Apresentação do trabalho final para a disciplina de técnicas de programação do curso de Engenharia de Computação
Turma S71 - 1 Sem/2023

Bruno Lapa, Matheus Cruz da Silva
brunolapa@aluno.utfpr.edu.br, silvam.2020@aluno.utfpr.edu.br

CAMPUS CURITIBA

UniversidadequeTransforma

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA

Tabela de requisitos

N.	*Requisitos Funcionais*	*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 cumprido via classe Menu e Botão.
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 cumprido via classe Jogador cujos objetos são agregados em jogo, podendo instanciar os dois jogadores.
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.	Cumprido via pacote Nível e pelo objeto da classe Menu, onde as fases podem ser escolhidas no menu principal.
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 projéteis contra o(s) jogador(es) e um dos inimigos dever ser um 'Chefão'.	Requisito cumprido via classe Inimigo, e dos objetos derivados da classe nível e da classe Projéteis.
5	Ter a cada fase ao menos dois tipos de inimigos com número aleatório de instâncias, podendo ser várias instâncias (definindo um máximo) e sendo pelo menos 3 instâncias por tipo.	Inimigos gerenciados e criados de maneira aleatória pelo ConstrutorNivel, cada fase instância uma quantidade aleatória.

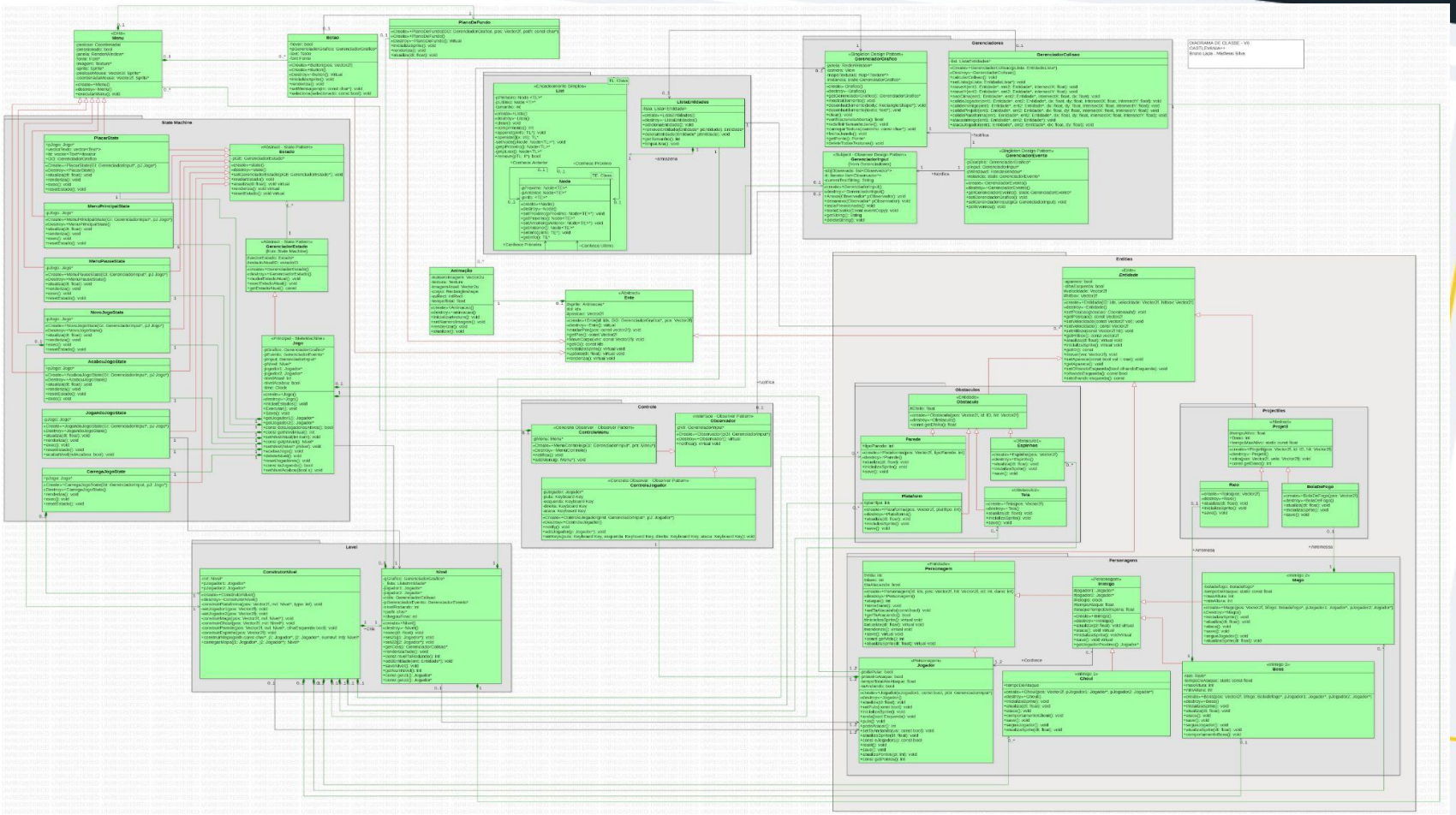
Tabela de requisitos

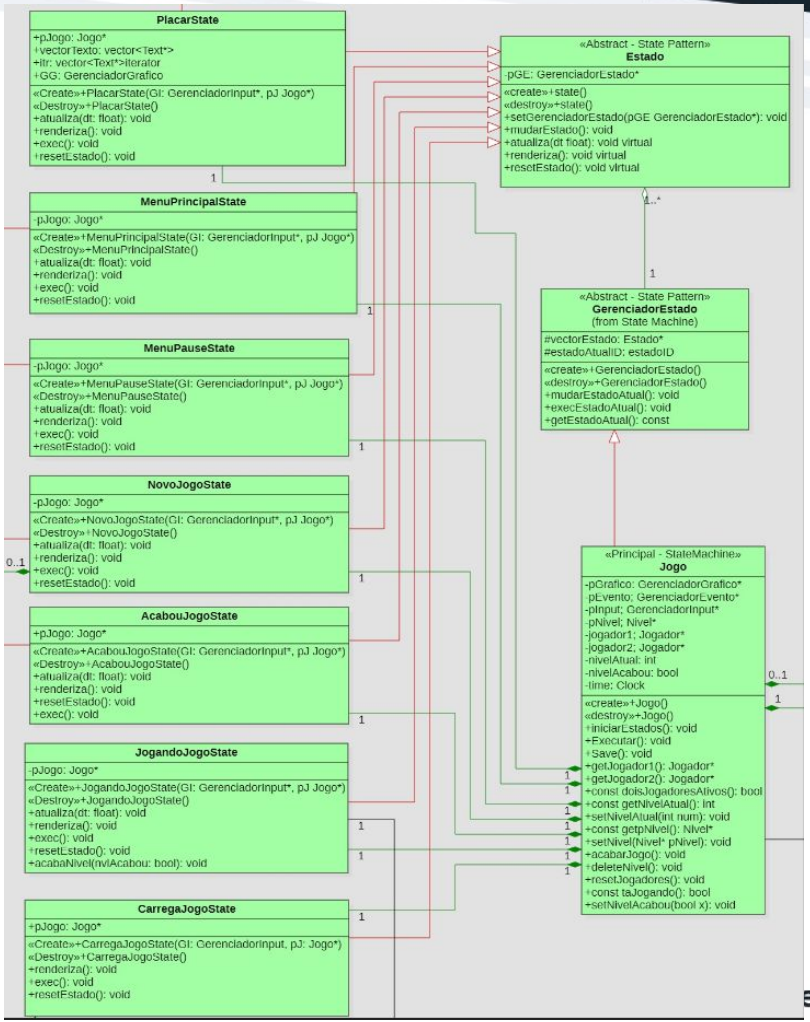
N.	***Requisitos Funcionais***	***Implementação***
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem.	Requisito cumprido via classe Obstáculos e seus objetos derivados.
7	Ter em cada fase ao menos dois tipos de obstáculos com número aleatório (definindo um máximo) de instâncias (*i.e.*, objetos), sendo pelo menos 3 instâncias por tipo.	Requisito cumprido via Gerenciador de Nível e Construtor Nível e seus derivados.
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.	Cumprido por meio da classe Nível e objetos de classes derivados de ConstrutorNível.
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 de alguma ´gravidade´ no âmbito deste jogo de plataforma vertical e 2D.	Cumprido por meio do uso das classes ListaEntidade e GerenciadorColisao e seus derivados, que itera as entidades e calcula suas colisões.
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.	Cumprido pelas classes GerenciadorEventos, PauseMenuState, CarregarJogoState e funções de salvar incluídas em cada Entidade.

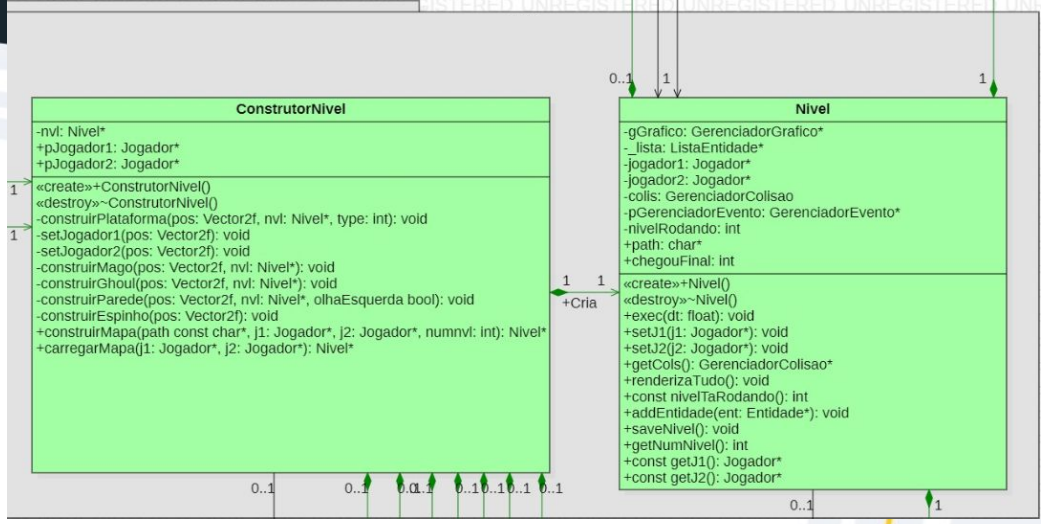
Tabela de requisitos

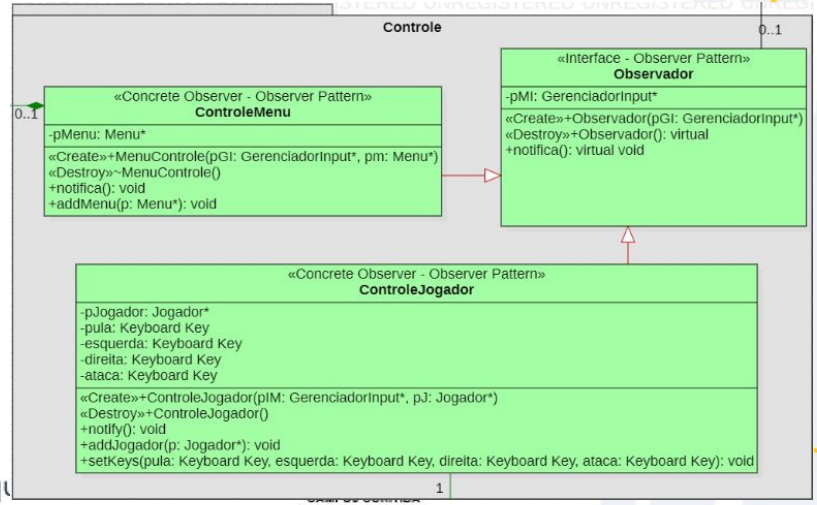
Total de requisitos funcionais apropriadamente realizados.	**100%** (cem por cento).

