Trabalho de Aprofundamento 2, Grupo 27

Universidade de Aveiro

Alexandre Martins, Luís Silva

Trabalho de Aprofundamento 2, Grupo 27

DETI

Universidade de Aveiro

Alexandre Martins, Luís Silva
(103552) alexandremartins@ua.pt, (103617) afonsotorres@ua.pt

30 de maio de 2021

Agradecimentos

Gostaríamos de agradecer a:

Professor António Manuel Adrego da Rocha , pelas aulas esclarecedoras que nos foram dadas sobre LaTeXno semestre passado, sem as quais não conseguiríamos ter completado este relatório e pelas aulas em python, linguagem na qual o trabalho foi realizado, e extensão do prazo de entrega.

João Rodrigo Faria (103361) , pelo pequeno mas relevante apoio que foi dado em LaTeXna estrutura do relatório.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este trabalho, desenvolvido no âmbito da Unidade Curricular (UC) de Laboratórios de Informática, consistiu na criação de um servidor que suporte a geração de um número inteiro aleatório (entre 0 e 100), designado por número secreto, bem como o número máximo de tentativas (entre 10 e 30) concedidas para o adivinhar. E um cliente que permita adivinhar esse número secreto. Ou seja um jogo de adivinha o número secreto.

O projeto do code.ua associado a este trabalho pode ser consultado em `http://code.ua.pt/projects/labi2021-ap2-g27/files`

Capítulo 2

Metodologia

Exposição da metodologia utilizada para a realização do projeto e obtenção de resultados.

2.1 client.py

2.1.1 Objetivos

O **client** contacta o servidor para dar início ao jogo (operação **START)** enviando um dicionário com o seguinte formato:

{"op": "START", "client _id": nome identificador do cliente}

Após a inicialização do jogo com **START** o jogador tem 3 opções, **QUIT**, **GUESS**, **STOP**:

- **QUIT**

O cliente contacta o servidor para desistir do jogo enviando um dicionário com o seguinte formato: { "op": "QUIT"}

- **GUESS**

O cliente contacta o servidor para fazer uma jogada enviando um dicionário com o seguinte formato: { "op": "GUESS", "number": valor numérico entre 0 e 100}

- **STOP**

O cliente contacta o servidor para terminar o jogo (operação STOP) enviando um dicionário com o seguinte formato: { "op": "STOP","number": último valor, "attempts": jogadas efectuadas }

2.1.2 Métodos

- **run_client()**

Recebe as acções do jogador e processa o jogo.

- **quit_action()**

Desiste do jogo, caso se verifiquem as condições necessárias para o fazer.

- **validate_response()**

Verifica se a resposta do servidor é válida.

2.2 server.py

2.2.1 Objetivos

O ficheiro **server.py** consiste num conjunto de métodos essenciais para a conexão e troca de mensagens com o servidor remoto.

- **START**

Após o comando **START** o servidor só deverá aceitar o cliente caso ele não esteja já inscrito na lista de clientes ativos do servidor. Nesse caso gera um número aleatório entre 0 e 100, um número máximo aleatório de jogadas entre 10 e 30 e inicializa o número de jogadas. Depois deve acrescentar este registo de cliente à sua estrutura de dados de clientes ativos (para posteriormente evitar a inscrição de outro cliente com a mesma identificação) e devolve um dicionário indicando que a operação de registo deste cliente foi feita com sucesso e indicando-lhe quantas jogadas ele dispõe (de maneira a ele poder controlar o seu jogo) com o seguinte formato:

{"op":"START","status": True, "max_attempts": nº máximo de jogadas}

Se pelo contrário um cliente com o mesmo identificador já se encontra a jogar então devolve-lhe um dicionário indicando que a operação de registo deste cliente não teve sucesso com o seguinte formato:

{ "op": "START", "status": False, "error": "Cliente existente"}

- **QUIT**

O servidor só deverá aceitar a desistência do jogador caso ele esteja presentemente ativo. Nesse caso atualiza o ficheiro de resultados com os dados do cliente e o resultado indicando a sua desistência ("**QUIT**"), remove o registo de cliente da sua estrutura de dados de clientes ativos (para posteriormente aceitar a

inscrição de outro cliente com a mesma identificação) e devolve um dicionário indicando que a operação de desistência deste cliente foi feita com sucesso com o seguinte formato:

{"op": "QUIT", "status": True}

Se pelo contrário o cliente não está presentemente a jogar então devolve-lhe um dicionário indicando que a operação de desistência deste cliente não teve sucesso com o seguinte formato:

{"op": "QUIT", "status": False, "error": "Cliente inexistente"}

- **GUESS**

O servidor só deverá aceitar a jogada do cliente caso ele já esteja inscrito na lista de clientes ativos do servidor. Nesse caso contabiliza essa jogada do cliente, compara o valor indicado pelo cliente com o número secreto. Se o valor indicado pelo cliente for maior do que o número secreto o servidor indica que na próxima jogada deve tentar um valor menor ("smaller"). Se pelo contrário o valor indicado pelo cliente for menor do que o número secreto o servidor indica que na próxima jogada deve tentar um valor maior ("larger"). Se o cliente acertou então o servidor indica que os valores são iguais ("equals"). O servidor envia ao cliente um dicionário com o seguinte formato:

{"op": "GUESS", "status": True, "result": "smaller"/"larger"/"equals"}

Se pelo contrário o cliente não está presentemente a jogar então devolve-lhe um dicionário indicando que a operação de adivinhação deste cliente não teve sucesso com o seguinte formato:

{"op": "GUESS", "status": False, "error": "Cliente inexistente"}

- **STOP**

O servidor só deverá aceitar a terminação do jogo caso o jogador esteja presentemente ativo. Nesse caso o servidor verifica se o número de jogadas indicadas pelo cliente é igual à sua própria contagem. Se o jogador indicar um número correto de jogadas então o servidor deve verificar o número recebido. Se ele for igual ao número secreto que o cliente devia adivinhar - e se o cliente não excedeu o número máximo de jogadas - então o resultado do jogo é de sucesso ("SUCCESS") caso contrário é de insucesso ("FAILURE"). Depois o servidor atualiza o ficheiro report.csv com os dados do cliente e o resultado indicando sucesso ou insucesso, remove o registo de cliente da sua estrutura de dados de clientes ativos (para posteriormente aceitar a inscrição de outro cliente com a mesma identificação) e devolve um dicionário indicando que a operação de terminação deste cliente foi feita com sucesso e indica-lhe qual era o número

secreto com o seguinte formato:

{"op": "STOP", "status": True, "guess": número secreto}

Quando o cliente recebe esta informação do servidor escreve no monitor uma mensagem a indicar se adivinhou ou não o número secreto e quantos jogadas efectuou.

Se pelo contrário o cliente não está presentemente a jogar ou enviou uma contagem de jogadas inconsistente o servidor devolve-lhe um dicionário indicando que a operação de terminação não teve sucesso e indicando o erro de "Cliente inexistente"ou de "Número de jogadas inconsistente", com o seguinte formato:

{"op": "QUIT", "status": False, "error": um dos erros indicados em cima}

2.2.2 Métodos

- **main()**

Este método contem a lógica principal de todo o projeto. É este que é chamado quando se executa o programa. Pode ser dividido em duas partes:
- Estabelecimento de conexão (encriptada) com o servidor.

- Receção e utilização contínua dos dados recebidos.

- **find_client_id()**

Devolve o ID do utilizador através do socket.

- **new_msg()**

Recebe as ações do client.

- **new_client()**

Gera um novo objecto relativo ao cliente ao qual é atribuído um socket.

- **clean_client() e quit_client()**

Removem um client que for especificado do dicionário global.

- **create_file() e update_file()**

Encarregadas de gerar e atualizar os ficheiros CSV.

- **guess_client()**

Compara o valor introduzido pelo utilizador ao valor do número secreto registado no dicionário global e devolve uma string relativa à comparação dos dois.

- **stop_client()**

Atualiza o ficheiro CSV com o resultado do jogo.

2.3 Segurança

2.3.1 Contexto

Troca do segredo partilhado

A troca de chaves Diffie Hellman é um método de troca segura de chaves criptográficas em um canal público e foi um dos primeiros protocolos de chave pública concebidos por Ralph Merkle e nomeados em homenagem a Whitfield Diffie e Martin Hellman. DH é um dos primeiros exemplos práticos de troca de chave pública implementada no campo da criptografia.
O método de troca de chave Diffie – Hellman permite que duas partes que não têm conhecimento prévio uma da outra estabeleçam em conjunto uma chave secreta compartilhada em um canal inseguro. Essa chave pode então ser usada para criptografar comunicações subsequentes usando uma cifra de chave simétrica.

Encriptação

Após ser gerado o segredo, é possível agora a **cifra das mensagens**, para que assegure um nível relativamente satisfatório de segurança.

