

MAGIC GOLENS

Disciplina : Técnicas de programação

Felipe Augusto Lee e Henrique Romaniuk Ramalho

Turma S71

Curitiba
2021



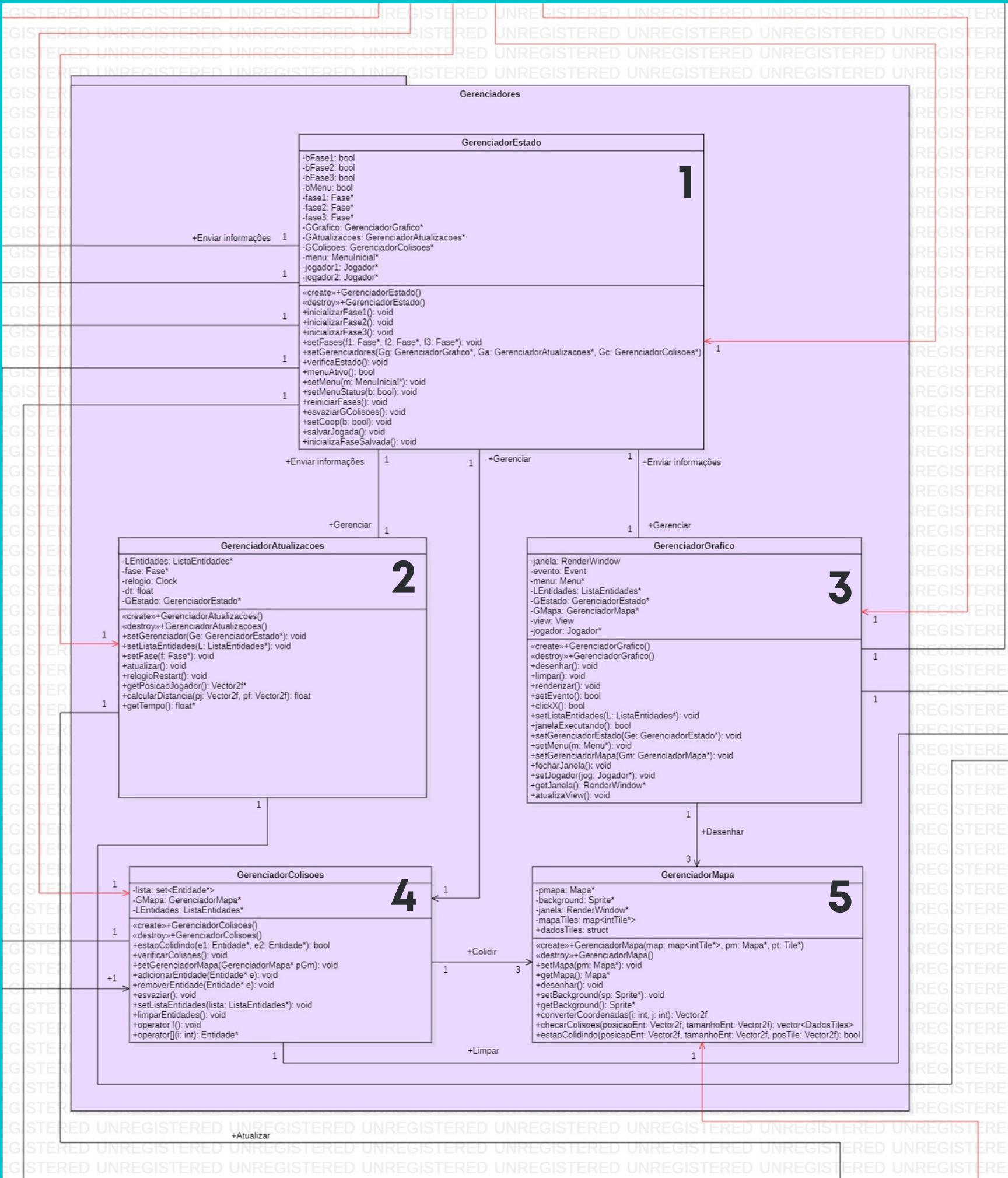
Tabela de requisitos

N.	Requisitos Funcionais	Situação	Implementação	fases.
1	Apresentar menu de opções aos usuários do Jogo.	Requisito previsto inicialmente e realizado.	Requisito cumprido na classe Menu e em suas classes herdadas: MenuFases, MenuInicial, MenuPause, MenuJogadores...	Requisito cumprido inclusive via classes Espinhos, Areia, Fogo e Estalactite, cada um instanciado nas fases, sendo o último instanciado aleatoriamente quanto número e posição.
2	Permitir um ou dois jogadores aos usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante.	Requisito previsto inicialmente e realizado.	Requisito cumprido com duas classes: Mago e Anjo e seus devidos objetos. No caso de um só jogador, é utilizado somente o Mago.	Requisito cumprido inclusive via funções da biblioteca gráfica SFML, que nos permitiu representar graficamente /em uma janela nossos devidos objetos.
3	Disponibilizar ao menos duas fases que podem ser jogadas sequencialmente ou selecionadas.	Requisito previsto inicialmente e realizado.	Requisito cumprido nas classes FasePedra, FaseFogo e FaseGelo, na respectiva sequência.	Requisito cumprido inclusive via a classe GerenciadorMapa, Mapa e Tile.
4	Ter três tipos distintos de inimigos (o que pode incluir ‘Chefão’, vide abaixo), sendo que pelo menos um dos inimigos deve ser capaz de lançar projétil contra o(s) jogador(es).	Requisito previsto inicialmente e realizado.	Requisito cumprido nas classes GolemFogo, GolemPedra, GolemGelo, herdadas da classe Inimigo, além da classe Chefao. O GolemGelo também é herdado da classe Atirador, o que significa que é capaz de lançar projéteis.	Requisito cumprido inclusive via classe GerenciadorColisoes.
5	Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias e sendo pelo menos 5 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via objetos das classes GolemPedra, GolemFogo e GolemGelo ao serem instanciados em números aleatórios.	Requisito cumprido inclusive via classe GerenciadorColisoes.
6	Ter inimigo “Chefão” na última fase	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via um objeto da classe Chefao.	Requisito cumprido inclusive via classe Leaderboard.
7	Ter três tipos de obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classes Espinhos, Areia, Fogo e Estalactite, cada um instanciado nas fases.	Requisito cumprido inclusive via métodos para salvar de cada Entidade e de métodos nas classes de cada fase para recuperá-los.
8	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório de instâncias (i.e., objetos) sendo pelo menos 5 instâncias por tipo.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classes Espinhos, Areia, Fogo e Estalactite, cada um instanciado nas fases, sendo o último instanciado aleatoriamente quanto número e posição.	
9	Ter representação gráfica de cada instância.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via funções da biblioteca gráfica SFML, que nos permitiu representar graficamente /em uma janela nossos devidos objetos.	
10	Ter em cada fase um cenário de jogo com os obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via a classe GerenciadorMapa, Mapa e Tile.	
11	Gerenciar colisões entre jogador e inimigos, bem como seus projeteis (em havendo).	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe GerenciadorColisoes.	
12	Gerenciar colisões entre jogador e obstáculos.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe GerenciadorColisoes.	
13	Permitir cadastrar/salvar dados do usuário, manter pontuação durante jogo, salvar pontuação e gerar lista de pontuação (ranking).	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe Leaderboard.	
14	Permitir Pausar o Jogo	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via classe MenuPause.	
15	Permitir Salvar Jogada.	Requisito previsto inicialmente e realizado.	Requisito cumprido inclusive via métodos para salvar de cada Entidade e de métodos nas classes de cada fase para recuperá-los.	

Requisitos implementados 15 de 15 (100%)