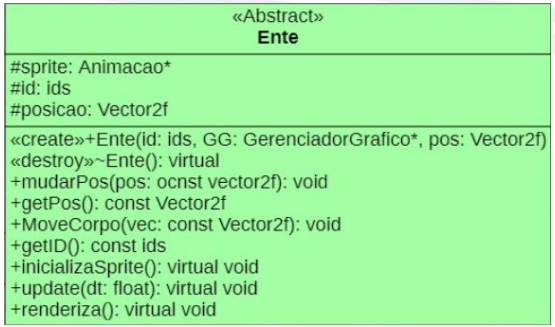
Diagrama de classes

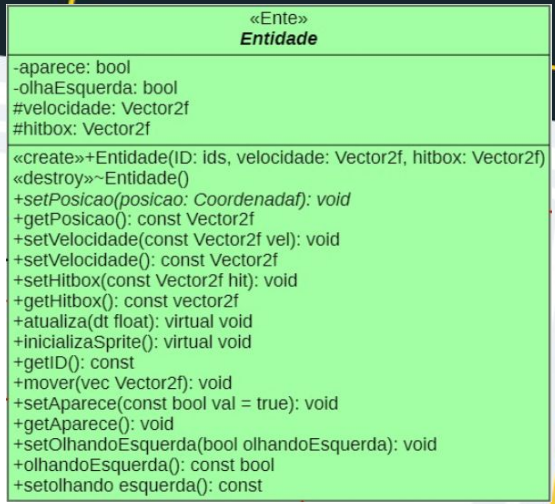


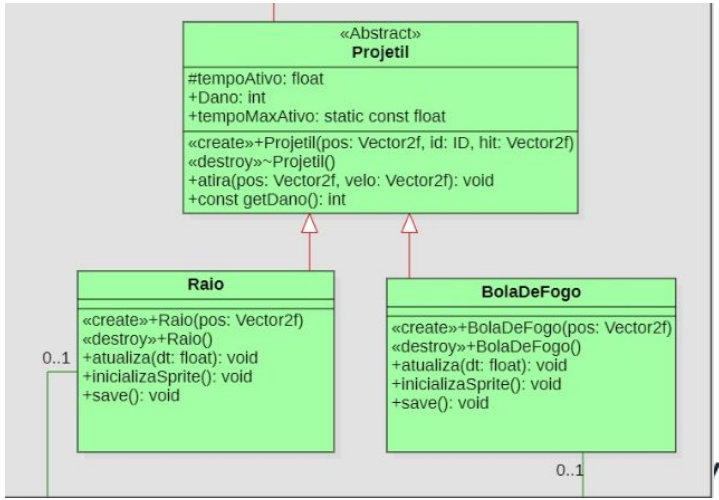

PlacarState
+pJogo: Jogo*
+vectorTexto: vector<Text*>
+itr: vector<Text*>iterator
+GG: GerenciadorGrafico
«Create»+PlacarState(GI: GerenciadorInput*, pJ Jogo*)
«Destroy»+PlacarState()
+atualiza(dt: float): void
+renderiza(): void
+exec(): void
+resetEstado(): void
«Abstract - State Pattern»
Estado
-pGE: GerenciadorEstado*
«create»+state()
«destroy»+state()
+setGerenciadorEstado(pGE GerenciadorEstado*): void
+mudarEstado(): void
+atualiza(dt float): void virtual
+renderiza(): void virtual
+resetEstado(): void virtual
MenuPrincipalState
-pJogo: Jogo*
«Create»+MenuPrincipalState(GI: GerenciadorInput*, pJ Jogo*)
«Destroy»+MenuPrincipalState()
+atualiza(dt: float): void
+renderiza(): void
+exec(): void
+resetEstado(): void
«Abstract - State Pattern»
GerenciadorEstado
(from State Machine)
#vectorEstado: Estado*
#estadoAtualID; estadoID
«create»+GerenciadorEstado()
«destroy»+GerenciadorEstado()
+mudarEstadoAtual(): void
+execEstadoAtual(): void
+getEstadoAtual(): const
MenuPauseState
-pJogo: Jogo*
«Create»+MenuPauseState(GI: GerenciadorInput*, pJ Jogo*)
«Destroy»+MenuPauseState()
+atualiza(dt: float): void
+renderiza(): void
+exec(): void
+resetEstado(): void
NovoJogoState
-pJogo: Jogo*
«Create»+NovoJogoState(GI: GerenciadorInput*, pJ Jogo*)
«Destroy»+NovoJogoState()
+atualiza(dt: float): void
+renderiza(): void
+exec(): void
+resetEstado(): void
«Principal - StateMachine»
Jogo
-pGrafico: GerenciadorGrafico*
-pEvento: GerenciadorEvento*
-pInput; GerenciadorInput*
-pNivel; Nivel*
-jogador1; Jogador*
-jogador2; Jogador*
-nivelAtual: int
-nivelAcabou: bool
-time; Clock
«create»+Jogo()
«destroy»+Jogo()
+iniciarEstados(): void
+Executar(): void
+Save(): void
+getJogador1(): Jogador*
+getJogador2(): Jogador*
+const doisJogadoresAtivos(): bool
+const getNivelAtual(): int
+setNivelAtual(int num): void
+const getpNivel(): Nivel*
+setNivel(Nivel* pNivel): void
+acabarJogo(): void
+deleteNivel(): void
+resetJogadores(): void
+const taJogando(): bool
+setNivelAcabou(bool x): void
AcabouJogoState
+pJogo: Jogo*
«Create»+AcabouJogoState(GI: GerenciadorInput*, pJ Jogo*)
«Destroy»+AcabouJogoState()
+atualiza(dt: float): void
+renderiza(): void
+resetEstado(): void
+exec(): void
JogandoJogoState
-pJogo: Jogo*
«Create»+JogandoJogoState(GI: GerenciadorInput*, pJ Jogo*)
«Destroy»+JogandoJogoState()
+atualiza(dt: float): void
+renderiza(): void
+exec(): void
+resetEstado(): void
+acabaNivel(nvlAcabou: bool): void
CarregaJogoState
+pJogo: Jogo*
«Create»+CarregaJogoState(GI: GerenciadorInput, pJ: Jogo*)
«Destroy»+CarregaJogoState()
+renderiza(): void
+exec(): void
+resetEstado(): void

