

Universidade Rural do Rio de Janeiro
Departamento de Ciências da Computação - DCC

Relatório sobre a Atividade Acadêmica de Redes

Aluno: Júnior Miranda e Rafael Leal
Professor: Marcel Silva

Junho
2019

Universidade Rural do Rio de Janeiro
UFRRJ
DCC

Relatório

Relatório do projeto desenvolvido na Atividade Acadêmica de Redes do Curso de Ciências da Computação da Universidade Federal Rural do Rio de Janeiro - UFRRJ, como requisito para aprovação.

Aluno: Júnior Miranda e Rafael Leal

Professor orientador: Marcel Silva

Junho
2019

Conteúdo

1	Resumo	1
2	Apresentação	2
3	Descrição da Atividade	3
4	Código fonte	4
4.1	Server	4
4.2	Cliente	6
4.3	Game	12
4.4	Network	15
5	Resultados Obtidos	17

1 Resumo

Como requisito de aprovação na disciplina, foi desenvolvido um jogo aplicando os conceitos aprendidos na disciplina de Redes do Curso de Ciências da Computação da UFRRJ. O aplicativo desenvolvido se trata de conceitos como utilização de sockets, desenvolvimento de servidor, clientes (ativo e passivo) e utilização de threads em rede.

2 Apresentação

O presente relatório visa apresentar o trabalho desenvolvido na Atividade Acadêmica de Redes. O requisito da atividade foi o desenvolvimento de uma aplicação que se comunica via rede com sockets (*User Datagram Protocol* - UDP ou *Transmission Control Protocol* - TCP). O protocolo utilizado neste trabalho foi o TCP. O objetivo era exercitar os conceitos aprendidos na disciplina, como o uso de protocolos de comunicação, conceitos de cliente/servidor e comunicação em escala.

Para esta atividade foi desenvolvido um jogo chamado **Pedra-Papel-Tesoura-Lagarto-Spock**. Se trata de uma expansão do tradicional **pedra-papel-tesoura**, porém com a redução da possibilidade de terminar em empate [1]. O jogo foi desenvolvido por [2] e popularizado pela série de televisão americana " *The Big Bang Theory*". A regra de funcionamento é descrita na figura 1.

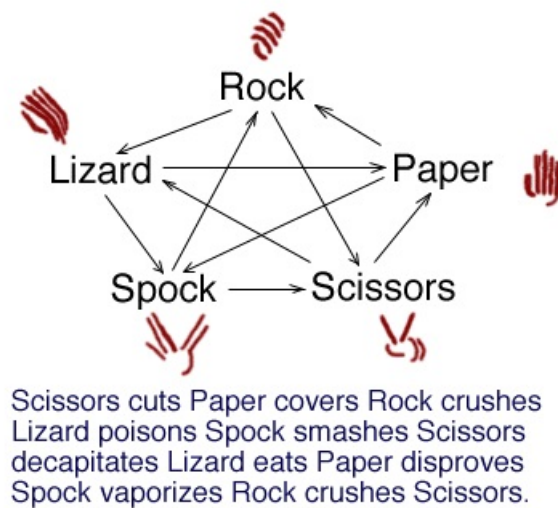


Figura 1: Regra do jogo Pedra-Papel-Tesoura-Lagarto-Spock

3 Descrição da Atividade

O desenvolvimento deste aplicativo foi dividido em 4 arquivos diferentes: *server*, *cliente*, *game* e *network*. A tarefa de cada um respectivamente é: Iniciar o servidor e permitir a conexão, fazer interface para com o usuário, responsável pela lógica do jogo/gestão das jogadas e fazer interface de conexão com o servidor para o cliente.

Na classe *server* são feitas as seguintes tarefas: verificação da conectividade, tempo máximo de espera por conexão e inicialização de dicionário de dados. Este arquivo também é responsável por criar as *threads* das conexões e verificar pares dos jogos abertos. O servidor é responsável por centralizar todas as conexões e distribuir as jogadas para os jogadores ativos e passivos conectados a determinada partida.

Além de ser responsável pela interface gráfica do cliente, esta classe é responsável por fazer verificações para que cada jogada seja feita apenas quando há o mínimo de jogadores conectados. A cada jogada feita é verificado do estado atingido.

O *game* é a classe responsável por reter as regras do jogo. É nela que são calculadas as jogadas e é verificado se o jogador é vencedor ou não. No código encontramos funções como *play* (para fazer movimentos), *winner* (função para verificar a possibilidade de vitória) e outras.

