Adivinha o Número Secreto

Universidade de Aveiro

Gonçalo Silva, Samuel Teixeira

VERSAO 1.0

Adivinha o Número Secreto

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

Gonçalo Silva, Samuel Teixeira
(103) goncalol@ua.pt, (103325) steixeira@ua.pt

DATA

Resumo

Este projeto foi realizado no âmbito da cadeira LABI (acr) do 1º ano do MIECT. Consiste em criar um servidor em que os clientes se conectam para jogar um jogo de adivinha o número secreto. Para além disso, também tivemos que fazer este mesmo relatório em que nós explicamos o projeto: objetivo, motivação, a metodologia utilizada, resultados e conclusões. Na metodologia, será relatado em pormenor o nosso código feito para criar o servidor, tanto código do cliente como do servidor, o modo de funcionamento, testagem e comandos git feitos para tal. Nos resultados, será mostrado o fruto de todo o nosso código que é o servidor a funcionar. Por fim, nas conclusões, retira-se o que se alcançou com este projeto, o que aprendemos, o quão útil este projeto é para compreendermos esta matéria da cadeira de LABI e o quão interessante foi realizá-lo.

Agradecimentos

Queremos agradecer a todos os professores da cadeira de LABI por nos terem dado um trabalho interessante, que nos ajudou a compreender os conceitos lecionados nas aulas. Queremos também agradecer a Guido van Rossum por ter criado a linguagem Python e ao Bill Gates por ter sido um dos grandes pilares para a criação de portáteis.

Índice

Capítulo 1

Introdução

O objetivo do jogo é que o cliente adivinhe um número secreto entre 0 e 100 criado aleatoriamente pelo servidor, dentro de um determinado número de tentativas dadas também pelo servidor. O cliente tem a possibilidade de realizar quatro operações diferentes. A primeira serve para iniciar o jogo (START), a segunda serve para tentar adivinhar o número secreto (GUESS), a terceira serve para desistir do jogo (STOP) e a última serve para sair (QUIT). O servidor deve responder adequadamente aquando da operação pretendida. Se houver algum erro, como um cliente inexistente, o servidor deve dar uma resposta adequada ao cliente. No final do jogo, o servidor guarda num ficheiro .csv os dados do cliente que jogou o jogo. Para efeitos de segurança, o cliente possui ainda a possibilidade de escolher se quer que os seus dados sejam encriptados ou não.

Este documento está dividido em quatro capítulos. Depois desta introdução, no Capítulo 2 é apresentada a metodologia seguida, no Capítulo 3 são apresentados os resultados obtidos, sendo estes discutidos no Capítulo 4. Finalmente, no Capítulo 5 são apresentadas as conclusões do trabalho.

Capítulo 2

Metodologia

2.1 Cliente

Nesta secção será apresentada a metodologica no ficheiro client.py

2.1.1 Função main

Devido ao facto do cliente ser invocado com o comando, esses python3 client.py client_id porto [máquina]"argumentos têm de ser validados. Para começar, devem ser colocados 3 ou 4 argumentos (máquina é opcional). Caso não tenha nenhuma destas quantidades, é enviada uma mensagem de erro. Caso o tamanho seja 3, a máquina é a local(127.0.0.1), caso seja 4 a máquina é o último argumento.

O client_id não possui qualquer tipo de restrição, logo a única verificação feita é se ele existe (len(argv[1])). Para a porta, existem 2 condições: a porta inserida é constituída apenas por números e esse número encontra-se entre 0 e 65535. Para isso, percorre-se todos os caractéres da string e, caso não seja um dígito, é enviada uma mensagem de erro.

Por fim, a máquina tem de ser verificada, os seus números entre cada "."estão compreendidos no intervalo]0, 255]. Os números são colocados num array através do método ".split('.')"e, caso algum não satisfaça a condição, é enviada mensagem de erro.

