Documento di Object Design
Pixel Arena

Riferimento	
Versione	0.6
Data	17/01/2024
Destinatario	Carmine Gravino
Presentato da	NC10: Antonio Ferrentino, Emanuele Rosapepe, Francesco Perilli
Approvato da	

Sommario

Team Composition

Nome	Matricola	Contatti
Antonio Ferrentino	0512113367	a.ferrentino50@studenti.unisa.it
Emanuele Rosapepe	0512113418	e.rosapepe2@studenti.unisa.it
Francesco Perilli	0512113802	f.perilli2@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autori
21/12/2023	0.1	Prima stesura	Antonio Ferrentino Emanuele Rosapepe Francesco Perilli
27/12/2023	0.2	Individuazione Packages	Antonio Ferrentino Emanuele Rosapepe Francesco Perilli
28/12/2023	0.3	Creazione schemi Packages	Antonio Ferrentino
04/01/2024	0.4	Aggiunta Interfacce	Antonio Ferrentino Emanuele Rosapepe Francesco Perilli
09/01/2024	0.5	Aggiunta Design Pattern	Antonio Ferrentino Francesco Perilli
17/01/2024	0.6	Correzioni e revisione	Antonio Ferrentino Emanuele Rosapepe Francesco Perilli

1. Introduzione

Lo scopo del gioco "Pixel Arena" è quello di intrattenere tutti i videogiocatori per svariate ore di gioco, attraverso il suo gameplay fluido e dinamico.

In questa sezione del presente documento verranno illustrate le linee guida per la scrittura del codice, elencate le definizioni, gli acronimi e le abbreviazioni che verranno usate in tutto il documento ed infine i riferimenti e link utili.

1.1 Linee Guida per la Scrittura del Codice

In questa sezione del documento verranno illustrate di seguito le regole da rispettare durante l'implementazione del sistema. È di seguito riportata una lista delle convenzioni utilizzate nella stesura del codice, con link ufficiali inclusi.

Documentazione ufficiale:

- **Java:** https://docs.oracle.com/en/java/
- **LibGDX:** https://libgdx.com/wiki/
- **Android Studio:** https://developer.android.com/studio/intro?hl=it

1.2 Definizioni, acronimi e abbreviazioni

Elenco delle definizioni, acronimi ed abbreviazioni utilizzate all'interno del documento:

Definizioni	
Package	Raggruppamento di classi, interfacce o file correlati
Design Pattern	Template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e flessibilità
Application Layer	Nel pattern Three-layer rappresenta la parte che si occupa della logica di business dell'applicazione
Presentation Layer	Nel pattern Three-layer rappresenta la parte che si occupa della logica di visualizzazione dell'applicazione
Data Layer	Nel pattern Three-layer rappresenta la parte che si occupa dell'interazione con il database

1.3 Riferimenti e Link utili

Elenco dei riferimenti e link utili ad altri documenti, utili durante la lettura:

- Statement of Work (SOW)
- Requirements Analysis Document (RAD)
- System Design Document (SDD)
- Test Plan (TP)
- Test Case Specification (TCS)
- Matrice di tracciabilità

2. Packages

Questa sezione del documento espone la divisione del progetto in package sulla base dell'architettura Three Layer definita nel System Design. La directory del progetto è stata scelta seguendo quella standard del framework LibGDX:

- Core: contiene i file sorgente

I. **Presentation**, contiene i pachetti relativi al Presentation Layer:
- *screens : pacchetto che contiene le classi relative alle schermate dell'app;*
- *HUD e UI : pacchetti che contengono le classi per l'interfaccia utente e la gestione dell'interfaccia di gioco;*

II. **Application**, contiene i pacchetti relativi all'Application Layer:
- *entities : pacchetto che contiene tutte le classi relative alle entità presenti nel gioco;*
- *IA : pacchetto che contiene le classi che permettono di far funzionare l'IA dei nemici;*
- *Game Logic : pacchetto che contiene tutte le classi che si occupano delle funzionalità del gioco;*

- assets, contiene i vari assets dell'app, come immagini , icone o file audio. Questo funge da **Data Layer** del sistema;

- .gradle: package generato dal sistema di build del progetto;

- .idea: package generati dall'IDE utilizzato;

- desktop: contiene i file per poter avviare l'applicazione sul desktop;

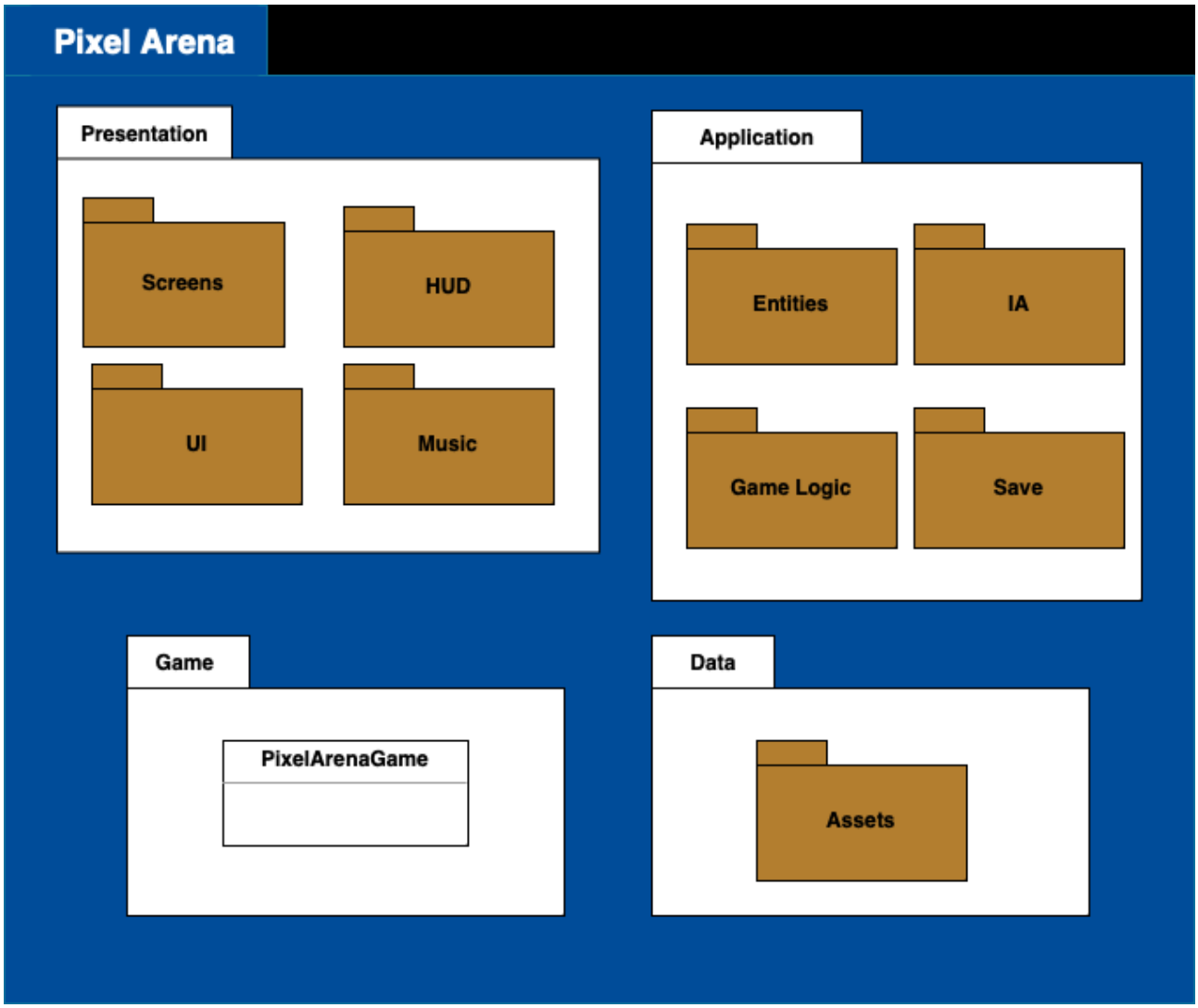
- test, contiene le classi relative al testing;

