**Survival Game**

**Manuale Tecnico**

**Introduzione**

Il presente manuale tecnico descrive lo sviluppo del progetto **"Survival Game"**, un gioco d'azione in stile *Vampire Survivors*, implementato utilizzando il linguaggio **Java** con supporto grafico fornito da **Swing** e **AWT**.

L'applicazione segue l'architettura **Model-View-Controller (MVC)**, che separa logicamente la gestione dei dati, la visualizzazione e le interazioni dell'utente, garantendo un design modulare e facilmente estensibile.

Il gioco mette il giocatore nei panni di un **UFO** che deve sopravvivere contro **ondate di asteroidi** evitando collisioni e raccogliendo oggetti per ripristinare la vita. Le principali funzionalità includono:

* **Movimento fluido del personaggio** tramite input da tastiera.
* **Gestione grafica** degli elementi di gioco.
* **Logica di gioco** per collisioni, raccolta oggetti e stato della partita.

Questo documento fornisce una **panoramica dettagliata** della struttura del codice, della logica implementata e delle funzionalità principali. Inoltre, offre indicazioni per l'esecuzione, la manutenzione e l'estensione del progetto.

**Indice delle Classi**

**Struttura del Progetto**

Il progetto **Survival Game** è organizzato nella seguente struttura a cartelle:

/giocoTipoVampireSurvivors

│

├── src

│ ├── controller

│ │ └── GameController.java

│ │

│ ├── main

│ │ └── Main.java

│ │

│ ├── model

│ │ ├── Nemico.java

│ │ ├── Personaggio.java

│ │ ├── Pozione.java

│ │ └── Proiettile.java

│ │

│ ├── view

│ │ ├── ControlliPanel.java

│ │ ├── GameOverPanel.java

│ │ ├── GiocoPanel.java

│ │ ├── HUD.java

│ │ ├── MenuPrincipale.java

│ │ └── PausePanel.java

│

├── asset

│ └── img

│ └── navicella50x50.png

│ └── (altre immagini...)

│

└── documentazione

└── manuale\_tecnico.pdf

**Descrizione delle Classi**

**1. Controller**

* **GameController.java**: Gestisce la logica di gioco principale, coordinando gli input e aggiornando il modello e la vista.

**2. Main**

* **Main.java**: Classe principale per avviare l'applicazione.

**3. Model**

Classi che rappresentano gli oggetti logici del gioco:

* **Nemico.java**: Rappresenta i nemici che si muovono verso il giocatore.
* **Personaggio.java**: Gestisce il giocatore (UFO) e le sue caratteristiche (posizione, movimento, vita).
* **Pozione.java**: Rappresenta gli oggetti di rigenerazione della vita.
* **Proiettile.java**: Gestisce i proiettili sparati dal personaggio.

**4. View**

Classi dedicate alla visualizzazione grafica e all'interfaccia utente:

* **ControlliPanel.java**: Mostra i controlli del gioco.
* **GameOverPanel.java**: Schermata di fine partita.
* **GiocoPanel.java**: Pannello principale del gioco dove vengono disegnati gli oggetti.
* **HUD.java**: Mostra informazioni sullo stato della partita (vita, punteggio, ecc.).
* **MenuPrincipale.java**: Schermata principale del menu.
* **PausePanel.java**: Schermata di pausa del gioco.

**5. Asset**

* **img/**: Contiene le risorse grafiche del gioco (es. immagini del personaggio, nemici, pozioni).