Por fim, a classe *network* é quem faz a conexão com o servidor. Nela são feitas as configurações da rede e das portas.

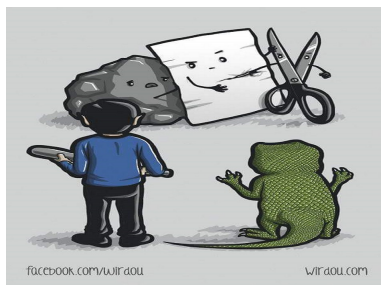


Figura 2: Tela de apresentação do jogo.

4 Código fonte

4.1 Server

```
import socket
from _thread import *
import pickle
from game import Game

server = "127.0.0.1"
port = 5555

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Inicia socket TCP.

try:
    s.bind((server, port))
except socket.error as e:
    str(e)

s.listen(2)
print("Esperando por uma conexão, Servidor Inicializado!")

connected = set()
games = {}
idCount = 0

def threaded_client(conn, p, gameId):
    global idCount
    conn.send(str.encode(str(p)))

    reply = ""
    while True: #função roda constantemente recebendo dados do cliente
```

```

try:
    data = conn.recv(4096).decode()

    if gameId in games:
        game = games[gameId]

        if not data:
            break
        else:
            if data == "reset":
                game.resetWent()
            elif data != "get":
                game.play(p, data)

            conn.sendall(pickle.dumps(game))
        else:
            break
except:
    break

print("Conexão Perdida")
try:
    del games[gameId]
    print("Fechando Jogo", gameId)
except:
    pass
idCount -= 1
conn.close()

while True: #função pra verificar os pares de jogos abertos

```



```

conn, addr = s.accept()
print("Conectado em:", addr)

idCount += 1
p = 0
gameId = (idCount - 1)//2
if idCount % 2 == 1:
    games[gameId] = Game(gameId)
    print("Criando um jogo novo...")
else:
    games[gameId].ready = True
    p = 1

start_new_thread(threaded_client, (conn, p, gameId))

```

4.2 Cliente

```

import pygame
from network import Network
import pickle
pygame.font.init()

width = 950
height = 700
win = pygame.display.set_mode((width, height))
foto = pygame.image.load('rpsls.png')
fotoSpock = pygame.image.load('spock.jpg')
pygame.display.set_caption("Client")

class Button:

```

```

def __init__(self, text, x, y, color):
    self.text = text
    self.x = x
    self.y = y
    self.color = color
    self.width = 150
    self.height = 100

def draw(self, win):
    pygame.draw.rect(win, self.color, (self.x, self.y,
    self.width, self.height))
    font = pygame.font.SysFont("comicsans", 40)
    text = font.render(self.text, 1, (255,255,255))
    win.blit(text, (self.x + round(self.width/2) -
    round(text.get_width()/2), self.y +
    round(self.height/2) - round(text.get_height()/2)))

def click(self, pos):
    x1 = pos[0]
    y1 = pos[1]
    if self.x <= x1 <= self.x + self.width and self.y <= y1
    <= self.y + self.height:
        return True
    else:
        return False

def redrawWindow(win, game, p):
    win.fill((128,128,128))
    imagem(275,10)

    if not(game.connected()):

```

```

font = pygame.font.SysFont("freesans", 80)
text = font.render("Esperando oponente...",
1, (255,255,255), True)
win.blit(text, (width/2 - text.get_width()/2,
height/2 - text.get_height()/2))
else:
font = pygame.font.SysFont("freesans", 60)
text = font.render("Você", 1, (0, 255,255))
win.blit(text, (120, 200))

text = font.render("Oponente", 1, (0, 255, 255))
win.blit(text, (590, 200))

move1 = game.get_player_move(0)
move2 = game.get_player_move(1)
if game.bothWent():
    text1 = font.render(move1, 1, (0, 0, 0))
    text2 = font.render(move2, 1, (0, 0, 0))
else:
    if game.p1Went and p == 0:
        text1 = font.render(move1, 1, (0, 0, 0))
    elif game.p1Went:
        text1 = font.render("Pronto", 1, (0, 0, 0))
    else:
        text1 = font.render("Esperando...", 1, (0, 0, 0))

    if game.p2Went and p == 1:
        text2 = font.render(move2, 1, (0,0,0))
    elif game.p2Went:
        text2 = font.render("Pronto", 1, (0, 0, 0))
    else:
        text2 = font.render("Esperando...", 1, (0, 0, 0))

