Trabalho de Redes – Relatório

Adriano de Souza Gonçalves

Introdução

Este documento tem como finalidade relatar e apresentar de forma detalhada as informações importantes para o desenvolvimento e implementação do Trabalho de Redes – Jogo da Memória.

Normalização

O primeiro passo no desenvolvimento foi a definição das tecnologias a serem utilizadas. O material disponibilizado incluía um arquivo de código em Python 2, contendo as instruções do Jogo da Memória. No entanto, considerando que Python 2 está obsoleto e que Python 3 é amplamente adotado como padrão nas principais referências, optou-se por converter o código para a versão mais recente da linguagem antes de iniciar a elaboração do trabalho.

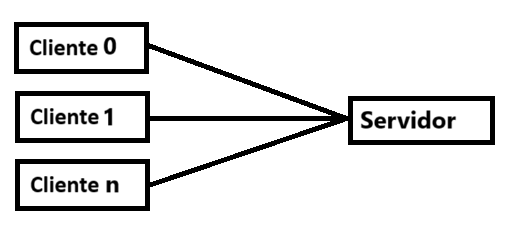
A conversão do código foi realizada com o auxílio da biblioteca externa "2to3", que automatiza o processo de migração de código Python 2 para Python 3. Mais informações, assim como a documentação completa da biblioteca estão disponíveis no link a seguir:

https://docs.python.org/3.11/library/2to3.html

Após a conversão do código para Python 3, deu-se início à implementação do trabalho, utilizando a versão mais recente da linguagem.

Relação Cliente-Servidor

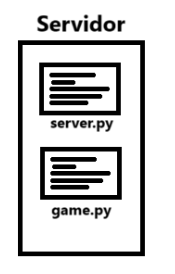
O sistema foi estruturado seguindo o padrão de arquitetura cliente-servidor, em que múltiplos clientes podem se conectar a um único servidor. O servidor, por sua vez, gerencia a comunicação simultânea com todos os clientes.



O cliente, que representa o jogador, consiste de apenas um arquivo: **client.py**



O servidor, é composto pelo programa **server.py** e pelo arquivo **game.py**, que consiste na implementação do jogo.



Dessa maneira, as regras do jogo e a lógica da aplicação são controladas exclusivamente pelo servidor. Os clientes não têm acesso direto ao código do jogo.

Handshake Inicial

Para iniciar a aplicação, é necessário primeiro inicializar o servidor. Isso pode ser feito executando o seguinte comando no terminal:

**python3 server.py**

Após a execução do comando, o servidor é iniciado e entra em estado de espera por conexões. Esse estado é gerenciado dentro da função **main()**.

Quando o servidor recebe uma solicitação de conexão de um jogador, ele a aceita e permanece em estado de espera até que o número de jogadores conectados atinja o valor definido pela variável MAX\_CONNECTIONS. O jogador, ao ter o pedido de conexão aceito, recebe uma mensagem de confirmação e a imprime no terminal.

O servidor imprime a relação de conexões estabelecidas no terminal.

Uma vez que a quantidade de jogadores requerida é atingida, o servidor inicia a partida. Se após o início da partida o servidor receber pedidos de conexão de outros jogadores, a conexão será recusada, e o jogador em questão receberá uma mensagem de recusa.

Para o executar o programa cliente, o seguinte comando deve ser executado no terminal:

**python3 client.py**

Ao executar o comando, o cliente enviará um pedido de conexão para o servidor. Se a conexão for aceita, será impressa a mensagem “Conexão Aceita!”. Caso o Servidor recuse a conexão, a mensagem “Conexão Recusada!” será impressa.

Início da Partida

No lado do cliente, a lógica da partida é gerenciada pela função **client\_play()**.

No servidor, para cada cliente conectado, é criada uma thread dedicada para representar a comunicação com aquele cliente específico. Dessa forma, cada thread no servidor controla a lógica do jogo com o cliente por meio da função **cliente\_thread()**.

A partir desse ponto, ambas as entidades executam uma ordem específica de instruções, representando o fluxo do jogo, com cada uma desempenhando suas tarefas designadas.

Lógica da Partida

Uma visualização de como o cliente e o servidor trabalham de forma síncrona para garantir a execução da aplicação pode ser ilustrado da seguinte forma:

Execução do Servidor Execução do Cliente

*#comentário sobre a ação do servidor #comentário sobre a ação do cliente*

**funcao\_sendo\_executada\_servidor() funcao\_sendo\_executada\_cliente()**

Em que as chamadas de funções na mesma linha indicam que essas funções são executadas pelo servidor e cliente simultaneamente.

Algumas funções auxiliares são chamadas dentro das funções principais, mas foram omitidas no esquema para facilitar a compreensão, assim como os argumentos de execução dos loops e condicionais.

