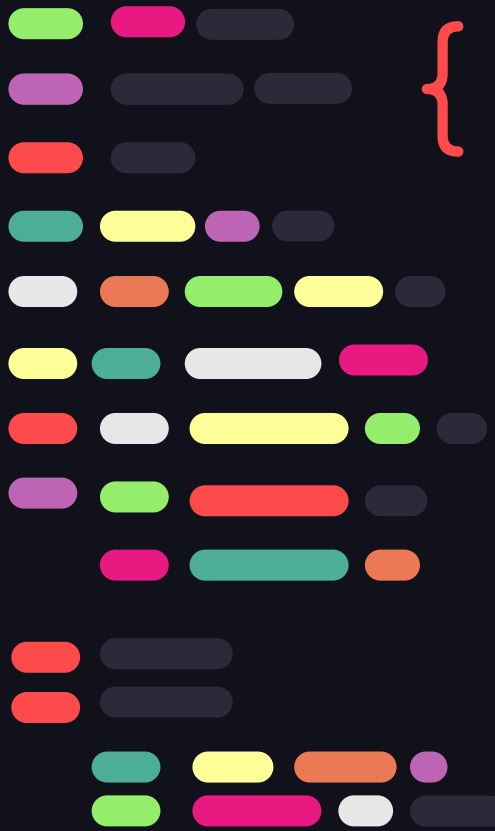


TRABALHO PARA A DISCIPLINA DE TÉCNICAS DE PROGRAMAÇÃO.

< Jogo - **Knight's Quest** >

Daniel Zaki Sommer - danielsommer@alunos.utfpr.edu.br S73
Enzo Westphal Tacla - enzotacla@alunos.utfpr.edu.br S73

Requisitos

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 (previstas nos demais requisitos).	Requisito previsto inicialmente e realizado.	Requisito cumprido com suporte da SFML via Classe Tela e Classe Menu (que funciona como um gerenciador dessas Telas).

Requisito 100% completo


Knight's Quest
Knight's Quest
Novo Jogo
Continuar
Ranking
Sair

Requisitos

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 realizado.	Requisito cumprido inclusive via classe Jogador cujos objetos são agregados em jogo, podendo ter um ou dois por fase criada, ficando a critério do usuário no Menu.

Requisito 100% completo


```
case 2:
    evento = tela1.verificaEventoTela();

    if (evento == 0)
    {
        n_jogadores = 1;
        telaAtual.push(5);
    }
    else if (evento == 1)
    {
        n_jogadores = 2;
        telaAtual.push(5);
    }
    else if (evento == 2)
    {
        popTela();
    }
```

No arquivo Menu.h, dependendo do modo selecionado a variável n_jogadores recebe um valor.
Após isso a fase selecionada é inicializada de acordo com o número de jogadores selecionados, através da função executarFase.

Requisitos

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.	Requisito realizado completamente porque as Classes Floresta e Ruínas (ambas derivadas da classe Fase e, portanto, fases distintas) podem ser executadas via menu ou sequencialmente, quando a primeira é finalizada. Essa lógica de conectá-las é feita via Classe Principal.

Requisito 100% completo

Seleção da fase a parir do menu.

A troca de fase também pode acontecer ao concluir a fase 1 e entrar no portal. Assim o(s) jogador(es) é teleportado à próxima fase.

Requisitos

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 deve ser um 'Chefão'.	Requisito previsto inicialmente e realizado.	Requisito realizado completamente, porque as Classes derivadas de Inimigo: Cogumelo, Olho Voador e Chefão, representam diferentes inimigos com diferentes comportamentos, atributos e funcionalidades.

Requisito 100% completo

Inimigos

- Olho Voador: São criaturas que voam e não colidem com as plataformas. Além de causar dano ao jogador causam lentidão e diminuem a força do pulo.
- Cogumelo: Os cogumelos colidem com as plataformas e envenenam o jogador por um determinado tempo, além disso há a chance de nascerem cogumelos especiais que possuem um veneno mais forte.
- Chefão: Só nasce na fase 2, possui 3 estados: bravo, muito bravo e enfurecido. Em cada um deles atira seus projéteis em uma velocidade diferente. Ao perder uma certa quantidade de vida, o Chefão se teletransporta através de portais.

Requisitos

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.	Requisito previsto inicialmente e realizado.	Requisito atendido, pois os tipos de inimigos e o número máximo permitido para cada tipo estão definidos nos arquivos .txt de cada fase. A criação aleatória de entidades é realizada por meio da função "bool Aleatorizar(char character)", que tem uma probabilidade de 1/3 de retornar "false". Isso impede a criação de uma entidade que já tenha excedido um número mínimo pré-definido (o mínimo é de 3 instâncias para cada tipo distinto de inimigo).

Requisito 100% completo

Instancia de entidades

Cada fase tem um arquivo .txt, tendo a versão para 1 ou 2 jogadores. As entidades são instanciadas por meio da associação de um determinado caractere com uma posição.

Para que o número de instâncias seja aleatório existe uma função aleatorizar, que atribui uma probabilidade de 33% de uma entidade não ser instanciada, porém sempre respeitando a quantidade mínima de inimigos e obstáculos previsto nos requisitos, através de variáveis que realizam a contagem.

Requisitos

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 realizado.	Requisito realizado completamente, porque as Classes derivadas de Obstáculo: Serra, Espinho e Slime representam diferentes obstáculos com diferentes comportamentos, atributos e funcionalidades.

Requisito 100% completo

Obstáculos

- Espinho: Os espinhos causam dano ao jogador quando são tocados, causando um dano instantâneo e outro por envenenamento.
- Serra: As serras são estruturas que causam dano ao jogador ao haver uma colisão entre os dois.
- Slime: O slime não causa dano ao jogador, porém deixa os movimentos do jogador debilitados, afeta tanto o pulo quando o caminhar.
- Parede e Plataformas: Além dos 3 obstáculos as fases contam com outras 2 estruturas de level design que não impactam diretamente na vida do jogador, mas colidem com ele.
- Portal: Utilizado pelo Chefão para se teletransportar para diferentes locais do mapa e também como artifício na troca de fase.

Requisitos

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 previsto inicialmente e realizado.	Requisito atendido, pois os tipos de obstáculos e o número máximo permitido para cada tipo estão definidos nos arquivos .txt de cada fase. A criação aleatória de entidades é realizada por meio da função "bool Aleatorizar(char character)", que tem uma probabilidade de 1/3 de retornar "false". Isso impede a criação de uma entidade que já tenha excedido um número mínimo pré-definido (o mínimo é de 3 instâncias para cada tipo distinto de obstáculo).