```

```

        if p == 1:
            win.blit(text2, (50, 350))
            win.blit(text1, (575, 350))
        else:
            win.blit(text1, (50, 350))
            win.blit(text2, (575, 350))

    for btn in btns:
        btn.draw(win)

    pygame.display.update()

btns = [Button("1 Rock", 10, 500, (0,0,0)),
        Button("2 Scissors", 190, 500, (255,0,0)),
        Button("3 Paper", 390, 500, (0,255,0)),
        Button("4 Lizard", 590, 500, (0,255,255)),
        Button("5 Spock", 790, 500, (255,0,255))]

def main():
    run = True
    clock = pygame.time.Clock()
    n = Network()
    player = int(n.getP())
    print("Você é o jogador", player)

    while run:

        clock.tick(60)
        try:
            game = n.send("get")

```

```

except:
    run = False
    print("Couldn't get game")
    break

if game.bothWent():
    redrawWindow(win, game, player)
    pygame.time.delay(500)
    try:
        game = n.send("reset")
    except:
        run = False
        print("Couldn't get game")
        break

font = pygame.font.SysFont("comicsans", 90)
if (game.winner() == 1 and player == 1) or
(game.winner() == 0 and player == 0):
    text = font.render("Vitória!!", 1, (255,255,255))
elif game.winner() == -1:
    text = font.render("Empate!", 1, (255,255,255))
else:
    text = font.render("Derrota...", 1, (255, 255, 255))

win.blit(text, (width/2 - (text.get_width()/2) - 40 ,
height/2 - (text.get_height()/2) - 35))
pygame.display.update()
pygame.time.delay(2000)

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False

```

```

pygame.quit()

if event.type == pygame.MOUSEBUTTONDOWN:
    pos = pygame.mouse.get_pos()
    for btn in btns:
        if btn.click(pos) and game.connected():
            if player == 0:
                if not game.p1Went:
                    n.send(btn.text)
            else:
                if not game.p2Went:
                    n.send(btn.text)

    redrawWindow(win, game, player)

def imagem(x,y):
    win.blit(foto,(x,y))

def imageSpock(x,y):
    win.blit(fotoSpock,(x,y))

def menu_screen():
    run = True
    clock = pygame.time.Clock()

    while run:
        clock.tick(60)
        win.fill((128, 128, 128))
        font = pygame.font.SysFont("freesans", 60)
        imageSpock(0,0)
        text = font.render("Clique", 1, (92,51,23))
        textDois = font.render("para", 1, (92,51,23))

```

```

        textTres = font.render("jogar!", 1, (92,51,23))
        win.blit(text, (400,300))
        win.blit(textDois, (410,400))
        win.blit(textTres, (410,500))

    pygame.display.update()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            run = False
        if event.type == pygame.MOUSEBUTTONDOWN:
            run = False

    main()

while True:
    menu_screen()

```

4.3 Game

```

class Game:
    def __init__(self, id):
        self.p1Went = False
        self.p2Went = False
        self.ready = False
        self.id = id
        self.moves = [None, None]
        self.wins = [0,0]
        self.ties = 0

    def get_player_move(self, p):

```

```

        return self.moves[p]

def play(self, player, move):
    self.moves[player] = move
    if player == 0:
        self.p1Went = True
    else:
        self.p2Went = True

def connected(self):
    return self.ready

def bothWent(self):
    return self.p1Went and self.p2Went

def winner(self):

    p1 = self.moves[0].upper()[0]
    p2 = self.moves[1].upper()[0]

    winner = -1
    if p1 == "1" and p2 == "2":
        winner = 0
    elif p1 == "1" and p2 == "3":
        winner = 1
    elif p1 == "1" and p2 == "4":
        winner = 0
    elif p1 == "1" and p2 == "5":
        winner = 1
    elif p1 == "2" and p2 == "1":
        winner = 1
    elif p1 == "2" and p2 == "3":

