RegularGame++

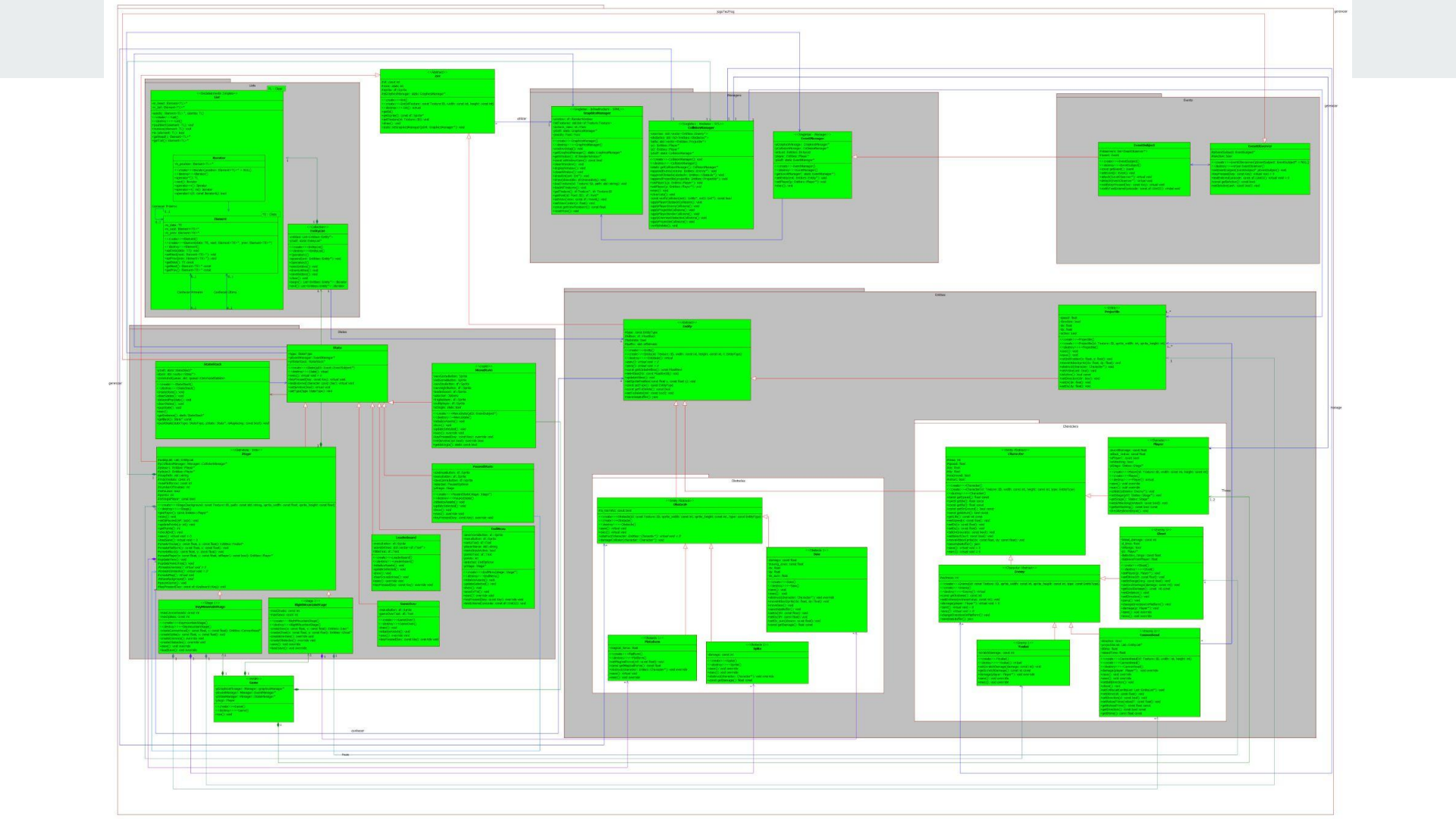
Técnicas de Programação S73

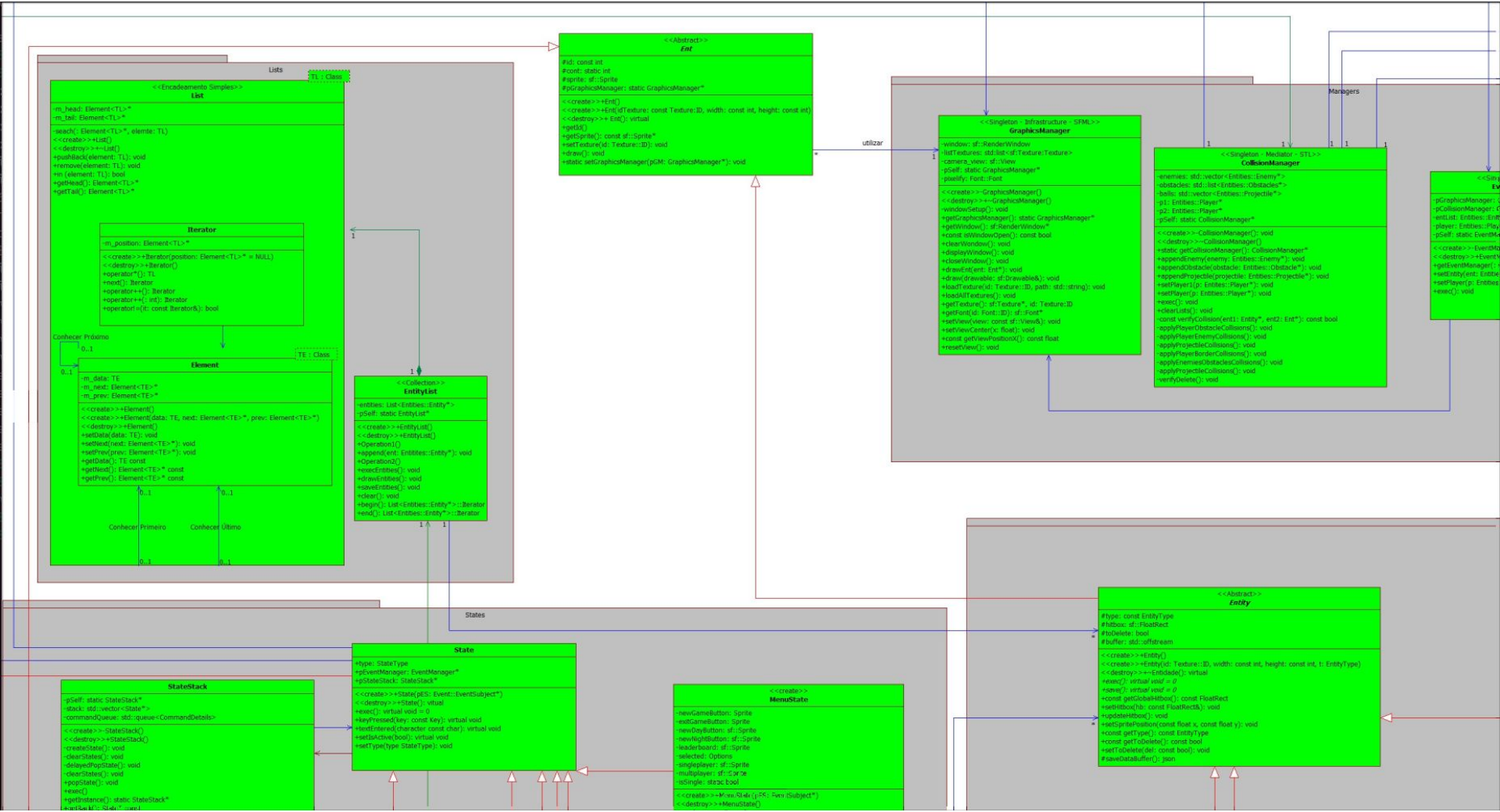
Lucas Tanaka e Yudi Gunzi

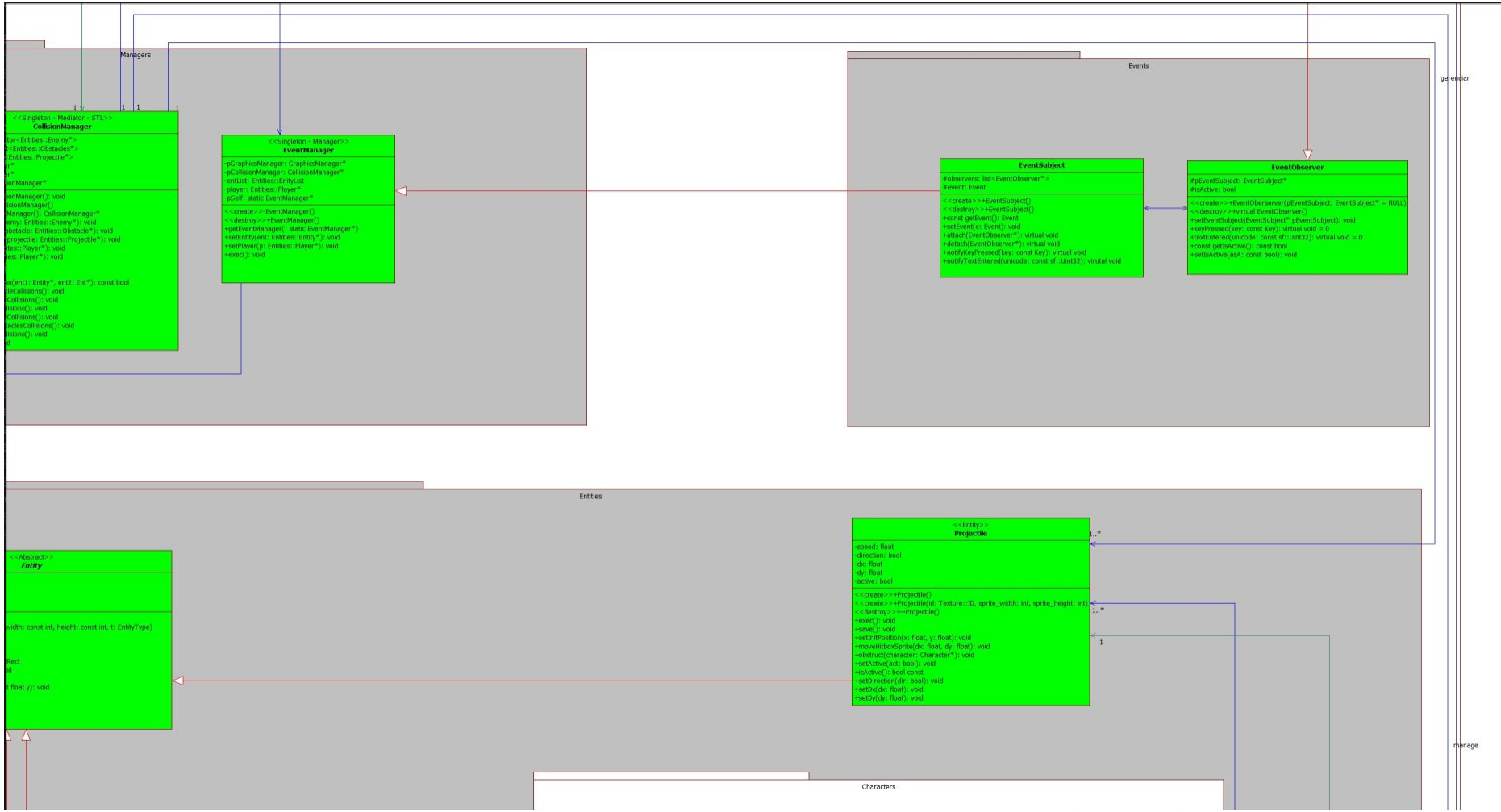
1	Apresentar graficamente menu de opções aos usuários do Jogo, no qual pode se escolher fases, escolher ver colocação (*ranking*) de jogadores e escolher demais opções pertinentes (previstas nos demais requisitos).	**REALIZADO**	Cf. classe MenuState, com suporte da SFML.
2	Permitir um ou dois jogadores com representação gráfica aos usuários do Jogo, sendo que no último caso é para que os dois joguem de maneira concomitante.	**REALIZADO**	Cf. classe Player cujos objetos estão agregados em Stage. A quantidade de jogadores pode ser escolhida vida menu. Os dois jogadores possuem diferenças visuais.
3	Disponibilizar ao menos duas fases distintas que podem ser jogadas sequencialmente ou selecionadas, via menu, nas quais jogadores tentam neutralizar inimigos por meio de algum artifício e vice-versa.	**REALIZADO**	Cf. classes DayMountainStage e NightMountainStage derivadas de Stage. Jogadores podem neutralizar inimigos via ataque.
4	Ter pelo menos três tipos distintos de inimigos, cada qual com sua representação gráfica, sendo que ao menos um deles deve poder lançar projetil contra o(s) jogador(es) e um dos inimigos dever ser um 'chefão'.	**REALIZADO**	Cf. classes Youkai, Cannonhead (lança projétil) e Ghost ('chefão').
5	Ter a cada fase ao menos dois tipos de inimigos (um deles exclusivo nela) com número aleatório de instâncias, podendo ser várias instâncias (definindo um máximo) e sendo pelo menos 3 instâncias para cada tipo que estiver na fase.	**REALIZADO**	Cf. Youkai (presente em ambas as fases), Cannonhead (exclusivo de DayMountainStage), Ghost (exclusivo de NightMountainStage).