Requisito 100% completo

Requisitos

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 realizado.	O requisito é satisfeito pelas classes "Plataforma" e "Parede" dentro do namespace "Obstáculos", que, em conjunto com o "Gerenciador de Colisões", desempenham um papel crucial na construção das fases e na dinâmica do movimento tanto dos jogadores quanto dos inimigos presentes no jogo.

Requisito 100% completo

Fase 1 - Floresta

Fase 2 - Ruínas

Requisitos

9	Gerenciar colisões entre jogador para com inimigos e seus projéteis, 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.	Requisito previsto inicialmente e realizado.	O requisito é atendido pela classe "Personagem" ao aplicar a gravidade. Por sua vez, a classe "Gerenciador_Colisoes" atua como um componente essencial ao oferecer mecanismos para lidar com colisões entre diferentes entidades no jogo. Ele facilita a interação entre

Requisito 100% completo

Gravidade e Colisões

```
void Personagem::cair()
{
    if (!voador)
    {
        // Gravidade
        if (isJumping)
        {
            direcao.y += gravity;
            direcao.y += velocity.y;
            if (gravity >= 0.18f)
                gravity = 0.18f;
            else
                gravity += 0.005f;
        }
        else
        {
            gravity = GRAVIDADE;
            direcao.y += gravity;
        }

        corpo.move(0.0, direcao.y);
    }

}
```

Ação da gravidade

Para as colisões são percorridos 3 loops. O primeiro que verifica a colisão de um jogador com um obstáculo e caso haja intersecção é dado dano no jogador

O segundo verifica a colisão dos personagens com as plataformas.

E o terceiro que calcula a distância entre jogadores e inimigos para que os inimigos ataquem os jogadores.

Requisitos

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/Recuperar** Jogada.	Requisito previsto inicialmente, mas realizado apenas PARCIALMENTE.	Tal requisito foi realizado PARCIALMENTE pois o sistema possui um mecanismo de Salvar as informações das Entidades em até 3 arquivos por Classe, porém, não é possível recuperar esses dados durante a execução do Jogo.

Requisito 50% completo

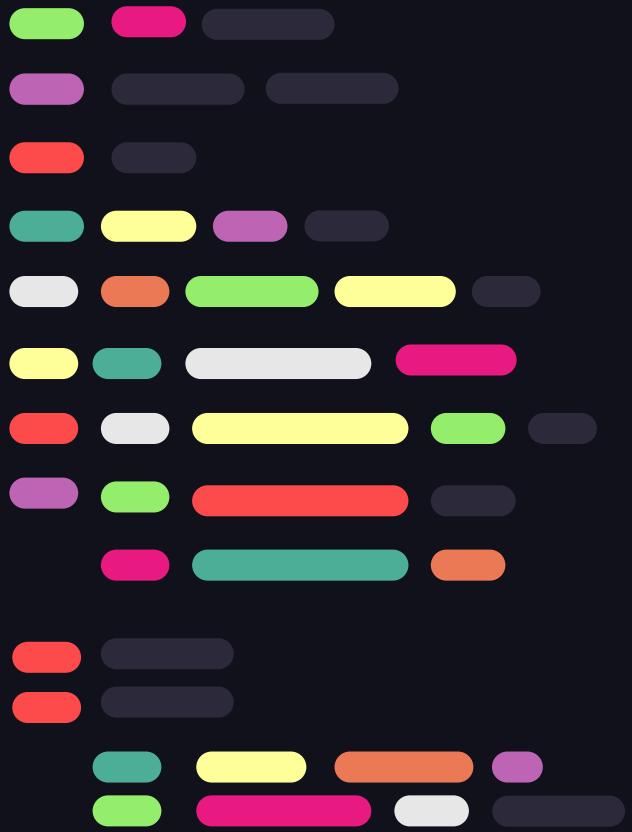
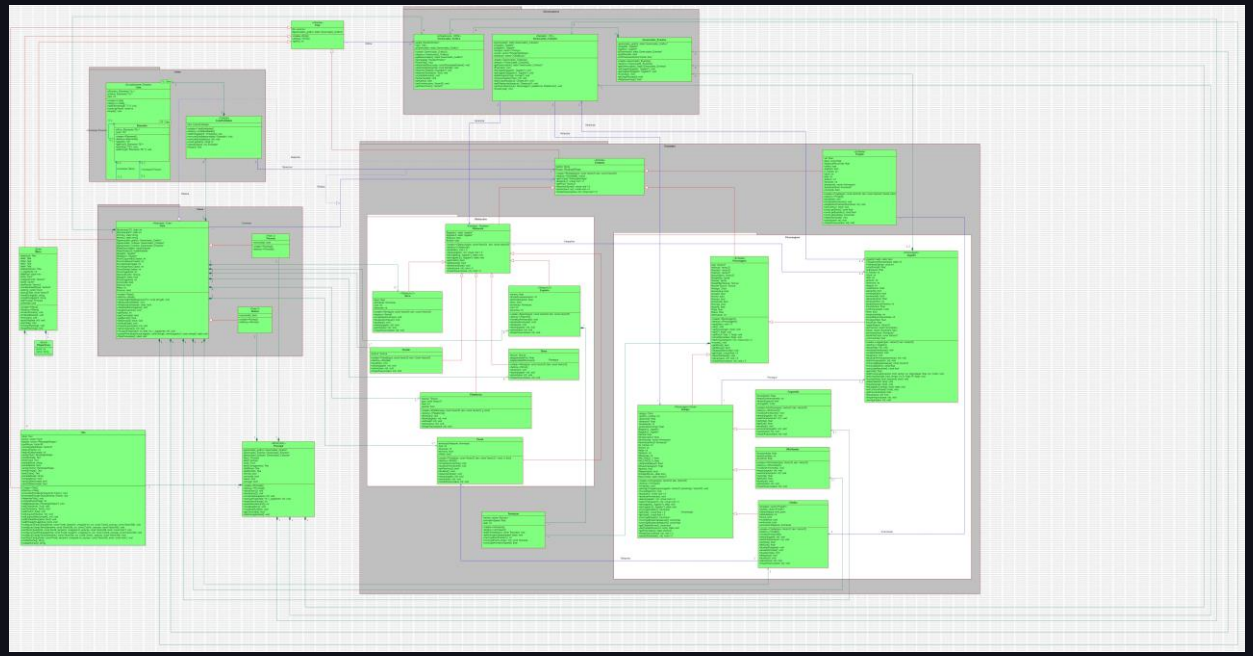
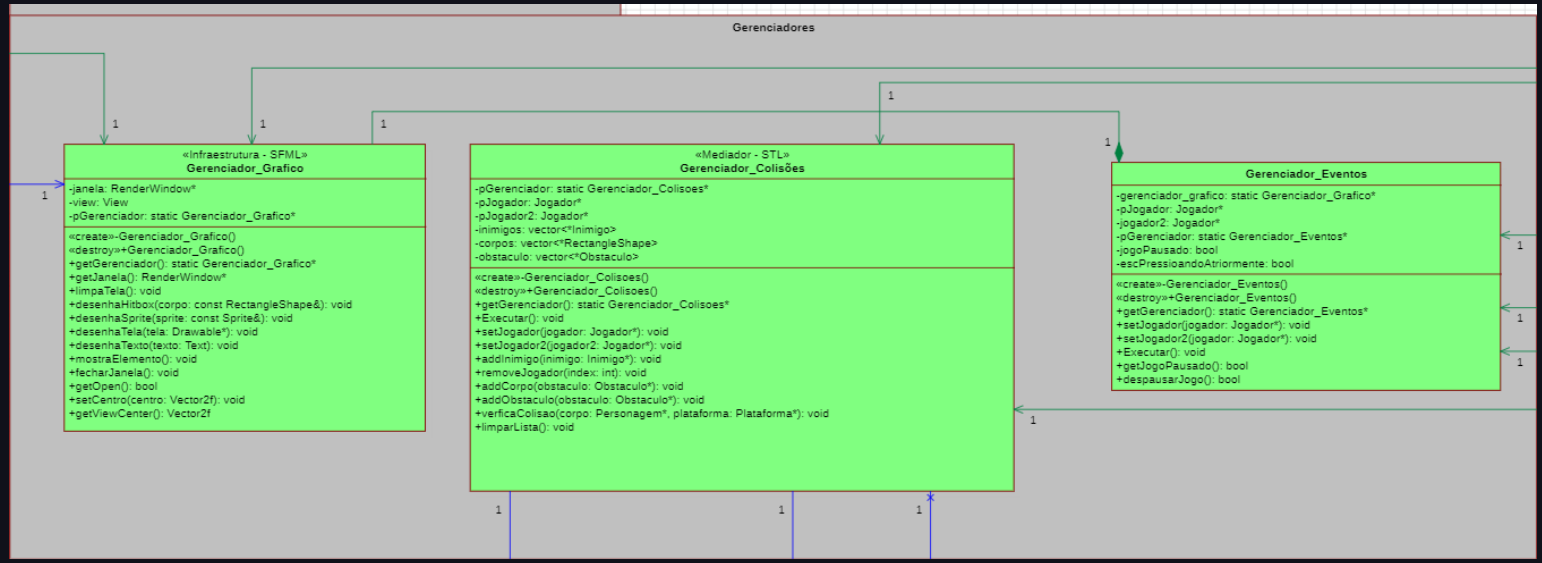


