Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria “Enzo Ferrari”

HeroQuest

Specifica dei Requisiti

Dichiaro che questo elaborato è frutto del mio personale lavoro, svolto sostanzialmente in maniera individuale e autonoma.

Parigi Luca

Matricola 118307

0. Prefazione

Il seguente documento costituisce la descrizione del software “HeroQuest”, comprensivo di Software Requirements Specification (SRS) e opportuni diagrammi UML.

La specifica dei requisiti è stata scritta secondo le direttive “IEEE Recommended Practice for Software Requirements Specification – IEEE Std 830 – 1998", che stabiliscono le linee guida per la stesura di un documento SRS completo, chiaro ed esaustivo.

Lo scopo è la descrizione e specificazione delle informazione del software, come funzionalità, vincoli e requisiti.

INDICE

1. INTRODUZIONE

La presente sezione ha lo scopo di riportare la visione globale dell'intero documento di Specifica dei Requisiti. La struttura del documento è quella suggerita dallo standard ANSI/IEEE 830 noto come SRS (Software Requirements Specifications).

1.1 Obiettivo

Lo scopo del presente documento è presentare, nel modo più preciso, consistente, non ambiguo e comprensibile, la struttura generale del programma HeroQuest.

Va tenuto presente che l'approccio seguito per lo sviluppo del software è di tipo prototipale, quindi nuovi requisiti potranno essere introdotti in seguito, oltre a tutti i vincoli che fino a questo momento non sono ancora stati individuati. Il presente documento, come il software, è rivolto a chiunque voglia utilizzarlo.

1.2 Campo d'Applicazione

Il programma sviluppato è HeroQuest, un semplice videogioco 2D (in due dimensioni) a scorrimento orizzontale, che propone al giocatore una serie di livelli con una ambientazione retrò in stile 8 bit.

Tale prodotto è stato progettato per un utilizzo extra-lavorativo. Si intende analizzare esclusivamente le funzioni principali del videogioco.

1.3 Definizioni, Acronimi e Abbreviazioni

Termini	**Definizioni**
HeroQuest, software, videogioco	Termini utilizzati per riferirsi al soggetto del presente documento.
Utente	Persona che utilizza il software.
Player (giocatore), eroe	L'eroe, il personaggio i cui movimenti sono controllati dall'utente. L'eroe è in grado di muoversi orizzontalmente, di saltare, dotato di attacchi a corto e ampio raggio. Ha un certo quantitativo iniziale di punti vita e punti mana.
Enemy (nemico)	Insieme di personaggi non controllati dall'utente generati all'interno dei vari livelli. Ciascuno infligge un certo quantitativo di danni al giocatore se entra in collisione con questo. Alcuni possono attaccare. Possono essere eliminati dall'eroe. Vi sono varie tipologie di nemici: Nightmare, Skeleton, Skull, etc...
Boss	Tipologia di nemico più resistente e più potente. Se presente nel livello, il giocatore deve eliminarlo per poter accedere al livello successivo del videogioco.
Potion (pozione)	Elemento generato all'interno della mappa. Se raccolto dall'eroe questo recupera un certo quantitativo di punti vita o mana.

Teleport (teletrasporto)	Elemento situato a fine del livello, se l'eroe entra in collisione con questo accede al livello successivo del videogioco. Talvolta viene generato solo in seguito all'eliminazione del boss del livello.
Game (gioco)	Istanza della sessione di gioco.
Level (livello)	Fase del videogioco. Al termine del livello viene mostrata una schermata con i punteggi (nemici sconfitti, tempo di completamento).
HUD	Head-Up Display, durante il gioco permette di visualizzare i punti di vita e di mana rimanenti e totali dell'eroe.
Map (mappa)	Luogo in cui si muovono e interagiscono l'eroe e i nemici.
GameOver	Nel caso in cui il giocatore perda tutti i punti vita termina il livello.

1.4 Fonti

Il documento è stato redatto sulla base delle seguenti fonti, usate durante l'analisi dei requisiti e l'implementazione del codice:

- Specifiche dettate da Parigi Luca, sviluppatore del software.
- IEEE Recommended Practice for Software Requirements Specification IEEE Std 830 – 1998.

1.5 Struttura del Documento

Il documento qui presente è costituito di 5 macro-sezioni. Da questo punto in poi, troveremo, altri 4 capitoli:

- **Capitolo 2, Descrizione Generale:** fornisce una descrizione generale del software, delle sue funzionalità, particolarità e dell'interfaccia con l'utente;
- **Capitolo 3, SRS:** racchiude una presentazione dei requisiti funzionali e non funzionali del programma la fine di racchiudere tutte le richieste del committente;
- **Capitolo 4, Data Model:** mostra una vista generale del funzionamento del gioco mediante i diagrammi UML principali;
- **Capitolo 5, Design Pattern:** descrive i design pattern usati durante lo sviluppo al fine di risolvere particolari problemi.