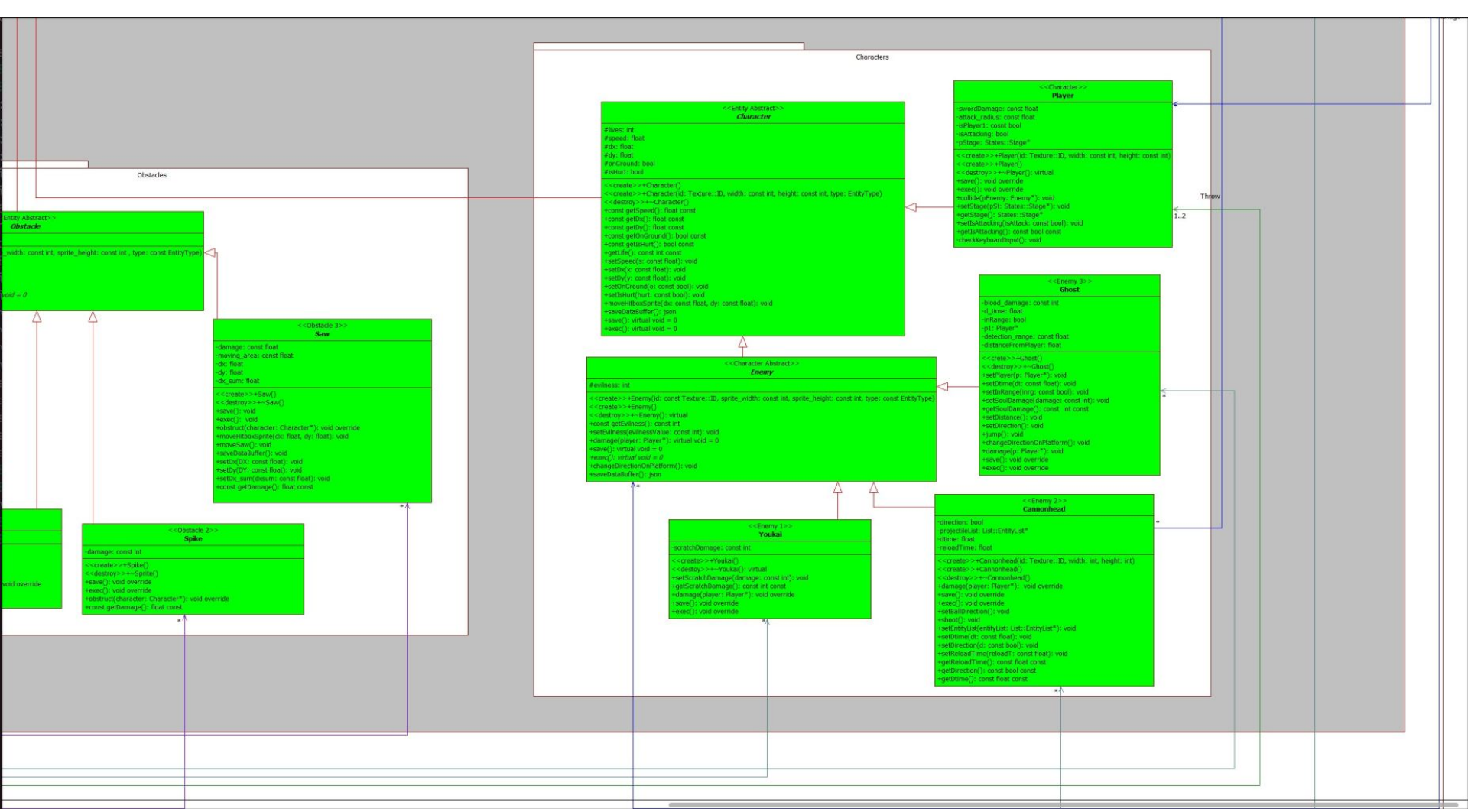
6	Ter três tipos de obstáculos, cada qual com sua representação gráfica, sendo que ao menos um causa dano em jogador se colidirem.	**REALIZADO**	Cf. classes Platform, Spike e Saw. Spike e Saw causam dano.
7	Ter em cada fase ao menos dois tipos de obstáculos (um deles exclusivo nela) com número aleatório (definindo um máximo) de instâncias (*i.e.*, objetos), sendo pelo menos 3 instâncias por tipo.	**REALIZADO**	Cf. Platform (presente em ambas as fases), Spike (exclusivo de DayMountainStage), Saw (exclusivo de NightMountainStage).
8	Ter em cada fase um cenário de jogo constituído por obstáculos, sendo que parte deles **devem ser** plataformas ou similares, sobre as quais pode haver inimigos e podem subir jogadores. Em cada fase, só poder ter um tipo coincidente de inimigo e um tipo coincidente de obstáculo (que é a	**REALIZADO**	Cf. classes Stage, Platform (obstáculo coincidente), Youkai(inimigo coincidente).

	plataforma) em relação as demais fases.		
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.	**REALIZADO**	Cf. classe CollisionManager, pacote Entities.
10	Permitir: (1) salvar nome do usuário, manter/salvar pontuação (incrementada via neutralização de inimigos) do jogador controlado pelo usuário e gerar lista de pontuação (*ranking*). **E** (2) Pausar e **Salvar/Recuperar** Jogada.	**NÃO** realizado.	Cf. classes DayMountainStage, NightMountainStage, StateStack.
Total de requisitos funcionais apropriadamente realizados.			**100%** (cem por cento).




