Universidade Federal de Santa Catarina
Ciências da Computação
INE5412 - Sistemas Operacionais I
Márcio Castro, Giovani Gracioli

Leonardo Seishi Yamazaki (20102712)
Tainá da Cruz  (20100547)
Vitor Katsuziro Egami (18200443)

# Projeto Final: Jogo das Naves

Florianópolis, 26 de Junho de 2023

**SUMÁRIO**

# 1. INTRODUÇÃO

Este relatório descreve o desenvolvimento de um “Jogo de Naves" inspirado em um clássico videogame portátil. O jogo foi implementado em C++ utilizando a biblioteca SFML para a interface gráfica, além da biblioteca de Threads desenvolvida ao longo do semestre.

O jogo apresenta uma nave controlada pelo jogador, posicionada inicialmente no centro da tela, que tem a capacidade de atirar em naves inimigas. No máximo quatro naves inimigas existem simultaneamente, todas girando e atirando randomicamente. O jogador tem três vidas, e cada acerto em uma nave inimiga resulta em 100 pontos adicionados ao score. O jogo também possui diferentes níveis de velocidade.

Durante o desenvolvimento, buscou-se estruturar o projeto seguindo o paradigma de orientação a objetos do C++, com cada elemento do jogo, como as naves inimigas, a nave do jogador e a interface gráfica, sendo representado por uma classe. Além disso, cada nave inimiga, a nave do jogador, o desenho da tela e o tratamento da entrada do teclado são gerenciados por threads separadas.

Este relatório detalha a arquitetura do jogo, a implementação das threads, a lógica e os desafios encontrados durante o desenvolvimento.

# 2. DIAGRAMAS UML

Os Diagramas UML são utilizados para visualizar, especificar, construir e documentar os artefatos de um sistema de software, facilitando a compreensão de sua estrutura e funcionamento. No contexto deste projeto, foram desenvolvidos três diagramas: Diagrama de Caso de Uso, Diagrama de Classes e Diagrama de Sequência. A implementação ocorreu com a ferramenta PlantUML, que permite criar diagramas a partir de uma linguagem de texto simples.

## 2.1 Diagrama de Casos de Uso

O diagrama de casos de uso ilustra as interações entre o Ator (neste caso, o jogador) e o sistema (o “Jogo de Naves"). Cada linha conectando o jogador a um caso de uso representa uma ação que o jogador pode realizar no jogo. Essas ações representam as principais funcionalidades do jogo e a maneira como o Jogador interage com o sistema.

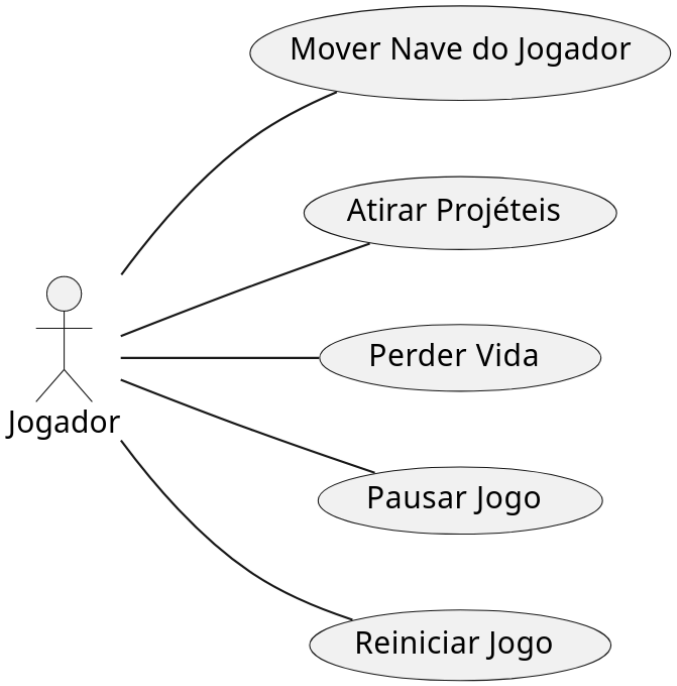



Figura 1 - Diagrama de Casos de Uso

## 2.2. Diagrama de Classes

O diagrama de classes está dividido em duas partes principais: o pacote da Biblioteca de Threads (*THREAD_LIBRARY*) e as classes relacionadas ao jogo. Em seguida, essas classes serão exemplificadas.

