UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
REDES DE COMPUTADORES

Diogo Tallys da Mota Amorim
Hugo Coelho da Silva
Thiago Ribeiro da Silva
Vinicius da Costa Neitzke

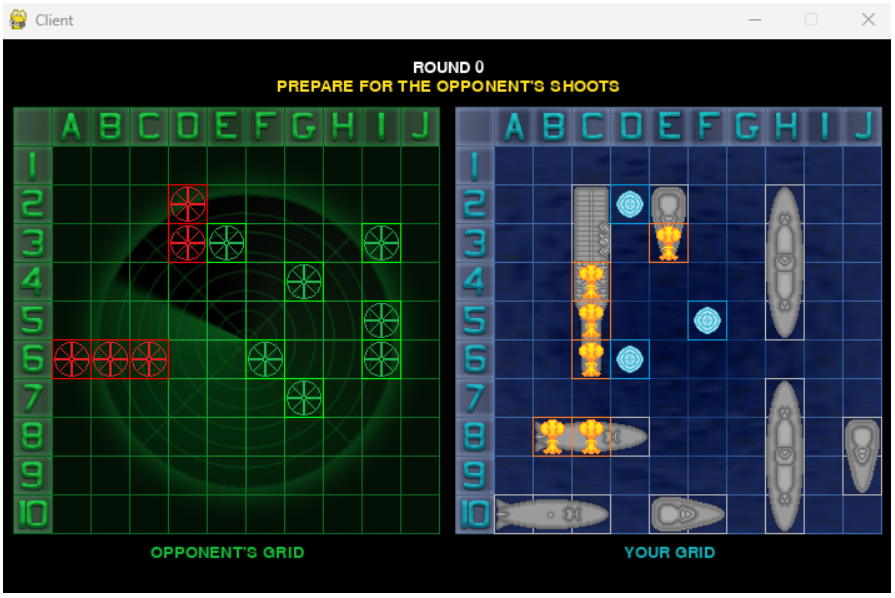
Projeto de Redes: Batalha Naval Multiplayer com Pygame e Socket

MACEIÓ
2023

SUMÁRIO

1. INTRODUÇÃO

Optamos por adotar o conceito do clássico jogo "Batalha Naval" como uma ótima ilustração de uma comunicação interativa entre pares de clientes, mediada por um servidor. Esta escolha nos permite oferecer uma demonstração prática do funcionamento dos sockets, facilitando a compreensão do processo de comunicação entre máquinas por meio do protocolo TCP, que é reconhecido por sua robustez na transferência segura de informações. O jogo em si apresenta uma estrutura simples, tornando-o acessível para a compreensão dos fluxos de comunicação entre cliente e servidor. Além disso, proporciona uma excelente oportunidade para o desenvolvimento de habilidades práticas na criação de jogos e na compreensão dos conceitos subjacentes às aplicações multiusuário.



2. FUNCIONALIDADES

A aplicação inclui um arquivo *Python* que tem como função principal inicializar o servidor. Este servidor é responsável por criar sessões do jogo e gerenciar a comunicação entre os clientes que participam da mesma partida, unindo-os em pares. O arquivo do cliente, por sua vez, inicia uma partida em uma máquina específica e aguarda a chegada de um segundo jogador, uma vez que uma partida de batalha naval requer a participação de dois jogadores.

Quando uma instância do cliente é iniciada, ela se conecta ao servidor, caso este esteja operando, e cria uma nova instância de jogo, desde que não haja outro cliente aguardando por um adversário. Como resposta, o cliente recebe seu identificador de jogador durante a sessão. A partir desse ponto, o cliente entra em um loop em que envia periodicamente uma mensagem "get" para receber a instância atualizada do jogo ao qual ele está participando.