Package Pixel Arena

Qui mostreremo la struttura dei package del sistema, questa divisione è stata scelta per 2 motivi principalmente:

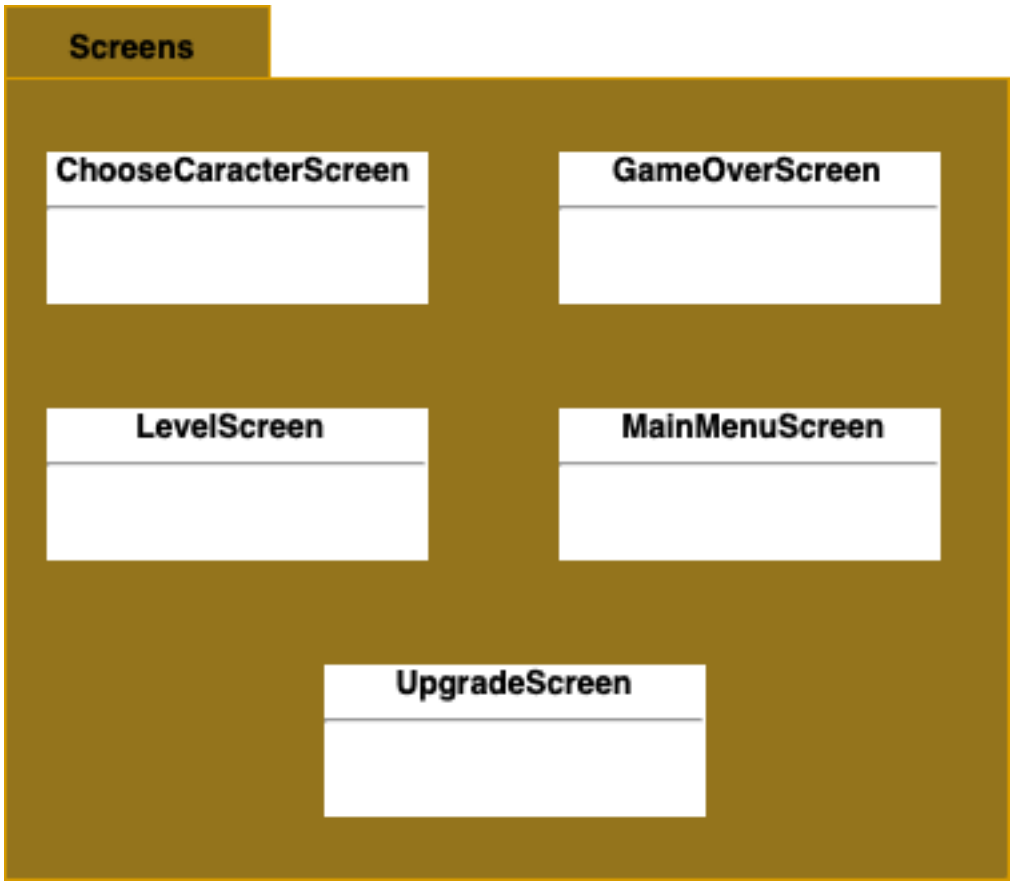
1. dividere le classi seguendo l'architettura scelta nel System Design in modo da avere le classi con responsabilità simili nella stessa posizione;

2. dividere le risorse multimediali dal codice effettivo;

