UNIVASSOURAS
ENGENHARIA DE SOFTWARE

LEANDRO LIMA CARDOSO (202323366)

DOCUMENTO DE ARQUITETURA DE SOFTWARE
DESENVOLVIMENTO DE UMA *GAME ENGINE* 2D EM FORMATO DE BIBLIOTECA EM C++

PRÁTICAS EXTENSIONISTAS INTEGRADORAS VI

SAQUAREMA
2026

LEANDRO LIMA CARDOSO (202323366)

DOCUMENTO DE ARQUITETURA DE SOFTWARE
DESENVOLVIMENTO DE UMA *GAME ENGINE* 2D EM FORMATO DE BIBLIOTECA EM C++

PRÁTICAS EXTENSIONISTAS INTEGRADORAS VI

Trabalho apresentado como requisito parcial para obtenção de nota da disciplina de **Práticas Extensionistas Integradoras VI** do curso de Engenharia de Software da Univassouras (campus universitário de Saquarema).

Prof. Dr. Altemar Sales

SAQUAREMA

2026

RESUMO

Este documento apresenta a arquitetura de software da *engine* de jogos bidimensionais desenvolvida na linguagem de programação C++. O objetivo deste documento é descrever a estrutura geral do sistema, seus principais componentes e a forma como esses elementos interagem entre si para fornecer as funcionalidades necessárias ao desenvolvimento de jogos 2D. A arquitetura proposta foi projetada com foco em modularidade, organização estrutural e facilidade de manutenção, permitindo que diferentes partes do sistema sejam desenvolvidas e evoluídas de forma independente. O documento descreve o estilo arquitetural adotado, os principais módulos que compõem a *engine* e as responsabilidades associadas a cada componente. Além disso, são apresentados os elementos estruturais que compõem o núcleo do sistema, incluindo gerenciamento do *loop* principal do jogo, renderização gráfica, gerenciamento de entidades, controle de entrada do usuário, gerenciamento de recursos e sistema de colisões. A definição clara da arquitetura do sistema contribui para orientar o desenvolvimento da *engine*, facilitar a compreensão de sua estrutura e apoiar futuras extensões ou melhorias.

Palavras-chave: *game engine*; C++; arquitetura de software; jogos 2D; desempenho.

SUMÁRIO

1 INTRODUÇÃO

1.1 Propósito

O objetivo deste documento é apresentar a arquitetura de software da *engine* de jogos 2D desenvolvida no contexto do Trabalho de Conclusão de Curso. A arquitetura define a estrutura organizacional do sistema, os principais módulos que o compõem e as responsabilidades associadas a cada componente.

A descrição arquitetural permite compreender como o sistema está estruturado e como seus diferentes módulos interagem para fornecer as funcionalidades necessárias ao desenvolvimento de jogos digitais.

Este documento serve como referência para o desenvolvimento, manutenção e evolução da *engine*, auxiliando na organização do código-fonte e na definição de responsabilidades dentro da estrutura do sistema.

1.2 Escopo

Este documento descreve a arquitetura da *engine* de jogos 2D desenvolvida em C++, apresentando os principais componentes responsáveis pelo funcionamento do sistema.

A arquitetura contempla módulos responsáveis por:

- O gerenciamento do núcleo da engine;
- O controle do loop principal do jogo;
- A renderização gráfica de elementos bidimensionais;
- O gerenciamento de entidades presentes no ambiente do jogo;
- A detecção de colisões entre objetos;

- O gerenciamento de recursos utilizados pela aplicação;
- O processamento de entrada do usuário.

A definição desses componentes permite estruturar o sistema de forma organizada, garantindo modularidade e facilitando futuras expansões.

1.3 Visão Geral do Documento

Este documento está organizado da seguinte forma:

- A Seção 2 apresenta uma visão geral da arquitetura do sistema;
- A Seção 3 descreve o estilo arquitetural adotado;
- A Seção 4 apresenta os principais componentes da *engine*;
- A Seção 5 descreve os módulos que compõem o sistema;
- A Seção 6 apresenta o fluxo geral de execução da *engine*.

