TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DA UTFPR TURMA S71

Adryan C. Feres e Thales G. B. dos Santos

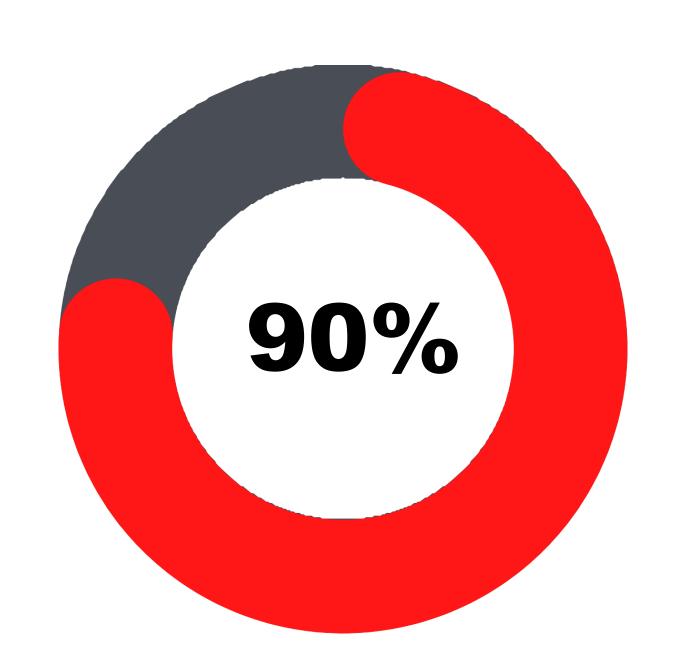
### Sonho++

# Tabela de Requisitos

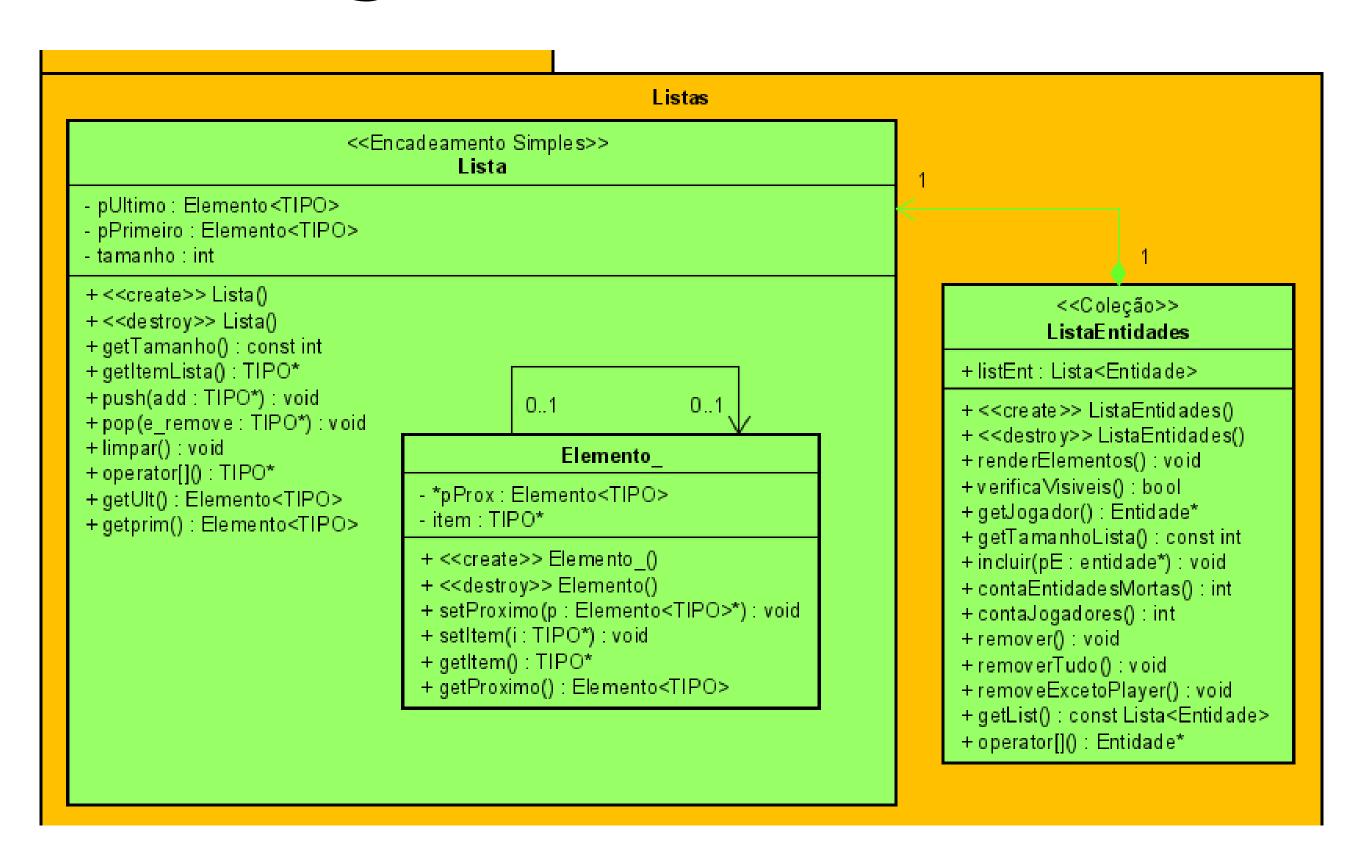
N.	Requisito	Status
1	Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, ver colocação ( <i>ranking</i> ) de jogadores e demais opções pertinentes.	
2	Permitir um ou dois jogadores com representação gráfica aos usuários do Jogo, sendo que no último caso seria para que os dois joguem de maneira concomitante	
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	
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'.	