Lists
TL : Class
<<Encadeamento Simples>>
List
-m_head: Element<TL>*
-m_tail: Element<TL>*
-seach(: Element<TL>*, elemte: TL)
<<create>>+List()
<<destroy>>+~List()
+pushBack(element: TL): void
+remove(element: TL): void
+in (element: TL): bool
+getHead(): Element<TL>*
+getTail(): Element<TL>*
Iterator
-m_position: Element<TL>*
<<create>>+Iterator(position: Element<TL>* = NULL)
<<destroy>>+Iterator()
+operator*(): TL
+next(): Iterator
+operator++(): Iterator
+operator++(: int): Iterator
+operator!=(it: const Iterator&): bool
Conhecer Próximo
0..1
0..1
TE : Class
Element
-m_data: TE
-m_next: Element<TE>*
-m_prev: Element<TE>*
<<create>>+Element()
<<create>>+Element(data: TE, next: Element<TE>*, prev: Element<TE>*)
<<destroy>>+Element()
+setData(data: TE): void
+setNext(next: Element<TE>*): void
+setPrev(prev: Element<TE>*): void
+getData(): TE const
+getNext(): Element<TE>* const
+getPrev(): Element<TE>* const
0..1
0..1
Conhecer Primeiro
Conhecer Último
0..1
0..1
1
1
<<Collection>>
EntityList
-entities: List<Entities::Entity*>
-pSelf: static EntityList*
<<create>>+EntityList()
<<destroy>>+EntityList()
+Operation1()
+append(ent: Entites::Entity*): void
+Operation2()
+execEntities(): void
+drawEntities(): void
+saveEntities(): void
+clear(): void
+begin(): List<Entities::Entity*>::Iterator
+end(): List<Entities::Entity*>::Iterator
1
1
<<Abstract>>
Ent
#id: const int
#cont: static int
#sprite: sf::Sprite
#pGraphicsManager: static GraphicsManager*
<<create>>+Ent()
<<create>>+Ent(idTexture: const Texture:ID, width: const int, height: const int)
<<destroy>>+ Ent(): virtual
+getId()
+getSprite(): const sf::Sprite*
+setTexture(id: Texture::ID): void
+draw(): void
+static setGraphicsManager(pGM: GraphicsManager*): void
utilizar
*
1
Managers
<<Singleton - Infrastructure - SFML>>
GraphicsManager
-window: sf::RenderWindow
-listTextures: std::list<sf::Texture:Texture>
-camera_view: sf::View
-pSelf: static GraphicsManager*
-pixelFy: Font::Font
<<create>>-GraphicsManager()
<<destroy>>+~GraphicsManager()
-windowSetup(): void
+getGraphicsManager(): static GraphicsManager*
+getWindow(): sf:RenderWindow*
+const isWindowOpen(): const bool
+clearWindow(): void
+displayWindow(): void
+closeWindow(): void
+drawEnt(ent: Ent*): void
+draw(drawable: sf:Drawable&): void
+loadTexture(id: Texture::ID, path: std::string): void
+loadAllTextures(): void
+getTexture(): sf:Texture*, id: Texture:ID
+getFont(id: Font::ID): sf::Font*
+setView(view: const sf::View&): void
+setViewCenter(x: float): void
+const getViewPositionX(): const float
+resetView(): void
1
1
<<Singleton - Mediator - STL>>
CollisionManager
-enemies: std::vector<Entities::Enemy*>
-obstacles: std::list<Entities::Obstacles*>
-balls: std::vector<Entities::Projectile*>
-p1: Entities::Player*
-p2: Entities::Player*
-pSelf: static CollisionManager*
<<create>>-CollisionManager(): void
<<destroy>>-~CollisionManager()
+static getCollisionManager(): CollisionManager*
+appendEnemy(enemy: Entities::Enemy*): void
+appendObstacle(obstacle: Entities::Obstacle*): void
+appendProjectile(projectile: Entities::Projectile*): void
+setPlayer1(p: Entites::Player*): void
+setPlayer(p: Entities::Player*): void
+exec(): void
+clearLists(): void
-const verifyCollision(ent1: Entity*, ent2: Ent*): const bool
-applyPlayerObstacleCollisions(): void
-applyPlayerEnemyCollisions(): void
-applyProjectileCollisions(): void
-applyPlayerBorderCollisions(): void
-applyEnemiesObstaclesCollisions(): void
-applyProjectileCollisions(): void
-verifyDelete(): void
States
StateStack
-pSelf: static StateStack*
-stack: std::vector<State*>
-commandQueue: std::queue<CommandDetails>
<<create>>-StateStack()
<<destroy>>+StateStack()
-createState(): void
-clearStates(): void
-delayedPopState(): void
-clearStates(): void
+popState(): void
+exec()
+getInstance(): static StateStack*
State
+type: StateType
+pEventManager: EventManager*
+pStateStack: StateStack*
<<create>>+State(pES: Event::EventSubject*)
<<destroy>>+State(): virtual
+exec(): virtual void = 0
+keyPressed(key: const Key): virtual void
+textEntered(character const char): virtual void
+setIsActive(bool): virtual void
+setType(type StateType): void
<<create>>
MenuState
-newGameButton: Sprite
-exitGameButton: Sprite
-newDayButton: sf::Sprite
-newNightButton: sf::Sprite
-leaderboard: sf::Sprite
-selected: Options
-singleplayer: sf::Sprite
-multiplayer: sf::Sprite
-isSingle: static bool
<<Abstract>>
Entity
#type: const EntityType
#hitbox: sf::FloatRect
#toDelete: bool
#buffer: std::offstream
<<create>>+Entity()
<<create>>+Entity(id: Texture::ID, width: const int, height: const int, t: EntityType)
<<destroy>>+~Entidade(): virtual
+exec(): virtual void = 0
+save(): virtual void = 0
+const getGlobalHitbox(): const FloatRect
+setHitbox(hb: const FloatRect&): void
+updateHitbox(): void
+setSpritePosition(const float x, const float y): void
+const getType(): const EntityType
+const getToDelete(): const bool
+setToDelete(del: const bool): void
#saveDataBuffer(): json


Managers
<<Singleton - Mediator - STL>>
CollisionManager
<<Singleton - Manager>>
EventManager
-pGraphicsManager: GraphicsManager*
-pCollisionManager: CollisionManager*
-entList: Entities::EntityList
-player: Entities::Player*
-pSelf: static EventManager*
<<create>>-EventManager()
<<destroy>>+EventManager()
+getEventManager(): static EventManager*
+setEntity(ent: Entities::Entity*): void
+setPlayer(p: Entities::Player*): void
+exec(): void
Events
EventSubject
#observers: list<EventObserver*>
#event: Event
<<create>>+EventSubject()
<<destroy>>+EventSubject()
+const getEvent(): Event
+setEvent(e: Event): void
+attach(EventObserver*): virtual void
+detach(EventObserver*): virtual void
+notifyKeyPressed(key: const key): virtual void
+notifyTextEntered(unicode: const sf::Uint32): virutal void
EventObserver
#pEventSubject: EventSubject*
#isActive: bool
<<create>>+EventObserver(pEventSubject: EventSubject* = NULL)
<<destroy>>+virtual EventObserver()
+setEventSubject(EventSubject* pEventSubject): void
+keyPressed(key: const Key): virtual void = 0
+textEntered(unicode: const sf::Uint32): virtual void = 0
+const getIsActive(): const bool
+setIsActive(asA: const bool): void
gerenciar
Entities
<<Abstract>>
Entity
<<Entity>>
Projectile
-speed: float
-direction: bool
-dx: float
-dy: float
-active: bool
<<create>>+Projectile()
<<create>>+Projectile(id: Texture::ID, sprite_width: int, sprite_height: int)
<<destroy>>+~Projectile()
+exec(): void
+save(): void
+setInitPosition(x: float, y: float): void
+moveWithinSprite(dx: float, dy: float): void
+obstruct(character: Character*): void
+setActive(act: bool): void
+isActive(): bool const
+setDirection(dir: bool): void
+setDx(dx: float): void
+setDy(dy: float): void
1..*
1..*
1
Characters
manage


