Distributed Black Jack

Universidade de Aveiro

Licenciatura em Engenharia Informática

Computação Distribuida

Ano Letivo 2022/23

Professores:

Diogo Gomes

Nuno Lau

Trabalho realizado por:

Cristiano Nicolau 108536

Introdução

No âmbito da unidade curricular de Computação Distribuída, foi nos proposto um projeto que consiste no desenvolvimento de um jogo P2P de black jack.

Este deveria ser 100% distribuído e sem casa/dealer, existindo apenas um servidor central com os baralhos de cartas infinitos que atende um jogador de cada vez, onde os jogadores competem entre si.

Requerimentos Utilizados

Todos os players comunicam uns com os outros e com o server central através de sockets, onde os argumentos, da porta, são passados no terminal. Estes argumentos servem ainda para identificar cada player, ou seja, o nome do player é o número da sua porta de conexão. No fim do jogo cada player, coloca as suas cartas na mesa, que neste projeto é implementada através de um servidor REDIS que guarda para cada jogador (chave é a sua porta de conexão) a lista de cartas.

Funcionamento do Sistema

O programa contém 3 scripts executáveis. O primeiro é o *player.py,* que cria e conecta-se aos outros players através de sockets TCP. Como este projeto respeita o protocolo P2P, o player.py funciona tanto como cliente quanto como servidor, permitindo compartilhamentos dados sem a necessidade de um servidor centralizado. O segundo é o *bad_player.py*, que acrescenta a possibilidade de um jogador fazer batota e mentir sobre as suas cartas, pontos, …, que é baseado no código de “player.py”

Por último temos o *deck.py*, que comunica com cada player através de sockets, que tem um protocolo próprio. O desck.py é um servidor central, com baralhos de cartas infinitos, que atende todos os players.

Protocolo

Todas as mensagens trocadas entre os players e o server, deck.py, respeitam um protocolo request-response em plain text:

- “GC”: devolve 2 chars e um \n correspondendo ao valor de cara de uma carta
- “HC”: devolve 32 chars e um \n com o md5sum da lista de strings com valor de cara das cartas já distribuídas

As mensagens trocadas entre os players, para manter todos atualizados em relaçao ao jogo são as seguintes:

1. **Mensagem de início do turno:**
   - **Descrição:** Esta mensagem é enviada pelo jogador atual, para todos os jogadores, onde cada jogador, no caso do seu ser igual id ser igual ao id do próximo jogador, arranca o seu turno. Alem disso, todos os jogadores atualizam o dicionário onde guardam os status de todos os outros jogadores.
   - **Formato:** "Next Player: {next_player_port}; port_player_anterior: {self_port}, player_status_anterior:{player_status[self_port]}
2. **Mensagem de jogador parado:**
   - **Descrição:** Esta mensagem é enviada pelo jogador atual para os outros jogadores quando ele escolhe pedir uma carta adicional do baralho.
   - **Formato:** "Player {self_port} draw another card"
3. **Mensagem de pedido de carta:**
   - **Descrição:** Esta mensagem é enviada pelo jogador atual para os outros jogadores quando ele decide parar de pedir cartas e encerrar seu turno, esperando a descoberta do vencedor. Cada jogador so pode dar stand uma vez, apos isto fica a espera do fim do jogo.
   - **Formato:** "Player {self_port} stood"
4. **Mensagem de vitória:**
   - **Descrição:** Esta mensagem é enviada pelo jogador atual para os outros jogadores quando ele declara vitória e encerra o jogo. Sendo de seguida, averiguado se o jogador realmente venceu o jogo.
   - **Formato:** "Player {self_port} claimed victory"
5. **Mensagem de derrota:**
   - **Descrição:** Esta mensagem é enviada pelo jogador atual para os outros jogadores quando ele decide desistir e encerrar o jogo, esperando a descoberta do vencedor.
   - **Formato:** "Player {self_port} folded in defeat"

6. **Mensagem de pedido de carta:**
   - **Descrição:** Esta mensagem é enviada pelo jogador atual para os outros jogadores quando o jogo acaba e é necessário verificar quem é o vencedor. Envia também a informação para todos os jogadores atualizarem o dicionário dos status.
   - **Formato:** "who won; port_player anterior: {self_port}, player_status_anterior:{player_status[self_port]}"

Deck.py

O deck.py, é o server central, que serve como baralho infinito de cartas, que responde a cada player, apos o seu pedido.