Após esta verificação, é criado o socket com a porta e máquina indicados na invocação, tenta-se estabelecer conexão e chama-se a função run_client, que é onde se vai passar o jogo. Quando terminar, é fechado o socket e o cliente termina.

2.1.2 Função run_client

A função run_client é invocada na main. O cliente é introduzido ao jogo e é-lhe questionado se pretende encriptação de dados (S/N). Enquanto a respotas for diferente dessas duas letras, é enviada uma mensagem de erro. Após

a inserção da opção, é criado um dicionário start com o id e a cifra do cliente. Se foi inserido "S", é criada uma chave que é inserida no dicionário start na chave cipher. É usada a função sendrecv_dict do common_comm como recomendado para enviar start e receber a resposta do servidor(recvstart). É verificado se houve algum erro no início do jogo através de validate_response, se houve termina, senão a variável maxAttempts fica com o valor correspondente em recvstart(desencriptada caso necessário). É escrito no ecrã do cliente as tentativas máximas que tem para adivinhar o número e um menu com as opções que pode fazer , onde o cliente pode escolher o que fazer. É verificado se o que o cliente inseriu é um número ou não através de um try/except , onde caso não seja inserido um número, a opção fica com o valor 999 por defeito pois no final da run_client, essa operação é reconhecida como desconhecida e é relatado. Se a opção inserida for 1, é iniciada a operação start, como indicada no menu.

```
ent.py > run_client
# Suporte da execução do cliente
#
def run_client(client_sock, client_id):
    tries = 0 # variavél usada para contar as tentativas feitas
    lastAttempt = 0 # variável usada para guardar a ultima tentativa do utilizador
    print("*# Bem vindo aooooooooooo Adivinha o número secreto!!!! *#")
    print("Deseja usar encriptação de dados? (S/N)")

    answer = input("-> ").upper() # Escolha da resposta pelo utilizador (text.upper() para não haver problem
    while answer != "S" and answer != "N": # Enquanto o utilizador não escolher o valor correto
        answer = input("Inválido").upper() # text.upper() para não haver problemas com letras maiúsculas e m

    start = {'op': "START", 'client_id': client_id, 'cipher' : None} # dicionário de defeito, de pedido de c
    key = None # Valor por defeito da chave de encriptação

    if answer == "S": # Se o utilizador escolheu encriptar os dados
        key = os.urandom(16) # Gerar uma chave de encriptação
        start['cipher'] = str (base64.b64encode (key), 'utf8') # guardar a chave (de modo codificado) no dic

    recvstart = sendrecv_dict(client_sock, start) # enviar os dados de começo de jogo para o servidor

    if validate_response(client_sock, recvstart): return None # validar que a resposta do servidor não conté

    maxAttempts = recvstart['max_attempts'] # variável global contendo o valor máximo de tentativas

    if (key != None) : # Se o valor máximo de tentativas estiver encriptado, desencripta
        maxAttempts = decrypt_intvalue(key, maxAttempts)
```

