Leonardo Rossi Vinagre - 202010277
Douglas Sturion - 202010264

Relatório Técnico: Implementação de Batalha Naval com Comunicação por Sockets

Introdução

Este relatório documenta o desenvolvimento de um jogo de batalha naval baseado em comunicação por sockets e entregando um jogo multijogador interativo.
O projeto leva como base o papel fundamental da comunicação de dados em jogos, permitindo a interatividade entre jogadores em tempo real através de redes. A proposta dele foi criar um jogo de batalha naval no qual os jogadores pudessem competir entre si junto com um modelo de comunicação entre clientes e servidor.

Objetivos Específicos

1. Implementar a comunicação cliente-servidor multithreaded:

   Desenvolvimento de clientes e servidor multithread utilizando sockets, capazes de lidar com múltiplas conexões simultâneas (Duas conexões para este caso).

2. Desenvolver um protocolo de comunicação eficiente:

   Desenvolver um protocolo de comunicação eficaz entre cliente e servidor, considerando o tratamento de mensagens textuais trocadas.

3. Mediador e Repositório de dados no servidor:

   Implementar o servidor como um mediador e repositório de dados compartilhados entre os clientes. Ele deve conhecer os dados e repassar aos clientes.

Descrição do Problema

A justificativa para abordar este problema é a importância de fornecer um sistema de multijogador eficiente, além disso o desenvolvimento de habilidades de comunicação em redes e a compreensão de protocolos e sistemas cliente-servidor utilizando sockets.

A comunicação entre cliente e servidor é o principal problema do projeto, o desenvolvimento de um jogo de batalha naval multijogador não é o maior desafio mas sim na criação dessa infraestrutura de forma eficiente.

O projeto enfrenta desafios técnicos que incluem além dos objetivos específicos:

- Garantia que o servidor funcione como mediador confiável e mantenha integridade dos dados do jogo
- Adaptação de códigos disponibilizados durante o curso para criar uma solução.

Modelagem da Solução

Comportamento das Entidades

Cliente

O cliente desempenha o papel de interação do jogador com o jogo de batalha naval. Principais comportamentos do cliente:

Conexão com o servidor

- **Estabelecimento de conexão:** O cliente estabelece uma conexão com o servidor para participar do jogo com uma thread separada.
- **Mensagens de entrada e saída:** O cliente possui uma thread separada para receber e enviar mensagens.

Interface do Usuário

- O cliente usará de interfaces gráficas Swing para configurar seus navios e interagir com o jogo através dos tabuleiros, cada cliente terá em sua tela o seu tabuleiro marcando onde o outro cliente já atacou e também terá o tabuleiro do adversário para que ele possa interagir e atacar, sem mostrar as posições dos barcos.

Envio e recebimento de mensagens

- **Formato das mensagens:** String, com delimitadores de informações.
- **Tratamento das mensagens:** Através de um separador programado para verificar a primeira parte da String separada por delimitadores que indica a ação a ser executada.

Servidor

Aguardando conexões

- **Espera por conexões:** O servidor quando iniciado fica a espera de conexões com um limite definido para dois jogadores e então encerra a possibilidade de mais conexões.
- **Identificação dos clientes:** O servidor cria uma thread diferente para cada cliente conectado e estabelece seu respectivo ID para controle geral das informações.

Coordenação do jogo e mensagens

- **Atualização do estado do jogo:** O servidor que já possui uma thread para cada cliente que seria sua conexão, também possui uma thread separada para lidar com o recebimento e envio de mensagens, isto junto com os ID's que foram estipulados ele realiza um sistema de turnos e consegue identificar de qual cliente foi a ação e realizar as alterações necessárias com consistência.
- **Resposta a ações dos clientes:** O servidor recebe a ação efetuada dos clientes em um thread, realiza as alterações necessárias e retorna aos clientes uma mensagem do tipo String que irá conter o tipo da alteração ou ação separado por um delimitador e em seguida a ação que foi realizada, para ser lida e efetuada na tela do cliente.

Repositório de dados

- **Repositório de dados:** O servidor guarda informação dos clientes e de seus tabuleiros para realizar as operações, e devolver aos clientes.
- **Segurança e integridade dos dados:** Tudo que foi dito, como threads separadas para cada cliente, threads para lidar com as mensagens de cada cliente, IDs únicos servem para fazer a distinção de cada um, assim o servidor consegue manter todos os dados separados com segurança e aplicar alterações individualmente garantindo a integridade.

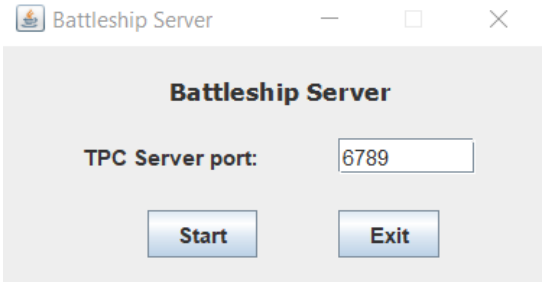
Layout de Telas

Apresentação de designs de telas(frames) e possíveis atividades

Servidor

Battleship Server

O servidor possui um único frame, que é o de criação do mesmo.

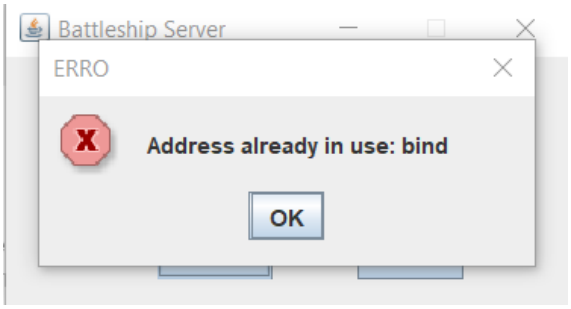


Ações

- Botão Start: Irá criar uma conexão socket com a porta definida no campo da tela, assim definindo o servidor e o deixando no modo de espera de conexões.
- Botão Exit: Irá encerrar o processo do aplicativo e fechar o servidor.

Erros

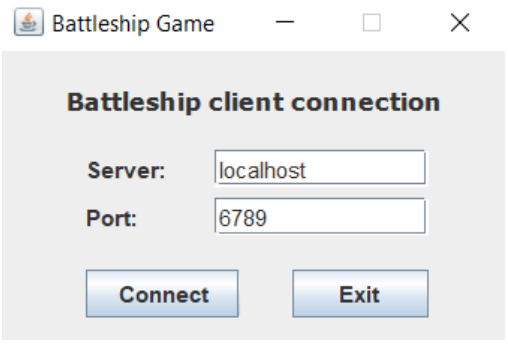
- Ao tentar criar uma nova conexão do servidor na mesma porta que foi marcada.



Cliente

Battleship client connection

Frame de início da conexão do cliente.



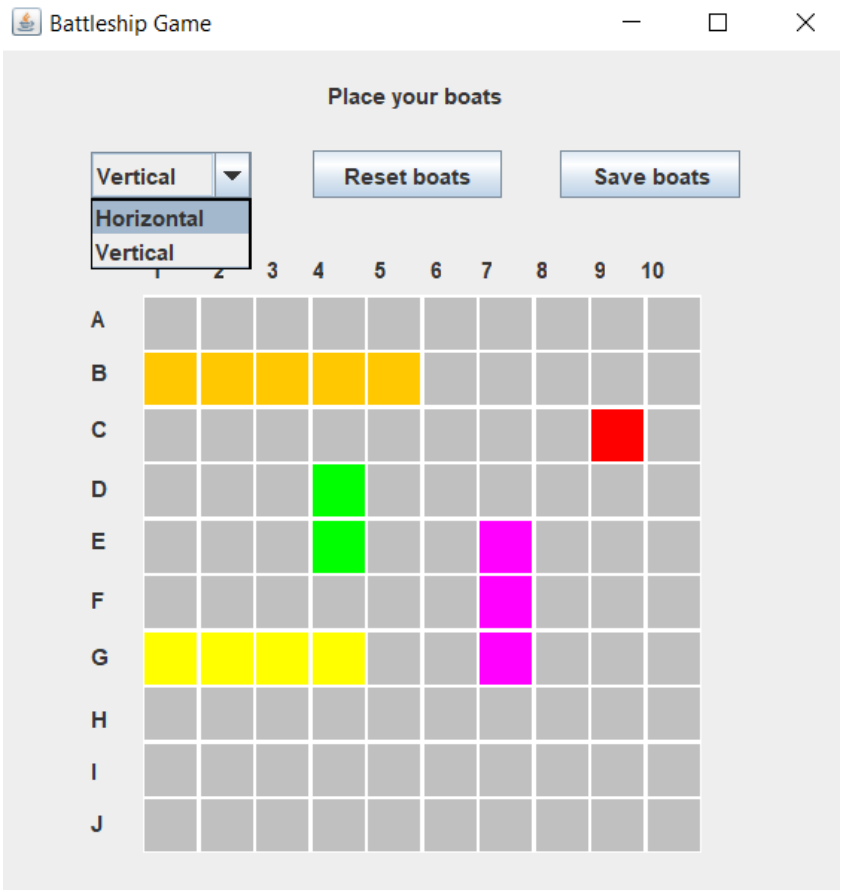
Ações

- Botão Connect: Irá criar uma conexão com o servidor através de um socket utilizando o servidor e porta definido na tela, assim se conectando com o servidor específico que deve ter sido criado previamente para que a conexão seja bem sucedida.
- Botão Exit: Irá encerrar o processo do cliente, sem conexões estabelecidas.

Com a conexão bem sucedida o cliente irá para a tela de definição do seu tabuleiro.

Battleship game: Place your boats

Tela em que o usuário irá definir as posições de seus barcos, foram definidos um total de 5 barcos para cada um sendo um de tamanho 5, um de 4, um de 3, um de 2 e um de 1.
O usuário deve clicar no grid nas posições que deseja inserir os barcos.

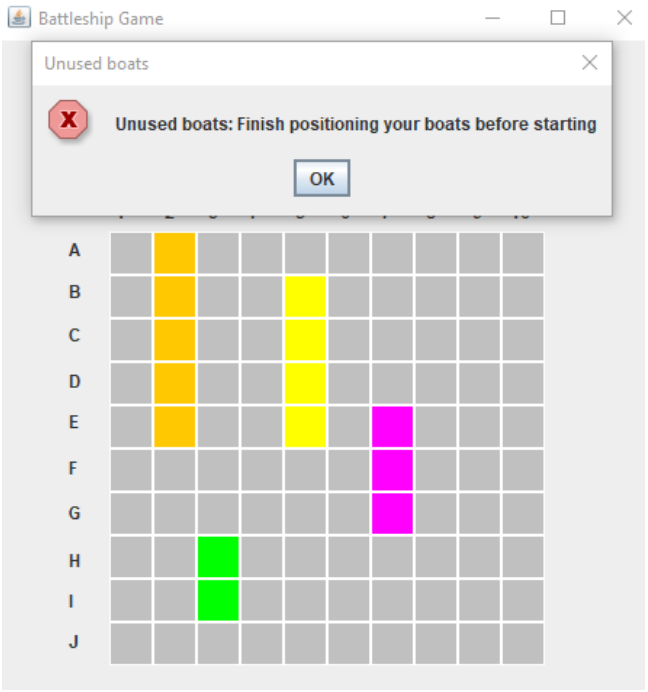


Ações

- Menu Drop: Seleciona a orientação em que irá inserir o barco, se selecionado na Horizontal será inserido na horizontal e se selecionado na Vertical será inserido na vertical.
- Botão Reset boats: Irá redefinir a tela e o grid ao início, retirando os barcos definidos anteriormentes.
- Botão Save boats: Irá enviar uma mensagem para o servidor de definição do tabuleiro, assim o servidor irá manter salvo as configurações do cliente.
- Botão de fechar a tela: Irá encerrar a conexão do cliente, enviando uma mensagem de desconexão para o servidor, esquecendo ele e liberando o ID dele para o próximo que entrar.

Erros

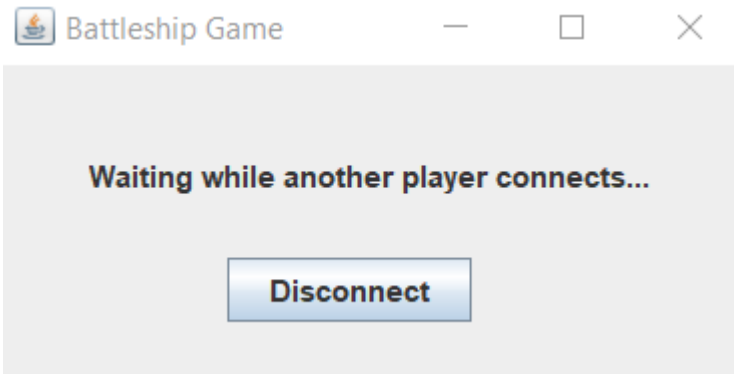
- Ao executar a ação Save boats sem ter definido todos seus 5 barcos.
  Por exemplo, não foi definido o barco de tamanho 1.



Ao executar a ação de Save boats corretamente, o cliente será enviado para uma tela de Lobby.

Battleship game: Lobby

Nesse frame o jogador fica no aguardo de outro jogador se conectar e ter definido seu board para poder iniciar o jogo.



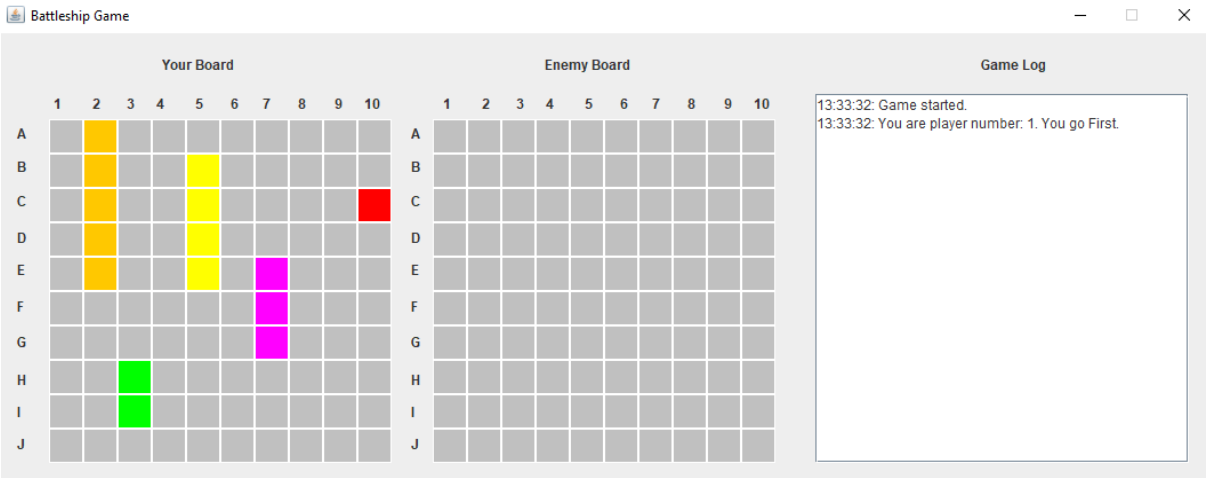
Assim que o servidor identifica que dois jogadores estão preparados para o início do jogo, ele envia uma mensagem de retirada do lobby e início do jogo.

Ações

- Botão Disconnect: Irá encerrar a conexão do cliente, enviando uma mensagem de desconexão para o servidor, esquecendo ele e liberando o ID dele para o próximo que entrar.

Battleship game: Frame principal do jogo

Nesse frame é onde o jogo se inicia, aqui onde os jogadores irão realizar suas ações de enviar bombas no tabuleiro do oponente.



- O tabuleiro da esquerda representa o tabuleiro do próprio usuário, terá acesso aos campos em que seu oponente atacou seu, podendo ter uma noção de como está o jogo dele.
- O tabuleiro da direita representa o de seu inimigo, as ações do usuário serão baseadas nele, será necessário clicar no bloco para realizar um envio de bomba.
- Game log: O servidor envia mensagens log para os clientes, relatando suas ações.

Cada jogador tem sua vez de jogar, definido pelo ID quem irá iniciar e mostrado no log.

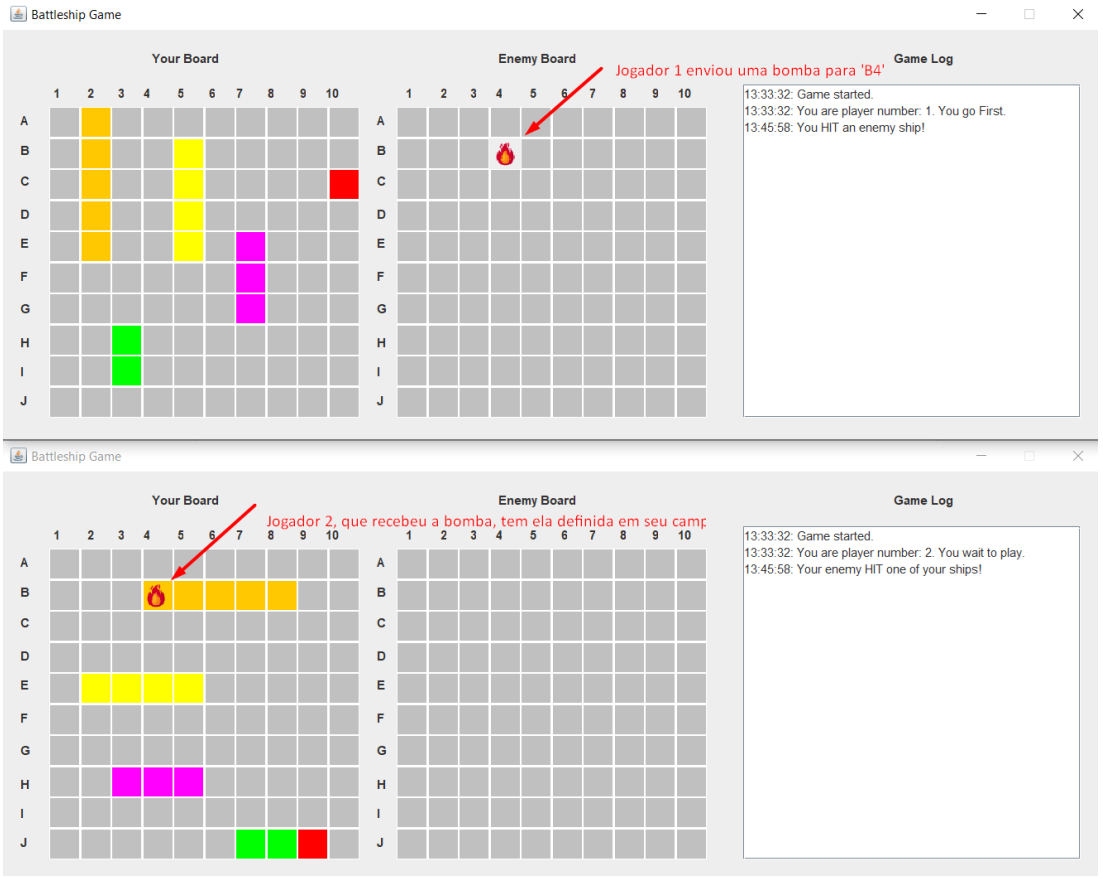
Ações

- Clicar em uma localização do Enemy Board: Se sua vez de jogar, deve clicar em uma localização para enviar uma bomba, o servidor irá receber essa mensagem, validar o campo, atualizar e devolver a resposta do ataque para os clientes.
- Botão de fechar a tela: Irá encerrar a conexão do cliente, enviando uma mensagem de desistência para o servidor, o servidor então retornará à vitória por desistência do outro cliente.

Respostas de ataque

HIT

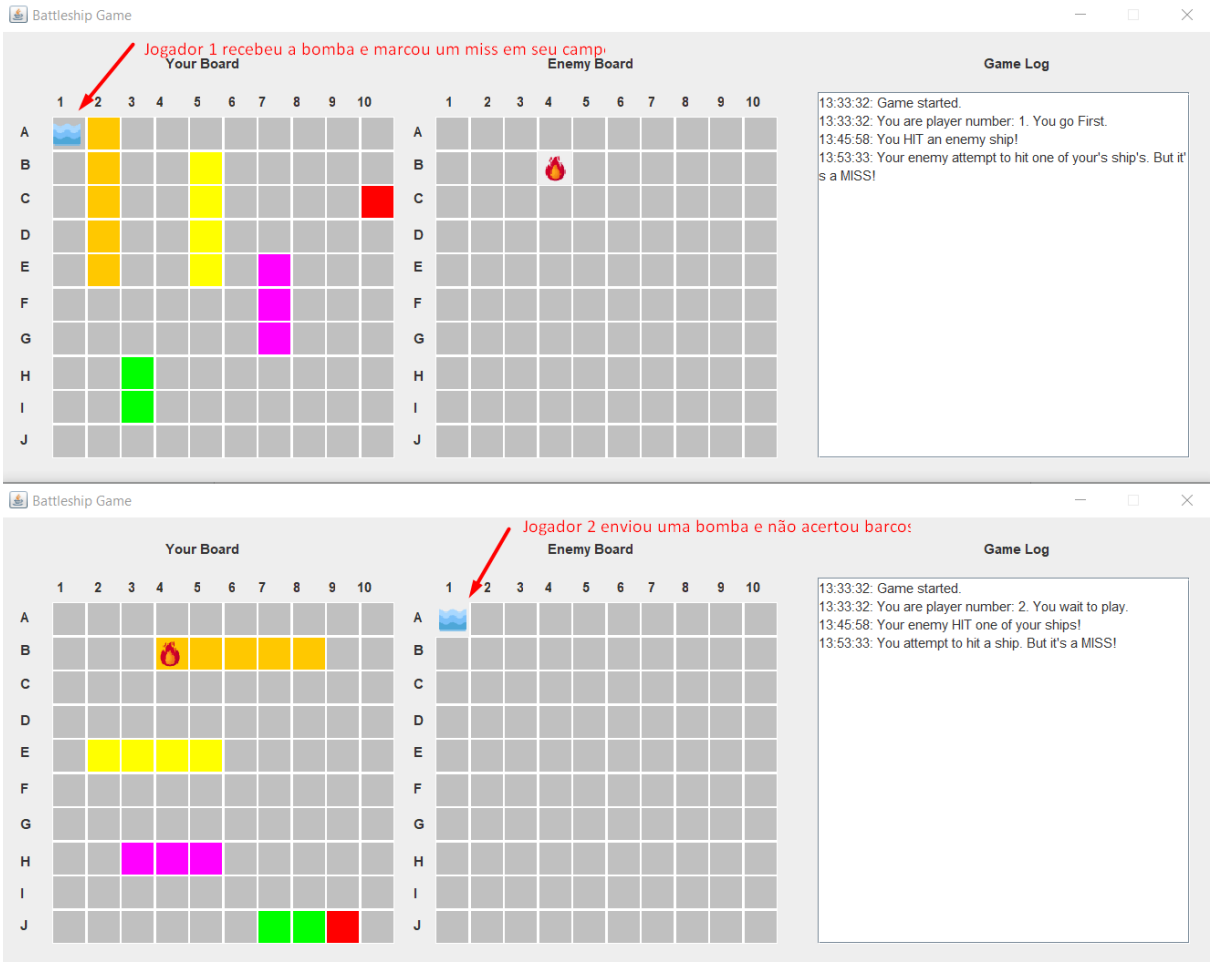
Um HIT define um ataque bem sucedido, em que o usuário acertou um barco inimigo.
Exemplo de resposta:



Definindo o HIT do usuário 1, ele mostrará no Enemy Board o ícone de HIT, e o usuário 2 mostrará no Your Board aonde ele foi acertado.

MISS

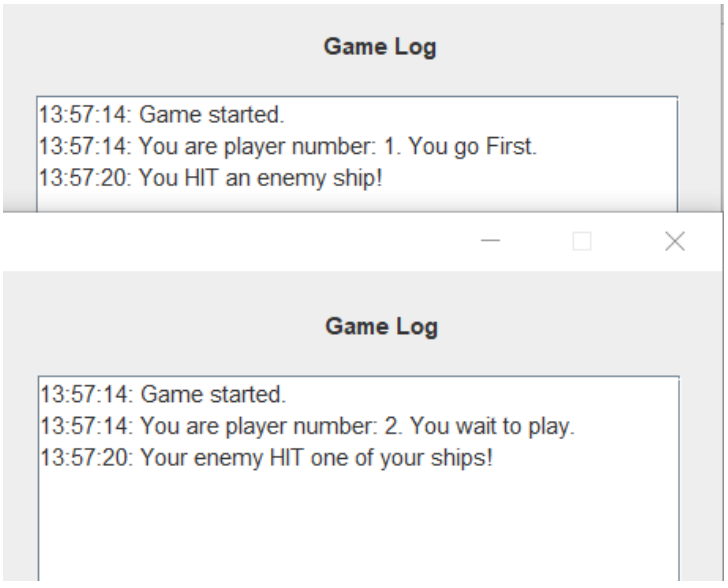
Um MISS define um ataque mal sucedido, em que o usuário não acertou um barco inimigo.
Exemplo de resposta:



Respostas de log do servidor

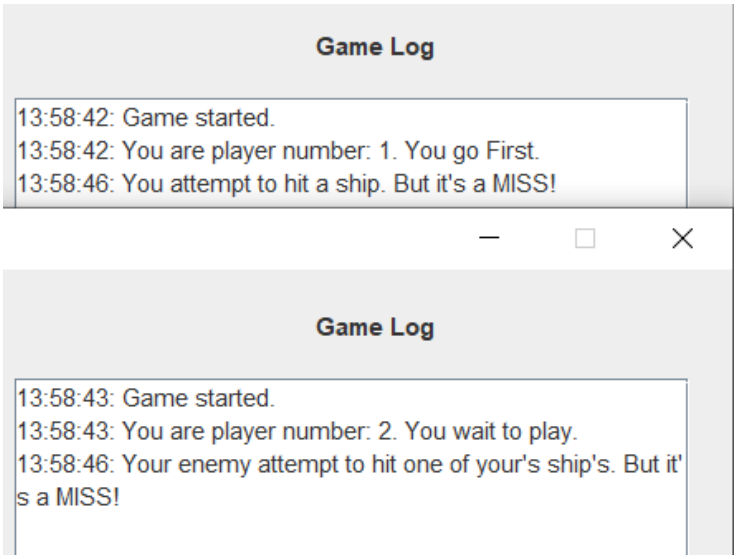
HIT

Exemplo de log de quando um HIT é enviado:



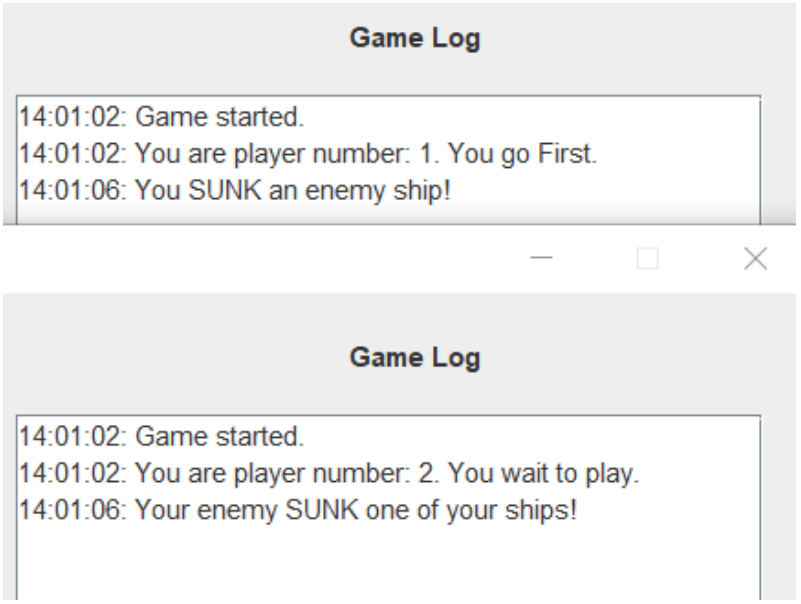
MISS

Exemplo de log de quando um MISS é enviado:



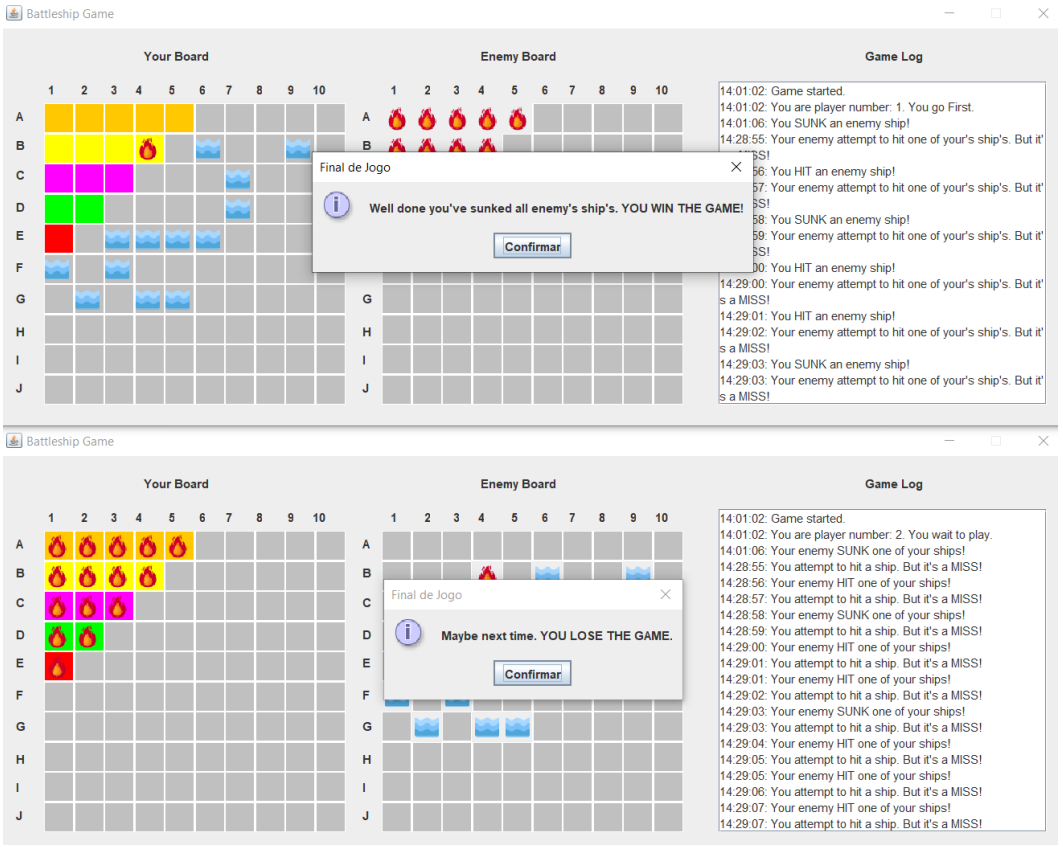
SUNK

SUNK é algo que o servidor irá projetar, ele recebe os HITS e quando perceber que um barco foi acertado em todas suas posições, retorna um SUNK.
Exemplo de log de quando um SUNK é projetado:



Critério de vitória

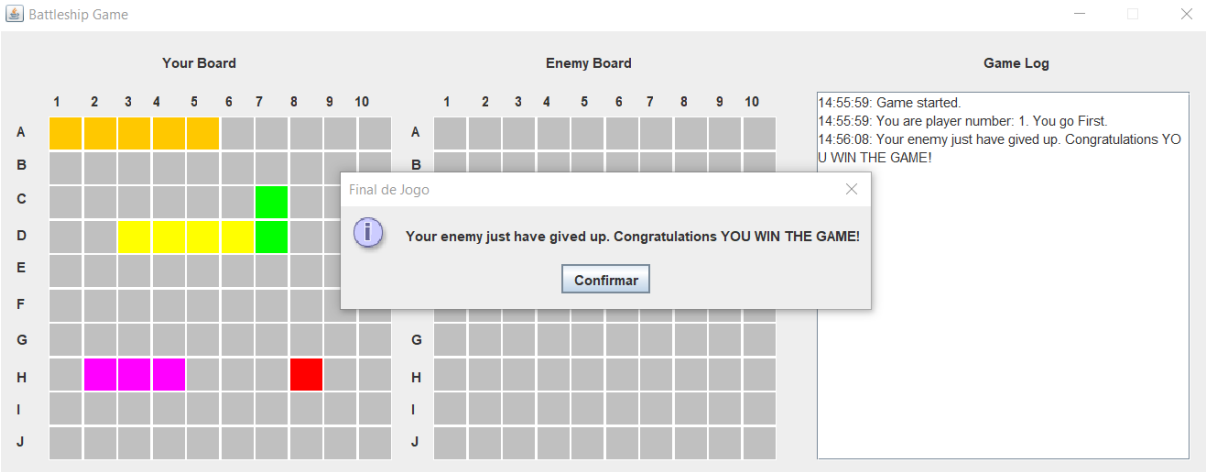
O critério de vitória do jogo é quando um jogador consegue executar um SUNK em todos os barcos inimigos.
Exemplo de resposta de vitória:



Para o jogo acabar, o servidor fica verificando o estado dos barcos e quando perceber o critério de vitória alcançado, ele irá enviar mensagens aos clientes e resetar todos os dados salvos de cada e mantendo o ID.
Ambos jogadores serão redirecionados para o frame de Battleship Game: Place your boats, para que possam configurar seus boards novamente e iniciar outro jogo.

Vitória por desistência

Se um dos jogadores no meio de um jogo clicar para sair do jogo, o outro irá receber uma vitória por desistência.
Exemplo de resposta de vitória por desistência:



Formato das Mensagens Trocadas

Modelo geral de mensagens trocadas entre cliente e servidor

Cliente e servidor usam o mesmo formato de mensagem. Quando um quer enviar uma mensagem ele irá utilizar um padrão criado no projeto de “Tokenização”, temos classes que definem tipos de mensagens a serem enviadas e temos um delimitador definido como “$”. A mensagem é montada da seguinte forma:

1. Ao ser selecionado uma ação, podendo ser do cliente ou do servidor, ele irá buscar nas classes que definem os comportamentos o tipo da mensagem a ser enviada.
2. Irá concatenar o tipo da mensagem com o delimitador que marcará o fim de uma parte específica da mensagem.
3. Em seguida irá concatenar com o que for necessário informar na parte do cliente ou do servidor.

Exemplo de formatação de envio de mensagem

Quando o cliente envia uma bomba para o servidor processar.

1. Será buscado nas classes que definem os comportamentos a mensagem de SEND_BOMB.
   Resultado da mensagem: “SEND_BOMB”
2. Adiciona o delimitador a mensagem indicando o fim do comportamento.
   Resultado da mensagem: “SEND_BOMB$”
3. Em seguida se adiciona a mensagem necessária a ser enviada, no caso de um SEND_BOMB ele irá adicionar a posição em que a bomba foi mandada, concatenando cada parte da mensagem pelo delimitador.

Por exemplo, enviada na parte da matriz (5,7).
Resultado final da mensagem: “SEND_BOMB$5$7”

Exemplo de recebimento de mensagem

Quando o cliente ou servidor recebe uma mensagem eles possuem a mesma lógica para decifrar ela. Seguindo o exemplo anterior mostra de envio de “SEND_BOMB”, será mostrado como o servidor recebe esta mensagem é lida com ela:

1. Ambos clientes e servidores possuem um único thread para o recebimento de mensagens. Ao recebe uma mensagem ele irá separar a mensagem em partes pelo delimitador, segue o exemplo:

   Mensagem recebida: “SEND_BOMB$5$7”
   Após a separação da mensagem por delimitadores:

   - Mensagem 0 = SEND_BOMB
   - Mensagem 1 = 5
   - Mensagem 2 = 7

   Sempre a mensagem 0 será o comportamento a ser executado.
2. A mensagem 0 passará por um switch case procurando o tipo do comportamento enviado, e passará as outras mensagens separadas para frente.
3. Será executado as ações necessárias para realizar a operação recebida.

*Nota-se que as etapas são um padrão, cliente e servidor irão sempre ter o comportamento no início da mensagem, assim sempre que alguém receber algo ele irá redirecionar para a função apropriada.

Classes definidoras de ações e comunicação:

Classe de ação

A classe de mensagens de ações possui definidos os tipos de ações que podem ser enviadas pelo comportamento do cliente e da resposta a esses comportamentos do servidor.

```java
public class ActionMessages {
    public static final String YOUR_MISS = "You attempt to hit a ship. But it's a MISS!";
    public static final String ENEMY_MISS = "Your enemy attempt to hit one of your's ship's. But it's a MISS!";
    public static final String YOUR_HIT = "You HIT an enemy ship!";
    public static final String ENEMY_HIT = "Your enemy HIT one of your ships!";
    public static final String YOUR_SUNK = "You SUNK an enemy ship!";
    public static final String ENEMY_SUNK = "Your enemy SUNK one of your ships!";
    public static final String YOU_WIN = "Well done you've sunked all enemy's ship's. YOU WIN THE GAME!";
    public static final String YOU_LOSE = "Maybe next time. YOU LOSE THE GAME.";
    public static final String WO_WIN_MESSAGE = "Your enemy just have gived up. Congratulations YOU WIN THE GAME!";
}
```

Classe de Comunicação

A classe de mensagens de comunicação definem os estados do jogo que o servidor usa.

```java
public class ComunicationMessages {
    public static final String SEND_BOATS = "SEND_BOATS";
    public static final String INIT_GAME = "INIT_GAME";
    public static final String START_TURN = "START_TURN";
    public static final String STOP_TURN = "STOP_TURN";
    public static final String CLOSE_CONNECTION = "CLOSE_CONNECTION";
    public static final String CHANGE_ID = "CHANGE_ID";
    public static final String SEND_BOMB = "SEND_BOMB";
    public static final String SEND_MISS = "SEND_MISS";
    public static final String SEND_HIT = "SEND_HIT";
    public static final String SEND_SUNK = "SEND_SUNK";
    public static final String SEND_WINNER = "SEND_WINNER";
    public static final String GIVE_UP = "GIVE_UP";
    public static final String WO_WIN = "WO_WIN";
}
```

Requisitos

- JVM: Necessário instalação do JVM execução do projeto.
- IDE: Utilização de uma IDE para abrir o projeto java.
- Versão utilizada no desenvolvimento: Java 18

Procedimentos para Instalação

- Instale a JVM
- Instale uma IDE, recomendado NetBeans.
- Baixe o projeto
- Descompacte (se aplicável)
- Abra o projeto na sua IDE

Executando o servidor

Após aberto o projeto em sua IDE, vá para a pasta frames, e execute o arquivo: ServerFrame.java

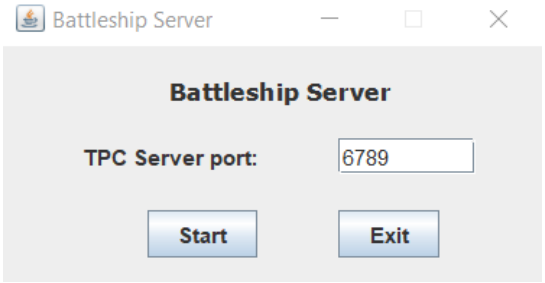
Executando cliente

Após aberto o projeto em sua IDE, vá para a pasta frames, e execute o arquivo: ClientFrame.java

Procedimentos de Teste

Servidor

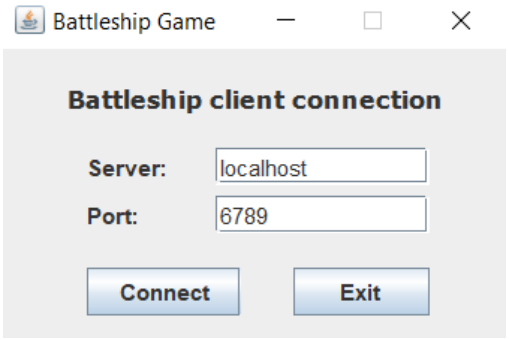
Na tela do servidor:
Basta inserir em TCP Server port o seu endereço do servidor em que deseja subir ele.



Padrão:
TCP Server port: localhost

Cliente

Na tela do cliente:



Basta inserir em Server o seu endereço do servidor e em Port o endereço da porta, que deseja realizar a conexão.

Padrão:
Server: localhost
Port: 6789

Com o cliente conectado, você pode testar todo o projeto seguindo as definições feitas em Layout de telas.