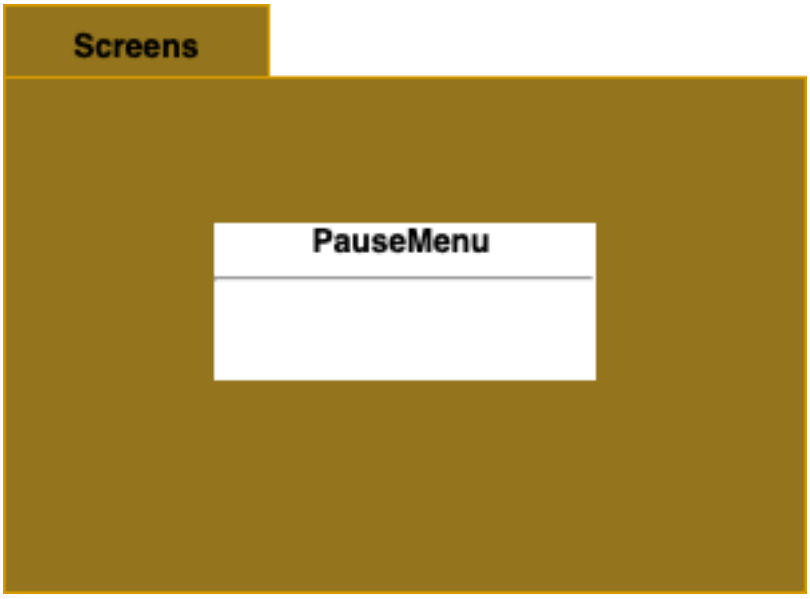


Package HUD

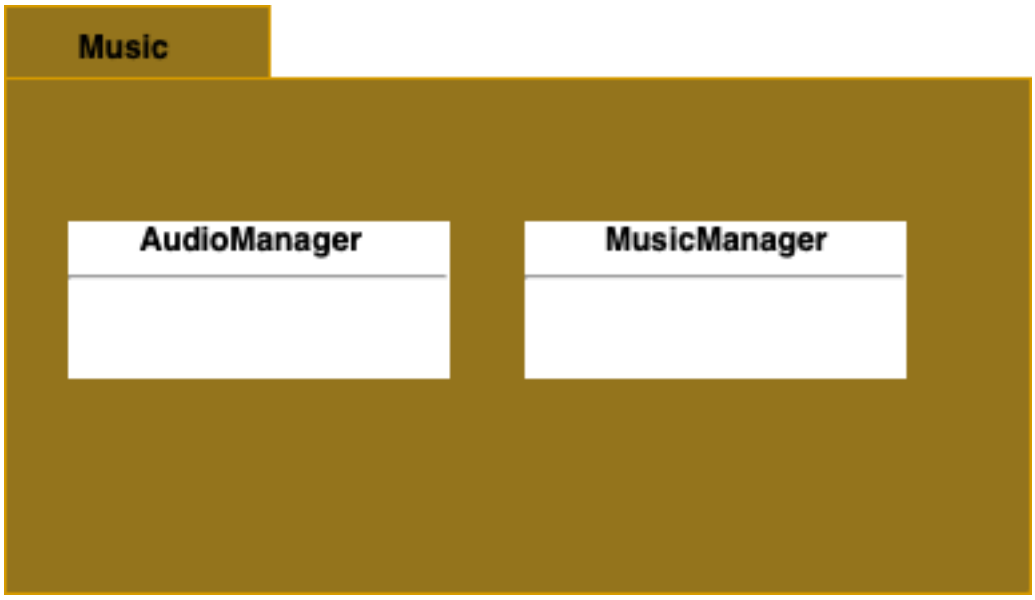
Package Screens



Package UI

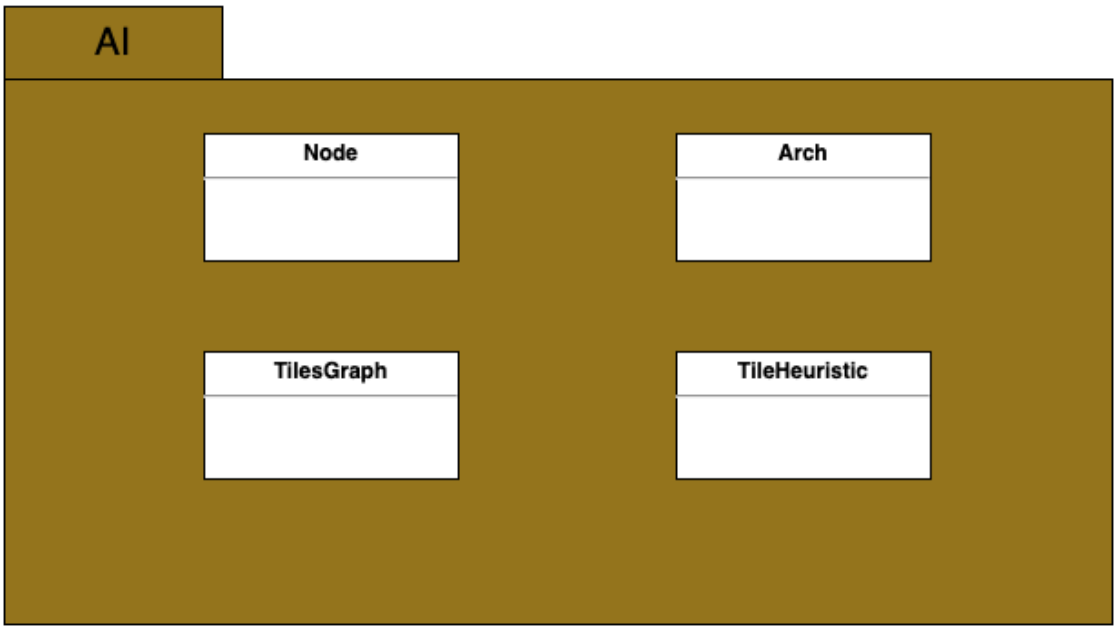


Package Music

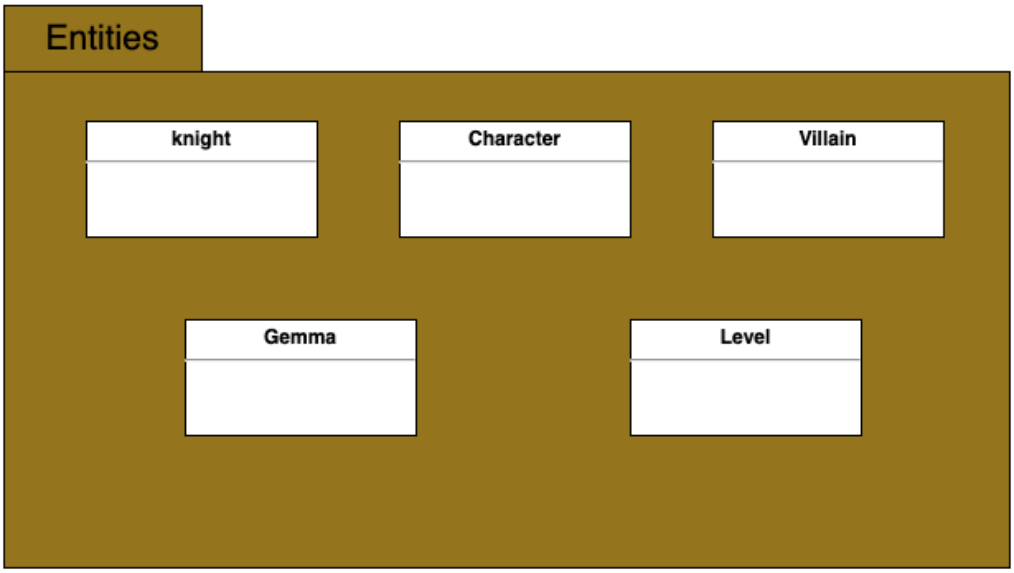


Package Assets

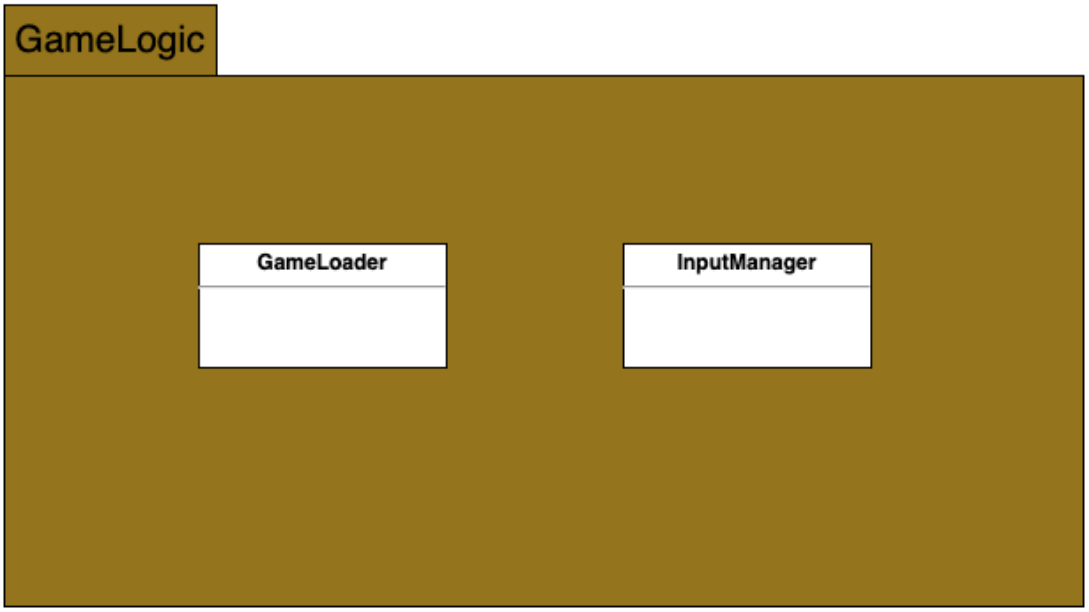
Package AI



Package Entities



Package GameLogic



Package Save

3. Class Interfaces

In questa sezione mostreremo nelle interfacce del sistema i metodi più significativi per ognuna di esse. Di fatto, abbiamo tralasciato solo i getters/setters e qualche metodo che incide pochissimo sulle funzionalità dell'applicazione.

