados e usados como afirmado previamente. Temos uma função auxiliar mysplit que recebe um texto input e um caractere separador para retornar uma lista com cada conteúdo da string separado pelo caractere e sendo inserido como elemento (usado para a lista de salas, por exemplo). Dentro das funções de desenhar, é passado um arquivo externo de fonte de texto para os botões. A função draw principal fica bem simples (constsMaq.desenha é uma flag para permitir ou não o desenho do mapa do tabuleiro):

```

function love.draw()
    if constsMag.desenha then
        --Desenho do mapa
        M:draw()
    end

    botao:draw()
    botaoEsq:draw()
    botaoDir:draw()
end

```

Servidor Lua

- Assim como na pasta love, a pasta do servidor possui os arquivos utility.lua e mqtt_library.lua.
- O servidor se inscreve no tópico MQTT "INF1350_LIG4" usando um ID próprio, configurado em config.lua.
- matriz.lua possui uma função principal criaMatrizVazia(), que cria e retorna a matriz do jogo, além de conter todos os métodos associados ao objeto matriz, como as funções imprime, dropPiece, toString (a matriz é passada como string para a aplicação LÖVE2D, como indicado na tabela de mensagens) e verifica (semelhante a da pasta nodemcu, utilizada aqui para reiniciar o jogo após seu término).
- main.lua define as informações de timeout e as salas. Cada sala tem uma matriz com a posição das peças jogadas, a posição x da peça em cima do tabuleiro para ser jogada, a quantidade de jogadores, o número do jogador que é o dono do turno e o timer responsável pelo controle do timeout.

```

local timeout = 60

local nomesSalas = {"a", "ab", "abc", "abcd", "g", "gf", "gfe", "gfed"}

local salas = {}

for i,nome in ipairs(nomesSalas) do
    salas[nome]={}
    salas[nome].matriz = criaMatrizVazia()
    salas[nome].x = 1
    salas[nome].qtdJogadores = 0
    salas[nome].vez=1
    salas[nome].timer=nil
end

```

A função principal é a mqttcb, que recebe o tópico MQTT e uma mensagem e envia uma mensagem em resposta. Nela, usamos a corrotina para controlar o timer de inatividade da sala. Declaramos uma função que reseta as informações da sala se o tempo de timeout passar, passado como parâmetro callback na função timer abaixo.

```

sala.timer = coroutine.create(timer)
coroutine.resume(sala.timer, timeout, function ()

```

```
sala.matriz = criaMatrizVazia()
sala.x = 1
sala.qtdJogadores = 0
sala.vez=1
sala.timer=nil
local msgSend = meuid .. ",BROADCAST," .. tmsg[3] .. ",RESET"
print(msgSend)
mqtt_client:publish(topic, msgSend)
end)
```

```
local timer = function(tempo, callback)
    local tempo1 = os.time()
    while true do
        local dt = os.time() - tempo1
        if dt >= tempo then
            callback()
            break
        end
        coroutine.yield()
    end
end
```

Link para apresentação do projeto:

<https://www.canva.com/design/DAF1WFOdmXU/OMRZOHkntEFpRgx6FA091Q/edit>