Immagine che contiene linea, schermata, Diagramma, diagramma

Descrizione generata automaticamente

MODEL

**Classe Personaggio**

**1. Descrizione della Classe**

La classe Personaggio appartiene al **Model** del progetto, seguendo l'architettura **MVC (Model-View-Controller)**. Essa rappresenta il **personaggio principale** del gioco (es. UFO) e gestisce i seguenti aspetti:

* Posizione e movimento del personaggio.
* Stato della vita (vita corrente e vita massima).
* Velocità di movimento.
* Caricamento e gestione dell'immagine associata.
* Disegno grafico sulla finestra di gioco.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | int | posizioneX | Coordinata X del personaggio. |
| private | int | posizioneY | Coordinata Y del personaggio. |
| private | int | vitaCorrente | Vita corrente del personaggio. |
| private | int | vitaMassima | Vita massima del personaggio. |
| private | int | velocita | Velocità di movimento del personaggio. |
| private | int | dimensione | Dimensione dell'icona grafica del personaggio. |
| private | Image | immagine | Immagine grafica del personaggio caricata. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| Personaggio(int startX, int startY) | Inizializza la posizione, la vita e carica l'immagine del personaggio. |
| Personaggio() | Costruttore vuoto. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| void muovi(int dx, int dy, int larghezzaGioco, int altezzaGioco) | Aggiorna la posizione del personaggio in base ai movimenti e controlla i **bordi** della finestra di gioco. |
| void disegna(Graphics g) | Disegna il personaggio sullo schermo usando l'immagine caricata o un fallback grafico. |

**Metodi Getter e Setter**

| **Metodo** | **Descrizione** |
| --- | --- |
| getPosizioneX() | Restituisce la coordinata X. |
| setPosizioneX(int x) | Imposta la coordinata X. |
| getPosizioneY() | Restituisce la coordinata Y. |
| setPosizioneY(int y) | Imposta la coordinata Y. |
| getVitaCorrente() | Restituisce la vita corrente. |
| setVitaCorrente(int vita) | Imposta la vita corrente (non eccede la vita massima). |
| getVitaMassima() | Restituisce la vita massima. |
| setVitaMassima(int vita) | Imposta la vita massima. |
| getVelocita() | Restituisce la velocità del personaggio. |

**3. Dettagli Tecnici**

* **Movimento**:
  + Il metodo muovi aggiorna le coordinate del personaggio in base ai valori dx e dy forniti.
  + La velocità è moltiplicata per i movimenti per controllare lo spostamento.
  + Viene implementato un controllo per **evitare di uscire dai bordi della finestra di gioco**:
    - **Larghezza limite**: 1280px.
    - **Altezza limite**: 720px meno spazio riservato per l'HUD.
* **Gestione dell'Immagine**:
  + L'immagine è caricata con ImageIO da una cartella /asset/img/.
  + In caso di errore (immagine mancante), viene disegnato un cerchio verde di fallback.

**4. Dipendenze**

La classe dipende dalle seguenti librerie:

* **javax.swing**: Per disegnare il personaggio utilizzando Graphics.
* **java.awt**: Per gestire l'immagine e il disegno.
* **javax.imageio.ImageIO**: Per caricare l'immagine grafica.

**5. Esempio d'Uso**

java

Copia codice

Personaggio ufo = new Personaggio(100, 200); // Posizione iniziale (100, 200)

// Muove il personaggio di 1 unità a destra e in basso

ufo.muovi(1, 1, 1280, 720);

// Disegna il personaggio in una finestra di gioco

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

ufo.disegna(g);

}

**6. Potenziali Estensioni**

* **Potenziamenti**: Aggiungere metodi per aumentare velocità o vita.
* **Collisioni**: Integrare controlli per rilevare collisioni con altri oggetti (es. asteroidi).
* **Animazioni**: Aggiungere animazioni al movimento tramite **spritesheet**.

**7. Gestione Errori**

* Se l'immagine non viene trovata:
  + Viene mostrato un messaggio di errore nella console.
  + Viene utilizzato un **fallback grafico** (cerchio verde).

**Classe Nemico**

**Manuale Tecnico - Classe Nemico**

**1. Descrizione della Classe**

La classe Nemico appartiene al **Model** del progetto e rappresenta gli **asteroidi**, ovvero i nemici che il giocatore deve evitare o eliminare. Ogni nemico possiede una posizione, una velocità fissa, e un'immagine grafica caricata in modo casuale da una lista predefinita.