Quando o player envia um pedido “GC”, uma carta de um baralho de cartas é escolhida de forma aleatoria, e é enviada para o player.

Temos ainda a opçao “HC”, que é um sistema anti batota, que apenas pode ser requisitada, no máximo, 2 vezes por jogo, que envia a Hash de todas as cartas em jogo.

Player.py

O player.py, respeita o protocolo P2P, ou seja, cada player, tanto é cliente como servido.

Quando cada user executa o script, tem de incluir a self port, *-s*, e tem de incluir a porta de todos os restantes players, *-p*. Cada player, no inicio cria a própria socket, na porta que passou com argumento –s, apos isto conecta-se a todos os players que vão jogar, conectando-se através de sockets nas portas passadas em –p.

Apos todos os jogadores se terem conectado uns aos outros, o jogo começa, no início são distribuídas 2 cartas para cada player, ou seja, cada player pede ao server deck.py 2 cartas. E o jogador com menor porta/id começa o seu turno.

Sempre que é o turno de um jogador, o mesmo só pode jogar caso, ainda não tenha feito nenhum turno, ou tenha pedido uma carta no turno anterior. No caso de o player, ter desistido, este fica a espera do fim do jogo para saber quem é

o vencedor. No caso de stand, cada jogador só pode dar stand uma vez, ficando assim até ao fim do jogo a espera de averiguar quem foi o vencedor.

Quando inicia o turno de um jogador e no caso deste poder jogar aparece uma interface, onde o jogador pode pedir mais uma carta, dar stand, desistir ou declarar-se vencedor. Enquanto o jogador joga o seu turno, os outros jogadores esperam a mensagem do próximo jogador a jogar. Quando o jogador termina o seu turno, envia a mensagem com a porta do próximo jogador a jogar e esse inicia o seu turno.

Quando já não existe mais nenhuma possibilidade de turnos, ou seja, todos os jogadores têm status de 'stand', ou no caso de já só haver um jogador a poder jogar, todos restantes desistiram ou deram bust, ou ainda caso algum jogador se declare como vencedor, passamos para a averiguação de quem ganhou.

O server testa os mecanismos de anti batota e analisa quem é o vencedor e apos isto, diz quem venceu a todos os jogadores no caso de não ter havido batota e diz quem é o batoteiro no caso de ter havido batota.

Possíveis formas de vencer, o jogador ter atingido um score de 21 (o score é calculado com a ajuda do script utils.py, que descobre a pontuação de todas as cartas de um utilizador), ser o único jogador em jogo com score menor/igual a 21, ou seja, todos os outros tenham desistido ou dado bust (quando pede uma carta, arrebentar e ultrapassar 21). No caso de já só existirem jogadores em stand, é todos com score menor a 21, é calculado e o que tem maior score é declarado como vencedor.

Bad_Player.py

O bad_player é um agente malcomportado, é um jogador que faz batota, e que os seus adversários não sabem, mas devem descobrir no fim do jogo que houve batota no jogo.

O bad _player é baseado no código do player bem-comportado, pode apenas realizar mais algumas opções na interface, mentir acerca dos pontos que tem, caso já tenha ultrapassado 21 por exemplo, pode ainda perder mais uma carta mesmo depois de já ter dado stand, e pode declarar-se vencedor sem ter ganho o jogo.

Foram desenvolvidas algumas funções para tentar a descoberta o bad player quando se averigua o vencedor do jogo.

Observações

- O jogo está 100% funcional, atribuindo a vitoria da forma certa e sem problemas de comunicação, a atender vários players bem-comportados ao mesmo tempo, tendo sido experimentado no máximo 3 players como requisitado guião, mas sendo possível adicionar ainda mais players.
- No que toca ao segundo requisito do guião, o player malcomportado, bad_player.py, foi desenvolvido baseado no player.py, onde foram adicionadas algumas possibilidades de batota.
- No que toca ao último requisito, de todos os players serem capazes de detetar a batota, foram também desenvolvidos alguns mecanismos de detecçao, detetando a maioria dos casos em que um yer faz batota.
- Nestes ultimos 2 pontos, não foram totalmente desenvolvidos devido a falta de tempo, podendo adicionar mais alguns mecanismos, por exemplo o uso da Hash das cartas em jogo, mas que levam a uma boa funcionalidade geral de todos os scripts.