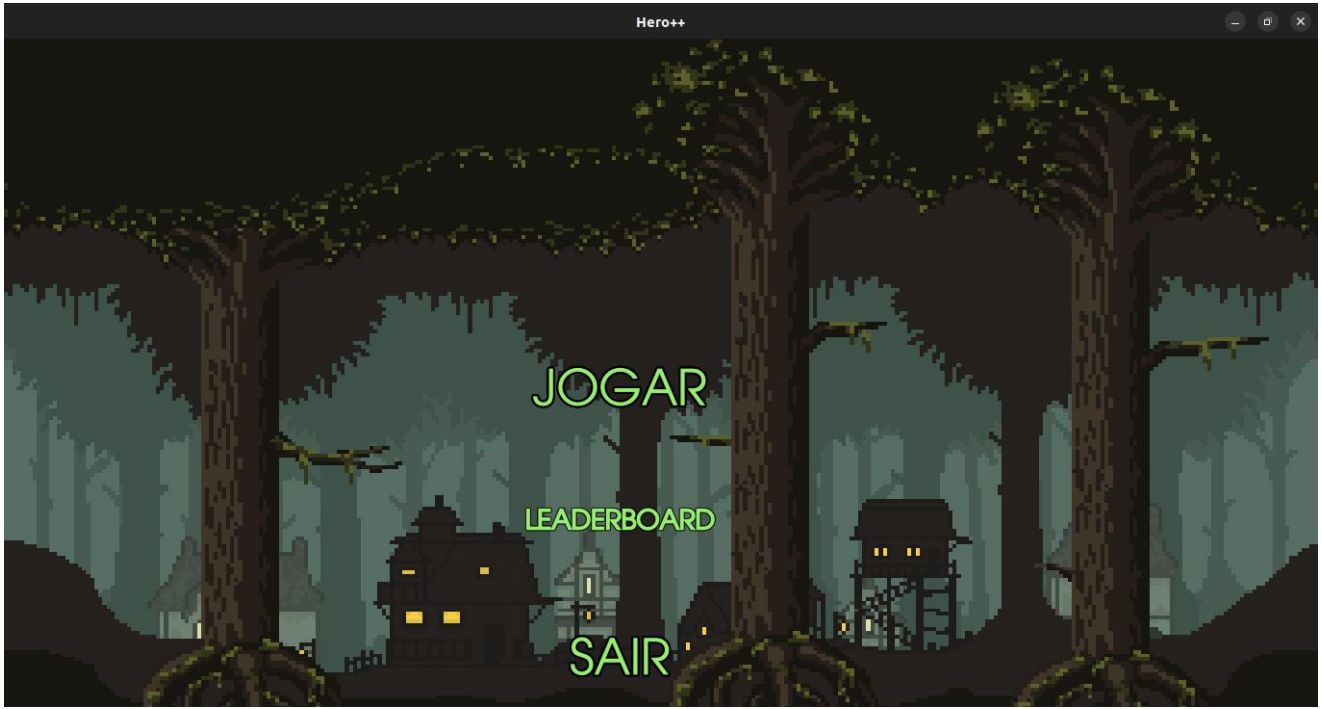
HERO++

Alunos: Ian Ishikawa, Pedro Neves
Prof. Dr. Jean M. Simão
Disciplina Técnicas de Programação - S71

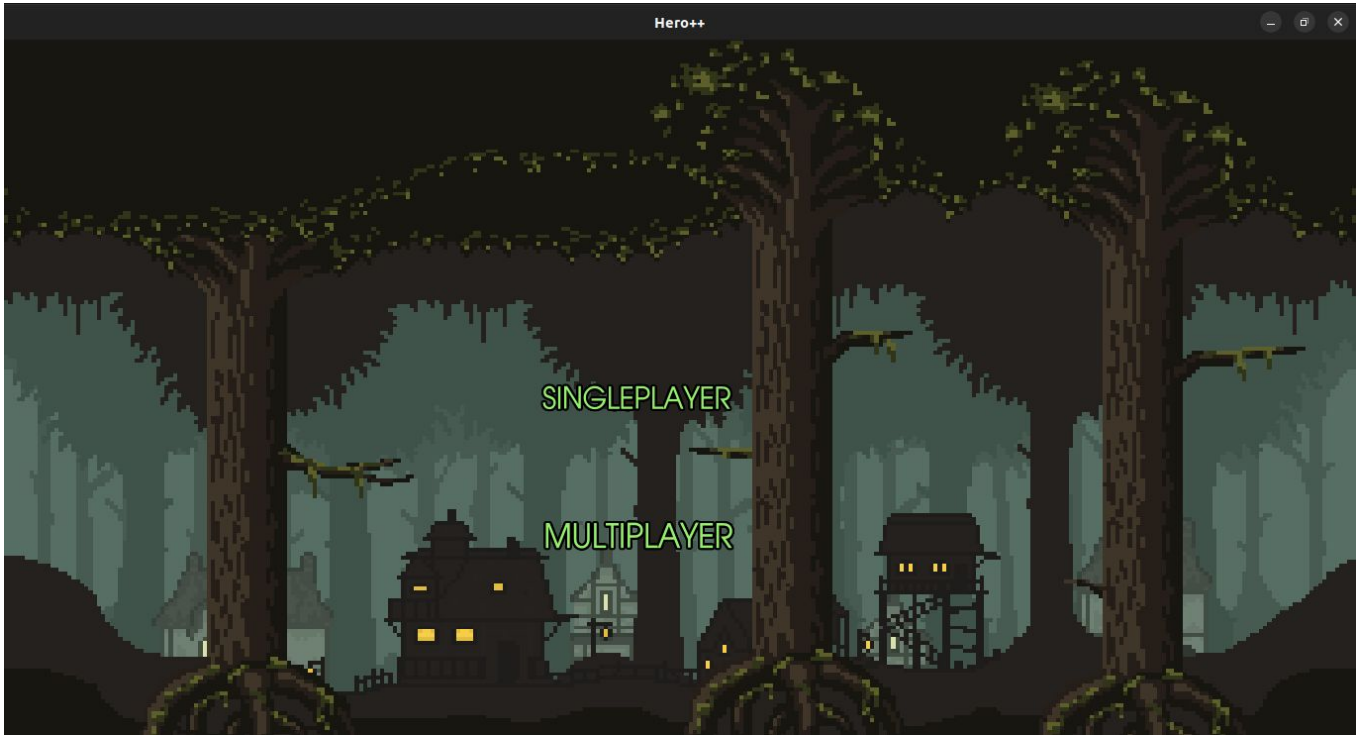
1 - Objetivos

- Prática de Programação Orientada Objetos
- Engenharia de Software
- Padrões de Projeto
- Trabalho em equipe
- Tratamento de requisitos
- Versionamento de Código