2. DESCRIZIONE GENERALE

Descriviamo ora i principali fattori che riguardano il software.

2.1 Descrizione del Software

Il software qui presentato è un semplice videogioco 2D a scorrimento orizzontale. Il concetto alla base del gioco è impersonare un eroe e attraversare i vari livelli superandone gli ostacoli, sopravvivendo ai nemici ed eliminando l'eventuale boss di fine livello.

L'eroe ha un'iniziale quantità di vita che può diminuire collidendo con i nemici in movimento all'interno del livello e/o recuperare con le pozioni. Se perde tutti i punti vita il livello termina automaticamente e può essere ricaricato. L'eroe dispone di due tipi di attacco, uno a corto raggio (attacco con spada) e uno ad ampio raggio (palla di fuoco) che consuma mana per ogni utilizzo.

Allo stato attuale dello sviluppo il videogioco contiene solo due livelli.

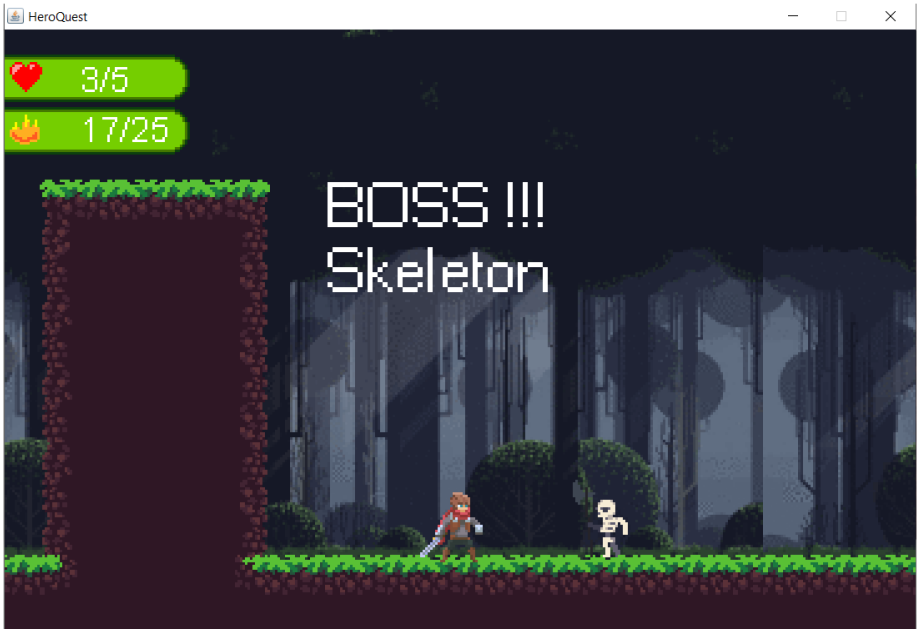


Figura 1: Schermata di gioco, boss scheletro.

2.1.1 Interfaccia sistema/utente

L'interfaccia utente dovrà presentare un menu principale molto semplice, dove l'utente potrà o iniziare la partita, consultare i comandi del videogioco o terminare l'esecuzione del software.

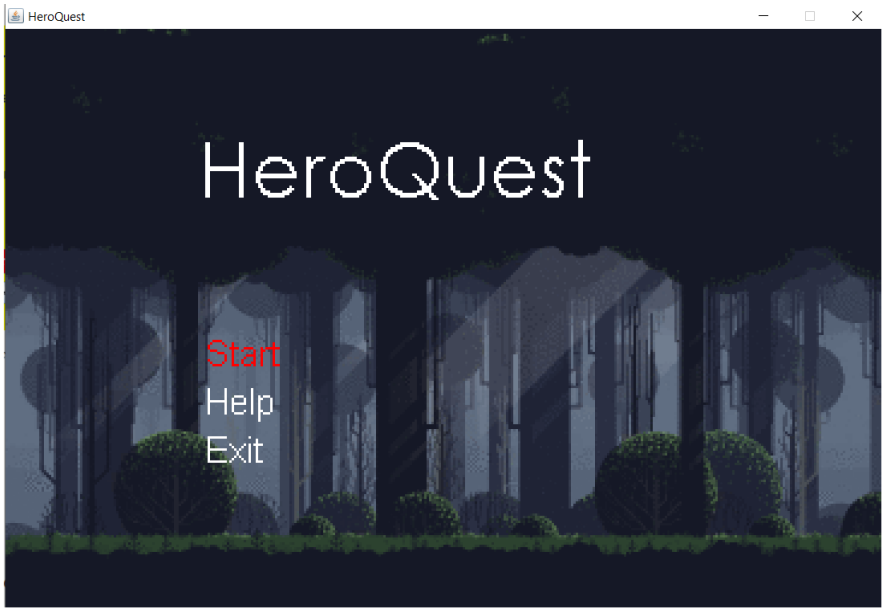


Figura 2: Schermata del menu principale

Dovrà anche essere possibile mettere in pausa il videogioco durante qualsiasi momento della partita.