Package HUD

Nome classe	**LevelHud**
Descrizione	Questa classe permette di visualizzare l'hud sullo schermo
Metodi	+LevelHud() +setHealtPoints(int healtPoints) +setGemsCounter(int gemsCounter) +setNumLevel(int enemiesNum) +render()
Invariante di classe	/

Nome metodo	**+ LevelHud()**
Descrizione	Inizializza tutti i vari assets presenti e li posiziona sullo schermo
Pre-condizione	/
Post-condizione	/

Nome metodo	**+setHealtPoints(int healtPoints)**
Descrizione	Aggiorna il numero di HP del personaggio
Pre-condizione	Il personaggio deve essere diverso da null
Post-condizione	/

Nome metodo	**+setGemsCounter(int gemsCounter)**
Descrizione	Aggiorna il numero di gemme del personaggio
Pre-condizione	Il personaggio deve essere diverso da null
Post-condizione	/

Nome metodo	**+setNumLevel(int enemiesNum)**
Descrizione	Aggiorna il numero dei nemici rimanenti
Pre-condizione	Il personaggio deve essere diverso da null
Post-condizione	/

Nome metodo	**+ render()**
Descrizione	Renderizza l'hud rendendolo visibile
Pre-condizione	/
Post-condizione	/

Package Screens

Nome classe	**ChooseCharacterScreen**
Descrizione	Questa classe permette di visualizzare la schermata di selezione del personaggio sullo schermo
Metodi	+show() +render()
Invariante di classe	/

Nome metodo	**+ render()**
Descrizione	Renderizza il menu di scelta del personaggio rendendolo visibile
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ show()**
Descrizione	Inizializza tutti i vari assets presenti e li posiziona sullo schermo, collegandogli i rispettivi ActionListeners
Pre-condizione	/
Post-condizione	/

Nome classe	**GameOverScreen**
Descrizione	Questa classe permette di visualizzare la schermata di Game Over sullo schermo
Metodi	+show() +render()
Invariante di classe	/

Nome metodo	**+ show()**
Descrizione	Inizializza tutti i vari assets presenti e li posiziona sullo schermo, collegandogli i rispettivi ActionListeners
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ render()**
Descrizione	Renderizza la schermata di Game Over rendendola visibile
Pre-condizione	/
Post-condizione	/

Nome classe	**LevelScreen**
Descrizione	Questa classe permette di visualizzare la schermata di selezione del personaggio sullo schermo
Metodi	+show() +render() +resize() +update() +mapCollision(+charactersCollision() +checkHealingBases() +updateAI() +checkGems()
Invariante di classe	/

Nome metodo	**+ show()**
Descrizione	Inizializza le variabili necessarie per eseguire l'applicazione
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ render()**
Descrizione	Aggiorna costantemente il frame corrente che permette l'aggiornamento visivo del gioco
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ resize(int width, int height)**
Descrizione	Viene richiamato ogni volta che cambia la finestra, e aggiorna il Viewport della camera con le nuove dimensioni della finestra
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ update()**
Descrizione	Metodo che aggiorna lo stato del gioco chiamando differenti metodi di check
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ mapCollision()**
Descrizione	Controlla se c'è stata una collisione tra il main character e i bordi della mappa, e tra i nemici e i bordi della mappa
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ charactersCollision()**
Descrizione	Controlla se c'è stata una collisione tra personaggi
Pre-condizione	/
Post-condizione	/

Nome metodo	+ checkHealingBases()
Descrizione	Controlla se il main character è sopra la base di ricarica della vita, se si chiama il metodo di cura
Pre-condizione	/
Post-condizione	/

Nome metodo	+ updateAI()
Descrizione	Questo metodo aggiorna tutte le variabili che riguardano la parte dei grafi dell'AI e i nodi associati al main character e ai nemici; Inoltre aggiorna i cammini per ogni nemico
Pre-condizione	/
Post-condizione	/

Nome metodo	+ checkGems()
Descrizione	Controlla se il giocatore ha raccolto una gemma
Pre-condizione	/
Post-condizione	/

Nome classe	MainMenuScreen
Descrizione	Questa classe permette di visualizzare la schermata del menù iniziale sullo schermo
Metodi	+show() +render() +resize(int width, int height)
Invariante di classe	/

Nome metodo	**+ show()**
Descrizione	Inizializza le variabili necessarie per eseguire l'applicazione, collegandogli i rispettivi ActionListeners
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ render()**
Descrizione	Renderizza il menu principale rendendolo visibile
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ resize(int width, int height)**
Descrizione	Viene richiamato ogni volta che cambia la finestra, e aggiorna il Viewport della camera con le nuove dimensioni della finestra
Pre-condizione	/
Post-condizione	/

Nome classe	**UpgradeScreen**
Descrizione	Questa classe permette di visualizzare la schermata dei potenziamenti de personaggio sullo schermo
Metodi	+show() +render() +update()
Invariante di classe	/

Nome metodo	**+ show()**
Descrizione	Inizializza le variabili necessarie per eseguire l'applicazione, collegandogli i rispettivi ActionListeners
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ render()**
Descrizione	Renderizza il menu di scelta del personaggio rendendolo visibile
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ update()**
Descrizione	Aggiorna i vari parametri presenti a schermo
Pre-condizione	/
Post-condizione	/

Package UI

Nome classe	**PauseMenu**
Descrizione	Questa classe permette di visualizzare la schermata dei potenziamenti de personaggio sullo schermo
Metodi	+PauseMenu(PixelArenaGame game) +render()
Invariante di classe	/

Nome metodo	**+ PauseMenu(PixelArenaGame game)**
Descrizione	Inizializza tutti i vari assets presenti e li posiziona sullo schermo, collegandogli i rispettivi ActionListeners
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ render()**
Descrizione	Renderizza il menu di pausa rendendolo visibile
Pre-condizione	/
Post-condizione	/

Package Music

Nome classe	**AudioManager**
Descrizione	Questa classe permette di riprodurre gli audio di gioco
Metodi	+AudioManager() +getInstance(): AudioManager +playHit() +playMiss() +playGemPickup()
Invariante di classe	/