2 VISÃO GERAL DA ARQUITETURA

A arquitetura da *engine* foi projetada com o objetivo de fornecer uma estrutura modular e organizada para o desenvolvimento de jogos bidimensionais utilizando a linguagem de programação C++. Essa abordagem permite que diferentes componentes do sistema sejam desenvolvidos e mantidos de forma independente, favorecendo a organização do código e facilitando futuras extensões.

A *engine* é composta por um conjunto de módulos responsáveis por diferentes aspectos do funcionamento do sistema. Cada módulo possui responsabilidades específicas e interage com os demais componentes de maneira estruturada, garantindo que o fluxo de execução do jogo ocorra de forma consistente.

De forma geral, a arquitetura do sistema é composta pelos seguintes componentes principais:

- ***Core* (Núcleo da *Engine*)**
  Responsável pela inicialização do sistema, gerenciamento do ciclo de execução e coordenação geral dos módulos da *engine*.

- ***Renderer* (Sistema de Renderização)**
  Responsável pela renderização de elementos gráficos bidimensionais na tela, incluindo sprites e outros componentes visuais utilizados no jogo.

- ***Entity System* (Sistema de Entidades)**
  Responsável pelo gerenciamento das entidades presentes no ambiente do jogo, permitindo a criação, atualização e remoção de objetos durante a execução.

- ***Collision System* (Sistema de Colisão)**
  Responsável pela detecção de colisões entre entidades, permitindo que interações físicas ou lógicas sejam processadas pelo sistema.

- ***Input System* (Sistema de Entrada)**
  Responsável por capturar e processar eventos de entrada do usuário provenientes de dispositivos como teclado e mouse.

- ***Resource Manager* (Gerenciador de Recursos)**
  Responsável pelo carregamento, armazenamento e gerenciamento de recursos utilizados pela aplicação, como imagens e arquivos de mapa.

- ***Scene Manager* (Gerenciador de Cenas)**
  Responsável pela organização do jogo em diferentes estados ou cenas, permitindo a transição entre diferentes contextos da aplicação.

A interação entre esses componentes permite que o sistema execute o ciclo fundamental de funcionamento de um jogo, que envolve o processamento de entrada do usuário, atualização da lógica do jogo e renderização dos elementos gráficos.

A organização modular da arquitetura facilita a compreensão da estrutura do sistema, além de permitir que novos componentes ou funcionalidades sejam adicionados sem comprometer a organização geral da *engine*.

3 ESTILO ARQUITETURAL

A arquitetura da *engine* foi projetada com base em um modelo modular, no qual o sistema é dividido em componentes independentes responsáveis por funcionalidades específicas. Esse estilo arquitetural permite que cada módulo seja desenvolvido e mantido de forma isolada, reduzindo o acoplamento entre os componentes e facilitando a evolução do sistema ao longo do tempo.

A utilização de uma arquitetura modular contribui para a organização do código-fonte e melhora a manutenibilidade do *software*, uma vez que alterações em um módulo tendem a causar menor impacto nos demais componentes do sistema.

Além da modularidade, a arquitetura também adota princípios de organização em camadas. Nesse modelo, os componentes do sistema são organizados de forma hierárquica, onde determinados módulos fornecem serviços para outros componentes localizados em níveis superiores da estrutura.

De maneira geral, a arquitetura da *engine* pode ser organizada em três níveis principais:

- ***Core Layer*** **(Núcleo do Sistema)**

Responsável pela inicialização da *engine*, gerenciamento do ciclo de execução do jogo e coordenação geral dos módulos do sistema.

- ***System Layer* (Camada de Sistemas)**
  Composta pelos módulos responsáveis por funcionalidades específicas da *engine*, como renderização gráfica, gerenciamento de entidades, sistema de colisão, processamento de entrada do usuário e gerenciamento de recursos.

