# Jartic - Trabalho 2 de Computação Distribuída - INE5418

Leonardo Schlüter
13200658

03 de maio, 2021

## 1 Ideia de Aplicação e Requisitos

A aplicação idealizada para esse projeto é algo semelhante ao jogo gartic. Os requisitão são então, numa versão mais básica, uma plataforma de desenho visível para todos os jogadores, sendo que apenas um jogador pode desenhar por rodada. Um chat verificador de respostas dos outros jogadores que estão apenas observando o desenho. Sobre o chat, vale ressaltar que a pontuação será baseado numa ordem FIFO, então as respostas dos jogadores serão infileiradas e pontuadas baseado num valor decrescente, começando em X e indo até 1, sendo X os números de jogadores menos o desenhista, o nodo coordenador do chat pode ser justamente o do jogador que está desenhando. Um coordenador (Máquina) que escolha palavras para serem desenhadas, escolha jogadores para desenharem e mantenha os pontos de cada jogador. A pontuação do desenhista vem da quantidade de jogadores que acertou seu desenho multiplicado por 2.

## 2 Arquitetura

Nessa seção descreverei como funcionará as trocas de mensagens entre os nodos para atender os requisitos definidos acima. A primeira definição é como funcionará a primeira rodada, que implica na criação do JChannel do chat e do desenho. No momento teremos um único channel para o jogo, mas conforme o trabalho ande, isso é passível de expansão, criando uma fase de seleção de channels/criação de channels ( consequentemente teremos em uma listagem de todos os channels abertos deste jogo ). Desta forma criando a impressão de um lobby de partidas em andamento, permitindo o usuário escolher qual quer entrar.

Na primeira vez que um jogador rodar a aplicação, ele será o nodo zero, assim como o líder. O coordenador do cluster fará papel de desenhista no ínicio do jogo. Assim que tivermos um líder no cluster, o comando */start* estará disponível para o mesmo. A palavra que o líder tem que desenhar só irá aparecer depois que ele entrar o comando de começo. Quando isso acontecer, além

de aparecer a palavra, um temporizador de 20 segundos será disparado para ele entender a palavra e como desenhará. Então, terá 90 segundos para realizar seu desenho. Nisso, o líder ficará escutando por respostas dos outros jogadores. As respostas serão infileiradas e tratadas pela ordem de chegada. Para serem tratadas paralelamente um identificador único ordenado será atribuído a cada requisição que chegar dos jogadores, além de uma informação que identifique o jogador que mandou a requisição. Numa versão mais avançada será guardado em uma memória compartilhada entre as threads as requisições que estiverem certas numa lista.Porém no momento teremos apenas uma única Thread tratando as respsotas e infileirando as mesmas na arquitetura de fila disponível pelo Java.

Depois que a rodada terminar, a lista de respostas certas será ordenada pelo seu identificador. Então, será atribuído pontuação aos jogadores, inclusive ao líder, seguindo as regras do jogo. Após isso, dar-se-á encerrada a primeira rodada. Então comunica-se a todos os nodos a pontuação de todos os jogadores e uma mensagem de histórico contendo um identificador para o jogador que acabou de jogar, dessa forma não sendo sortiado novamente até todos jogarem. Cada nodo atualiza sua pontuação baseado na mensagem do líder.

Agora temos que preparar as coisas para a próxima rodada. Para isso basta que o líder leia todos os jogadores disponíveis, removendo os que já foram ( no caso do fim da primeira rodada, é apenas o próprio líder), sortei um novo líder e então comunique a todos os jogadores do novo líder. Após isso, o novo líder terá o botão "start round" disponível, dessa forma podendo dar início ao processo a partir do clique em start round como já descrito. Caso o líder caia durante sua rodada, o nodo 0 é definido novamente como o novo líder e o jogo continua a partir do último estado. Aqui cabe uma verificação do estado dos jogadores que já jogaram, caso esse novo líder esteja nesse grupo, o atual líder deve executar o sorteio de lideres novamente, considerando os que já foram.

# 3 MVP da implementação

Sendo um grande fã de NBA, resolvi fazer o trocadilho ( Most Valuable Player - Most Valuable Part of the code ). Na implementação utilizei de base as Demos de Chat e Draw para ter uma base para o jogo. Mas muitos desafios surgem baseado nos requisitos inicias dados acima:

## 3.1 Desafio: deixar o demo Draw desenhável apenas por um usuário

O desafio é autoexplicativo, atualmente o Demo abre uma instância do panel para todos e permite que todos interajam com o mesmo. Para arrumar isso foi alterado o construtor do demo Draw para que receba um parâmetro chamado canDraw, cujo nome é autoexplicativo, se esse valor for verdadeiro, o usuário não pode interagir ( a não ser por sair do jogo ) com o painel de desenho. Esse

parâmetro é definido na construção do MyDraw pelo método isLeader, que diz se um dado nodo é líder.

```
private boolean isLeader() {
    if(leader == null){
        leader = 0 ;
    }
    Address leaderAddress = channel.getView().getMembers().get(leader);
    Address currentAddress = channel.getAddress();
    return leaderAddress.equals(currentAddress);
}
```

Vale alguns detalhes desse método, o primeiro é que se não há um líder, então por padrão o nodo 0 é o líder. Outro detalhe é que o isLeader poderia ser um boolean já definido na classe.

