Programação Distribuida - Remote Gaming Server

> Autor: Ariel Rossetto Ril
> Matrícula: 15105050
> Email: ariel.ril@edu.pucrs.br

Introdução

Neste trabalho é apresentado uma implementação de um servidor remoto para um jogo onde os jogadores podem conectar no servidor (**register**), executar jogadas (**play**) e parar de jogar (**stop**). O servidor pode se comunicar com os jogadores para "iniciar" um jogador (**start**), verificar se um jogador está vivo (**liveness**) e bonificar um jogador após uma jogada (**bonus**).

O servidor desenvolvido neste trabalho utiliza o protocolo de comunicação **Java RMI** (*Remote Method Invocation*), o qual permite que um programa execute chamadas a métodos disponíveis em outros programas de forma remota. Para que exista uma comunicação nos dois sentidos (servidor ↔ jogador), foi desenvolvido um servidor para o jogo e um servidor para o jogador fazendo com que o servidor possa executar invocações remotas em métodos do jogador da mesma maneira que o jogador executa invocações remotas no jogo.

Desenvolvimento

Para realizar a utilização de **Java RMI**, todo desenvolvimento foi realizado utilizando a linguagem de programação Java.

Organização

Neste projeto foi densenvolvido 5 classes, 2 interfaces e um enum:

1. Interfaces
    1. IGame
    2. IPlayer
2. Classes
    1. Game → implementa IGame
    2. GameServer
    3. Player → implementa IPlayer
    4. PlayerServer
    5. Result
3. Enum
    1. ResultType

Interface IGame

Esta interface disponibiliza os métodos para serem utilizados durante a comunicação entre jogador e jogo.

```
public interface IGame extends Remote {
  public int register() throws RemoteException, ServerNotActiveException;
  public int play(int playerId) throws RemoteException;
  public int giveUp(int playerId) throws RemoteException;
  public int stop(int playerId) throws RemoteException;
  public void getResult(int playerId, int result, ResultType resultType) throws RemoteException;
}
```

Todos os métodos da interface devem retornar a exceção `RemoteExeception`, pois esta é uma exceção que pode originar de uma comunicação remota utilizando **RMI**.

Interface IPlayer

Esta interface disponibiliza os métodos para serem utilizados durante a comunicação entre o jogo e jogador.

```
public interface IPlayer extends Remote {
  public void start() throws RemoteException;
  public void bonus() throws RemoteException;
  public void check() throws RemoteException;
  public void getResult(int result, ResultType resultType) throws RemoteException;
}
```

Todos os métodos da interface devem retornar a exceção `RemoteExeception`, pois esta é uma exceção que pode originar de uma comunicação remota utilizando **RMI**.

Classe GameServer

Esta classe implementa a funcionalidade de servidor do jogo, o qual realiza a iniciação do jogo após os jogadores estarem registrados, realiza a solicitação de processamento de resultados dentro do jogo - solução de envio de resultado assíncrono para clientes - e solicitação para o jogo verificar se seus jogadores estão "vivos".

Nesta classe existe dois métodos principais:

- `public static void main(String[] args) throws RemoteException`
  - Inicializa a instância do servidor do jogo e o jogo
- `private static void run(Game game)`
  - Executa as funcionalidades do jogo, uma forma de separar a responsábilidade de executar as funcionalidades do jogo e "ser" o jogo

Classe Game

Esta classe implementa a interface `IGame` para que os métodos definidos na inteface possam ser invocados remotamente por algum jogador. Nesta classe foi implementado todas funcionalidade que um jogo possui:

- `public boolean isGameReady()`
  - verifica se já é possível iniciar o jogo
- `public boolean runningWithNoPlayers()`
  - verifica se ainda existe algum jogador vivo no jogo
- `public void start()`
  - inicia o jogo
- `public int register() throws RemoteException`
  - registra os jogadores
- `public void checkPlayers()`
  - verifica se os jogadores estão "vivos"
- `public int play(int playerId) throws RemoteException`
  - executa uma jogada de um jogador
- `public int giveUp(int playerId) throws RemoteException`
  - executa a desistência de um jogador
- `public int stop(int playerId) throws RemoteException`
  - executa a finalização do jogo para um jogador
- `public void processResults()`
  - processa a lista de resultados do jogo. Este método foi criado para que quando um jogador executa uma ação no jogo, o jogo possa enviar os resultados de maneira assíncrona para não "travar" o jogo
- `public void getResult(int playerId, int result, ResultType resultType) throws RemoteException`
  - recebe resultados de ações executadas nos jogadores

Classe PlayerServer

Esta classe foi desenvolvida para realizar o controle das ações que um jogador irá executar durante a sua execução. Nesta classe existe 3 métodos principais:

- `public static void main(String[] args) throws RemoteException`
  - inicializa e realiza o registro do servidor para o jogador
- `private static void run(Player player)`
  - executa as funcionalidades de um jogador: registro, execução de jogada, envio de resultados e finalização do processo
- `private static void registration(Player player)`
  - executa o registro de um jogador no jogo. Este método foi separado porque é executado um processo de *rebind* do local onde o jogador esta definido. Ao inicializar um jogador ele é registrado em uma porta padrão no *registry* e após o registro do jogador no jogo é realizado a modificação do registro do jogador no *registry* para facilitar a inicialiação dos processos e a resolução de nomes do jogador dentro do jogo.

Classe Player

Esta classe implementa a interface `IPlayer` para que os métodos definidos na interface possam ser invocados pelo jogo de forma remota. Os principais métodos definidos nesta classe são:

- `public void play()`
  - este método executa uma jogada no jogo
- `public void start() throws RemoteException`
  - este método é utilizado pelo jogo para informar ao jogador que o jogo foi iniciado
- `public void bonus() throws RemoteException`
  - este método é utilizado pelo jogo para informar ao jogador do recebimento de um bonus em um de suas jogadas
- `public void check() throws RemoteException`
  - este método é utilizado pelo jogo para solicitar ao jogador que informe ao jogo se esta vivo
- `public void getResult(int result, ResultType resultType) throws RemoteException`
  - este método é resposável por receber o resultado de alguma ação realizada no jogo
- `public void processResults()`
  - este método é responsável por enviar os resultados gerados de uma ação executada pelo jogo no jogador
- `public void stopPlaying()`
  - este método executa a finalização de jogo para um jogador, solicitando ao jogo para que o jogador seja removido da lista de jogadores

Classe Result

Esta classe foi desenvolvida apenas para servir como um envelope que contém o resultado de alguma ação executada no jogo ou no jogador.

```
public class Result {
  public ResultType type;
```

```
  public int value;
  public int playerId;

  public Result(int playerId, ResultType type, int value) {
    this.type = type;
    this.value = value;
    this.playerId = playerId;
  }
}
```

Demonstração

Compilação

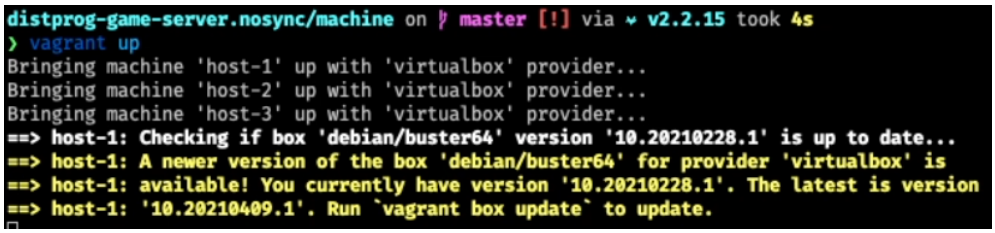
Para compilar (*build*) o código é necessário ir para a pasta `data/code`, a qual o contém o codigo Java da implementação. Após estar dentro da pasta que contém o código é necessário executar o comando `make` para compilar o código Java e gerar os arquivos `.class`.