- ***Application Layer* (Camada de Aplicação)**
  Representa o código do jogo desenvolvido pelo usuário da *engine*, que utiliza os serviços fornecidos pelos módulos internos do sistema.

Essa organização permite separar claramente as responsabilidades de cada parte do sistema, facilitando a compreensão da arquitetura e a manutenção do código.

Outro aspecto importante do estilo arquitetural adotado é a utilização de interfaces bem definidas entre os módulos. Essas interfaces permitem que diferentes componentes se comuniquem de forma estruturada, reduzindo dependências diretas e tornando o sistema mais flexível para futuras modificações.

A adoção desses princípios arquiteturais contribui para a criação de uma *engine* organizada, extensível e adequada para fins educacionais e experimentação no desenvolvimento de jogos 2D.

4 COMPONENTES DO SISTEMA

A arquitetura da *engine* é composta por um conjunto de componentes responsáveis por diferentes funcionalidades essenciais para o funcionamento do sistema. Cada

componente possui responsabilidades bem definidas e interage com os demais módulos de forma estruturada, contribuindo para a organização e modularidade da *engine*.

A divisão do sistema em componentes permite reduzir o acoplamento entre as partes do *software*, facilitando a manutenção, a expansão e a reutilização de funcionalidades.

A seguir são apresentados os principais componentes que compõem a arquitetura da *engine*.

4.1 *Core*

O componente *Core* representa o núcleo da *engine* e é responsável por coordenar o funcionamento geral do sistema.

Entre suas principais responsabilidades estão a inicialização da *engine*, o gerenciamento do ciclo de execução do jogo e a coordenação dos demais módulos que compõem o sistema.

O *Core* também é responsável por controlar o *loop* principal da aplicação, garantindo que as etapas de atualização da lógica do jogo e renderização gráfica ocorram de maneira organizada durante a execução.

4.2 *Renderer*

O componente *Renderer* é responsável pelo processamento e exibição dos elementos gráficos do jogo na tela.

Esse módulo gerencia a renderização de *sprites* e outros elementos bidimensionais utilizados na construção do ambiente do jogo. O sistema de renderização recebe

informações das entidades presentes na cena e processa sua representação visual durante cada ciclo de execução do jogo.

4.3 *Entity System*

O *Entity System* é responsável pelo gerenciamento das entidades presentes no ambiente do jogo.

Uma entidade representa qualquer objeto existente no mundo do jogo, como personagens, elementos do cenário ou objetos interativos. Esse componente permite a criação, atualização e remoção de entidades durante a execução da aplicação.

O gerenciamento estruturado das entidades facilita a organização da lógica do jogo e permite que diferentes objetos sejam manipulados de forma consistente.

4.4 *Collision System*

O *Collision System* é responsável pela detecção de colisões entre entidades presentes no ambiente do jogo.

Esse módulo verifica a interseção entre objetos e fornece informações que podem ser utilizadas pela lógica do jogo para determinar interações, como bloqueios de movimento ou eventos específicos.

A implementação de um sistema de colisão é fundamental para permitir a interação entre elementos presentes no ambiente do jogo.

4.5 *Input System*

O *Input System* é responsável por capturar e processar eventos de entrada provenientes do usuário.

Esse componente permite que a aplicação receba comandos através de dispositivos como teclado e mouse, convertendo esses eventos em ações que podem ser utilizadas pela lógica do jogo.

O processamento adequado das entradas do usuário é essencial para permitir a interação entre o jogador e o ambiente do jogo.

4.6 *Resource Manager*

O *Resource Manager* é responsável pelo gerenciamento dos recursos utilizados pela aplicação.

Entre esses recursos estão arquivos de imagem, *sprites* e outros elementos necessários para a execução do jogo. Esse módulo centraliza o carregamento e o armazenamento desses recursos, evitando duplicação e facilitando o controle de memória.