### 2.2.1 Biblioteca de Threads

O pacote contém várias classes e estruturas que ajudam na criação de threads e sincronização para o jogo, como demonstra o diagrama abaixo:

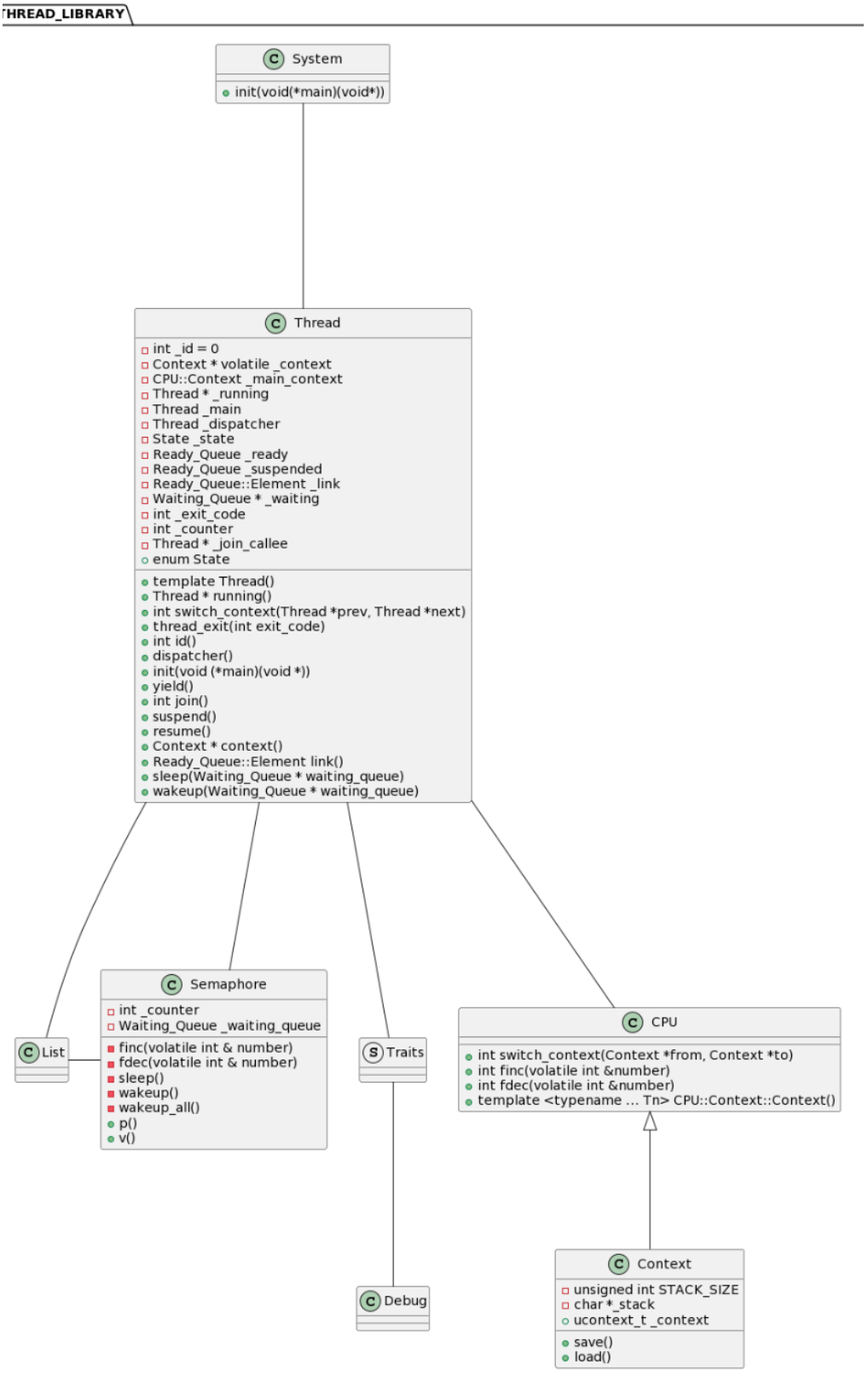



Figura 2 - Diagrama de Classes da Biblioteca de Threads

1. **Thread**: Esta classe realiza a implementação de uma thread, com propriedades para o contexto, estado, id, etc. Contém métodos para gerenciamento de threads, como troca de contexto, suspensão, retomada, etc.
2. **CPU**: É uma abstração de uma CPU, com métodos para troca de contexto e operações atômicas de incremento/decremento.
3. **Semaphore**: Classe que gerencia a sincronização entre threads. Possui métodos para incrementar e decrementar um contador e para colocar uma thread para dormir e acordá-la.
4. **System e Context:** A classe System realiza a inicialização em todo o sistema. A classe Context é usada para gerenciar o contexto de uma thread (seu estado e dados associados).

### 2.2.2 Jogo das Naves

Já na parte do diagrama relacionada ao jogo, temos classes como:

1. **Game Entity:** Esta é a interface para as entidades do jogo, com métodos e propriedades que todas as entidades devem ter, como velocidade, x, y, direção, e métodos para movimento, desenho, etc.
