

# Trabalho Prático - Redes de Computadores

## Jogo “21” (BlackJack) com Sockets UDP

**Aluno:** Gabriel Rodrigo dos Santos Miguel

**Repositório Github:** <https://github.com/Gabriel-GR1/projeto-blackjack/tree/main>

### 1. Introdução

Este trabalho prático tem como objetivo consolidar o conhecimento na matéria de redes de computadores através da implementação de um jogo de cartas utilizando comunicação via sockets UDP. O desafio proposto foi criar uma versão funcional do jogo de cartas “21” (Blackjack), com gerenciamento de múltiplos jogadores e troca de mensagens controladas.

A aplicação foi desenvolvida em Python, por conta da facilidade de manipular sockets e pela clareza de estrutura, sem bibliotecas externas. A proposta abrange os requisitos mínimos exigidos, com foco em comunicação UDP, gerenciamento de múltiplos jogadores e troca de mensagens controladas.

### 2. Desenvolvimento

#### 2.1 Arquitetura da Aplicação

A arquitetura foi dividida em dois componentes principais:

- **Servidor UDP:** Responsável pela lógica do jogo, sorteio de cartas, controle de pontuação e envio de mensagens.
- **Cliente UDP:** Gerencia a entrada do jogador, permite interação via terminal e recebe feedback do servidor.

A troca de mensagens segue um protocolo textual simples e padronizado.

#### 2.2 Protocolos de comunicação

As mensagens trocadas entre cliente e servidor são compostas por comandos e parâmetros separados por dois-pontos (:). Os principais comandos são:

Comando	Origem → Destino	Significado
ENTRAR:<nome>	Cliente → Servidor	Jogador entra na sala com seu nome
PEDIR_CARTA	Cliente → Servidor	Solicita uma nova carta
CARTA:<valor>	Servidor → Cliente	Resposta com valor da carta sorteada
PARAR	Cliente → Servidor	Jogador decide parar e aguarda resultado
MENSAGEM:<texto>	Servidor → Cliente	Informações gerais e feedback textual
RESULTADO:ganhou	Servidor → Cliente	Jogador venceu a rodada
RESULTADO:perdeu	Servidor → Cliente	Jogador perdeu a rodada
RESULTADO:empate	Servidor → Cliente	Rodada terminou em empate com outro jogador

## 2.3 Fluxo do jogo

1. O jogador inicia o cliente e informa seu nome.
2. O servidor responde com uma mensagem de boas-vindas.
3. A cada turno, o jogador pode:
  - a. **Pedir uma carta**, acumulando pontos.
  - b. **Parar**, congelando sua pontuação.
4. Se ultrapassar 21 pontos, o jogador perde automaticamente.
5. Quando todos os jogadores estiverem inativos (pararam ou perderam), o servidor:
  - a. Verifica a maior pontuação válida ( $\leq 21$ ).
  - b. Envia o resultado (ganhou, perdeu ou empate) para cada cliente.
6. A rodada termina e o jogo é reiniciado, aceitando novos jogadores.

## 2.4 Funcionalidades Implementadas

### Servidor:

- Gerencia múltiplos jogadores via UDP.
- Sorteia cartas aleatórias e controla pontuação.

- Avalia condição de vitória, derrota ou empate.
- Envia mensagens estruturadas conforme protocolo.

#### Cliente:

- Permite entrada com nome.
- Exibe pontuação parcial após cada carta.
- Permite interações (pedir carta/parar).
- Exibe o resultado da rodada.

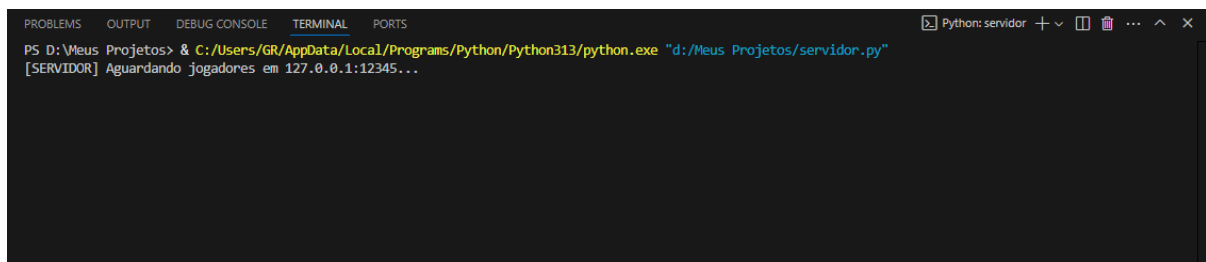
## 2.5 Decisões de Projeto

- **Linguagem escolhida:** Python 3.13.5, por ser leve, portátil e ter suporte nativo a sockets.
- **Protocolo UDP:** Escolhido conforme exigência do trabalho, exigindo controle manual de entrega e tratamento de estados.
- **Protocolo textual:** Ssimples, legível e facilmente interpretado pelos sockets UDP.
- **Sem bibliotecas externas:** Tudo implementado com a biblioteca padrão do Python.
- **Reinício automático:** Após o fim da rodada, o servidor limpa os dados dos jogadores e espera por novos, permitindo novas partidas sem reiniciar o servidor.