Nome metodo	**+ AudioManager()**
Descrizione	Cerca i vari file audio e li inizializza
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ getInstance(): AudioManager**
Descrizione	La classe presenta un'istanza da prendere se si vogliono attivare i suoni
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ playHit()**
Descrizione	Avvia la riproduzione del suono Hit
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ playMiss()**
Descrizione	Avvia la riproduzione del suono Miss
Pre-condizione	/
Post-condizione	/

Nome metodo	+ playGemPickup()
Descrizione	Avvia la riproduzione del suono GemPickup
Pre-condizione	/
Post-condizione	/

Nome classe	AudioManager
Descrizione	Questa classe permette di riprodurre gli audio di gioco
Metodi	+MusicManager() +getInstance(): MusicManager +playIntro() +stopIntro() +playGameOver() +stopGameOver() +playBattle() +stopBattle()
Invariante di classe	/

Nome metodo	+ MusicManager()
Descrizione	Cerca i vari file audio e li inizializza
Pre-condizione	/
Post-condizione	/

Nome metodo	+ getInstance(): MusicManager
Descrizione	La classe presenta un'istanza da prendere se si vogliono attivare i suoni
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ playIntro()**
Descrizione	Avvia la riproduzione della canzone per il Menu
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ stopIntro()**
Descrizione	Ferma la riproduzione della canzone per il Menu
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ playGameOver()**
Descrizione	Avvia la riproduzione della canzone per il GameOver
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ stopGameOver()**
Descrizione	Ferma la riproduzione della canzone per il GameOver
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ playBattle()**
Descrizione	Avvia la riproduzione della canzone durante il gioco
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ stopBattle()**
Descrizione	Ferma la riproduzione della canzone durante il gioco
Pre-condizione	/
Post-condizione	/

Package AI

Nome classe	**Arch**
Descrizione	Questa classe permette di costruire un arco partendo da due nodi vicini
Metodi	+Arch(Node fromTile,Note toTile)
Invariante di classe	I nodi che rappresentano le istanze della classe non devono essere nulli

Nome metodo	**+ Arch(Node fromTile,Node toTile)**
Descrizione	Inizializza la connessione tra due nodi e calcola il costo di connessione
Pre-condizione	fromTile e toTile non devono essere nulli
Post-condizione	/

Nome classe	**Node**
Descrizione	Questa classe permette di creare un Nodo di un grafo sull'intera mappa
Metodi	+Node(int x, int y) +isAdjacentTo(Node node, int maxDistance):boolean
Invariante di classe	/

Nome metodo	**+ Node(int x, int y)**
Descrizione	Crea un Nodo di un grafo sull'intera mappa
Pre-condizione	X e y non devono essere nulli
Post-condizione	/

Nome metodo	**+ isAdjacentTo(Node node, int maxDistance):boolean**
Descrizione	Controlla se due nodi sono connessi tramite un raggio di maxDistance
Pre-condizione	maxDistance deve essere un valore maggiore di 0
Post-condizione	/

Nome classe	**TileHeuristic**
Descrizione	Classe che permette di stimare il costo per raggiungere un nodo obiettivo da un nodo di partenza
Metodi	+estimate(Node node, Node endNode):float
Invariante di classe	/

Nome metodo	**+ estimate(Node node, Node endNode):float**
Descrizione	Stima il costo per raggiungere un nodo obiettivo da un nodo di partenza
Pre-condizione	Node e endNode devono essere nodi non nulli
Post-condizione	Il valore di ritorno deve essere > 0

Nome classe	**TilesGraph**
Descrizione	Classe che permette di creare un intero grafo tracciato sulla mappa
Metodi	+getConnections(Node fromNode):Array<Connection<Node>> +addTiles(Node node) +connectAdjacentNodes(Node fromNode, Node toNode) +findPath(Node startNode, Node goalNode):GraphPath<Node>
Invariante di classe	/

Nome metodo	**+ getConnections(Node fromNode):Array<Connection<Node>>**
Descrizione	Ritorna tutti le connessioni di fromNode
Pre-condizione	Nodo fromNode non deve essere nullo
Post-condizione	/

Nome metodo	**+ addTiles(Node node)**
Descrizione	Aggiunge nodi all'ArrayList, e aggiorna l'indice che verrà usato in IndexedA*
Pre-condizione	Nodo node non deve essere nullo
Post-condizione	/

Nome metodo	**+ connectAdjacentNodes(Node fromNode, Node toNode)**
Descrizione	Collega i nodi vicini
Pre-condizione	Numeri di nodi nel grafo maggiore di 1, fromNode e toNode devono essere nodi non null e diversi tra loro
Post-condizione	/

Nome metodo	**+ findPath(Node startNode, Node goalNode):GraphPath<Node>**
Descrizione	Trova un percorso nel grafo usando IndexedA*
Pre-condizione	startNode e goalNode devono essere nodi non null e diversi tra loro
Post-condizione	Esiste un percorso fra startNode e goalNode

Package Entities

Nome classe	**Character**
Descrizione	Classe che permette di creare un personaggio presente nella mappa di gioco
Metodi	+Character(int maxHealthPoints, int attackPower, float standardSpeed, float x, float y) +setSpeed(float standardSpeed) +setPosition(float x, float y) +setPreviousPosition(float x, float y) +decreaseHealth(int amount) +doStopAndIdle() +doAttack() +doWalk() +doRun() +collisionCheck(Character v) +doHeal(float StateTime) +checkAttack(Character attacked):boolean
Invariante di classe	La vita del personaggio deve sempre essere > 0

Nome metodo	**+ Character(int maxHealthPoints, int attackPower, float standardSpeed, float x, float y)**
Descrizione	Inizializza un personaggio
Pre-condizione	I valori inseriti non devono essere nulli
Post-condizione	/

Nome metodo	**+ setSpeed(float standardSpeed)**
Descrizione	Imposta la velocità di camminata al valore passato e la velocità di corsa il 50% più veloce di questa
Pre-condizione	Il valore inserito non deve essere nullo
Post-condizione	/

Nome metodo	**+ setPosition(float x, float y)**
Descrizione	Aggiorna la posizione di un personaggio e le sue box
Pre-condizione	X e y non devono essere nulli
Post-condizione	/

Nome metodo	**+ setPreviousPosition(float x, float y)**
Descrizione	Cambia le ultime coordinate del personaggio
Pre-condizione	X e y non devono essere nulli
Post-condizione	/

Nome metodo	**+ decreaseHealth(int amount)**
Descrizione	Diminuisce la vita del personaggio di un valore passato nei parametri
Pre-condizione	Amount deve essere maggiore di 0
Post-condizione	/

Nome metodo	**+ doStopAndIdle()**
Descrizione	Imposta l'animazione del personaggio ad idle nella direzione in cui sta guardando
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ doAttack()**
Descrizione	Imposta l'animazione del personaggio all'attacco nella direzione in cui sta guardando
Pre-condizione	/
Post-condizione	/