Diagrama de classe

pacote Gerenciadores



1 - GerenciadorEstado

2- GerenciadorAtualizacoes

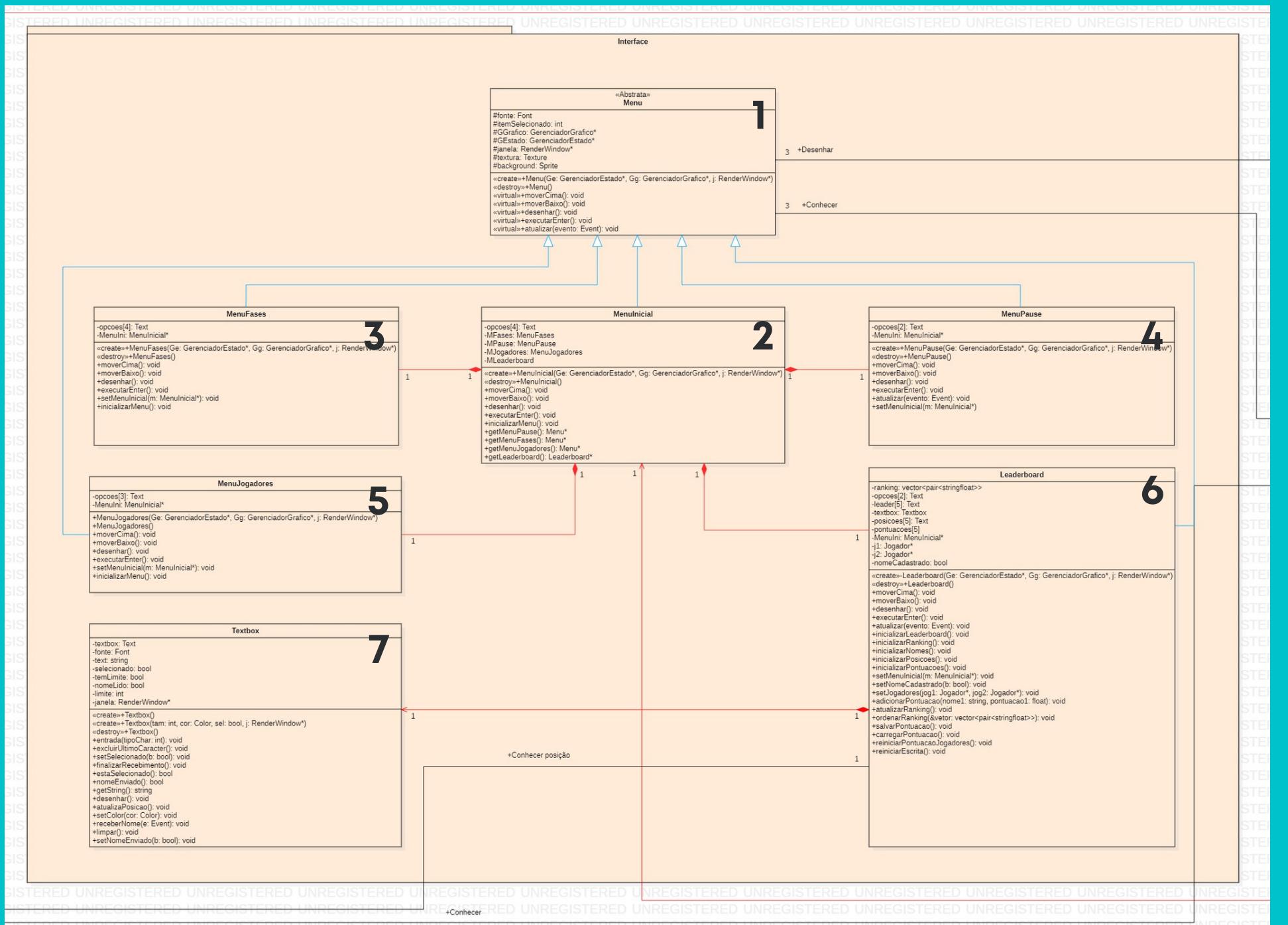
3 - GerenciadorGrafico

4 - GerenciadorColisoes

5 - GerenciadorMapa

Diagrama de classe

pacote Interface



1 - Menu (abstrata)

