universidade de aveiro  deti departamento de electrónica, telecomunicações e informática

Relatório "Distributed Black Jack"

Licenciatura em Engenharia Informática

2020/2021 2º Semestre

Pedro Jorge 98418

Introdução

Com este projeto, no âmbito da cadeira Computação Distribuída e com vista a reunir todas as competências lecionadas nas aulas teórico-práticas e práticas da disciplina, foi pedido aos alunos para desenvolver um jogo P2P de Black Jack.

Está em avaliação o protocolo definido e implementado no enunciado do projeto, assim como as *features* implementadas de acordo com os objetivos incrementais:

- player.py consegue jogar um jogo solitário;
- 2 player.py conseguem jogar um jogo;
- 3 player.py conseguem jogar um jogo;
- "bad_player" implementado no mínimo com as opções mencionadas no enunciado;
- Restantes jogadores são capazes de detetar que houve batota no jogo.

Protocolo

No que toca ao protocolo, a comunicação entre jogadores foi realizada implementando um protocolo peer-to-peer. A conexão entre jogadores foi efetuada através de sockets TCP, em cada conexão entre jogadores é criada uma socket para enviar ou receber mensagens. Por fim, executa-se "socket.close()" para prevenir erros decorrentes de sockets que não foram fechadas. O padrão de mensagens seguido foi o seguinte:

- "H"- Hit / Jogador pede uma carta;
- "S"- Stand / Jogador não joga e passa a vez a outro;
- "W"- Vitória / Jogador declara ter vencido o jogo;
- "D"- Derrota / Jogador declara ter perdido o jogo;
- "TB" – Tudo Bem / Não há diferença entre o hash fornecido e o hash calculado;
- "B" – Batota / Há diferença entre o hash fornecido e o hash calculado;

- "I" – Indisponível / Ocorre quando não é possível estabelecer ligação com o deck.

Funções Usadas

Maior parte das funções já se encontram comentadas no código, contudo fica aqui a lista das funções usadas na implementação:

- conectar_Redis() – Conecta com o servidor Redis no localhost;
- obter_Hash() – Obter o hash das cartas distribuídas;
- obter_Carta() – Pede uma carta ao deck. Pode retornar "I" caso a ligação não seja bem-sucedida;
- aceitar_Conexao() – Invocada pelo selector ao testar a conexão com os jogadores. Recebe e envia o nome do jogador;
- receber_Mensagem() – Invocada pelo selector para receber uma mensagem de outro jogador;
- informa_Jogadores() – Estabelece ligação com cada porta e envia mensagens. Recebe como argumentos uma mensagem e uma lista de portas e número do jogador para onde vai enviar a mensagem.

Desenvolvimento

Single Player / Solitário

O modo single-player/solitário é iniciado quando não são passados argumentos "-p", apenas o argumento "-s". O jogo começa com a obtenção de duas cartas iniciais e depois desenrola-se de forma normal, o jogador pode pedir cartas, passar a vez e declarar que ganhou ou perdeu o jogo (não vai ser feita a verificação de quem ganhou).

Multiplayer / Vários jogadores

Quanto ao modo Multiplayer, é iniciado quando são passados pelo menos 1 argumento "-p" (2 jogadores a jogar) ou 2 argumentos "-p" (3 jogadores a jogar).

O jogo começa perguntando ao jogador qual é o seu **número**, de forma a identificá-lo no decorrer do jogo, e esse número vai ser guardado na lista "conectados", assim como a porta passada no argumento "-s".

De seguida vai ser iniciado um **teste de ligação** entre todos os jogadores, primeiro, o jogador vai tentar estabelecer ligação com os restantes e caso consiga vai enviar o seu número e número de porta para ligações futuras e recebe como resposta o número do jogador conectado.

Caso haja ligações que falharam, o **programa vai entrar em modo de espera** e aguardar pelos restantes jogadores para se conectarem. Para tal, é usado um *selector* no modo EVENT_READ e uma socket ligada ao localhost e à porta do jogador.

