Trabalho de Sistemas Distribuídos - Computação Paralela e Distribuída

O trabalho a que o presente documento diz respeito, no âmbito da unidade curricular Computação Paralela e Distribuída, tem como objetivo criar um sistema cliente-servidor utilizando sockets TCP em Java que permita que vários utilizadores joguem o jogo "Quem acerta mais perto". Para isso, a solução desenvolvida permite que os jogadores se autentiquem e, depois, se juntem a um de dois lobbies (Simple Lobby e Rank Lobby), que constituirão partidas com um número de jogadores parametrizável, de formas distintas. Uma vez formado o grupo de jogo, uma nova instância de Game é criada, dando início a uma partida entre os jogadores selecionados.

O Jogo

Uma partida do jogo implementado é composta por rondas eliminatórias. Em cada ronda, é gerado um número inteiro aleatório de 0 a 100. Cada jogador é desafiado a adivinhar qual o número em jogo, dando o seu palpite.

No final da ronda, o jogador cujo palpite estiver mais afastado da resposta correta é eliminado, fazendo com que o jogo prossiga para a seguinte ronda com menos um concorrente, e assim sucessivamente.

Naturalmente, a última ronda será disputada entre dois jogadores, vencendo aquele cujo palpite se encontre mais perto do número real nessa ronda.

Compilação e Execução

Antes de executar qualquer outra ação, deve executar comando `make`, para compilar os ficheiros.

Depois, para correr o servidor, deve executar o comando `make run_server`.

É possivel também iniciar o servidor parametrizando a porta onde o mesmo estará à escuta e o número de jogadores necessários para iniciar um jogo, com `make run_server PORT=<PORT> NUM_PLAYERS=<NUM_PLAYERS>`.

Para correr o cliente, deve executar o comando `make run_client`.

É possivel também correr o cliente personalizando o host e a porta onde procurará o servidor, com `make run_client HOST=<HOST> PORT=<PORT>`.

Servidor

Arquitetura

Base de Dados

Ficheiro `csv` que assegura a persistência da informação dos jogadores registados, nomeadamente: nome de utilizador (*username*), palavra-passe e pontuação total.

Server.java:

Esta classe representa o servidor. Inicialmente apenas existe uma thread que é responsável por aceitar novas conexões. Sempre que uma nova conexão é estabelecida (novo socket), uma nova thread é criada, que lida com todas as mensagens provenientes do socket correspondente.

Seguem-se as mensagens que o servidor espera receber dos clientes, caracterizadas pela sua sintaxe e pela ação que despoleta:

- `AUTH <username> <password>`
    - Ação: Autentica o jogador com username e password fornecidos. Se a mesma for bem-sucedida, enviará uma mensagem com o token atribuído ao cliente, que o deve utilizar nas mensagens seguintes, para se identificar. Caso o servidor detete que o jogador se encontrava num lobby ou num jogo, envia essa informação ao cliente para que este consiga restaurar o seu estado interno correspondente.
- `REGISTER <username> <password>`
    - Ação: Regista um novo jogador com username e password fornecidos.
- `SIMPLE <token>`
    - Ação: Adiciona o jogador ao lobby simples, se o jogador não estiver num lobby, nem num jogo.
- `RANK <token>`
    - Ação: Adiciona o jogador ao rank lobby, se o jogador não estiver num lobby, nem num jogo.
- `LEAVE_LOBBY <token>`
    - Ação: Caso o jogador esteja num lobby, remove-o do mesmo.
- `POINTS <token>`
    - Ação: Devolve o número de pontos que o jogador tem atualmente.
- `PLAY <token> <guess>`
    - Ação: Caso o jogador esteja num jogo, define o valor indicado como o seu palpite na jogada atual.

Caso o servidor receba uma outra mensagem que não esteja listada a cima, responderá com `ERROR: Command: Invalid command.`.

Player.java

Esta classe representa um jogador, **do ponto de vista do servidor**.

Além disso, na classe é guardado, de forma estática, o conjunto de todos os jogadores que possuams sessão iniciada no momento.

Em cada objeto é guardado o seu nome de utilizador, o seu número de pontos, o seu token atual e o último socket através do qual o jogador comunicou com o servidor. A password apenas é guardada na base de dados. Embora o socket correspondente ao cliente seja guardado em cada objeto, o jogador pode enviar mensagens através qualquer socket (desde que devidamente identificadas com o respetivo token). Caso seja diferente do guardado, este será atualizado.