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.	
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem	
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.	
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.	
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.	
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.	

## Percentual de Requisitos cumpridos



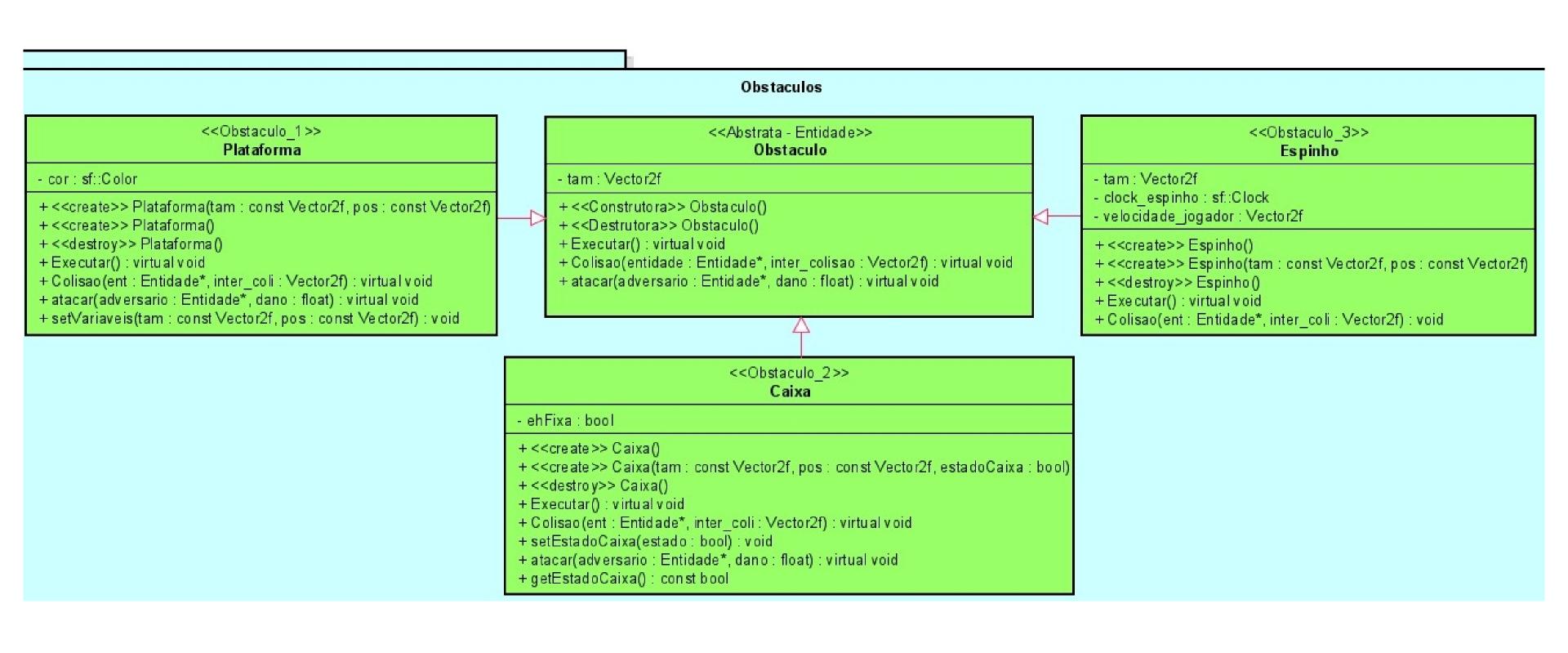
## Diagrama de Classes



#### Personagens

```
<<Abstrata - Entidade>>
                            Personagens
# q_vida : int
# vivo : bool
# atacou : bool
# pode Andar Direita: bool
# olhando Direita : bool
# podeAndarEsquerda : bool
                                                                                                             <<Personagens>>
# olhando Esquerda : bool
                                                                                                                  Jogador
# pode Pular : bool
                                                                                     - tempo : Clock
+ <<create>> Personagens(q_vida:int)
                                                                                     - pontuacao : int
+ <<destroy>> Personagens()
+ set Atacou(ataque : bool) : void
                                                                                     + << create>> Jogador()
+ getAtacou() : bool
                                                                                     + <<destroy>> Jogador()
+ set Quantidade Vida(q:int): void
                                                                                     + andar(i : int) : void
+ getQuantidade \vida() : const int
                                                                                     + Mover() : virtual void
+ const get'vida() : const bool
                                                                                     + Executar(): virtual void
                                                                                     + Colisao(entidade : Entidade*, inter_coli : Vector2f) : virtual void
+ verifica\/ida(): void
                                                                                     + ganha Pontos(): void
+ perde Vida(): void
+ getOlhar() : const \/ector2<bool>
                                                                                     + get Pontos(): int
+ verifica Pode Andar(): void
+ verifica Pode Andar(pos: Vector2f): void
+ getPode Andar() : const Vector2 <bool>
+ pular(tam_pulo : double) : void
+ Executar() : virtual void
+ Mover() : virtual void