É usado um ciclo **"while = True"** porque não se sabe ao certo quantos turnos é que o jogo vai ter, cada turno vai ocorrer de forma ordenada e da menor para a maior porta. Quando um jogador identifica que está na sua vez de jogar "(conectados[jogador_Atual][1] == self_port)" verifica-se se é a ronda inicial ou não, visto que caso seja a ronda inicial, vão ser pedidas 2 cartas ao deck (através da função obter_Carta()) e de seguida é invocada a função "interact_with_user1" que vai retornar 1 dos 4 valores ("H", "S", "W" ou "D"). Esse valor é adicionado à lista "jogadas" de forma a saber a ordem das cartas jogadas pelos jogadores. Por sua vez, a função "interact_with_user1" pode devolver:

- **"H"** – Pede uma carta ao deck e esta é guardada na variável "ultima_Carta" e só será adicionada à lista "minhas_Cartas" no próximo turno. Por fim, a função "informa_Jogadores()" vai avisar os outros jogadores desta ação;
- **"S"** – Neste caso declara-se a variável "last_card = "" " para que no próximo turno não seja adicionada nenhuma carta à lista "minhas_Cartas". Por fim, a função "informa_Jogadores()" vai avisar os outros jogadores desta ação;
- "**W**" – Vitória do jogador e break do ciclo "while = True". Por fim, a função "informa_Jogadores()" vai avisar os outros jogadores desta ação
- "**D**" – Derrota do jogador e remoção da lista "aJogar". Por fim, a função "informa_Jogadores()" vai avisar os outros jogadores desta ação

Caso o programa identifique que não seja a sua vez de jogar, espera que o outro jogador o informe da sua jogada.

Todo este processo só acaba quando um jogador declarar que ganhou o jogo ou só ficar um jogador a jogar (implica que os restantes tenham perdido o jogo/excedido os 21 pontos).

De seguida vai ser executado o processo de **colocar as cartas na mesa através do REDIS.** É um processo feito de forma crescente, de acordo com as portas dos jogadores.

Se o programa identifica que está na sua vez, inicia um cliente REDIS e guarda todas as cartas numa lista com a mesma chave que "self_port". Senão, vai esperar receber uma mensagem de "TB" do outro jogador.

O processo de **obter as cartas dos outros jogadores** é feito através de uma procura da lista de cartas de cada jogador no REDIS. De seguida, o programa vai guardar a porta, pontuação e lista de cartas do jogador na lista "pontuação_Jogadores".

A forma de **determinar o vencedor do jogo** é bastante simples, vai-se utilizar a lista "pontuação_Jogadores" para consultar a pontuação de cada jogador e:

- Se o jogador tiver 21 pontos ganha automaticamente o jogo;
- Se o jogador tiver mais do que 21 pontos perder automaticamente e está fora do jogo;
- Se o jogador tiver menos do que 21 pontos, mas mais pontos que o jogador anterior a si, ganha o jogo.

Cada jogador consegue saber/verificar quem ganhou ou perdeu o jogo.

Para ajudar na tarefa de encontrar os batoteiros o deck fornece um **md5sum.** Os dois jogadores com maior porta vão ficar encarregues de pedir e comparar a hash das cartas com a hash do deck e informar os jogadores se houve batotice ou não. Para pedir a hash das cartas, usa-se a função "obter_Hash()" e para verificar usa-se a lista "jogadas", que vai conter as cartas distribuídas pelo deck, as jogadas e o jogador que as efetuou.

De seguida, calcula-se a hash dessa lista, compara-se com a hash do deck e informa-se os restantes jogadores:

- "TB"- Se os hash forem iguais;
- "B" – Se os hash forem diferentes.

Os jogadores não escolhidos para calcular a hash vão ficar à espera de receber a mensagem (hash_jogado1 e hash_jogado2) dos outros jogadores.

Para fazer a **validação do jogo,** esta é feita comparando o hash calculado e o hash do deck e pelos mesmos jogadores que verificaram o hash anteriormente. Senão, vai ser verificada através da comparação das duas mensagens que os jogadores que verificaram o deck enviaram para os restantes jogadores. Se a mensagem for "TB", então não houve batota.

Por fim, é feito o close das *sockets* e dos *selectors* que foram usados e o jogo acaba.

Bad_Player

O bad_player (malcomportado) funciona de forma semelhante ao jogador "normal", tanto em termos de código como em termos funcionais, tendo apenas algumas diferenças:

- Sempre que pede uma carta, pode tirar outra carta e eliminar a que acabou de tirar, de forma a enganar o sistema;
- Sempre que é escolhido para verificar a hash das cartas e tenha havido batota, pode mentir sobre a hash e informar os restantes jogadores que a hash está correta e enganar os jogadores.

Conclusão

Concluindo, o desenvolvimento deste projeto foi importante e útil para meter em prática todas as competências adquiridas ao longo da cadeira "Computação Distribuída" e penso que, de forma geral, o trabalho desenvolvido cumpre os requisitos inicialmente impostos, implementa o protocolo peer-to-peer e é tolerante a falhas.