Disciplina de Sistemas Distribuídos I 2024-2

Prof. Rafael Oliveira Vasconcelos

Jogo Online Web do Projeto de Sistemas Distribuídos - *SpaceWar*

Documento de Visão e Requisitos

Versão 0.1

Rafael Miranda Oliveira Nascimento

William Santos Silva

Matheus Victor Fontes Santos

Caio Victor Prado Cruz

1. Introdução

O presente documento descreve o projeto de um jogo online do tipo I.O ambientado no espaço, enfatizando uma arquitetura distribuída, comunicação entre dispositivos e sincronização de estados. O projeto tem como objetivo proporcionar aprendizado prático sobre sistemas distribuídos, abordando aspectos como consistência, tolerância a falhas e escalabilidade

1.1. Objetivo do Jogo

O jogo consiste em partidas nas quais cada jogador controla uma nave, com coordenadas (x,y), para desviar de obstáculos, e dispara projéteis com o objetivo de eliminar o(s) outro(s) jogador(es) o maior número de vezes em um determinado limite de tempo; vence o jogador que eliminou o(s) outro(s) mais vezes até o fim da partida.

1.2. Jogabilidade

1.3. Funcionamento Geral do Jogo

- Jogadores assumem o controle de espaçonaves e devem competir para obter a maior pontuação.
- O jogo ocorre em um espaço aberto com meteoros, power-ups e inimigos controlados por simulação de IA ou IA.
- Os jogadores podem atacar, desviar e coletar recursos para evoluir suas espaçonaves.
- O jogo é baseado em sessões rápidas, com respawn imediato após a eliminação.
- O jogador também tem a opção de mudar o tipo de projétil da nave, após o upgrade.
- O jogo vai disponibilizar diferentes classes de espaçonaves, cada uma com habilidades especiais.
- Eventos dinâmicos no mapa, como tempestades espaciais e zonas de gravidade alterada.
- O jogo vai ter limites de mapa.
- O jogo vai disponibilizar vários mapas diferentes com eventos diferentes

2. Concerns

2.1. Consistência de Dados

Para garantir a consistência dos dados no jogo, adotaremos o modelo de **consistência eventual**, que é essencial para nosso jogo online. Esse modelo permite que as atualizações sejam propagadas com um pequeno atraso, garantindo um equilíbrio entre desempenho e sincronização. Os benefícios são:

- **Baixa latência** – O jogo continua fluindo sem a necessidade de aguardar confirmações em tempo real.

- **Alto desempenho** – O servidor processa as ações rapidamente e distribui as atualizações de forma eficiente.
- **Escalabilidade** – Suporta milhares de jogadores simultâneos sem comprometer a jogabilidade.

Para garantir o uso desse modelo, utilizaremos a **sincronização inteligente no cliente.** Esse método permite que os clientes **façam previsões** e ajustam os dados quando recebem atualizações do servidor. Evitando travamentos durante a partida.

2.2. Tolerância a Falhas

Para garantir a **tolerância a falhas**, o sistema será projetado para preservar o progresso dos jogadores e minimizar interrupções durante a partida.

O sistema será implementado com backup de estado do jogo, onde cada jogador tem um **backup contínuo do estado,** se caso o servidor cair o jogador pode **conectar sem perder progresso**. A fim de implementar essa funcionalidade, salvaremos os snapshots periódicos no banco de dados PostgreSQL, além de reenviar estados para novos jogadores ao reconectar.

Com a utilização do **WebSocket**, o sistema poderá detectar desconexões inesperadas e **reconectar automaticamente os jogadores**. Isso evita que pequenos problemas de rede causem desconexões permanentes, proporcionando uma experiência mais estável e contínua.

2.3. Escalabilidade

Para garantir que o jogo suporte milhares de jogadores simultâneos sem comprometer o desempenho, adotaremos uma arquitetura escalável e eficiente. O uso de **WebSockets** permitirá uma comunicação em tempo real com **baixa latência** e **menor consumo de recursos**, tornando a experiência do jogador mais fluida.

Além disso, implementaremos o **balanceamento de carga** por meio de **serviços de nuvem**, como **AWS ALB ou NGINX**, permitindo distribuir conexões entre múltiplos servidores. Essa abordagem possibilita a **escalabilidade horizontal**, adicionando novos servidores conforme a demanda.

2.4. Segurança

Utilizaremos o uso de WebSockets (WS) permitindo uma comunicação bidirecional em tempo real, essencial para jogos multiplayer. Porém, a fim de garantir a segurança da comunicação é essencial o uso do **WebSockets Seguros (WSS)**.