Opção 1

O cliente insere a sua tentativa de adivinhar o número secreto, que é verificada num try/except e, caso seja inválido, dá erro e é pedida novamente porque encontra-se dentro de um ciclo "while True". Depois de escolhida a opção, é encriptada caso haja encriptação. É criado um dicionário típico da operação start ('op': "GUESS", 'number': num, sendo num a tentativa) e é enviado para o servidor, recebendo ao mesmo tempo a resposta através de sendrecv_dict e guardada em recvguess. A resposta é é validada (validate_response) e a tentativa é registada em tries. Se o jogador acertou no número, é dito ao cliente e é redirecionado para a opção 2, se foi maior ou menor ("smaller"/"larger") é também escrito no ecrã esse resultado. Se o número de tentativas exceder o máximo estipulado pelo servidor, é dito ao cliente e este é redirecionado para a opção 2.

```python
#Operação Guess
if option == 1:
    while True :
        try :  # testar se o cliente introduziu um caracter ou número
            print("Adivinha o número secreto entre 0 e 100: ")
            num = int(input("-> "))
            if (num <= 100 and num >= 0): # verificar se foi introduzido um número fora da condição
                break
            else :
                print("Erro!: Escolhe números entre 0 e 100")
        except :
            print("Erro!: Por favor introduz apenas números")

    lastAttempt = num # variável global que guarda a última tentativa

    if (key != None) : # Se escolher encriptar, encripta a tentativa
        num = encrypt_intvalue(key, num)

    guess = {'op': "GUESS", 'number': num} # dicionário para ser enviado ao servidor
    recvguess = sendrecv_dict(client_sock, guess) # enviar o dicionário

    if validate_response(client_sock, recvguess): break # validar que não foi obtido um erro como resposta

    tries += 1 # registar que o utilizador fez uma tentativa e foi processada

    if recvguess['result'] == "equals": # o jogador acertou o número secreto
        print("*#* Conseguiste adivinhar *#*")
        option = 2 # mudar a escolha do utilizador, de forma a ser processado o término do jogo, pois acertou
```

```python
    elif recvguess['result'] == "smaller": # o jogador introduziu um número superior ao número secreto
        print("DICA: O número secreto é menor do que o inserido")
    elif recvguess['result'] == "larger": # o jogador introduziu um número inferior ao número secreto
        print("DICA: O número secreto é maior do que o inserido")
    else:
        print("Erro!: Ocorreu um erro") # ocorreu algum Erro

    if(tries >= maxAttempts) : # deixa o jogador jogar n tentativas até m jogadas máximas, de forma a n = m
        print("RESULTADO: Número máximo de tentativas obtido. O fim do jogo vai ser processado")
        option = 2 # mudar a escolha do utilizador, de forma a ser processado o término do jogo, pois o jogad
```

Opção 2

Na opção 2, é guardada a última tentativa do jogador em lastAttempt_toSend, que é encriptada caso tenha sido escolhida encriptação. É então criado o dicionário típico desta operação ("op": "STOP", "number": lastAttempt_toSend, "attempts": tries_toSend, sendo tries_toSend o número de tentativas feitas). É enviado este dicionário e recebido outro pelo mesmo processo que foi descrito para a operação 1, só que a resposta é armazenada em recvstop. Essa resposta é validada (validate_response, caso haja quer dizer que o jogador excedeu o número de jogadas e perdeu o jogo, é dito ao cliente isso e sai do jogo) e o número secreto, que está na chave 'guess' de recvstop é armazenada em returnGuess, que é desencriptado se houver encriptação. Se a última tentativa do jogador for igual ao número secreto, o jogador acertou e ganhou o jogo, senão perdeu. Em ambos os casos o cliente é informado do seu resultado e sai do jogo.

Opção 3

Na opção 3, o jogador é enviado para a função quit_action, onde é sistematizada toda a operação esta operação. O output da função é armazenado em condition. Se condition tiver o valor None, não houve erro e o cliente sai do jogo. Caso contrário, o erro é dito ao cliente e este sai do jogo.

No fim de todo este código, encontra-se a condição que é executada se o número da operação inserida pelo cliente for inválido. Nessa situação, o cliente

é informado deste problema e, visto que tudo isto se encontra dentro de um ciclo "while True", o jogador volta ao menu para escolher outra opção.

```
#Operação Quit
if option == 3:
    condition = quit_action(client_sock, tries)
    if condition == None: break # Se não houve erro
    else: # Se houver erro
        print(condition)
        break

if(option < 1 or option > 3): print("ERRO!: Valor de operação inválido") #Se a opção inserida for inválida
```