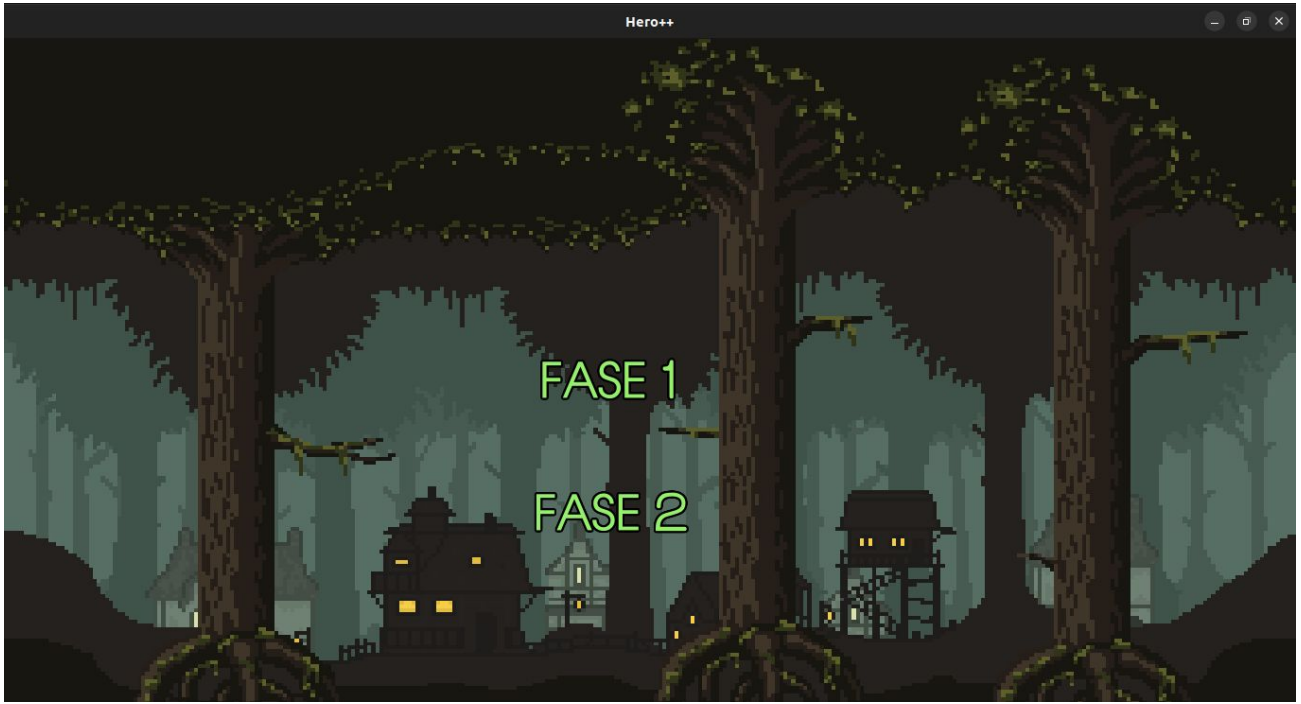
2- Menu Inicial



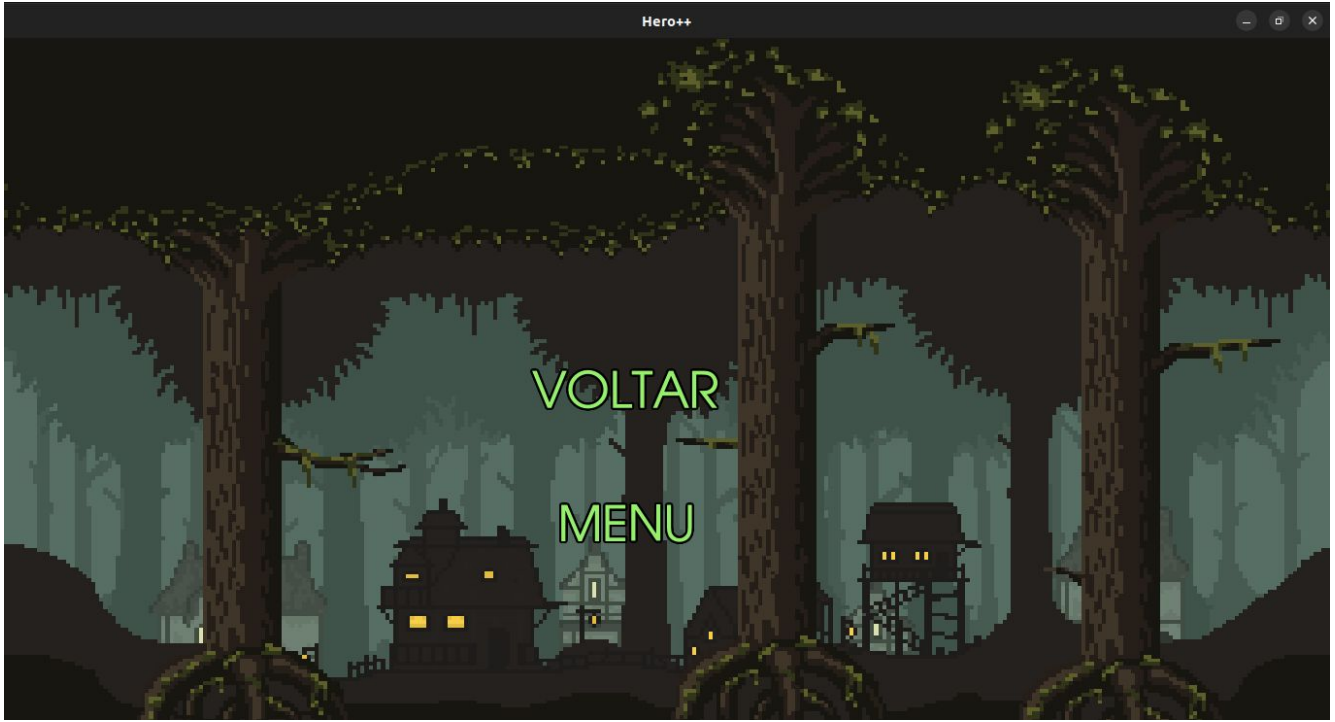
3 - Segundo Menu



4 - Terceiro Menu



5 - Menu Pause



6 - Fase Bosque

7 - Fase Castelo

8 - Tabela de Requisitos

N.	Requisitos Funcionais	Situação	Implementação
1	Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, ver colocação (*ranking*) de jogadores e demais opções pertinentes.	Requisito previsto inicialmente e cumprido	Requisito cumprido via classe Menu e seu respectivo objeto, o qual é agregado pela classe Jogo. Requisito cumprido com auxílio da SFML.
2	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.	Requisito previsto inicialmente e cumprido na totalidade	Requisito cumprido inclusive via classe Jogador cujos objetos são agregados em jogo, podendo ser apenas um jogador.
3	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.	Requisito previsto inicialmente e realizado na totalidade	Requisitos realizado nas classes Bosque, Castelo e Fase, sendo que a classe Fase é herdada por Bosque e Castelo.
4	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'.	Requisito previsto inicialmente e cumprido na totalidade.	Requisito cumprido por meio da classe Inimigo, Mago, Esqueleto e Boss, sendo que Mago, Esqueleto e Boss herdam a classe base Inimigo e o Mago consegue lançar um projétil.

9 - Tabela de Requisitos

5	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.	Requisito previsto inicialmente e cumprido na totalidade.	Requisito cumprido na classe Bosque e Castelo, que criam quantidades aleatórias de inimigos, sendo no mínimo 3.
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem.	Requisito previsto inicialmente e cumprido na totalidade.	Requisito cumprido por meio das classes Obstaculo, Plataforma, Caixa e Lava, sendo que Plataforma, Caixa e Lava herdam obstáculo e a Lava pode causar dano no jogador.
7	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.	Requisito previsto inicialmente e realizado.	Requisito cumprido na classe Bosque e Castelo, que instanciam quantidades aleatórias de Caixas e Lavas, sendo no mínimo 3 instancias de cada.
8	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.	Requisito previsto inicialmente e cumprido na totalidade.	Requisito cumprido na classe Plataforma, que permite Inimigos e Jogadores subirem sobre a mesma.

10 - Tabela de Requisitos