2- Menutnicial

3 - MenuFases

4 - MenuPause

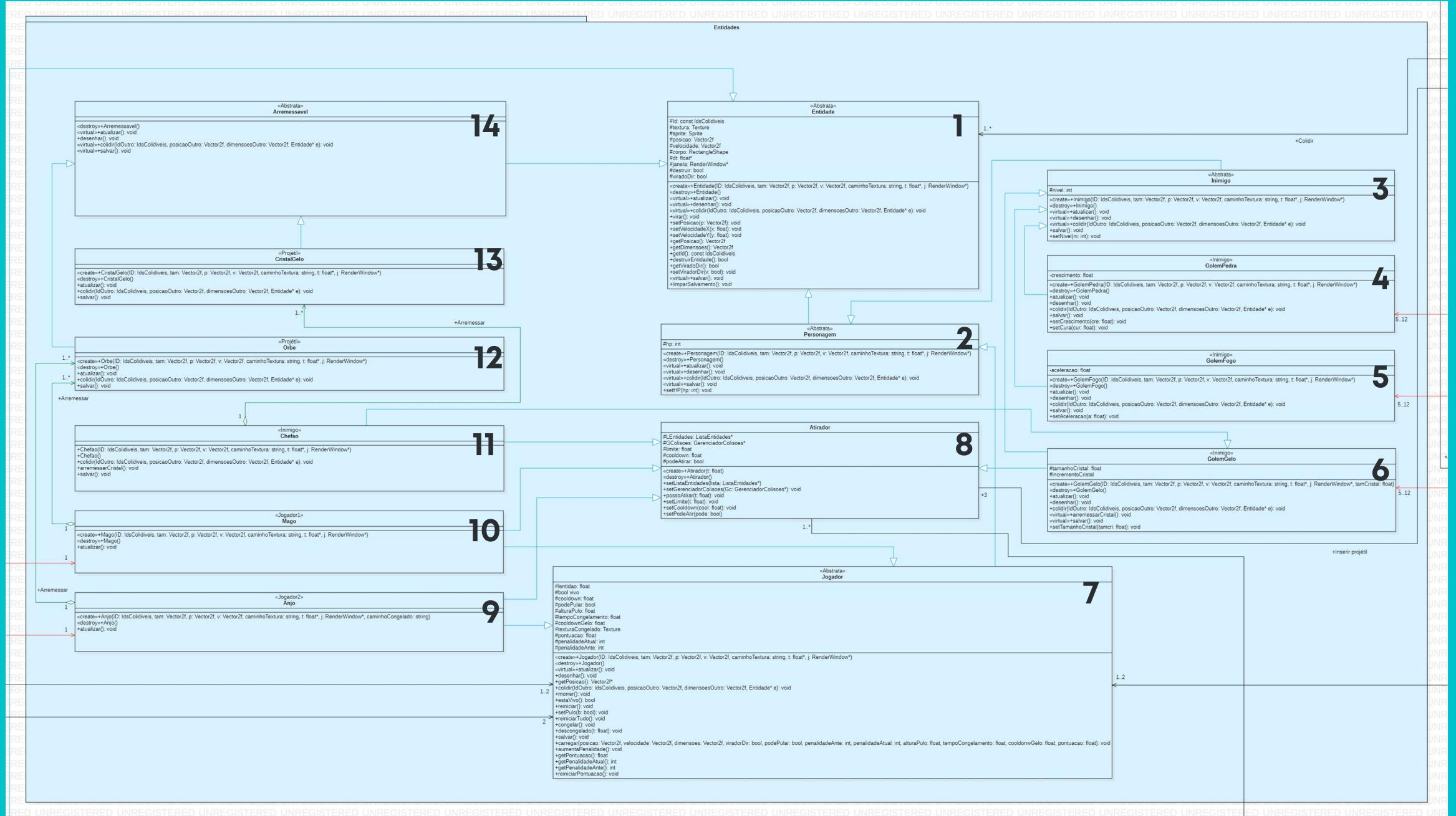
5 - MenuJogadores

6 - Leaderboard

7 - Textbox

Diagrama de classe

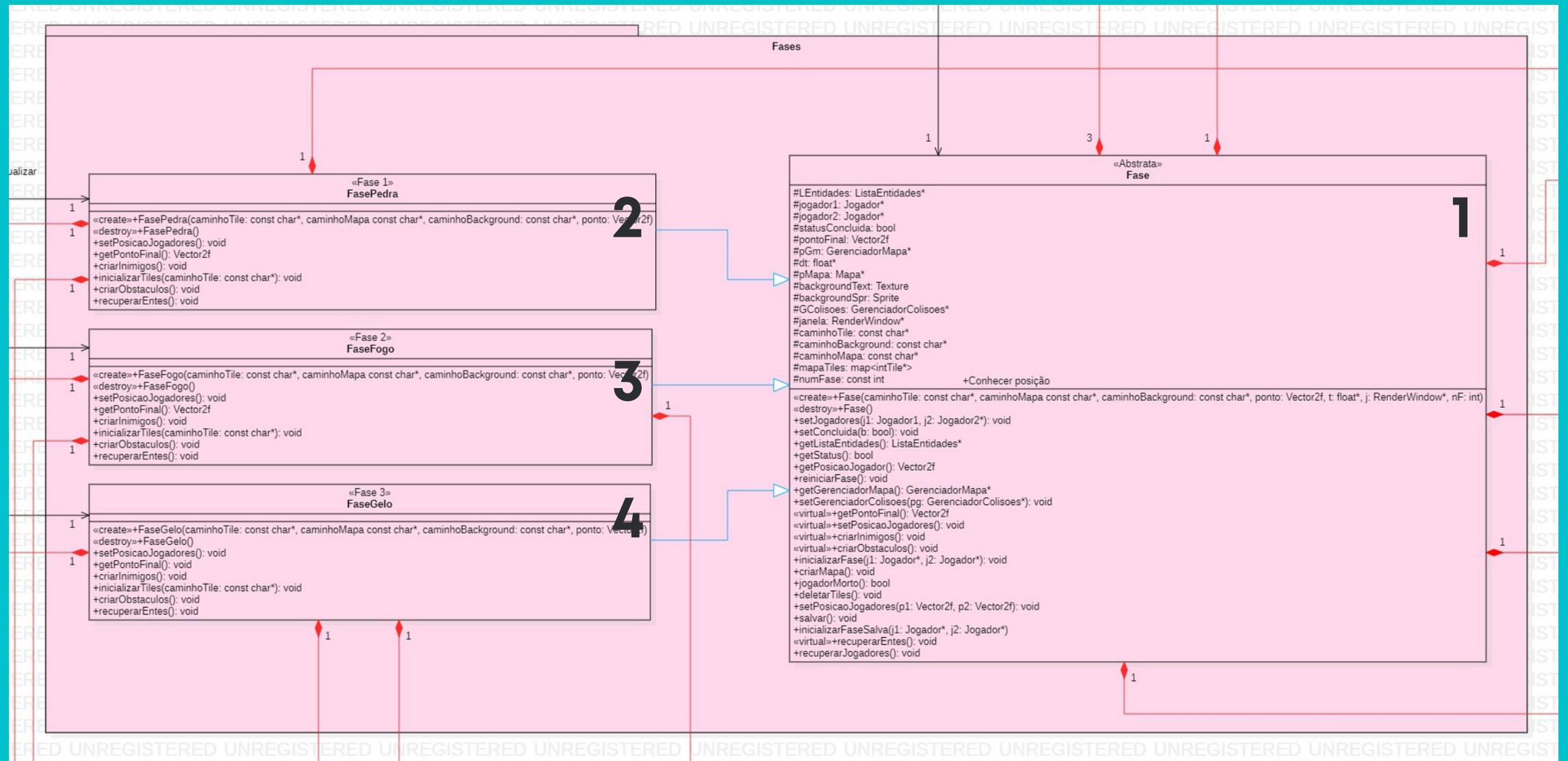
pacote Entidades



- 1 - *Entidade (abstrata)*
- 2- *Personagem (abstrata)*
- 3 - *Inimigo (abstrata)*
- 4 - *GolemPedra*
- 5 - *GolemFogo*
- 6 - *GolemGelo*
- 7 - *Jogador (abstrata)*
- 8 - *Atirador*
- 9 - *Anjo*
- 10 - *Mago*
- 11 - *Chefao*
- 12 - *Orbe*
- 13 - *CristalGelo*
- 14 - *Arremessavel (abstrata)*