## 3. Resultados

O sistema foi testado com múltiplos jogadores simultaneamente em terminais diferentes. Todos os testes apresentaram o comportamento esperado.

Resposta do Servidor ao ser executado:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: servidor + - [ ] ... ^ x
PS D:\Meus Projetos> & C:/Users/GR/AppData/Local/Programs/Python/Python313/python.exe "d:/Meus Projetos/servidor.py"
[SERVIDOR] Aguardando jogadores em 127.0.0.1:12345...
```

Resultado 1: O cliente estoura a quantidade de pontos

```
PS D:\Meus Projetos> & C:/Users/GR/AppData/Local/Programs/Python/Python313/python.exe "d:/Meus Projetos/cliente.py"
Digite seu nome para entrar no jogo: Gabriel
MENSAGEM:Bem-vindo, Gabriel! Use PEDIR_CARTA ou PARAR.

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:6
● Sua pontuação atual: 6

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:9
● Sua pontuação atual: 15

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:9
● Sua pontuação atual: 24

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 2
⌛ Aguardando resultado final...
● RESULTADO:perdeu
⚡ Rodada finalizada.
PS D:\Meus Projetos> |
```

Resultado 2: O cliente ganha a partida

```

PS D:\Meus Projetos> & C:/Users/GR/AppData/Local/Programs/Python/Python313/python.exe "d:/Meus Projetos/cliente.py"
Digite seu nome para entrar no jogo: Gabriel
MENSAGEM:Bem-vindo, Gabriel! Use PEDIR_CARTA ou PARAR.

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:6
● Sua pontuação atual: 6

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:7
● Sua pontuação atual: 13

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:7
● Sua pontuação atual: 20

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 2
⚡ Aguardando resultado final...
● MENSAGEM:Você parou. Aguardando os outros...
⚡ Esperando os outros jogadores pararem...
⚡ Esperando os outros jogadores pararem...
⚡ Esperando os outros jogadores pararem...
⚡ Esperando os outros jogadores pararem...
⚡ Esperando os outros jogadores pararem...
● RESULTADO:ganhou
⚡ Rodada finalizada.

```

### Resultado 3: O cliente perde a partida

```

PS D:\Meus Projetos> & C:/Users/GR/AppData/Local/Programs/Python/Python313/python.exe "d:/Meus Projetos/cliente.py"
Digite seu nome para entrar no jogo: Luis
MENSAGEM:Bem-vindo, Luis! Use PEDIR_CARTA ou PARAR.

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:3
● Sua pontuação atual: 3

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:3
● Sua pontuação atual: 6

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:9
● Sua pontuação atual: 15

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 1
● CARTA:8
● Sua pontuação atual: 23

Escolha uma ação:
1 - Pedir carta
2 - Parar
Opção: 2
⚡ Aguardando resultado final...
● RESULTADO:perdeu
⚡ Rodada finalizada.

```

### Resultado 4: Empate

<pre>Digite seu nome para entrar no jogo: Luis MENSAGEM:Bem-vindo, Luis! Use PEDIR_CARTA ou PARAR.  Escolha uma ação: 1 - Pedir carta 2 - Parar Opção: 1 ● CARTA:6 ● Sua pontuação atual: 6  Escolha uma ação: 1 - Pedir carta 2 - Parar Opção: 2 ⌚ Aguardando resultado final... ● MENSAGEM:Você parou. Aguardando os outros... ● RESULTADO:empate ❖ Rodada finalizada.</pre>	<pre>Digite seu nome para entrar no jogo: Gabriel MENSAGEM:Bem-vindo, Gabriel! Use PEDIR_CARTA ou PARAR.  Escolha uma ação: 1 - Pedir carta 2 - Parar Opção: 1 ● CARTA:6 ● Sua pontuação atual: 6  Escolha uma ação: 1 - Pedir carta 2 - Parar Opção: 2 ⌚ Aguardando resultado final... ● MENSAGEM:Você parou. Aguardando os outros... ● RESULTADO:empate ❖ Rodada finalizada.</pre>
--	--

## 4. Conclusão

Este trabalho permitiu a aplicação prática dos conceitos de redes de computadores, como comunicação com sockets, controle de estado e protocolo de mensagens. Mesmo usando UDP — um protocolo sem garantia de entrega — foi possível construir um jogo funcional e robusto com controle de inatividade e múltiplas rodadas. O projeto cumpriu todos os requisitos e implementou funcionalidades bônus, demonstrando domínio da linguagem, estruturação de rede e lógica de aplicação. A reinicialização dinâmica do servidor reforçou a confiabilidade e flexibilidade da aplicação.

## 5. Observações

- Não foram utilizadas bibliotecas externas.
- Nenhuma ferramenta de geração de código foi utilizada.
- Todo o código foi escrito manualmente e testado em ambiente local.
- A lógica de empate foi implementada seguindo as **regras oficiais do Blackjack**.