+ Colisao(entidade : Entidade*, inter_colisao : Vector2f) : virtual void
+ atacar(adversario : Entidade*, dano : foat) : virtual void
```

<< Personagens >> <<Inimigo\_4>> 1... himigo Anjo <<hi>imigo\_2>> -tempo mov: Clock - alvo : Personagens\* Hydra - projetil : Projetil\* - movRandom : int - pos Alvo : Vector2f raio Deteccao : Vector2f - pos Hydra: Vector2f player : Jogador\* -tempo atk:foat + <<create>> Hydra(Jogador\* p : int) - direcao alvo : string + <<create>> Inimigo() + <<destroy>> Hydra() setAlvo : int + <<destroy>> himigo() + Mover(): void + < <create >> Anjo(p1 : Jogador\*, p2 : Jogador\*, b : Projetil\*) + movAleatorio() : void + Executar() : void + < <destroy> > Anjo() + Mover(): virtual void + inicializa(): void +Mover(): virtual void + Executar() : virtual void + Colisao(entidade : Entidade\*, inter\_colisao : Vector2f) : virtual void + Executar() : virtual void + Colisao(entidade : Entidade\*, inter\_colisao : Vector2f) : virtual + persegue(pos himigo: Vector2f, posJogador: Vector2f): void void +inicializa\/ariaveis(): void + atacar(): virtual void + Encontra PosAlvo(): void + get Alvo (alvo: Personagens\*): const Personagens\* Fantasma : +getProjetil():const Projetil\* tempo\_de\_ataque : foat + atirar\_projetil() : virtual void + Executar(): virtual void + getTempoAtaque() : const foat + <<destroy>> Fantasma() + inicializa \ariaveis(): void + Colisao(entidade : Entidade\*, inter\_colisao : Vector2f) : virtual void + Mover(): virtual void + <<create>> Fantasma()



```
<<Abstrata>>
                             Entidade
#speed: Vector2f
# gravidade : bool
# dano : float
#emCima:bool
#repulsao_direita: Vector2f
# noChao: bool
#posicao: Vector2f
# colisaoPlataforma : bool
#corpo:RectangleShape
#repulsao_esquerda: Vector2f
#texture: Texture
+ setDano(dano : float) : void
+ corrigeColisoes(a : Entidade*, inter : Vector2f) : void
+ setColor(cor : Color) : void
+ getVelocidade() : const Vector2f*
+ <<destro y>> Entidade()
+ setTextura(str : std::string) : void
+ movimentaEntidade(mov : Vector2f) : void
+ Colisao (entidade : Entidade*, inter_colisao : Vector2f) : virtual void
+ setVelocidade(velocidade : Vector2f) : void
+ getDano(): const float
+ <<create >> Entidade(idd : ID)
+ getCorpo(): RectangleShape
+verificarPosInvalida(): const bool
+ render() : void
+ verificarPosInvalidaEmY(): const bool
+ setColisao(a : bool) : void
+ setPosEntidade(pos: Vector2f): void
+ atacar(adversario : Entidade*, dano : float) : virtual void
+ mov Gravidade(): void
+ Executar(): virtual void
+ setColisaoPlataforma(a : bool) : void
```