Characters
<<Entity Abstract>>
Character
#lives: int
#speed: float
#dx: float
#dy: float
#onGround: bool
#isHurt: bool
<<create>>+Character()
<<create>>+Character(id: Texture::ID, width: const int, height: const int, type: EntityType)
<<destroy>>+~Character()
+const getSpeed(): float const
+const getDx(): float const
+const getDy(): float const
+const getOnGround(): bool const
+const getIsHurt(): bool const
+getLife(): const int const
+setSpeed(s: const float): void
+setDx(x: const float): void
+setDy(y: const float): void
+setOnGround(o: const bool): void
+setIsHurt(hurt: const bool): void
+moveHitboxSprite(dx: const float, dy: const float): void
+saveDataBuffer(): json
+save(): virtual void = 0
+exec(): virtual void = 0
<<Character>>
Player
-swordDamage: const float
-attack_radius: const float
-isPlayer1: const bool
-isAttacking: bool
-pStage: States::Stage*
<<create>>+Player(id: Texture::ID, width: const int, height: const int)
<<create>>+Player()
<<destroy>>+~Player(): virtual
+save(): void override
+exec(): void override
+collide(pEnemy: Enemy*): void
+setStage(pSt: States::Stage*): void
+getStage(): States::Stage*
+setIsAttacking(isAttack: const bool): void
+getIsAttacking(): const bool const
-checkKeyboardInput(): void
Throw
1..2
Obstacles
Entity Abstract>>
Obstacle
_width: const int, sprite_height: const int , type: const EntityType)
void = 0
<<Obstacle 3>>
Saw
-damage: const float
-moving_area: const float
-dx: float
-dy: float
-dx_sum: float
<<create>>+Saw()
<<destroy>>+~Saw()
+save(): void
+exec(): void
+obstruct(character: Character*): void override
+moveHitboxSprite(dx: float, dy: float): void
+moveSaw(): void
+saveDataBuffer(): void
+setDx(DX: const float): void
+setDy(DY: const float): void
+setDx_sum(dxsum: const float): void
+const getDamage(): float const
<<Obstacle 2>>
Spike
-damage: const int
<<create>>+Spike()
<<destroy>>+~Sprite()
+save(): void override
+exec(): void override
+obstruct(character: Character*): void override
+const getDamage(): float const
void override
<<Character Abstract>>
Enemy
#evilness: int
<<create>>+Enemy(id: const Texture::ID, sprite_width: const int, sprite_height: const int, type: const EntityType)
<<create>>+Enemy()
<<destroy>>+~Enemy(): virtual
+const getEvilness(): const int
+setEvilness(evilnessValue: const int): void
+damage(player: Player*): virtual void = 0
+save(): virtual void = 0
+exec(): virtual void = 0
+changeDirectionOnPlatform(): void
+saveDataBuffer(): json
<<Enemy 3>>
Ghost
-blood_damage: const int
-d_time: float
-inRange: bool
-p1: Player*
-detection_range: const float
-distanceFromPlayer: float
<<crete>>+Ghost()
<<destroy>>+~Ghost()
+setPlayer(p: Player*): void
+setDtime(dt: const float): void
+setInRange(inrg: const bool): void
+setSoulDamage(damage: const int): void
+getSoulDamage(): const int const
+setDistance(): void
+setDirection(): void
+jump(): void
+changeDirectionOnPlatform(): void
+damage(p: Player*): void
+save(): void override
+exec(): void override
<<Enemy 1>>
Youkai
-scratchDamage: const int
<<create>>+Youkai()
<<destoy>>+~Youkai(): virtual
+setScratchDamage(damage: const int): void
+getScratchDamage(): const int const
+damage(player: Player*): void override
+save(): void override
+exec(): void override
<<Enemy 2>>
Cannonhead
-direction: bool
-projectileList: List::EntityList*
-dtime: float
-reloadTime: float
<<create>>+Cannonhead(id: Texture::ID, width: int, height: int)
<<create>>+Cannonhead()
<<destroy>>+~Cannonhead()
+damage(player: Player*): void override
+save(): void override
+exec(): void override
+setBallDirection(): void
+shoot(): void
+setEntityList(entityList: List::EntityList*): void
+setDtime(dt: const float): void
+setDirection(d: const bool): void
+setReloadTime(reloadT: const float): void
+getReloadTime(): const float const
+getDirection(): const bool const
+getDtime(): const float const