Todas as mensagens enviadas para o jogador sê-lo-ão através do último socket conhecido.

SimpleLobby.java

Esta classe é a implementação do lobby simples. Qualquer jogador pode juntar-se a este lobby. Assim que nele se encontrar o número de jogadores necessário para iniciar uma partida, é iniciado um jogo com os mesmos, independentemente da sua pontuação, e o lobby é esvaziado.

RankLobby.java

Aqui é implementado o lobby por rank.

Este lobby tenta criar jogos com jogadores cujas pontuações totais sejam similares, mas esta condição vai sendo gradualmente relaxada, de modo que os jogadores não tenham que esperar eternamente por um jogo.

Qualquer jogador pode juntar-se a este lobby. Ao fazê-lo, ser-lhe-á associado um raio de pesquisa, que ditará a amplitude de pontuações de jogadores com quem poderá começar a jogar, inicializado a 0 pontos e incrementado em 5 unidades a cada segundo.

Além disso, a cada segundo, o RankLobby cria todos os conjuntos em que parte do raio de pesquisa esteja contido em todos os outros elementos destes conjuntos.

Se algum destes conjuntos contiver o número de jogadores necessário, é iniciado um jogo com os mesmos, e os jogadores são retirados do RankLobby.

O processo é executado por uma thread dedicada apenas ao Rank Lobby.

Para criar os conjuntos, o programa ordena todos os valores de entrada e saída (por exemplo, um jogador com pontuação de 2000 pontos e com um raio de pesquisa, no momento, de 300 terá um valor de entrada de 2300 e 1700 como valor de saída). De seguida, ocorre iteração pela lista ordenada, e sempre que um valor de entrada aparece, o jogador que lhe corresponde é adicionado à lista de intervalos abertos; sempre que um valor de saida aparece, o jogador correspondente é removido da lista de intervalos abertos e uma cópia deste conjunto é adicionada à lista de conjuntos cujos raios de pesquisa estejam contidos em todos os outros intervalos do grupo.

Game.java

Responsável pela implementação de uma partida do jogo e da interação ao longo das suas rondas.

Cliente

Arquitetura

Client.java

A classe Client é o ponto de entrada principal do *client-side*. Mantém um estado interno influenciado pelo *input* do utilzador e pelas mensagens recebidas do servidor. É responsável por garantir a interface textual de utilizador. Ainda, guarda o token fornecido pelo servidor, essencial para se identificar nas mensagens que lhe envia.

ClientState.java

Essencial para a representação interna do estado de um cliente. Especifica os estados possíveis e as transições entre os mesmos, consoante as mensagens recebidas do servidor.

ClientStub.java

Atua como ponto de entrada e de saída de comunicações com o servidor. Estabelece e encerra o socket de comunicação e assegura o envio e receção de mensagens de e para o servidor a que o socket se associa, consoante *hostname* e porta especificados.

Exemplo de Execução

Ao iniciar a aplicação, o utilizador verá o menu de autenticação com opções de autenticação, registo e saída. Dependendo da escolha, a aplicação irá solicitar as informações necessárias e comunicar com o servidor, que processá-las-á, desencadear a ação correspondente e retornar uma resposta.

Após autenticação, o utilizador pode escolher entre diferentes lobbies de jogos.

Durante uma partida, será chamado a dar o seu palpite (efetivamente, executar a sua jogada) e ser-lhe-á apresentada informação sobre o desenrolar do jogo, incluindo a própria prestação, bem como a dos adversários, em cada ronda.

SSL

Todas as comunicações entre o servidor e cliente são levadas a cabo por sockets que respeitam o protocolo Secure Sockets Layer (SSL), através da biblioteca SSLSocket, de modo a manter todas as mensagens seguras, principalmente as referentes à autenticação e registo dos jogadores.

Para o efeito, foi criado um certificado (`src/server/certificate/keystore.jks`), indispensável para que um cliente possa estabelecer um socket de comunicação com o servidor.

Nota

O mecanismo de detação de tecla pressionada presente na classe Keyboard foi adaptado do Stack Overflow (https://stackoverflow.com/questions/18037576/how-do-i-check-if-the-user-is-pressing-a-key), dada a complexidade do mesmo.

Grupo 17, 20/05/2024

- António Augusto de Sousa, 202000705
- Pedro de Almeida Lima, 202108806
- Pedro Simão Januário Vieira, 202108768