JUMP KING

Técnicas de Programação – S71

Henrique Mendes & Rafael Felipe

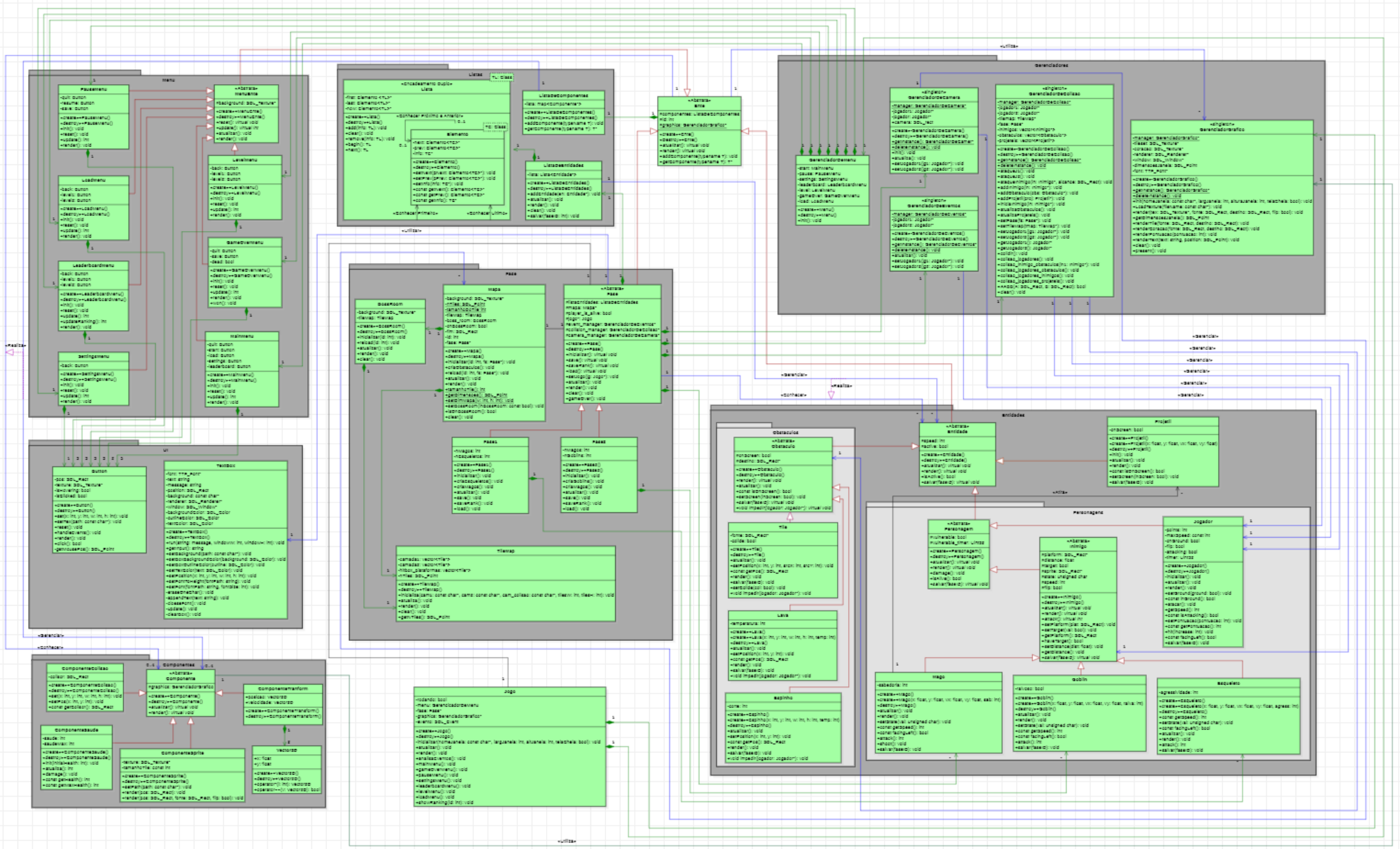
Tabela de Requisitos

N.	1	2	3	4	5
Requisitos Funcionais	Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, ver colocação (*ranking*) de jogadores e demais opções pertinentes.	Permitir um ou dois jogadores com representação gráfica aos usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante.	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas, via menu, nas quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa.	Ter pelo menos três tipos distintos de inimigos, cada qual com sua representação gráfica, sendo que ao menos um dos inimigos deve ser capaz de lançar projetil contra o(s) jogador(es) e um dos inimigos dever ser um 'Chefão'.	Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias e sendo pelo menos 3 instâncias por tipo.
Situação	Requisito previsto inicialmente e realizado.	Requisito previsto inicialmente e realizado	Requisito previsto inicialmente e realizado	Requisito previsto inicialmente e realizado	Requisito previsto inicialmente e realizado
Implementação	Realizado através do *namespace* Menu e da classe Jogo	Realizado através das classes Jogador e GerenciadorDeEventos	Realizado através das classes Fase, Fase1, e Fase2	Realizado através das classes Inimigo, Esqueleto, Goblin e Mago	Realizado através do método virtual inicializar() da Fase, que instancia os inimigos pseudo-aleatoriamente

N.	6	7	8	9	10
Requisitos Funcionais	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem.	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório de instâncias (*i.e.*, objetos), sendo pelo menos 3 instâncias por tipo.	Ter em cada fase um cenário de jogo constituído por obstáculos, sendo que parte deles seriam plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores.	Gerenciar colisões entre jogador para com inimigos e seus projeteis, bem como entre jogador para com obstáculos. Ainda, todos eles devem sofrer o efeito da gravidade no âmbito deste jogo de plataforma vertical e 2D.	Permitir: (1) salvar nome do usuário, manter/salvar pontuação do jogador (incrementada via neutralização de inimigos) controlado pelo usuário e gerar lista de pontuação (*ranking*). **E** (2) Pausar e **Salvar** Jogada.
Situação	Requisito previsto inicialmente e realizado.	Requisito previsto inicialmente e realizado	Requisito previsto inicialmente e realizado	Requisito previsto inicialmente e realizado	Requisito previsto inicialmente e realizado
Implementação	Realizado através das classes Obstaculo, Lava, Espinho e Tile	Realizado através da classe TileMap, que gera os obstáculos aleatoriamente dentro do mapa da Fase	Realizado na construção das diferentes fases	Realizado através do GerenciadorDeColisao	Realizado através das classes derivadas de Fase e do GerenciadorDeSave

Percentagem concluída: 100%

Diagrama de Classes UML

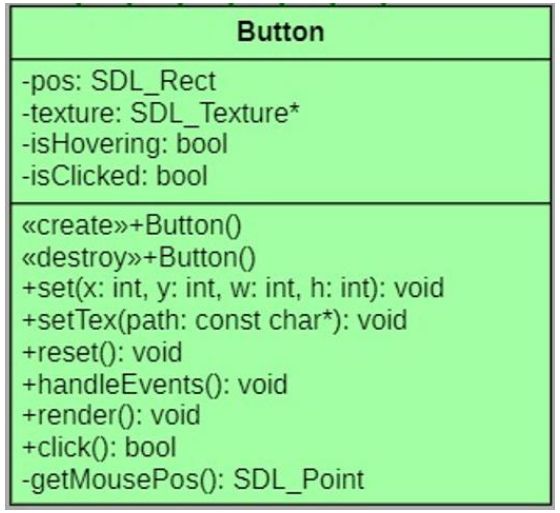


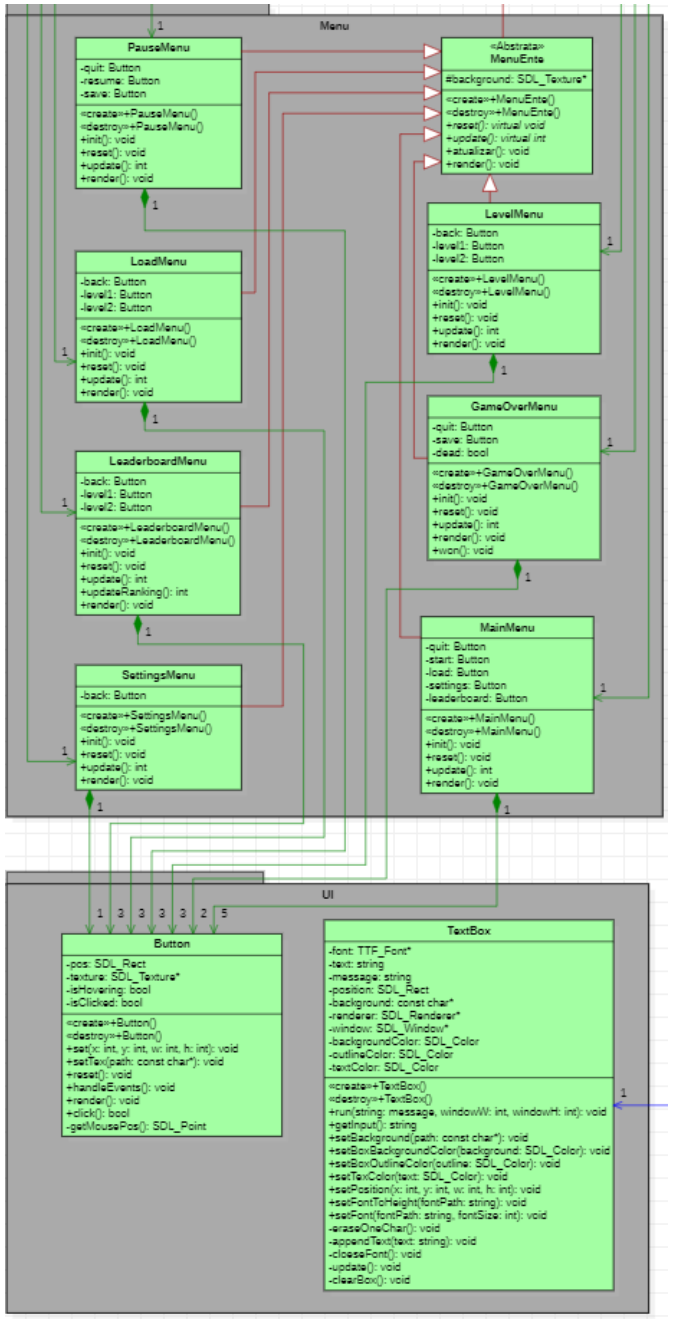
O projeto foi desenvolvido dividido em 7 pacotes Principais:

- Menu
- UI (User Interface)
- Listas
- Fases
- Gerenciadores
- Componentes
- Entidades

Menu

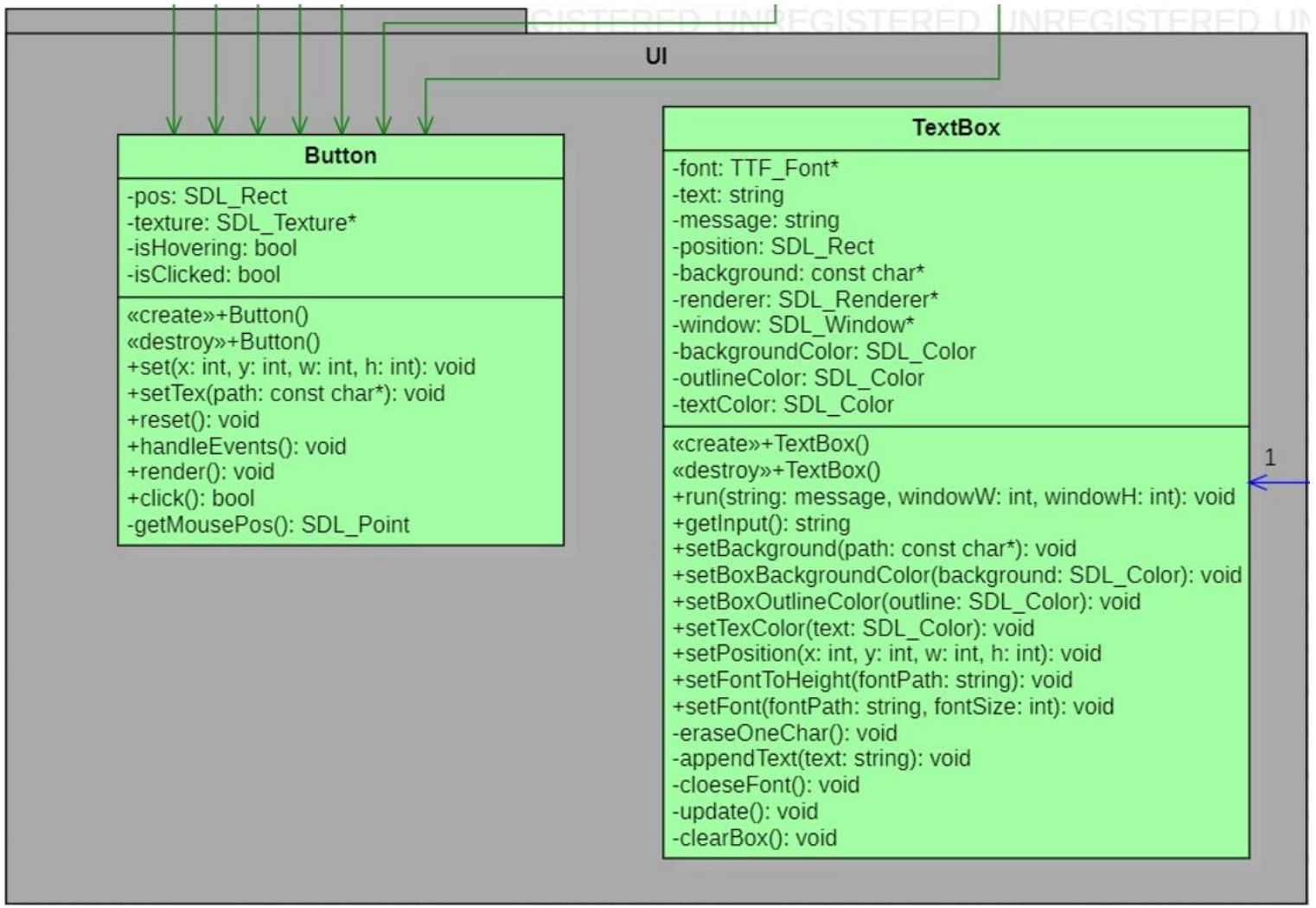
O Menu consiste em diversas subclasses de MenuEnte (derivada de Ente), que por sua vez agregam diferentes Botões com diferentes comportamentos (classe Button do UI).
Um objeto de cada classe derivada de MenuEnte é agregado ao GerenciadorDeMenu, que intermedia a relação dessas classes com a classe jogo (Principal).





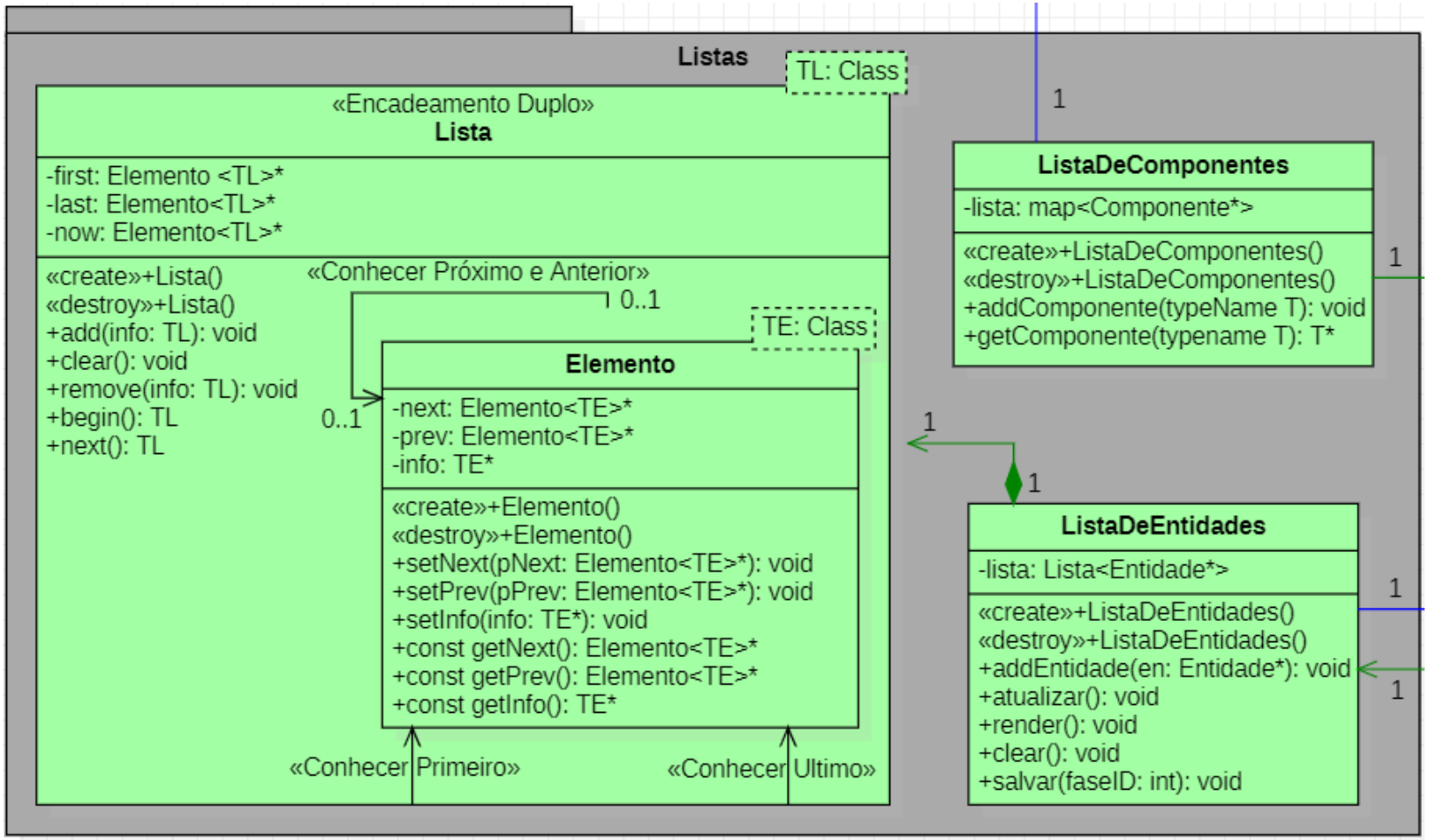
UI

O pacote UI consiste em um conjunto de classes que caracterizam a interface do usuário (User Interface). Contém as classes Button (botão) e TextBox (caixa de texto).



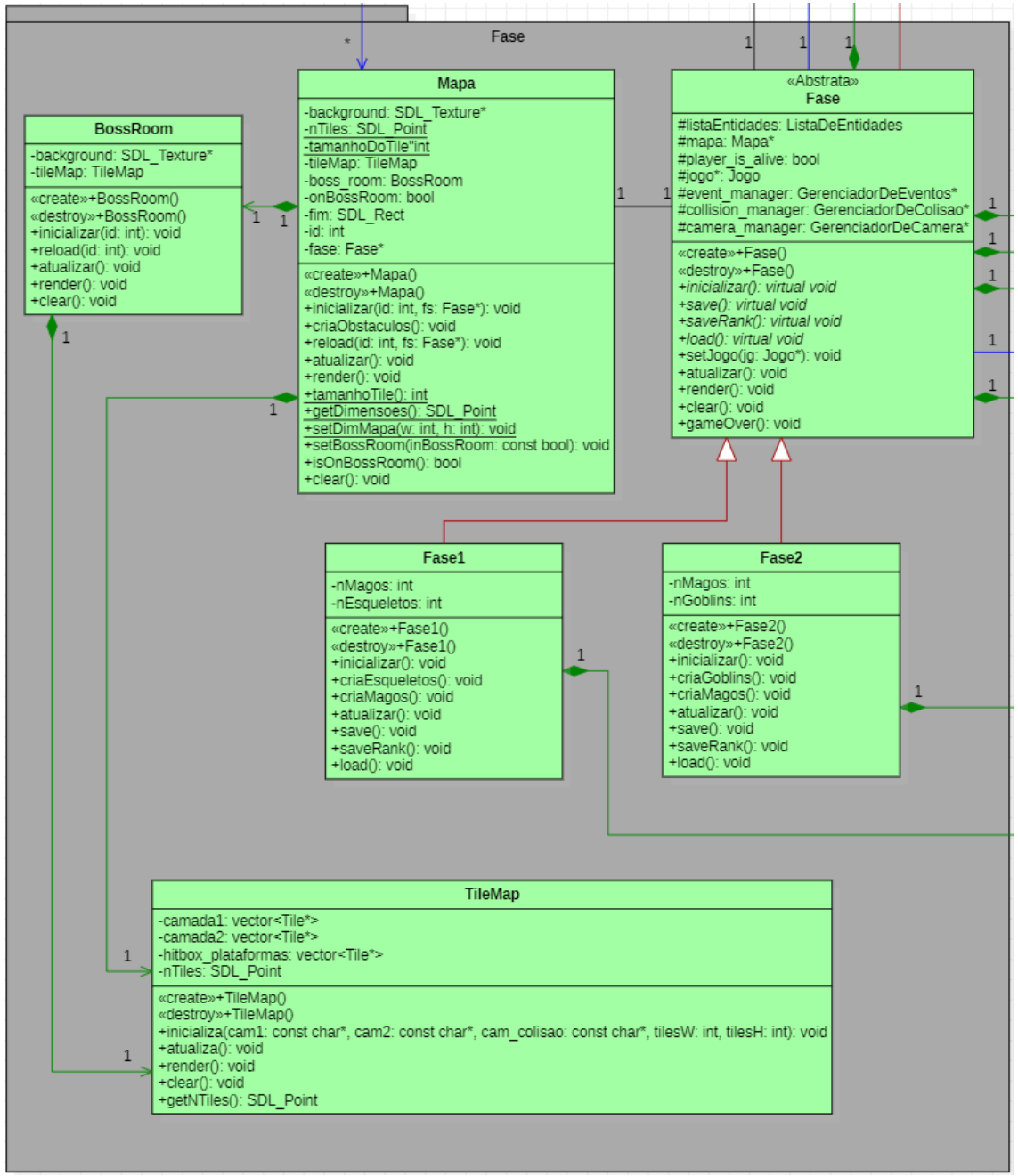
Listas

As listas realizam as relações entre classes e múltiplos objetos com uma classe mãe em comum, gerenciando o comportamento destes objetos.



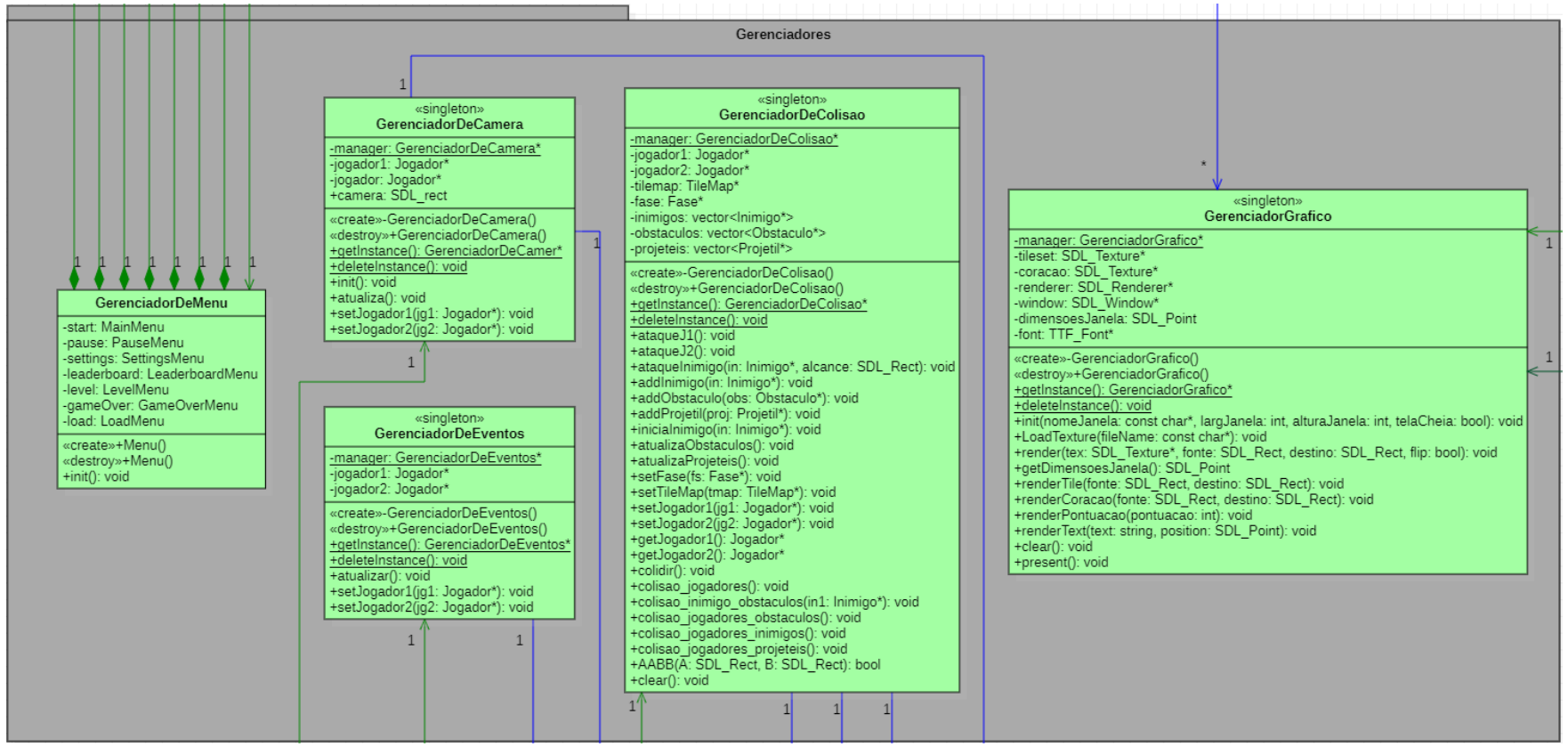
Fase

O pacote das Fases reúne as classes que controlam a geração das diversas fases de um jogo, gerando mapa e instanciando os diversos objetos utilizados dentro dela.



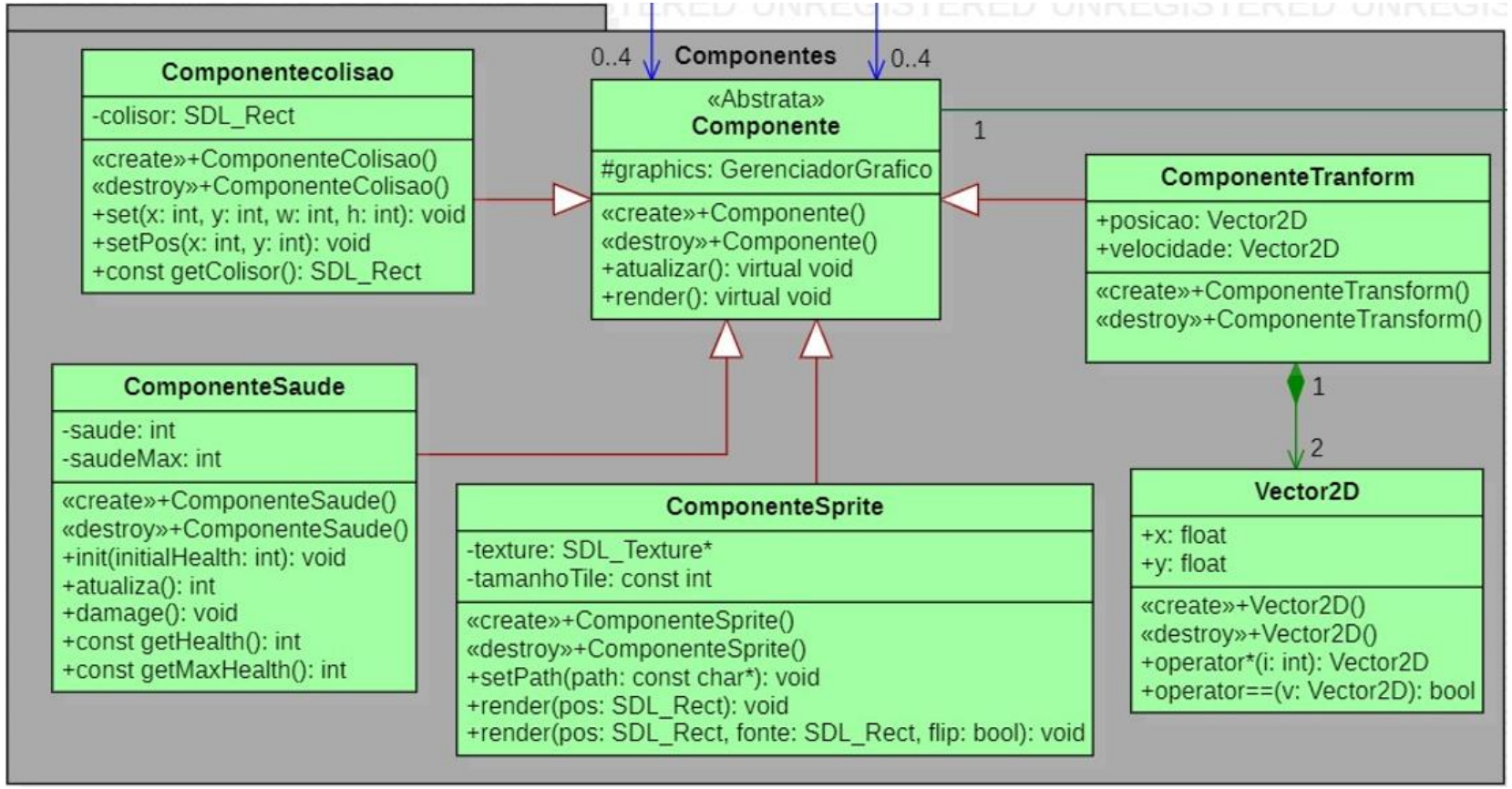
Gerenciadores

Os gerenciadores são responsáveis pelos tratamentos de objetos que, por boa prática de paradigma da Orientação à Objeto, não devem ficar agregadas aos objetos em si. São esses:
-GerenciadorDeMenu
-GerenciadorDeSave
-GerenciadorDeEventos
-GerenciadorDeColisao
-GerenciadorDeCamera
-GerenciadorGráfico



Componentes

Os componentes são diferentes classes que definem o comportamento de um ente, se esse ente tem um componente visual, ele possui um ComponenteSprite, se ele colide, um ComponenteColisao, se ele se movimenta, ComponenteTransform, se ele possui vida, ComponenteSaude.
O Ente conhece os componentes através da uma ListaDeComponentes.



Entidades

As Entidades são os entes que efetivamente aparecem na tela durante a execução dos níveis.
Se dividem em 3:
-Projetil
-Obstaculo (namespace Obstaculos)
-Personagem (namespace Personagens

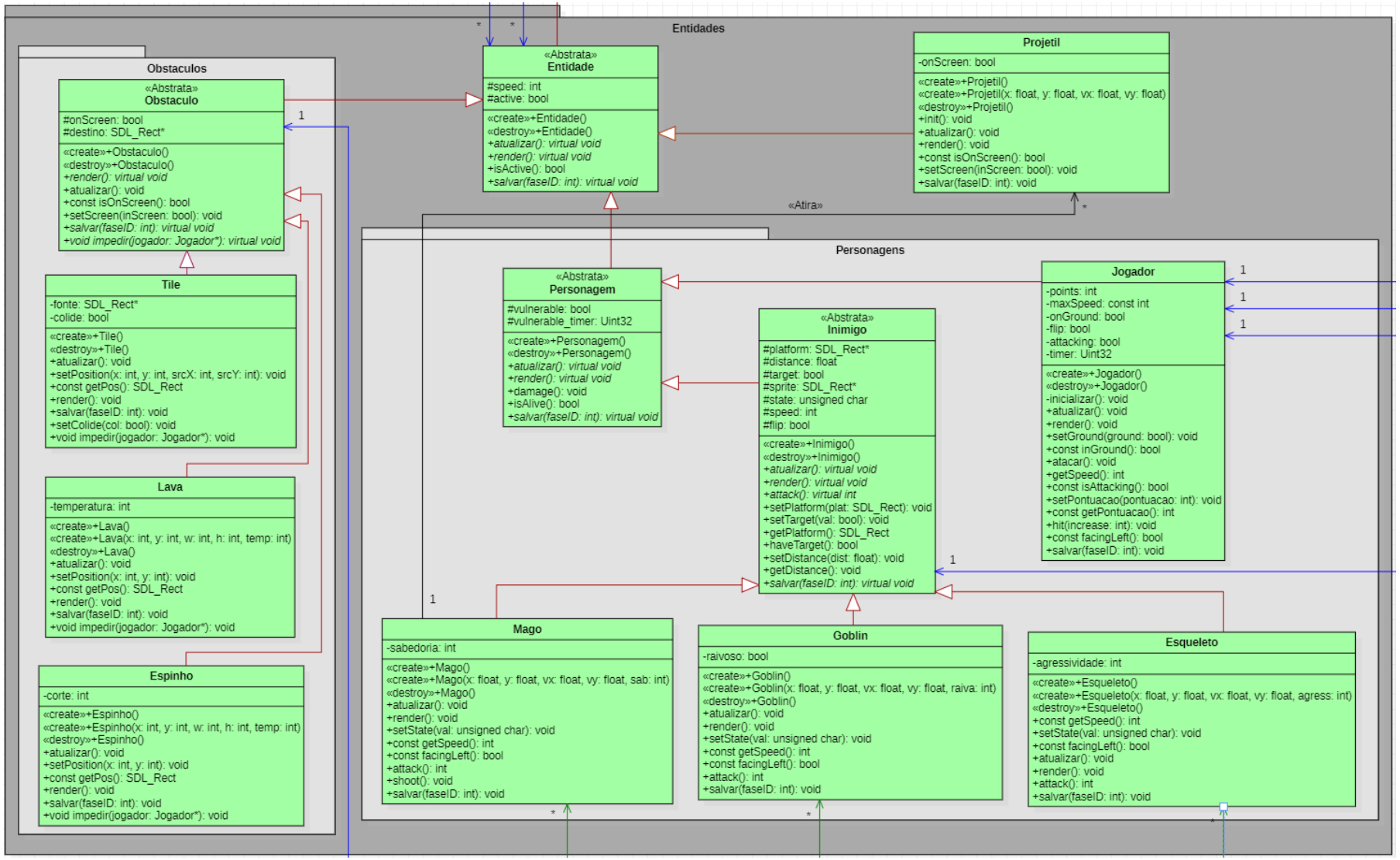


Tabela de Conceitos

N.	Conceitos	Uso	Onde/O quê
1	**Elementares:**		
1.1	- Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno).	Sim	Todos .h e .cpp
1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Maioria dos .h e .cpp
1.3	- Classe Principal	Sim	Main.cpp & Jogo.h/.cpp
1.4	- Divisão em .h e .cpp	Sim	Em todo o desenvolvimento
2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional	Sim	Associações direcionais de Gerenciadores com diversas classes e bidirecionais em Fase/Jogo
2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	Em diversos pontos do desenvolvimento (Gerenciadores, Fases, Listas)
2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	Em toda a estrutura do projeto
2.4	- Herança múltipla	Não	

N.	Conceitos	Uso	Onde/O quê
3	**Ponteiros, generalizações e exceções**		
3.1	- Operador *this* para fins de relacionamento bidirecional.	Sim	Na associação bidirecional entre Fase e Jogo
3.2	- Alocação de memória (*new* & *delete*).	Sim	Em todo o desenvolvimento
3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).	Sim	Template na ListaDeEntidades e ListaDeComponentes
3.4	- Uso de Tratamento de Exceções (*try catch*).	Sim	Na criação de texturas no GerenciadorGrafico
4	**Sobrecarga de:**		
4.1	- Construtoras e Métodos.	Sim	No GerenciadorGrafico e Construtoras de Inimigos
4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).	Sim	Na classe Vector2D. Operadores * e ==
---	**Persistência de Objetos (via arquivo de texto ou binário)**		
4.3	- Persistência de Objetos.	Sim	Salvamento de Entidades
4.4	- Persistência de Relacionamento de Objetos.	Sim	Salvamento de Entidades

Tabela de Conceitos

N.	Conceitos	Uso	Onde/O quê
5	**Virtualidade:**		
5.1	- Métodos Virtuais Usuais	Sim	Nas destrutoras de classes abstratas
5.2	- Polimorfismo	Sim	Em funções atualizar, render dos Entes
5.2	- Métodos Virtuais Puros / Classes Abstratas	Sim	Nas classes abstratas Ente, Fase, MenuEnte, Entidade, Obstaculo, Personagem e Componente
5.3	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto	Sim	Em todo o desenvolvimento
6	**Organizadores e Estáticos**		
6.1	- Espaços de Nomes (*Namespace*) criada pelos autores	Sim	Namespaces Gerenciadores, Menu, Fase, Componentes, UI, Entidades, Obstaculos e Personagens
6.2	- Classes aninhadas (*Nested*) criada pelos autores	Sim	Classe Elemento aninhada à classe Lista
6.3	- Atributos estáticos e métodos estáticos	Sim	Em todos os gerenciadores Singleton.
6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método	Sim	Em todo o desenvolvimento

N.	Conceitos	Uso	Onde/O quê
7	**Standard Template Library (*STL*) e String OO**		
7.1	- A classe Pré-definida *String* ou equivalente. **&** - *Vector* e/ou *List* da *STL* (p/ objetos ou ponteiros de objetos de classes definidos pelos autores)	Sim	Utilizadas String e Vector
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	Utilizado Mapa na leitura das pontuações
---	**Programação concorrente**		
7.3	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time **OU** Win32API ou afins.	Não	
7.4	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.	Não	

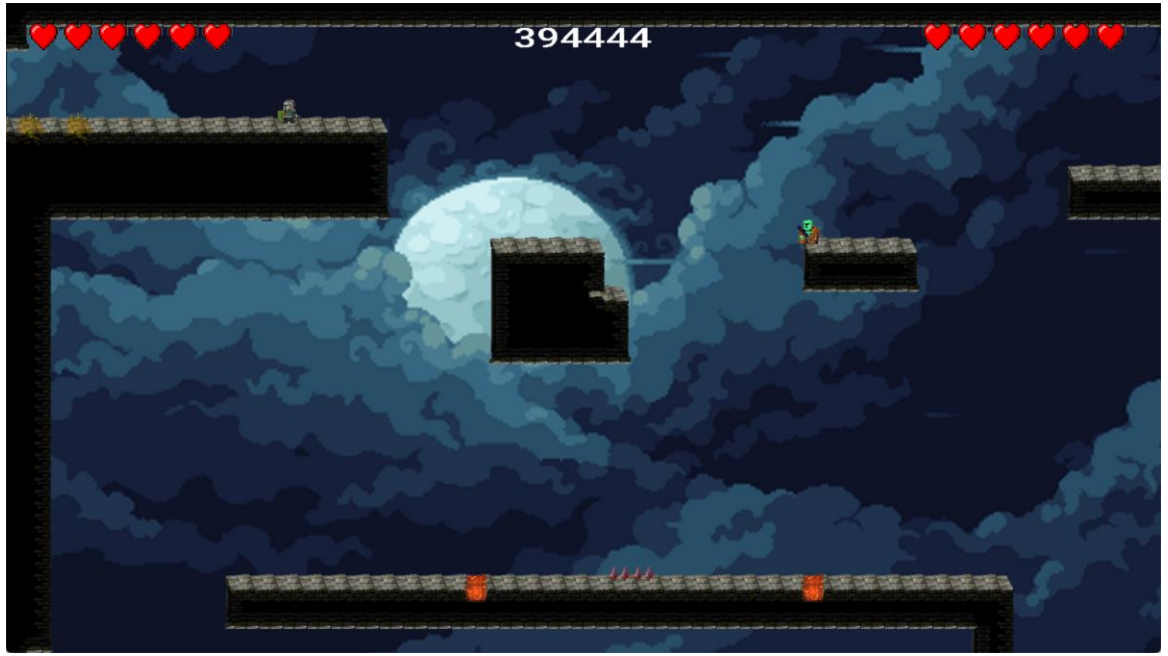
Percentagem concluída: 87,5%

Tabela de Conceitos

N.	Conceitos	Uso	Onde/O quê
8	**Biblioteca Gráfica / Visual**		
8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: tratamento de colisões duplo *buffer*	Sim	Em todo o desenvolvimento, especialmente no GerenciadorGrafico
8.2	- Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive) em algum ambiente gráfico. **OU** *- RAD – Rapid Application Development* (Objetos gráficos como formulários, botões etc).	Sim	No GerenciadorDeEventos
---	**Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.**		
8.3	- Ensino Médio efetivamente	Sim	Conceitos de Movimento Retilíneo Uniformemente Variado (MRUV)
8.4	- Ensino Superior efetivamente	Sim	Tratamento de Colisões Elásticas (Física 1)

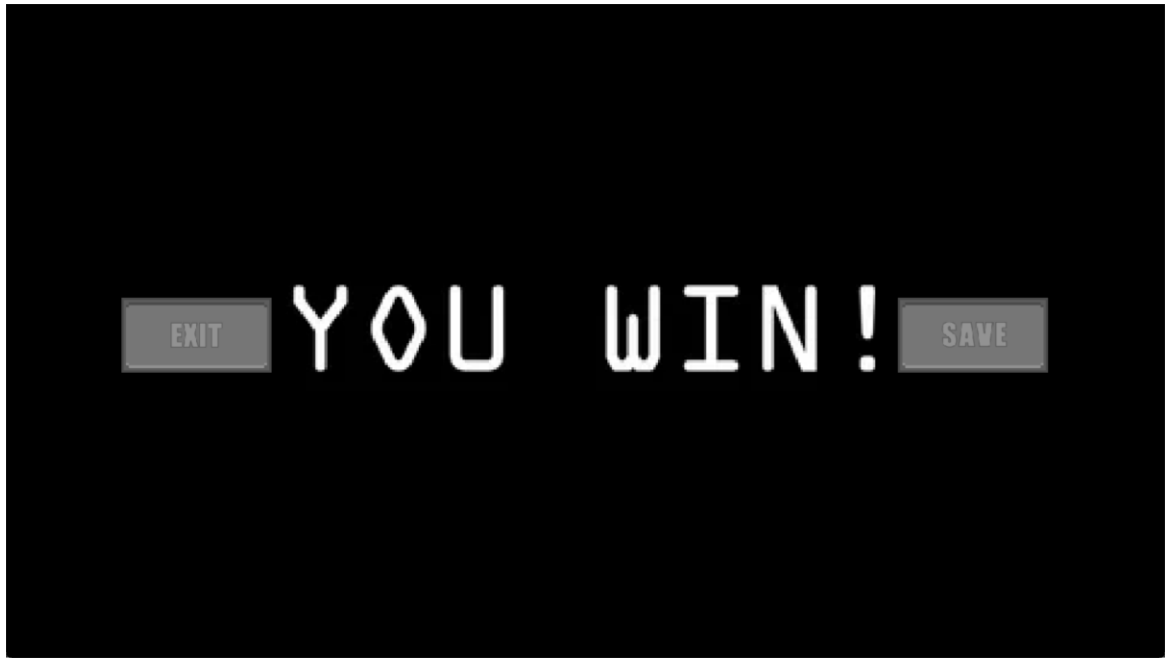
N.	Conceitos	Uso	Onde/O quê
9	**Engenharia de Software**		
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos	Sim	Em todo o desenvolvimento
9.2	- Diagrama de Classes em UML.	Sim	Em todo o desenvolvimento
9.3	- Uso efetivo e intensivo de padrões de projeto *GOF, i.e.,* mais de 5 padrões	Não	Singleton e Template Method
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes	Sim	Em todo o desenvolvimento
10	**Relações de:**		
10.1	- Controle de versão de modelos e códigos automatizado (via github e/ou afins). & - Uso de alguma forma de cópia de segurança (*i.e.*, *backup*).	Sim	Em todo o desenvolvimento (github)
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Realizadas 4 reuniões agendadas
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Não	
10.4	- Revisão do trabalho escrito de outra equipe e vice-versa..	Sim	João André e Alan

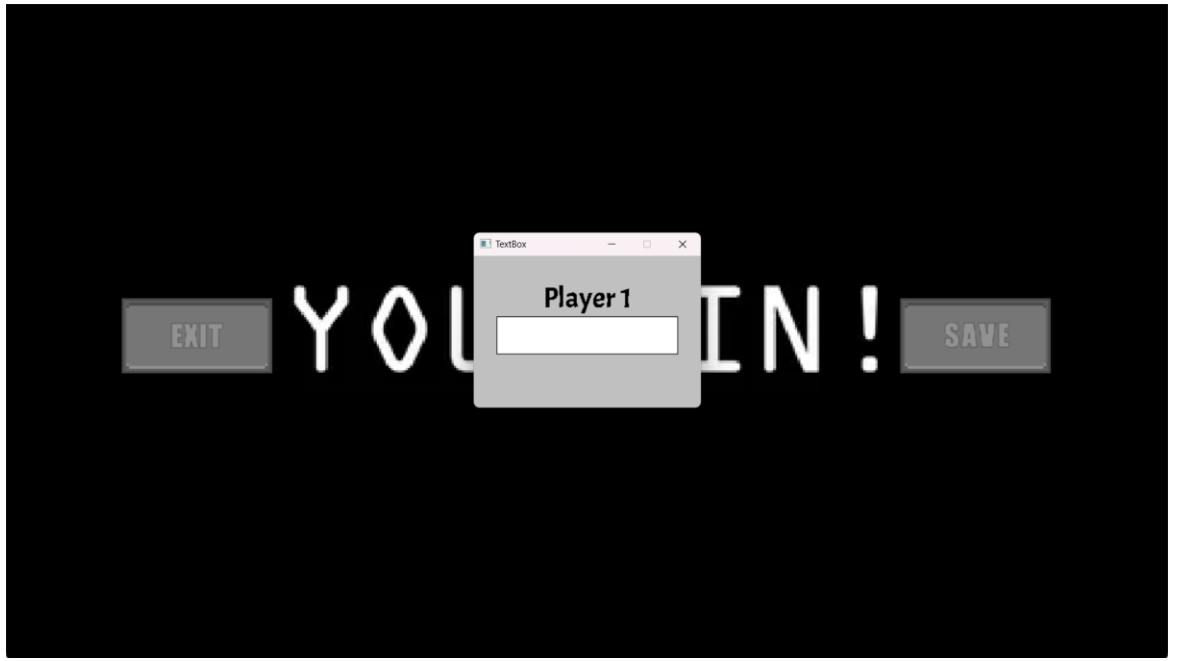
Fase 1



Fase 2

Salvamento e Ranking





Conclusão

Com o desenvolvimento deste projeto, foi possível adquirir contato e familiaridade com o paradigma da orientação a objetos, além da experiência com ferramentas de sincronização de código (Git/Github), engenharia de software (desenvolvimento de diagrama em UML), utilização de bibliotecas gráficas para C/C++, dentre outros.

Sobretudo, a experiência do desenvolvimento de projeto supervisionado com base em modelo é primordial para a formação de um profissional.

Desse modo, o esforço e tempo gastos, seja efetivamente codificando, analisando requisitos, entregando resultados constantemente em reuniões, dentre outros, é o que caracteriza uma rotina de profissionais dequalidade.