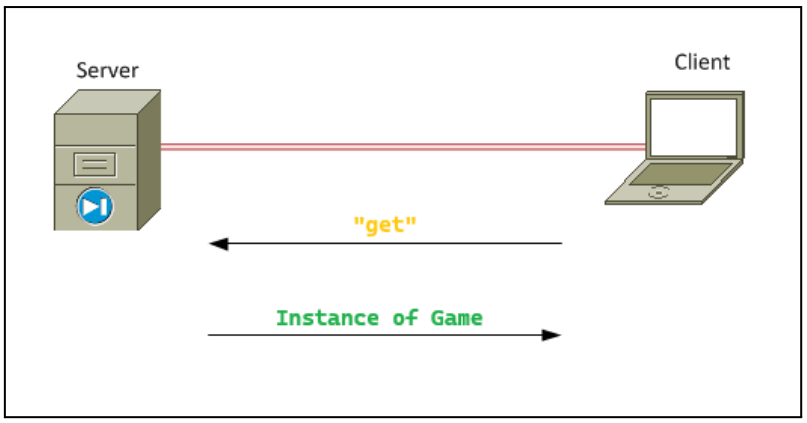
Conforme um segundo cliente se conecta ao servidor, ele é automaticamente incorporado à instância existente do jogo, marcando o status desta como "pronta para iniciar". Neste ponto, os dois clientes estão conectados e o servidor atua como intermediário, transmitindo instâncias atualizadas do jogo que evoluem com base nas ações dos jogadores.

Durante as partidas, cada jogador tem um limite de 3 tentativas de tiro no campo naval do oponente. Uma vez que esse limite seja atingido, o estado do jogador atual é atualizado, indicando a vez do próximo jogador. O vencedor da partida é determinado quando um jogador destrói todo o campo naval do adversário. Nesse momento, o jogo é reiniciado, com a configuração de novos campos para o próximo round, continuando assim até que a conexão de um dos clientes seja encerrada.

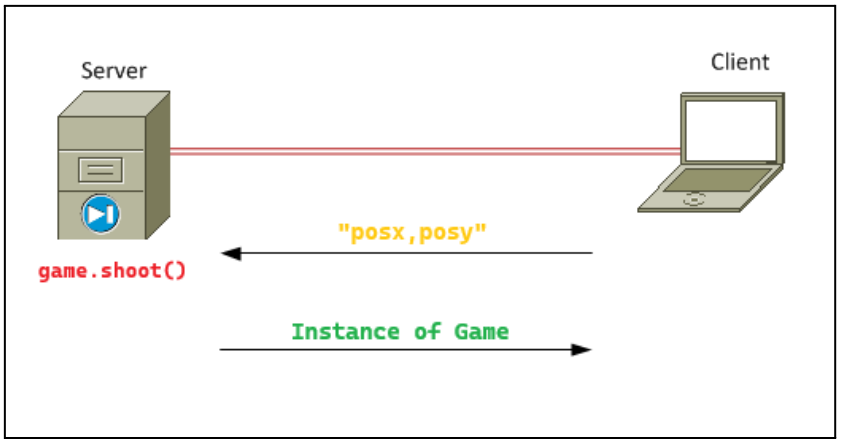
2.1. Comunicações com o servidor

Seguem as principais comunicações que ocorrem entre o cliente e o servidor, os retornos e os processos que são feitos juntamente dessas comunicações.

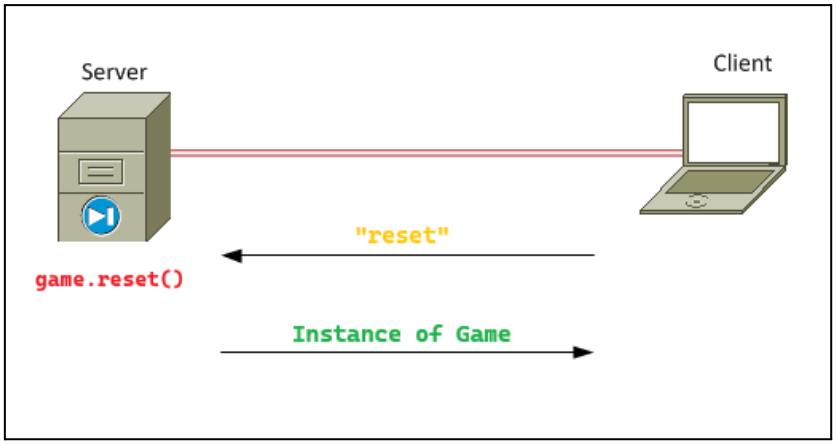
- **Get:** Ao enviar um “get”, o cliente indica que está conectado e está esperando instâncias atualizadas do jogo



➢ **Position:** Ao enviar uma posição, o cliente indica que houve um ataque ao campo naval do adversário e a instância do jogo trata dessas decisões sobre o ataque no servidor.