2.1.2 Interfaccia Hardware

Il prodotto dovrà supportare unicamente la tastiera per giocare e navigare nei menu.

2.1.3 Interfaccia Software

Il gioco sarà interamente programmato in JAVA e richiede sul proprio dispositivo una versione installata di JAVA 8.x o superiore.

2.1.4 Interfaccia di Comunicazione

HeroQuest essendo un'applicazione per utenti singoli non richiede alcuna interfaccia di comunicazione.

2.1.5 Vincoli relativi all'occupazione di memoria (N/A)

2.1.6 Operazioni

Le operazioni principali effettuate dal software sono descritte al paragrafo 2.1.1 di questo documento.

2.1.7 Vincoli di Installazione

Il software non richiede particolari vincoli poiché è distribuito come file eseguibile e non richiede nessun sistema di installazione.

2.2 Macro-Funzionalità del Sistema

Le principali funzionalità di HeroQuest sono:

- Avviare una nuova partita
- Giocare
- Calcolare i punteggi di fine livello

2.3 Caratteristiche degli Utenti

L'utente non necessita di caratteristiche particolari per poter giocare. I comandi sono descritti all'interno del software.

2.4 Vincoli generali (N/A)

2.5 Assunzioni e Dipendenze

Qualunque modifica su HeroQuest avrà ripercussioni sul presente documento. In caso di modifica sarà necessario un aggiornamento di questo elaborato.

2.6 Requisiti da analizzare in futuro

- **Storia:** implementazione di una storia, anche basilare, in modo da rendere più interessante il progredire tra i livelli.
- **Musica:** implementazione di musica in background e effetti sonori per le azioni dell'eroe.
- **Nuovi livelli e ambientazioni:** implementazione di livelli differenti per rendere più vario il videogioco.
- **Skin:** inserimento di diverse vesti grafiche per l'eroe impersonato dall'utente.

3. SPECIFICHE FUNZIONALI E NON FUNZIONALI DEL SISTEMA

3.1 Requisiti funzionali

3.1.1 Gestione del menu principale

RF01	Gestione menu principale.
Introduzione	*Attori coinvolti* Giocatore. *Descrizione generale della funzione* Il menu principale viene avviato automaticamente all'avvio del software. Il menu principale presenta al giocatore le seguenti funzioni: • Avvio gioco • Informazioni • Chiudi
Input	*Descrizione generica dei dati* Pressione del tasto ENTER (tastiera). Pessione dei tasti FRECCIA-SU/FRECCIA-GIU' (tastiera).
Descrizione (Processo)	*Sequenza di operazioni* Il sistema deve stampare il titolo del gioco "HeroQuest" con le possibili interazioni per creare una nuova partita, visualizzare i comandi, uscire dal software. Gli effetti dei tasti sono: • FRECCIA-SU/FRECCIA-GIU': muoversi nel menu. • ENTER: selezionare l'opzione nel menu.
Output	A seconda dell'opzione scelta: • Inizializzazione di una nuova partita. • Visualizzazione comandi. • Terminazione dell'esecuzione del software.

3.1.2 Inizializzazione di una nuova partita

RF02	Inizializzazione nuova partita, qualsiasi livello.
Introduzione	*Attori coinvolti* Giocatore. *Descrizione generale della funzione* Viene avviata una nuova partita dal menu principale.
Input	Nessuno
Descrizione (Processo)	*Sequenza di operazioni* Il sistema deve: • Caricare lo sfondo e la mappa. • Inizializzare il giocatore e posizionarlo all'inizio della mappa. • Inizializzare i nemici e posizionarli nella mappa. • Inizializzare gli altri elementi della mappa (pozioni, teletrasporto).
Output	Viene avviato un livello del gioco.

3.1.3 Gestione del gioco

RF03	Gestione di un qualsiasi livello.
Introduzione	*Attori coinvolti* Giocatore. *Descrizione generale della funzione* Il giocatore gestisce i movimenti dell'eroe nella mappa. Lo scopo è superare gli ostacoli della mappa e sconfiggere i nemici incluso l'eventuale boss. Alcuni nemici seguono percorsi prestabiliti, altri si muovono nella direzione dell'eroe (entro un range prestabilito). Alcuni nemici attaccano, altri no.
Input	*Descrizione generica dei dati* Pressione dei tasti W, A, D, R, F, ESC (tastiera).
Descrizione (Processo)	*Sequenza di operazioni* Gli effetti dei tasti sono: • W: l'eroe salta verso l'alto. • A: l'eroe si muove verso sinistra. • D: l'eroe si muove verso destra. • R: l'eroe esegue un attacco con la spada. • F: l'eroe esegue un attacco palla di fuoco. • ESC: apre il menu di pausa. Il sistema deve: • Gestire il movimento dei nemici. • Gestire il conteggio della vita sia dell'eroe che dei nemici. • Gestire l'eliminazione dei nemici dalla mappa, quando questi perdono i loro punti vita. • Gestire le collisioni tra nemici ed eroe. • Gestire le collisioni tra gli attacchi dell'eroe e i nemici. • Gestire le collisioni tra eroe e mappa, tra nemici e mappa. • Gestire la condizione di invulnerabilità dell'eroe nel caso collida con un nemico. • Tenere conto del numero di nemici eliminati. • Cronometrare il tempo di completamento del livello.
Output	L'eroe e i nemici si muovono nella mappa.