Nome metodo	+ doWalk()
Descrizione	Imposta l'animazione del personaggio a walk nella direzione in cui sta guardando
Pre-condizione	/
Post-condizione	/

Nome metodo	+ doRun()
Descrizione	Imposta l'animazione del personaggio a corsa nella direzione in cui sta guardando
Pre-condizione	/
Post-condizione	/

Nome metodo	+ collisionCheck(Character v)
Descrizione	Controlla se c'è stata una collisione tra 2 personaggi e la risolve
Pre-condizione	V non deve essere nullo
Post-condizione	/

Nome metodo	+ doHeal(float StateTime)
Descrizione	Cura il personaggio di 10 punti vita se questo non è a vita massima
Pre-condizione	/
Post-condizione	/

Nome metodo	+ checkAttack(Character attacked):boolean
Descrizione	Controlla se il personaggio attaccato è stato colpito
Pre-condizione	Attacked deve essere diverso da nullo
Post-condizione	/

Nome classe	**Gemma**
Descrizione	Classe che permette di creare una gemma presente nella mappa di gioco
Metodi	+Gemma(float x, float y)
Invariante di classe	/

Nome metodo	**+ Gemma(float x, float y)**
Descrizione	Crea una nuova gemma con una hitbox e una texture
Pre-condizione	X e y non devono essere nulli
Post-condizione	/

Nome classe	**Knight**
Descrizione	Classe che permette di creare un main character presente nella mappa di gioco
Metodi	+Knight(float x, float y) +getLoneNode():Node
Invariante di classe	/

Nome metodo	**+ Knight(float x, float y)**
Descrizione	Inizializza un cavaliere con i suoi valori di maxHealtPoints, attack power, standard speed, la sua posizione e le animazioni di camminata, corsa ed attacco
Pre-condizione	X e y non devono essere nulli
Post-condizione	/

Nome metodo	**+ getLondeNode():Node**
Descrizione	Ritorna un nodo che non appartiene a nessun grafo della mappa
Pre-condizione	/
Post-condizione	Il nodo deve appartenere alla mappa e non deve appartenere ad un grado

---

Nome classe	**Level**
Descrizione	Classe che permette di creare un main character presente nella mappa di gioco
Metodi	+Level(int numLivello)
Invariante di classe	/

Nome metodo	**+ Level(int numLivello)**
Descrizione	Inizializza la mappa da passare al LevelScreen
Pre-condizione	Il numero passato deve avere una mappa corrispondente negli assets
Post-condizione	/

Nome classe	**Villain**
Descrizione	Classe che permette di creare un main character presente nella mappa di gioco
Metodi	+Villain(float x, float y) +checkDirection(Character character) +getNextStep(float StateTime, Character character)
Invariante di classe	/

Nome metodo	**+ Villain(float x, float y)**
Descrizione	Inizializza un nemico con i suoi valori di maxHealtPoints, standard speed, la sua posizione e le animazioni di camminata, corsa ed attacco
Pre-condizione	X e y non devono essere nulli
Post-condizione	/

Nome metodo	**+ checkDirection(Character character)**
Descrizione	Controlla in che direzione si trova il personaggio del giocatore e si gira verso di esso
Pre-condizione	Character non deve essere nullo
Post-condizione	/

Nome metodo	**+ getNextStep(float StateTime, Character character)**
Descrizione	Seleziona qual è il prossimo frame del nemico da renderizzare e lo sposta se cammina
Pre-condizione	Character non deve essere nullo
Post-condizione	/

Package GameLogic

Nome classe	**GameLoader**
Descrizione	Classe che permette di caricare un livello tra quelli presenti
Metodi	+GameLoader(Game game, Character mainCharacter, int numLevel) +load()
Invariante di classe	/

Nome metodo	**+ GameLoader(Game game, Character mainCharacter, int numLevel)**
Descrizione	Carica una lista di nemici adeguata al numero di livello passato
Pre-condizione	Character non deve essere nullo
Post-condizione	/

Nome metodo	**+ load()**
Descrizione	Carica un level screen con la lista di nemici inizializzata
Pre-condizione	/
Post-condizione	/

Nome classe	**InputManager**
Descrizione	Classe che permette di recepire tutti gli input necessari per interagire nel gioco
Metodi	+InputManager(Character character, HashMap<TilesGraph,Villain> graphVillainHashMap) +keyDown(int keycode):boolean -toggleRun() +nextFrame(float stateTime): TextureRegion +keyup(int keycode):boolean
Invariante di classe	/

Nome metodo	**+ InputManager(Character character, HashMap<TilesGraph,Villain> graphVillainHashMap**
Descrizione	Inizializza un input manager
Pre-condizione	Character e graphVillainHashMap non devono essere nulli
Post-condizione	/

Nome metodo	**+ keyDown(int keycode):boolean**
Descrizione	Cattura la pressione continua di un tasto e esegue le azioni corrispondenti
Pre-condizione	/
Post-condizione	Ritorna sempre true

Nome metodo	**- toggleRun()**
Descrizione	Un interruttore per la corsa del personaggio
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ nextFrame(float stateTime): TextureRegion**
Descrizione	Il metodo sceglie il prossimo frame del personaggio in base all'input da tastiera dell'utente
Pre-condizione	/
Post-condizione	/

Nome metodo	**+ keyup(int keycode):boolean**
Descrizione	Cattura il rilascio di un tasto ed esegue le azioni corrispondenti
Pre-condizione	/
Post-condizione	Ritorna sempre true