Diagrama de classe

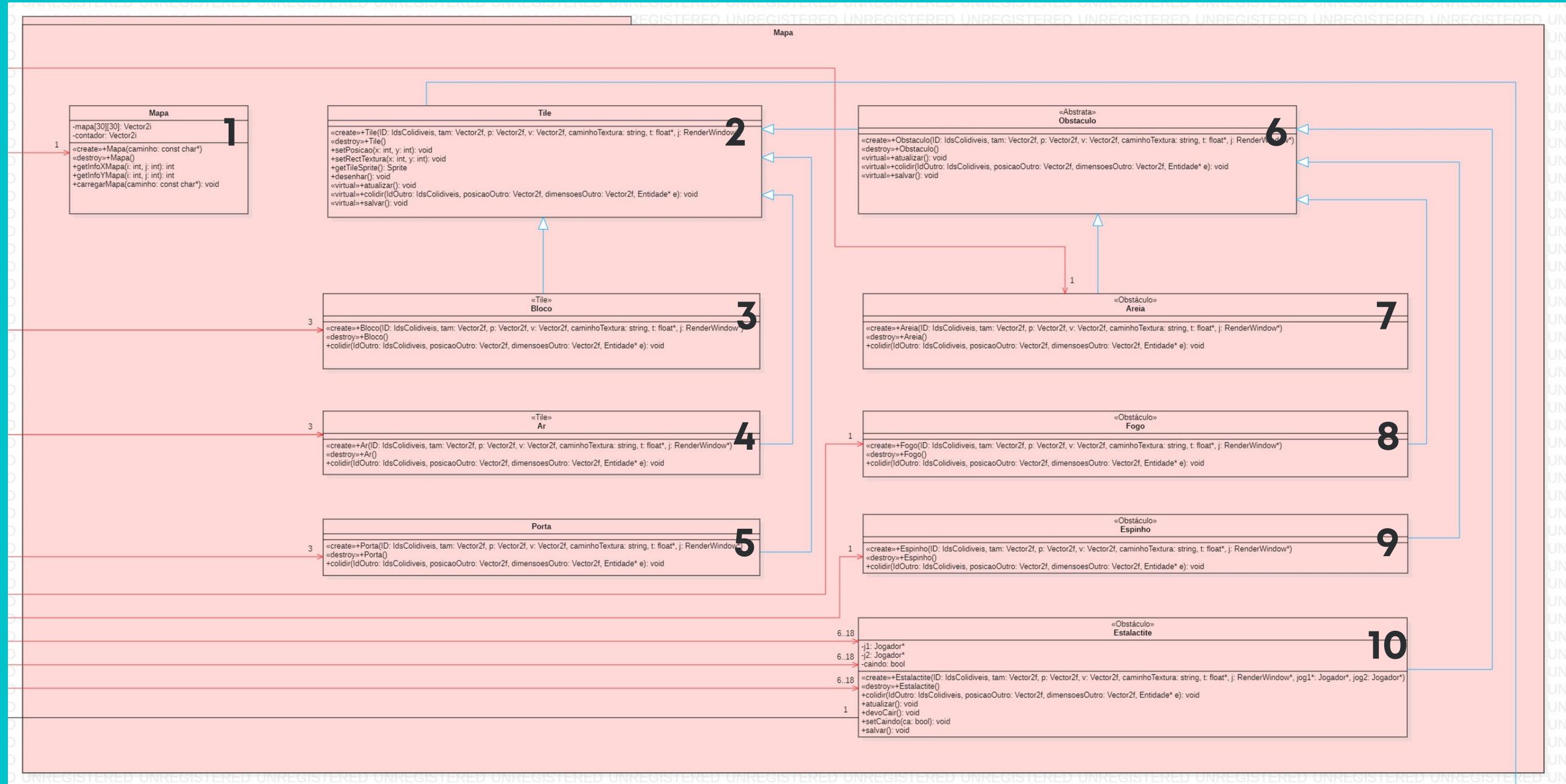
pacote Fases



- 1 - *Fase (abstrata)*
- 2 - *FasePedra*
- 3 - *FaseFogo*
- 4 - *FaseGelo*

Diagrama de classe