3.1.4 Gestione schermata punteggio

RFO4	Gestione schermata punteggio
Introduzione	*Attori coinvolti* Giocatore *Descrizione generale della funzione* Viene caricata la schermata di punteggio.
Input	*Descrizione generica dei dati* Fine livello, quando l'eroe collide con il teletrasporto. Pressione del tasto ENTER (tastiera).
Descrizione (Processo)	*Sequenza di operazioni*

	Quando il giocatore termina il livello, collidendo con il teletrasporto a fine mappa, viene caricata la schermata con i punteggi relativi al livello (numero eliminazioni, tempo trascorso). Alla pressione del tasto ENTER viene caricato il livello successivo.
Output	Viene aperta la schermata di punteggio. Il programma carica il livello successivo.

3.1.5 Gestione del GameOver

RFO5	Gestione GameOver
Introduzione	*Attori coinvolti* Giocatore *Descrizione generale della funzione* Viene caricata la schermata di GameOver.
Input	*Descrizione generica dei dati* Quando i punti vita dell'eroe vengono ridotti a 0 (o meno). Pressione del tasto ENTER (tastiera). Pressione del tasto ESC (tastiera).
Descrizione (Processo)	*Sequenza di operazioni* Quando i punti vita dell'eroe vengono ridotti a 0 (o meno) viene caricata la schermata di GameOver, con la possibilità di ricaricare dall'inizio il livello con la pressione del tasto ENTER, o ritornare al menu principale con la pressione del tasto ESC.
Output	Viene caricata la schermata di GameOver. Il programma ricarica il livello o il menu principale.

3.1.6 Gestione del menu di pausa

RFO6	Gestione del menu di pausa
Introduzione	*Attori coinvolti* Giocatore *Descrizione generale della funzione* Viene caricata la schermata del menu di pausa.
Input	*Descrizione generica dei dati* Pressione del tasto ESC (tastiera). Pressione del tasto W (tastiera).
Descrizione (Processo)	*Sequenza di operazioni* Durante il livello il giocatore può mettere in pausa il gioco alla pressione del tasto ESC. Con la pressione del tasto ESC si ritorna al gioco nell'esatto punto in cui era stato messo in pausa. Con la pressione del tasto W si ritorna al menu principale.
Output	Viene caricata la schermata di pausa. Viene ripreso il gioco o caricato il menu principale.

3.2 Requisiti non funzionali

3.2.1 Portabilità

RNF01	Portabilità
Descrizione	Per garantire il funzionamento su diversi sistemi operativi sarà necessario adottare JAVA come linguaggio di programmazione per il software.