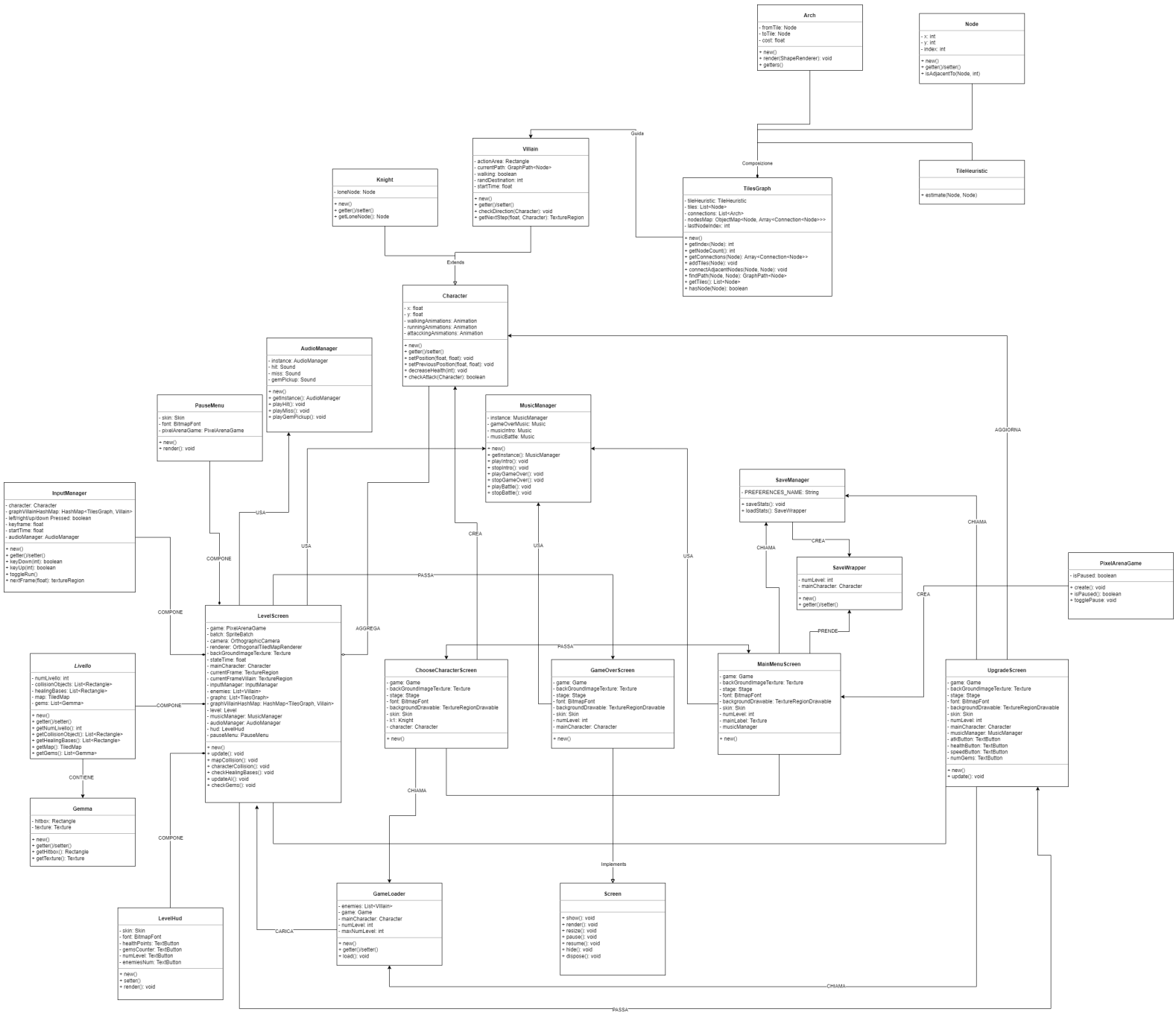
Package Save

Nome classe	**SaveManager**
Descrizione	Classe che permette di salvare la partita corrente e di caricarla
Metodi	+saveStats(Character mainCharacter, int numLivello) +loadStats():SaveWrapper
Invariante di classe	/

Nome metodo	**+ saveStats(Character mainCharacter, int numLivello)**
Descrizione	Metodo che permette il salvataggio del livello e delle statistiche del giocatore in un file .xml
Pre-condizione	mainCharacter e numLivello non devono essere nulli
Post-condizione	/

Nome metodo	**+ loadStats():SaveWrapper**
Descrizione	Metodo che permette di riprendere tutte le statistiche del giocatore ed il livello da cui ripartirà a giocare
Pre-condizione	/
Post-condizione	Il SaveWrapper non deve essere nullo

4. Class Diagram Ristrutturato


Arch
- fromTile: Node
- toTile: Node
- cost: float
+ new()
+ render(ShapeRenderer): void
+ getters()
Node
- x: int
- y: int
- index: int
+ new()
+ getter()/setter()
+ isAdjacentTo(Node, int)
Villain
- actionArea: Rectangle
- currentPath: GraphPath<Node>
- walking: boolean
- randDestination: int
- startTime: float
+ new()
+ getter()/setter()
+ checkDirection(Character): void
+ getNextStep(float, Character): TextureRegion
Guida
Composizione
Knight
- loneNode: Node
+ new()
+ getter()/setter()
+ getLoneNode(): Node
TilesGraph
- tileHeuristic: TileHeuristic
- tiles: List<Node>
- connections: List<Arch>
- nodesMap: ObjectMap<Node, Array<Connection<Node>>>
- lastNodeIndex: int
+ new()
+ getIndex(Node): int
+ getNodeCount(): int
+ getConnections(Node): Array<Connection<Node>>
+ addTiles(Node): void
+ connectAdjacentNodes(Node, Node): void
+ findPath(Node, Node): GraphPath<Node>
+ getTiles(): List<Node>
+ hasNode(Node): boolean
TileHeuristic
+ estimate(Node, Node)
Extends
Character
- x: float
- y: float
- walkingAnimations: Animation
- runningAnimations: Animation
- attackingAnimations: Animation
+ new()
+ getter()/setter()
+ setPosition(float, float): void
+ setPreviousPosition(float, float): void
+ decreaseHealth(int): void
+ checkAttack(Character): boolean
AudioManager
- instance: AudioManager
- hit: Sound
- miss: Sound
- gemPickup: Sound
+ new()
+ getInstance(): AudioManager
+ playHit(): void
+ playMiss(): void
+ playGemPickup(): void
PauseMenu
- skin: Skin
- font: BitmapFont
- pixelArenaGame: PixelArenaGame
+ new()
+ render(): void
MusicManager
- instance: MusicManager
- gameOverMusic: Music
- musicIntro: Music
- musicBattle: Music
+ new()
+ getInstance(): MusicManager
+ playIntro(): void
+ stopIntro(): void
+ playGameOver(): void
+ stopGameOver(): void
+ playBattle(): void
+ stopBattle(): void
AGGIORNA
InputManager
- character: Character
- graphVillainHashMap: HashMap<TilesGraph, Villain>
- leftright/up/down Pressed: boolean
- keyframe: float
- startTime: float
- audioManager: AudioManager
+ new()
+ getter()/setter()
+ keyDown(int): boolean
+ keyUp(int): boolean
+ toggleRun()
+ nextFrame(float): textureRegion
USA
COMPONE
CREA
SaveManager
- PREFERENCES_NAME: String
+ saveStats(): void
+ loadStats(): SaveWrapper
CHIAMA
SaveWrapper
- numLevel: int
- mainCharacter: Character
+ new()
+ getter()/setter()
PRENDE
PixelArenaGame
- isPaused: boolean
+ create(): void
+ isPaused(): boolean
+ togglePause: void
PASSA
AGGREGA
LevelScreen
- game: PixelArenaGame
- batch: SpriteBatch
- camera: OrthographicCamera
- renderer: OrthogonalTiledMapRenderer
- backGroundImageTexture: Texture
- stateTime: float
- mainCharacter: Character
- currentFrame: TextureRegion
- currentFrameVillain: TextureRegion
- inputManager: inputManager
- enemies: List<Villain>
- graphs: List<TilesGraph>
- graphVillainHashMap: HashMap<TilesGraph, Villain>
- level: Level
- musicManager: MusicManager
- audioManager: AudioManager
- hud: LevelHud
- pauseMenu: PauseMenu
+ new()
+ update(): void
+ mapCollision(): void
+ characterCollision(): void
+ checkHealingBases(): void
+ updateAI(): void
+ checkGems(): void
Livello
- numLivello: int
- collisionObjects: List<Rectangle>
- healingBases: List<Rectangle>
- map: TiledMap
- gems: List<Gemma>
+ new()
+ getter()/setter()
+ getNumLivello(): int
+ getCollisionObject(): List<Rectangle>
+ getHealingBases(): List<Rectangle>
+ getMap(): TiledMap
+ getGems(): List<Gemma>
CONTIENE
Gemma
- hitbox: Rectangle
- texture: Texture
+ new()
+ getter()/setter()
+ getHitbox(): Rectangle
+ getTexture(): Texture
ChooseCharacterScreen
- game: Game
- backGroundImageTexture: Texture
- stage: Stage
- font: BitmapFont
- backgroundDrawable: TextureRegionDrawable
- skin: Skin
- k1: Knight
- character: Character
+ new()
GameOverScreen
- game: Game
- backGroundImageTexture: Texture
- stage: Stage
- font: BitmapFont
- backgroundDrawable: TextureRegionDrawable
- skin: Skin
- numLevel: int
- mainCharacter: Character
+ new()
MainMenuScreen
- game: Game
- backGroundImageTexture: Texture
- stage: Stage
- font: BitmapFont
- backgroundDrawable: TextureRegionDrawable
- skin: Skin
- numLevel: int
- mainLabel: Texture
- musicManager
+ new()
UpgradeScreen
- game: Game
- backGroundImageTexture: Texture
- stage: Stage
- font: BitmapFont
- backgroundDrawable: TextureRegionDrawable
- skin: Skin
- numLevel: int
- mainCharacter: Character
- musicManager: MusicManager
- atkButton: TextButton
- healthButton: TextButton
- speedButton: TextButton
- numGems: TextButton
+ new()
+ update(): void
CHIAMA
Implements
LevelHud
- skin: Skin
- font: BitmapFont
- healthPoints: TextButton
- gemsCounter: TextButton
- numLevel: TextButton
- enemiesNum: TextButton
+ new()
+ setter()
+ render(): void
GameLoader
- enemies: List<Villain>
- game: Game
- mainCharacter: Character
- numLevel: int
- maxNumLevel: int
+ new()
+ getter()/setter()
+ load(): void
CARICA
Screen
+ show(): void
+ render(): void
+ resize(): void
+ pause(): void
+ resume(): void
+ hide(): void
+ dispose(): void
CHIAMA
PASSA