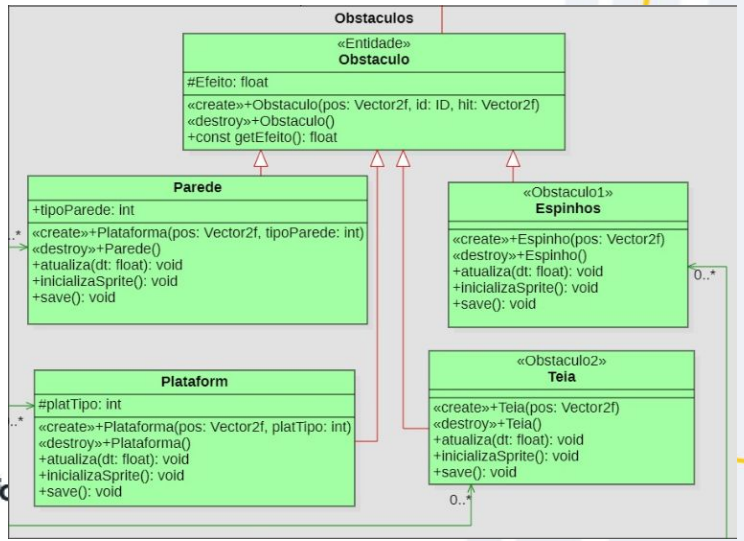

ConstrutorNivel
-nvl: Nivel*
+pJogador1: Jogador*
+pJogador2: Jogador*
«create»+ConstrutorNivel()
«destroy»~ConstrutorNivel()
-construirPlataforma(pos: Vector2f, nvl: Nivel*, type: int): void
-setJogador1(pos: Vector2f): void
-setJogador2(pos: Vector2f): void
-construirMago(pos: Vector2f, nvl: Nivel*): void
-construirGhoul(pos: Vector2f, nvl: Nivel*): void
-construirParede(pos: Vector2f, nvl: Nivel*, olhaEsquerda bool): void
-construirEspinho(pos: Vector2f): void
+construirMapa(path const char*, j1: Jogador*, j2: Jogador*, numnvl: int): Nivel*
+carregarMapa(j1: Jogador*, j2: Jogador*): Nivel*
+Cria
Nivel
-gGrafico: GerenciadorGrafico*
-_lista: ListaEntidade*
-jogador1: Jogador*
-jogador2: Jogador*
-colis: GerenciadorColisao
-pGerenciadorEvento: GerenciadorEvento*
-nivelRodando: int
+path: char*
+chegouFinal: int
«create»+Nivel()
«destroy»~Nivel()
+exec(dt: float): void
+setJ1(j1: Jogador*): void
+setJ2(j2: Jogador*): void
+getCols(): GerenciadorColisao*
+renderizaTudo(): void
+const nivelTaRodando(): int
+addEntidade(ent: Entidade*): void
+saveNivel(): void
+getNumNivel(): int
+const getJ1(): Jogador*
+const getJ2(): Jogador*

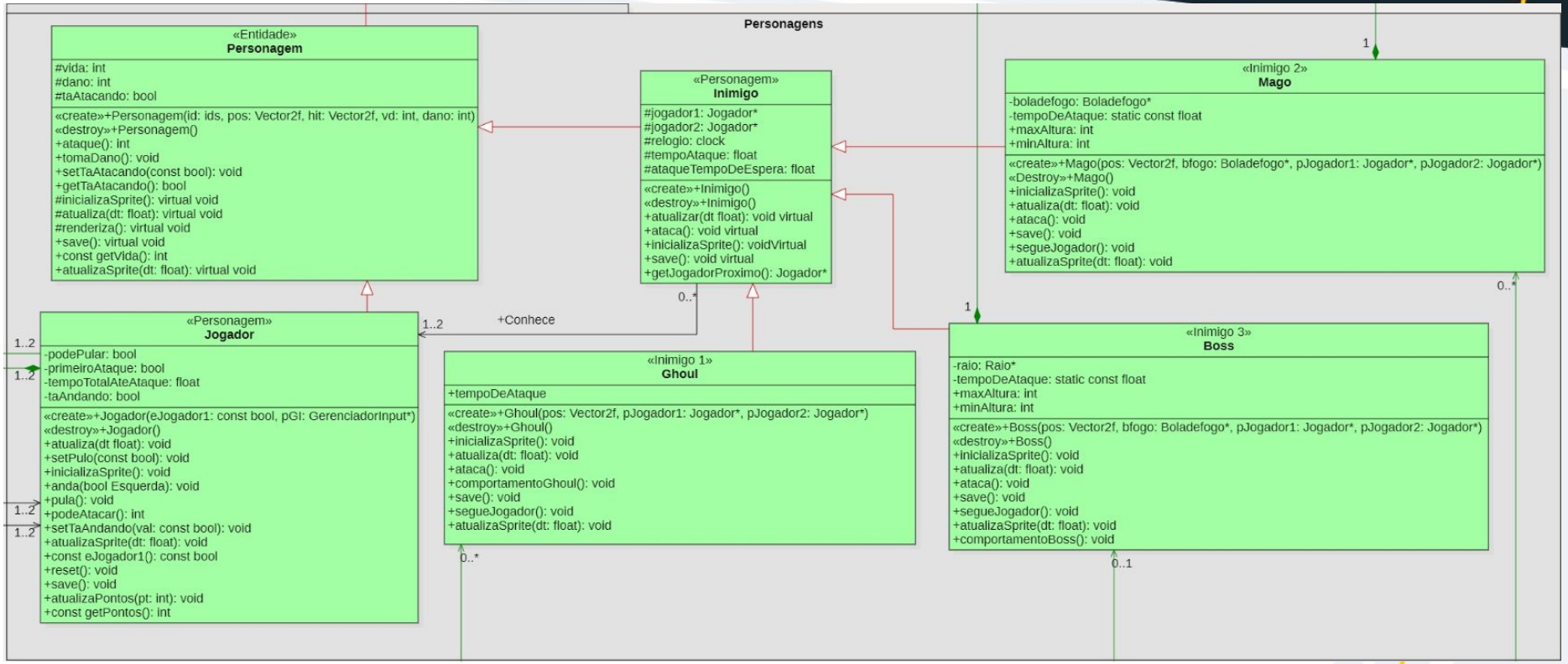

Controle
«Concrete Observer - Observer Pattern»
ControleMenu
-pMenu: Menu*
«Create»+MenuControle(pGI: GerenciadorInput*, pm: Menu*)
«Destroy»~MenuControle()
+notifica(): void
+addMenu(p: Menu*): void
«Interface - Observer Pattern»
Observador
-pMI: GerenciadorInput*
«Create»+Observador(pGI: GerenciadorInput*)
«Destroy»+Observador(): virtual
+notifica(): virtual void
«Concrete Observer - Observer Pattern»
ControleJogador
-pJogador: Jogador*
-pula: Keyboard Key
-esquerda: Keyboard Key
-direita: Keyboard Key
-ataca: Keyboard Key
«Create»+ControleJogador(pIM: GerenciadorInput*, pJ: Jogador*)
«Destroy»+ControleJogador()
+notify(): void
+addJogador(p: Jogador*): void
+setKeys(pula: Keyboard Key, esquerda: Keyboard Key, direita: Keyboard Key, ataca: Keyboard Key): void