4.7 *Scene Manager*

O *Scene Manager* é responsável pela organização do jogo em diferentes cenas ou estados.

Cada cena representa um contexto específico da aplicação, como *menus*, fases ou telas de configuração. Esse componente permite controlar a transição entre diferentes partes do jogo de forma estruturada.

5 DESCRIÇÃO DOS MÓDULOS

Esta seção apresenta uma descrição mais detalhada dos módulos que compõem a arquitetura da *engine*, destacando suas responsabilidades principais e sua interação com os demais componentes do sistema.

A divisão da *engine* em módulos permite organizar melhor o código-fonte e garantir que cada parte do sistema possua responsabilidades bem definidas, contribuindo para a manutenção e evolução do *software*.

5.1 Módulo *Core*

O módulo *Core* é responsável por controlar o funcionamento geral da *engine*. Ele atua como o ponto central de coordenação entre os diferentes componentes do sistema.

Entre suas principais responsabilidades estão:

- Inicialização da *engine*;
- Criação e gerenciamento da janela de execução;
- Controle do *loop* principal do jogo;
- Coordenação da atualização dos sistemas internos da *engine*;
- Encerramento seguro da aplicação.

O *Core* interage diretamente com os demais módulos da *engine*, garantindo que o fluxo de execução ocorra de forma organizada durante cada ciclo do jogo.

5.2 Módulo de Renderização

O módulo de Renderização (*Renderer*) é responsável por exibir os elementos gráficos do jogo na tela.

Esse componente recebe informações sobre os objetos presentes na cena e realiza o processo de renderização dos elementos visuais, como *sprites* e outros objetos gráficos.

Entre suas principais responsabilidades estão:

- Renderização de *sprites*;
- Atualização da tela a cada ciclo de execução;
- Gerenciamento do processo de desenho dos elementos do jogo.

Esse módulo trabalha em conjunto com o sistema de entidades e com o gerenciador de cenas para determinar quais elementos devem ser exibidos durante cada quadro da execução.

5.3 Módulo de Entidades

O *Entity System* é responsável pelo gerenciamento das entidades presentes no ambiente do jogo.

Cada entidade representa um objeto dentro do mundo do jogo, podendo possuir diferentes características e comportamentos.

Entre as responsabilidades desse módulo estão:

- Criação de entidades;
- Atualização de entidades durante o *loop* do jogo;
- Remoção de entidades quando necessário;
- Organização das entidades dentro da cena atual.

Esse sistema permite que a lógica do jogo manipule diferentes objetos de forma estruturada.

5.4 Módulo de Colisão

O *Collision System* é responsável por identificar quando ocorre interação entre entidades presentes no ambiente do jogo.

Esse módulo analisa a posição e os limites das entidades para verificar possíveis interseções entre objetos.

Entre suas principais responsabilidades estão:

- Detecção de colisões entre entidades;
- Identificação de interseções entre objetos;
- Fornecimento de informações para a lógica do jogo sobre colisões detectadas.

Essas informações podem ser utilizadas para controlar movimentação, bloqueios ou eventos específicos dentro do jogo.

5.5 Módulo de Entrada

O *Input System* é responsável por capturar eventos de entrada provenientes do usuário.

Esse módulo recebe sinais gerados por dispositivos como teclado e mouse e os converte em comandos que podem ser utilizados pela lógica do jogo.

Entre suas principais responsabilidades estão:

- Captura de eventos de teclado;
- Captura de eventos de *mouse*;
- Disponibilização dessas informações para o restante da *engine*.

Esse componente permite que o jogador interaja com o ambiente do jogo.

5.6 Módulo de Gerenciamento de Recursos

O *Resource Manager* é responsável pelo carregamento e gerenciamento dos recursos utilizados pela aplicação.

Entre esses recursos estão arquivos de imagem, *sprites* e outros elementos necessários para a execução do jogo.