2. **Game**: Essa classe inicializa e gerencia o estado do jogo, definindo a criação das threads, janela, entrada, estado do jogo, e estatísticas. Contém métodos para gerenciar o ciclo de vida do jogo, como iniciar, pausar, atualizar, terminar, etc.
3. **SpaceShip, PlayerShip, EnemyShip**: Essas classes representam as naves do jogo. A classe SpaceShip é filha de Game Entity, enquanto que PlayerShip e EnemyShip são classes filhas de SpaceShip. Cada nave tem um conjunto próprio de propriedades e métodos que definem seu comportamento.

   3.1 **SpaceShip** : terá como função exclusiva shoot() que será responsável por criar Threads que irão rodar o Projectile::runProjectile(), e após deletá-los, funcionando de maneira parecida a main, porém um nível de threads filhas abaixo.
4. **Projectile**: Representa um projétil que pode ser disparado pelas naves.
5. **Input**: É a classe utilizada para gerenciar a entrada do jogador.

6. **Game Window**: Define o gerenciamento da janela do jogo e renderização das entidades de jogo.

Essas classes interagem entre si para formar a estrutura do jogo, e as relações entre elas são mostradas abaixo. A linha saindo da classe Game representa a ligação com a Biblioteca de Threads, como pode ser verificado no Anexo 1.