Questa classe gestisce:

* Il **movimento dei nemici** verso un bersaglio (come il giocatore).
* La **visualizzazione grafica** dei nemici con immagini casuali.
* Lo **stato attivo** dei nemici (ad esempio, se vengono colpiti da un proiettile).

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | int | posizioneX | Coordinata X del nemico. |
| private | int | posizioneY | Coordinata Y del nemico. |
| private | int | velocita | Velocità di movimento del nemico (costante). |
| private | int | dimensione | Dimensione (diametro) del nemico. |
| private | boolean | attivo | Stato del nemico: attivo (true) o eliminato (false). |
| private | Image | immagine | Immagine grafica associata al nemico. |
| private | static String[] | immaginiNemici | Lista dei percorsi delle immagini disponibili. |

**Manuale Tecnico - Classe Nemico**

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| Nemico(int startX, int startY) | Inizializza posizione, velocità, dimensione e carica un'immagine casuale. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| void muoviVerso(int targetX, int targetY) | Muove il nemico verso un bersaglio specifico (coordinate X e Y). |
| void disegna(Graphics g) | Disegna il nemico sulla finestra grafica, usando l'immagine o un fallback grafico. |
| void disattiva() | Imposta il nemico come **non attivo** (usato ad esempio dopo una collisione). |
| boolean isAttivo() | Restituisce lo stato del nemico: attivo o eliminato. |

**Metodi Getter e Setter**

| **Metodo** | **Descrizione** |
| --- | --- |
| getPosizioneX() | Restituisce la coordinata X. |
| setPosizioneX(int x) | Imposta la coordinata X. |
| getPosizioneY() | Restituisce la coordinata Y. |
| setPosizioneY(int y) | Imposta la coordinata Y. |
| getVelocita() | Restituisce la velocità del nemico. |
| getDimensione() | Restituisce la dimensione del nemico. |
| setAttivo(boolean attivo) | Imposta lo stato attivo del nemico. |

**3. Dettagli Tecnici**

* **Caricamento Immagine**:
  + Le immagini dei nemici vengono caricate in modo **casuale** da una lista predefinita (immaginiNemici) tramite il metodo Random.
  + Se l'immagine non è disponibile, viene gestito un errore di fallback con un rettangolo rosso.
* **Movimento**:
  + Il metodo muoviVerso muove il nemico in **direzione diagonale** verso un bersaglio.
  + La logica è basata su condizioni per determinare se il nemico deve avvicinarsi lungo X, Y, o entrambi.
* **Disegno Grafico**:
  + Se l'immagine è caricata, viene disegnata con g.drawImage.
  + In caso contrario, viene visualizzato un **rettangolo rosso** come rappresentazione di fallback.

**4. Dipendenze**

La classe dipende dalle seguenti librerie:

* **javax.imageio.ImageIO**: Per caricare immagini dalle risorse.
* **java.awt**: Per la gestione grafica (disegno di immagini e rettangoli).
* **java.util.Random**: Per selezionare casualmente un'immagine da una lista.

**5. Esempio d'Uso**

java

Copia codice

// Creazione di un nuovo nemico in posizione (100, 50)

Nemico asteroide = new Nemico(100, 50);

**Manuale Tecnico - Classe Nemico**

// Movimento del nemico verso la posizione del giocatore (300, 400)

asteroide.muoviVerso(300, 400);

// Disegno del nemico in un pannello grafico

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

asteroide.disegna(g);

}

**6. Potenziali Estensioni**

* **Velocità Variabile**: Permettere che i nemici abbiano velocità diverse per aumentare la difficoltà.
* **Animazioni**: Implementare animazioni attraverso **spritesheet** per un effetto visivo più dinamico.

**7. Gestione Errori**

* Se il caricamento dell'immagine fallisce:
  + Stampa un messaggio di errore in console.
  + Mostra un rettangolo rosso di fallback nella posizione del nemico.