➢ **Reset:** Ao enviar um "reset", o cliente indica que uma partida chegou ao fim e então a instância do jogo realiza a reinicialização das variáveis e inicia a nova partida.



3. PRINCIPAIS DESAFIOS

Os principais desafios se situam tanto no campo da logística para implementação do socket, quanto com as decisões referentes à comunicação entre clientes, intermediada pelo servidor e também a geração dos campos navais com sua lógica específica.

Inicialmente, o desafio estava em entender como se construía uma comunicação com sockets usando Python. Estudamos a fundo a biblioteca **socket** para compreender como instanciar um servidor e clientes, e então fazer com que o servidor escute chamadas do cliente. Ocorreram alguns problemas para compreender como utilizar as threads para que múltiplos clientes consigam se comunicar com o servidor paralelamente, mas todos os problemas foram resolvidos.

Em segundo momento, os problemas surgiram nas decisões acerca do que seria enviado para o servidor e recebido pelo cliente. Houveram tentativas de fazer esse processo enviando e recebendo instâncias da classe Jogo, mas foram seguidas de muitos erros, de modo que simplificamos o processo através de *strings* enviadas pelo cliente e instâncias do jogo pelo servidor.

Por último, complicações na geração dos tabuleiros que representam os campos navais necessitaram de uma análise maior. O tabuleiro deve conter 7 barcos com comprimentos diferentes, que variam suas orientações entre horizontal e vertical, organizados de modo a não encostarem um ao outro. Tendo em mente que os tabuleiros dos campos navais devem seguir uma padronização específica, encontramos dificuldades em gerar os espaços necessários entre os navios para que não houvesse choques entre os barcos e que se mantivesse uma distância aceitável de no mínimo um quadrado de água. A solução para o problema foi alcançada ao criar cada tipo de barco no tabuleiro individualmente, em vez de agrupá-los todos em um único loop.

5. CÓDIGO FONTE

O código fonte está disponível no github, através do seguinte link: https://github.com/ThiagoORuby/Network-Battleship-Pygame. Apenas os arquivos .py principais estão anexados a esse relatório.

server.py

```
import socket
from _thread import *
import pickle
from components import *
from constants import ADDRESS, PORT, MAX_CLIENTS

# initiliaze the server socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    s.bind((ADDRESS, PORT))
except socket.error as e:
    str(e)

s.listen(MAX_CLIENTS)
print("Waiting for a connection, Server Started")

games = {} # list of Game instances
idCount = 0 # num of lcinets connected

# Start a thread for a client
def threaded_client(conn, p, gameId):
    global idCount
    conn.send(str.encode(str(p)))

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

            if gameId in games:
                game = games[gameId]

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

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

    print("Lost connection")
    try:
        del games[gameId]
        print("Closing Game...", gameId)
```

```
    except:
        pass
    idCount -= 1
    conn.close()



while True:
    conn, addr = s.accept()
    print("Connected to:", addr)

    idCount += 1
    p = 0
    gameId = (idCount - 1)//2

    if idCount % 2 == 1:
        games[gameId] = Game(gameId)
        games[gameId].players[p] = Player()
        games[gameId].current_player_id = p
        print("Creating a new game...")
    else:
        p = 1
        games[gameId].players[p] = Player()
        games[gameId].ready = True


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

client.py

```
import pygame
from network import Network
from constants import *
from components import *
from utils import *
from math import floor
import time

# Initialize Pygame
pygame.init()
win = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
pygame.display.set_caption("Client")

# Redraw Game Window
def redrawWindow(win, game, p):
    win.fill((0,0,0))

    if not(game.connected()):
        animation_time = int(time.time() * 2)
        dots_count = (animation_time % 4)
        dots = '.' * dots_count
```

```
        font = pygame.font.SysFont("assets/font.ttf", 28)
        text = font.render("WAITING FOR PLAYER" + dots, 1, (255,255,255), True)
        win.blit(text, (WINDOW_WIDTH/2 - text.get_width()/2, WINDOW_HEIGHT/2 -
text.get_height()/2))
    else:
        win.blit(radar_map, (12,54))
        win.blit(ocean_map, (12*2 + 342,54))

        draw_grid(win, game.players[p].grid)

        font = pygame.font.SysFont("assets/font.ttf", 20)

