



universidade
de aveiro

Comunicação entre Aplicações Cliente - Servidor

Universidade de Aveiro

Rui Machado
João Vieira

Comunicação entre Aplicações

Cliente - Servidor

Departamento de Electrónica Telecomunicações e Informática

Universidade de Aveiro

Rui Machado, N^o Mec.: 65081

ruimmachado@ua.pt

João Vieira, N^o Mec.: 50458

joaopvieira@ua.pt

Resumo

O presente relatório aborda a comunicação entre aplicações seguindo um modelo Cliente-Servidor, usando conceitos abordados no decorrer da Unidade Curricular de Laboratórios de Informática, nomeadamente programação de sockets TCP, Criptografia, Documentação JSON, programação em Python, etc... Este trabalho é relevante pois com o avanço tecnológico actual, a aprendizagem dos conceitos anteriormente referidos torna-se imperativa para os estudantes de cursos ligados à tecnologia de informação.

O projecto foi realizado por duas pessoas, tendo como objectivo o desenvolvimento das capacidades de trabalho em grupo, desenvolvimento e estruturação de código, realização de testes funcionais e planeamento através da plataforma **code.ua.pt**.

Foi possível implementar com sucesso um modelo de comunicação entre aplicações Cliente-Servidor em género "Jogo High-Low" com funcionalidade de segurança (encriptação) de forma simples, prática e eficaz, conseguindo-se corrigir todos os erros encontrados.

Índice

1	Introdução	1
1.1	Ferramentas	1
1.2	Regras	2
2	Implementação	3
2.1	Socket	4
2.2	Segurança	15
3	Testes de Funcionalidade e Resultados	17
3.1	Invocação das aplicações	17
3.2	Funcionalidade do Cliente	20
3.3	Testes do Servidor	24
3.4	Análise de Resultados	26
4	Conclusão:	27

Lista de Figuras

1	Exemplo de conexão de um Cliente ao Servidor.	3
2	Código inicial do Servidor.	4
3	Utilização da lista nicknames[] para impressão de mensagens internas.	5
4	Cliente envia mensagem	5
5	Função new_msg()	6
6	Função start_client()	6
7	Função guess_client()	7
8	Função find_client_id()	7
9	Função quit_client()	7
10	Tratamento de erros de argumentos.	8
11	Tratamento da exceções da função main.	9
12	Mensagem de boas-vindas e variável global userID.	10
13	Função main de cliente.	10
14	Funções de suporte: opExit, opQuit e retry_game	12
15	Função de suporte: validate_response	12
16	Função de suporte: initQuestion	13
17	Função de suporte: opGuessing	13
18	Função run_client - Parte 1	14
19	Função run_client - Parte 2	15
20	Funções de segurança: Servidor.	16
21	Funções de segurança: Cliente.	16
22	Exemplo iniciação de Servidor	17
23	Erros de comando	18
24	Erros de comando	18
25	Fechar o servidor correctamente	19
26	Fechar o servidor de forma errada	19
27	Disponibilizar porta via comando terminal linux.	20

28	Servidor com 10 clientes conectados ao mesmo tempo, com ID diferente uns dos outros . . .	20
29	Servidor: Actividade recente dos jogadores	21
30	Cliente muda ou não muda de id	21
31	Escolha de um novo ID similar a outro ID já registado	22
32	Funcionamento da operação <i>Guess</i>	22
33	Simulaçãp de jogo completo. Cliente escolhe se pertende jogar novamente	23
34	Funcionamento da operação <i>Quit</i>	24
35	Operação <i>Guess</i> enviada por um Cliente não activo	24
36	Operação <i>Quit</i> enviada por um Cliente não activo	25
37	Operação <i>Stop</i> enviada por um Cliente não activo	25
38	Incoerência entre tentativas registadas pelo Servidor e Cliente	25
39	Cliente envia operação inexistente ao Servidor	26

Acrónimos

MIECT Mestrado Integrado em Engenharia de Computadores e Telemática

UC Unidade Curricular

LABI Laboratórios de Informática

AF_INET Address Family INET

IPv4 Internet Protocol version 4

SOCKET_STREAM Stream Sockets

TCP Transmission Control Protocol

AES-128 Advanced Encryption Standard - 128 bits key

ECB Electronic CodeBook

UTF-8 8-bit Unicode Transformation Format

JSON JavaScript Object Notation

Capítulo 1

Introdução

Devido ao grande avanço tecnológico das últimas décadas, a internet tornou-se uma ferramenta essencial para o funcionamento da sociedade. Sendo a internet um sistema global de redes de computadores interligados entre si usando um conjunto próprio de protocolos, torna-se fundamental o estudo da comunicação entre aplicações, já que esta é a base fundamental da web.

Assim, no decorrer do segundo semestre no âmbito da Unidade Curricular (UC) de Laboratórios de Informática (LABI) do Mestrado Integrado em Engenharia de Computadores e Telemática (MIECT), realizou-se este trabalho com o objetivo de aprender e aprofundar o estudo de diversos temas, nomea-

damente a programação em Python, criptografia, depuração, comunicação entre aplicações e documentos tipo CSV e JSON.

O presente trabalho surge no tema de desenvolvimento de uma aplicação Cliente que tenta adivinhar um número inteiro aleatório secreto (entre 0 e 100) gerado por um Servidor, usando protocolos e sockets API numa rede de computador.

Ao longo deste relatório será indicado e descrito o processo de implementação das aplicações, da sua protecção e segurança, os testes efectuados de modo a garantir o funcionamento pretendido e a análise de resultados obtidos.

1.1 Ferramentas

No desenvolvimento deste trabalho foram utilizadas algumas ferramentas e conhecimentos teóricos e práticos abordados no decorrer da Unidade Curricular (UC), LABI:

- *Sistema Operativo Linux*: Ubuntu Ver.20.04
- *Editor de texto Geany*: Editor de texto que suporta a linguagem de programação Python3 que irá ser usada para a elaboração deste trabalho.
- *Ficheiros Python*: Fornecidos pelos docentes da UC para inicialização do trabalho.

1.2 Regras

A aplicação Servidor–Cliente rege-se pelas seguintes normas:

O Servidor aceita vários Clientes simultaneamente desde que não tenham IDs iguais. Se um Cliente entrar com um ID igual a outro que esteja actualmente a jogar o Servidor deve negar o pedido de conexão.

Se a conexão for aceite, o Servidor deve gerar um número inteiro aleatório, entre 0 e 100, para o usuário adivinhar e também um número aleatório de tentativas, entre 10 e 30, que tem para o fazer.

O Servidor deve criar e actualizar um ficheiro de tipo csv que regista os resultados dos diversos Clientes quando estes terminam o jogo correctamente. O usuário pode terminar o jogo normalmente quando acerta no número (nesse caso o Servidor deve registar o jogo como sucesso) ou por esgotamento de tentativas (o jogo deve ser registado como fracasso).

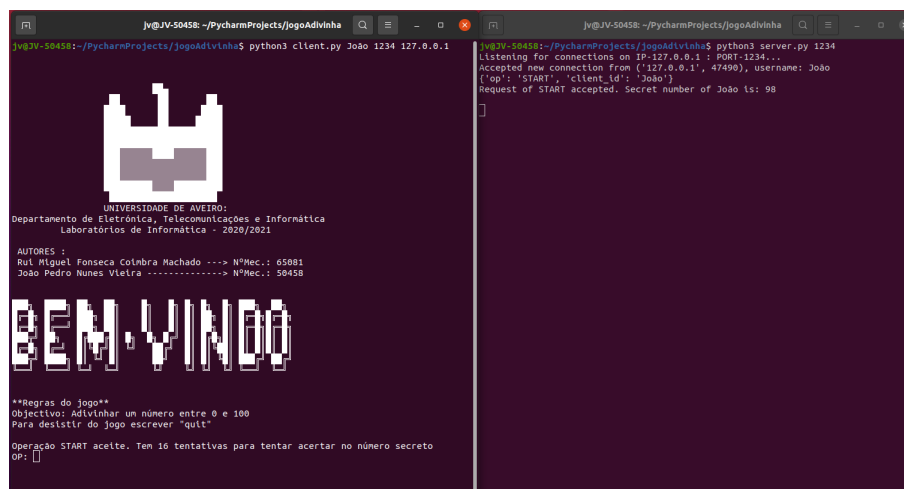
Além disso, o usuário pode desistir do jogo a qualquer momento com a operação *quit* (o Servidor regista o jogo como desistência). Sempre que o Cliente faz um pedido de operação ao Servidor deve receber alguma notificação a indicar se foi aceite ou não.

Nota: Decidiu-se fazer algumas alterações ao trabalho proposto de forma a explorar as possibilidades das ferramentas usadas. Nomeadamente, implementou-se a hipótese de o usuário mudar o seu ID quando recebe uma notificação do Servidor a indicar que o seu ID é igual a um já existente e, para além disso, também se adicionou no fim do jogo a possibilidade de o usuário jogar novamente caso assim queira.

Implementação

Tanto a aplicação-Cliente como a aplicação-Servidor são iniciadas usando uma interface de linha de comandos (Consola/terminal Linux ou Linha de Comandos CMD Microsoft) e invocando os comandos:

Por exemplo, um Cliente com `client_id` "João" conecta-se ao Servidor de porto 1234 com endereço 127.0.0.1 como ilustrado na figura 1.



O porto do servidor é definida pelo Servidor na sua iniciação, sendo o seu endereço pré-definido.

O usuário deve introduzir um identificador pessoal (`client_id`) que ainda não tenha sido escolhido por outro, como será demonstrado no capítulo 3

Uma vez aceite a conexão pelo Servidor, é fornecido ao Cliente um número máximo de tentativas para adivinhar o número secreto. Tanto o número secreto como o número máximo de tentativas são gerados aleatoriamente pelo Servidor, sendo atribuídos números diferentes a cada Cliente. Sempre que o usuário tenta adivinhar o número tanto o Servidor como o Cliente registam essa tentativa e o Servidor informa o usuário se o número introduzido é inferior (*smaller*) ou superior (*larger*) ao número secreto. O usuário pode então voltar a introduzir outro número tentando acertar no número secreto. O jogo acaba caso o usuário acerte no número secreto, recebendo a mensagem *equals*, ou caso esgote o seu número de tentativas, sendo-lhe dada a opção de jogar novamente. Além disso, o usuário pode desistir a qualquer momento através da operação *quit*.