StateStack
-pSelf: static StateStack*
-stack: std::vector<State*>
-commandQueue: std::queue<CommandDetails>
<<create>>-StateStack()
<<destroy>>+StateStack()
-createState(): void
-clearStates(): void
-delayedPopState(): void
-clearStates(): void
+popState(): void
+exec()
+getInstance(): static StateStack*
+getBack(): State* const
+pushState(stateType: StateType, pState: State*, isReplacing: const bool): void
+pStateStack: StateStack*
<<create>>+State(pES: Event::EventSubject*)
<<destroy>>+State(): virtual
+exec(): virtual void = 0
+keyPressed(key: const Key): virtual void
+textEntered(character const char): virtual void
+setIsActive(bool): virtual void
+setType(type StateType): void
<<create>>
MenuState
-newGameButton: Sprite
-exitGameButton: Sprite
-newDayButton: sf::Sprite
-newNightButton: sf::Sprite
-leaderboard: sf::Sprite
-selected: Options
-singleplayer: sf::Sprite
-multiplayer: sf::Sprite
-isSingle: static bool
<<create>>+MenuState(pES: EventSubject*)
<<destroy>>+MenuState()
-initializeAssets(): void
-draw(): void
-updateSelected(): void
+exec(): override void
+keyPressed(key: const Key): override void
+setIsActive(isA bool): override bool
+getIsSingle(): static const bool
<<destroy>>+~Entidade(): virtual
+exec(): virtual void = 0
+save(): virtual void = 0
+const getGlobalHitbox(): const FloatRect
+setHitbox(hb: const FloatRect&): void
+updateHitbox(): void
+setSpritePosition(const float x, const float y): void
+const getType(): const EntityType
+const getToDelete(): const bool
+setToDelete(del: const bool): void
#saveDataBuffer(): json
<<Abstratata - Ente>>
Stage
#entityList: List::EntityList
#pCollisionManager: Manager::CollisionManager*
#player1: Entities::Player*
#player2: Entities::Player*
#mapPath: std::string
#maxYoukais: const int
-maxPlatforms: const int
#numberOfYoukais: int
#isPaused: bool
#points: int
#isSinglePlayer: const bool
<<create>>+Stage(background: const Texture::ID, path: const std::string, sprite_width: const float, sprite_height: const float)
<<destroy>>+~Stage()
-getPlayer(): const Entities::Player*
+exec(): void
+setIsPaused(isP: bool): void
+updatePoints(p: int): void
+getPoints(): int
+checkEnd(): void
+save(): virtual void = 0
+loadSave(): virtual void = 0
#createYoukai(y: const float, x: const float): Entities::Youkai*
#createPlatform(y: const float, x: const float): void
#createBlock(x: const float, y: const float): void
#createPlayer(x: const float, y: const float, isPlayer1: const bool): Entities::Player*
#updateView(): void
#updatePointsText(): void
#createEnemies(): virtual void = 0
#createObstacles(): virtual void = 0
#createMap(): virtual void
#drawBackground(): void
#pauseGame(): void
#keyPressed(key: const sf::Keyboard::Key): void
PausedState
-continueButton: sf::Sprite
-menuButton: sf::Sprite
-saveGameButton: sf::Sprite
-selected: PausedOptions
-pStage: Stage
<<create>>+PausedState(stage: Stage*)
<<destroy>>+PausedState()
-initializeAssets(): void
-updateSelected(): void
-draw(): void
+exec(): override void
+keyPressed(key: const Key): override void
Leaderboard
-menuButton: sf::Sprite
-scoreEntries: std::vector<sf::Text*>
-titleText: sf::Text
<<create>>+Leaderboard()
<<destroy>>+Leaderboard()
-initializeAssets(): void
-updateSelected(): void
-draw(): void
-clearScoreEntries(): void
+exec(): override void
+keyPressed(key: const Key): override void
EndMenu
-saveScoreButton: sf::Sprite
-menuButton: sf::Sprite
-nameText: sf::Text
-playerName: std::string
-nomeInputActive: bool
-pointsText: sf::Text
-points: int
-selected: EndOptions
-pStage: Stage*
<<create>>+EndMenu(stage: Stage*)
<<destroy>>+EndMenu()
-initializeAssets(): void
-updateSelected(): void
-draw(): void
-saveOnTxt(): void
+exec(): override void
+keyPressed(key: const Key): override void
+textEntered(unicode: const sf::Uint32): void
<<Stage 1>>
DayMountainStage
-maxCannonheads: const int
-maxSpikes: const int
<<create>>+DaymountainStage()
<<destroy>>+~DaymountainStage()
-createCannonhead(y: const float, x: const float): Entities::Cannonhead*
-createSpike(y: const float, x: const float): void
-createEnemies(): override void
-createObstacles(): override void
+save(): void override
+loadSave(): void override
<<Stage 2>>
NightMountainStage
-maxGhosts: const int
-maxSaws: const int
<<create>>+NightMountainStage()
<<destroy>>+NightMountainStage()
-createSaw(x: const float, y: const float): Entities::Saw*
-createGhost(x: const float, y: const float): Entities::Ghost*
-createEnemies(): override void
-createObstacles(): override void
+save(): void override
+loadSave(): void override
GameOver
-menuButton: sf::Sprite
-gameOverText: sf::Text
<<create>>+GameOver()
<<destroy>>+GameOver()
-draw(): void
-initializeAssets(): void
+exec(): override void
+keyPressed(key: const Key): override void
<<MAIN>>
Game
-pGraphicsManager: Manager::graphicsManager*
-pEventManager: Manager::EventManager*
-pStateManager: Manager::StateManager*
-pJog1: Player
<<create>>+Game()
<<destroy>>+~Game()
+run(): void
Obstacles
<<Entity Abstract>>
Obstacle
#is_harmful: const bool
<<create>>+Obstacle(id: const Texture::ID, sprite_width: const int, sprite_height: const int , type: const EntityType)
<<create>>+Obstacle()
<<destroy>>+~Obstacle()
+save(): virtual void
+exec(): virtual void
+obstruct(character: Entities::Character*): virtual void = 0
+damageCollision(character: Character*): void
-damage: const float
-moving_area: const float
-dx: float
-dy: float
-dx_sum: float
<<create>>+Saw()
<<destroy>>+~Saw()
+save(): void
+exec(): void
+obstruct(character: Character
+moveHitboxSprite(dx: float, dy
+moveSaw(): void
+saveDataBuffer(): void
+setDx(DX: const float): void
+setDy(DY: const float): void
+setDx_sum(dxsum: const float
+const getDamage(): float cons
<<Obstacle 1>>
Plataform
-magical_force: float
<<create>>+Platform()
<<destroy>>+~Platform()
+setMagicalForce(mf: const float): void
+const getMagicalForce(): const float
+obstruct(character: Entities::Character*): void override
+exec(): virtual void
+exec(): void override
<<Obstacle 2>>
Spike
-damage: const int
<<create>>+Spike()
<<destroy>>+~Sprite()
+save(): void override
+exec(): void override
+obstruct(character: Character*): void override
+const getDamage(): float const

1	**Elementares:**		
1.1 &	- Classes, objetos. & -Atributos(privado), variáveis e constantes. - Métodos (com e sem retorno).	Sim	- Todos .h e .cpp, como nas classes nos *namespaces* States e Entities. - Classes, Objetos, Atributos e Métodos foram utilizados porque são conceitos elementares na orientação a objetos. Exemplo de '&' em getSprite() na classe Ent, usado para retornar endereço de sprite.
1.2 &	- Métodos (com retorno *const* e parâmetro *const*). - Construtores (sem/com parâmetros) e destrutores	Sim	- Na maioria dos .h e .cpp, como nas classes nos *namespaces* Entities e States. - A constância pertinente evita mudanças equivocadas, construtores são mandatórios para inicializar atributos e destrutores pertinentes para finalizações como desalocacões.
1.3	- Classe Principal.	Sim	- main.cpp
1.4	- Divisão em .h e .cpp.	Sim	- No desenvolvimento como um todo, como nas classes nos *namespaces* Entities e States. - Permite organizar as classes e afins que compõem o sistema.
2	**Relações de:**		
2.1	- Associação direcional. & - Associação bidirecional.	Sim	- Há associações direcionais evidentes, como a classe game, possuindo ponteiros para `Graphics Manager`, Event Manager. Associações bidirecionais são observadas entre Player e Stage.
2.2 &	- Agregação via associação. - Agregação propriamente dita.	Sim	- Stage agrega Obstáculos e Inimigos. As duas classes agregadas existem apenas dentro de uma fase.
2.3 &	- Herança elementar. - Herança em vários níveis.	Sim	- Conceito fundamental em POO. Observado nas heranças de Ent, Entity, Character, Player, por exemplo.
2.4	- Herança múltipla.	Sim	- A classe Stage herda de Ent e de State.

3	**Ponteiros, generalizações e exceções**		
3.1	- Operador *this* para fins de relacionamento bidirecional	Sim	- A classe State usa *this* para se adicionar ao EventManager. - Utilizado para adicionar Observer à lista de observers do Subject.
3.2	- Alocação de memória (*new* & *delete*).	Sim	- *new* usado nos métodos create dos stages, para criar objetos que são adicionados a EntityList. - Na destrutora de EntityList, esses objetos são deslocados com *delete*.
3.3	- Gabaritos/*Templates* criada/adaptados pelos autores para Listas.	Sim	- Criação de List.h, que por conseguinte é usada para criar EntityList.
3.4	- Uso de Tratamento de Exceções (*try catch*).	Sim	- Usado em Stage na função createYoukai(), para tratar possíveis erros na alocação de Youkai.
4	**Sobrecarga de:**		
4.1	- Construtoras e Métodos.	Sim	- Character, Enemy, Entity, Ent usam mais de uma construtora. - changeDirectionOnPlatform() em Ghost é uma sobrecarga da mesma função em Enemy.
4.2	- Operadores (2 tipos de operadores pelo menos).	Sim	Em Iterator do namespace List, utilizado para melhor navegação dos elementos da lista.
---	**Persistência de Objetos (via arquivo de texto ou binário)**		
4.3	- Persistência de Objetos.	Sim	-saveOnTxt de EndMenu salva informações em leaderboard.txt -saveDataBuffer() e save() salvam dados em save.json
4.4	- Persistência de Relacionamento de Objetos.	Não	