2.1.3 Função validate_response

A função validate_response procura por uma chave "error"no dicionário response, que corresponde à resposta de um servidor ao que foi enviado pelo cliente. Visto que, quando existe um erro, seja qual for a operação, esta chave é enviada, é feita essa procura e, se existir, é enviado um valor booleano True, caso contrário é enviado False

```
# Função para encriptar valores a enviar em formato jsos com codificação base64
# return int data encrypted in a 16 bytes binary string coded in base64
def encrypt_intvalue(cipherkey, data):
    cipher = AES.new (cipherkey, AES.MODE_ECB) # Definir o algoritmo de encriptação tendo em conta a chave fo
    result = cipher.encrypt (bytes("%16d" % (int(data)), 'utf8'))
    return str (base64.b64encode (result), 'utf8') # resultado encriptado


# Função para desencriptar valores recebidos em formato json com codificação base64
# return int data decrypted from a 16 bytes binary strings coded in base64
def decrypt_intvalue(cipherkey, data):
    cipher = AES.new (cipherkey, AES.MODE_ECB) # Definir o algoritmo de encriptação tendo em conta a chave fo
    result = base64.b64decode (data)
    result = cipher.decrypt (result)
    return int (str (result, 'utf8')) # resultado desencriptado


# verify if response from server is valid or is an error message and act accordingly
# If true there's an error
def validate_response(client_sock, response):
    if "error" in response: # Se existir a chave error no dicionário enviado pelo server
        print("Erro!: " + response['error']) # Mostra o erro
        return True
    else:
        return False
```

2.1.4 Função quit_operation

Nesta função, é criado um dicionário quit com uma única chave com o nome da operação "QUIT". O dicionário recvquit é criado para receber a resposta do servidor ao enviar quit através de sendrecv_dict. Se a função validate_response verificar que existe um erro, é enviado o return da chave do erro. Se não houver erro, o cliente é informado que desistiu depois de x tentativas, sendo x "attempts"

```python
# process QUIT operation
def quit_action(client_sock, attempts):
    quit = {"op": "QUIT"} # dicionário para ser enviado
    recvquit = sendrecv_dict(client_sock, quit) # enviar o dicionário
    if validate_response(client_sock, recvquit): return recvquit['error'] # se houver um erro, mostra-o
    else:
        print(f"Desistiu do jogo depois de {attempts} tentativas") # Avisa o jogador que a operação foi efet
        return None
```

2.2 Servidor

Nesta secção será apresentada e explicada a metodologia docódigo no ficheiro client.py

2.2.1 Função main

A função main é a função principal do cliente e é a primeira a ser executada quando o servidor é iniciado. Para iniciar o servidor, é usado o comando UNIX (ACRÓNIMO) **'python3 server.py <número do porto>'**. Após o começo do programa, os argumentos de entrada serão validados, começando por verificar se existe um valor do porto e se este é inteiro e está entre 0 e 65535. De seguida o programa tenta iniciar o servidor com a porta fornecida. Se a porta fornecida pelo utilizador for reservada o programa avisa o utilizador e termina, similarmente, se a porta fornecida pelo utilizador já estiver a ser usada por um outro servidor a correr, o programa irá comunicar ao utilizador e terminar.

Caso não exista nenhum erro nas condições acima, o servidor irá ser iniciado e criará o ficheiro de resultados dos jogadores "report.csv", bem como passará a ficar "à escuta"de conecções dos clientes.

Com o servidor a funcionar, este vai ficar à espera de conecções de clientes. Quando um cliente tentar se conectar, o servidor irá obter os valores do seu socket e verificar se já existe algum cliente na lista com os mesmos valores, se não existir, então irá adicioná-lo à lista.

Tendo o cliente estabelecido a conecção com sucesso, este irá comunicar com o servidor através de pedidos. Esses pedidos são "ouvidos"pelo servidor e comunicados à Função new_msg que os irá processar.

Finalmente, como estão a ser usados sockets TCP (ACRÓNIMO), estes "sentem"se alguma exceção aconteceu e se o cliente se desconectou do servidor, caso isso aconteça, o servidor irá remover este cliente da lista de clientes, executar a Função clean_client e fechar o socket do cliente.