9	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.	Requisito previsto inicialmente e cumprido na totalidade	Requisito cumprido por meio da classe Gerenciador de Colisões e de métodos próprios das classes Personagens e Obstaculos.
10	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.	Requisito previsto e **NÃO** realizado.	Requisito NÃO realizado.

Total de requisitos funcionais apropriadamente realizados. *(Cada tópico vale 10%, sendo que para ser contabilizado deve estar realizado efetivamente e não parcialmente)*	**90%** (setenta por cento).

11 - Tabela de Conceitos

Tabela 2. Lista de Conceitos Utilizados e Não Utilizados no Trabalho.

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. Como na classe Mago.
1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Na maioria dos .h e .cpp, como na classe Personagem .
1.3	- Classe Principal.	Sim	Main.cpp & Jogo.h/cpp.
1.4	- Divisão em .h e .cpp.	Sim	Em todas as classes, com exceção da classe Lista, que possui somente .h.
2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional.	Sim	Nas classes Menu e Jogo.
2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	Em alguns dos .h e .cpp, como na classe Mago, nos atributos Projetil e ListaObst.
2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	Em alguns dos .h e .cpp, como nas classes Entidades, que herda Ente e na classe Mago, que herda classes em diversos niveis.
2.4	- Herança múltipla.	Não	Precisamente nos .h e .cpp, das classes C, D e F.

12 - Tabela de Conceitos

	3	**Ponteiros, generalizações e exceções**		
	3.1	- Operador *this* para fins de relacionamento bidirecional.	Sim	Precisamente nos .h e .cpp, da classe Jogo, em sua construtora.
	3.2	- Alocação de memória (*new* & *delete*).	Sim	Em diversos .h e .cpp, como na destrutora da classe Gerenciador de Colisões e no método Criar Mago da classe Fase.
	3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).	Sim	Especificamente no arquivo Lista.h.
	3.4	- Uso de Tratamento de Exceções (*try catch*).	Sim	Especificamente nas classes Bosque e Castelo.
	4	**Sobrecarga de:**		
	4.1	- Construtoras e Métodos.	Sim	Especificamente no método setGameState da classe Jogo e na contrutora da classe Mago.
	4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).	Sim	Foi usado o operator[] e o operador= na ListaEntidades.cpp.
	---	**Persistência de Objetos (via arquivo de texto ou binário)**		
	4.3	- Persistência de Objetos.	Não	. . .
	4.4	- Persistência de Relacionamento de Objetos.	Não	. . .

