Relatório
Trabalho 3 de Redes I

GRR20197154 - Giordano Henrique Silveira
GRR20190427 - João Lucas Cordeiro

1 Desenvolvimento

Desenvolvemos uma versão com as regras decididas em sala do jogo Yahtzee em uma rede em anel com bastão usando a linguagem python. Dividimos o programa em 3 arquivos de código; e 4 terminais conseguem jogar na rede *local host*.

2 Organização dos Arquivos

O trabalho foi organizado em três arquivos: *jogador.py*, *lib_mensagem.py* e *Mensagem.py*.

2.1 Mensagem.py

Neste arquivo, temos apenas a definição do tipo mensagem (*msg_t*) com seus campos: marcador_de_inicio, combinacao, saldo, jogador, dados, bastao e paridade.

2.2 lib_mensagem.py

Nele, temos a implementação de funções que geram mensagens e a função que calcula a paridade de uma.

2.3 jogador.py

Aqui, além da implementação da main, temos o resto das funções necessárias pro programa: funções que enviam e recebem mensagens, teste de paridade, joga dados, alisa resultado, detecta erro, imprime o jogo na tela e devolve as portas que serão utilizadas na rede.

3 Estrutura do Tipo Mensagem

Temos aqui os seguintes campos da estrutura da mensagem e sua utilidade:

marcador_de_inicio	Campo responsável por carregar o valor “126” (01111110). Caso ele não esteja com esse valor, um erro é detectado.
combinacao	Em mensagens que carregam a combinação que está sendo apostada na rodada, este campo carrega o “*id*” da dita combinação. Exemplo: caso a combinação da rodada seja “trio”, este campo carrega “2” nas mensagens que usam a combinação.
saldo	Em mensagens que carregam algum valor de fichas, este campo guarda o valor que deseja ser comunicado. Exemplos: o valor apostado na rodada ou o saldo total de quem jogou no final da rodada.
jogador	Este campo carrega informação sobre qual jogador está com a maior aposta na rodada, ou seja, quem vai jogar/jogou ou quem terá o bastão.Caso este campo tenha valor “-1”, a mensagem é um erro, ou seja, uma mensagem que notifica um erro para as outras máquinas.
dados	Aqui, carregamos a informação sobre os valores que os dados do jogador rolaram na rodada.
bastao	Caso este campo na mensagem esteja **verdadeiro**, esta mensagem está passando o bastão, caso contrário, é outro tipo de mensagem.
paridade	Campo que carrega a paridade da mensagem atual. Ela é calculada em cima de toda a mensagem, com o campo da paridade zerado.

4 Fluxo da Main

4.1 Início

Começamos o programa perguntando qual é a máquina do usuário e ele deve responder com um número de 1 a 4. Dependendo da resposta, o programa escolherá qual será a porta usada para a entrada e qual será usada para a saída. Caso seja a máquina 1, ela começa com o bastão.

4.2 Começo do Laço

Depois, entramos em um laço infinito que pode seguir dois caminhos: o caminho do bastão e o caminho do sem bastão, mas explicaremos o fluxo do anel com os dois para ficar mais claro.
No bastão, o jogador é perguntado sobre qual combinação quer apostar. A aposta pra este jogador é sempre 1. O programa então gera uma mensagem com a informação do valor da aposta, da combinação e de quem apostou, a envia e espera a próxima mensagem.
Nos jogadores sem bastão, eles recebem uma mensagem. Se for uma mensagem de bastão, ele pega mais uma mensagem. Esse detalhe fica mais claro lá na frente.
Ele então está com a mensagem com o valor da aposta, combinação e jogador apostador. Com isso, ele é perguntado se quer aumentar a aposta pra disputar a jogada da rodada. Se sim, perguntamos em quanto ele quer aumentar, geramos uma mensagem parecida com a anterior com as informações novas, enviamos e esperamos uma nova. Caso não, ele só envia o que recebeu e espera uma nova mensagem.

4.3 Rolada de Dados

Com isso, a mensagem chegou no bastão de novo, completando um ciclo. Agora, acontece a mesma coisa no bastão e nos outros: recebe a mensagem, se a informação de jogador identificar ele mesmo, ele joga os dados, se não, ele repassa a mensagem.
Para jogar os dados, ele entra na função *joga_dados*. Nela, imprimimos pro jogador o que ele pode fazer e as roladas de dados. O jogador rola os dados até 3 vezes e pode travar os dados que quiser entre as rodadas. Caso ele decida não jogar, consideramos o resultado obtido. Caso jogue, ele pode travar os dados que quiser, de nenhum até todos eles. Então geramos a mensagem com as informações dos dados, do jogador e do seu saldo. Depois disso, eles esperam uma mensagem nova.

4.4 Informando os Jogadores do Resultado

O fluxo ainda fica parecido nos dois casos: uma mensagem com os dados, o jogador e o seu saldo chega. Repassamos a mensagem. Essas informações são então exibidas na tela e, caso o saldo seja menor ou igual à 0, entramos na função *termina_jogo* com argumento identificando o jogador que perdeu.
Com este argumento, essa função comunica que este jogador perdeu pois está com saldo nulo ou negativo e termina o jogo. Como a mensagem com as informações foram repassadas antes de fechar, caso o jogo tenha acabado, os jogos vão sendo fechados.

4.5 Terminando a Rodada

Voltando agora pro fluxo do bastão, ele recebe a mesma mensagem novamente, com as informações do resultado da rodada. Mas ele não faz nada com ela, pois agora chegou a hora de passar o bastão. Ele então gera a mensagem de bastão com quem deve recebê-lo e envia, e isso acaba o laço no fluxo do bastão.
Já no fluxo do jogador sem o bastão, ele recebe uma mensagem que será a mensagem do bastão. Caso o campo do jogador o identifique, ele pega o bastão e repassa a mensagem. Caso não, ele apenas repassa a mensagem. Isso então termina o fluxo do laço nos jogadores sem bastão.
Agora, o jogador com o bastão na última jogada vai receber uma mensagem com o bastão que ele mesmo enviou. Ela não serve pra nada, por isso, no começo do laço do sem bastão, lemos uma mensagem a mais caso receba uma mensagem que passa o bastão.
Assim, o laço se repete com o bastão no próximo jogador.

4.6 Sobre a Detecção de Erros

Bom ressaltar que na função que recebe as mensagens (*recebe_mensagem*), temos a detecção de erros e, caso encontre, envia uma mensagem informando o erro. Quando ela recebe uma mensagem do tipo, ela a repassa e fecha o jogo, chamando a função *termina_jogo* com argumento 0.

5 Detalhes de Implementação

Aqui nesse capítulo, a intenção é explicar alguns detalhezinhos usados na hora de implementar as soluções.

5.1 Envio e Recebimento de Mensagens

Para enviar mensagens, usamos uma função chamada “pickle.dumps” que pega a estrutura da mensagem e transforma em bytes. Pegamos então esses bytes e enviamos.
Na hora de receber, pegamos esses bytes e usamos a função “pickle.loads” para converter de volta para a estrutura que usamos no programa.

5.2 Entrada de Comandos

Na hora do jogador digitar a combinação, ele entra alguma das strings que representam as combinações (dupla, trio, d_dupla, full_house, seq_alta, seq_baixa, quadra e quinteto). Então, o programa usa um dicionário para converter essa string pro "*id*" da combinação, que será usado nas mensagens e nas funções que jogam dados, analisam resultados, entre outras.