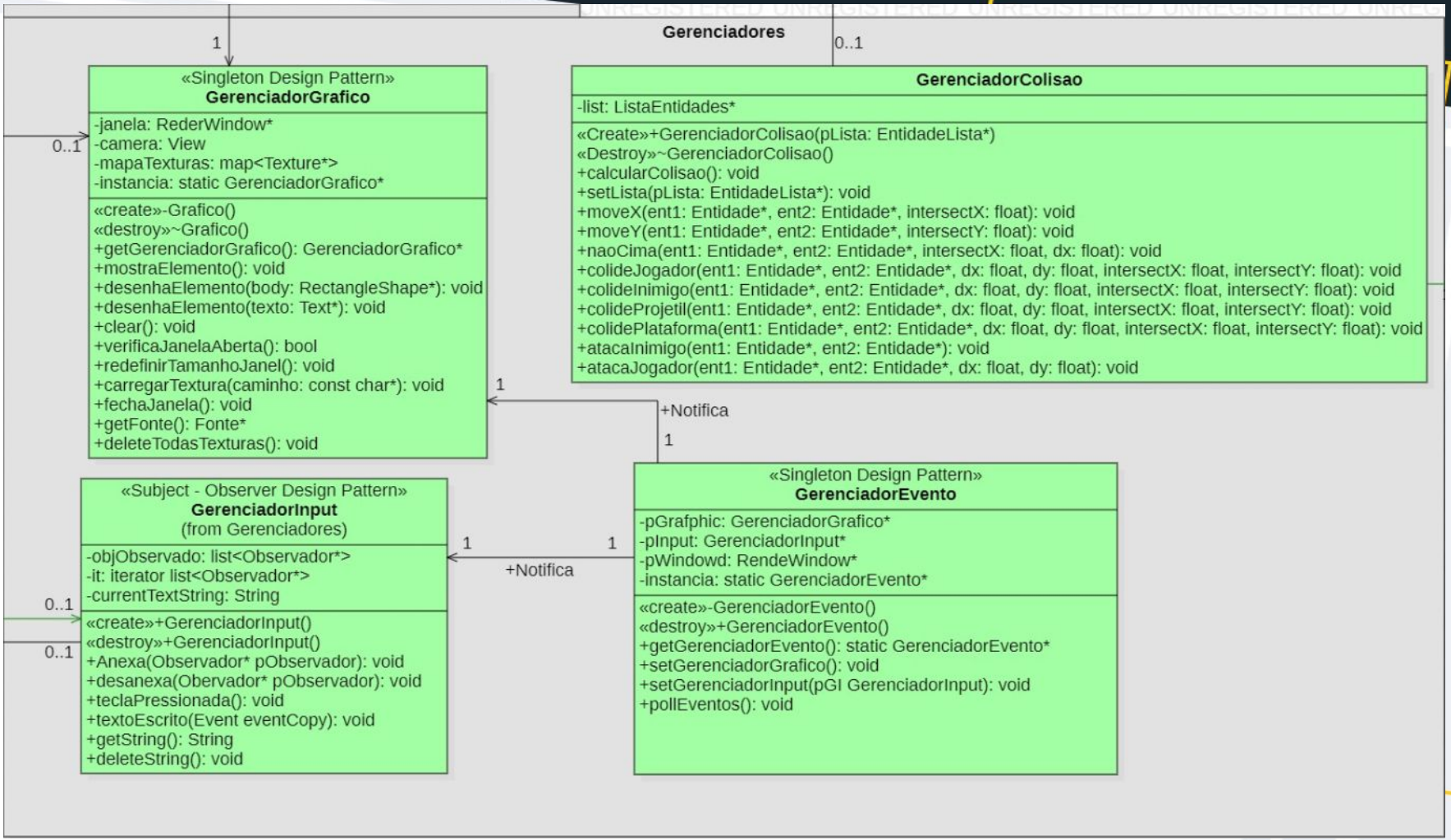

«Abstract»
Ente
#sprite: Animacao*
#id: ids
#posicao: Vector2f
«create»+Ente(id: ids, GG: GerenciadorGrafico*, pos: Vector2f)
«destroy»~Ente(): virtual
+mudarPos(pos: ocnst vector2f): void
+getPos(): const Vector2f
+MoveCorpo(vec: const Vector2f): void
+getID(): const ids
+inicializaSprite(): virtual void
+update(dt: float): virtual void
+renderiza(): virtual void