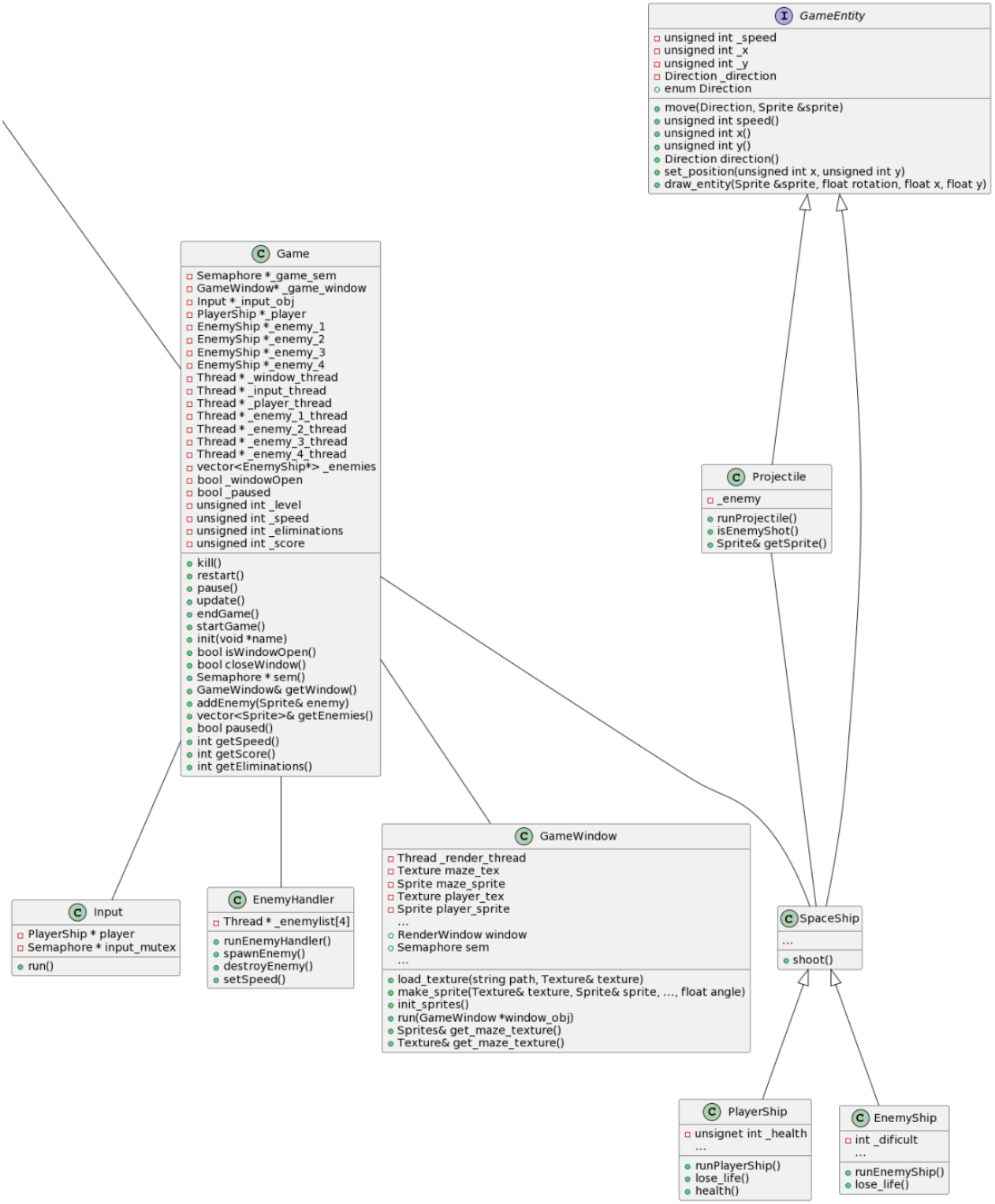



Figura 3 - Diagrama de Classes do Jogo das Naves

## 2.3 Diagrama de Sequência

### 2.3.1 Criação das Threads

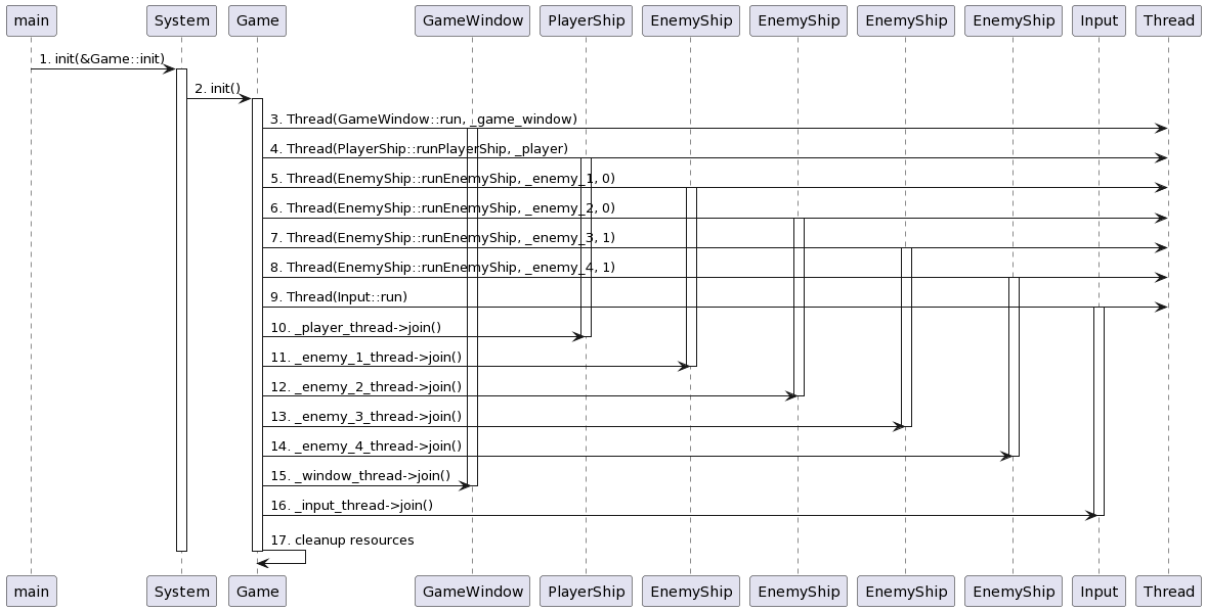