### <<Entidade>> Projetil

- clock : Clock - tempo : float - repouso : bool

+ <<create>> Projetil()

+ <<destroy>> Projetil() + Mover(): virtual void

+ Executar(): virtual void

+ Colisao(): virtual void

+ getRepouso() : const bool

+ setPortador(ptdr : Personagens\*) : void

+ getPortador() : Personagens\*

+ atacar(dano: float, adv: Entidade\*): virtual void

+ atirar() : void + attPos() : void

+ estado inicial(): void

#### Menus

### <<Abstrata>> Menu

#### # pos : int

- # pressed : bool
- # theselect : bool
- # font : Font\*
- # image : Texture\*
- #bg:Sprite\*

MenuPrincipal

+ <<create>> MenuPrincipal()

+ <<destroy>> MenuPrincipal()

# set\_values():void

+ run\_menu():void

# loop\_events(): void

- # options : vector<const char>
- # coords : vector<Vector2f>
- # texts : vector<Text>
- # sizes : vector<size\_t>
- # draw\_all(): void
- # set\_values() : virtual void
- # loop\_events(): virtual void
- + <<create>> Menu()
- + <<destroy>> Menu()
- + run\_menu() : void
- + Executar(): void

#### MenuPause

- # set\_values(): void
- # loop\_events(): void
- + <<create>> MenuPause()
- + <<destroy>> MenuPause()
- + run\_menu() : virtual void

#### MenuJogadores

 $\wedge$ 

- # set\_values(): void
- # loop\_events(): void
- + <<create>> MenuJogadores()
- + <<destroy>> MenuJogadores()
- + run\_menu(): virtual void

### <<Infraestrutura - SFML - Singleton>> Gerenciador Grafico

- window : RenderWindow\*
- Clock : Clock
- pGrafico : GerenciadorGrafico\*
- <<create>> Gerencia dor Grafico : .
- estado : ID
- + dt : float
- fase1 : bool
- umJogador : bool
- + Gerenciador Grafico()
- + getGerenciadorGrafico(): GerenciadorGrafico\*
- + getWindow() : RenderWindow\*
- + Limpar(): void
- + Desenhar(corpo: Rectangle Shape): void
- + Mostrar() : void
- + FecharJanela(): void
- + isWindowOpen() : bool
- + atualizaTempo(): void
- + setEstado(id : const ID) : void
- + getEstado(): ID
- +verificaEventos(evento: Event): const bool
- + getUmJogador() : const bool
- + setUmJogador(bool t : int) : void
- + setFase1(t : bool) : void + getFase1() : const bool