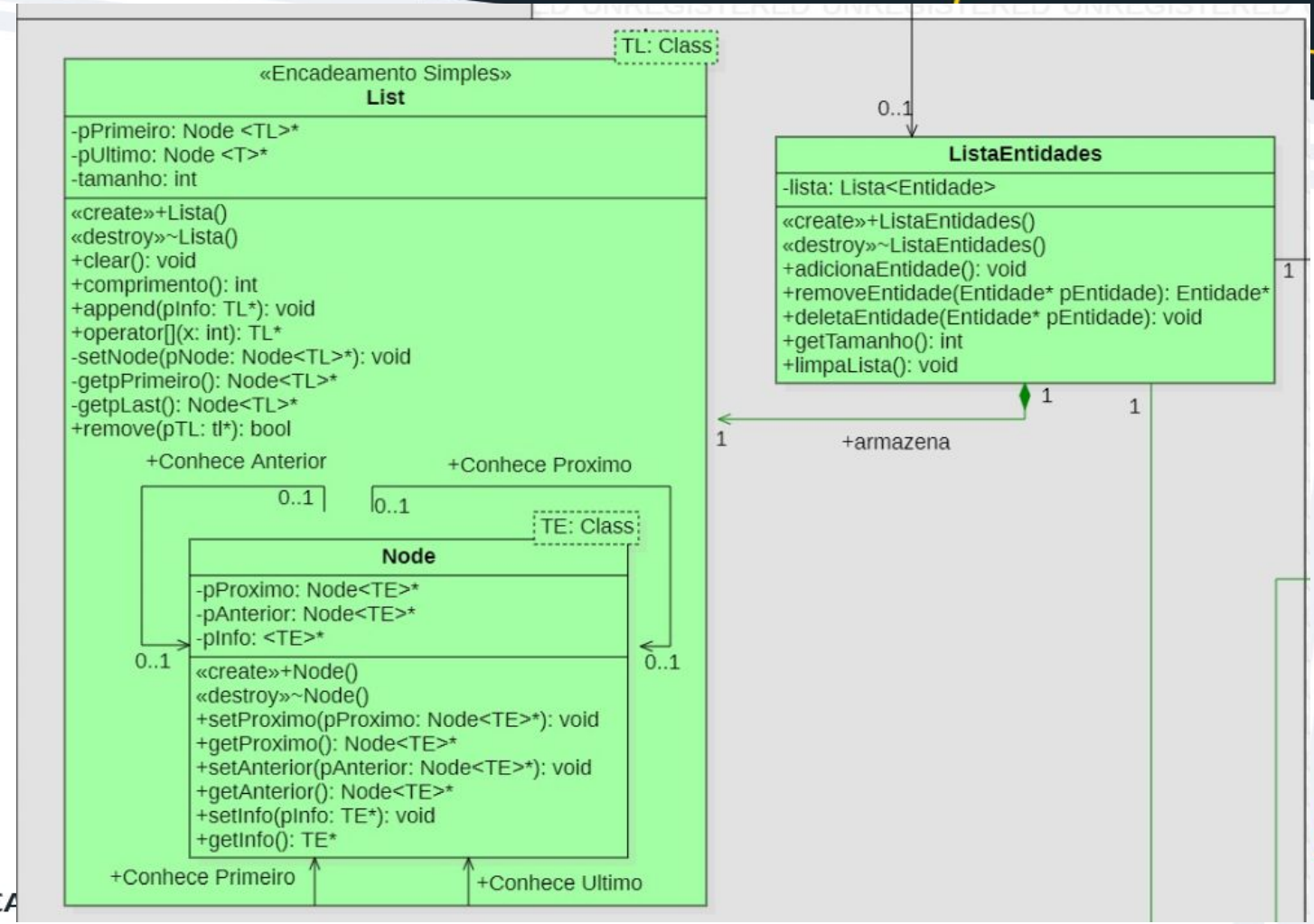

«Ente»
Entidade
-aparece: bool
-olhaEsquerda: bool
#velocidade: Vector2f
#hitbox: Vector2f
«create»+Entidade(ID: ids, velocidade: Vector2f, hitbox: Vector2f)
«destroy»~Entidade()
+setPosicao(posicao: Coordenadaf): void
+getPosicao(): const Vector2f
+setVelocidade(const Vector2f vel): void
+setVelocidade(): const Vector2f
+setHitbox(const Vector2f hit): void
+getHitbox(): const vector2f
+atualiza(dt float): virtual void
+inicializaSprite(): virtual void
+getID(): const
+mover(vec Vector2f): void
+setAparece(const bool val = true): void
+getAparece(): void
+setOlhandoEsquerda(bool olhandoEsquerda): void
+olhandoEsquerda(): const bool
+setolhando esquerda(): const


«Abstract»
Projetil
#tempoAtivo: float
+Dano: int
+tempoMaxAtivo: static const float
«create»+Projetil(pos: Vector2f, id: ID, hit: Vector2f)
«destroy»~Projetil()
+atira(pos: Vector2f, velo: Vector2f): void
+const getDano(): int
Raio
«create»+Raio(pos: Vector2f)
«destroy»+Raio()
+atualiza(dt: float): void
+inicializaSprite(): void
+save(): void
0..1
BolaDeFogo
«create»+BolaDeFogo(pos: Vector2f)
«destroy»+BolaDeFogo()
+atualiza(dt: float): void
+inicializaSprite(): void
+save(): void
0..1


Obstaculos
«Entidade»
Obstaculo
#Efeito: float
«create»+Obstaculo(pos: Vector2f, id: ID, hit: Vector2f)
«destroy»+Obstaculo()
+const getEfeito(): float
Parede
+tipoParede: int
«create»+Plataforma(pos: Vector2f, tipoParede: int)
«destroy»+Parede()
+atualiza(dt: float): void
+inicializaSprite(): void
+save(): void
«Obstaculo1»
Espinhos
«create»+Espinho(pos: Vector2f)
«destroy»+Espinho()
+atualiza(dt: float): void
+inicializaSprite(): void
+save(): void
0..*
Plataform
#platTipo: int
«create»+Plataforma(pos: Vector2f, platTipo: int)
«destroy»+Plataforma()
+atualiza(dt: float): void
+inicialiaSprite(): void
+save(): void
«Obstaculo2»
Teia
«create»+Teia(pos: Vector2f)
«destroy»+Teia()
+atualiza(dt: float): void
+inicializaSprite(): void
+save(): void
0..*


