Programação Distribuída

T1

Willian(16111123-2) , Gustavo Duarte (17204295-4), Marco Antonio(18103713-6)

O presente relatório tem como objetivo apresentar o desenvolvimento de uma aplicação distribuída utilizando Java RMI. Sua organização se encontra dividida em 3 seções, sendo elas: organização do código descrevendo como o código está organizado no projeto, utilização do programa mostrando passo a passo como utilizar o programa e demonstração da implementação mostrando screenshots junto com a explicação do programa em execução.

1) Organização do Código

Linguagem de programação/Organização

O trabalho foi desenvolvido utilizando a linguagem de programação Java junto com o RMI. Sua organização se encontra em Classes onde cada Classe tem sua responsabilidade no programa. Como uma visão macro temos o módulo Logic com todas as classes que lidam com a lógica, módulo Server que lida com o servidor, interface com as interfaces e Client que lida com o cliente.

Classes

A seguir descrevemos o que cada classe tem de responsabilidade no projeto:

- GameManager: Classe responsável por toda lógica relacionada ao jogo pelo lado do servidor. Nela temos todos os usuários do jogo, a quantidade de jogadores e a lógica relacionada a bonificação com probabilidade de 3% de ocorrência. A classe utiliza semáforos nas seções críticas.

- PlayerManager: Classe responsável por toda lógica relacionada ao Jogador pelo lado do Cliente. Nela temos os dados do usuário e qual servidor com o qual o jogador está se comunicando. Consta também a informação de se o jogador pode começar a jogar.

- User: Classe que serve para mapear o que é um usuário para o programa tendo seu IP e ID.

- HeartBeatPlayersTask: Classe responsável por verificar se os jogadores continuam vivos a cada 5 segundos. Caso o jogador não esteja mais vivo, ele é removido do servidor. Como o HeartBeat deve ser realizado a cada 5 segundos, é utilizada a classe TimeTask da própria linguagem, que executa uma thread a cada *x* unidades de tempo passadas.

- JogoInterface: Classe fornecida para o trabalho com as funções que o servidor deve implementar.

- JogadorInterface: Classe fornecida para o trabalho com as funções que o Cliente deve implementar.

- Server: Classe que implementa as funções da interface JogoInterface e usa a classe GameManager para utilizar a lógica do jogo no lado do servidor.

- Client: Classe que implementa as funções da interface JogadorInterface e usa a classe PlayerManager para utilizar a lógica do jogo no lado do Cliente.

- ClientRegisterToServerThread: Classe responsável por enviar para o servidor a solicitação de se registrar no jogo, assim como informações do jogador como seu IP. Esta classe é implementada utilizando Thread, tendo o objetivo de não bloquear a execução do programa no lado do Cliente.

- ServerSendToClientStartThread: Classe responsável por enviar aos jogadores a indicação de que eles podem iniciar o jogo. A classe utiliza Thread para não bloquear a execução do servidor ao enviar as informações para *N* jogadores.

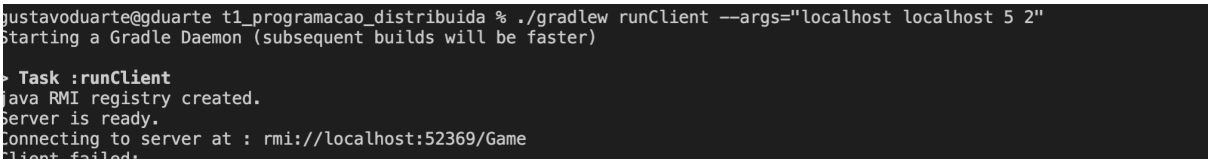
2) Utilização do Programa

Para utilizar o programa é necessário instalar o servidor inicialmente, a forma que o projeto foi construído permite que apenas 1 servidor possa ser instanciado numa mesma máquina. Múltiplas máquinas podem instanciar múltiplos servidores, mas uma mesma máquina não, caso isso seja feito somente o último servidor instanciado funcionará.

Para iniciar o servidor, se passa como parâmetro o IP da máquina que o mesmo irá utilizar e o número de jogadores que poderão jogar, caso execute o programa em diversas máquinas não pode passar o nome localhost tem que informar o IP. Ex: ./gradlew runServer --args=">IP do servidor> <número de jogadores>” Linux ou no Windows gradlew.bat runServer --args=">IP do servidor> <número de jogadores>”. Como a figura abaixo:

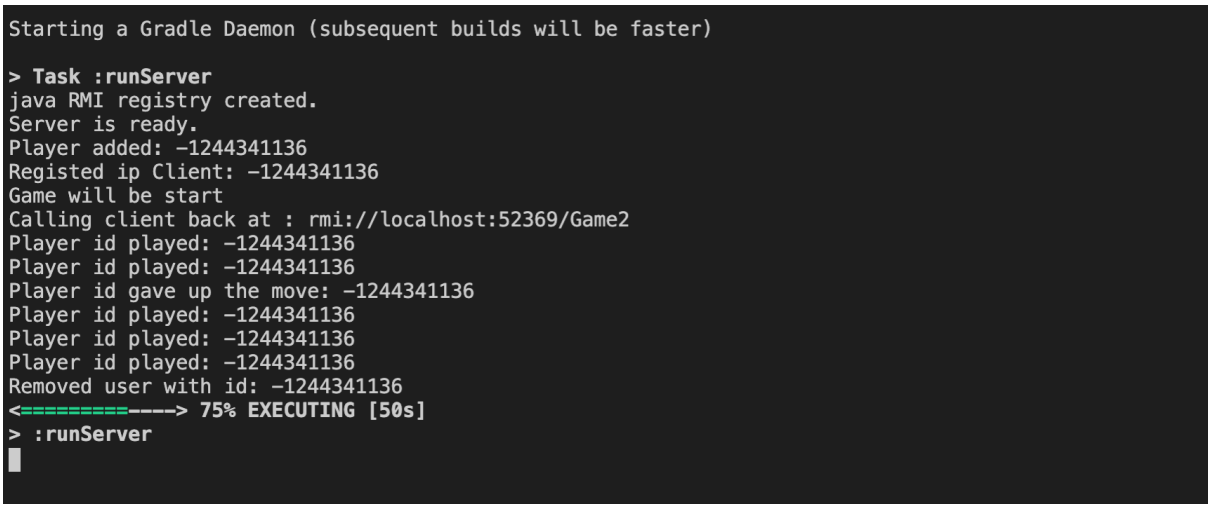
Para utilizar o cliente da aplicação, pode-se instanciar vários clientes numa mesma máquina, bastando apenas especificar qual porta utilizar para cada cliente, seguido do número de jogadas que se quer realizar e o número de desistências, caso execute o programa em diversas máquinas não pode passar o nome localhost tem que informar o IP assim como no server. As desistências serão realizadas em jogadas aleatórias, por

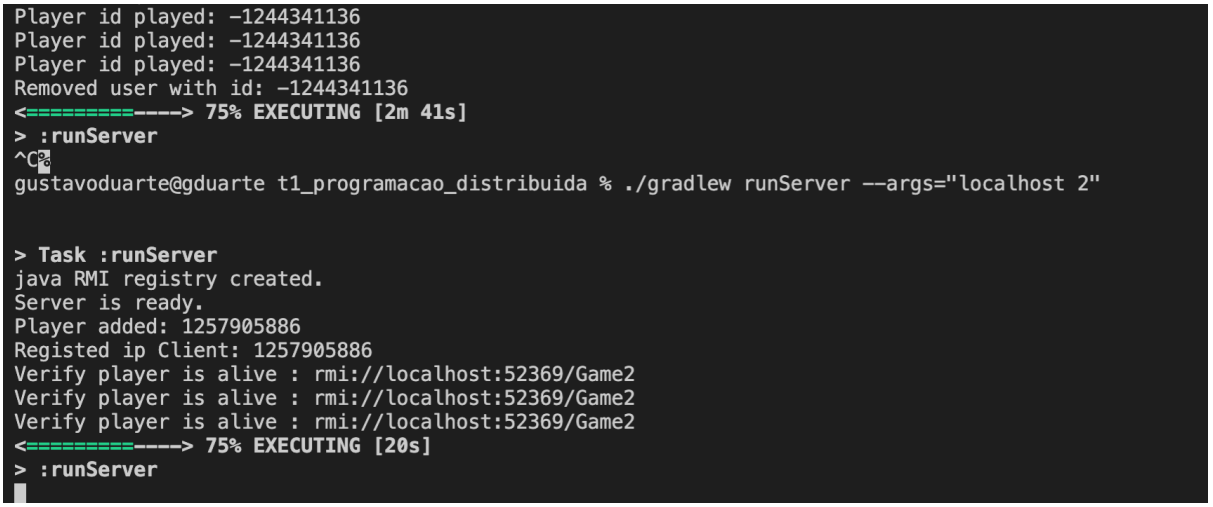
exemplo, caso tenha 5 jogadas e 2 desistências ao longo dessas 5 jogadas, 2 desistências irão ocorrer. Na figura, abaixo há o comando onde primeiro informa-se o IP do servidor, então o IP do cliente seguido opcionalmente da porta de rede (caso não seja informado será um valor default) e a quantidade de jogadas que se deseja realizar, bem como a quantidade de desistências :



3) Demonstração da implementação

Após iniciado o programa com os comandos mostrados na seção 2 o servidor exibe no seu terminal todas as informações seja do HeartBeat quando de solicitações e respostas aos clientes. Quando o jogo pode ser iniciado a mensagem “Game will be start” é exibida, depois cada jogada dos jogadores é exibida junto com seu ID e toda a solicitação do HeartBeat é exibida a mensagem “verify player is alive: <ip info do cliente>” e toda vez que um jogador é removido também é exibido no terminal conforme as figuras abaixo:





O cliente também exibe no seu terminal todas as informações a seu respeito e de quando realiza solicitações ao servidor. Quando recebe uma verificação do servidor se continua vivo o mesmo responde e exibe a mensagem “I’m a live” e quando realiza alguma jogada exibe a mensagem “You Played”, quando o cliente recebe um bonificação exibe “Gift from server… ” e ao começar o jogo o mesmo exibe a mensagem “Game Started” conforme a figura abaixo: