Engenharia de Telecomunicações — Prof. Emerson Ribeiro de Mello

STD29006 – Sistemas Distribuídos — mello@ifsc.edu.br

Projeto prático Jogo de captura de bandeiras

21/11/2022

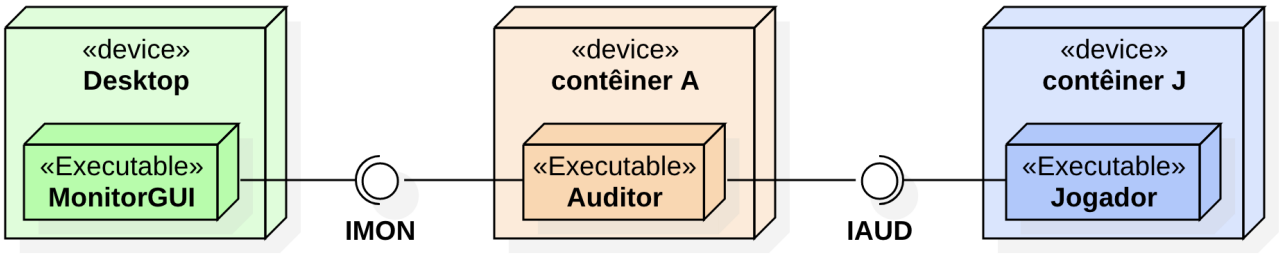
- Na raiz do repositório crie o arquivo `.gitignore` para ignorar arquivos desnecessários para esse projeto. O conteúdo desse arquivo poderá ser gerado por meio do site https://gitignore.io.
- Na raiz do repositório crie o arquivo `Readme.md` e coloque nele uma breve descrição sobre o projeto, as instruções para compilar e para executar as aplicações desse projeto. Deve-se também indicar, de forma clara, se todas funcionalidades foram implementadas e quais não foram.

Data para entrega: Até o dia 11/12/2022 via Github Classroom.

1 Descrição

A captura de bandeiras é um jogo que conta com um processo auditor e no mínimo dois jogadores. O auditor é responsável por acompanhar a movimentação dos jogadores pelo mapa, determinar qual jogador capturou quais bandeiras e, no final da partida, indicar qual jogador foi o vencedor. Jogadores e auditor são processos distribuídos executados em computadores distintos e usam a rede de computadores para se comunicarem.

Figura 1: Diagrama de implantação da solução



Na Figura 1 é apresentado um diagrama de implantação UML que mostra a distribuição dos processos **Auditor** e **Jogador** dentro de contêineres Docker. O processo **MonitorGUI** consiste de uma aplicação gráfica que só permite observar, em tempo real, a movimentação dos jogadores pelo mapa durante a partida e saber quem ganhou a partida. O processo **Auditor** provê duas interfaces de comunicação:

- `IAUD` – para interação com os processos **Jogador**;
- `IMON` – para interação com o processo **MonitorGUI**.