```



```
winner = 0
elif p1 == "2" and p2 == "4":
    winner = 0
elif p1 == "2" and p2 == "5":
    winner = 1
elif p1 == "3" and p2 == "1":
    winner = 0
elif p1 == "3" and p2 == "2":
    winner = 1
elif p1 == "3" and p2 == "4":
    winner = 1
elif p1 == "3" and p2 == "5":
    winner = 0
elif p1 == "4" and p2 == "1":
    winner = 1
elif p1 == "4" and p2 == "2":
    winner = 1
elif p1 == "4" and p2 == "3":
    winner = 0
elif p1 == "4" and p2 == "5":
    winner = 0
elif p1 == "5" and p2 == "1":
    winner = 0
elif p1 == "5" and p2 == "2":
    winner = 0
elif p1 == "5" and p2 == "3":
    winner = 1
elif p1 == "5" and p2 == "4":
    winner = 1

return winner
```

```
def resetWent(self):  
    self.p1Went = False  
    self.p2Went = False
```

4.4 Network

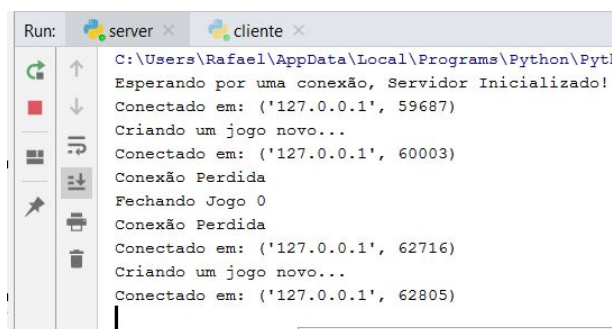
```
import socket  
import pickle  
  
class Network:  
    def __init__(self):  
        self.client = socket.socket(socket.AF_INET,  
                                     socket.SOCK_STREAM)  
        self.server = "127.0.0.1"  
        self.port = 5555  
        self.addr = (self.server, self.port)  
        self.p = self.connect()  
  
    def getP(self):  
        return self.p  
  
    def connect(self):  
        try:  
            self.client.connect(self.addr)  
            return self.client.recv(2048).decode()  
        except:  
            pass  
  
    def send(self, data):  
        try:  
            self.client.send(str.encode(data))
```

```
        return pickle.loads(self.client.recv(2048*2))
    except socket.error as e:
        print(e)
```

5 Resultados Obtidos

Nas telas que se seguem são apresentados os resultados das conexões feitas entre o servidor e os clientes.

Primeiramente o servidor é iniciado e aguarda as conexões, como pode ser visto na figura 3. Para cada conexão feita o servidor "imprime" o IP e a porta do cliente.



```
Run: server x cliente x
C:\Users\Rafael\AppData\Local\Programs\Python\Pyt
Esperando por uma conexão, Servidor Inicializado!
Conectado em: ('127.0.0.1', 59687)
Criando um jogo novo...
Conectado em: ('127.0.0.1', 60003)
Conexão Perdida
Fechando Jogo 0
Conexão Perdida
Conectado em: ('127.0.0.1', 62716)
Criando um jogo novo...
Conectado em: ('127.0.0.1', 62805)
```

Figura 3: Saídas do servidor: inicialização ao recebimento de conexões.

Depois do servidor ativo os cliente pode fazer as devidas conexões, como podemos ver na figura 4.



Figura 4: Tela inicial exposta ao conectar e tela de espera após o clique para iniciar o jogo.

Nas próximas imagens estão as telas dos clientes devidamente conectados e o exemplo de uma ação de um *jogador1* e a espera do movimento do *jogador2*.

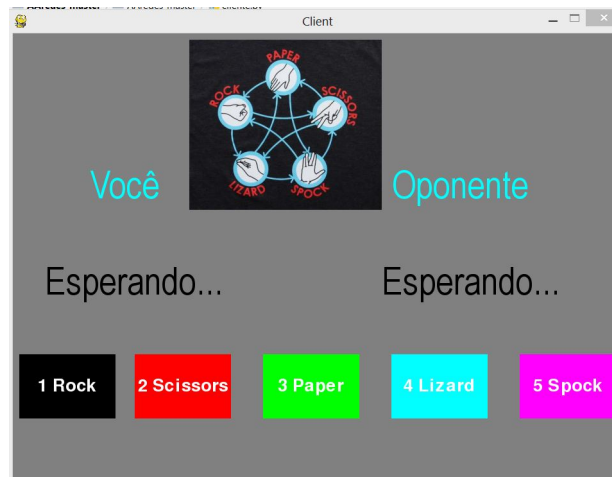


Figura 5: Conexões entre jogadores concluídas com sucesso.