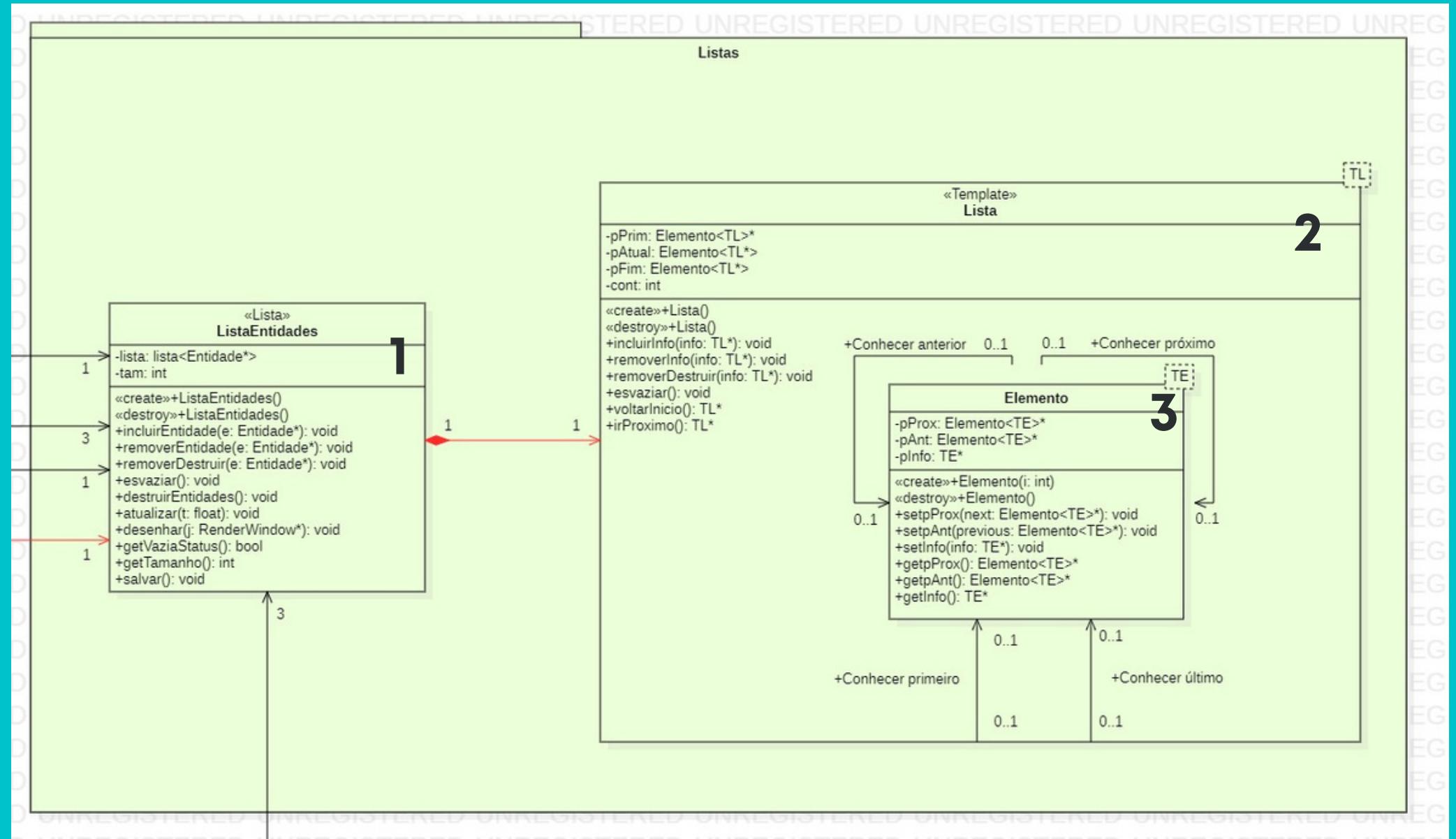
pacote Mapa



- 1 - Mapa
- 2 - Tile (abstrata)
- 3 - Bloco
- 4 - Ar
- 5 - Porta
- 6 - Obstaculo (abstrata)
- 7 - Areia
- 8 - Fogo
- 9 - Espinho
- 10 - Estalactite