2.2.2 Função find_client_id

A função find_client_id aceita como parâmetros o socket do cliente e procura se este se encontra registado no dicionário de jogadores ativos, gamers e devolve o id do cliente que está associado. Caso contrário devolve "None", indicando que não existe nenhum cliente associado.

2.2.3 Função encrypt_intvalue

O objetivo desta função é encriptar um valor inteiro, para tal, aceita como parâmetros o id do cliente e um valor inteiro, devolvendo uma palavra (String) codificada que representa esse valor. De seguida, é usada a chave previamente guardada no dicionário associada ao id do cliente e criada uma cifra do tipo AES-128 (ACRONIMO). Esta crifa criada, será usada para encriptar o número em forma de bytes do tipo 'utf8'. Por final, convertemos os bytes que obtivemos no passo anterior para texto (String) usando a função **base64.b64encode** (Acrónimo).

2.2.4 Função decrypt_intvalue

O funcionamento desta função é próximo ao da Função encrypt_intvalue, tendo apenas o funcionamento oposto. Para desencriptar os números, a função aceita como parâmetros o id do cliente e um texto (String). Começa por criar a chave de encriptação do tipo AES-128 (ACRONIMO) e de seguida converte o texto que recebemos em bytes usando a função **base64.b64decode** (Acrónimo). Tendo obtido o valor em bytes, procede à desencriptação usando a cifra criada anteriormente. Por fim, converte os valores bytes obtidos em uma String desencriptada, convertendo essa String para inteiro e devolvendo-a.

```
# return the client_id of a socket or None
def find_client_id (client_sock):
    for val in gamers:  # val tem o client_id de cada registo
        if client_sock == gamers[val][0]['socket']: # check socket of each entry
            return val # client_id found
    return None


# 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 = AES.new (gamers[client_id][0]['cipher'], AES.MODE_ECB) # Definir o algoritmo de encriptação ten
    result = cipher.encrypt (bytes("%16d" % (int(data)), 'utf8'))
    return str (base64.b64encode (result), 'utf8') # resultado encriptado



# Função para desencriptar 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 = AES.new (gamers[client_id][0]['cipher'], AES.MODE_ECB) # Definir o algoritmo de encriptação ten
    result = base64.b64decode (data)
    result = cipher.decrypt (result)
    return int (str (result, 'utf8')) #resultado desencriptado
```

2.2.5 Função new_msg

A função new_msg é usada como um centro de processamento das mensagens enviadas pelo cliente. QUando um cliente envia um pedido de processamento para o servidor, precisa de conter o campo 'op' que corresponde à operação escolhida pelo utilizador, caso este campo não exista no pedido do cliente, ou apresenta uma operação inválida, o servidor envia uma resposta de erro ao cliente.

Se o cliente enviou um pedido corretamente formulado ao servidor, este pode processar quatro tipos de mensagens:

- **'START'** - Pedido do começo do jogo (Função new_client)
- **'QUIT'** - Processar a desistência do jogador (Função quit_client)
- **'GUESS'** - Processar a adivinha do jogador (Função guess_client)
- **'STOP'** - Processar o término do jogo (Função stop_client)

Cada uma destas operações devolve uma resposta para ser enviada ao cliente, dependendo da operação e do seu sucesso. Por fim, tendo processado a resposta ao pedido do cliente, a função o seu resultado ao cliente.

```
def new_msg (client_sock):
    response = {}
    request = recv_dict (client_sock)
    try : # se um cliente aceder sem o formato correto
        op = request['op'].upper() # Operação escolhida pelo cliente (text.upper para não haver problemas com
    except :
        return None
    #cipher = request['cipher'] -> cifra escolhida pelo utilizador
    print("Request" + str(request)) # DEBUG
    if (op == 'START') :
        response = new_client(client_sock, request)
    elif (op == 'QUIT') :
        response = quit_client(client_sock, request)
    elif (op == 'GUESS') :
        response = guess_client(client_sock, request)
    elif (op == 'STOP') :
        response = stop_client(client_sock, request)
    else :
        response = { 'op': request['op'], 'status': False, 'error' : 'Operação Inválida' }

    print("Response: " + str(response)) # DEBUG

    send_dict (client_sock, response) # enviar o resultado para o cliente
    return response
```