Personagens
«Entidade»
Personagem
#vida: int
#dano: int
#taAtacando: bool
«create»+Personagem(id: ids, pos: Vector2f, hit: Vector2f, vd: int, dano: int)
«destroy»+Personagem()
+ataque(): int
+tomaDano(): void
+setTaAtacando(const bool): void
+getTaAtacando(): bool
#inicializaSprite(): virtual void
#atualiza(dt: float): virtual void
#renderiza(): virtual void
+save(): virtual void
+const getVida(): int
+atualizaSprite(dt: float): virtual void
«Personagem»
Inimigo
#jogador1: Jogador*
#jogador2: Jogador*
#relogio: clock
#tempoAtaque: float
#ataqueTempoDeEspera: float
«create»+Inimigo()
«destroy»+Inimigo()
+atualizar(dt float): void virtual
+ataca(): void virtual
+inicializaSprite(): voidVirtual
+save(): void virtual
+getJogadorProximo(): Jogador*
«Inimigo 2»
Mago
-boladefogo: Boladefogo*
-tempoDeAtaque: static const float
+maxAltura: int
+minAltura: int
«create»+Mago(pos: Vector2f, bfogo: Boladefogo*, pJogador1: Jogador*, pJogador2: Jogador*)
«Destroy»+Mago()
+inicializaSprite(): void
+atualiza(dt: float): void
+ataca(): void
+save(): void
+segueJogador(): void
+atualizaSprite(dt: float): void
«Personagem»
Jogador
-podePular: bool
-primeiroAtaque: bool
-tempoTotalAteAtaque: float
-taAndando: bool
«create»+Jogador(eJogador1: const bool, pGI: GerenciadorInput*)
«destroy»+Jogador()
+atualiza(dt float): void
+setPulo(const bool): void
+inicializaSprite(): void
+anda(bool Esquerda): void
+pula(): vold
+podeAtacar(): int
+setTaAndando(val: const bool): void
+atualizaSprite(dt: float): void
+const eJogador1(): const bool
+reset(): void
+save(): void
+atualizaPontos(pt: int): void
+const getPontos(): int
+Conhece
«Inimigo 1»
Ghoul
+tempoDeAtaque
«create»+Ghoul(pos: Vector2f, pJogador1: Jogador*, pJogador2: Jogador*)
«destroy»+Ghoul()
+inicializaSprite(): void
+atualiza(dt: float): void
+ataca(): void
+comportamentoGhoul(): void
+save(): void
+segueJogador(): void
+atualizaSprite(dt: float): void
«Inimigo 3»
Boss
-raio: Raio*
-tempoDeAtaque: static const float
+maxAltura: int
+minAltura: int
«create»+Boss(pos: Vector2f, bfogo: Boladefogo*, pJogador1: Jogador*, pJogador2: Jogador*)
«destroy»+Boss()
+inicializaSprite(): void
+atualiza(dt: float): void
+ataca(): void
+save(): void
+segueJogador(): void
+atualizaSprite(dt: float): void
+comportamentoBoss(): void
1
0..*
1..2
0..1


Gerenciadores
1
0..1
«Singleton Design Pattern»
GerenciadorGrafico
-janela: RederWindow*
-camera: View
-mapaTexturas: map<Texture*>
-instancia: static GerenciadorGrafico*
«create»-Grafico()
«destroy»~Grafico()
+getGerenciadorGrafico(): GerenciadorGrafico*
+mostraElemento(): void
+desenhaElemento(body: RectangleShape*): void
+desenhaElemento(texto: Text*): void
+clear(): void
+verificaJanelaAberta(): bool
+redefinirTamanhoJanel(): void
+carregarTextura(caminho: const char*): void
+fechaJanela(): void
+getFonte(): Fonte*
+deleteTodasTexturas(): void
0..1
GerenciadorColisao
-list: ListaEntidades*
«Create»+GerenciadorColisao(pLista: EntidadeLista*)
«Destroy»~GerenciadorColisao()
+calcularColisao(): void
+setLista(pLista: EntidadeLista*): void
+moveX(ent1: Entidade*, ent2: Entidade*, intersectX: float): void
+moveY(ent1: Entidade*, ent2: Entidade*, intersectY: float): void
+naoCima(ent1: Entidade*, ent2: Entidade*, intersectX: float, dx: float): void
+colideJogador(ent1: Entidade*, ent2: Entidade*, dx: float, dy: float, intersectX: float, intersectY: float): void
+colideInimigo(ent1: Entidade*, ent2: Entidade*, dx: float, dy: float, intersectX: float, intersectY: float): void
+colideProjetil(ent1: Entidade*, ent2: Entidade*, dx: float, dy: float, intersectX: float, intersectY: float): void
+colidePlataforma(ent1: Entidade*, ent2: Entidade*, dx: float, dy: float, intersectX: float, intersectY: float): void
+atacaInimigo(ent1: Entidade*, ent2: Entidade*): void
+atacaJogador(ent1: Entidade*, ent2: Entidade*, dx: float, dy: float): void
1
+Notifica
1
«Subject - Observer Design Pattern»
GerenciadorInput
(from Gerenciadores)
-objObservado: list<Observador*>
-it: iterator list<Observador*>
-currentTextString: String
«create»+GerenciadorInput()
«destroy»+GerenciadorInput()
+Anexa(Observador* pObservador): void
+desanexa(Obervador* pObservador): void
+teclaPressionada(): void
+textoEscrito(Event eventCopy): void
+getString(): String
+deleteString(): void
0..1
0..1
1
1
+Notifica
«Singleton Design Pattern»
GerenciadorEvento
-pGrafphic: GerenciadorGrafico*
-pInput: GerenciadorInput*
-pWindowd: RendeWindow*
-instancia: static GerenciadorEvento*
«create»-GerenciadorEvento()
«destroy»+GerenciadorEvento()
+getGerenciadorEvento(): static GerenciadorEvento*
+setGerenciadorGrafico(): void
+setGerenciadorInput(pGI GerenciadorInput): void
+pollEventos(): void


TL: Class
«Encadeamento Simples»
List
-pPrimeiro: Node <TL>*
-pUltimo: Node <T>*
-tamanho: int
«create»+Lista()
«destroy»~Lista()
+clear(): void
+comprimento(): int
+append(pInfo: TL*): void
+operator[](x: int): TL*
-setNode(pNode: Node<TL>*): void
-getpPrimeiro(): Node<TL>*
-getpLast(): Node<TL>*
+remove(pTL: tl*): bool
0..1
ListaEntidades
-lista: Lista<Entidade>
«create»+ListaEntidades()
«destroy»~ListaEntidades()
+adicionaEntidade(): void
+removeEntidade(Entidade* pEntidade): Entidade*
+deletaEntidade(Entidade* pEntidade): void
+getTamanho(): int
+limpaLista(): void
1
1
1
1
+armazena
+Conhece Anterior
+Conhece Proximo
0..1
0..1
TE: Class
Node
-pProximo: Node<TE>*
-pAnterior: Node<TE>*
-pInfo: <TE>*
0..1
0..1
«create»+Node()
«destroy»~Node()
+setProximo(pProximo: Node<TE>*): void
+getProximo(): Node<TE>*
+setAnterior(pAnterior: Node<TE>*): void
+getAnterior(): Node<TE>*
+setInfo(pInfo: TE*): void
+getInfo(): TE*
+Conhece Primeiro
+Conhece Ultimo

Animação

-numeroImagem: Vector2u
-textura: Texture
-imagemAtual: Vector2u
-corpo: Rectangleshape
-uvRect: intRect
-tempoTotal: float

«create»+Animacao()
«destroy»+animacao()
+Inicializartextura(): void
+setNumeroImagem(): void
+renderizar(): void
+atualizar(): void