As principais responsabilidades desse módulo incluem:

- Carregamento de recursos gráficos;
- Armazenamento e gerenciamento desses recursos;
- Fornecimento dos recursos para outros módulos da *engine* quando necessário.

Esse gerenciamento centralizado contribui para melhor controle de memória e organização dos arquivos utilizados pelo jogo.

5.7 Módulo de Gerenciamento de Cenas

O *Scene Manager* é responsável pela organização da aplicação em diferentes cenas.

Cada cena representa um estado específico do jogo, como *menus*, níveis ou telas de pausa.

Entre as responsabilidades desse módulo estão:

- Criação e gerenciamento de cenas;
- Controle da cena atualmente ativa;
- Transição entre diferentes cenas do jogo.

Esse sistema permite estruturar a aplicação de forma organizada, separando diferentes partes do jogo em contextos distintos.

6 FLUXO DE EXECUÇÃO DA *ENGINE*

O fluxo de execução da *engine* descreve a sequência de operações realizadas durante a execução de um jogo desenvolvido utilizando o sistema. Esse fluxo representa o ciclo fundamental de funcionamento da aplicação, conhecido como *loop* principal do jogo.

O *loop* principal é responsável por garantir que o jogo seja atualizado continuamente enquanto a aplicação estiver em execução. Durante cada ciclo do *loop*, a *engine* realiza uma série de etapas que permitem processar a interação do usuário, atualizar o estado do jogo e renderizar os elementos gráficos na tela.

De forma geral, o fluxo de execução da *engine* pode ser dividido nas seguintes etapas:

6.1 Inicialização do Sistema

A execução da *engine* inicia com a fase de inicialização do sistema. Nesse momento, o módulo Core é responsável por configurar os componentes principais necessários para o funcionamento da aplicação.

Durante essa etapa são realizadas operações como:

- A inicialização da *engine*;
- A criação da janela de execução;
- A inicialização dos módulos internos do sistema;
- O carregamento inicial de recursos necessários para o jogo.

Após a conclusão da fase de inicialização, a *engine* entra no *loop* principal de execução.

6.2 Processamento de Entrada

Durante cada ciclo do *loop* do jogo, o *Input System* é responsável por capturar e processar eventos provenientes do usuário.

Esses eventos podem incluir:

- O pressionamento de teclas;
- A movimentação do *mouse*;
- Os cliques de *mouse*.

As informações capturadas são disponibilizadas para a lógica do jogo, permitindo que o jogador interaja com o ambiente virtual.

6.3 Atualização da Lógica do Jogo

Após o processamento das entradas do usuário, o sistema realiza a atualização da lógica do jogo.

Durante essa etapa, o *Entity System* atualiza o estado das entidades presentes no ambiente do jogo. Essa atualização pode incluir:

- A movimentação de personagens;
- A atualização de comportamentos;
- A execução de ações específicas definidas pela lógica do jogo.

Essa etapa garante que o estado do jogo evolua ao longo do tempo.

6.4 Detecção de Colisões

Após a atualização das entidades, o *Collision System* realiza a verificação de possíveis colisões entre os objetos presentes no ambiente do jogo.

Caso sejam detectadas interseções entre entidades, essas informações podem ser utilizadas pela lógica do jogo para determinar interações específicas, como bloqueio de movimento, coleta de objetos ou ativação de eventos.

6.5 Renderização da Cena

Após a atualização da lógica do jogo e a verificação de colisões, o módulo *Renderer* realiza o processo de renderização da cena atual.

Durante essa etapa, os elementos gráficos associados às entidades presentes na cena são desenhados na tela, permitindo que o jogador visualize o estado atual do jogo.

Esse processo ocorre repetidamente a cada ciclo do *loop* principal, atualizando continuamente a imagem exibida na tela.