Diagrama de Classes

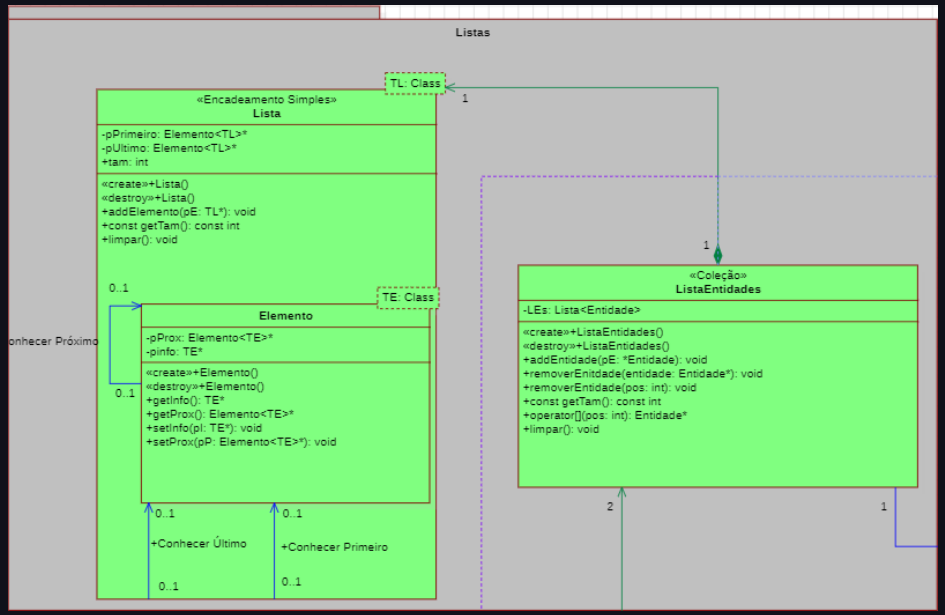
Diagrama de Classes



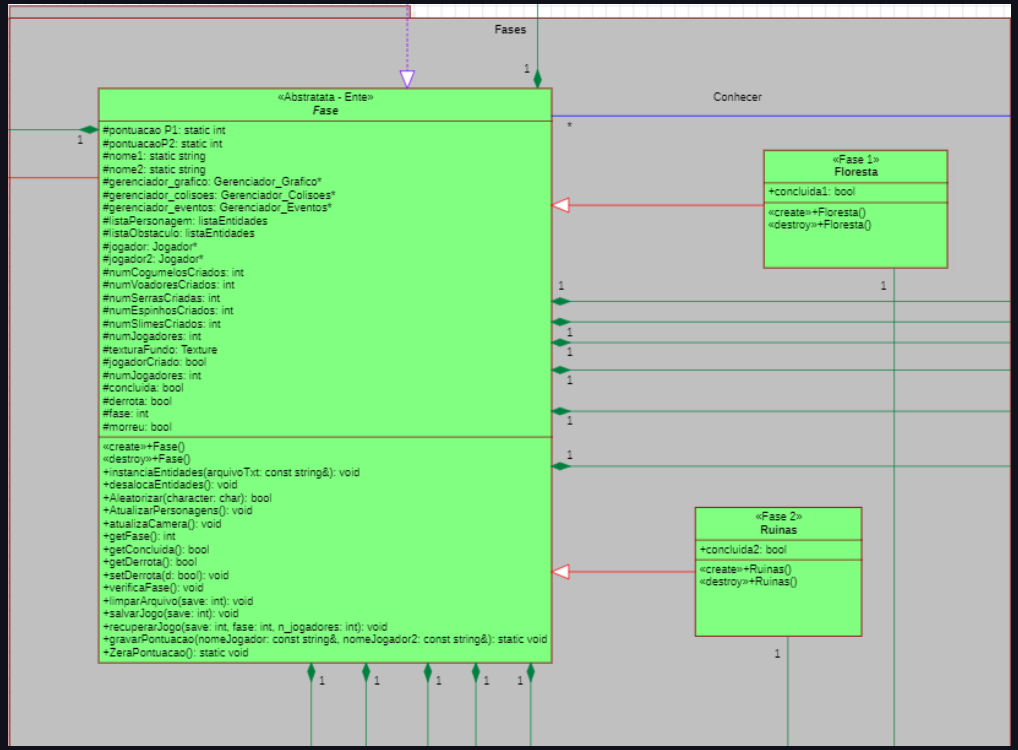
Gerenciadores



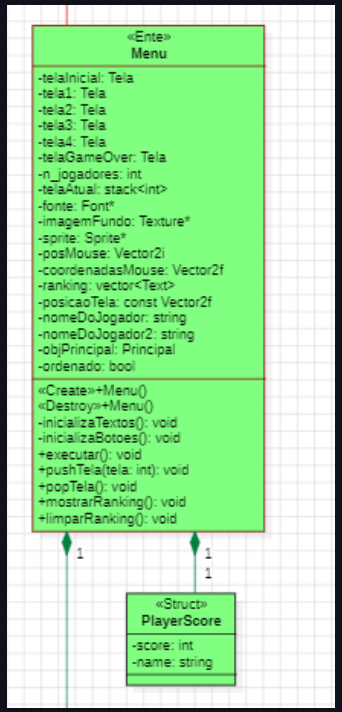
Listas

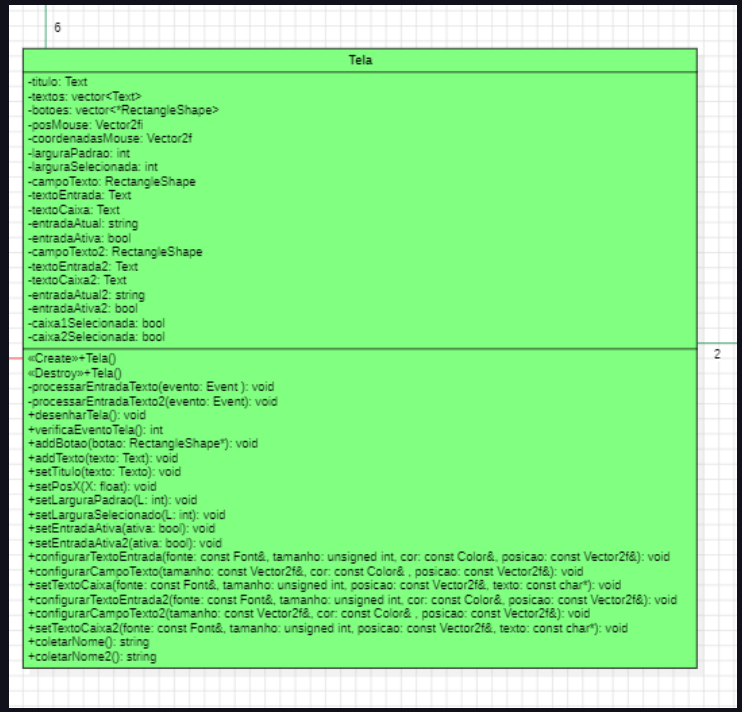

Listas
TL: Class
«Encadeamento Simples»
Lista
-pPrimeiro: Elemento<TL>*
-pUltimo: Elemento<TL>*
+tam: int
«create»+Lista()
«destroy»+Lista()
+addElemento(pE: TL*): void
+const getTam(): const int
+limpar(): void
1
1
«Coleção»
ListaEntidades
-LEs: Lista<Entidade>
«create»+ListaEntidades()
«destroy»+ListaEntidades()
+addEntidade(pE: *Entidade): void
+removerEnitdade(entidade: Entidade*): void
+removerEntidade(pos: int): void
+const getTam(): const int
+operator[](pos: int): Entidade*
+limpar(): void
0..1
TE: Class
Elemento
-pProx: Elemento<TE>*
-pInfo: TE*
«create»+Elemento()
«destroy»+Elemento()
+getInfo(): TE*
+getProx(): Elemento<TE>*
+setInfo(pI: TE*): void
+setProx(pP: Elemento<TE>*): void
onhecer Próximo
0..1
0..1
0..1
+Conhecer Último
0..1
0..1
+Conhecer Primeiro
0..1
2
1