Botao

-hover: bool
+pGerenciadorGrafico: GerenciadorGrafico*
-text: Texto
-font Fonte

«Create»+Button(pos: Vector2f)
«Create»+Button()
«Destroy»~Button(): virtual
+inicializaSprite(): void
+renderiza(): void
+setMensagem(m: const char*): void
+seleciona(selecionado: const bool): void

«Ente»
Menu

-posicao: Coordenadai
-pressionado: bool
-janela: RenderWindow*
-fonte: Font*
-imagem: Texture*
-sprite: Sprite*
-posicaoMouse: Vector2i: Sprite*
-coordenadaMouse: Vector2f: Sprite*

«create»+Menu()
«destroy»~Menu()
+executarMenu(): void

PlanoDeFundo

«Create»+PlanoDeFundo(GG: GerenciadorGrafico, pos: Vector2f, path: const char*)
«Create»+PlanoDeFundo()
«Destroy»~PlanoDeFundo(): virtual
+inicializaSprite(): void
+renderiza(): void
+atualiza(dt: float): void

Tabela de conceitos utilizados

1	**Elementares:**		
1.1	- Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno).	Sim	Todos arquivos .hpp e .cpp.
1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Basicamente em todos arquivos .hpp e .cpp, facilitando a manutenção do código.
1.3	- Classe Principal.	Sim	Main.cpp & Jogo.h/.cpp
1.4	- Divisão em .h e .cpp.	Sim	Em todo o projeto, seguindo boas práticas.

Tabela de conceitos utilizados

2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional.	Sim	Direcional entre Nível come GerenciadorGrafico; Bidirecional entre GerenciadorEstado com GerenciadorInput e GerenciadorGrafico.
2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	Associação entre Menu (e derivadas) e MenuControle, propriamente dita entre Entidade e ListaEntidades.
2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	Elementar entre Entidade e Obstáculos, diversos níveis entre Entidade e Personagens.
2.4	- Herança múltipla.	Sim	Classe de MenuPrincipalState, herança de Estado e Menu.

Tabela de conceitos utilizados

3	**Ponteiros, generalizações e exceções**		
3.1	- Operador *this* para fins de relacionamento bidirecional.	Sim	Múltiplas classes como Animacao e ControleJogador.
3.2	- Alocação de memória (*new* & *delete*).	Sim	Todas as classes que instanciam uma nova Entidade.
3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).	Sim	Lista e ListaEntidade.
3.4	- Uso de Tratamento de Exceções (*try catch*).	Sim	No desenvolvimento do projeto para testar componentes, prontamente retiradas depois.

Tabela de conceitos utilizados

4	**Sobrecarga de:**		
4.1	- Construtoras e Métodos.	Sim	Classe ConstrutorNivel (Construtora) e Animacao (método inicializaTextura)
4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).	Sim	Foi usado o *operator*== e o *operator*++ ***[Em animacao: imagemAtual.x++ e textura ==]***
---	**Persistência de Objetos (via arquivo de texto ou binário)**		
4.3	- Persistência de Objetos.	Sim	Pacote Nível.
4.4	- Persistência de Relacionamento de Objetos.	Sim	Idem anterior (relação entre diferentes Entidades)

Tabela de conceitos utilizados

5	**Virtualidade:**		. . .
5.1	- Métodos Virtuais Usuais.	Sim	Classe Obstaculo, por exemplo.
5.2	- Polimorfismo.	Sim	Chamadas como atualiza e renderiza de Ente em animaçao.
5.3	- Métodos Virtuais Puros / Classes Abstratas.	Sim	Classe Ente.
5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.	Sim	Em todo o projeto.

Tabela de conceitos utilizados

6	**Organizadores e Estáticos**		
6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	Gerenciadores, Entidades, Controle etc..
6.2	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	Classe Lista, por exemplo.
6.3	- Atributos estáticos e métodos estáticos.	Sim	Classe GerenciadorEstado, método getEstadoAtual. Classe ID, com atributos estáticos.
6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	Em todo o Projeto.

Tabela de conceitos utilizados

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	Classe Entidade, por exemplo.
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	GerenciadorEventos (Múltiplos eventos), Nivel (múltiplos níveis).
---	**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	. . .

Tabela de conceitos utilizados

8	**Biblioteca Gráfica / Visual**		
8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: ● tratamento de colisões ● duplo *buffer*	Sim	Carregar e desenhar imagens, texto e duplo buffer utilizando a biblioteca SFML.
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	Classes GerenciadorEvento e Botao, além do uso das funcionalidades de evento do SFML.
---	**Interdisciplinaridades via utilização de Conceitos de <u>Matemática Contínua e/ou Física</u>.**		
8.3	- Ensino Médio Efetivamente.	Sim	Cálculo de velocidade e aceleração, além de coordenadas.
8.4	- Ensino Superior Efetivamente.	Sim	Cálculo de projéteis e vetores.

Tabela de conceitos utilizados

9	**Engenharia de Software**		
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	Progresso com reuniões entre os monitores e professor, além de registros no GitHub e versões do Diagrama de classes.
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.	Sim	Padrões utilizados: Singleton, State, Observer, Builder e Iterator.
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	Reuniões com monitores e professor.

Tabela de conceitos utilizados

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	Repositório no GitHub. https://github.com/Matheussilva05/CastleVania.git
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	*Sim*	1ª: 02/06 2ª: 06/06.
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	***1ª:20/05*** ***2ª:24/05*** ***3ª:29/05*** ***4ª:30/05***
10.4	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Rafael Bergo e Ariel Wilson

Tabela de conceitos utilizados

Total de conceitos apropriadamente utilizados.	**95%** (noventa e cinco por cento).

Fotos do jogo executando

Fotos do jogo executando

Fotos do jogo executando

Fotos do jogo executando

Fotos do jogo executando

Fotos do jogo executando

Fotos do jogo executando

Fotos do jogo executando

Vídeo do jogo executando

Vídeo do jogo executando

Conclusões finais

- Um planejamento adequado antes e durante do desenvolvimento do projeto é crucial.

- Melhor entendimento de paradigmas orientados a objetos.

- Enfatização da importância de padrões de projetos bem implementados.

- Conhecimento sobre como melhor utilizar GitHub e sincronização de áreas de trabalho.

- Fazer jogos dá um trabalho do cão.