2.1 Socket

Tanto o Cliente como o Servidor devem ter sockets de conexão estruturados com a mesma família, tipo e nome/interface.

A comunicação entre as aplicações é feita de forma local ou remota, sendo que, para este efeito, foram implementados sockets com protocolo de rede da família Address Family INET (AF_INET) (apenas suporta Internet Protocol version 4 (IPv4)) de tipo Stream Sockets (SOCKET_STREAM) (orientado à ligação) com um protocolo de transmissão Transmission Control Protocol (TCP).

Aplicação-Servidor

A base deste trabalho é a programação entre Sockets TCP. O código da aplicação-Servidor começa por criar uma socket da família IPV4 do tipo TCP com a instrução `socket`. Seguidamente, dá-se um nome ao socket com a instrução `bind`, nome esse que é composto por um endereço (IPv4 e porto) para os Clientes poderem estabelecer uma conexão. Finalmente, coloca-se o socket à escuta com o comando `listen()` para que esteja disponível para conexões. Além disso, chama-se a função `create_file` que cria um ficheiro do tipo csv para registar os resultados dos Clientes. O nome do ficheiro a ser criado é declarado no cabeçalho do código com uma variável global, a função `create_file`, e antes de criar um ficheiro certifica-se que não existe nenhum ficheiro com o mesmo nome no directório actual, só então cria um ficheiro novo.

```
IP = "127.0.0.1"
PORT = int(sys.argv[1])

# Create a socket # AF_INET - address family Ipv4 #SOCK_STREAM - TCP
server socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Sets REUSEADDR (as a socket option) to 1 on socket
server socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server socket.bind((IP, PORT))
server socket.listen(10)

#Criar ficheiro CSV se este ainda não existir
create_file()
```

Figura 2: Código inicial do Servidor.

Para que um Servidor possa atender vários Clientes ele necessita de ter uma lista para armazenar os sockets associados aos Clientes ativos. Essa lista é a lista `clients[]` declarada no cabeçalho do programa. Além dessa lista criou-se também uma lista `nicknames[]` com os IDs dos Clientes para o Servidor poder imprimir mensagens internas e ser mais fácil de o monitorizar. O método `select` permite ficar à escuta de informação de múltiplas fontes, indicando depois qual das fontes possui informação a ser consumida. Este método é usado neste trabalho apenas para ver que Clientes enviaram mensagens.

Sempre que um Cliente novo se conecta ao Servidor, este é adicionado à lista `clients[]` e é-lhe criado um endereço particular para que o Servidor consiga enviar mensagens só para ele. Além disso, o Cliente está programado para enviar automaticamente o seu ID, assim o Servidor recebe esse ID e adiciona-o à lista `nicknames[]`. A partir desse momento todas as mensagens internas imprimidas no Servidor usam esse ID de forma a facilitar a compreensão do Servidor

```
Accepted new connection from ('127.0.0.1', 43662), username: Daniel
{'op': 'START', 'client_id': 'Daniel', 'cipher': None}
Request of START accepted. Secret number of Daniel is: 23
{'op': 'QUIT'}
Daniel has disconnected
```

Figura 3: Utilização da lista `nicknames[]` para impressão de mensagens internas.

Sempre que um Cliente com conexão já estabelecida envia uma mensagem é invocada a função `new_msg()`, que lida com as mensagens. Se um Cliente terminar o programa por alguma razão, o Servidor elimina-o das listas `clients[]` e `nicknames[]` e também do dicionário `gamers[]`, que iremos falar mais à frente. Também fecha o endereço criado para esse Servidor.

```
# Or an existing client
else:
    # See if client sent a message
    if len(client sock.recv(1, socket.MSG_PEEK)) != 0:
        # client socket has a message
        new_msg(client sock)
    else: # Or just disconnected
        index = clients.index(client sock)
        client id = nicknames[index]
        print(f'{client id} has disconnected\n')
        clean_client(client sock)
        nicknames.remove(client id)
        clients.remove(client sock)
        client sock.close()
        break # Reiterate select
```

Figura 4: Cliente envia mensagem

Como dito anteriormente, a função `new_msg()` lida com as mensagens. Esta foi implementada para apenas lidar com mensagens dicionário no formato esperado (primeira chave do dicionário deve ser "op"). A função verifica a operação enviada pelo Cliente e, de acordo com a operação encontrada, encaminha o programa para as funções pretendidas. As opções de operação são: Start, Guess, Quit e Stop. Se receber uma mensagem no formato esperado, mas com uma operação fora das opções estabelecidas, retorna ao Cliente uma mensagem de erro. Inicialmente adicionou-se uma linha de código que imprime as mensagens recebidas do Servidor, com intenções de a apagar e apenas com o intuito de ajudar no desenvolvimento do projecto. No entanto, no fim decidiu-se mantê-la de forma a ser mais fácil verificar o funcionamento do programa, especialmente em modo Safe.

```
def new_msg(client sock):
    try:
        message = recv_dict(client sock)
        print(message) # teste

        if (message["op"] == "START"): start_client(client sock, message)
        elif (message["op"] == "GUESS"): guess_client(client sock, message)
        elif (message["op"] == "QUIT"): quit_client(client sock)
        elif (message["op"] == "STOP"): stop_client(client sock, message)

        #Operação Inexistente
        else:
            op = message["op"]
            opUnknown = {"op": op, "status": False, "error": "Operação inexistente"}
            send_dict(client sock, opUnknown)

    except:
        print("Erro a receber mensagem de cliente")
```

Figura 5: Função new_msg()

A função start_client() é invocada sempre que o Cliente envia a operação Start. Esta é a função que inicializa o jogo e começa por verificar se o Cliente que fez o pedido tem um ID válido (diferente de todos os jogadores activos). Se o Cliente tiver um ID válido são gerados aleatoriamente o número secreto e o número de tentativas e é iniciada a contagem de tentativas. Seguidamente adicionam-se os dados do Cliente ao dicionário Gamers. É através deste dicionário que a função start verifica se o Cliente tem ou não um ID válido e é também através dele que todas as outras funções verificam se as mensagens recebidas provêm de Clientes activos ou não. Se o Cliente for válido o Servidor envia uma mensagem ao Cliente a informar que o jogo iniciou, juntamente com o número máximo de tentativas. Se isto não for verdade o Servidor envia uma mensagem de erro.

```
def start_client(client sock, request):
    global safeMode

    client_id = request["client id"]
    cipherkey = request["cipher"]

    index = clients.index(client sock) #Código desnecessário ao funcionamento do programa
    nicknames[index] = client_id #apenas corrige mensagens internas do servidor

    if cipherkey != None: safeMode = True
    else: safeMode = False

    #Verifica que não existe um cliente com o mesmo ID
    if gamers.get(client_id) == None:
        secret number = random.randint(0, 100)
        max attempts = random.randint(10, 30)
        turns count = 0

        gamers.update({client_id: {'socket': str(client sock), 'cipher': cipherkey, 'guess': secret number, 'max attempts': max attempts, 'attempts': turns count}})

    if safeMode: max attempts = encrypt_intvalue(client_id, max attempts) #Safe Mode - encrypta o Max Attempts
    opStart = {"op": "START", "status": True, "max attempts": max attempts}
    send_dict(client sock, opStart)

    print(f'Request of START accepted. Secret number of {client_id} is: {str(secret number)}\n')

    #Existe um cliente com o mesmo ID
    else:
        print(f'Request of START from {client_id} wasn't accepted. The Client ID is already taken\n')
        opStart = {"op": "START", "status": False, "error": "Cliente existente"}
        send_dict(client sock, opStart)
```

Figura 6: Função start_client()

A função guess_client() é invocada sempre que o Cliente envia a operação Guess. Esta função deve ser utilizada sempre que o usuário faz uma tentativa de jogo. Inicialmente, a função verifica se o Cliente que fez o pedido está registado como um jogador activo. Para fazer isso é invocada a função find_client_id. Se o Cliente não estiver registado é-lhe enviada uma mensagem de erro. Caso contrário, o programa regista a tentativa incrementando o número de jogadas feitas associadas ao Cliente (através do dicionário gamers). Seguidamente, o Servidor compara o número adivinhado com o número secreto associado ao Cliente (através do dicionário gamers) e envia uma mensagem com o resultado dessa comparação(smaller/larger/equals)

```
def guess_client(client_sock, request):
    client_id = find_client_id(client_sock)

    # Cliente não está na lista de jogadores activos
    if gamers.get(client_id) == None:
        opGuess = {"op": "GUESS", "status": False, "error": "Cliente inexistente"}
        send_dict(client_sock, opGuess)

    #Funcionamento correcto
    else:
        gamers[client_id]['attempts'] += 1 #registar tentativa
        data = request["number"]

        if safeMode: attempt = decrypt_intvalue(client_id, data) # Safe Mode - Decifra o Number Guess
        else: attempt = int(data) # Standart Mode

        secret_number = gamers[client_id]['guess']

        if (attempt < secret_number): result = "smaller"
        elif (attempt > secret_number): result = "larger"
        else: result = "equals"

        opGuess = {"op": "GUESS", "status": True, "result": result}
        send_dict(client_sock, opGuess)
```

Figura 7: Função guess_client()

A função find_client_id() é chamada para obter um ID duma socket em específico. Esta função consiste em iterar todos os IDs registados no dicionário gamers e suas sockets verificando se existe uma socket igual à que fez o pedido. Se tal for o caso, a função retorna o ID pretendido, caso contrário envia None.

```
def find_client_id(client_sock):
    found = False

    for key in gamers:
        if gamers[key]['socket'] == str(client_sock):
            client_id = key
            found = True
            return client_id

    if not found: return None
```

Figura 8: Função find_client_id()

A função quit_client() é invocada sempre que o Cliente envia a operação Quit. Esta função deve ser utilizada sempre que o usuário pretende desistir do jogo e apenas verifica se o Cliente que fez o pedido estava registado na lista de jogadores activos. Nesse caso, actualiza o ficheiro csv com o resultado "QUIT", envia uma mensagem indicando que a operação teve sucesso e elimina o jogador do dicionário gamers. Se este não estava registado o Servidor envia uma mensagem de erro

```
def quit_client(client_sock):
    client_id = find_client_id(client_sock)

    # Cliente não está na lista de jogadores activos
    if gamers.get(client_id) == None:
        opQuit = {"op": "QUIT", "status": False, "error": "Cliente inexistente"}
        send_dict(client_sock, opQuit)

    #Funcionamento correcto
    else:
        opQuit = {"op": "QUIT", "status": True}
        send_dict(client_sock, opQuit)

        update_file(client_id, "QUIT")
        clean_client(client_sock)
```

Figura 9: Função quit_client()

A função `stop_client()` é invocada sempre que o Cliente envia a operação Stop. Esta função deve ser utilizada sempre que o usuário termina o jogo normalmente e começa por verificar se o Cliente que fez o pedido estava registado na lista de jogadores activos. Além disso, verifica se o número de tentativas registadas pelo Servidor é igual ao número de tentativas reportadas pelo Cliente. Se não encontrar nenhum problema, verifica se o último número jogado é igual ao número secreto. Se tal for verdade significa que o jogador ganhou e actualiza o ficheiro csv com o resultado "SUCESS". Caso contrário, significa que o jogo terminou por esgotamento de tentativas e neste caso o ficheiro é actualizado com o resultado "FAILURE".

Aplicação-Cliente

A iniciação da aplicação-Cliente é feita com a invocação da função `main()` (última linha de código do programa) com recurso à variável `__main__`, permitindo que o código Python seja executado directamente pelo Cliente como programa e não como módulo. Inicialmente, a função `main()` verifica os argumentos de entrada que o usuário inseriu para executar a sua conexão ao Servidor. Caso os argumentos inseridos estejam incorrectos o programa apresenta uma mensagem de erro e termina.

- *Cliente insere um número inválido de argumentos:* Para executar uma conexão comum o usuário terá sempre que inserir 4 argumentos, tal como demonstrado anteriormente na figura 1. Caso um destes argumentos não seja especificado a função `main` detecta o erro recorrendo a um ciclo `if`, comparando assim o número de argumentos inseridos (`len(sys.argv)`) com o número de argumentos esperados (4). Caso o número de argumentos inseridos seja diferente de 4 (`!=4`), o programa executa um `print` da mensagem **Erro1** e nega o acesso do Cliente ao Servidor através de `sys.exit()`, tal como demonstrado na figura. 10.
- *Cliente insere argumento para porto não numérico ou numérico não inteiro:* Neste caso, o Cliente indicou um valor de entrada não numérico ou numérico inteiro real para especificar o porto de conexão ao Servidor. Assim, o Servidor executa o `print` da mensagem **Erro2** e nega o acesso do Cliente ao Servidor através de `sys.exit()`, tal como demonstrado na figura 10.

```
#-----  
#Error Handling -- Argumentos no chamamento do terminal têm de estar correctos  
if len(sys.argv) != 4:  
    print("Erro 1: Numero inválido de argumentos\n")  
    sys.exit()  
      
if not sys.argv[2].isnumeric():  
    print("Erro 2: Valor do Port Inválido (argv[2]) -- Deve ser um número inteiro\n")  
    sys.exit()  
      
    
```

Figura 10: Tratamento de erros de argumentos.

Se não houve nenhum erro na invocação do programa, o Cliente usa os argumentos de entrada dados pelo usuário para definir as suas variáveis **PORT** e **IP** e criar a socket (variável *client_sock*) necessária à conexão usando a família `AF_INET`, orientado à ligação `SOCKET_STREAM` em protocolo TCP, tal como definido no Servidor: `client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`.

Posteriormente, tenta executar uma conexão válida usando as variáveis **PORT** e **IP** definidas anteriormente, verificando as exceções destas variáveis, tal que:

- *Exceção OverflowError*: Neste caso o usuário inseriu um porto cujo número é "out of range" do permitido, ie, o usuário deve inserir um número inteiro entre 1024 e 65535, pois o sistema tem apenas 65535 portos e os de valores inferiores a 1024 são portos já utilizados pelo sistema (**Well-Known Ports**). Assim, o sistema detecta a exceção e executa um `print` da mensagem **Erro2** onde indica que o valor do argumento 2 **PORT** é inválido, indicando os valores dentro do alcance permitido.
- *Exceção gaierror*: Neste caso, o usuário insere um valor **IP** inválido, sendo que o programa detecta o erro e executa um `print` da mensagem **Erro2** onde indica que o argumento 3 IP tem valor inválido.
- *ConnectionRefusedError*: Neste caso, o usuário inseriu todos os argumentos de entrada correctamente, contudo o valor d porto especificado não é o valor definido pelo Servidor, negando assim a conexão. Assim, o programa executa um `print` da mensagem **Erro2** onde indica que o porto especificado é incorrecto, pedindo ao utilizador que tente nova conexão com um porto válido (i.e. correcto).

```
#-----
#Main Function
try:
    PORT = int(sys.argv[2])
    IP = sys.argv[3]

    client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_sock.connect((IP, PORT))

    client_sock.send(userID.encode('utf-8'))
    regras = client_sock.recv(1024).decode('utf-8')

    print(welcome)
    print(regras)

    run_client(client_sock, userID)

    opExit(client_sock)

#-----
#Error Handling - Argumentos no chamamento do terminal têm de estar correctos
except gaierror:
    print("Erro 4: Valor do IP inválido (argv[3])\n")

except ConnectionRefusedError:
    print("Erro 5: Conexão falhada, tente outro valor Port (argv[2])\n")

except OverflowError:
    print("Erro 2: Valor do Port Inválido (argv[2]) - Deve ser um número entre 1024 e 65535\n")
#-----
```

Figura 11: Tratamento da exceções da função main.

Caso nenhum erro/exceção tenha sido detectado o cliente irá enviar o seu `userID` (variável global de argumento 1, ver figura 12) para servidor usando método `encode` em **utf8!** (**utf8!**) e recorrendo à função **send**. Posteriormente recebe as regras do jogo em 1024 caracteres, das quais irá executar `decode` em 8-bit Unicode Transformation Format (UTF-8). É feito um **print** de boas-vindas definido na variável global "welcome" e das regras do jogo recebidas do servidor e definidas na variável "REGRAS" da função **main()** do servidor (figura 12). Seguidamente é invocada a função `run_client` com os parametros `socket` (i.e. `client_sock`) e `userID` (figura 13).

```
#ClientID
userID = sys.argv[1]

#Apresentação dos autores
welcome = ("\\n\\n\\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "\\n                                     \\n"
           "UNIVERSIDADE DE AVEIRO:"
           "Departamento de Eletrônica, Telecomunicações e Informática"
           "\\n      Laboratórios de Informática - 2020/2021"
           "\\n \\n AUTORES : "
           "\\n Rui Miguel Fonseca Coimbra Machado ---> NºMec.: 65081"
           "\\n João Pedro Nunes Vieira -----> NºMec.: 50458"
           "\\n \\n \\n"
           "EEN-VINDO")
```

Figura 12: Mensagem de boas-vindas e variável global userID.

A mensagem de boas-vindas contém a identificação dos autores deste trabalho por forma a ser identificado pelo utilizador com facilidade.

```

#Main Function
try:
    PORT = int(sys.argv[2])
    IP = sys.argv[3]

    client sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client sock.connect((IP, PORT))

    client sock.send(userID.encode('utf-8'))
    regas = client sock.recv(1024).decode('utf-8')

    print(welcome)
    print(regas)

    run client(client sock, userID)

    opExit(client sock)

```

Figura 13: Função main de cliente.

*Nota: A operação **opExit(client_sock)** não tem qualquer influencia no código. Apenas foi mantida por estar presente nos ficheiros fornecidos pelos docentes da UC.*

Cliente: Funções de suporte

Durante a execução da função `run_clinet()` são usadas as seguintes funções (ilustradas nas figuras 14, 15):

- **Função `opExit(client_sock)`:** Quando invocada para o argumento de entrada `client_sock`, a função executa um `print` da mensagem de despedida (*Até à próxima*), fecha o socket do cliente (`client_sock`) e executa a operação de saída `sys.exit(0)`.
- **Função `validate_response(client_sock, response)`:** Verifica se a resposta do servidor é válida ou é uma mensagem de erro actual da seguinte forma. Se a resposta do servidor for válida (*True*) então a função não faz nada de concreto, permitindo a sua passagem e display. Contudo se a resposta do servidor não for válida (ie. *False*) a função faz o display da resposta bem como executa `print` de mensagens de erro ("Operação não aceite" e "Operação inválida") invocando a função `opExit` para o `client_sock`.
- **Função `opQuit(client_sock)`:** É utilizada para, como o nome indica, executar a saída do cliente do programa. Quando invocada, gera um dicionário `opQuit` que envia como pedido ao servidor e recebe a resposta de aceitação de pedido. Seguidamente faz invocação da função `validate_response` que valida a resposta do servidor e para finalizar executa a invocação da função `opExit(client_sock)`.
- **Função `retry_game(client_sock, client_id)`:** Esta função será usada pelo cliente para disponibilizar ao utilizador a opção de poder voltar a jogar novamente caso o jogo tenha terminado por vitória ou derrota (SUCCESS/FAILURE). Assim, quando invocada, faz uso de um ciclo `while True`, no qual faz um `Print` com uma pergunta ("Jogar Novamente?") e instruções para a resposta. Seguidamente, cria uma variável `replay` que requer um input por parte do utilizador, que ignora letras minúsculas transformando-as em letras maiúsculas para serem usadas como factor de selecção de resposta. Recorrendo a ciclo `if`, a resposta é tratada de diferentes formas:
 - **Cliente responde "Y":** Se o cliente responder "Y", será invocada novamente a função `run_client` usando como parâmetros de entrada `client_sock` e `client_id`, fazendo assim um novo jogo para o cliente.
 - **Cliente responde "N":** Se o cliente responder "N", será executado um `print` com uma mensagem de despedida e é invocada a função `opExit`.
 - **Cliente responde algo inválido:** Se o cliente responder algo que não seja "Y" ou "N", é executado um `print` com uma mensagem de erro (*Resposta Inválida*) e o cliente terá que executar novo input com uma resposta correcta (Y/N).

```

# -----
# process EXIT
def opExit(client sock):
    print("Até à próxima")
    client sock.close()
    sys.exit(0)

# -----
#process RETRY
def retry game(client sock, client id):
    while True:
        print("\nQuer jogar novamente? (Y/N)?")
        replay = input("").upper()
        if replay == "Y":
            run client(client sock, client id)
        elif replay == "N":
            print("\nFoi um prazer jogar consigo!")
            opExit(client sock);
        else:
            print("Resposta Inválida")

# -----
# process OP. QUIT
def opQuit(client sock):
    opQuit = {"op": "QUIT"}
    inMSG = sendrecv dict(client sock, opQuit)
    validate response(client sock, inMSG)
    print("Operação QUIT aceite.")
    opExit(client sock)

```

Figura 14: Funções de suporte: opExit, opQuit e retry_game .

```

# -----
# verify if response from server is valid or is an error message and act accordingly
def validate response(client sock, response):
    if response['status'] == False:
        op = response['op']
        print(response)
        print(f'Operação {op} não foi aceite!')
        print('Erro 3: Operação inválida -.' + response['error'])
        opExit(client sock)

```

Figura 15: Função de suporte: validate_response .