Fases

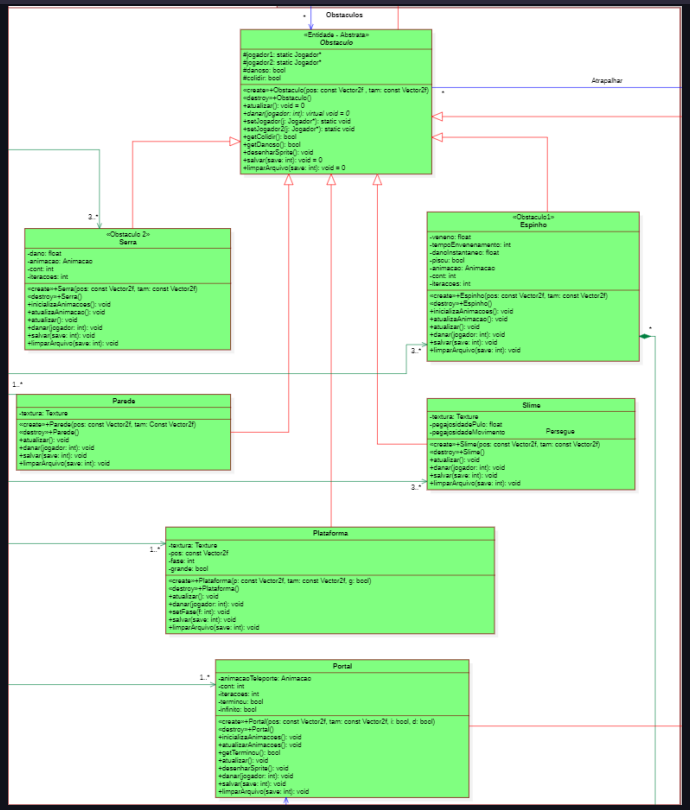

Fases
1
Conhecer
*
«Abstratata - Ente»
Fase
#pontuacao P1: static int
#pontuacaoP2: static int
#nome1: static string
#nome2: static string
#gerenciador_grafico: Gerenciador_Grafico*
#gerenciador_colisoes: Gerenciador_Colisoes*
#gerenciador_eventos: Gerenciador_Eventos*
#listaPersonagem: listaEntidades
#listaObstaculo: listaEntidades
#jogador: Jogador*
#jogador2: Jogador*
#numCogumelosCriados: int
#numVoadoresCriados: int
#numSerrasCriadas: int
#numEspinhosCriados: int
#numSlimesCriados: int
#numJogadores: int
#texturaFundo: Texture
#jogadorCriado: bool
#numJogadores: int
#concluida: bool
#derrota: bool
#fase: int
#morreu: bool
«create»+Fase()
«destroy»+Fase()
+instanciaEntidades(arquivoTxt: const string&): void
+desalocaEntidades(): void
+Aleatorizar(character: char): bool
+AtualizarPersonagens(): void
+atualizaCamera(): void
+getFase(): int
+getConcluida(): bool
+getDerrota(): bool
+setDerrota(d: bool): void
+verificaFase(): void
+limparArquivo(save: int): void
+salvarJogo(save: int): void
+recuperarJogo(save: int, fase: int, n_jogadores: int): void
+gravarPontuacao(nomeJogador: const string&, nomeJogador2: const string&): static void
+ZeraPontuacao(): static void
«Fase 1»
Floresta
+concluida1: bool
«create»+Floresta()
«destroy»+Floresta()
«Fase 2»
Ruinas
+concluida2: bool
«create»+Ruinas()
«destroy»+Ruinas()

Menu e Tela

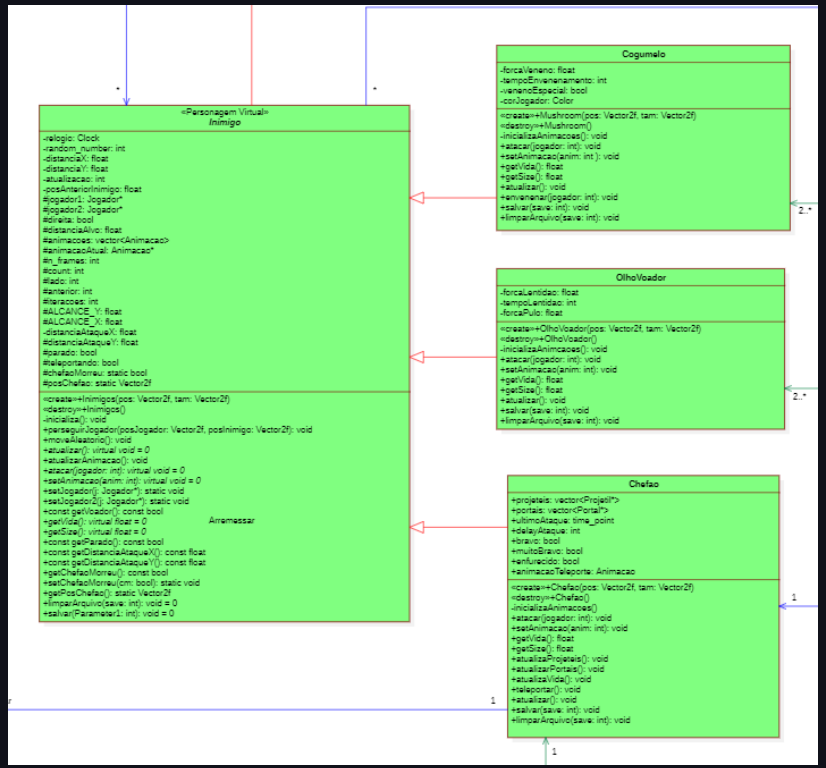

«Ente»
Menu
-telaInicial: Tela
-tela1: Tela
-tela2: Tela
-tela3: Tela
-tela4: Tela
-telaGameOver: Tela
-n_jogadores: int
-telaAtual: stack<int>
-fonte: Font*
-imagemFundo: Texture*
-sprite: Sprite*
-posMouse: Vector2i
-coordenadasMouse: Vector2f
-ranking: vector<Text>
-posicaoTela: const Vector2f
-nomeDoJogador: string
-nomeDoJogador2: string
-objPrincipal: Principal
-ordenado: bool
«Create»+Menu()
«Destroy»+Menu()
-inicializaTextos(): void
-inicializaBotoes(): void
+executar(): void
+pushTela(tela: int): void
+popTela(): void
+mostrarRanking(): void
+limparRanking(): void
1
1
1
«Struct»
PlayerScore
-score: int
-name: string


6
Tela
-titulo: Text
-textos: vector<Text>
-botoes: vector<*RectangleShape>
-posMouse: Vector2fi
-coordenadasMouse: Vector2f
-larguraPadrao: int
-larguraSelecionada: int
-campoTexto: RectangleShape
-textoEntrada: Text
-textoCaixa: Text
-entradaAtual: string
-entradaAtiva: bool
-campoTexto2: RectangleShape
-textoEntrada2: Text
-textoCaixa2: Text
-entradaAtual2: string
-entradaAtiva2: bool
-caixa1Selecionada: bool
-caixa2Selecionada: bool
«Create»+Tela()
«Destroy»+Tela()
-processarEntradaTexto(evento: Event): void
-processarEntradaTexto2(evento: Event): void
+desenharTela(): void
+verificaEventoTela(): int
+addBotao(botao: RectangleShape*): void
+addTexto(texto: Text): void
+setTitulo(texto: Texto): void
+setPosX(X: float): void
+setLarguraPadrao(L: int): void
+setLarguraSelecionado(L: int): void
+setEntradaAtiva(ativa: bool): void
+setEntradaAtiva2(ativa: bool): void
+configurarTextoEntrada(fonte: const Font&, tamanho: unsigned int, cor: const Color&, posicao: const Vector2f&): void
+configurarCampoTexto(tamanho: const Vector2f&, cor: const Color& , posicao: const Vector2f&): void
+setTextoCaixa(fonte: const Font&, tamanho: unsigned int, posicao: const Vector2f&, texto: const char*): void
+configurarTextoEntrada2(fonte: const Font&, tamanho: unsigned int, cor: const Color&, posicao: const Vector2f&): void
+configurarCampoTexto2(tamanho: const Vector2f&, cor: const Color& , posicao: const Vector2f&): void
+setTextoCaixa2(fonte: const Font&, tamanho: unsigned int, posicao: const Vector2f&, texto: const char*): void
+coletarNome(): string
+coletarNome2(): string
2