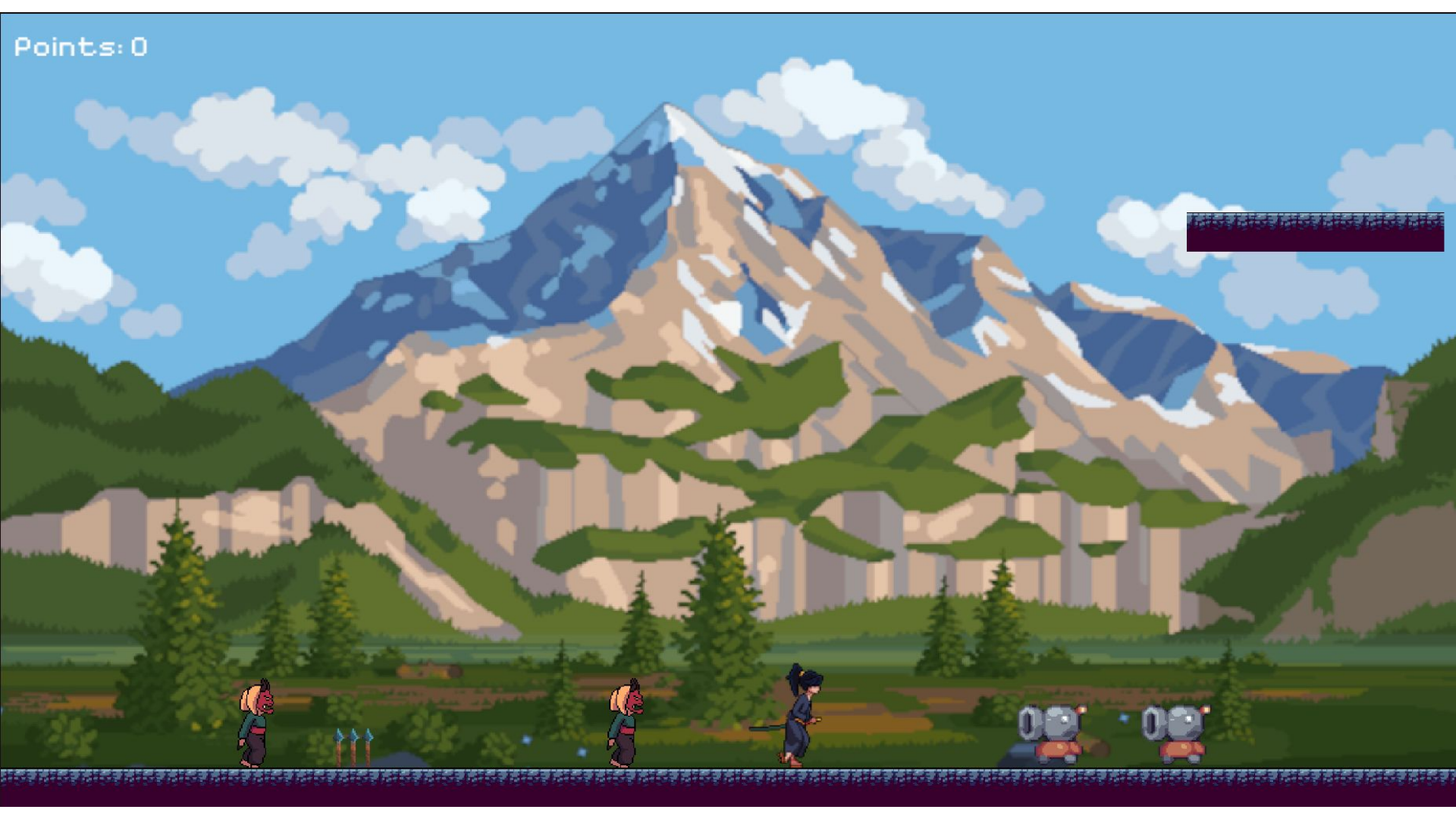
5	**Virtualidade:**		
5.1	- Métodos Virtuais Usuais.	Sim	-Há diversos métodos virtuais, por exemplo save() e exec() em Entity, que se tornam usuais em classes derivadas como Player, Youkai e Ghost, que são utilizados para aplicação de polimorfismo.
5.2	- Polimorfismo.	Sim	-execEntities() em EntityList chama o método virtual exec() de Entity dos objetos da lista, que por sua vez chama a implementação de exec() das classes derivadas.
5.3	- Métodos Virtuais Puros / Classes Abstratas.	Sim	-exec() e save() em Entity são utilizados para aplicação de polimorfismo. O mesmo acontece em obstruct() de Obstacle.
5.4	- Coesão/Desacoplamento efetiva e intensa com o apoio de padrões de projeto (mais de 5 padrões).	Sim	-Singleton usado em todos as classes no namespace Manager, pois são usados apenas um Manager de cada tipo em todo o jogo. -Iterator utilizado em List para melhor navegação da lista.
6	**Organizadores e Estáticos**		
6.1	- Espaço de Nomes (*Namespace*) criada pelos autores.	Sim	-Namespace State, Entities, Manager, Event, por exemplo, usados para melhor organização do código.
6.2	- Classes aninhadas (*Nested*) criada pelos autores.	Sim	-As classes Element e Iterator de List
6.3	- Atributos estáticos e métodos estáticos.	Sim	-Na aplicação do Singleton para os managers -Em *cont* em Ent para incremento de *id*.
6.4	- Uso extensivo de constante (*const*) parâmetro, retorno, método...	Sim	-Em métodos set e get o const é constantemente usado em parâmetros e retorno. -Em parâmetros de construtoras const é constantemente usado.

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	-Std string usada em EndMenu para receber e mudar o nome inserido. -São usadas *List* e *Vectors* para controle de entidades em CollisionManager.
7.2	- Pilha, Fila, Bifila, Fila de Prioridade, Conjunto, Multi-Conjunto, Mapa **OU** Multi-Mapa.	Sim	-O controle dos States e feitos por StateStack, que possui um stack de States*.
---	**Programação concorrente**		
7.3	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos, utilizando Posix, C-Run Time **OU** Win32API ou afins.	*Nao*	
7.4	- *Threads* (Linhas de Execução) no âmbito da Orientação a Objetos com uso de Mutex, Semáforos, **OU** Troca de mensagens.	Nao	
8	**Biblioteca Gráfica / Visual**		
8.1 **&**	- Funcionalidades Elementares. - Funcionalidades Avançadas como: tratamento de colisões, duplo *buffer* ou outros.	Sim	-Foram usados elementos da biblioteca gráfica como sprites, window, textures e respectivos métodos presentes nessas classes, pois esses elementos são essenciais para o jogo. - Demasiada utilização de FloatRects e elementos parecidos presentes na biblioteca para navegação de coordenadas. -Tratamento de colisões feito com elementos de coordenadas da biblioteca e seus respectivos métodos como *Intersect()*.