1.1 Dinâmica do jogo

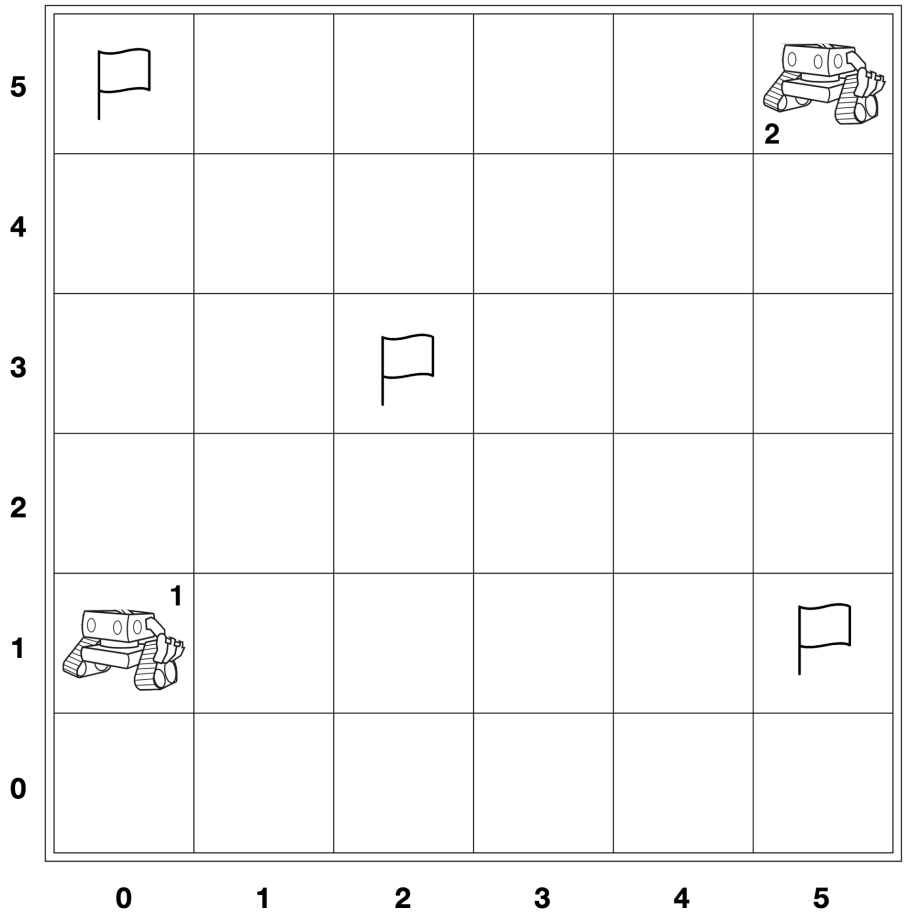
O processo **Auditor** deve estar ativo antes que os processos **Jogador** e **MonitorGUI** fiquem ativos. Os processos **Jogador** e **MonitorGUI** deverão conhecer previamente detalhes para interagirem com

o processo **Auditor**. Por exemplo, seu endereço IP (ou nome), porta, fila de mensagens, nome do objeto distribuído, etc.

O processo **Auditor** deverá iniciar a partida assim que ficar ciente que todos os jogadores, necessários para a partida, estão ativos. Não é necessário aguardar que o processo **MonitorGUI** fique ativo, ou seja, o processo **MonitorGUI** poderá se conectar ao processo **Auditor** em qualquer momento, seja antes ou durante a execução de uma partida.

No início da partida o **Auditor** envia para todos os processos **Jogador** o identificador único que atribuiu para cada jogador (p. ex. `jogador1`) e o mapa a ser explorado (indicando onde encontra-se cada bandeira e cada jogador). O mapa é formado por quadrantes, identificados por coordenada `(x,y)`, e cada jogador ou bandeira estará em um quadrante. Na Figura 2 é ilustrada uma instância do jogo com dois jogadores e três bandeiras. No exemplo tem-se um mapa com largura e altura igual a 6, sendo que a coordenada `(0,0)` fica no canto inferior esquerdo e a coordenada `(5,5)` fica no canto superior direito.

Figura 2: Instância de um mapa com 3 bandeiras e dois jogadores



Assim que o processo **Jogador** receber o mapa, este poderá executar sua estratégia para capturar as bandeiras. O deslocamento de cada Jogador sempre ocorre alterando uma unidade por vez no eixo x ou no eixo y. Por exemplo, se o jogador 1 iniciar a partida na coordenada $(x, y) = (0, 1)$ e quiser ir para a bandeira que está na coordenada `(2,3)`, então deveria se deslocar para `(0,2)`, depois `(0,3)`, depois `(1,3)` e por fim, `(2,3)`.

Assim que o **Jogador** finalizar um deslocamento (p. ex. de `(0,1)` para `(0,2)`), este deve enviar ao **Auditor** sua coordenada atual. Se na coordenada atual deste **Jogador** houver uma bandeira, então o **Auditor** deve enviar uma mensagem para todos os jogadores e informar que a bandeira na coordenada `(x,y)` fora capturada pelo Jogador X. Não há restrições para dois jogadores ocuparem uma mesma coordenada em um mesmo instante. Porém, um jogador só poderá enviar ao Auditor no máximo 1 mensagem por segundo, para indicar que fez um deslocamento. O Auditor deve responder

com uma mensagem erro se o jogador tentar enviar mais de uma mensagem por segundo.

2 Solução a ser desenvolvida

Desenvolva o auditor, jogador e monitorGUI de forma a permitir a comunicação explicitada na Subseção 1.1.