- **Protegida contra ataques Man-in-the-Middle (MitM)**, evitando interceptação de mensagens.
- **Resistente a Spoofing e Replay Attacks**, garantindo que mensagens são legítimas.
- **Blindada contra Engenharia Reversa**, dificultando a criação de cheats e exploits.

Além disso, utilizaremos no backend, o **JSON Web Token (JWT)** é um método seguro e eficiente para autenticação e troca de informações entre clientes e servidores. A fim de **evitar interceptação de credenciais e dados sensíveis**.

3. Decisões Arquiteturais

3.1. Cliente-Servidor

A arquitetura cliente-servidor foi escolhida para o “SpaceWar” com o objetivo de centralizar a lógica do jogo e garantir a integridade e consistência dos dados, mesmo em um ambiente com múltiplos jogadores simultâneos. Nesta abordagem, o servidor é responsável por processar e validar todas as ações dos jogadores , atualizar o estado do jogo e distribuir as informações necessárias aos clientes. Já os clientes ficam responsáveis por renderizar a interface e enviar os dados de suas ações. Tudo será estruturado por meio de micro serviços para promover modularidade, escalabilidade e facilidade de manutenção.

Micro Serviço de Escalabilidade e Monitoramento:

Gerencia o balanceamento de carga (por meio de AWS ALB ou NGINX) e monitora a performance, permitindo a escalabilidade horizontal para suportar milhares de jogadores simultâneos.

Micro Serviço de Gerenciamento do Jogo:

Processa a lógica central do jogo, incluindo o controle das naves, disparos e sincronização do estado global. Ao centralizar essas funções, é possível validar as ações dos jogadores e garantir a integridade da partida.

Micro Serviço de Backup e Sincronização:

Realiza o salvamento periódico de snapshots do estado do jogo no PostgreSQL para possibilitar a recuperação rápida em caso de falhas, preservando o progresso dos jogadores.

3.2. Linguagens e Ferramentas

No Backend:

- Linguagem Java 17+
- Framework SpringBoot 3.3.1
- Lint - Checkstyle 10.17.0
- Ferramenta de documentação - Swagger, versão 5.17.14

No Frontend:

- Linguagem JavaScript, versão 5.5.2
- Framework React, versão 18.3.1+
- ESLint, versão 8.57.0
- Ferramenta de documentação - DocuSaurus, versão 2.8.0

Banco de dados:

- PostgreSQL, versão 15.3

Biblioteca de comunicação:

- WebSockets/Socket.IO

Infraestrutura:

- Aws Academy

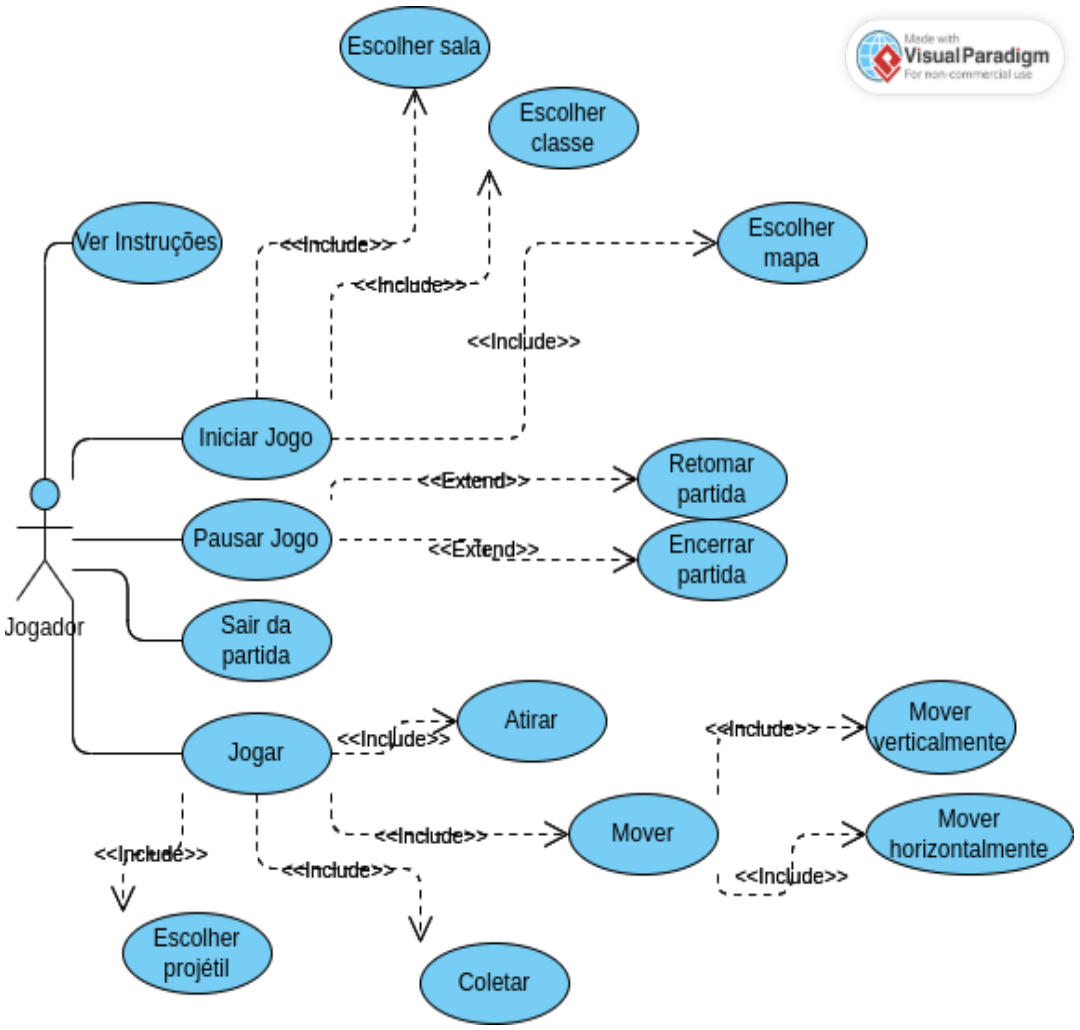
4. Representação Arquitetural

4.1. Visão de Cenários

O diagrama de caso de uso representa a interação do **Jogador** com o sistema do SpaceWar. Nesse jogo, os jogadores controlam espaçonaves para **atirar em outras espaçonaves adversárias** e **desviar de meteoros** enquanto disputam para sobreviver e acumular pontos.

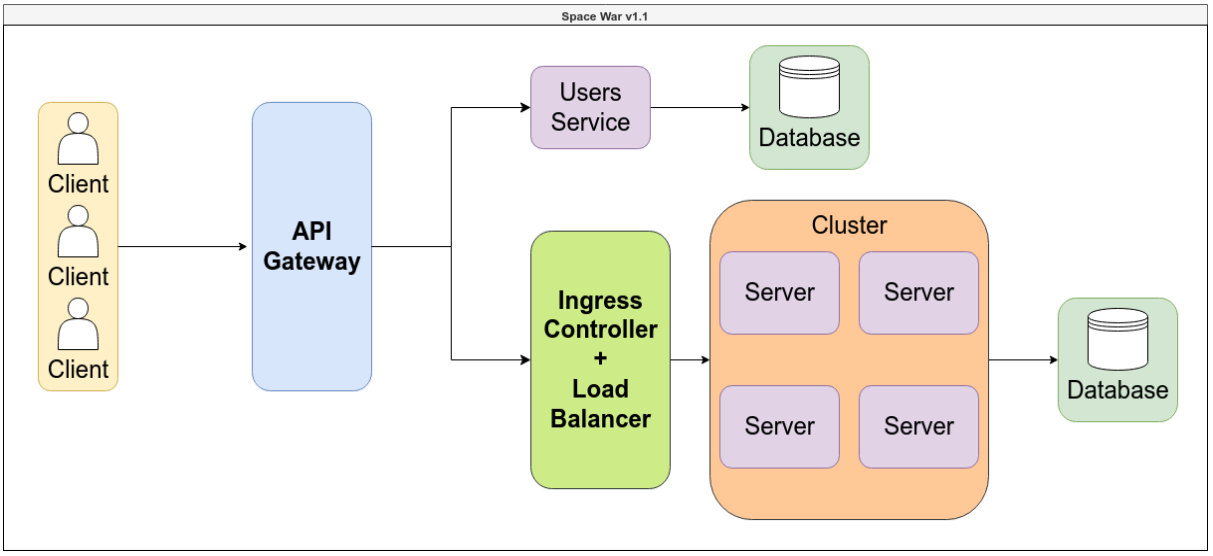
O único ator representado no diagrama é o **Jogador**, que pode realizar as seguintes ações:

- **Ver Instruções:** O jogador pode acessar um menu ou uma tela de ajuda que explica as regras do jogo, os controles e os objetivos principais.
- **Iniciar Jogo:** O jogador pode começar uma nova partida, seja entrando em um lobby existente ou criando uma nova sessão no servidor.
- **Jogar:** Durante a partida, o jogador pode **mover sua espaçonave**, **atirar em oponentes**, **desviar de meteoros** e **coletar bônus ou power-ups** que aparecem no cenário.
- **Pausar Jogo:** Em algumas modalidades, o jogador pode pausar a partida para acessar configurações ou aguardar antes de continuar. Caso o jogo seja multiplayer em tempo real, essa opção pode estar restrita ou funcionar apenas para ajustes rápidos.
- **Sair da Partida:** O jogador pode abandonar a partida antes do fim, voltando ao menu principal. Dependendo das regras do jogo, isso pode afetar sua pontuação ou estatísticas.