### 3.2 Desafio: Iniciar um round

Aqui é o momento em que o atual líder digita /start no chat. Agora devemos comunicar todos que um ROUND irá começar, então utilizei um JChannel.send com uma mensagem padrão e Thread.sleep para aguardar o tempo pré-definido de 20 segundos( aqui vale notar que esse tempo poderia ser configurável via comando de chat antes de iniciarmos uma partida, mas fica para uma eventual melhoria). Antes de usar o sleep é feito o sorteio da palavra que será o desafio do desenhista ( líder ). Atualmente o sorteio é de uma lista hardcoded de palavras. Tudo isso ficou no método startRound, arquivo Jartic.java. Alguns detalhes: este método só é executado pelo nodo líder; o timestamp do começo do round é tirado no nodo líder e comunicado a todos para ser utilizado futuramente.

### 3.3 Desafio: Finalizar o round

Há algumas coisas acontecendo aqui. O método responsável é o finishRound() no Jartic.java. Segue o método para explicitar a quantidade de coisa que faz:

```
private void finishRound() {
    // Calculate points
    int leaderPoints = calculateAndUpdatePlayerPoints();
    leaderPoints = leaderPoints * 2;

    score.updateScore(leader.toString(), leaderPoints);

    Address address = score.electNewLeader();

    // Share with others the score and new Leader
    chat.send(MSG_updateScore+score.toSerializable());
    chat.send(MSG_newLeader + address.toString());

    // Clear draw, wordToDraw and the answers submited in this round
```

```
        resetRound();
    }
```

Espero que o método seja verboso e comentado que chega para seu entendimento.

Da finalização do round surge um dois desafios, como comunicar e como eleger um novo líder.

### 3.4 Desafio: Elegendo um novo líder

Para resolver esse desafio tive que manter um histórico de quais jogadores já jogaram. Faço isso através da classe Player, mantendo uma propriedade hasBeenLeader atualizada. Essa propriedade é atualizada em algumas situaçaões, primeira é quando um novo líder é eleito, então o Player eleito tem o hasBeenLeader setado. A segunda é quando o primeiro jogador entra no jogo. A terceira é quando todos os jogadores do nosso Score tem essa propriedade como verdadeira antes da eleição de um novo lider. Esta ultima parte fica explícito no código a seguir ( localizado em Score.java ) :

```
    public Address electNewLeader() {
        Address newLeader = null;
        // Reset leadership of all players when everyone has already played
        resetLeadership();

        while (newLeader == null){
            Player proposedLeader = sortLeader();
            if(!proposedLeader.hasBeenLeader()){
                newLeader = proposedLeader.address;
            }
        }

        return newLeader;
    }
```

Então sortíamos da lista de jogadores um possível jogador, se ele já foi lider, descarta e tenta novamente até achar alguém que não foi líder. Como verifico antes de eleger se todos são líderes, esse algoritmo sempre irá parar, mas não é o mais elegante, muito menos o mais performático. Pra aumentar a perfomance poderíamos, por exemplo, manter uma bag atualizada dos players que não foram lider. Então basta pegar sempre alguém aleatoriamente da bag, até a bag esvaziar. Obviamente temos que checar o tamanho da bag antes de inicar o algorítmo de eleição.

### 3.5 Desafio: comunicando o Score

Esse desafio surgiu de dois problemas: sincronização de score entre os jogadores; Address do Jgroups não é serializável, por isso não podemos simplesmente compartilhar o objeto Score entre o channel. A forma que resolvi isso foi definir um toSerializable e updateScoreFromText no Score.java e toSerializible

no Player.java. Basicamente toSerializable retorna uma string mapeada com separadores pré-definidos do Score e todo o seu conteúdo. UpdateScoreFromText desfaz a operação que acabei de descrever utilizando split e os separadores definidos.

Poderia ter sido implementado de outra forma esse desafio, mas quando percebi esse problema, já estava com a implementação de Player usando Address do jgroups pronta. Uma outra forma seria ter feito um Score e Player que se utilizassem de propriedades que são serializáveis.

# 4 Execução

Foi criado uma pasta chamada 'executável' dentro do zip que enviei ao professor. Basta executar o .jar dentro desta pasta, os arquivos necessários se encontram dentro da pasta. Inclusive da pra brincar de adicionar palavras ao dicionário do jogo.

# 5 Conclusão

Tendo implementado esses desafios temos uma versão funcionando do Jartic. Mas muitos problemas de jogabilidade, interface e desempenho ainda existem. Porém consegui implementar uma versão simples funcional do jogo Gartic conforme descrito nos requisitos. Alguns problemas de interface são: o console fica printando mensagens de advertência que não consegui achar a fonte e não consegui remover, o chat e controle do jogo é todo via console, o painel de desenho começa minimizado, o chat fica aberto para digitar qualquer coisa mesmo quando não temos um round em andamento. O comando específico de começo de round é tratado de forma errada quando disparado por alguém que não é líder ( ele fica como se fosse uma resposta submetida ao desenho ). Alguns problemas de jogabilidade são: poucas palavras no jogo, apenas 1000 ( e são as 1000 mais comuns do inglês); não tem detecção automática do vencendor que alcançou uma pontuação pré definida, tem que ser combinado uma pontuação alvo entre os jogadores e depois usar o comando /reset para recomeçar o jogo. Alguns problemas de desempenho são: os inúmeros fors em cima dos membros da view; a constante recriação do objeto score e a comunicação do mesmo;

Em termos de comunicação em grupos temos alguns pontos que poderiam ser melhorados, no método receive tratamos as mensagens e comandos como String sempre. Porém, poderíamos definir um Enum que fosse serializável e comunicá-lo, definindo um método tratador pra cada comando. Bastaria enviar esse comando serializável via o JChannel.send, realizar o cast e chamar o tratador daquele comando. Isso já resolveria o if else gigantesco no método receive. Um outro problema são os aplicativos que usei de base, Chat e Draw, que contém código legado que não consegui arrancar a tempo da entrega do projeto. O código legado envolve troca de mensagens de forma desnecessaria.