13 - Tabela de Conceitos

	5	**Virtualidade:**		. . .
	5.1	- Métodos Virtuais Usuais.	Sim	Especificamente no Inimigo.cpp
	5.2	- Polimorfismo.	Sim	Nas classes Jogador e Inimigo especificamente no método atualizar.
	5.3	- Métodos Virtuais Puros / Classes Abstratas.	Sim	Na classe Entidade, especificamente no método atualizar.
	5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.	Não	Foi feito o uso de coesão e desacoplamento no gerenciador gráfico, porém sem apoio de padrões de projeto.
	6	**Organizadores e Estáticos**		
	6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Especificamente na classe Mago.
	6.2	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Na Lista.h foi feito o aninhamento da classe Elemento.
	6.3	- Atributos estáticos e métodos estáticos.	Sim	Especificamente no Gerenciador_Gráfico.
	6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	Especificamente na Personagem.h.

14 - Tabela de Requisitos

	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	Para salvar o caminho das texturas foi utilizado const char* na classe Gerenciador_Gráfico. Vector foi utilizado para salvar o ranking dos jogadores, na classe Menu_Leaderboard.
	7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	Foi usado o Mapa na Classe Gerenciador Grafico.
	---	**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*	. . .

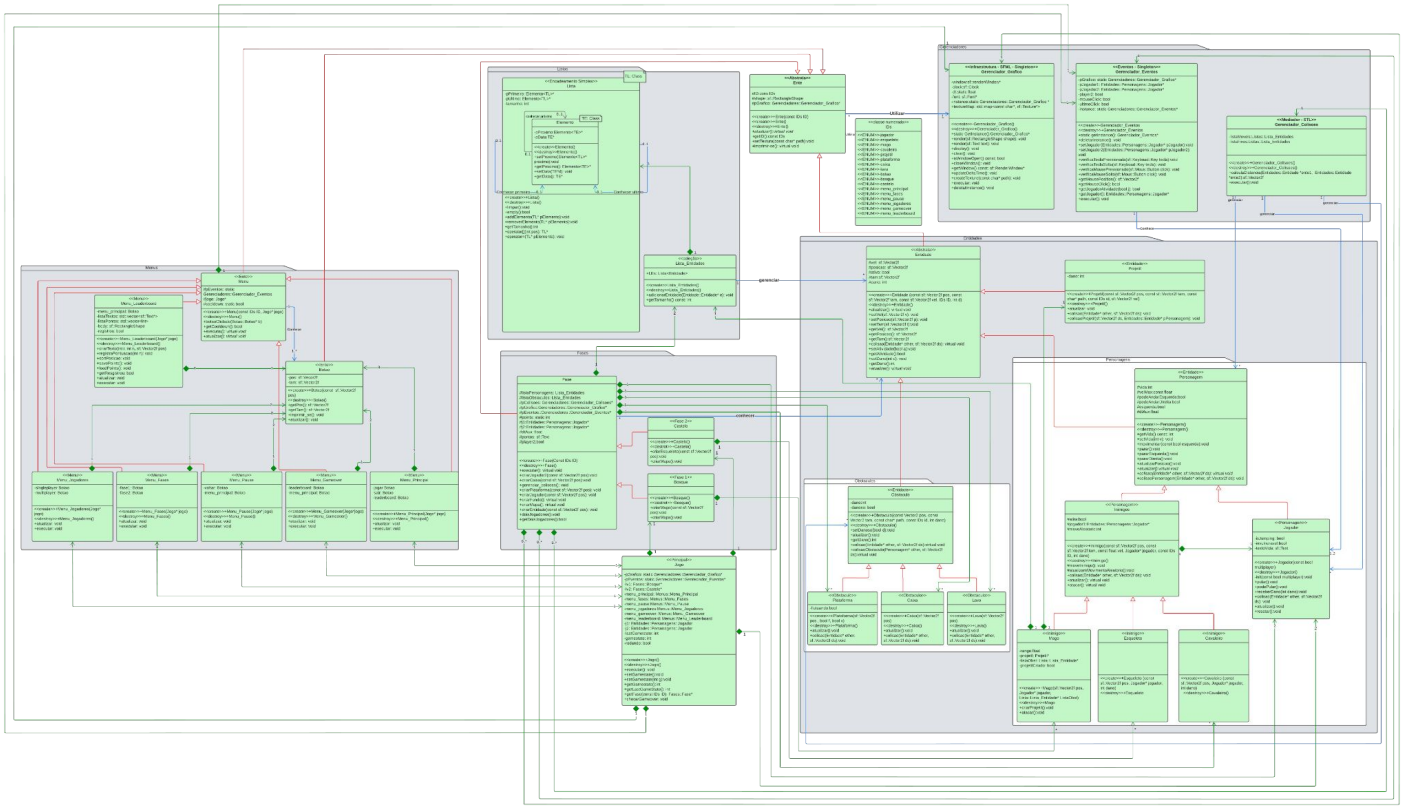
15 - Tabela de Requisitos

	8	**Biblioteca Gráfica / Visual**		
	8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: • tratamento de colisões • duplo *buffer*	Sim	Tratamento da movimentação e de Colisão na Gerenciador_Colisoes e na Gerenciador_Eventos.
	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	Na classe Gerenciador_Eventos.
	---	**Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.**		
	8.3	- Ensino Médio Efetivamente.	*Sim*	Torricelli para tratar a gravidade em diversos .cpp, como o Caixa.cpp.
	8.4	- Ensino Superior Efetivamente.	*Sim*	Efeito magnus para fazer o projétil no Projetil.cpp.
	9	**Engenharia de Software**		
	9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	Realizado no início e fim do projeto.
	9.2	- Diagrama de Classes em *UML*.	Sim	Atualizado durante todo o andamento do projeto.
	9.3	- Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões.	Parcial	Foi usado somente o padrão de projeto Singleton.