**Classe Proiettile**

**Manuale Tecnico - Classe Proiettile**

**1. Descrizione della Classe**

La classe **Proiettile** appartiene al **Model** del progetto e rappresenta i **proiettili** sparati dal giocatore. Essa gestisce la posizione, la velocità, la direzione, e lo stato del proiettile (attivo o eliminato). Inoltre, include la logica per il movimento e il controllo delle collisioni con i nemici.

Questa classe è fondamentale per implementare le meccaniche di **attacco** e **collisione** del gioco.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | int | posizioneX | Coordinata X del proiettile. |
| private | int | posizioneY | Coordinata Y del proiettile. |
| private | int | velocita | Velocità del movimento del proiettile. |
| private | int | direzioneX | Direzione lungo l'asse X (-1, 0, 1). |
| private | int | direzioneY | Direzione lungo l'asse Y (-1, 0, 1). |
| private | int | dimensione | Diametro del proiettile. |
| private | boolean | attivo | Stato del proiettile: attivo (true) o disattivo. |
| private | Image | immagine | (Non usata) immagine grafica del proiettile. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| Proiettile(int startX, int startY, int direzioneX, int direzioneY) | Inizializza la posizione, la direzione e imposta la velocità e dimensione. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| void muovi() | Aggiorna la posizione del proiettile lungo le direzioni X e Y in base alla velocità. |
| void disegna(Graphics g) | Disegna il proiettile come un cerchio giallo con un bordo nero. |
| boolean controllaCollisione(Nemico nemico) | Controlla se il proiettile ha una collisione con il rettangolo del nemico. |
| void disattiva() | Disattiva il proiettile, impostando lo stato su **false**. |
| boolean isAttivo() | Restituisce lo stato del proiettile: attivo o eliminato. |

**Metodi Getter e Setter**

| **Metodo** | **Descrizione** |
| --- | --- |
| getPosizioneX() | Restituisce la coordinata X del proiettile. |
| setPosizioneX(int x) | Imposta la coordinata X. |
| getPosizioneY() | Restituisce la coordinata Y del proiettile. |
| setPosizioneY(int y) | Imposta la coordinata Y. |
| getVelocita() | Restituisce la velocità del proiettile. |
| setVelocita(int velocita) | Imposta la velocità del proiettile. |
| getDirezioneX() | Restituisce la direzione lungo X. |
| setDirezioneX(int direzione) | Imposta la direzione lungo X. |
| getDirezioneY() | Restituisce la direzione lungo Y. |
| setDirezioneY(int direzione) | Imposta la direzione lungo Y. |
| getDimensione() | Restituisce la dimensione del proiettile. |
| setDimensione(int d) | Imposta la dimensione del proiettile. |
| setAttivo(boolean attivo) | Imposta lo stato del proiettile. |

**3. Dettagli Tecnici**

* **Movimento**:
  + Il metodo muovi sposta il proiettile nella direzione specificata dai valori direzioneX e direzioneY moltiplicati per la velocità (10 di default).
* **Disegno Grafico**:
  + Il proiettile viene disegnato come un **cerchio giallo** con un bordo nero tramite i metodi fillOval e drawOval di Graphics.
* **Collisioni**:
  + Il metodo controllaCollisione utilizza **rettangoli di collisione** (Rectangle) per verificare se il proiettile interseca un nemico.
  + La dimensione dei rettangoli è specificata per il proiettile (10x10) e per il nemico (50x50).

**4. Dipendenze**

La classe dipende dalle seguenti librerie:

* **java.awt**: Per il disegno grafico e la gestione delle collisioni tramite Rectangle.
* **javax.imageio.ImageIO** (potenzialmente inutilizzata): Per il caricamento delle immagini.

