Distributed Black Jack

Projeto de Recurso
Computação Distribuída

Liliana Ribeiro, 108713, P4
Matilde Teixeira, 108193, P4

LifeCycle

1. Ligar player (**-s porta próprio -p portas dos restantes players**).
2. **Conectar** players entre si.
3. Receber **2 Cartas do deck (todos os players)** e guarda as **cartas na redis (por porta).**
4. Decidir **vez de jogar, pela menor porta.**
5. Player **joga**:
    a) (H)it -> pede **1 carta** e **publica carta na redis**
    b) (S)tand-> **passa** ao próximo jogador
    c) (W)in-> só pode se tiver **pontuação 21**
    d) (D)efeat-> só se tiver **mais que 21**
6. O **próximo jogador joga**.
7. Isto repete-se até que **só fique 1** em jogo (todos os **outros deram (D)efeat**) ou **alguém deu (W)in**) e só após isto é que se segue para o **próximo ponto**.
8. Se **múltiplos jogadores (+ de 2)**, **seleção do coordenador** (pela pior pontuação) e de **quem vai fazer o hashing** (mandar mensagem ao deck), pelas **2 piores pontuação** (não podendo ser o coordenador), porém no caso de **2 jogadores** os 2 iram efetuar o hashing, mas não haverá um coordenador e no **caso de 1** não irá ser feito o **hashing.**
9. Para verificar **se não houve batota**, os jogadores selecionados anteriormente irão executar o **hashing da lista de todas as cartas em jogo** (obtido através do redis) cujos **elementos irão ser permutados**, executando o **hashing destes até ser igual ao hashing dado pelo deck** enviando **VOTE_COMMIT,** caso isso **não aconteça** o player enviará um **VOTE-ABORT.**
10. Consoante os votos dos players(VOTE_COMMIT/VOTE_ABORT), o coordenador decide se **houve batota ou não (**se houver pelo **menos** um **VOTE_ABORT,** a **decisão final** será de **GLOBAL_ABORT)** e envia a sua **decisão final** aos outros players.

Protocolo

Tipos Mensagens	Comando	Utilidade
AfterPlay	`{"command": played, "player_port": self.player_port}`	Usado pelo player para comunicar que jogou
ReceivedCard	`{"command": receivedCard}`	Usado pelo player para comunicar que recebeu 2 cartas iniciais
PlayerDefeated	`{"command": defeated, "player_port": self.player_port}`	Usado pelo player para comunicar que deu (D)efeat
PlayerStand	`{"command": stand, "player_port": self.player_port}`	Usado pelo player para comunicar que deu (S)tand
PlayerWin	`{"command": win, "player_port": self.player_port}`	Usado pelo player para comunicar que deu (W)in
PlayerDecision	`{"command": decision, "decision": self.decision}`	Usado pelos players eleitos para o hashing para comunicar sua decisão ao coordenador
CoordenadorDecision	`{"command": coorDecision, "decision": self.decision}`	Usado pelo coordenador para comunicar decisão do final aos restantes players

Bad Player

Opções:

Obrigatório:

- Tirar uma carta a mais (sem a ir buscar ao deck)
- Mentir sobre ter ganho (accionar (W)in sem ter 21)
- Mentir sobre o valor das suas cartas (mentir sobre a pontuação ser 21)
- Mentir sobre o valor de hash obtido do deck (mandar VOTE_COMMIT com o resultado do hashing falso)

Opcional:

- Retirar carta do baralho (retirar carta do redis)
- Mentir que houve batota quando escolhido para hashing

Protocolo Commit Distribuído (2PC)

- Será escolhido um **coordenador** e **dois players** para executar o hashing.
- Para verificar **se não houve batota**, os jogadores selecionados anteriormente irão executar o **hashing da lista de todas as cartas em jogo** (obtido através do redis) cujos **elementos irão ser permutados**, executando o **hashing destes até ser igual ao hashing dado pelo deck** enviando **VOTE_COMMIT,** caso isso **não aconteça** o player enviará um **VOTE-ABORT.**
- Consoante os votos dos players(VOTE_COMMIT/VOTE_ABORT), o coordenador decide se **houve batota ou não (**se houver pelo **menos** um **VOTE_ABORT,** a **decisão final** será de **GLOBAL_ABORT)** e envia a sua **decisão final** aos outros players.

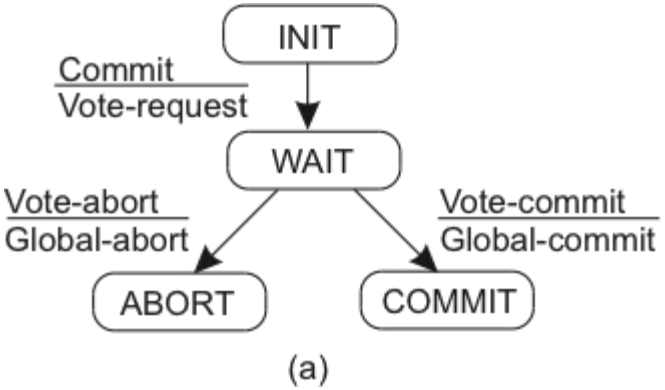

Fig. 1-Representação do protocolo de Commit Distribuído (2PC)