8.2 **OU**	- Programação orientada e evento efetiva (com gerenciador apropriado de eventos inclusive, via padrão de projeto *Observer*) em algum ambiente gráfico. - *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	-Movimento retilíneo uniforme na movimentação de personagens e gravidade. Gravidade é implementada com aceleração.
8.4	- Ensino Superior Efetivamente.	Sim	- Velocidade terminal implementada em Player.
9	**Engenharia de Software**		

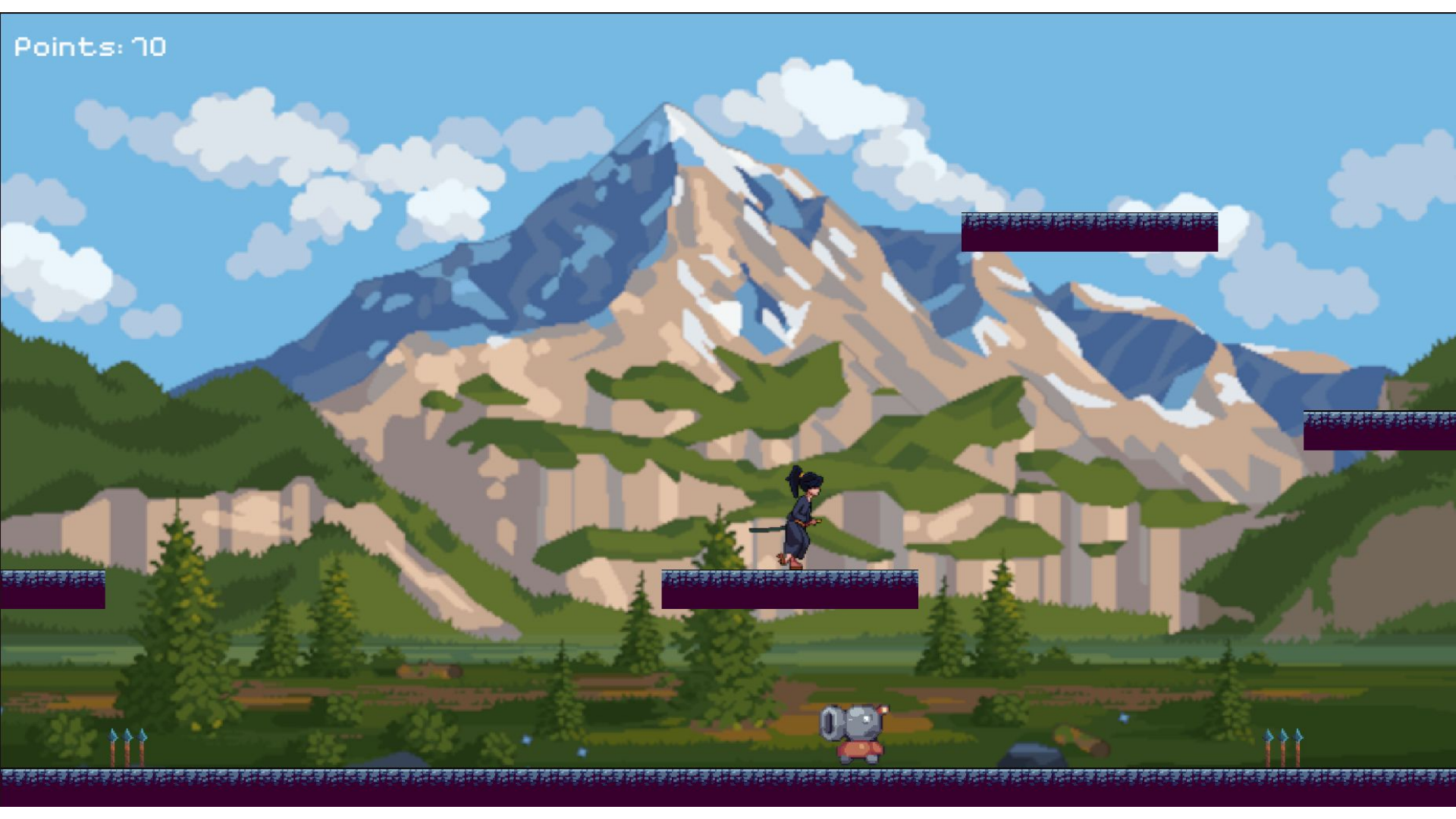
9.1	- Compreensão, melhoria e rastreabilidade de cumprimento de requisitos.	Sim	- Foi seguido o diagrama base disponibilizado pelo professor vigente. - Foram seguidos os requisitos presentes neste documento na aba Requisitos.
9.2	- Diagrama de Classes em *UML*.	Sim	- Diagrama realizado utilizando o software StarUML e com os conceitos vistos em sala de aula.
9.3	- Uso efetivo e intensivo de padrões de projeto *GOF*, *i.e.*, mais de 5 padrões.	Não	Foram implementados 2
9.4	- Testes à luz da Tabela de Requisitos e do Diagrama de Classes.	Sim	- Jogo foi realizado seguindo a tabela de requisitos e o diagrama de classes base como referência.

10	**Execução de Projeto**		
10.1 &	- Controle de versão de modelos e códigos automatizado (via github). - Uso de alguma forma de cópia de segurança (*i.e.*, *backup*).	Sim	Foi utilizado git e github https://github.com/Chico7854/RegularGame
10.2	- Reuniões com o professor para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Foram feitas duas reuniões com o professor nos dias 11/06/2025 e 18/06/2025.
10.3	- Reuniões com monitor da disciplina para acompanhamento do andamento do projeto. **[ITEM OBRIGATÓRIO PARA A ENTREGA DO TRABALHO]**	Sim	Foram cumpridas 2 horas de reuniões com o PETECO, por ambos os autores, por meio das oficinas: -Gerenciador de Colisões e Gerenciador Gráfico; - Fase, TileMap e Padrão de Projeto;
10.4 &	- Escrita do trabalho e feitura da apresentação - Revisão do trabalho escrito de outra equipe e vice-versa.	Sim	Revisão do trabalho escrito realizada por Victor Hugo e pela dupla formada por Nicolas Krause e Gustavo Moretto.
Total de conceitos apropriadamente utilizados.			**87%** (oitenta e sete por cento).


Points: 0


Points: 0


Points: 28


Points: 70


Points: 0

Conclusão

Ao longo da realização deste trabalho foi possível ver uma notável evolução da equipe na construção de códigos em C++ orientada a objetos. Conceitos importantes como herança, templates, agregação e polimorfismo foram aplicados e compreendidos de forma eficaz.

Além dos aprendizados no C++, o uso de ferramentas como a biblioteca gráfica SFML, e o GitHub que é amplamente utilizado no mercado de trabalho são conhecimentos valiosos para possuir. Este projeto proporcionou também uma boa introdução a conceitos usados na indústria como o seguimento de requisitos sólidos e elaboração de relatórios robustos.

O resultado do projeto foi positivo, com o cumprimento de todos os requisitos solicitados de forma a criar um jogo coeso e alinhado aos objetivos propostos.