Diagrama de classe

pacote Lista



- 1 - *ListaEntidades*
- 2 - *Lista<TL*>*
- 3 - *Elemento<TE*>*

Diagrama de classe

visão geral

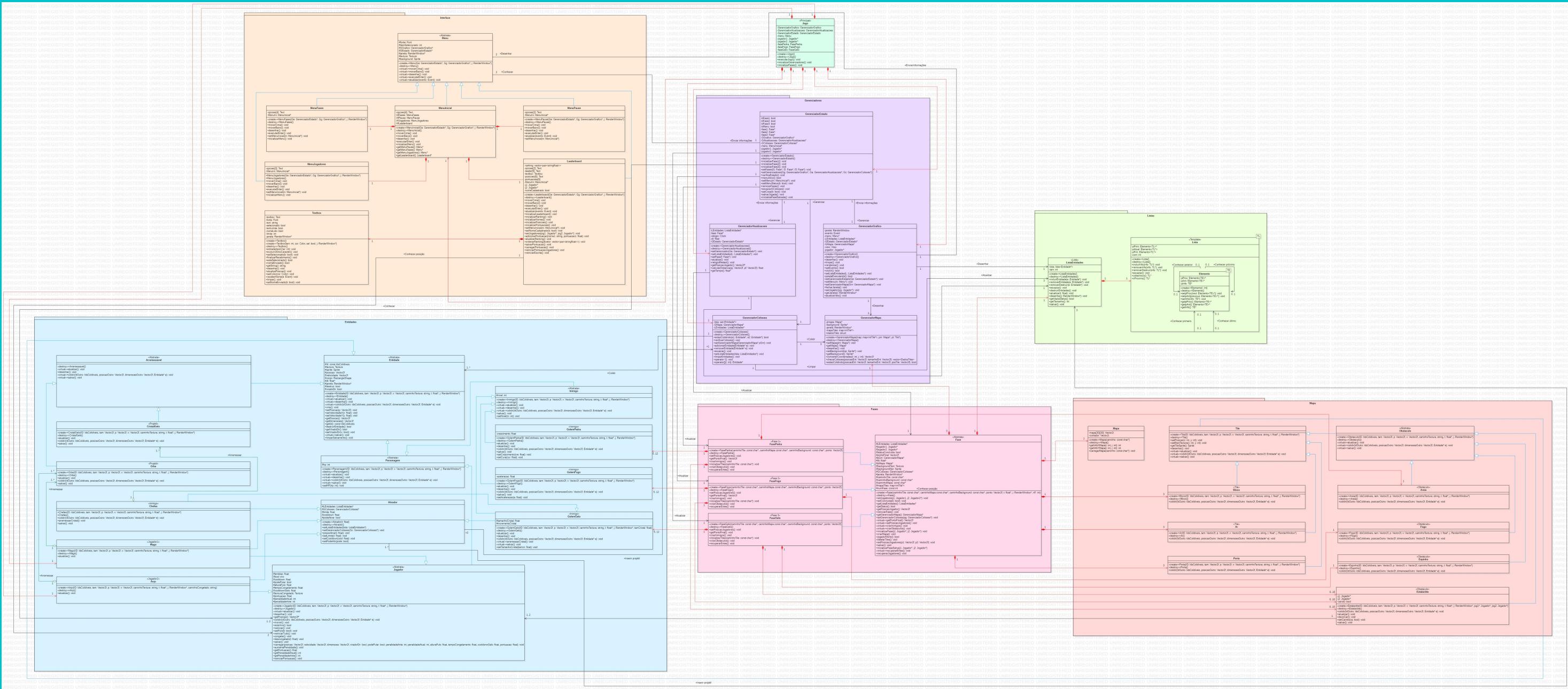


Tabela de conceitos

N.	Conceitos	Uso	Onde / O que
1	Elementares: - Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno) - Métodos (com retorno <i>const</i> e parâmetro <i>const</i>). & - Construtores (sem/com parâmetros) e destrutores - Classe Principal - Divisão em h e .cpp	Sim Sim Sim Sim Sim	Todos.h e .cpp Todos.h e .cpp Jogo.h & Jogo.cpp No desenvolvimento como um todo.
2	Relações de: - Associação direcional. & - Associação bidirecional. - Agregação via associação. & - Agregação propriamente dita. - Herança elementar. & - Herança em diversos níveis. - Herança múltipla.	Sim Sim Sim Sim	Na maioria das classes (conforme diagrama de classes e respectivo código). Exemplo: Associação bidirecional entre MenuFases e MenuInicial, Associação direcional entre ListaEntidades e Entidade. Na maioria das classes (conforme diagrama de classes e respectivo código). Exemplo: Agregação forte entre MenuInicial (agregador) e MenuFases, Agregação fraca entre ListaEntidades (agregador) e Entidade. Na maioria das classes (conforme diagrama de classes e respectivo código). Exemplo: Herança elementar entre Menu (superclasse) e MenuInicial, Herança em diversos níveis entre as classes Entidade (superclasse), Personagem, Inimigo e GolemFogo. Jogador tem herança da classe Personagem e Atirador, assim como GolemGelo.
3	Ponteiros, generalizações e exceções - Operador <i>this</i> . - Alocação de memória (<i>new</i> & <i>delete</i>). Gabaritos <i>Templates</i> criada/adaptados pelos autores (e.g. Listas Encadeadas via <i>Templates</i>)	Sim Sim Sim	Usado em diversos casos, como por exemplo: na construtora da classe Fase, no arquivo Fase.cpp Usado em diversos casos, como por exemplo: o <i>new</i> foi usado na função <i>criarMapa()</i> da classe Fase, no arquivo Fase.cpp; o <i>delete</i> foi usado na destrutora da classe Fase, no arquivo Fase.cpp. Foi criado uma lista template duplamente encadeada na classe Lista, utilizada para criar a ListaEntidades.
4	- Uso de Tratamento de Exceções (<i>try</i> <i>catch</i>). Sobrecarga de: - Construtoras e Métodos. - Operadores (2 tipos de operadores pelo menos).	Sim Sim Sim	Construtora da classe Fase, no arquivo Fase.cpp Sobrecarga de construtora na classe Textbox, e sobrecarga do método <i>setPosicaoJogadores()</i> na classe Fase. Sobrecarga do operador [] e do operador ! no GerenciadorColisoes.
5	Persistência de Objetos (via arquivo de texto ou binário) - Persistência de Objetos. - Persistência de Relacionamento de Objetos.	Sim Sim	Há persistência de Objetos em arquivos de texto presentes na pasta <i>salvar</i> . Posteriormente pode-se recuperar tais objetos. Cada Entidade têm seu próprio método de se salvar. Cada Fase consegue recuperar objetos salvos. As relações que as estalactites têm com os jogadores são recuperadas.
6	Organizadores e Estáticos - Espaço de Nomes (<i>Namespace</i>) criada pelos autores. - Classes aninhadas (<i>Nestea</i>) criada pelos autores. - Atributos estáticos e métodos estáticos. - Uso extensivo de constante (<i>const</i>) parâmetro, retorno, método...	Sim Sim Não Sim	Foi utilizado no arquivo IdsColidiveis.h Dentro do arquivo Lista.h, na classe Lista há uma classe aninhada chamada Elemento. No desenvolvimento como um todo.
7	Standard Template Library (<i>STL</i>) e String <i>OO</i> - A classe Pré-definida <i>String</i> ou equivalente. - <i>Vector</i> e/ou <i>List</i> da <i>STL</i> (p/ objetos ou ponteiros de objetos de classes definidos pelos autores) - Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa OU Multi-Mapa.	Sim Sim	A classe <i>string</i> foi utilizada no desenvolvimento como um todo. <i>Vector</i> foi utilizado dentro da classe GerenciadorColisoes. Foi utilizado <i>Conjunto</i> (<i>set</i>) na classe GerenciadorColisoes, <i>Mapa</i> (<i>map</i>) na classe GerenciadorMapa.
	Programação concorrente		
	- <i>Threads</i> (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time OU Win32API ou afins.	Não	