Comandos

```
cd <project_root>/data/code
make # gera os arquivos .class
```

Máquinas Virtuais

Para inicias as máquinas virtuais para a demonstração é necessário ter a ferramenta **vagrant**(https://www.vagrantup.com) instalada além de possuir VirtualBox instalado. Após instalar as ferramentas necessárias é preciso estar na pasta `machine`, a qual contém o arquivo `Vagrantfile` com as configurações necessárias para inciar 3 máquinas virtuais, para executar o comando `vagrant up`.

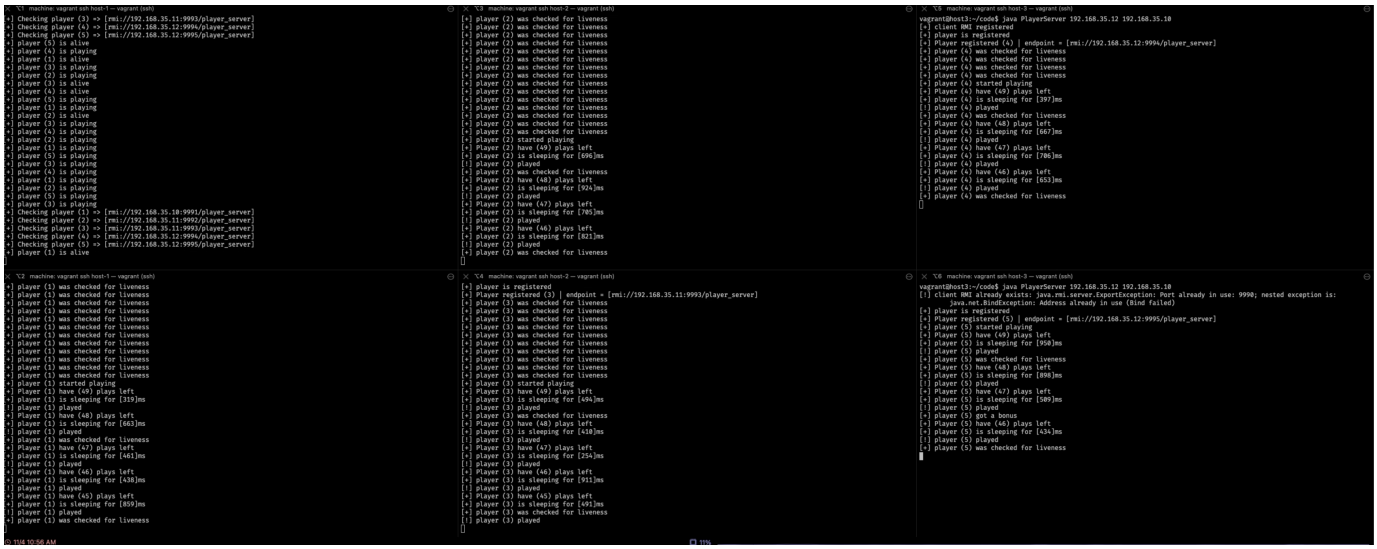


Simulação

Para executar a simulação é possível iniciar a quantidade que quiser de conexões com as máquinas virtuais, para este trabalho foi utilizado 3 máquinas virtuais e 6 conexões (duas para cada máquina virtual).

Na máquina 1 (**host-1**) foram executados o servidor do jogo (**GameServer**) e o jogador 1. Na máquina 2 (**host-2**) foram executados os jogadores 2 e 3. Na máquina 3 (**host**) foram executados os jogadores 4 e 5. Após conectar nas máquinas virtuais é preciso modificar o diretório para `~/code`, o qual é sincronizado com o diretório `data/code` da máquina local.

Após inicializar o servidor do jogo e incializar todos os jogadores, o programa ira inicializar o jogo e todos os jogadores irão começar a executar as suas jogadas.



Assim que todos jogadores finalizam suas jogadas e saem do jogo, o servidor do jogo finaliza o processo.