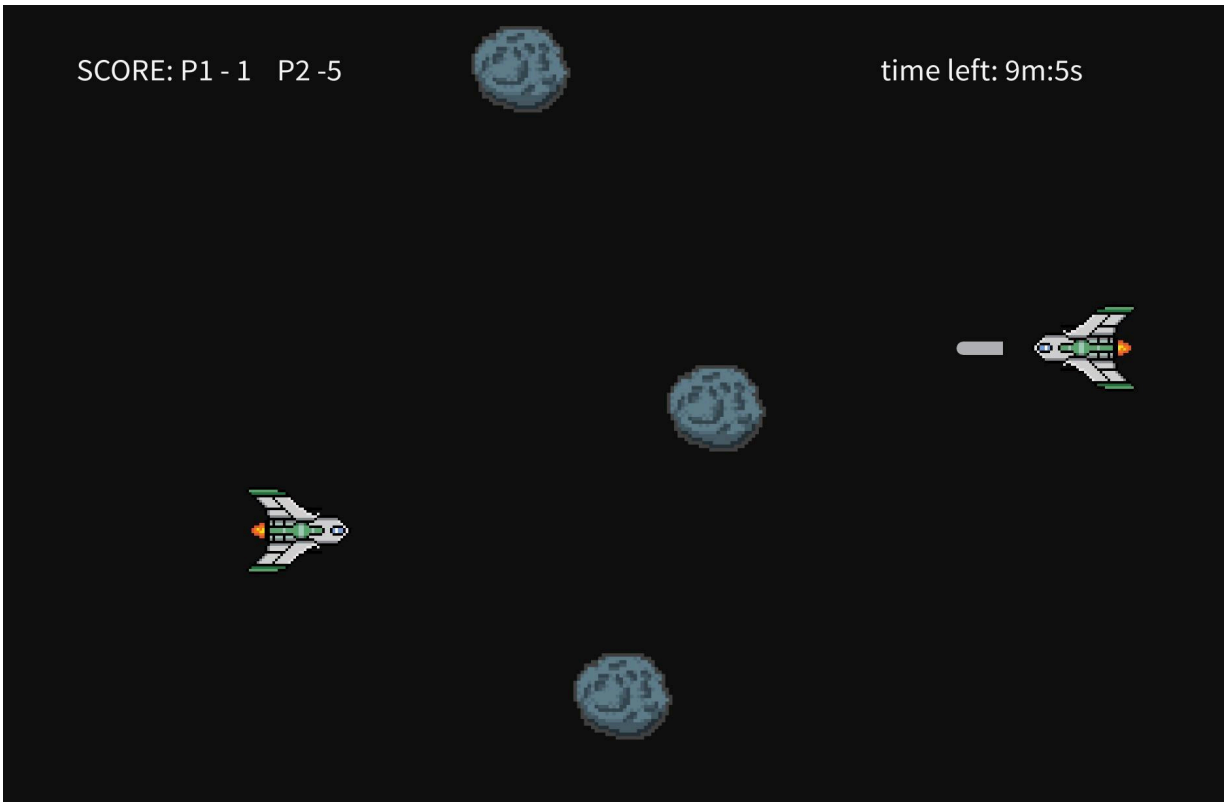
4.2. Visão de Implantação da Arquitetura

A arquitetura desejada seguirá um padrão de cliente-servidor implementada utilizando micro serviços, na qual as requisições dos clientes, inicialmente, serão tratadas por uma API Gateway para que sejam redirecionadas para o serviço correto. O serviço “users service” é uma API responsável por gerenciar as contas e a autenticação dos jogadores, para diminuir a latência dos servidores responsáveis pelo gerenciamento do estado do jogo. No caso do usuário já está autenticado, ou a requisição for relacionada ao estado do jogo, ela será encaminhada para o cluster, que usará um load balancer para decidir em qual servidor o jogador deve ser conectado, e um ingress controller para fazer o roteamento da requisição no cluster.



5. Protótipos de Telas

A imagem retrata o combate espacial entre duas espaçonaves, que trocam disparos de laser em meio ao vazio sideral. Os feixes de luz cruzam o espaço, deixando rastros brilhantes enquanto as naves realizam manobras evasivas. No cenário, diversos meteoros flutuam e se movem rapidamente, representando um obstáculo adicional para os jogadores. Algumas rochas espaciais parecem estar girando descontroladamente, enquanto outras seguem trajetórias imprevisíveis, tornando a batalha ainda mais desafiadora.



A imagem é a continuação da imagem acima, com mais um jogador entrando em combate com as duas espaçonaves.