6.6 Encerramento da Aplicação

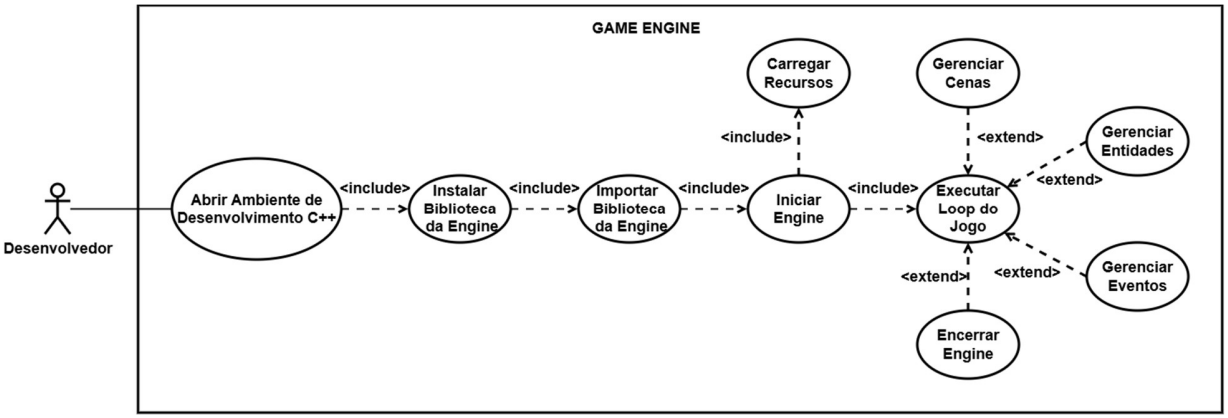
O *loop* principal da *engine* permanece em execução até que seja solicitado o encerramento da aplicação.

Quando isso ocorre, o módulo *Core* é responsável por finalizar a execução do sistema, liberando os recursos utilizados pela *engine* e encerrando a aplicação de forma segura.