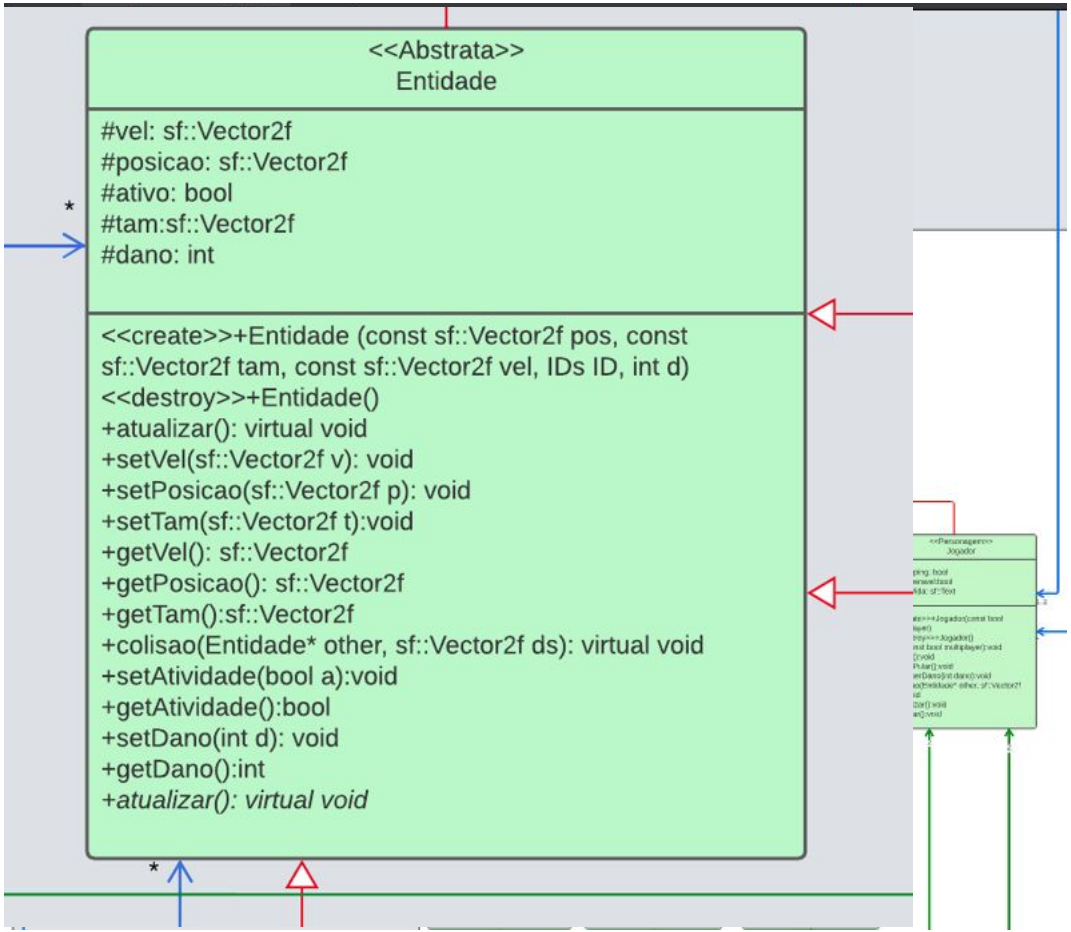
16- Tabela de Requisitos

	9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	Durante todo o projeto.
	10	**Execução de Projeto**		
	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*	Por meio do github.
	10.2	- Reuniões com o professor para acompanhamento do andamento do projeto.	*Sim*	4 reuniões -3/11 -7/11 -17/11 -24/11
	10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	*parcial*	*5 reuniões* *-28/09* *-28/10* *-28/10* *-4/11* *-16/11*
	10.4	- Revisão do trabalho escrito de outra equipe e vice-versa.	*Sim*	Vitor Errera e Thiago Seiji
	Total de conceitos apropriadamente utilizados. *(Cada grande tópico vale 10% do total de conceitos. Assim, por exemplo, caso se tenha feito metade de um tópico, então valeria 5%.)*			**80%** (oitenta porcento).