Figura 4 - Diagrama de Sequência do Ciclo de Vida das Threads

### 2.3.2 Tiro Disparado

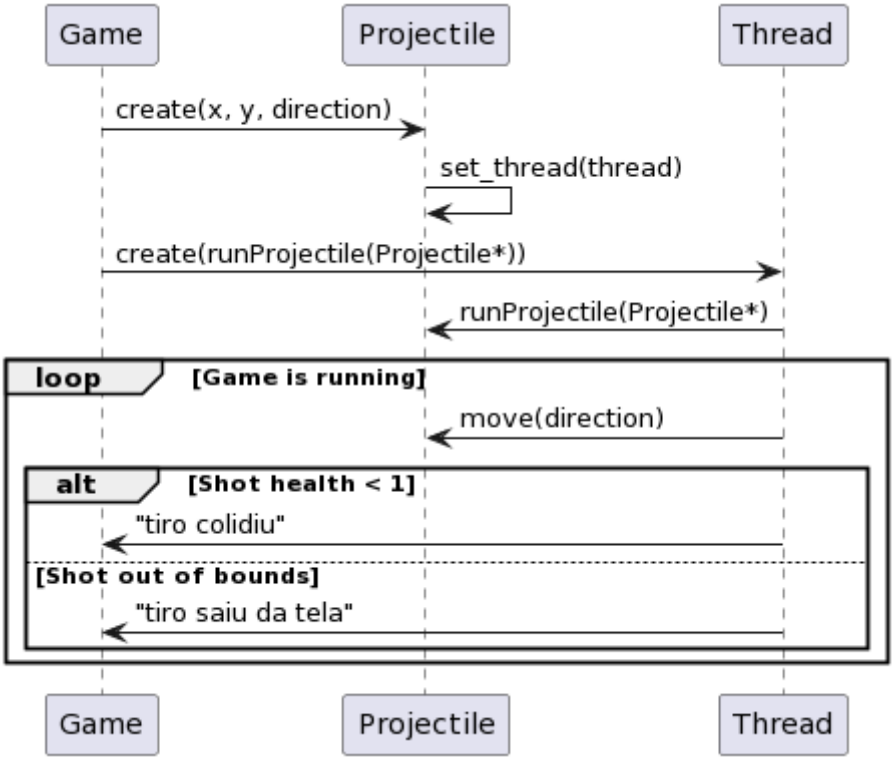
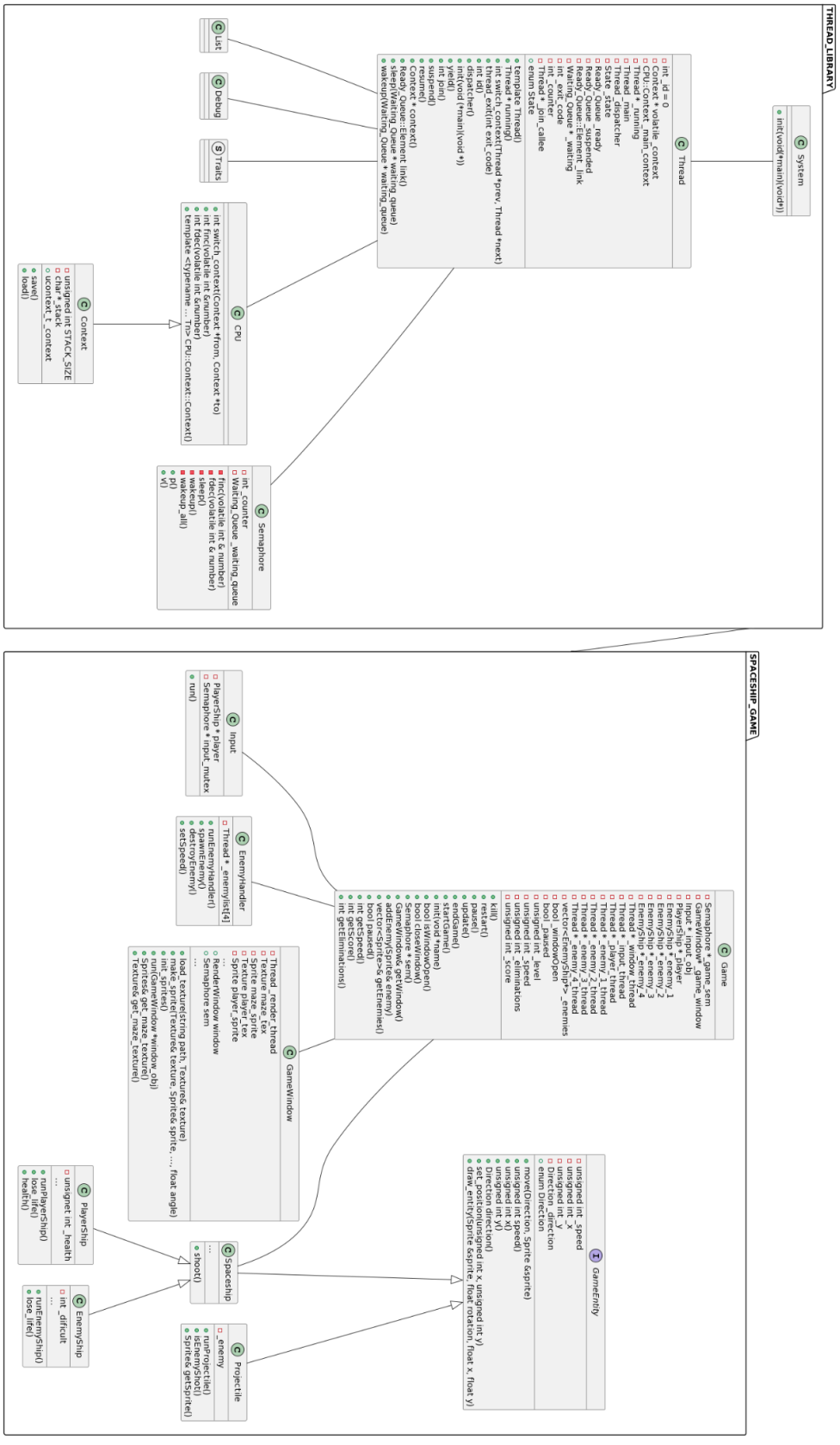



Figura 5 - Diagrama de Sequência de Tiro Disparado

# ANEXO 1


THREAD_LIBRARY
System
Thread
List
Debug
Traits
CPU
Context
Semaphore
SPACESHIP_GAME
Input
EnemyHandler
Game
GameWindow
GameEntity
Spaceship
PlayerShip
EnemyShip
Projectile