- **Auditor**
  - O **Auditor** não deve possuir uma interface que permita a interação com o usuário e deverá iniciar uma partida automaticamente assim que os todos jogadores se conectarem;
  - Ao iniciar o processo **Auditor** é necessário informar todos os detalhes da partida por meio de argumentos de linha de comando ou por meio de um arquivo de configuração (p.ex arquivo YAML ou JSON). As posições das bandeiras e dos jogadores no mapa deverão ser geradas de forma aleatória;
  - Os detalhes da partida são: tamanho do mapa (sempre uma área quadrada), número de jogadores e número de bandeiras;
  - Ao término da partida o o processo **Auditor** deverá enviar o resultado para todos os jogadores, indicando quantas bandeiras cada jogador capturou e quem foi o vencedor. Por fim, o processo auditor deve ser encerrado.
- **Jogador**
  - O **Jogador** não deve possuir interface para interação com o usuário, porém deverá imprimir em seu console toda a troca de mensagem que fizer com o auditor (envio e recebimento de mensagem). Por exemplo, quando receber o mapa, quando indicar ao coordenador que se deslocou para uma coordenada, quando receber a confirmação que uma bandeira foi capturada etc;
  - Para facilitar a visualização do deslocamento do jogador pelo mapa, cada deslocamento deverá demorar de 1 a 2 segundos. Sendo assim, o processo jogador, quando se deslocar para uma coordenada, deve dormir por um tempo aleatório, não menor que 1 segundo e não maior que 2 segundos, antes de enviar a mensagem ao Auditor informando sua nova coordenada.
- **Monitor**
  - O processo **MonitorGUI** deve ser uma aplicação gráfica que exiba um mapa em tempo real semelhante àquele apresentado na Figura 2. Para tal, pode-se basear no exemplo em Java apresentado em https://github.com/poo29004/demo-evento-mouse.

- A solução deverá obrigatoriamente ser construída sobre contêineres Docker, exceto o **MonitorGUI** que deverá ser executado fora de um contêiner. Sendo assim, no repositório é esperado um arquivo `docker-compose.yml` e, se necessário, os arquivos `Dockerfile` para cada processo (auditor e jogador);
  - Uso obrigatório do serviço de nomes presente no Docker para descoberta e resolução dos endereços IPs, além de fazer o mapeamento de portas para permitir que o processo **MonitorGUI** consiga conectar no auditor.
- A solução pode ser desenvolvida na linguagem Java;
- A solução pode ser desenvolvida com ZMQ, gRPC, RMI ou fila de mensagens (RabbitMQ). A tecnologia escolhida irá indicar como será a implementação das interfaces do Auditor (veja Figura 1), bem como a comunicação entre os processos;
  - Se fizer uso do RabbitMQ, então este deverá ser executado em um contêiner próprio.

Garanta que atenda os pontos abaixo

- ☑ Fez todos os laboratórios práticos apresentados nessa disciplina
- ☐ Repositório com arquivos Readme.md, .gitignore e docker-compose.yml, subdiretório específico para cada projeto
- ☐ Arquivo Readme.md contém explicações em como executar cada processo para permitir iniciar e observar uma partida
- ☐ Arquivo Readme.md contém uma lista dos requisitos atendidos e não atendidos.
- ☐ Caso o projeto tenha sido desenvolvido em dupla, no arquivo Readme.md deve-se apresentar a carga de trabalho de cada aluno no projeto (quem foi responsável por implementar o quê). Isso precisa estar refletido no histórico de *commits*
- ☐ Após clonar o repositório, é possível compilar e executar cada projeto de forma individual (auditor, jogador ou monitor) usando o gradle (pode ser com `gradle run` ou `gradle installDist`)
- ☐ É possível executar somente o contêiner com o processo auditor
- ☐ É possível executar `N` contêineres com o processo jogador e esses conseguem conectar corretamente no processo auditor
- ☐ É possível executar corretamente o monitor e esse consegue conectar corretamente no processo monitor
- ☐ Processo Auditor atendeu todos os requisitos
- ☐ Processo Jogador atendeu todos os requisitos
- ☐ Processo Monitor atendeu todos os requisitos

Este documento está licenciado sob Creative Commons “Atribuição 4.0 Internacional”.