Finalmente, é descrita a implementação das funções **initQuestion** e **opGuessing**, cujas ilustrações se encontram nas figuras 16, 17.

- Função **initQuestion()**: É usada para perguntar ao usuário o modo de inicialização que deseja utilizar. Assim, recorrendo a um ciclo **while True** a função recorre a um **print** fazendo display de uma questão de escolha de modo de iniciação, sendo que seguidamente define uma variável **response** que requer um input por parte do utilizador (que ignora letras minúsculas transformando-as em letras maiúsculas) para serem usadas como factor de selecção de resposta. Recorrendo a ciclo **if**, a resposta é tratada de diferentes formas:
 - **Cliente responde "Y"**: Se o cliente responder "Y", a função irá fazer um display de mensagem a confirmar a inicialização de modo de segurança e define a variável global **safeMode** com valor boolean **True**.
 - **Cliente responde "N"**: Se o cliente responder "N", a função irá fazer um display de mensagem a confirmar a inicialização de modo de standard (não seguro) e define a variável global **safeMode** com valor boolean **False**.
 - **Cliente responde algo inválido**: Se o cliente responder algo que não seja "Y" ou "N", é ser executado um **print** com uma mensagem de erro (**Resposta Inválida**) e o cliente terá que executar novo input com uma resposta correcta (Y/N).
- Função **opGuessing(client_sock, clientInput)**: Esta função processa a opGuess. Inicialmente define as variáveis globais **turnCount**, **max_attempts** e **safeMode**, definindo posteriormente a variável **sucess** com valor boolean **False**, a qual vai ser usada como forma de verificar se a escolha do número

do utilizador foi a correcta (igual a número secreto) ou não.

Assim, recorrendo a ciclos `if` a função irá triar a informação da seguinte forma:

- **Cliente escolhe um número inferior a 0 ou superior a 100:** Neste caso a função faz o display de mensagem: *A operação 'Guess' só aceita valores entre 0 e 100. Tente outra vez..* Assim, o cliente é forçado a escolher novamente um numero que esteja dentro dos limites permitidos/desejados pelo jogo.
- **Cliente escolhe um número adequado:** A função gera um dicionário `opGuess` que envia como pedido ao servidor e recebe a resposta de aceitação de pedido e faz invocação da função `validate_response` que valida a resposta do servidor. Posteriormente faz uma adiciona mais uma unidade à contagem de tentativas (`turnCount` e faz retira o resultado da mensagem do servidor (variável `inMSG`) e executa o display do resultado obtido (`result`)), da contagem de tentativas (`turncount`) e do máximo de tentativas que o utilizador tem (`max_attempts`).
 - * **Tratamento de dados (modo de segurança:** Os dados serão colocados numa variável `data` e cifrados pela função `encrypt_intvalue` usando como paramentos uma chave de cifra (`cipherkey`) gerada automaticamente e os seu número escolhido (`clientInput`). A função `encrypt_intvalue` será discutida mais à frente na secção 2.2.
 - * **Tratamento de dados (modo standard:** Os dados serão simplesmente aceites e enviados tal como estão, sendo colocados numa variável `data`.
- **Cliente acertou no número secreto:** Se o cliente acertar no número secreto (`result == "equals"`), a função altera o valor booleano da sua variável `"sucess"` para **True**.

```
# -----
# Question Safe Mode
def initQuestion():
    #Ciclo While para perguntar ao usuário o Modo de inicialização
    while True:
        print("\nDeseja executar o jogo em modo Seguro?(Y/N)")
        response = input("Resposta:").upper()
        #Cliente quer entrar com modo de Segurança
        if response == "Y":
            print("A aplicação vai ser inicializada em modo de Segurança\n")
            safeMode = True
            break
        #Cliente quer entrar com modo Não-Seguro
        elif response == "N":
            print("A aplicação vai ser inicializada em modo Standart(Não Seguro)\n")
            safeMode = False
            break
        #Resposta inválida
        else:
            print("Resposta inválida\n")
    return safeMode
# -----
```

Figura 16: Função de suporte: `initQuestion` .

```
# -----
# Process OP GUESS
def opguessing(client sock, clientInput):
    global turnCount
    global max_attempts
    global safeMode
    sucess = False

    # Aplicação só aceita valores dentro dos parametros
    if (int(clientInput) < 0 or int(clientInput) > 100):
        print("A operação 'Guess' só aceita valores entre 0 e 100\nTente outra vez.\n")
    # Função GUESS
    else:
        if safeMode: data = encrypt_intvalue(cipherkey, int(clientInput)) # Safe Mode - Encripta o Number Guess
        else: data = int(clientInput) #Standard Mode
        opguess = {'op': 'GUESS', 'number': data}
        inMSG = sendrecv dict(client sock, opguess)
        validate_response(client sock, inMSG)

        #Registrar tentativa
        turnCount += 1

        result = inMSG['result']
        if result == "equals": sucess = True
        print(f'Resultado: {result}')
        print(f'Jogadas feitas: {str(turnCount)}\{str(max_attempts)}\n')
    return sucess
# -----
```

Figura 17: Função de suporte: `opGuessing` .

Cliente: Função run_client

A função **run_client** tem como argumentos de entrada o `client_sock` e o `client_id`. Após ser invocada pela função **main()**, a função define variáveis globais `turnCount`, `max_attempts`, `safeMode`, `cipherkey` definindo posteriormente as variáveis `skip`, `sucess` (ambas com valores boolean **False**) e `turnCount = 0`. Posteriormente usa a variável `safeMode` para invocar a função **initQuestion()**, por forma a dar ao utilizador a opção de escolher entre *modo standard* ou *modo de segurança*. Se for escolhido *modo de segurança* (`safeMode == True`) será gerada uma `cipherkey` aleatória de 16 bits a qual será posteriormente usada para cifrar os dados comunicados entre Cliente e Servidor, sendo criada uma variável `cipherkey_send` que será codificada em base64 (para ser compatível com JavaScript Object Notation (JSON)). Seguidamente o Cliente irá executar a função **opStart** da seguinte forma:

- **Modo de Segurança:** A operação **opStart** será executada enviando ao Servidor, para além do `client_id`, a `cipherkey_tosend` gerada anteriormente.
- **Modo Standard:** A operação **opStart** será executada enviando ao Servidor apenas o `client_id` tendo o valor da cifra *None*.

Nota: Em todas as etapas da função run_client, caso o utilizador tenha optado pelo modo de segurança, os dados comunicados entre as aplicações Cliente-Servidor serão cifrados no envio e decifrados na recepção, por forma a obter uma conexão segura.

```
# -----
# Suporte da execução do cliente
def run_client(client_sock, client_id):
    global turnCount
    global max_attempts
    global safeMode
    global cipherkey
    skip = False
    sucess = False
    turnCount = 0

    safeMode=initQuestion()
    if safeMode == True:
        cipherkey = os.urandom(16) # Parametro de cifra 1 : Chave / KEY 16 bits aleatória
        cipherkey_tosend = str(base64.b64encode(cipherkey), 'utf-8')

    #Ciclo While para dar a oportunidade ao usuário de corrigir o ID
    while not skip:
        if safeMode: opStart = {'op': 'START', 'client_id': client_id, 'cipher': cipherkey_tosend} #Safe Mode - Envia cifra
        else: opStart = {'op': 'START', 'client_id': client_id, 'cipher': None} #Standard Mode - Não envia cifra
        START = sendrecv_dict(client_sock, opStart)

        # Operação Start foi aceite pelo Servidor
        if START['status'] == True:
            max_attempts = START['max_attempts']
            if safeMode: max_attempts = decrypt_intvalue(cipherkey, max_attempts) #Safe Mode - Decifra Max Attempts
            print(f'Operação START aceite. Tem {max_attempts} tentativas para tentar acertar no número secreto')
            skip = True

        # Operação Start não foi aceite pelo Servidor. Perguntar ao cliente se quer mudar ID
        else:
            while True:
                print(f'Erro: Já existe um cliente com esse ID. Deseja mudar ID? (Y/N)?')
                response = input('Resposta:').upper()
                # Nova tentativa de Start
                if response == 'Y':
                    print('Novo ID:')
                    client_id = input('')
                    print('Y')
                    break
            # Cliente não quer prosseguir
            elif response == 'N':
                opExit(client_sock)
            #Resposta inválida
            else:
                print('Resposta inválida')
```

Figura 18: Função run_client - Parte 1 .

```

#Ciclo Main - Jogo foi inicializado
while True:
    try:
        clientInput = input("OP: ").lower()
        # Input é número, assume que é uma tentativa/op:Guess
        if clientInput.isnumeric():
            success = opGuessing(client sock, clientInput)
            # Função QUIT
            elif clientInput == "quit":
                opQuit(client sock)
            # Input inválido, imprime mensagem de erro
            else:
                print("Operação Inválida!\n")

        # Função STOP
        if turnCount >= max_attempts or success == True: # PERDEU POR ESGOTAR TENTATIVAS / ACERTOU NO NÚMERO
            if safeMode: #Safe Mode
                clientInput = encrypt_intvalue(cipherkey, int(clientInput)) #Encrypta clientInput
                turnCount = encrypt_intvalue(cipherkey, turnCount) #Encrypta turnCount
            opStop = {"op": "stop", "number": clientInput, "attempts": turnCount}
            inMSG = sendrecv_dict(client sock, opStop)
            validate_response(client sock, inMSG)
            secret number = inMSG["guess"]
            if safeMode: secret number = decrypt_intvalue(cipherkey, secret number) # Safe Mode - Decifra secret number
            print(f"O jogo acabou, O número secreto era {secret number}")
            if success: print("Você ACERTOU")
            else: print("Você esgotou as tentativas")
            retry_game(client sock, client id)

        # Error Handling
        except IOError as e:
            print(f"Reading error1: {str(e)}")
            sys.exit()
        except Exception as e:
            print(f"Reading error2: {str(e)}")
            sys.exit()

```

Figura 19: Função run_client - Parte 2 .