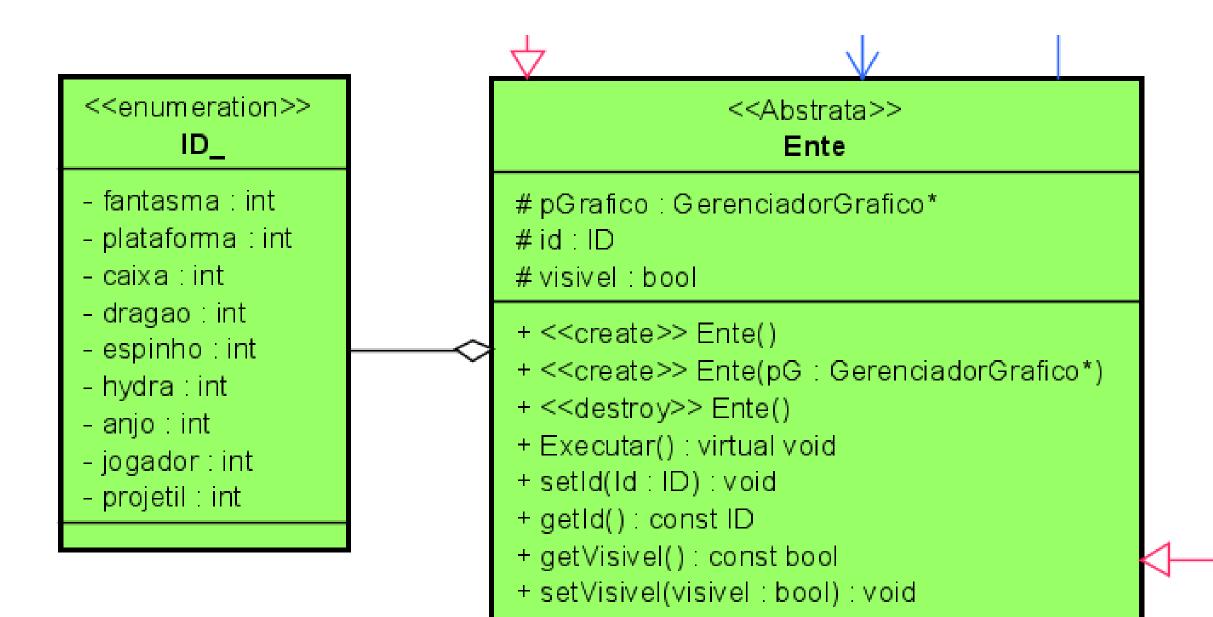
#### Gerenciadores

### <<Mediador - STL>> Gerenciador Colisoes

- lista\_obstaculos : Lista Entidades\*
- lista personagens : Lista Entidades\*
- + <<create>> Gerenciador\_Colisoes(I\_personagem : ListaEntidades\*, I\_obstaculo : ListaEntidades\*)
- + <<destroy>> Gerenciador\_Colisoes()
- + calcula Coliso es(ent1 : Entida de\*, ent2 : Entida de\*) : Vector2f
- +verificaColisoes():void
- + colisao Personagens(): void
- + colisaoPersonagemObstaculos(): void
- + setListas(lper: ListaEntidades\*, lobs: ListaEntidades\*): void
- + Executar(): void
- + colisao Obstaculos(): void

#### GerenciadorEvento

- pGrafico : GerenciadorGrafico\*
- Jogador1 : Jogador\*
- Jogador2 : Jogador\*
- tecla\_pres : int
- pEvento: gerenciadorEvento\*
- gerenciadorEvento():
- + getGerenciadorEvento(): void
- + setJogador1(j: Jogador\*): void
- + setJogador2(j : Jogador\*) : void + verificaTeclaPressionada() : void
- + operation29(): void
- +verificaTeclaPressionada(tecla: Keyboard::Key):void
- +verificaTeclaSolta(tecla: Keyboard::Key):void
- + Executar(): void



#### Fases <<Ab strata>> Fase #j1:Jogador\* #j2:Jogador\* #hydra: Hydra\* # plataforma\_fase : Plataforma\* # caixa: Caixa\* #lista personagem: ListaEntidades\* #lista obstaculos: ListaEntidades\* #pColisoes: Gerencia dorColisoes\* #numero instancias : unsigned int # ativa : bool #criar entidades():virtualvoid <<Fase\_1>> <<Fase\_2>> Fase1 Fase2 +<<create>><<destroy>> Fase() + <<destroy>><<create>> Fase(j1: Jogador\*, j2: Jogador\*) fantasma: Fantasma\* - criar entidades : virtual void + gerarNumAleatorio(max : int, min : int) : int criar\_entidades(): virtual void - anjo : Anjo\* + gerarNumeroAleatorio(max: float, min: float): int - projetil anjo : Projetil\* + <<create>><<destroy>> Fase1() +verificaTerminoFase():void - espinho : Espinho\* + <<destroy>><<create>> Fase1(j1 : Jogador\*, j2 : Jogador\*) + getAtiva() : const bool + <<create>><<destroy>> Fase2(j1 : Jogador\*, j2 : Jogador\*) + criarFantasma(): void + pontuacaoJogador(): void + <<destroy>><<create>> Fase2() + criarHydra():void + Executar(): virtual void + criarLista(): virtual void + criarAnjo(): void + criarCaixa(): void + criarPlataforma(): void + deletaListas(): void + criarCaixa(): void + getListaPersonagem(): ListaEntidades\* + Executar(): virtual void + criarEspinhos(): void + getListaObstaculo(): ListaEntidades\* + criarPlataforma(): void + criar Caixa(): virtual void + Executar(): virtual void + criarPlataforma(): virtual void + criarHydra(): void + cirarHydra(): virtual void