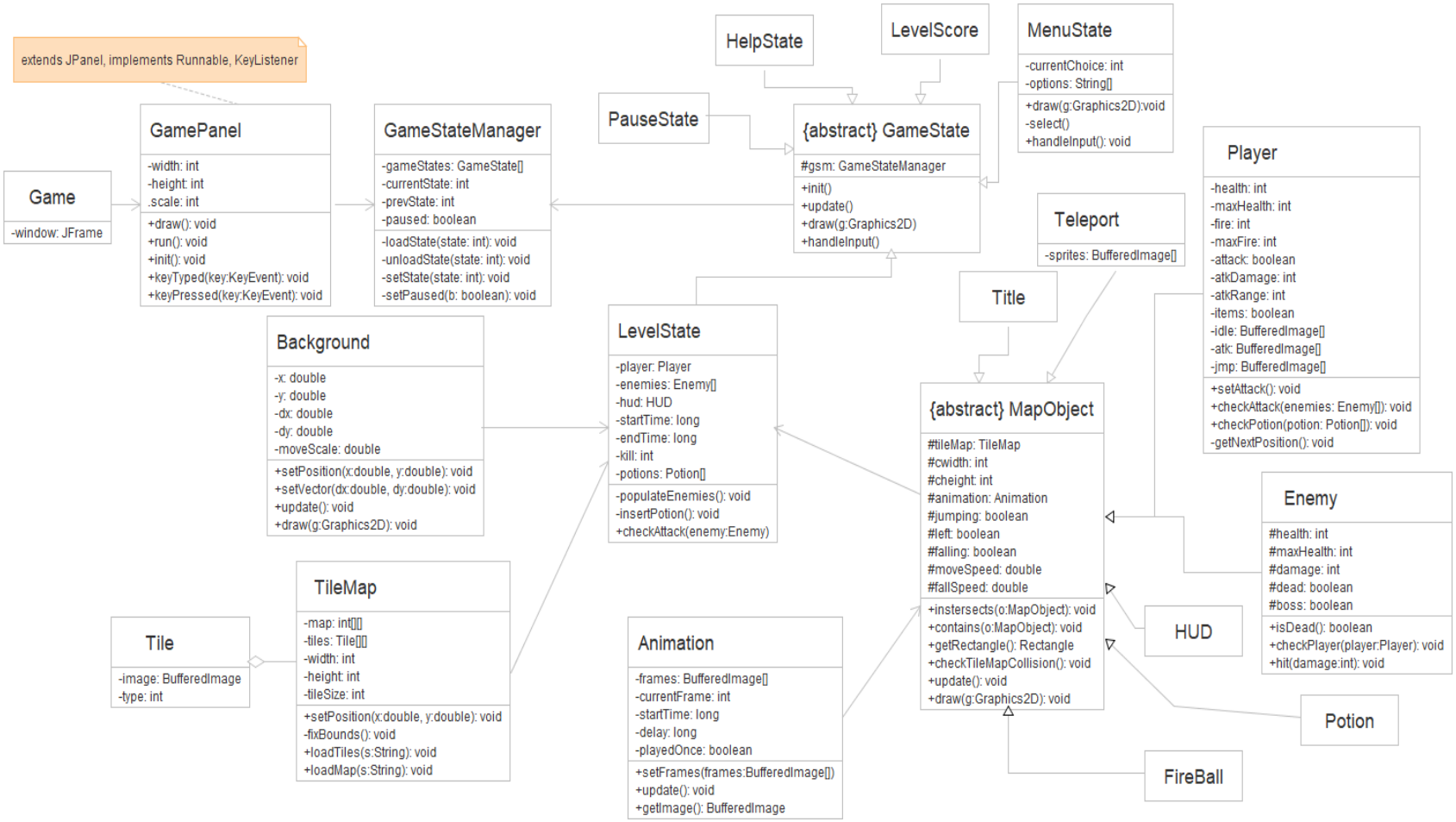
3.2.2 Manutenibilità

RNF02	Manutenibilità
Descrizione	Il software dovrà essere mantenibile in modo da poter aggiungere in futuro nuove funzionalità.