2.2 Segurança

A transmissão entre Cliente (pedidos) e Servidor (respostas), são cifrados por cifras simétricas por blocos Advanced Encryption Standard - 128 bits key (AES-128) em modo Electronic CodeBook (ECB).’

Implementação

Nesta secção iremos abordar as funções de segurança implementadas nas aplicações Cliente-Servidor:

Server:

No servidor, existem três funções implementadas para a segurança: **encrypt_intvalue**, **decrypt_intvalue** e **find_cipher**. A implementação destas funções será abeguidamente e pode ser visualizada na figura 20:

- **Função find_cipher:** Serve para encontrar a (*cipherkey*) do cliente com determinado userID, ie., a função faz uma busca no dicionário gamers pela referencia *cipher* para obter a cipherkey correspondente ao *client_id* do cliente em questão. Posteriormente faz o decode() de base64 e obtém a cipherkey pronta a ser usada para criar a cifra (*cipher* = AES.new(cipherkey, mode)) em modo ECB (mode = AES.MODE_ECB) e propagar esta através do **return**.
- **Função encrypt_intvalue(client_id, data):** De argumentos de entrada *client_id* e *data*, esta função obtém a cifra do cliente invocando a função **find_cipher** com o *client_id* pretendido. Seguidamente, os dados (variável *data*) são cifrados numa string binária de 128 bits (16 octetos) em UTF-8, sendo posteriormente usados para criar uma nova variável *data_tosend* codificando a mesma em base64 modo UTF-8. Finalmente os dados a enviar (*data_tosend*) são retornados através de **return**.
- **Função decrypt_intvalue:** Similar à função anterior, a função **decrypt_intvalue** tem argumentos de entrada *client_id* e *data* e obtém a cifra do cliente invocando a função **find_cipher** com o *client_id* pretendido. Posteriormente executa o decode de base64 dos dados (variável *data*) e usa a cifra (*cipher*) para decifrar os dados. Finalmente cria uma nova variável para dados a enviar chamada *data_tosend* a qual converte os dados novamente para integer e propaga os dados fazendo uso do **return**.

```

# -----
# return the cipher of a client socket
def find_cipher(client_id):
    gamer_cipher = gamers[client_id]['cipher']
    cipherkey = base64.b64decode(gamer_cipher)
    mode = AES.MODE_ECB
    cipher = AES.new(cipherkey, mode)
    return cipher

# Função para descriptar valores recebidos em formato json com codificação base64
# return int data decrypted from a 16 bytes binary string and coded base64
def decrypt_intvalue(client_id, data):
    cipher = find_cipher(client_id)
    data = base64.b64decode(data) # cifra do cliente com "client id" x
    data = cipher.decrypt(data) # decifrar str binaria de 128 bits (8 x 16)
    data_tosend = int(str(data, 'utf8')) # str para integer
    return data_tosend

# -----
# Função para encriptar valores a enviar em formato json com codificação base64
# return int data encrypted in a 16 bytes binary string and coded base64
def encrypt_intvalue(client_id, data):
    cipher = find_cipher(client_id)
    data = cipher.encrypt(bytes("%16d" % data, 'utf8')) # cifrar str binaria de 128 bits (8 x 16)
    data_tosend = str(base64.b64encode(data), 'utf8') # str base64 dict
    return data_tosend

```

Figura 20: Funções de segurança: Servidor.

Cliente:

No Cliente, existem duas funções implementadas para a segurança: **encrypt_intvalue**, **decrypt_intvalue**, sendo a geração de uma cipherkey feita dentro da função **run_client** tal como descrito na secção ?? . A implementação destas funções será abeguidamente e pode ser visualizada na figura 21:

- **Função encrypt_intvalue:** de argumentos de entrada *client_id* e *data*, a função cria uma variável que define o modo ECB a utilizar (mode = AES.MODE_ECB) que usa juntamente com a cipherkey criada no decorrer da função **run_client** , para gerar uma cifra na varável *cipher* que usa para cifrar os dados contidos na variável *data* numa string de 128 bits (16 octetos). Posteriormente, cria uma variável *data_tosend* a qual faz e guarda uma cópia dos dados presentes na variável *data* fazendo o encode de base64 e é propagada no **return**.
- **Função decrypt_intvalue:** de argumentos de entrada *client_id* e *data*, a função cria uma variável que define o modo ECB a utilizar (mode = AES.MODE_ECB) que usa juntamente com a cipherkey, criada no decorrer da função **run_client**, para gerar uma cifra na varável *cipher*. Posteriormente executa o decode de base64 dos dados da variável *data* e usa a cifra criada para decifrar o seu conteúdo. Posteriormente cria uma variável para dados a enviar chamada *data_tosend* a qual converte os dados novamente para integer e propaga os dados fazendo uso do **return**.

```

# -----
# Função para encriptar valores a enviar em formato json com codificação base64
# return int data encrypted in a 16 bytes binary string coded in base64
def encrypt_intvalue(cipherkey, data):
    mode = AES.MODE_ECB
    cipher = AES.new(cipherkey, mode)
    data = cipher.encrypt(bytes("%16d" % data, 'utf8'))
    data_tosend = str(base64.b64encode(data), 'utf8')
    return data_tosend # dict = string encriptada

# -----
# Função para descriptar valores recebidos em formato json com codificação base64
# return int data decrypted from a 16 bytes binary string coded in base64
def decrypt_intvalue(cipherkey, data):
    mode = AES.MODE_ECB
    cipher = AES.new(cipherkey, mode)
    data = base64.b64decode(data)
    data = cipher.decrypt(data)
    data_tosend = int(str(data, 'utf8'))
    return data_tosend # dict = string decifrada

```

Figura 21: Funções de segurança: Cliente.

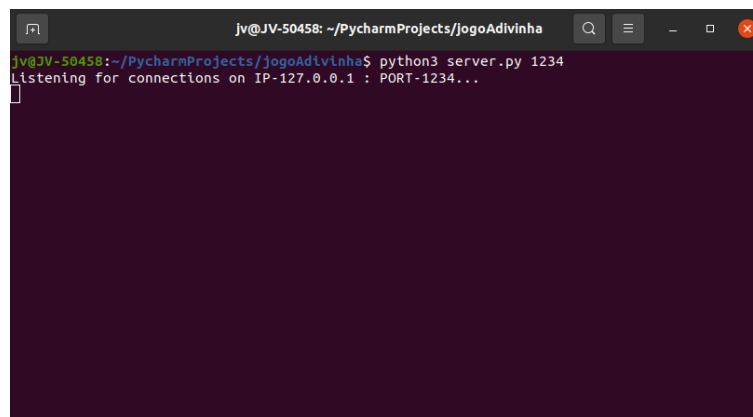
Capítulo 3

Testes de Funcionalidade e Resultados

Além das normas de implementação da aplicação, o enunciado proposto continha regras para lidar com determinadas situações de erro e garantir que o programa funciona correctamente. Nesta secção serão abordados os testes realizados para confirmar que o código desenvolvido está preparado para lidar não só com essas situações mas também com outros erros que foram encontrados ao longo do processo.

3.1 Invocação das aplicações

Para inicializar o Servidor invoca-se o seguinte comando no terminal, como indicado na figura 22: `python3 server.py port`.



```
jv@JV-50458: ~/PycharmProjects/jogoAdvinha
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Listening for connections on IP-127.0.0.1 : PORT-1234...
```

Figura 22: Exemplo iniciação de Servidor

Nota: O Servidor encontra-se iniciado e à "escuta" de novos pedidos de conexão.

Se o Servidor for inicializado com um número incorrecto de argumentos é apresentada uma mensagem de erro. Além disso, o argumento referente ao porto deve ser um número inteiro compreendido entre 1024 e 65535. Este limite deve-se ao facto de o sistema ter apenas 65535 portos e os de valores inferiores a 1024 já estão obrigatoriamente a ser utilizados pelo sistema (também conhecidas como *Well-Known Ports*).


```
lab1@lab1-VirtualBox: ~/lab12021-ap2-g42/client-server$ python3 server.py
Erro 1: Numero Inválido de argumentos
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 server.py 12345 127.0.0.1
Erro 1: Numero Inválido de argumentos
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 server.py 1023
Erro 2: Valor do Port Inválido (argv[2]) - Deve ser um número entre 1024 e 65535
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 server.py 65536
Erro 2: Valor do Port Inválido (argv[2]) - Deve ser um número entre 1024 e 65535
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 server.py 12345
listening for connections on IP-127.0.0.1 : PORT-12345...
```

Figura 23: Erros de comando

Nota: Poderiam ser usados portos entre 0 e 1023, contudo tal é desaconselhado uma vez que seria necessário "root privilege" do sistema operativo.