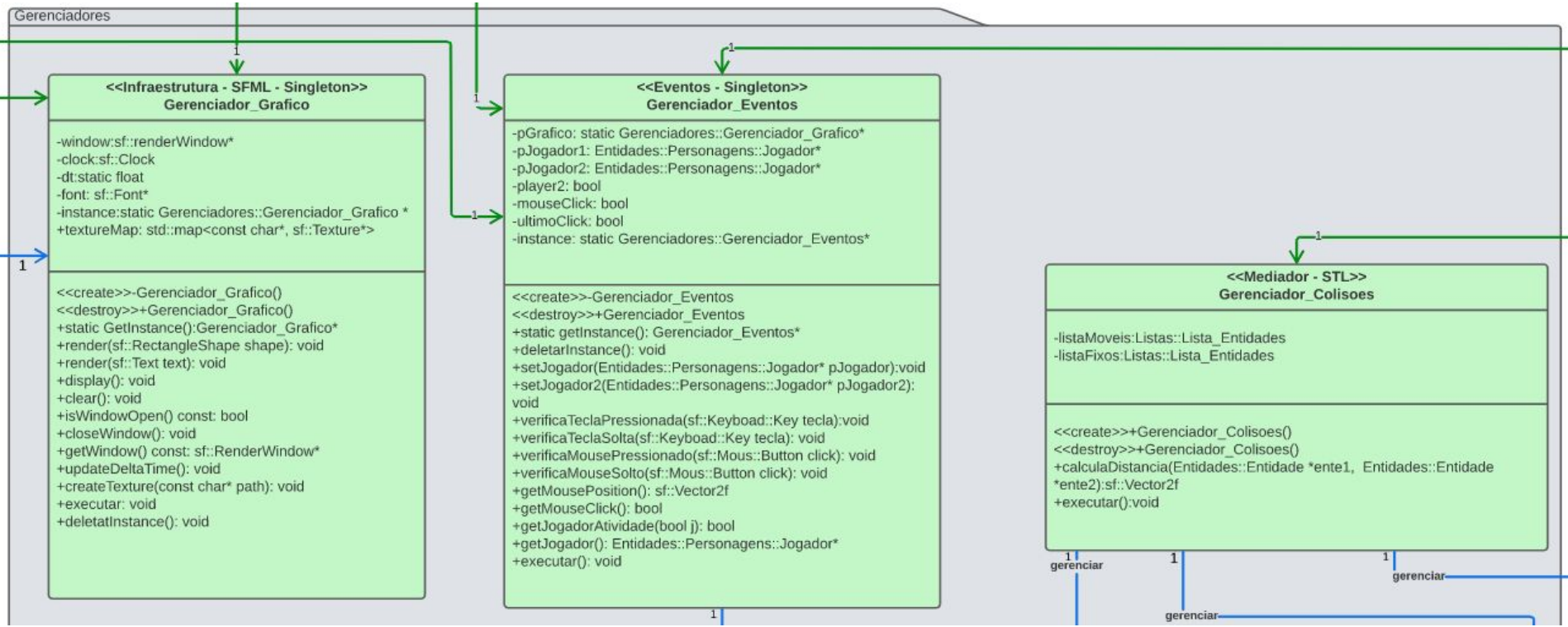
17 - Diagrama UML



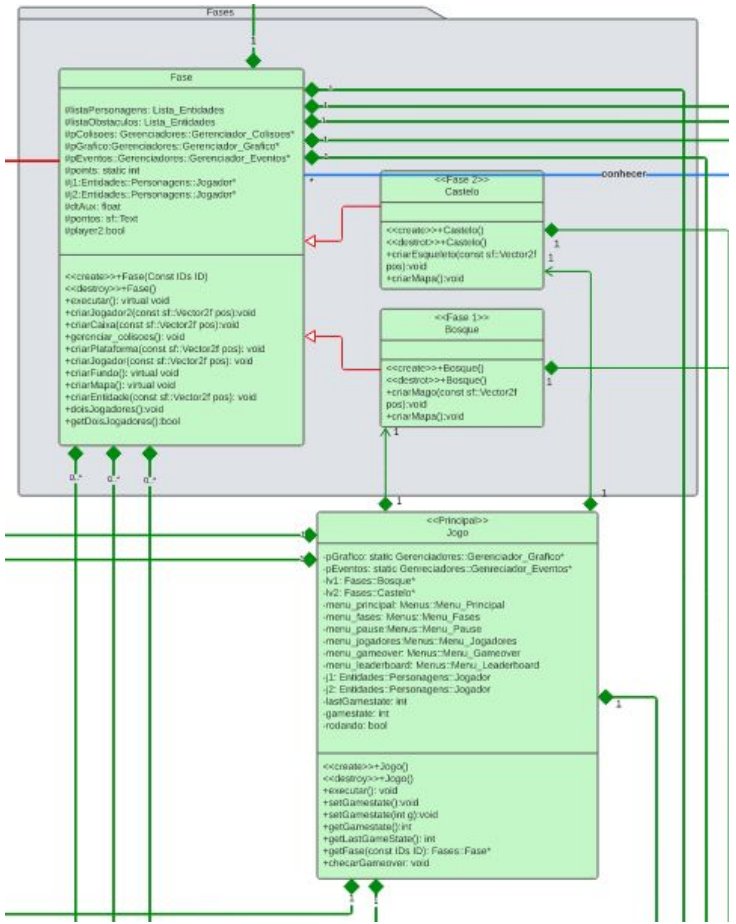
18 - Entidades



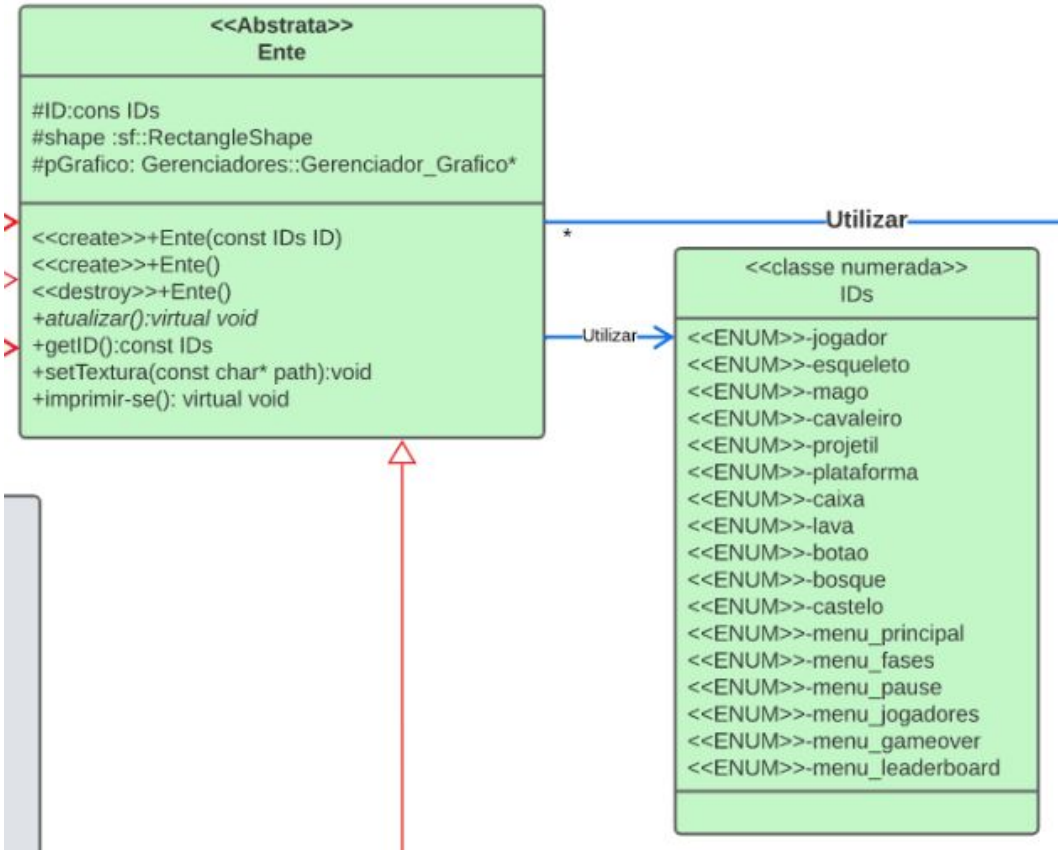
19 - Gerenciadores



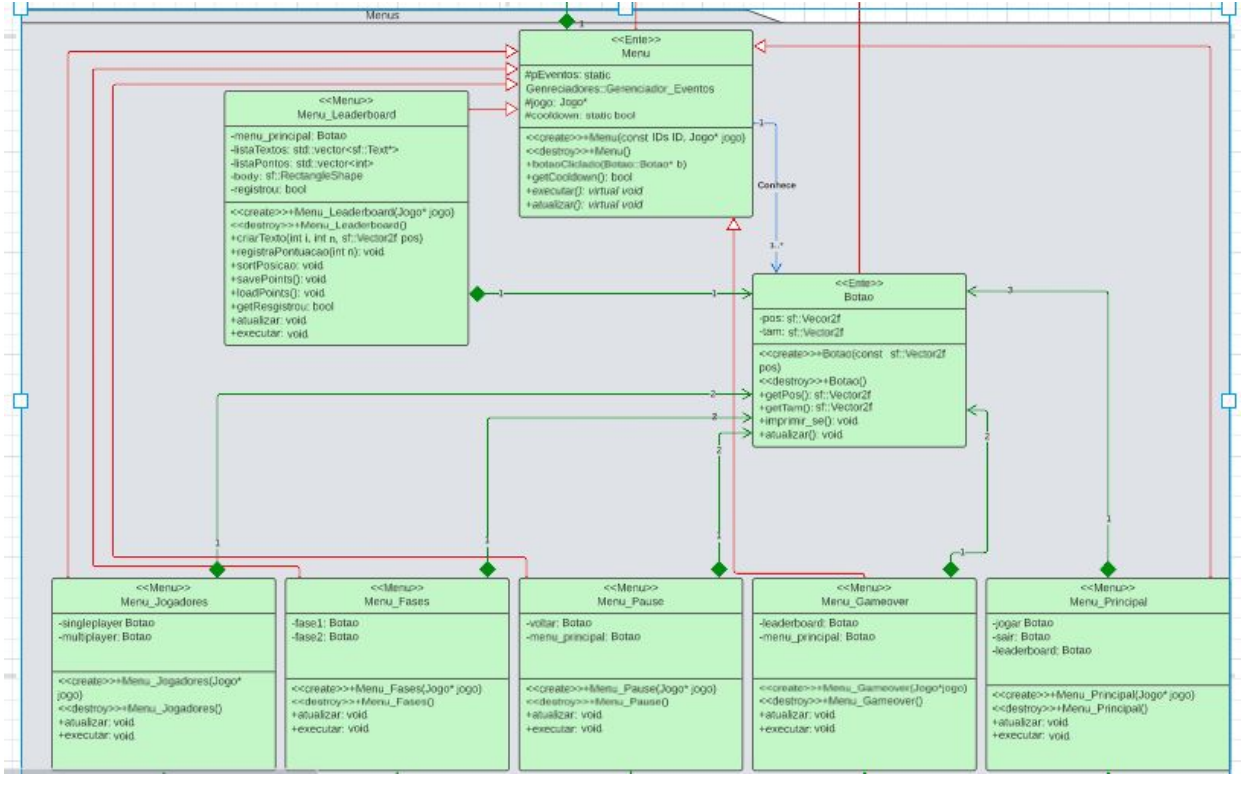
20 - Fase e Jogo



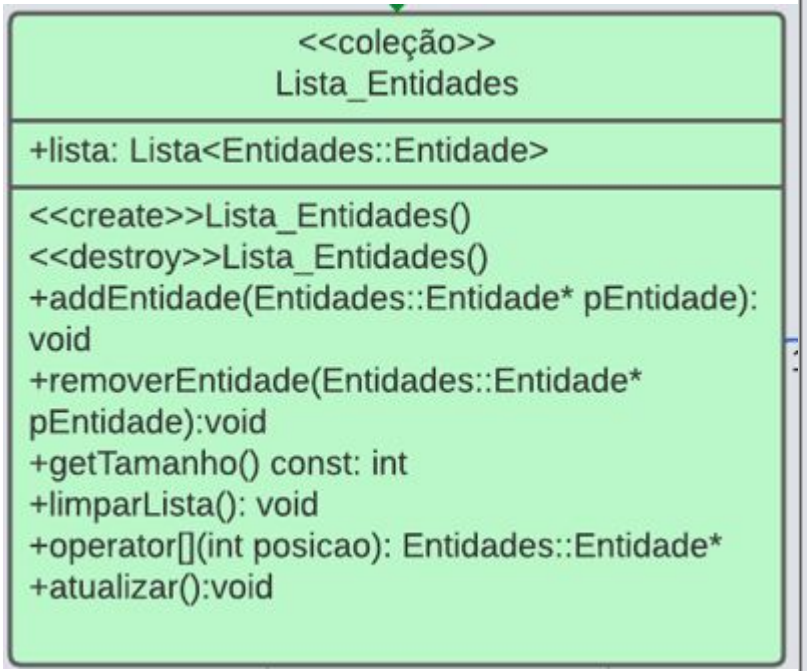
21 - Ente e IDs

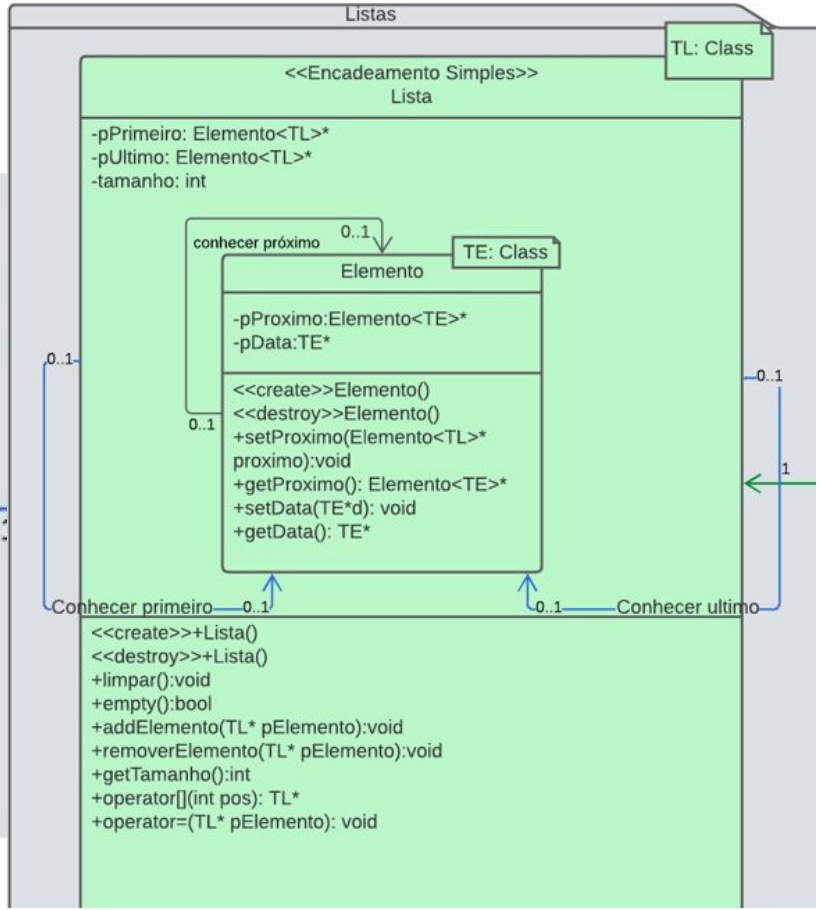

<<Abstrata>>
Ente
#ID:cons IDs
#shape :sf::RectangleShape
#pGrafico: Gerenciadores::Gerenciador_Grafico*
<<create>>+Ente(const IDs ID)
<<create>>+Ente()
<<destroy>>+Ente()
+atualizar():virtual void
+getID():const IDs
+setTextura(const char* path):void
+imprimir-se(): virtual void
Utilizar
*
Utilizar
<<classe numerada>>
IDs
<<ENUM>>-jogador
<<ENUM>>-esqueleto
<<ENUM>>-mago
<<ENUM>>-cavaleiro
<<ENUM>>-projetil
<<ENUM>>-plataforma
<<ENUM>>-caixa
<<ENUM>>-lava
<<ENUM>>-botao
<<ENUM>>-bosque
<<ENUM>>-castelo
<<ENUM>>-menu_principal
<<ENUM>>-menu_fases
<<ENUM>>-menu_pause
<<ENUM>>-menu_jogadores
<<ENUM>>-menu_gameover