2.2.6 Função new_client

A função new_client processa o pedido de começo de jogo do cliente, devolvendo um dicionário com o número máximo de tentativas, cajo seja processada com sucesso. Se a função não for processada com sucesso, significa que o cliente não está inscrito como como jogador ativo. A função aceita como parâmetros de entrada o socket do cliente e o pedido do jogador.

Além das verificação faladas acima, a função também verifica se o jogador enviou os campos necessários para efetuar o pedido, nomeadamente o campo "client_id".

Se o formato do pedido estiver correto e o cliente existir, a função irá gerar o número secreto para o jogador adivinhar e o número máximo de jogadas. De seguida adicionará o cliente à lista de jogadores ativos e terá em conta se foi escolhido encriptação, tomando as ações necessárias para encriptar os dados. recorrendo à função Função encrypt_intvalue.

```python
def new_client (client_sock, request):
    # obter o request e verificar guardar um novo registo no dicionário gamers
    # Check the return value and print message
    try : # "testar" o campo client_id para verificar se existe
        request['client_id']
    except :
        return { 'op': 'START', 'status': False, 'error': 'Condições inválidas' }

    if (search_gamers(request['client_id']) != None) : # Procurar se o client já está registado
        return { 'op': 'START', 'status': False, 'error': 'Cliente existente' }

    secret_number = random.randint(0, 100) # Gera o número secreto
    print("****** NÚMERO SECRETO: " + str(secret_number) + " de " + str(request['client_id']) + " ******") # DEBUG
    max_Plays = random.randint(10, 30) # Gera o número máximo de tentativas

    try : # testar se o campo cipher existe
        cipherkey = base64.b64decode (request['cipher']) # Decode cypherkey
    except : # se o campo cipher não existir, quer dizer que o utilizador não escolheu encriptar a ligação
        cipherkey = None

    gamers.update({request['client_id'] : [{ 'socket': client_sock, 'cipher' : cipherkey,
            'guess' : secret_number, 'max_attempts' : max_Plays, 'attempts' : 0 }]}) # Atualizar o dicionário

    if(cipherkey != None) : # Se o cliente escolheu comunicar por encriptação
        max_Plays = encrypt_intvalue(request['client_id'], max_Plays) # encriptar o número de jogadas

    return { 'op': 'START', 'status': 'True', 'max_attempts' : max_Plays }
```

2.2.7 Função clean_client

Esta função tem como objetivo, eliminar o cliente da lista de jogadores ativos, para isso, aceita como parâmetro o socket do cliente e procura o id do mesmo. Se o o id não existir, significa que o cliente não está registado, retornando "False". Caso esteje registado, então apaga-o do dicionário de jogadores ativos e devolve "True".

```python
def clean_client (client_sock):
    client_id = find_client_id(client_sock) # Id do cliente devolvido pela função
    if (client_id == None):
        return False
    del gamers[client_id] # eliminar o cliente do dicionário de jogadores ativos
    return True
```

2.2.8 Função quit_client

A função quit_client processa a operação de desistência do jogo, tendo como pré-requisito a existência de o cliente na lista de jogadores ativos e o formato de pedido de operação correto.

Tendo estas condições sido verificadas, o servidor irá processar a desisência do cliente, recorrendo à Função update_file para atualizar os dados do cliente e à Função clean_client para remover o cliente da lista de jogadores ativos.

Se o pedido tiver sido processado corretamente, a função retornará o sucesso da mesma, caso contrário devolverá o erro sucedido.

2.2.9 Função create_file

Esta função é usada para criar o ficheiro "Report.csv"que posteriormente será usado para guardar os resultados dos jogos dos cliente, quando estes o terminam.

