

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

Eventuais agradecimentos. Comentar bloco caso não existam agradecimentos a fazer.

Índice

1	Introdução	1
2	Metodologia	2
2.1	Cliente	2
2.1.1	Função main	2
2.1.2	Função run_client	3
2.1.3	Função validate_response	4
2.1.4	Função quit_operation	4
2.2	Servidor	4
2.2.1	Função main	4
2.2.2	Função find_client_id	5
2.2.3	Função encrypt_intvalue	5
2.2.4	Função decrypt_intvalue	5
2.2.5	Função new_msg	6
2.2.6	Função new_client	6
2.2.7	Função clean_client	6
2.2.8	Função quit_client	7
2.2.9	Função create_file	7
2.2.10	Função update_file	7
2.2.11	Função guess_client	7
2.2.12	Função stop_client	8
2.3	Git	8
2.4	Code UA	8
2.5	Exemplos	8
3	Resultados	9
4	Análise	10
5	Conclusões	11

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 caracteres 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 resposta 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.

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 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.

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.

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`.

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"`.

2.2 Servidor

Nesta secção será apresentada e explicada a metodologia do código 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 conexões dos clientes.

Com o servidor a funcionar, este vai ficar à espera de conexões 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 conexão 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 descriptar 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 descriptação usando a cifra criada anteriormente. Por fim, converte os valores bytes obtidos em uma String descriptada, convertendo essa String para inteiro e devolvendo-a.

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.

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 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.

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 id não existir, significa que o cliente não está registado, retornando "False". Caso esteja registado, então apaga-o do dicionário de jogadores ativos e devolve "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 desistência 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.

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 descriptar 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 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 campos "attempts" e "number" foram fornecidos no pedido do cliente. Além disso, a função também levará em conta se estes campos fornecidos estão encriptados, se estiverem, irá utilizar a Função decrypt_intvalue para proceder à descriptação.

Caso as condições anteriores se verifiquem, o servidor vai dar início 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.

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 as 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