1	< <principal>&gt;</principal>	
•	Jogo	
1	- Jogador1 : Jogador*	
-	- listaPersonagem : int	
	- listaObstaculos : int	
•	- pEvento : int	
•	- pColisoes : int	
	- fase1 : Fase1*	
	- fase2 : Fase2*	
	- menu : MenuPrincipal*	
	- menup : MenuPause*	
	- menuj : MenuJogadores*	
	- qJogadores : int	
	+ < <create>&gt; Jogo()</create>	
	+ < <destroy>&gt; Jogo()</destroy>	
	+ Exectuar() : void	
	+ deletaFase1() : void	
	+ deletaFase2(): void	
	+ iniciaFase1() : void	
	+ iniciaFase2() : void	
	+ controlaFases() : void	
	+ iniciarMenu() : void	

## Tabela de Conceitos

N.	Conceitos	Status
1. 1	- Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno).	
1. 2	- Métodos (com retorno <i>const</i> e parâmetro <i>const</i> ). & - Construtores (sem/com parâmetros) e destrutores	
1. 3	- Classe Principal.	
1. 4	- Divisão em .h e .cpp.	

2.1	- Associação direcional. & - Associação bidirecional.	
2.2	- Agregação via associação. & - Agregação propriamente dita.	
2.3	- Herança elementar. & - Herança em diversos níveis	
2.4	- Herança múltipla.	
3.1	- Operador this para fins de relacionamento bidirecional.	
3.2	- Alocação de memória (new & delete)	
3.3	-Gabaritos/Templates criada/adaptados pelos autores (e.g., Listas Encadeadas via Templates).	
3.4	- Uso de Tratamento de Exceções (try catch).	
4.1	- Construtoras e Métodos.	

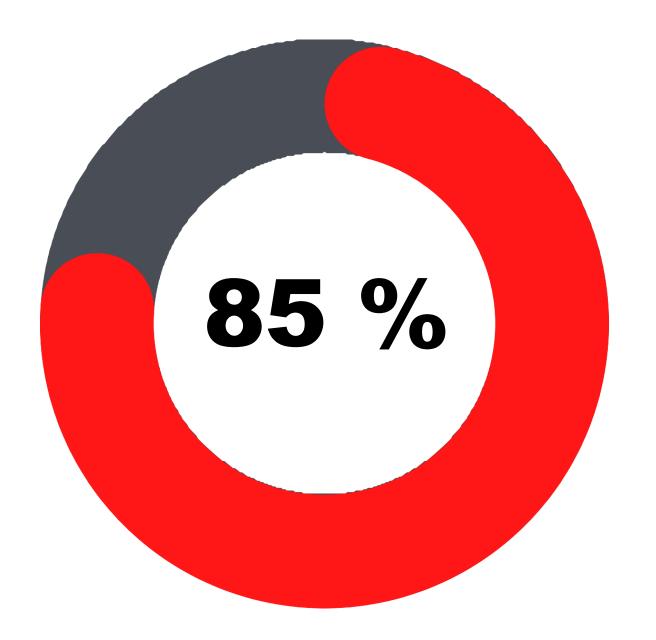
4.2	- Operadores (2 tipos de operadores pelo menos – Quais? ).	
4.3	- Persistência de Objetos.	
4.4	- Persistência de Relacionamento de Objetos.	
5.1	- Métodos Virtuais Usuais.	
5.2	- Polimorfismo.	
5.3	- Métodos Virtuais Puros / Classes Abstratas.	
5.4	Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.	
6.1	- Espaço de Nomes (Namespace) criada pelos autores.	
6.2	- Classes aninhadas (Nested) criada pelos autores.	