Capítulo 3

Testes

Para o fazer, primeiro foram declaradas diversas variáveis com strings válidas e inválidas, e, para cada string válida, um dicionário correspondente (para comparar com o resultado de com a string usada como argumento).

Não foram feitos testes unitários para as restantes funções visto que essas dependiam de comunicação com o servidor.

Capítulo 4

Conclusões

4.1 Resultados

No que toca aos resultados obtidos pela execução do programa, este funciona corretamente.

Como mostra a imagem 4.1, observa-se que os dados que são enviados pelo servidor são corretamente impressos no terminal.

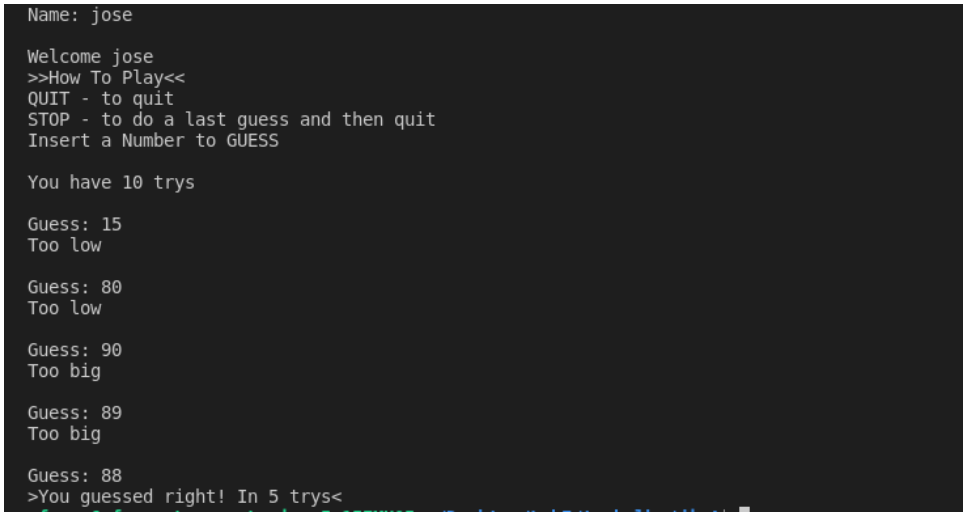

Figura 4.1: Simulação da tentativa de um jogador chamado jose

Na imagem 4.2 está representada uma simulação do que aconteceria se um jogador escolhesse fazer **STOP**

Figura 4.2: Exemplo da utilização de **STOP**

Na imagem 4.3 está representada uma simulação do que aconteceria se um jogador escolhesse fazer **QUIT**

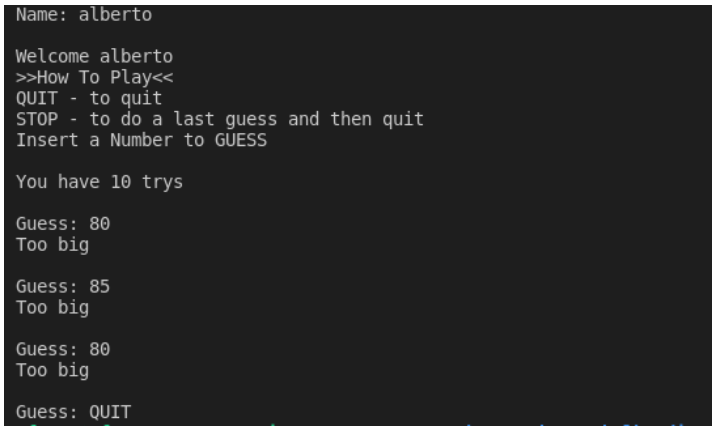

Figura 4.3: Exemplo da utilização de **QUIT**

4.2 Conclusão

Este trabalho de aprofundamento permitiu aos autores do mesmo consolidar a implementação de sockets, manipulação de estruturas de ficheiros ou mensagens na linguagem de programação Python. Para além disso, forneceu uma ideia

geral na negociação de um segredo partilhado pelo método de Diffie-Hellmann.

Contribuições dos autores

A contribuição de ambos os membros para o bom funcionamento do grupo e projeto foi igual, isto é 50% 50%.

Acrónimos

UC Unidade Curricular

Capítulo 5

Bibliografia

Enunciado utilizado na criação do projeto, do qual foram retirados definições de funções e partes do código.

Pagina utilizada para melhor compreender o conceito de criptografia.