<<ENUM>>-menu_leaderboard

22 - Menus


Menus
<<Ente>>
Menu
#pEventos: static
Genreciadores::Gerenciador_Eventos
#jogo: Jogo*
#cooldown: static bool
<<create>>+Menu(const IDs ID, Jogo* jogo)
<<destroy>>+Menu()
+botaoClicIado(Botao::Botao* b)
+getCooldown(): bool
+executar(): virtual void
+atualizar(): virtual void
Conhece
<<Menu>>
Menu_Leaderboard
-menu_principal: Botao
-listaTextos: std::vector<sf::Text*>
-listaPontos: std::vector<int>
-body: sf::RectangleShape
-registrou: bool
<<create>>+Menu_Leaderboard(Jogo* jogo)
<<destroy>>+Menu_Leaderboard()
+criarTexto(int i, int n, sf::Vector2f pos)
+registraPontuacao(int n): void
+sortPosicao: void
+savePoints(): void
+loadPoints(): void
+getRegistrou: bool
+atualizar: void
+executar: void
<<Ente>>
Botao
-pos: sf::Vecor2f
-tam: sf::Vector2f
<<create>>+Botao(const sf::Vector2f pos)
<<destroy>>+Botao()
+getPos(): sf::Vector2f
+gerTam(): sf::Vector2f
+imprimir_se(): void
+atualizar(): void
<<Menu>>
Menu_Jogadores
-singleplayer: Botao