2.2.10 Função update_file

A função update_file é usada para atualizar o ficheiro "Report.csv"com os resultados dos jogos dos clientes. Tem como parâmetros o id do cliente e o a linha de dados para serem acrescentados.

A função abre o ficheiro em modo de append, para não sobrescrever os dados, mas sim para acrescentar mais uma linha com resultados ao ficheiro.

```python
def create_file ():
    # create report csv file with header
    file = open('report.csv', 'w') # abrir ou criar o ficheiro se não existir
    writer = csv.DictWriter(file, delimiter=';', fieldnames=['client_id', 'secret_number', 'max_plays', 'current_plays', 'result'])
    writer.writeheader() # Escrever os nomes das colunas
    file.close()
    return None

#
# Suporte da actualização de um ficheiro csv com a informação do cliente e resultado
#
def update_file (client_id, result): # client_id é redudante
    file = open('report.csv', 'a') # abrir o ficheiro em modo append (Não sobrescrever os dados existentes)
    writer = csv.DictWriter(file, delimiter=';', fieldnames=['client_id', 'secret_number', 'max_plays', 'current_plays', 'result'])
    writer.writerow(result) # Escrever uma nova linha no ficheiro
    file.close()
    return None
```

2.2.11 Função guess_client

Para o cliente efetuar uma tentativa do número secreto, é usada a função guess_client, que aceita como parâmetros de entrada o socket do cliente e o pedido.

Antes de efetuar a operação de guess, a função verifica se existe um jogador ativo com o socket fornecido e se o campo "number"foi fornecido no pedido do cliente.

Se as condições anteriores forem verificadas, o servidor contará uma tentativa realizada e caso o cliente tenha escolhido encriptação de dados, irá desencriptar o campo "number", recorrendo à Função decrypt_intvalue. Após ter o valor da tentativa, o servidor irá comparar o número da tentativa com o número secreto que tem armazenado e irá devolver um dos três resultados:

- **'larger'** - Se o cliente escolheu um número inferior ao número secreto;
- **'smaller'** - Se o cliente escolheu escolheu um número superior ao número secreto;
- **'equals'** - Se o cliente acertou o número secreto;

2.2.12 Função stop_client

A função stop_client processa o término do jogo. Aceita como parâmetros de entrada, o socket do cliente e o pedido do cliente.

Para processar a operação, a função garante que existe um jogador ativo, com o socket fornecido, usando a Função find_client_id e se os campo "attempts"e "number"foram fornecido no pedido do cliente. Além disso, a função também

levará em conta se estes campos fornecidos estão encriptados, se estivrem, irá utilizar a Função decrypt_intvalue para proceder à desencriptação.

Caso as condições anteriores se verifiquem, o servidor vai dar inicio ao processamento do término do jogo. Se o cliente **não** tiver enviado um número correto de tentativas ou exceder o número máximo de tentativas, o servidor vai enviar a mensagem de **erro** "Número de jogadas inconsistente". Caso contrário, verifica se o cliente acertou o jogo. Se o cliente tiver acertado o número secreto, envia dicionário com o número secreto para o utilizador e guarda que o cliente acertou, caso contrário, envia também o dicionário para o cliente, mas guarda que o cliente falhou o número secreto.

Após as verificações anteriores, o servidor irá atualizar o ficheiro 'report.csv' usando a Função update_file escolhendo o campo "Result"como "SUCCESS"ou "FAILURE", se o jogador acertou ou falhou o número secreto ou apresenta um número de jogadas inconsistente, respetivamente.