Um resumo de como o cliente e servidor executam as funções referentes a partida:

Servidor Cliente

**client\_thread():** **client\_play():**

*#espera a partida ter o valor de jogadores suficientes. A partida inicia ao valor ser atingido.*

**wait\_for\_players()** **wait\_for\_start()**

*#gera e envia o número do jogador #recebe o número do jogador*

**send\_player\_number()** **receive\_player\_number()**

*#loop enquanto a partida acontece*

**while(**o jogo ainda não terminou**):** **while(**1**):**

*#verifica e envia se é a vez do jogador #recebe a informação se é a vez do jogador*

**send\_round()** **receive\_turn()**

*#caso seja a vez do jogador #caso seja a vez do jogador*

**if(**minha vez de jogar**):** **if(**minha vez de jogar**):**

*#recebe a coordenada da 1º peça* *#escolhe e envia a coord. da 1º peça*

**client\_choose\_card()** **client\_choose\_card()**

*#recebe a coordenada da 2º peça* *#escolhe e envia a coord. da 2º peça*

**client\_choose\_card()** **client\_choose\_card()**

*#envia o tabuleiro atualizado #recebe o tabuleiro atualizado*

**send\_table()** **receive\_table()**

*#verifica e envia se as peças são iguais*

**check\_equals()**

*#recebe informação se as peças são iguais*

**if(**peças são iguais**):**

*#incrementa a vez para o próximo jogador*  imprimir ponto

vez++

*#caso não seja a vez do jogador* *#caso não seja a vez do jogador*

**elif(**não é vez de jogar**)** **elif(**não é vez de jogar**)**

*#mostrar o tabuleiro* *#recebe o tabuleiro*

**send\_table()** **receive\_table()**

*#se o servidor sair do loop while, a partida acabou*

*#anuncia e envia o vencedor* *#caso o servidor anuncie o fim de jogo*

**send\_winner()** **elif(**fim de jogo**)**

break

*#mostra o vencedor*

**receive\_winner()**

*#desconecta o jogador**#encerra a conexão com o servidor*

**exit\_game() socket.close()**

Observações:

* A partida só é iniciada se a quantidade de jogadores definida for alcançada, caso contrário, os demais jogadores continuam em espera.
* O número do jogador, é definido pelo servidor. Os jogadores apenas os recebem.
* O servidor é o único que interage com o código do jogo.
* O servidor decide a vez dos jogadores. Os jogadores apenas recebem informações se a sua vez chegou ou não.
* Os jogadores enviam as coordenadas das cartas que foram viradas para o servidor. O servidor analisa o resultado, e retorna para o jogador se algum ponto foi marcado.
* Ao marcar um ponto, o servidor é que atualiza o placar.
* Caso a condição de saída do loop que indica o fim de jogo seja alcançada, o servidor envia um aviso de que o jogo acabou.
* O servidor verifica quem foi o vencedor, ou vencedores, e envia para os jogadores. Os jogadores recebem a informação e a imprimem na tela.

Assim, podemos garantir que toda a lógica, as regras de negócio e o controle de fluxo da aplicação são totalmente gerenciados pelo servidor. Os clientes apenas enviam e recebem informações, sem visibilidade sobre o funcionamento interno ou as regras da aplicação.

Final da Partida

A partida termina quando o número de pares encontrados é igual ao total de pares disponíveis, indicando que todos os pares foram identificados e o jogo chegou ao fim. Ao final da partida, o servidor envia uma mensagem a todos os jogadores.

Os jogadores, que estavam aguardando a mensagem de sua vez ou a de aguardar a vez, interpretam a mensagem de término do jogo, interrompem a execução e avançam para o bloco de código responsável por exibir o vencedor.

O servidor então analisa as pontuações dos jogadores e identifica quem foi o vencedor ou se houve um empate. Essa informação é enviada a cada jogador, que exibe na tela os resultados da partida.

Handshake Final

Após o término da partida e a exibição das informações dos vencedores, o servidor e os jogadores realizam o handshake de encerramento, garantindo que a conexão seja finalizada de forma segura. O servidor então registra no terminal as conexões que estão sendo encerradas e decrementa o número de conexões ativas.

A partir desse ponto, novos clientes podem se conectar para iniciar uma nova partida. O servidor opera em loop infinito, aceitando conexões continuamente, e só pode ser interrompido manualmente por meio de um comando no terminal.

Observações e melhorias:

Nesta seção, algumas observações e melhorias que foram implementadas no desenvolvimento do trabalho.

* O programa principal do jogo, que antes atuava como ponto de entrada do jogo, foi removido. O código foi refatorado para que ele não seja executável ao ser instanciado. Todas as funcionalidades agora são acessíveis ao servidor por meio de funções, fazendo com que o código funcione como um arquivo de módulos. As variáveis globais, responsáveis pela configuração dos parâmetros do jogo, foram movidas para o servidor.
* Uma situação de dessincronização pode ocorrer quando o servidor envia muitas mensagens e o cliente demora a processá-las, fazendo com que as mensagens se acumulem no buffer da conexão. Ao ler os dados do buffer, o cliente pode, equivocadamente, interpretar todo o conteúdo como uma única mensagem. Para resolver esse problema, adotou-se uma solução simples: definir um tamanho fixo para cada pacote transmitido entre os programas. Dessa forma, o cliente foi configurado para recuperar uma quantidade específica de bytes por vez, garantindo que as mensagens sejam lidas de forma individual e não se misturem.