5. Elementi di riuso

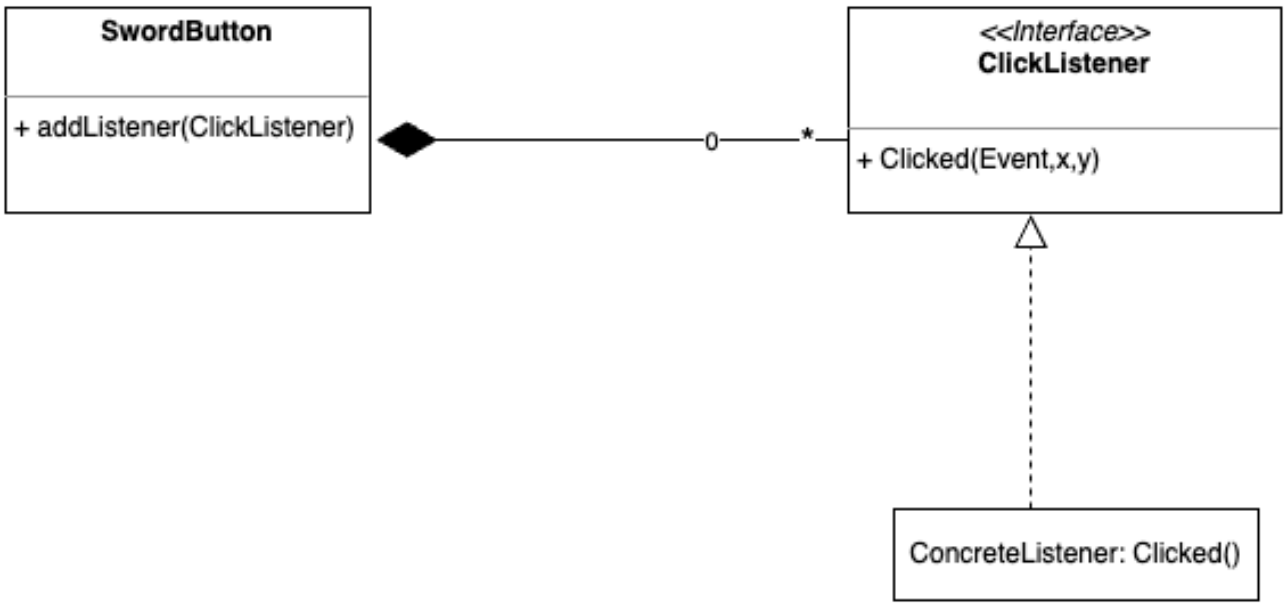
5.1 Design Pattern usati

All'interno del sistema sono stati usati dei pattern per risolvere alcuni problemi di programmazione e per rendere il codice più comprensibile.

Observer

Poiché avevamo bisogno di aggiornare elementi a schermo nelle varie schermate dei vari menù di gioco, ma sopratutto nella schermata di potenziamento del personaggio, abbiamo optato di utilizzare l'Observer Design Pattern.

Dovendo cambiare stato all'oggetto Character, una volta premuto il bottone questo va a richiamare l'interfaccia ClickListener con il metodo clicked(), dove poi abbiamo implementato ciò che doveva aggiornare tramite @Override.
Lo stesso viene utilizzato anche con altri bottoni presenti nelle varie interfacce di gioco, sia per cambiare Screen che per aggiornare statistiche di gioco e del giocatore.



5.2 Componenti terzi

All'interno del sistema è stato usato un componente COTS necessario al corretto funzionamento dell'intero motore di gioco e della logica di gioco: Tiled.

Tiled è un editor di livelli 2D. La sua funzione primaria è quella di editare tile della mappa in varie forme, restando molto flessibile ma intuitivo per gli utenti.

6. Glossario

Termine	Definizione
Package	Raggruppamento di classi ed interfacce
Three-Layer	Modello di organizzazione del codice applicativo basato sulla separazione delle funzionalità logiche
Componenti COTS	Componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti
Assets	Risorse multimediali presenti all'interno dei file di gioco
Viewport	L'area visibile dall'utente sullo schermo, che varia da dispositivo in dispositivo
Texture	In informatica, piccola immagine ripetuta moltissime volte, fino a riempire la superficie di un oggetto virtuale
Euristica	Nell'ambito di una scienza, la metodologia di ricerca di fatti o verità, ovvero di fonti e documenti, preliminare allo studio specifico