A aplicação-Cliente também deve ser inicializada com um número correcto de argumentos, caso contrário é apresentada uma mensagem de erro. Quando é invocado o comando de inicialização do Cliente os argumentos referentes ao porto e ao IP também devem cumprir certos requisitos. No caso do porto, este já deve estar estabelecido para receber conexões e o argumento deve ser um número inteiro compreendido entre 1024 e 65535. No caso do IP, este deve ser invocado no formato X.X.X.X onde X representa um número inteiro entre 0 e 255.

```
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto
Erro 1: Numero Inválido de argumentos
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto 12345
Erro 1: Numero Inválido de argumentos
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto 12345 127.0.0.1 123
Erro 1: Numero Inválido de argumentos
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto Host 127.0.0.1
Erro 2: Valor do Port Inválido (argv[2]) - Deve ser um número inteiro
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto 12345 Host
Erro 4: Valor do IP Inválido (argv[3])
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto 12345 127.0.0.2
Erro 5: Conexão falhada, tente outro valor Port (argv[2])
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto 55666 127.0.0.1
Erro 5: Conexão falhada, tente outro valor Port (argv[2])
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$ python3 client.py Roberto 80100 127.0.0.1
Erro 2: Valor do Port Inválido (argv[2]) - Deve ser um número entre 1024 e 65535
lab1@lab1-VirtualBox:~/lab12021-ap2-g42/client-server$
```

Figura 24: Erros de comando

Nota: Tal como referido anteriormente, poderiam ser usados portos entre 0 e 1023, contudo tal é desaconselhado uma vez que seria necessário "root privilege" do sistema operativo.

O Servidor deve ser desligado pela acção *keyboard interrupt*, usando a combinação de teclas **ctrl** + **c**, ou simplesmente desligando a consola/terminal de comandos, como se pode constatar na figura 25:

```
jv@JV-50458: ~/PycharmProjects/jogoAdvinha
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Listening for connections on IP-127.0.0.1 : PORT-1234...
^C

Server connection has been interrupted by Keyboard Interrupt.
Closing PORT usage:

Traceback (most recent call last):
  File "server.py", line 349, in <module>
    main()
  File "server.py", line 308, in main
    available = select.select([server_socket] + clients, [], [])[0]
KeyboardInterrupt

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "server.py", line 352, in <module>
    os.system("sudo lsof -t -i tcp:" + PORT + " | xargs kill -9")
NameError: name 'PORT' is not defined
jv@JV-50458:~/PycharmProjects/jogoAdvinha$
```

Figura 25: Fechar o servidor correctamente

Nota: Como se pode verificar, o Servidor indica mensagem de "Server connection has been interrupted" fazendo de seguida o comando "kill -9" para disponibilizar o porto utilizada.

Caso sejam usados outro métodos, como a combinação de teclas **ctrl** + **z**, o uso do porto não é libertado para uso futuro, como demonstrado na figura 26:

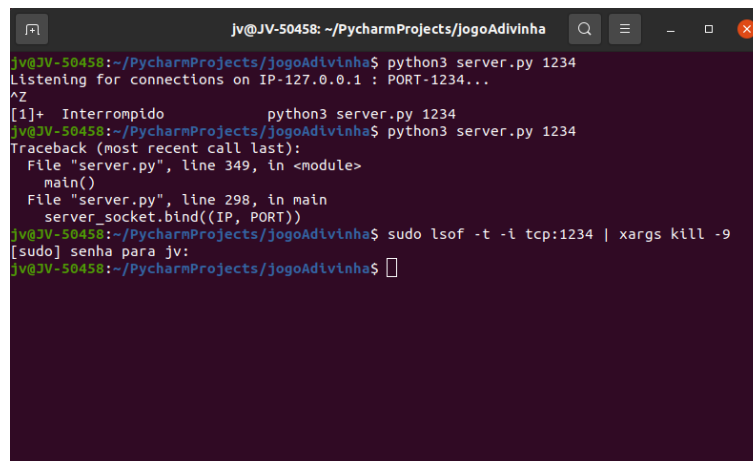
```
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Traceback (most recent call last):
  File "server.py", line 349, in <module>
    main()
  File "server.py", line 298, in main
    server_socket.bind((IP, PORT))
OSError: [Errno 98] Address already in use
jv@JV-50458:~/PycharmProjects/jogoAdvinha$

jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Listening for connections on IP-127.0.0.1 : PORT-1234...
^Z
[1]+  Interrompido      python3 server.py 1234
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Traceback (most recent call last):
  File "server.py", line 349, in <module>
    main()
  File "server.py", line 298, in main
    server_socket.bind((IP, PORT))
OSError: [Errno 98] Address already in use
jv@JV-50458:~/PycharmProjects/jogoAdvinha$
```

Figura 26: Fechar o servidor de forma errada

*Nota: Como se pode verificar, a consola/terminal da direita executou **ctrl** + **z** para fechar o Servidor, tendo como resultado nem este terminal ou outro poderem fazer uso do porto 1234*

Caso tal ocorra, existem duas alternativas simples de libertar o porto para uso futuro. A mais simples seria desligar o terminal inicialmente usado para correr o Servidor. Se isso não for o pretendido pode-se correr o comando demonstrado na figura 27



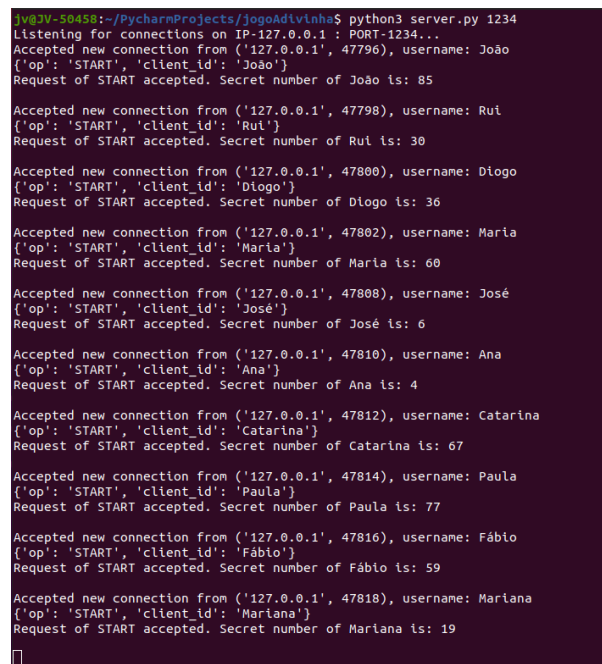
```
jv@JV-50458: ~/PycharmProjects/jogoAdvinha
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Listening for connections on IP-127.0.0.1 : PORT-1234...
^Z
[1]+  Interrompido                  python3 server.py 1234
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Traceback (most recent call last):
  File "server.py", line 349, in <module>
    main()
  File "server.py", line 298, in main
    server_socket.bind((IP, PORT))
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ sudo lsof -t -i tcp:1234 | xargs kill -9
[sudo] senha para jv:
jv@JV-50458:~/PycharmProjects/jogoAdvinha$
```

Figura 27: Disponibilizar porta via comando terminal linux.

Comando: `sudo lsof -t -i tcp:port | xargs kill -9`

3.2 Funcionalidade do Cliente

O Servidor deve conseguir estabelecer conexão com vários Clientes ao mesmo tempo e garantir o correcto funcionamento do jogo para todos os usuários conectados, garantindo que nenhum usuário tenha de esperar que outros acabem para poder executar o jogo, como demonstrado na figura 28



```
jv@JV-50458:~/PycharmProjects/jogoAdvinha$ python3 server.py 1234
Listening for connections on IP-127.0.0.1 : PORT-1234...
Accepted new connection from ('127.0.0.1', 47796), username: João
{'op': 'START', 'client_id': 'João'}
Request of START accepted. Secret number of João is: 85

Accepted new connection from ('127.0.0.1', 47798), username: Rui
{'op': 'START', 'client_id': 'Rui'}
Request of START accepted. Secret number of Rui is: 30

Accepted new connection from ('127.0.0.1', 47800), username: Diogo
{'op': 'START', 'client_id': 'Diogo'}
Request of START accepted. Secret number of Diogo is: 36

Accepted new connection from ('127.0.0.1', 47802), username: Maria
{'op': 'START', 'client_id': 'Maria'}
Request of START accepted. Secret number of Maria is: 60

Accepted new connection from ('127.0.0.1', 47808), username: José
{'op': 'START', 'client_id': 'José'}
Request of START accepted. Secret number of José is: 6

Accepted new connection from ('127.0.0.1', 47810), username: Ana
{'op': 'START', 'client_id': 'Ana'}
Request of START accepted. Secret number of Ana is: 4

Accepted new connection from ('127.0.0.1', 47812), username: Catarina
{'op': 'START', 'client_id': 'Catarina'}
Request of START accepted. Secret number of Catarina is: 67

Accepted new connection from ('127.0.0.1', 47814), username: Paula
{'op': 'START', 'client_id': 'Paula'}
Request of START accepted. Secret number of Paula is: 77

Accepted new connection from ('127.0.0.1', 47816), username: Fábio
{'op': 'START', 'client_id': 'Fábio'}
Request of START accepted. Secret number of Fábio is: 59

Accepted new connection from ('127.0.0.1', 47818), username: Mariana
{'op': 'START', 'client_id': 'Mariana'}
Request of START accepted. Secret number of Mariana is: 19


```

Figura 28: Servidor com 10 clientes conectados ao mesmo tempo, com ID diferente uns dos outros