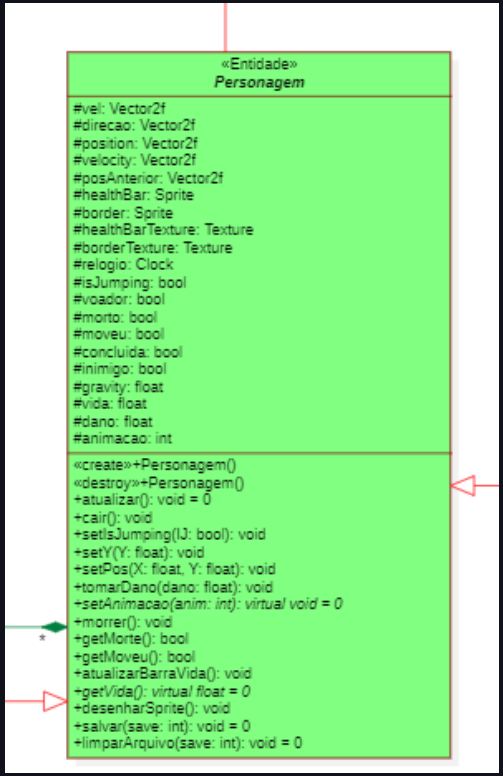
Obstáculos

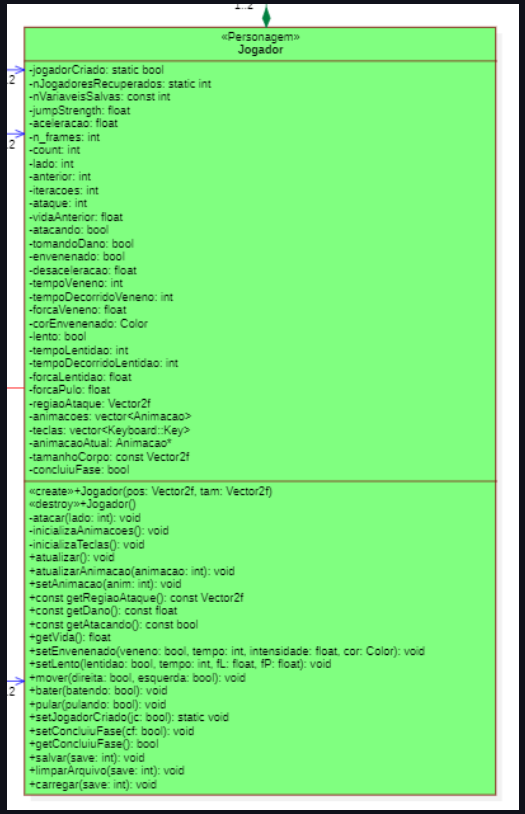

Obstaculos
«Entidade - Abstrata»
Obstaculo
#jogador1: static Jogador*
#jogador2: static Jogador*
#dano: int
#ativo: bool
#colidir: bool
+create+Obstaculo(pos: const Vector2f, tam: const Vector2f)
+destroy+Obstaculo()
+atualizar(): void = 0
+danar(jogador: int): virtual void = 0
+setJogador(): Jogador*): static void
+setJogador2(): Jogador*): static void
+getColidir(): bool
+getDanoso(): bool
+desenharSprite(): void
+salvar(save: int): void = 0
+limparArquivo(save: int): void = 0
Atrapalhar
«Obstaculo 2»
Serra
-dano: float
-animacao: Animacao
-cont: int
-iteracoes: int
+create+Serra(pos: const Vector2f, tam: const Vector2f)
+destroy+Serra()
+inicializaAnimacoes(): void
+atualizaAnimacao(): void
+atualizar(): void
+danar(jogador: int): void
+salvar(save: int): void
+limparArquivo(save: int): void
«Obstaculo1»
Espinho
-veneno: float
-tempoEnvenenamento: int
-danoInstantaneo: float
-ativo: bool
-animacao: Animacao
-cont: int
-iteracoes: int
+create+Espinho(pos: const Vector2f, tam: const Vector2f)
+destroy+Espinho()
+inicializaAnimacoes(): void
+atualizaAnimacao(): void
+atualizar(): void
+danar(jogador: int): void
+salvar(save: int): void
+limparArquivo(save: int): void
Parede
-textura: Textura
+create+Parede(pos: const Vector2f, tam: Const Vector2f)
+destroy+Parede()
+atualizar(): void
+danar(jogador: int): void
+salvar(save: int): void
+limparArquivo(save: int): void
Slime
-textura: Textura
-pegajosidadePulo: float
-pegajosidadeMovimento
Perseguia
+create+Slime(pos: const Vector2f, tam: const Vector2f)
+destroy+Slime()
+atualizar(): void
+danar(jogador: int): void
+salvar(save: int): void
+limparArquivo(save: int): void
Plataforma
-textura: Textura
-pos: const Vector2f
-fase: int
-grande: bool
+create+Plataforma(p: const Vector2f, tam: const Vector2f, g: bool)
+destroy+Plataforma()
+atualizar(): void
+danar(jogador: int): void
+setFase1(int): void
+salvar(save: int): void
+limparArquivo(save: int): void
Portal
-animacaoTeleporte: Animacao
-cont: int
-iteracoes: int
-jaminou: bool
-infinito: bool
+create+Portal(pos: const Vector2f, tam: const Vector2f, i: bool, d: bool)
+destroy+Portal()
+inicializaAnimacoes(): void
+atualizaAnimacoes(): void
+getTerminou(): bool
+atualizar(): void
+desenharSprite(): void
+danar(jogador: int): void
+salvar(save: int): void
+limparArquivo(save: int): void