-multiplayer: Botao
<<create>>+Menu_Jogadores(Jogo* jogo)
<<destroy>>+Menu_Jogadores()
+atualizar: void
+executar: void
<<Menu>>
Menu_Fases
-fase1: Botao
-fase2: Botao
<<create>>+Menu_Fases(Jogo* jogo)
<<destroy>>+Menu_Fases()
+atualizar: void
+executar: void
<<Menu>>
Menu_Pause
-voltar: Botao
-menu_principal: Botao
<<create>>+Menu_Pause(Jogo* jogo)
<<destroy>>+Menu_Pause()
+atualizar: void
+executar: void
<<Menu>>
Menu_Gameover
-leaderboard: Botao
-menu_principal: Botao
<<create>>+Menu_Gameover(Jogo*jogo)
<<destroy>>+Menu_Gameover()
+atualizar: void
+executar: void
<<Menu>>
Menu_Principal
-jogar Botao
-sair: Botao
-leaderboard: Botao
<<create>>+Menu_Principal(Jogo* jogo)
<<destroy>>+Menu_Principal()
+atualizar: void
+executar: void

23 - Lista de Entidades





24- Conclusão

- Resultado Positivo
  - Engenharia de Software
  - Versionamento de código
  - Prática de POO
- Resultado Negativo
  - Padrões de Projeto
  - Trabalho em equipe
  - Tratamento de requisitos