        if game.current_player_id == p:
            info_text = f"YOUR TURN. YOU HAVE {NUM_SHOOTS - game.shoot_count} SHOOTS!"
        else:
            info_text = f"PREPARE FOR THE OPPONENT'S SHOOTS"

        text = font.render(info_text, 1, (255, 255,255))
        text_rect = text.get_rect()
        text_rect.center = (WINDOW_WIDTH // 2, WINDOW_HEIGHT // 12)
        win.blit(text, text_rect)

        round_text = font.render(f"ROUND {(game.battle_count) // 2}", 1,
(255,255,255))
        round_text_rect = round_text.get_rect()
        round_text_rect.center = (WINDOW_WIDTH // 2, WINDOW_HEIGHT // 20)
        win.blit(round_text, round_text_rect)


        draw_ocean_mask(win, game.players[p].mask)
        draw_radar_mask(win, game.players[1 - p].mask)

    pygame.display.update()

# Load the Main Game Screen
def main():
    run = True
    clock = pygame.time.Clock()
    n = Network()
    player = int(n.getP())
    print("You are player", player)

    while run:
        clock.tick(60)
        try:
            game = n.send("get")
        except:
            run = False
            print("Couldn't get game")
            break

        if game.connected() and game.winner() != -1:
```

```
            redrawWindow(win, game, player)
            pygame.time.delay(500)

            font = pygame.font.SysFont("assets/font.ttf", 70)
            if (game.winner() == 1 and player == 1) or (game.winner() == 0 and player
== 0):
                text = font.render("YOU WON!", 1, (255, 255, 255))
            else:
                text = font.render("YOU LOST...", 1, (255, 255, 255))

            text_rect = text.get_rect()
            text_rect.center = (WINDOW_WIDTH // 2, WINDOW_HEIGHT // 2)
            pygame.draw.rect(win, (0,0,0), (0, WINDOW_HEIGHT // 2 - 40, WINDOW_WIDTH,
80))

            win.blit(text, text_rect)
            pygame.display.update()

            try:
                game = n.send("reset")
            except:
                run = False
                print("Couldn't get game")
                break
            pygame.time.delay(3000)

        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()
                if RADAR_GRID_X < pos[0] < RADAR_GRID_X + GRID_WIDTH - TILE_SIZE and \
                    OCEAN_GRID_Y < pos[1] < OCEAN_GRID_Y + GRID_HEIGHT - TILE_SIZE:
                    pos_grid = [floor(abs(pos[i] - RADAR_GRID_POS[i])//(TILE_SIZE -
1)) for i in range(2)]
                    data = f"{pos_grid[0]},{pos_grid[1]}"
                    if game.current_player_id == player and game.connected():
                        n.send(data)

        redrawWindow(win, game, player)


# Load the Menu Screen
def menu_screen():
    run = True
    clock = pygame.time.Clock()

    blinking_counter = 0
```

```
    while run:
        clock.tick(60)
        win.fill((0,0,50))
        win.blit(titleScreen, (0,0))

        font = pygame.font.Font("assets/font.ttf", 18)
        text = font.render("PRESS SPACE TO PLAY", 1, (255,255,255))
        text_rect = text.get_rect()
        text_rect.center = (WINDOW_WIDTH // 2, 3 * (WINDOW_HEIGHT) // 4)

        if blinking_counter % 60 < 20:
            win.blit(text, text_rect)

        blinking_counter += 1
        pygame.display.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                run = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    gameSounds['play'].set_volume(0.05)
                    gameSounds['play'].play()
                    run = False

    main()

if __name__ == '__main__':
    while True:
        menu_screen()
```

network.py

```
import socket
import pickle
from constants import ADDRESS, PORT

class Network:
    """Class that represents a client conection with the server
    """

    def __init__(self):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.addr = (ADDRESS, PORT)
        self.p = self.connect()

    def getP(self):
        """Get the player id

        Returns:
```

```
            int: Player id
        """
        return self.p

    def connect(self):
        """Connect to the server

        Returns:
            int: Player id of the client sent by server
        """
        try:
            self.client.connect(self.addr)
            return self.client.recv(2048).decode()
        except:
            pass

    def send(self, data):
        """Send data to the server and receive the response

        Args:
            data (str): Client status

        Returns:
            Game : The Game instance sent by the server
        """
        try:
            self.client.send(str.encode(data))

            return pickle.loads(self.client.recv(2048*3))
        except socket.error as e:
            print(e)
```