Inimigos

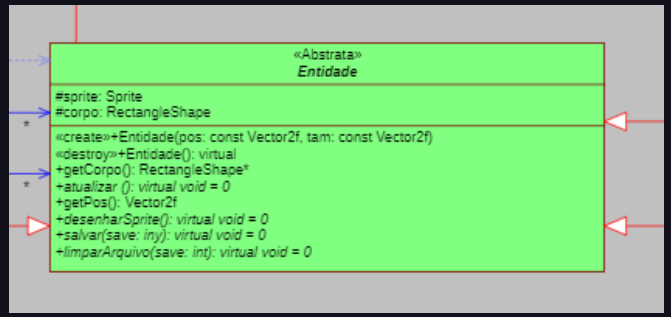

«Personagem Virtual»
Inimigo
-relogio: Clock
-random_number: int
-distanciaX: float
-distanciaY: float
-atualizacao: int
-posAnteriorInimigo: float
#jogador1: Jogador*
#jogador2: Jogador*
#direita: bool
#distanciaAlvo: float
#animacoes: vector<Animacao>
#animacaoAtual: Animacao*
#n_frames: int
#count: int
#lado: int
#anterior: int
#interacoes: int
#ALCANCE_Y: float
#ALCANCE_X: float
-distanciaAtaqueX: float
#distanciaAtaqueY: float
#parado: bool
#teleportando: bool
#chefaoMorreu: static bool
#posChefao: static Vector2f
+create+Inimigos(pos: Vector2f, tam: Vector2f)
+destroy+Inimigos()
-inicializa(): void
+perseguirJogador(posJogador: Vector2f, posInimigo: Vector2f): void
+moveAleatorio(): void
+atualizar(): virtual void = 0
+atualizarAnimacao(): void
+atacar(jogador: int): virtual void = 0
+setAnimacao(anim: int): virtual void = 0
+setJogador(j: Jogador*): static void
+setJogador2(j: Jogador*): static void
+const getVoador(): const bool
+getVida(): virtual float = 0
Arremessar
+getSize(): virtual float = 0
+const getParado(): const bool
+const getDistanciaAtaqueX(): const float
+const getDistanciaAtaqueY(): const float
+getChefaoMorreu(): const bool
+setChefaoMorreu(cm: bool): static void
+getPosChefao(): static Vector2f
+limparArquivo(save: int): void = 0
+salvar(Parameter1: int): void = 0
Cogumelo
-forcaVeneno: float
-tempoEnvenenamento: int
-venenoEspecial: bool
-corJogador: Color
+create+Mushroom(pos: Vector2f, tam: Vector2f)
+destroy+Mushroom()
-inicializaAnimacoes(): void
+atacar(jogador: int): void
+setAnimacao(anim: int): void
+getVida(): float
+getSize(): float
+atualizar(): void
+envenenar(jogador: int): void
+salvar(save: int): void
+limparArquivo(save: int): void
OlhoVoador
-forcaLentidao: float
-tempoLentidao: int
-forcaPulo: float
+create+OlhoVoador(pos: Vector2f, tam: Vector2f)
+destroy+OlhoVoador()
-inicializaAnimacoes(): void
+atacar(jogador: int): void
+setAnimacao(anim: int): void
+getVida(): float
+getSize(): float
+atualizar(): void
+salvar(save: int): void
+limparArquivo(save: int): void
Chefao
+projeteis: vector<Projetil*>
+portais: vector<Portal*>
+ultimoAtaque: time_point
+delayAtaque: int
+bravo: bool
+muitoBravo: bool
+enfurecido: bool
+animacaoTeleporte: Animacao
+create+Chefao(pos: Vector2f, tam: Vector2f)
+destroy+Chefao()
-inicializaAnimacoes()
+atacar(jogador: int): void
+setAnimacao(anim: int): void
+getVida(): float
+getSize(): float
+atualizaProjeteis(): void
+atualizaPortais(): void
+atualizaVida(): void
+teleportar(): void
+atualizar(): void
+salvar(save: int): void
+limparArquivo(save: int): void
2..*
2..*
1
1
1

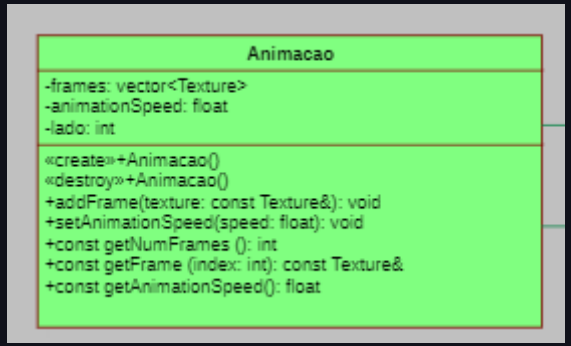
Personagem e Jogador

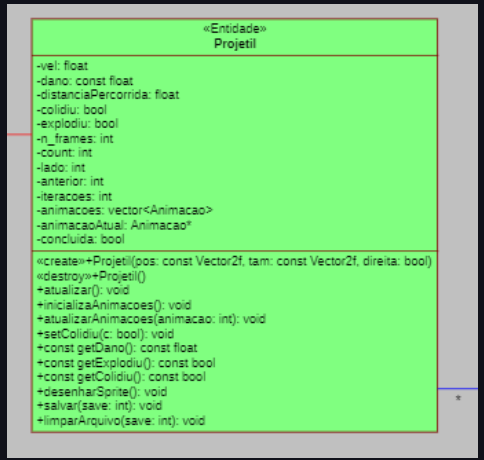




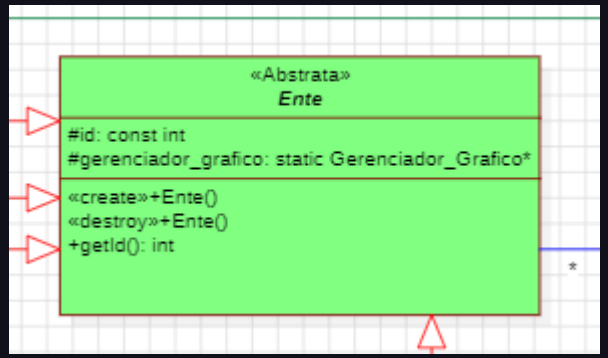
Outras entidades

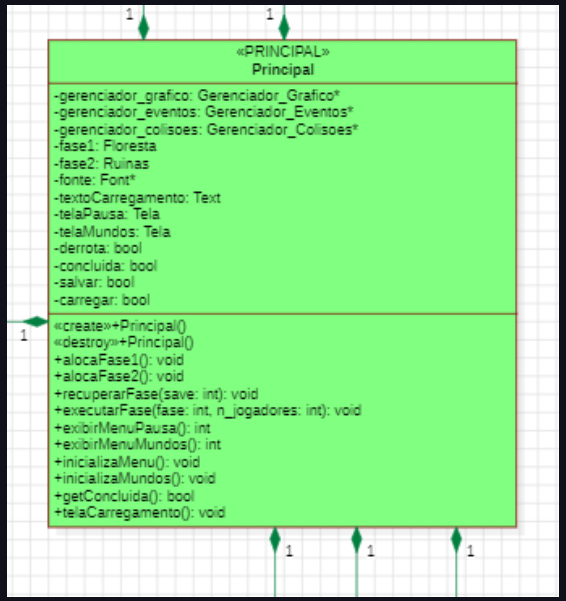






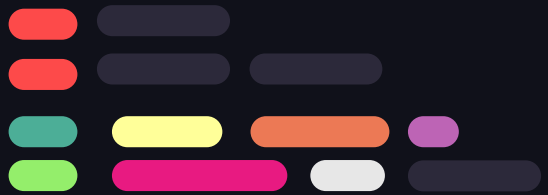
Ente e Principal





01 ..

Tabela de Conceitos



} ..

