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 da partida. 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).

Na primeira vez que um jogador rodar a aplicação, ele será o nodo zero, assim como o líder. O líder do cluster fará papel de desenhista no jogo. Assim que tivermos um líder no cluster, o botão start round estará disponível para o mesmo. A palavra que o líder tem que desenhar só irá aparecer depois que ele clicar em "start round". Quando o usuário clicar em "start round", 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. Será guardado em uma memória compartilhada entre as threads as requisições que estiverrem certas numa lista. 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 Chat.java:

```
private void startRound() throws Exception {
        Message msg = new ObjectMessage(null, MSG_warningStartRound);
        channel.send(msg);

        wordToDraw = sortNextWord();

        System.out.println(MSG_wordToDraw+wordToDraw);
        Thread.sleep(1000);

        draw.clearPanel();

        Date ts = new Date();
        msg = new ObjectMessage(null, MSG_roundStarted+ts.getTime());
        channel.send(msg);

        Thread.sleep(1000*90);
        finishRound();
    }
```

4 Referências

1. SiFive Interrupt cookbook - Disponível em `https://sifive.cdn.prismic.io/sifive/d1984d2b-c9b9-4c91-8de0-d68a5e64fa0f_sifive-interrupt-cookbook-v1p2.pdf`

2. An Introduction to RISC-V Architecture - Disponível em `https://cdn2.hubspot.net/hubfs/3020607/An%20Introduction%20to%20the%20RISC-V%20Architecture.pdf`

3. The RISC-V Instruction Set Manual Volume II: Privileged Architecture Privileged Architecture Version 1.10 - Disponível em `https://riscv.org//wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf`

4. SiFive FU540-C000 Manual - Disponível em `https://sifive.cdn.prismic.io/sifive%2F834354f0-08e6-423c-bf1f-0cb58ef14061_fu540-c000-v1.0.pdf`