O enunciado do trabalho não especifica o que a aplicação Servidor deve apresentar no seu terminal. Assim, decidiu-se que seria interessante o Servidor apresentar certas mensagens de forma a monitorizar

algumas das suas operações internas.

```

jv@JV-50458: ~/Desktop/MIECT/LI/LAB/Lab2021-ap2-g42/client-server
BEM-VINDO

**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Operação START aceite. Ten 10 tentativas para tentar acertar no número s
secreto
OP: 68
Resultado: smaller
Jogadas feitas: 1

OP: 69
Resultado: equals
Jogadas feitas: 2

O jogo acabou. O número secreto era 69
Você ACERTOU

Quer jogar novamente? (Y/N)?
Y
Operação START aceite. Ten 10 tentativas para tentar acertar no número s
secreto
OP: quit
Operação QUIT aceite.
Até à próxima

jv@JV-50458: ~/Desktop/MIECT/LI/LAB/Lab2021-ap2-g42/client-server$

jv@JV-50458: ~/Desktop/MIECT/LI/LAB/Lab2021-ap2-g42/client-server$ python3 server.py 2234
Listening for connections on IP:127.0.0.1 : PORT:2234...
Accepted new connection from ("127.0.0.1", 55740), username: Rui
{"op": "START", "client_id": "Rui"}
Request of START accepted. Secret number of Rui is: 69

{"op": "GUESS", "number": "68"}
Rui
{"op": "GUESS", "number": "69"}
Rui
{"op": "STOP", "number": "69", "attempts": 2}
{"op": "START", "client_id": "Rui"}
Request of START accepted. Secret number of Rui is: 69

{"op": "QUIT"}
Rui has disconnected
[]

```

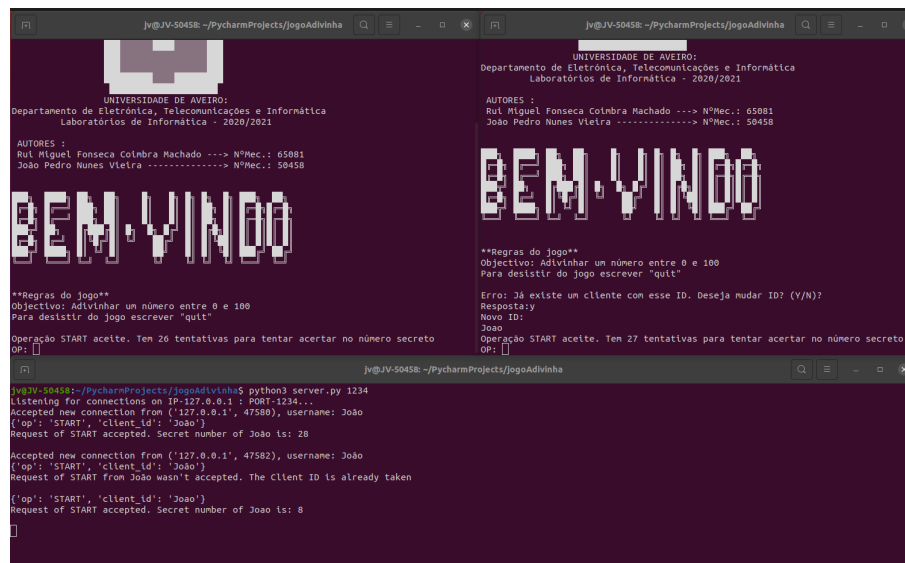
Figura 29: Servidor: Actividade recente dos jogadores

Após ser inicializado, o Cliente apresenta uma mensagem com as regras do jogo e envia automaticamente um pedido ao Servidor para começar o jogo. Se não existir nenhum usuário com o mesmo ID a jogar o Servidor aceita o pedido, caso contrário o Servidor envia uma mensagem a negar o pedido. Nesse caso, a aplicação-Cliente informa o usuário e pergunta-lhe se quer mudar o seu ID. O usuário pode trocar de ID até encontrar um válido ou sair da aplicação.



Figura 30: Cliente muda ou não muda de id

A escolha de um novo ID apenas ligeiramente diferente do ID já registado é permitido pelo Servidor, como exemplificado na figura 31:



The image shows three terminal windows. The top-left window displays the game's welcome screen with ASCII art and rules. The top-right window shows the same screen but with a different ID chosen. The bottom window shows the server's log, indicating that a client with the same ID was already registered, leading to an error message for the new client.

```
UNIVERSIDADE DE AVEIRO:
Departamento de Eletrónica, Telecomunicações e Informática
Laboratórios de Informática - 2020/2021

AUTORES :
Rui Miguel Fonseca Coimbra Machado --> N°Mec.: 65881
João Pedro Nunes Vieira -----> N°Mec.: 50458

BEM-VINDO

**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Operação START aceite. Tem 26 tentativas para tentar acertar no número secreto
OP: 1
```

```
UNIVERSIDADE DE AVEIRO:
Departamento de Eletrónica, Telecomunicações e Informática
Laboratórios de Informática - 2020/2021

AUTORES :
Rui Miguel Fonseca Coimbra Machado --> N°Mec.: 65881
João Pedro Nunes Vieira -----> N°Mec.: 50458

BEM-VINDO

**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Erro: Já existe um cliente com esse ID. Deseja mudar ID? (Y/N)?
Resposta: y
Novo ID:
João
Operação START aceite. Tem 27 tentativas para tentar acertar no número secreto
OP: 1
```

```
Jo@JV-50458: ~/PycharmProjects/jogoAdivinha$ python3 server.py 1234
Listening for connections on IP-127.0.0.1 : PORT-1234...
Accepted new connection from ('127.0.0.1', 47588), username: João
('op': 'START', 'client_id': 'João')
Request of START accepted. Secret number of João is: 28

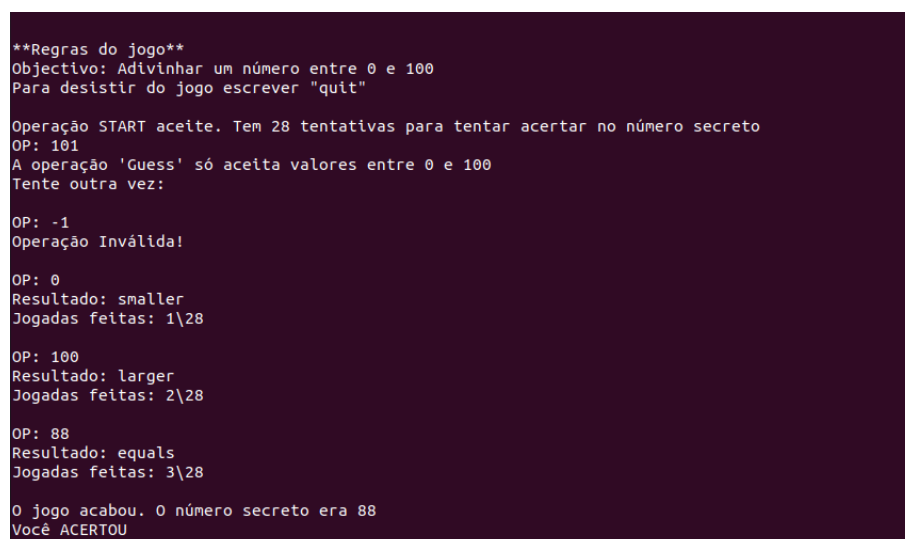
Accepted new connection from ('127.0.0.1', 47582), username: João
('op': 'START', 'client_id': 'João')
Request of START from João wasn't accepted. The client ID is already taken

('op': 'START', 'client_id': 'João')
Request of START accepted. Secret number of João is: 8
```

Figura 31: Escolha de um novo ID similar a outro ID já registado

Nota: Desta forma, clientes podem escolher ID do estilo nickname com caracteres diferentes

Como já referido anteriormente, após o começo do jogo o usuário pode fazer várias tentativas para adivinhar o número secreto escrevendo um número compreendido entre 0 e 100 no terminal da aplicação-Cliente. O Servidor recebe a tentativa e retorna uma mensagem com informação relativa à mesma, dizendo se a tentativa está abaixo ou acima do número secreto, ou se acertou. Além disso, sempre que é feita uma tentativa a aplicação-Cliente apresenta no terminal o número de tentativas realizadas e o número máximo de tentativas possíveis.



The image shows a terminal window displaying the game's guess operation. The user enters a guess of 101, which is invalid. Then, they enter 0, which is smaller than the secret number. Next, they enter 100, which is larger than the secret number. Finally, they enter 88, which is the correct secret number, and the game ends.

```
**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Operação START aceite. Tem 28 tentativas para tentar acertar no número secreto
OP: 101
A operação 'Guess' só aceita valores entre 0 e 100
Tente outra vez:

OP: -1
Operação Inválida!

OP: 0
Resultado: smaller
Jogadas feitas: 1\28

OP: 100
Resultado: larger
Jogadas feitas: 2\28

OP: 88
Resultado: equals
Jogadas feitas: 3\28

O jogo acabou. O número secreto era 88
Você ACERTOU
```

Figura 32: Funcionamento da operação Guess

Se o usuário acertar no número secreto ou esgotar as suas tentativas a aplicação-Cliente apresenta uma mensagem com o resultado do jogo e o número secreto gerado pelo Servidor, perguntando ainda ao

usuário se pretende jogar mais uma vez. O usuário pode começar um jogo novo ou terminar a aplicação.

```

j@JV-50458: ~/PycharmProjects/JogoAdivinha
UNIVERSIDADE DE AVEIRO:
Departamento de Eletrónica, Telecomunicações e Informática
Laboratórios de Informática - 2020/2021

AUTORES :
Rui Miguel Fonseca Coimbra Machado ---- N°Mec.: 65081
João Pedro Nunes Vieira ----- N°Mec.: 50458

BEM-VINDO

**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Operação START aceite. Tem 16 tentativas para tentar acertar no número secreto
QP: 100
Resultado: larger
Jogadas feitas: 1

QP: 97
Resultado: smaller
Jogadas feitas: 2

QP: 98
Resultado: equals
Jogadas feitas: 3

O jogo acabou. O número secreto era 98
Você ACERTOU

Quer jogar novamente? (Y/N)?
Operação START aceite. Tem 11 tentativas para tentar acertar no número secreto
QP:

```

(a) Jogo completo. Cliente ganhou e escolhe jogar novamente

```

j@JV-50458: ~/Desktop/MIECT/LI/LAB/lab2021-ap2-g42/client-...
UNIVERSIDADE DE AVEIRO:
Departamento de Eletrónica, Telecomunicações e Informática
Laboratórios de Informática - 2020/2021

AUTORES :
Rui Miguel Fonseca Coimbra Machado ---- N°Mec.: 65081
João Pedro Nunes Vieira ----- N°Mec.: 50458

BEM-VINDO

**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Operação START aceite. Tem 23 tentativas para tentar acertar no número secreto
QP: 10
Resultado: smaller
Jogadas feitas: 1

QP: 98
Resultado: larger
Jogadas feitas: 2

QP: 89
Resultado: equals
Jogadas feitas: 3

O jogo acabou. O número secreto era 89
Você ACERTOU

Quer jogar novamente? (Y/N)?
n
Foi um prazer jogar consigo! Até à próxima!
Até à próxima

j@JV-50458: ~/Desktop/MIECT/LI/LAB/lab2021-ap2-g42/client-server$