Por fim, o servidor desvincula o client, recorrendo à Função clean_client e remove o cliente da lista de jogadores ativos.

```
def stop_client (client_sock, request):
    client_id = find_client_id(client_sock) # Id do cliente devolvido pela função

    try : # testar se os campos existem
        user_attempts = request['attempts']
        user_number = request['number']
    except : # Se o campo number e attempts não existir no dicionário enviado
        return { 'op': 'STOP', 'status': False, 'error': 'Condições inválidas' }

    if (client_id == None): # Se não existir nenhum cliente no dicionário de jogadores atuais
        return { 'op': 'STOP', 'status': False, 'error': 'Cliente inexistente' }

    if (gamers[client_id][0]['cipher'] != None) : # se o cliente escolheu encriptar os números, vamos desencriptá-los
        user_attempts = decrypt_intvalue(client_id, user_attempts)
        user_number = decrypt_intvalue(client_id, user_number)

    response = {'op' : 'QUIT', 'status' : False, 'error' : 'Número de jogadas inconsistente'} # resposta por defeito
    write = 'FAILURE' # resultado de defeito a ser guardado no ficheiro report.csv

    if (int(user_attempts) == int(gamers[client_id][0]['attempts']) and int(user_attempts) <= int(gamers[client_id][0]['max_attempts'])):
        response = {'op' : 'STOP', 'status' : True, 'guess' : gamers[client_id][0]['guess']} # Se o jogador indicar o número correto de
        if (gamers[client_id][0]['cipher'] != Non (variable) client_id: Any ): encriptar os dados, vamos encriptar o número secreto
            response['guess'] = encrypt_intvalue(client_id, gamers[client_id][0]['guess']) # alterar o dicionário para incluir o dado en
        if ((int(user_number) == int(gamers[client_id][0]['guess']))) : # se o cliente acertou o número secreto
            write = 'SUCCESS' # valor a ser guardado como resultado no ficheiro report.csv

    result = { 'client_id' : client_id, 'secret_number' : gamers[client_id][0]['guess'],
    'max_plays' : gamers[client_id][0]['max_attempts'], 'current_plays' : gamers[client_id][0]['attempts'],
    'result' : write} # Dicionário para guardar no ficheiro json

    update_file(client_id, result) # atualizar o ficheiro report.csv
    clean_client(client_sock) # apagar o cliente do dicionário de jogadores ativos

    return response # devolver a resposta para ser enviada ao cliente
```

2.3 Git

As funcionalidades do git foram muito utilizados neste projeto, desde a simples sincronização de ficheiros e código, até à criação, junção e gestão de branches (**IMAGEM DE EXEMPLO**)

2.4 Code UA

As funcionalidades do Code UA forneceram bastante ajuda ao desenvolvimento projeto, desde a própria visualização dos branches disponíveis, bem como a própria gestão e visualização do código, até à criação de funcionalidades a serem desenvolvidas e bugs a serem resolvidos. Pode visualizar o projeto no Code UA, através do link: http://code.ua.pt/projects/labi2021-ap2-g29

2.5 Exemplos

Capítulo 3

Resultados

Descreve os resultados obtidos.

Capítulo 4

Análise

Analisa os resultados.

Capítulo 5

Conclusões

Com este trabalho, conseguimos solidificar o nosso conhecimento de servidores em python, sockets, interações entre cliente e servidor e criação de algoritmos para tal, bem como encriptação e desencriptação de dados para uma partilha de informação mais segura. Houve também uma aproximação à linguagem Python e a toda à sua sintaxe e características. Sendo Python uma linguagem com bastante procura no mercado de trabalho, a criação do servidor veio ajudar a compreender a sua autenticidade e pas suas diferenças em relação a outras linguagens com que já estamos habituados (Ex: Java) Apesar das adversidades, acreditamos que o trabalho foi conseguido com sucesso, conseguimos criar um servidor com o jogo referido e com todas as características necessárias para tal, utilizando os recursos que nos foram dados e auxiliando a sua compreensão com este relatório.

Contribuições dos autores

Resumir aqui o que cada autor fez no trabalho. Usar abreviaturas para identificar os autores, por exemplo AS para António Silva. No fim indicar a percentagem de contribuição de cada autor.

Acrónimos