4. DIAGRAMMI

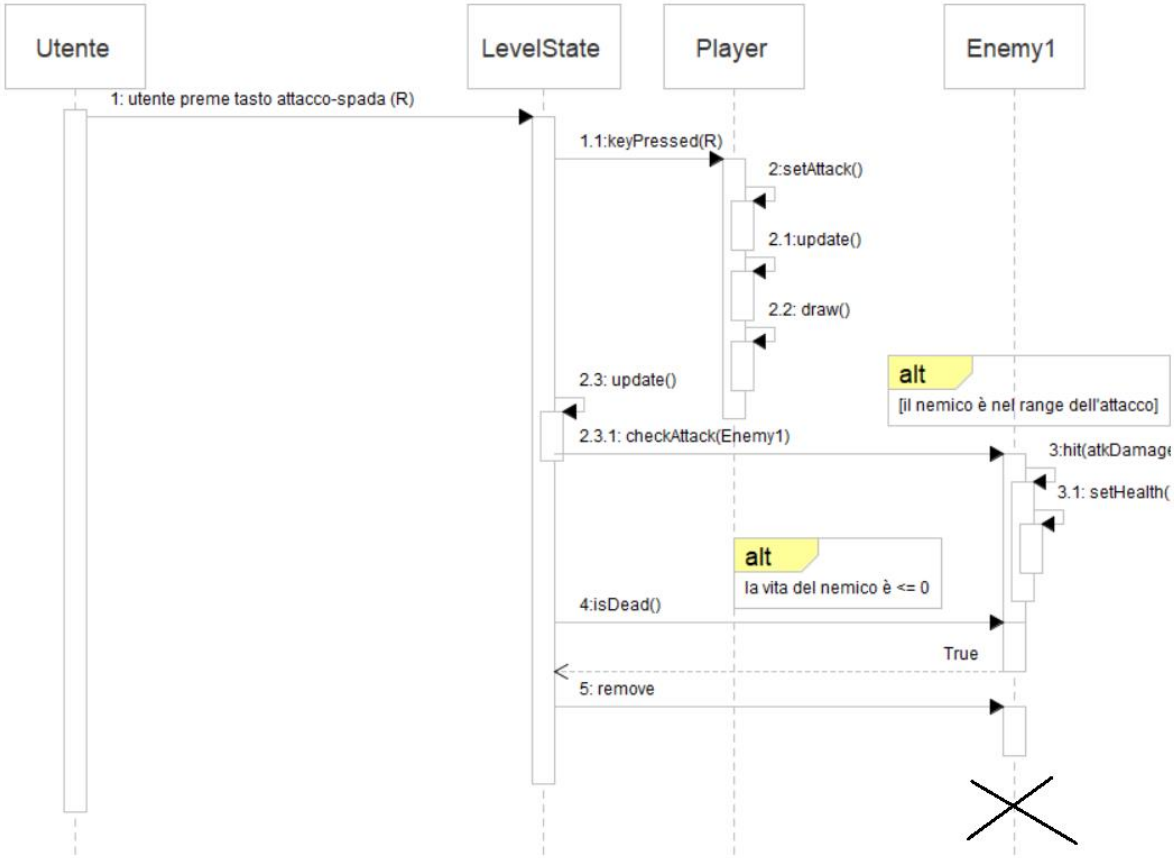
4.1 Use Case Diagram

4.2 Class Diagram

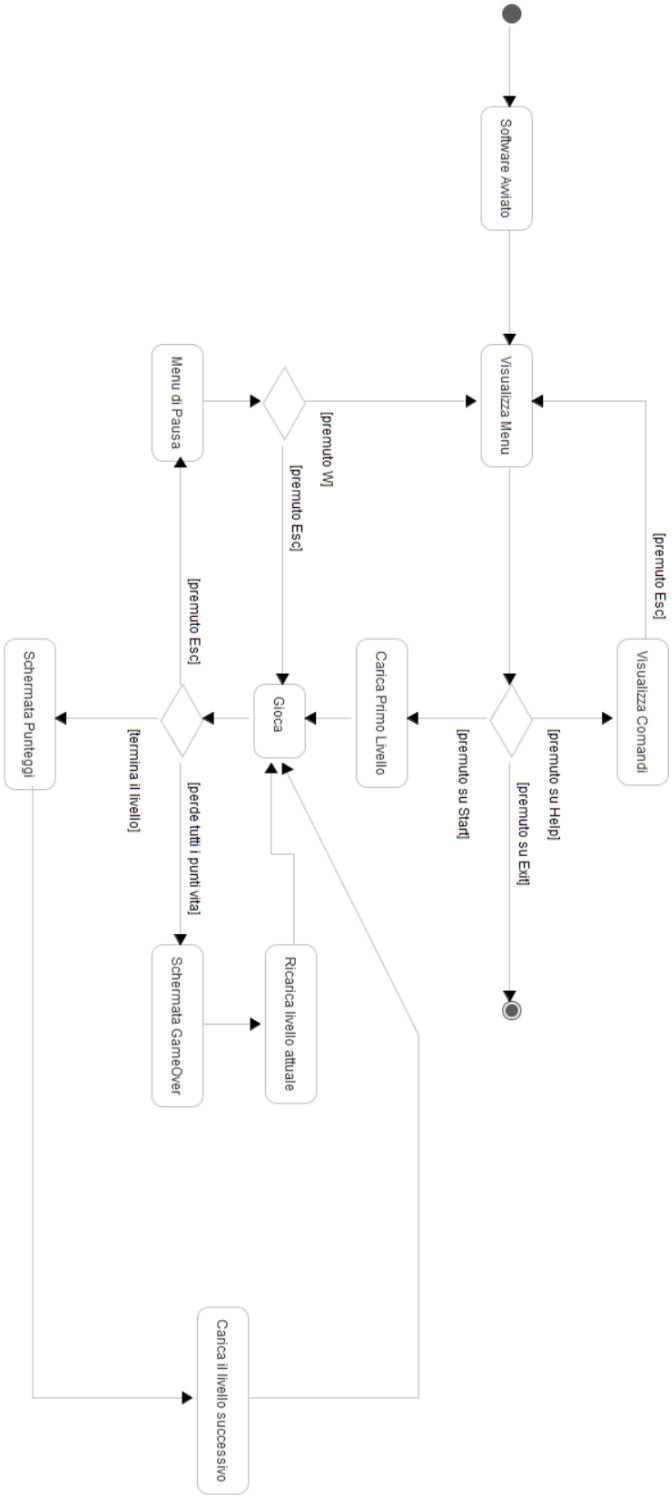

extends JPanel, implements Runnable, KeyListener
Game
-window: JFrame
GamePanel
-width: int
-height: int
.scale: int
+draw(): void
+run(): void
+init(): void
+keyTyped(key:KeyEvent): void
+keyPressed(key:KeyEvent): void
GameStateManager
-gameStates: GameState[]
-currentState: int
-prevState: int
-paused: boolean
-loadState(state: int): void
-unloadState(state: int): void
-setState(state: int): void
-setPaused(b: boolean): void
HelpState
LevelScore
MenuState
-currentChoice: int
-options: String[]
+draw(g:Graphics2D):void
-select()
+handleInput(): void
PauseState
{abstract} GameState
#gsm: GameStateManager
+init()
+update()
+draw(g:Graphics2D)
+handleInput()
Player
-health: int
-maxHealth: int
-fire: int
-maxFire: int
-attack: boolean
-atkDamage: int
-atkRange: int
-items: boolean
-idle: BufferedImage[]
-atk: BufferedImage[]
-jmp: BufferedImage[]
+setAttack(): void
+checkAttack(enemies: Enemy[]): void
+checkPotion(potion: Potion[]): void
-getNextPosition(): void
Teleport
-sprites: BufferedImage[]
Title
Background
-x: double
-y: double
-dx: double
-dy: double
-moveScale: double
+setPosition(x:double, y:double): void