```

(b) Jogo completo. Cliente ganhou e escolhe não jogar novamente

```

j@JV-50458: ~/Desktop/MIECT/LI/LAB/lab2021-ap2-g42/client-server
Operação START aceite. Tem 12 tentativas para tentar acertar no número secreto
QP: 90
Resultado: smaller
Jogadas feitas: 1

QP: 98
Resultado: smaller
Jogadas feitas: 2

QP: 99
Resultado: smaller
Jogadas feitas: 3

QP: 99
Resultado: smaller
Jogadas feitas: 4

QP: 99
Resultado: larger
Jogadas feitas: 5

QP: 99
Resultado: larger
Jogadas feitas: 6

QP: 99
Resultado: larger
Jogadas feitas: 7

QP: 99
Resultado: larger
Jogadas feitas: 8

QP: 100
Resultado: larger
Jogadas feitas: 9

QP: 100
Resultado: smaller
Jogadas feitas: 10

QP: 100
Resultado: equals
Jogadas feitas: 11

O jogo acabou. O número secreto era 100
Você ACERTOU

```

(c) Jogo completo. Cliente ganhou na última.

```

j@JV-50458: ~/Desktop/MIECT/LI/LAB/lab2021-ap2-g42/client-server
Jogadas feitas: 14
QP: 74
Resultado: smaller
Jogadas feitas: 15

QP: 79
Resultado: smaller
Jogadas feitas: 16

QP: 88
Resultado: smaller
Jogadas feitas: 17

QP: 82
Resultado: smaller
Jogadas feitas: 18

QP: 85
Resultado: smaller
Jogadas feitas: 19

QP: 83
Resultado: smaller
Jogadas feitas: 20

O jogo acabou. O número secreto era 87
Você esgotou as tentativas

Quer jogar novamente? (Y/N)?

```

(d) Jogo completo. Cliente perdeu por esgotar tentativas.

Figura 33: Simulação de jogo completo. Cliente escolhe se pretende jogar novamente

Notas:

1ª Jogada: Cliente adivinha um número inferior ao número secreto (*smaller*).

2ª Jogada: Cliente adivinha um número superior (*larger*).

3ª Jogada: Cliente acerta no número secreto (*equals*).

Posteriormente, o cliente escolhe se pretende ou não jogar novamente.

O usuário pode sair do jogo a qualquer momento através do comando *quit*. Nesse caso, a aplicação-Cliente apenas apresenta uma mensagem a informar se o Servidor aceitou a operação e o processo ocorreu sem erros. Qualquer input por parte do usuário que não seja um número ou a operação *quit* gera uma mensagem de erro *Operação Inválida* e nada acontece.

```
**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Operação START aceite. Tem 14 tentativas para tentar acertar no número secreto
OP: xpto
Operação Inválida!

OP: close
Operação Inválida!

OP: guess 12
Operação Inválida!

OP: quit
Operação QUIT aceite.
Até à próxima
```

Figura 34: Funcionamento da operação Quit

3.3 Testes do Servidor

Na secção anterior foi explicado e ilustrado o correcto funcionamento da aplicação-Cliente. Nesta secção são descritos os testes feitos à aplicação-Servidor de forma a confirmar que o código está construído de forma que, caso surjam situações fora do expectável, estas são corrigidas. Para tal foi criado um directório intitulado de *FicheirosTeste* com alguns ficheiros de código que forçam situações de erro e mostram as mensagens enviadas pelo Servidor.

Quando o pedido de inicialização do jogo é aceite pelo Servidor este acrescenta à lista de jogadores activos os dados do Cliente que efectuou o pedido. Sempre que o Servidor recebe uma mensagem/operação de um Cliente a primeira coisa que deve ser feita é confirmar que esse Cliente está a jogar, verificando a lista de jogadores activos. Se o Servidor recebe uma mensagem de um Cliente não activo, o Servidor deve enviar uma mensagem de erro. É importante referir que o Cliente está implementado de forma que termine imediatamente sempre que receba uma mensagem de erro. Esta funcionalidade é expressamente pedida no enunciado do trabalho.

```
**Regras do jogo**
Objectivo: Adivinhar um número entre 0 e 100
Para desistir do jogo escrever "quit"

Mensagem do Servidor:
{'op': 'GUESS', 'status': False, 'error': 'Cliente inexistente'}

Operação GUESS não foi aceite!
Erro 3: Operação inválida
Erro: Cliente inexistente
Até à próxima
```

Figura 35: Operação *Guess* enviada por um Cliente não activo

```
**Regras do jogo**  
Objectivo: Adivinhar um número entre 0 e 100  
Para desistir do jogo escrever "quit"  
  
Mensagem do Servidor:  
{'op': 'QUIT', 'status': False, 'error': 'Cliente inexistente'}  
  
Operação QUIT não foi aceite!  
Erro 3: Operação inválida  
Erro: Cliente inexistente  
Até à próxima
```

Figura 36: Operação *Quit* enviada por um Cliente não activo

```
**Regras do jogo**  
Objectivo: Adivinhar um número entre 0 e 100  
Para desistir do jogo escrever "quit"  
  
Mensagem do Servidor:  
{'op': 'STOP', 'status': False, 'error': 'Cliente inexistente'}  
  
Operação STOP não foi aceite!  
Erro 3: Operação inválida  
Erro: Cliente inexistente  
Até à próxima
```

Figura 37: Operação *Stop* enviada por um Cliente não activo

Na operação *Stop* o Servidor faz uma verificação adicional para se certificar que o número de tentativas jogadas reportado pelo Cliente é igual ao número de tentativas registadas pelo próprio Servidor. Se houver alguma discrepância entre os dois registos o Servidor envia uma mensagem de erro.

```
**Regras do jogo**  
Objectivo: Adivinhar um número entre 0 e 100  
Para desistir do jogo escrever "quit"  
  
Mensagem do Servidor:  
{'op': 'START', 'status': True, 'max_attempts': 23}  
  
Mensagem do Servidor:  
{'op': 'STOP', 'status': False, 'error': 'Número de tentativas incoerente entre Servidor e Cliente'}  
  
Operação STOP não foi aceite!  
Erro 3: Operação inválida  
Erro: Número de tentativas incoerente entre Servidor e Cliente  
Até à próxima
```

Figura 38: Incoerência entre tentativas registadas pelo Servidor e Cliente

Se por alguma razão o Servidor receber uma operação fora dos parâmetros possíveis, este deve enviar uma mensagem de erro informando que a operação é inexistente.


```
**Regras do jogo**  
Objectivo: Adivinhar um número entre 0 e 100  
Para desistir do jogo escrever "quit"  
  
Mensagem do Servidor:  
{'op': 'START', 'status': True, 'max_attempts': 30}  
  
Mensagem do Servidor:  
{'op': 'XPTO', 'status': False, 'error': 'Operação inexistente'}  
  
Operação XPTO não foi aceite!  
Erro 3: Operação inválida  
Erro: Operação inexistente  
Até à próxima
```

Figura 39: Cliente envia operação inexistente ao Servidor

Ao longo do desenvolvimento deste projecto foram encontrados e corrigidos vários erros, no entanto, para o âmbito deste relatório, decidiu-se não os explorar exhaustivamente devido à sua pouca relevância. A maioria destes erros estavam relacionados com as funções *find-client*, *clean-client*, *create-file*, *update-file* e variáveis globais.

Também se considerou importante referir que aplicação começou por ser implementada duma maneira diferente do proposto por algum lapso de interpretação do enunciado, sendo que o código desenvolvido nessa fase encontra-se na pasta *Alt-ServClient*. Essa versão da aplicação, apesar de não ter sido terminada, está funcional e pode ser útil para futuros projectos uma vez que contém vários mecanismos adicionais à versão final para manter a sua robustez.

3.4 Análise de Resultados

Como se pode verificar nas secções anteriores, o programa aparenta funcionar correctamente e de acordo com as especificações pedidas no enunciado. As funcionalidades adicionadas pelos autores também aparentam funcionar de maneira correcta, complementando o projecto. Todos os bugs encontrados foram corrigidos, e todas as situações de erro previstas foram tratadas, tornando assim o programa o mais robusto possível dadas as normas do enunciado e o prazo de entrega.

Capítulo 4

Conclusão:

Dados os resultados obtidos podemos concluir que a elaboração deste projecto foi bem sucedida. Não só foi possível implementar o que era pedido, como ainda desenvolvemos funcionalidades extra que consideramos complementarem o projecto.

Este projecto foi bastante útil para desenvolver competências de programação Python, tanto em termos de domínio de linguagem, como de estruturação de código. Além disso este trabalho abordou uma variedade de outras matérias nomeadamente programação de Sockets, documentos JSON e CSV, criptografia e GIT. Finalizamos o trabalho com a noção de que, apesar do seu sucesso, tendo um período mais alargado seria possível expandir consideravelmente o projecto.

Contribuições dos Autores

Tabela 1: Contribuições dos Autores.

Contribuição	Rui Machado	João Vieira
Produção de Relatório	50 %	50 %
Desenvolvimento de Código	80 %	20 %
Implementação de Segurança	60 %	40 %
Testes e Depuração	60 %	40 %

Bibliografia

- [1] Allen Downey *Think Python - How to Think Like a Computer Scientist*. | Green Tea Press, 2nd Edition, Version 2.4.0, 2015
- [2] PyCryptodome documentation. | pycryptodome.readthedocs.io, acedido 18/05/2021.
- [3] Wikipédia: Enciclopédia livre. | pt.wikipedia.org, acedido 19/05/2021.
- [4] ServerFault Website. | serverfault.com, acedido 22/05/2021.
- [5] Internet Assigned Numbers Authority. | www.iana.org, acedido 22/05/2021.