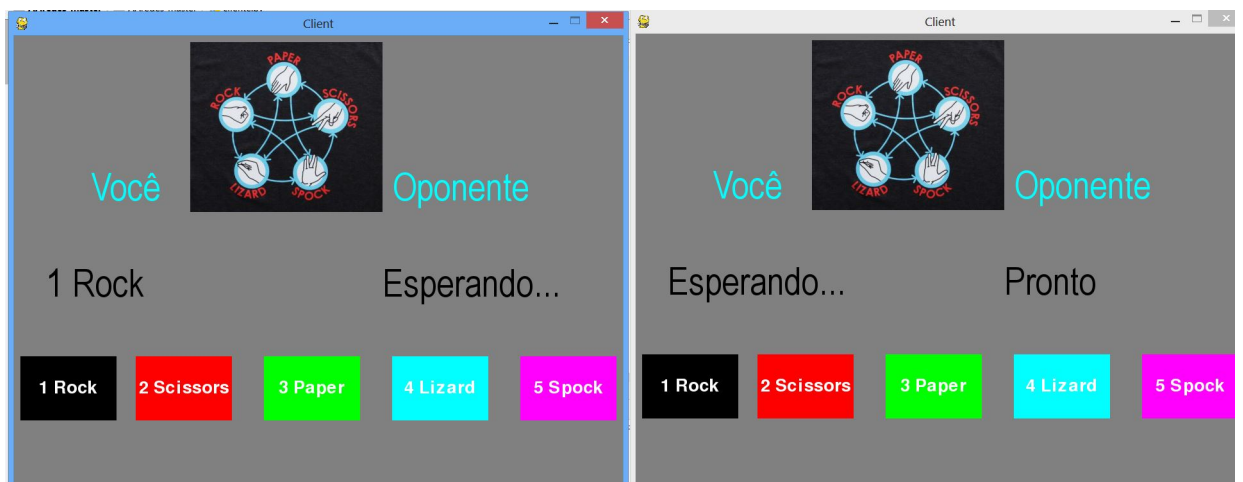


Figura 6: Jogador 1 jogou e jogador 2 está em espera.

A última imagem é o resultado de uma jogada que resulta no fim do jogo.

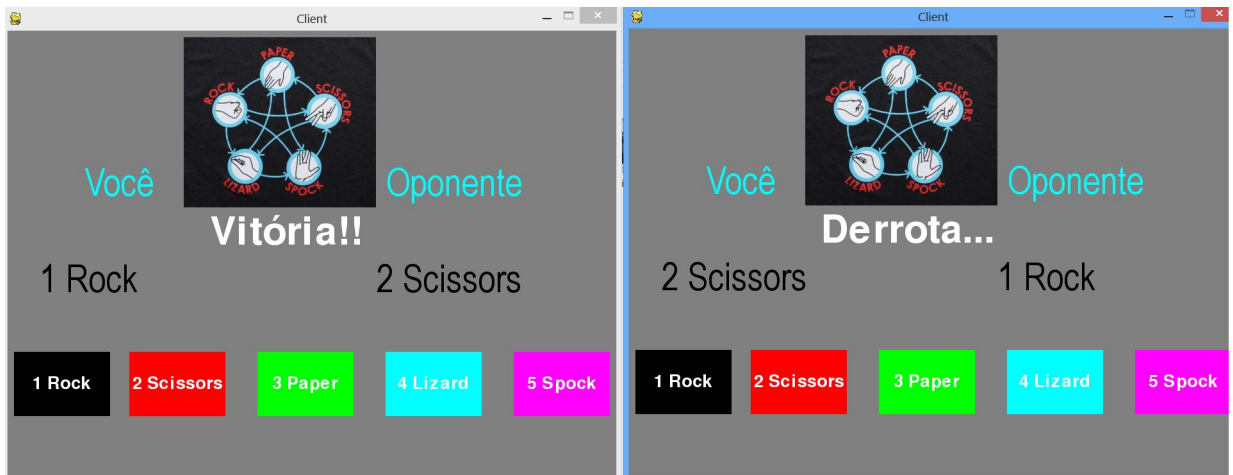


Figura 7: Jogador 1 ganhou a partida.

Referências

- [1] Desconhecido. Pedra-papel-tesoura-lagarto-spocks, 2019.
- [2] Sam Kass e Karen Bryla. Rock paper scissors spock lizard, 2010.