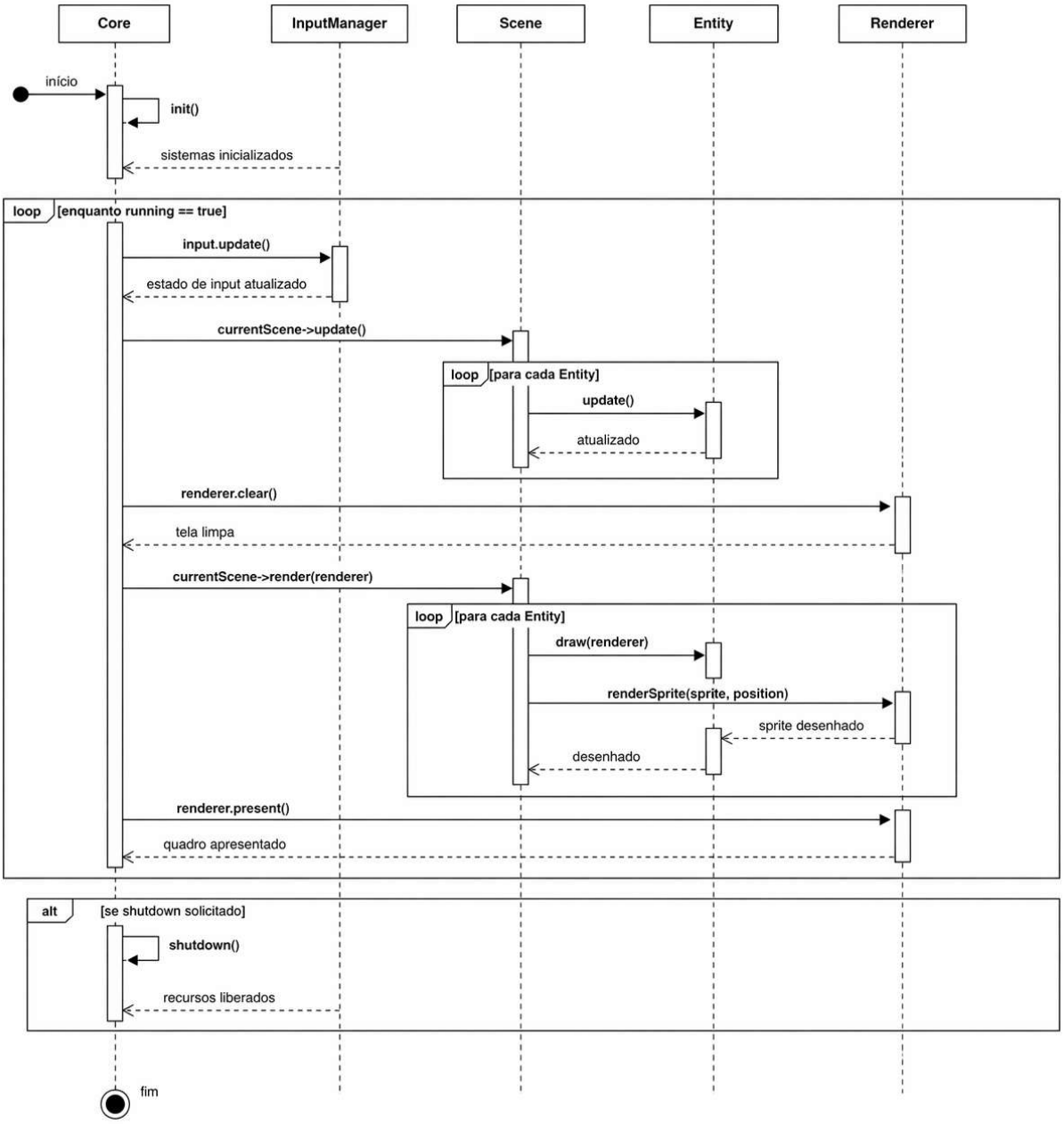
7 DIAGRAMAS

Os diagramas foram estruturado seguindo os princípios de baixo acoplamento e responsabilidade única, utilizando composição para representar dependência forte e associação para relações mais flexíveis, conforme a notação padrão da UML.

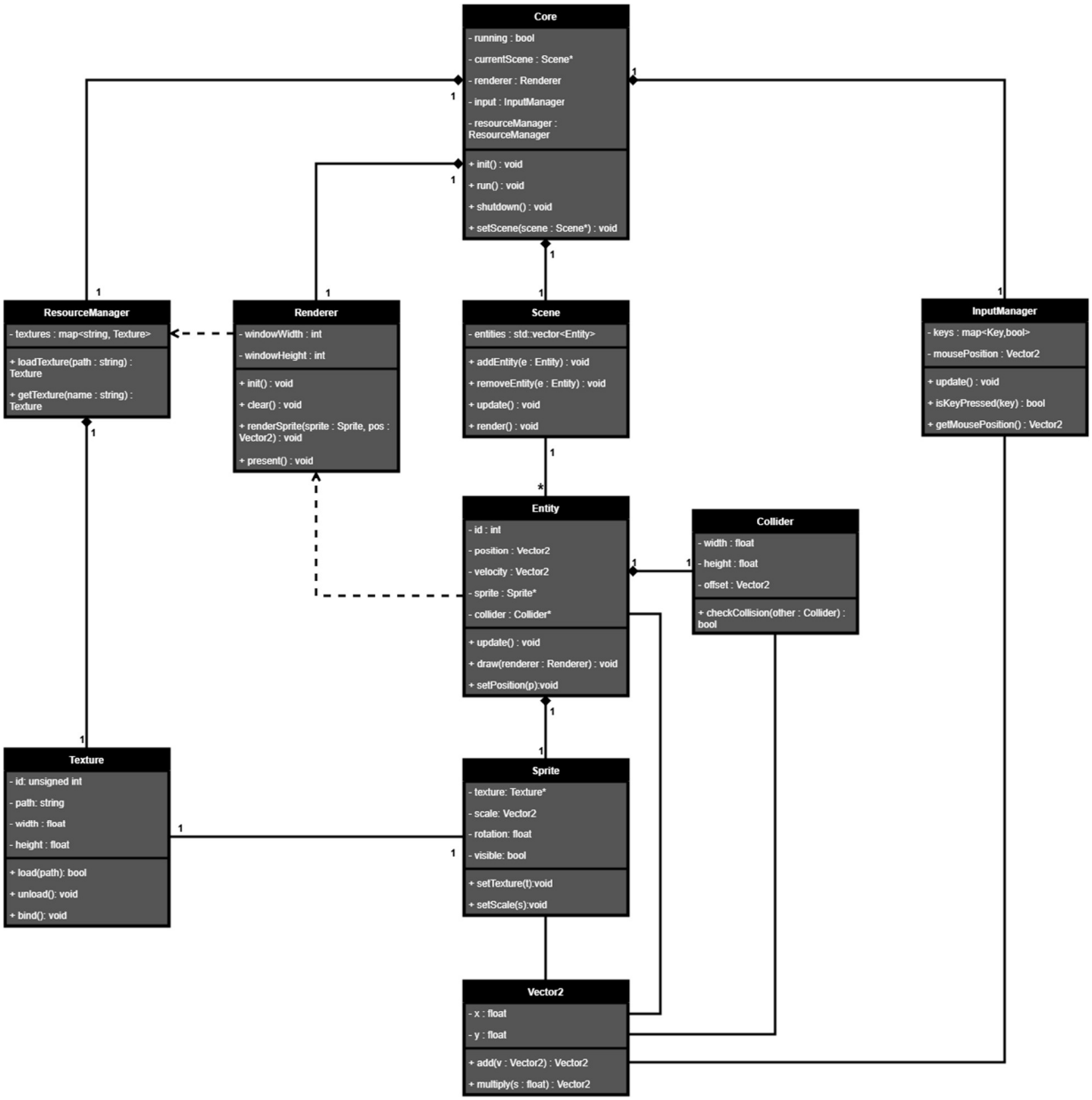
7.1 Caso de Uso



7.2 Diagrama de Sequencia



7.3 Classe e Relacionamento


Core
- running : bool
- currentScene : Scene*
- renderer : Renderer
- input : InputManager
- resourceManager : ResourceManager
+ init() : void
+ run() : void
+ shutdown() : void
+ setScene(scene : Scene*) : void
ResourceManager
- textures : map<string, Texture>
+ loadTexture(path : string) : Texture
+ getTexture(name : string) : Texture
Renderer
- windowWidth : int
- windowHeight : int
+ init() : void
+ clear() : void
+ renderSprite(sprite : Sprite, pos : Vector2) : void
+ present() : void
Scene
- entities : std::vector<Entity>
+ addEntity(e : Entity) : void
+ removeEntity(e : Entity) : void
+ update() : void
+ render() : void
InputManager
- keys : map<Key,bool>
- mousePosition : Vector2
+ update() : void
+ isKeyPressed(key) : bool
+ getMousePosition() : Vector2
Entity
- id : int
- position : Vector2
- velocity : Vector2
- sprite : Sprite*
- collider : Collider*
+ update() : void
+ draw(renderer : Renderer) : void
+ setPosition(p):void
Collider
- width : float
- height : float
- offset : Vector2
+ checkCollision(other : Collider) : bool
Texture
- id: unsigned int
- path: string
- width : float
- height : float
+ load(path): bool
+ unload(): void
+ bind(): void
Sprite
- texture: Texture*
- scale: Vector2
- rotation: float
- visible: bool
+ setTexture(t):void
+ setScale(s):void
Vector2
- x : float
- y : float
+ add(v : Vector2) : Vector2
+ multiply(s : float) : Vector2