+setVector(dx:double, dy:double): void
+update(): void
+draw(g:Graphics2D): void
LevelState
-player: Player
-enemies: Enemy[]
-hud: HUD
-startTime: long
-endTime: long
-kill: int
-potions: Potion[]
-populateEnemies(): void
-insertPotion(): void
+checkAttack(enemy:Enemy)
{abstract} MapObject
#tileMap: TileMap
#cwidth: int
#cheight: int
#animation: Animation
#jumping: boolean
#left: boolean
#falling: boolean
#moveSpeed: double
#fallSpeed: double
+instersects(o:MapObject): void
+contains(o:MapObject): void
+getRectangle(): Rectangle
+checkTileMapCollision(): void
+update(): void
+draw(g:Graphics2D): void
Enemy
#health: int
#maxHealth: int
#damage: int
#dead: boolean
#boss: boolean
+isDead(): boolean
+checkPlayer(player:Player): void
+hit(damage:int): void
HUD
Potion
FireBall
TileMap
-map: int[][]
-tiles: Tile[][]
-width: int
-height: int
-tileSize: int
+setPosition(x:double, y:double): void
-fixBounds(): void
+loadTiles(s:String): void
+loadMap(s:String): void
Tile
-image: BufferedImage
-type: int
Animation
-frames: BufferedImage[]
-currentFrame: int
-startTime: long
-delay: long
-playedOnce: boolean
+setFrames(frames:BufferedImage[])
+update(): void
+getImage(): BufferedImage

4.3 Sequence Diagram

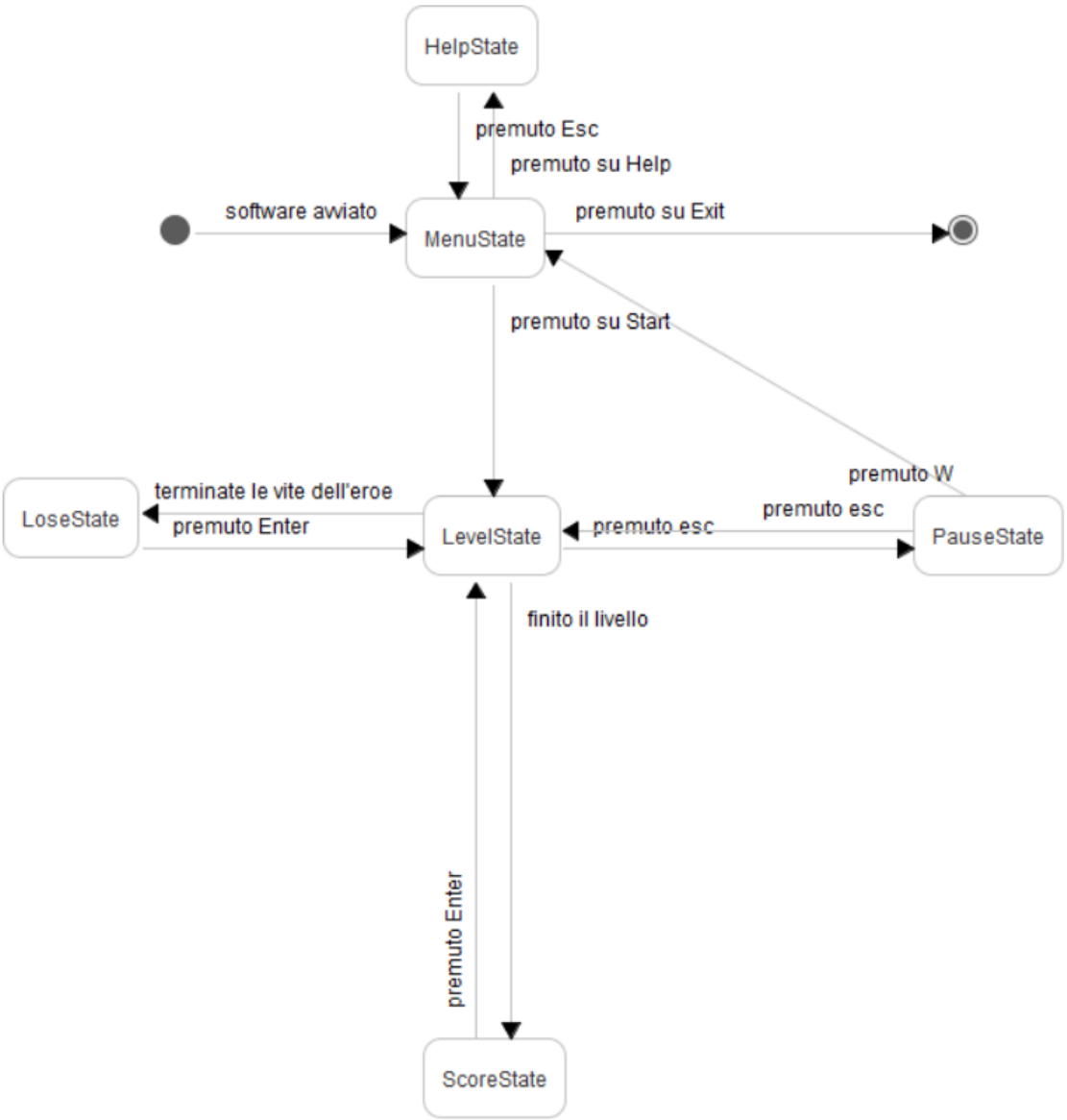
Il Player attacca un nemico, eliminandolo.