Tabela de conceitos

	- Threads (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, OU Troca de mensagens	Não	
8	<p>Biblioteca Gráfica / Visual</p> <p>- Funcionalidades Elementares. & - Funcionalidades Avançadas como:</p> <ul style="list-style-type: none"> • tratamento de colisões • duplo buffer <p>- Programação orientada a evento em algum ambiente gráfico.</p> <p>OU</p> <p>- RAD – Rapid Application Development (Objetos gráficos como formulários, botões etc).</p>	Sim	<p>Especificar aqui quais funcionalidades. Da biblioteca gráfica SFML, foram utilizadas as funcionalidades: Vector2f e Vector2i, estruturas que armazenam dois valores, float e int respectivamente; classes Texture, Sprite, RectShape, RenderWindow, View e seus métodos; das funcionalidades de eventos e input por meio de teclado e mouse, entre outros.</p> <p>Uso de um sistema dentro da Biblioteca SFML que suporta eventos.</p>
	Interdisciplinaridades por meio da utilização de Conceitos de Matemática e/ou Física.		
	- Ensino Médio	Sim	Conhecimento sobre plano cartesiano, eixo das abscissas e ordenadas, conceitos de aceleração, desaceleração, movimento retílineo uniformemente variado, aceleração gravitacional.
	- Ensino Superior	Sim	Conceitos mais avançados sobre vetores, normalização de vetores, soma e subtração de vetores, entendimento da sequência de Fibonacci, distância entre dois pontos aplicados à corpos extensos.

9	Engenharia de Software		
	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	Foi utilizado na primeira etapa do ciclo clássico de Engenharia de Software, onde todos os requisitos foram considerados e, de antemão, compreendidos.
	- Diagrama de Classes em UML	Sim	Foi utilizado na segunda etapa do ciclo clássico de Engenharia de Software, onde se deu o planejamento e modelagem de análise e projeto via Diagrama de Classes em UML.
	- Uso efetivo (quicá) intensivo de padrões de projeto (particularmente GOF)	Não	
	- Testes a luz da Tabela de Requisitos e do Diagrama de Classes	Sim	Foi utilizado na quarta e última etapa do ciclo clássico de Engenharia de Software, onde após implementado código, foi-se testado visando verificar requisitos e funcionalidades.
10	Execução de Projeto		
	- Controle de versão de modelos e códigos automatizado (via SVN e/ou afins) OU manual (via cópias manuais). &	Sim	Foi-se utilizado extensivamente um gerenciador de versões de arquivo, de arquitetura/sistema distribuído, o git, conjuntamente ao GitHub, formando repositórios, tanto locais quanto online. Ainda nesse sistema, visando a segurança, criou-se branches de backup do projeto.
	- Uso de alguma forma de cópia de segurança (backup)		

	- Reuniões com o professor para acompanhamento do andamento do projeto.	Sim	Feitas todas as 4 Reuniões: (21/04), (28/04), (05/05) e (07/05).
	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto.	Sim	Foram feitas somente 6 reuniões, pois as dúvidas foram agrupadas para serem atendidas, visto que havia comumente duplas já em reunião. Um outro fator foi a restrição de horário da dupla. Reuniões: Skora (11/03), Skora (22/03), Augusto (08/04), Augusto (26/04), Skora (29/04), Augusto (07/05).
	- Revisão do trabalho escrito de outra equipe e vice-versa	Sim	Equipe Garret & Yuske.

Requisitos implementados 36 de 40 (90%)

Requisitos não implementados : Static, Threads, Padrões de Projeto e 10 reuniões com monitores

O jogo

Jogadores



Inimigos



Projéteis



O jogo



MAGIC GOLENS		
1	THROCKMORTON	999999
2	DEZDEZ	10000
3	TESTE	5000
4	TSETSE	3333
5	DI	294

SALVAR PONTUAÇÃO

MENU PRINCIPAL

O jogo



Conclusão geral

Portanto, fica claro que o projeto MagicGolens, desenvolvido para a disciplina Técnicas de Programação da Universidade Tecnológica do Paraná, teve sucesso no cumprimento dos requisitos propostos inicialmente, bem como visou atender a maior parte dos conceitos relacionados à Programação Orientada a Objetos (POO) ministrados durante o semestre.

Outrossim, é inegável a relevância do trabalho para a formação profissional. Tal fato se deve à prática dos conceitos e habilidades adquiridas durante as aulas, como também à descoberta de uma gama de novos conteúdos a serem aprendidos e aperfeiçoados, incrementando, assim, ainda mais o conhecimento de POO.