Conceitos

Conceitos 100% concluídos

1	**Elementares:**		
1.1	- Classes, objetos. & - Atributos (privados), variáveis e constantes. & - Métodos (com e sem retorno).	Sim	Todos .hpp e .cpp, como nas classes Jogador e Fase, nos *namespaces* Entidades e Fases, respectivamente.
1.2	- Métodos (com retorno *const* e parâmetro *const*). & - Construtores (sem/com parâmetros) e destrutores	Sim	Na maioria dos .hpp e .cpp, como na classe Jogador no *namespace* Entidades.
1.3	- Classe Principal.	Sim	Main.cpp & Principal.hpp/.cpp
1.4	- Divisão em .h e .cpp.	Sim	Em todo o desenvolvimento, cada classe implementada foi separada em .hpp e .cpp.

Conceitos

Conceitos 100% concluídos

2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional.	Sim	Em vários dos .hpp e .cpp, como nas classes Inimigo e Jogador.
2.2	- Agregação via associação. & - Agregação propriamente dita.	Sim	Em vários dos .hpp e .cpp, como nas classes Fase com as classes Obstáculo e Entidades.
2.3	- Herança elementar. & - Herança em diversos níveis.	Sim	Em alguns dos .hpp e .cpp, como nas classes nos *namespace Entidades*.
2.4	- Herança múltipla.	Sim	Precisamente nos .hpp e .cpp, das classes OlhoVoador, Cogumelo e Chefao.

Conceitos

Conceitos 75% concluídos

3	**Ponteiros, generalizações e exceções**		
3.1	- Operador *this* para fins de relacionamento bidirecional.	Sim	Como nos 3 Gerenciadores (Gráfico, de Colisões e de Eventos).
3.2	- Alocação de memória (*new & delete*).	Sim	Como na classe Menu.
3.3	- Gabaritos/*Templates* criada/adaptados pelos autores (*e.g.*, Listas Encadeadas via *Templates*).	Sim	Como nas Classes Lista e ListaEntidade, do *namespace* Lista.
3.4	- Uso de Tratamento de Exceções (*try catch*).	Não	

Conceitos

Conceitos 0% concluído

4	**Sobrecarga de:**		
4.1	- Construtoras e Métodos.	Não	
4.2	- Operadores (2 tipos de operadores pelo menos – Quais?).	Não	
---	**Persistência de Objetos (via arquivo de texto ou binário)**		
4.3	**- Persistência de Objetos.**	Não	Parcialmente. Os objetos têm a capacidade de armazenar seus atributos em arquivos de salvamento, porém não os recuperam.
4.4	**- Persistência de Relacionamento de Objetos.**	Não	

Conceitos

Conceitos 100% concluídos

5	**Virtualidade:**		
5.1	- Métodos Virtuais Usuais.	Sim	Como na Classe Entidade.
5.2	- Polimorfismo.	Sim	Como na Classe Entidade e suas derivadas.
5.3	- Métodos Virtuais Puros / Classes Abstratas.	Sim	Como na Classe Ente e Entidade.
5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto.	Sim	Como nas Classes do *namespace* Gerenciadores, que foi utilizado o padrão Singleton.

Conceitos

Conceitos 50% concluídos

6	**Organizadores e Estáticos**		
6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.	Não	Foram utilizados apenas os *Namespaces* pré-determinados pelo modelo do professor.
6.2	- Classes aninhadas (*Nested*) criada pelos autores.	Não	
6.3	- Atributos estáticos e métodos estáticos.	Sim	Como na Classe Fase e na Classe Jogador.
6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	Como na Classe Fase e na Classe Jogador.

Conceitos

Conceitos 50% concluídos

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	Como nas classes Personagem ou na Classe Chefão, ambas do *namespace* Entidades::Personagens.
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	Como na Classe ListaEntidade, na Classe Fase e na Classe Gerenciador_Colisoes.
---	**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	

Conceitos

Conceitos 50% concluídos

8	**Biblioteca Gráfica / Visual**		
8.1	- Funcionalidades Elementares. **&** - Funcionalidades Avançadas como: • tratamento de colisões • duplo *buffer*	Sim	O Gerenciador_Colisoes é responsável por verificar e tratar colisões entre os elementos do jogo, como jogadores, inimigos e obstáculos. Esse gerenciador assegura que as interações entre personagens, inimigos e obstáculos no jogo ocorram de maneira apropriada, incluindo danos causados por colisões e tratamento adequado de movimentos sobre plataformas.
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).	Não	
---	**Interdisciplinaridades via utilização de Conceitos de Matemática Contínua e/ou Física.**		
8.3	- Ensino Médio Efetivamente.	*Sim*	Foi utilizado o conceito de queda livre para aplicar gravidade nas Entidades e de lançamento vertical para o pulo dos jogadores.
8.4	- Ensino Superior Efetivamente.	*Não*	

Conceitos

Conceitos 75% concluídos

9	**Engenharia de Software**		
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos. &	Sim	O grupo trabalhou intensivamente na busca pela reprodução mais fiel possível dos requisitos requisitados, através de recursos como o site da disciplina e reuniões com Monitores/Professor.
9.2	- Diagrama de Classes em *UML*.	Sim	Durante todo o desenvolvimento do Jogo foi elaborado um Diagrama de Classes para melhor organização e compreensão do código.
9.3	- Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões.	Não	O número foi inferior a 5 padrões de projeto.
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	A Tabela de Requisitos e o Diagrama de Classes fornecidos pelo professor foram os principais guias no desenvolvimento do jogo.

Conceitos

Conceitos 100% concluídos

Total 70%

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	Utilizou-se um repositório no Github para compartilhamento do código entre os participantes. link: https://github.com/danisommer/JOGO-PLATAFORMA
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Reuniões realizadas nos dias 17/11, às 10h30 e 21/11, às 14h00.
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Reuniões com o Monitor Ariel realizadas nos dias 18/11, das 11h às 13h10 e 11/11, das 11h às 13h30. Ambos os integrantes estiveram presentes nesses períodos citados. Além disso, Daniel esteve presente no primeiro Peteco, enquanto Enzo esteve nos quatro.
10.4	- Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Otávio e Letícia, da S73.

Vídeo Gameplay

Conclusão

Sobre os resultados do projeto quanto ao requisitos sinto que poderíamos ter ido melhor, mas muito se deve ao trabalho de certa forma desigual em relação ao código.
Contudo a realização desse projeto ajudou muito a melhorar o nosso entendimento sobre a orientação à objetos, sua implementação e finalidade.
Além disso, consideramos positivo o fato de termos aprendido novas bibliotecas do C++, desenvolvimento de diagramas de classes e sentido um pouco de como é a programação na indústria.