6.3	- Atributos estáticos e métodos estáticos.	
6.4	- Uso extensivo de constante (const) parâmetro, retorno, método	
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)	
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa OU Multi-Mapa.	
7.3	- Threads (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run-Time <b>OU</b> Win32API ou afins	
7.4	- Threads (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, <b>OU</b> Troca de mensagens.	
8.1	- Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive) em algum ambiente gráfico. <b>OU</b> - RAD — Rapid Application Development (Objetos gráficos como formulários, botões etc).	
8.2	- Funcionalidades Elementares. & Funcionalidades Avançadas como: tratamento de colisões , duplo buffer	

8.3	Ensino Médio Efetivamente.	
8.4	- Ensino Superior Efetivamente.	
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	
9.2	- Diagrama de Classes em <i>UML</i> .	
9.3	- Uso efetivo e intensivo de padrões de projeto <i>GOF</i> , <i>i.e.</i> , mais de 5 padrões.	
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	
10.1	- Controle de versão de modelos e códigos automatizado (via github e/ou afins). & - Uso de alguma forma de cópia de segurança ( <i>i.e.</i> , <i>backup</i> ).	
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto.	
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto	

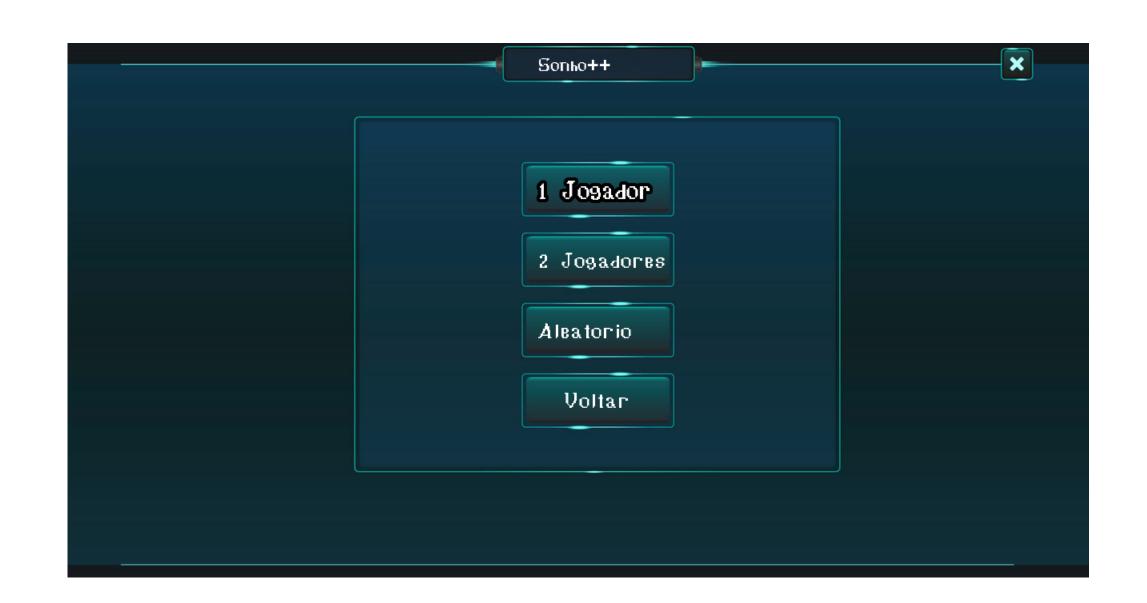


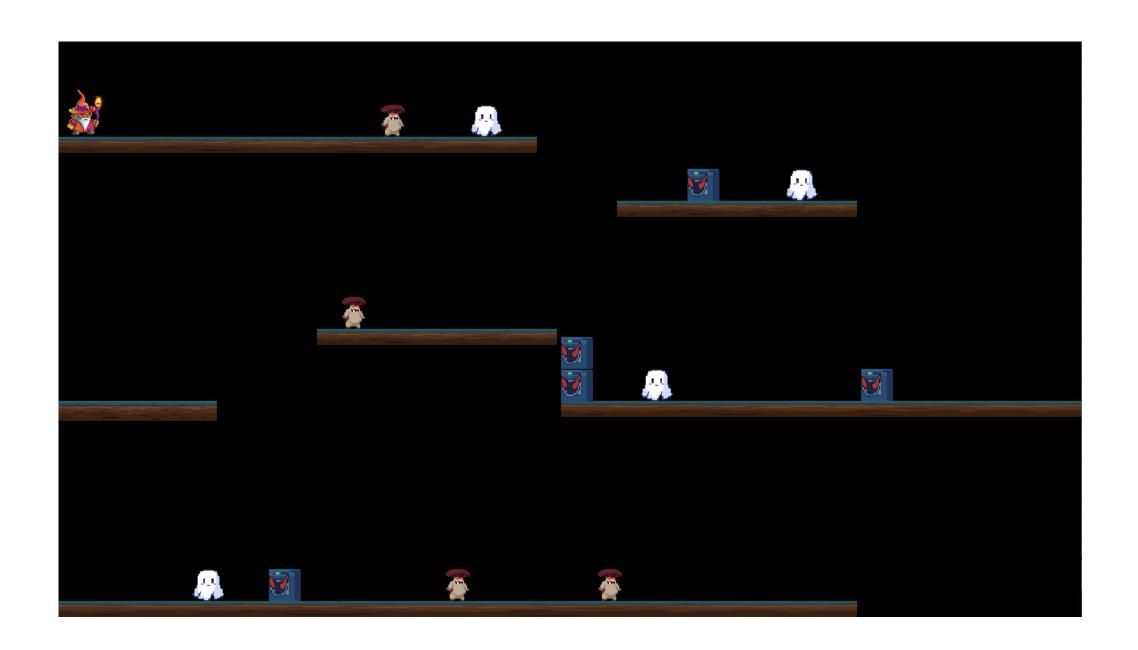
### Percentual de Conceitos cumpridos

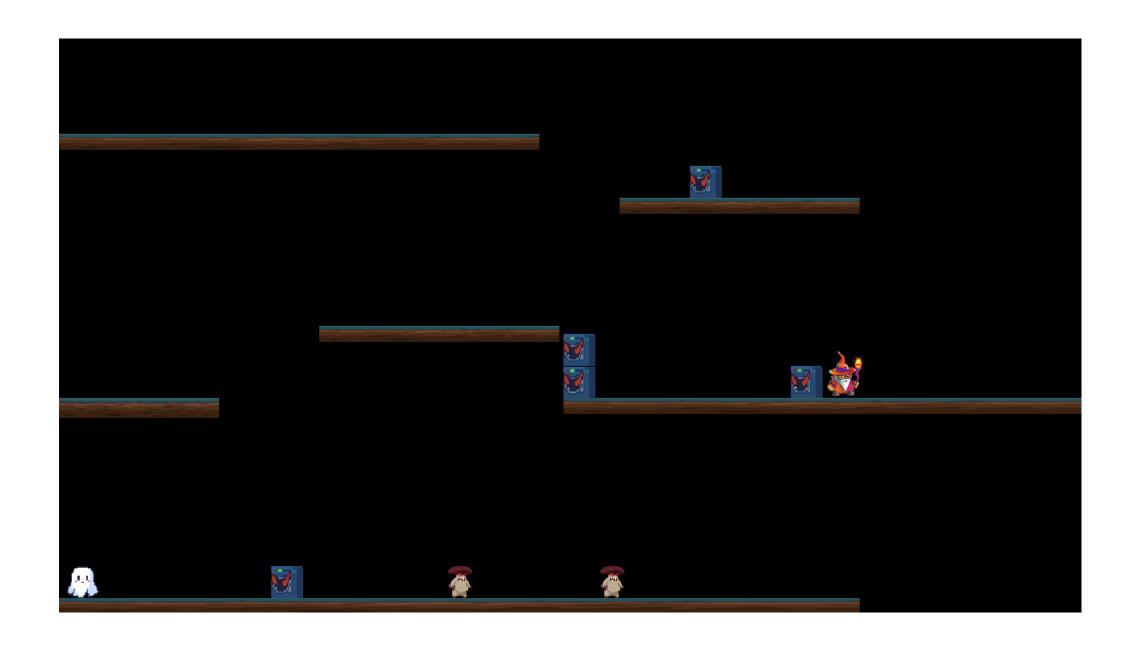


## Jogo Executando



















### Conclusão Geral acerca dos Resultados

Apesar do tempo exíguo, tem-se em mãos um jogo com menus, personagens texturizados, gerenciamento de colisões, gerenciamento de projéteis e mais tarefas relativas a um jogo básico de plataforma. Assim sendo, os membros da equipe consideram o resultado satisfatório.