4.4 Activity Diagram



4.5 State Diagram



5. DESIGN PATTERN

Il software, attraverso i seguenti design pattern, può essere strutturato in modo da migliorarne la modularità e la manutenibilità.

5.1 Strategy Pattern

Si è scelto di modellare le classi relative ai nemici con uno Strategy Pattern, poiché i nemici possono avere vari stili di combattimento. Per esempio alcuni tipi di nemici saranno passivi e non attaccheranno l'eroe, mentre agli altri sarà associato uno stile di attacco.

Sarà poi successivamente possibile implementare nuovi tipi di attacco e nuovi tipi di nemici.

5.1.1 Class Diagram

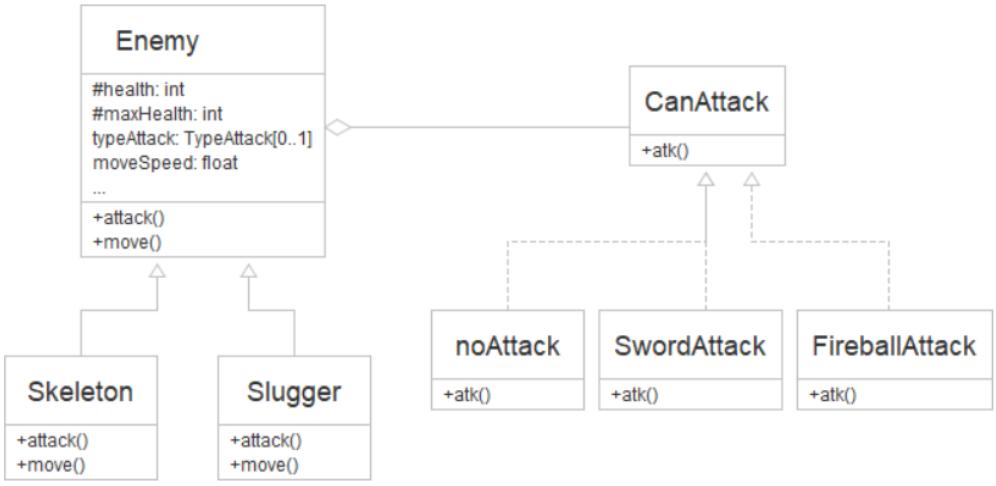


Figura 3: sprites dell'animazione di attacco dello scheletro.

5.1.2 Codice

```
public abstract class Enemy {
    TypeAttack typeAttack;
    //altre variabili

    public void Enemy() {
        //codice
    }

    public void attack() {
        public void typeAttack.attack();
    }
    //altri metodi
}


public class Skeleton extends Enemy {

    public Skeleton() {
        typeAttack = new SwordAttack();
    }
}

public interface TypeAttack {
    public void attack();
}


public class SwordAttack implements TypeAttack {

    public void atk() {
        //codice
    }
}
```

5.2 State Pattern

Si è scelto di modellare le classi relative ai vari stati assunti dal gioco (come i vari menu e i livelli) con uno State Pattern. Tutti gli stati condividono e implementano a modo le quattro funzioni indicate nel class diagram nell'interfaccia GameState. Ogni classe aggiunge eventualmente funzioni e variabili.

Il GameStateManager gestisce la transizione da uno stato all'altro in base alle azioni dell'utente.

5.2.1 Class Diagram