**5. Esempio d'Uso**

java

Copia codice

// Creazione di un nuovo proiettile in posizione (100, 200), direzione destra

Proiettile proiettile = new Proiettile(100, 200, 1, 0);

// Movimento del proiettile

proiettile.muovi();

// Disegno del proiettile in un pannello grafico

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

proiettile.disegna(g);

}

// Controllo collisione con un nemico

Nemico nemico = new Nemico(150, 200);

if (proiettile.controllaCollisione(nemico)) {

proiettile.disattiva();

nemico.disattiva();

}

**6. Potenziali Estensioni**

* **Immagini del Proiettile**: Implementare l'uso di immagini grafiche personalizzate per il proiettile.
* **Velocità Variabile**: Aggiungere logica per proiettili con velocità diverse in base a potenziamenti.
* **Durata Limitata**: Implementare un limite di tempo o distanza per la durata del proiettile.
* **Animazioni**: Aggiungere animazioni o effetti visivi durante il movimento o alla collisione.

**7. Gestione Errori**

* La classe non carica immagini al momento, ma può essere estesa con un controllo simile alla classe Nemico per includere immagini personalizzate.

**Classe Pozione**

**Manuale Tecnico - Classe Pozione**

**1. Descrizione della Classe**

La classe **Pozione** appartiene al **Model** del progetto e rappresenta un **oggetto rigenerativo** che il giocatore può raccogliere per aumentare la propria vita. Ogni pozione ha:

* **Coordinate casuali** sulla mappa.
* **Valore rigenerativo** casuale che determina la quantità di vita ripristinata.
* Una dimensione variabile basata sul valore rigenerativo.
* Uno stato di attivazione per gestire la sua presenza nel gioco.

La pozione viene disegnata con un'immagine grafica caricata o, in alternativa, con un **fallback grafico** a forma di croce gialla.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | int | posizioneX | Coordinata X della pozione. |
| private | int | posizioneY | Coordinata Y della pozione. |
| private | int | valoreVita | Quantità di vita che la pozione rigenera. |
| private | int | dimensione | Dimensione grafica della pozione. |
| private | boolean | attivo | Stato della pozione: attiva (true) o raccolta. |
| private | Image | immagine | Immagine grafica della pozione. |
| private | Random | random | Oggetto per generare valori casuali. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| Pozione() | Inizializza la posizione, il valore rigenerativo, la dimensione e carica l'immagine. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| void disegna(Graphics g) | Disegna la pozione sulla finestra di gioco (immagine o fallback grafico). |
| void raccogli() | Disattiva la pozione, simulando la sua raccolta da parte del giocatore. |
| boolean isAttivo() | Restituisce lo stato della pozione: attiva o raccolta. |
| static int controlloDimensione(int valoreVita) | Determina la dimensione della pozione in base al valore rigenerativo. |

**Metodi Getter e Setter**

| **Metodo** | **Descrizione** |
| --- | --- |
| getPosizioneX() | Restituisce la coordinata X della pozione. |
| setPosizioneX(int x) | Imposta la coordinata X. |
| getPosizioneY() | Restituisce la coordinata Y della pozione. |
| setPosizioneY(int y) | Imposta la coordinata Y. |
| getValoreVita() | Restituisce il valore rigenerativo della pozione. |
| getDimensione() | Restituisce la dimensione grafica della pozione. |

**3. Dettagli Tecnici**

* **Generazione Casuale**:
  + La posizione della pozione (posizioneX, posizioneY) viene generata in modo **casuale** entro un'area definita:
    - X: da 0 a 850.
    - Y: da 0 a 650.
  + Il valore della vita è anch'esso generato casualmente tra **10 e 50**.
* **Dimensione Variabile**:
  + La dimensione della pozione dipende dal valore rigenerativo (valoreVita):
    - **10 px** per valori < 20.
    - **15 px** per valori tra 20 e 40.
    - **30 px** per valori > 40.
* **Disegno Grafico**:
  + Se l'immagine della pozione è caricata correttamente, viene ridimensionata e disegnata.
  + In caso di errore nel caricamento, viene disegnata una **croce gialla** come fallback.

**4. Dipendenze**

La classe dipende dalle seguenti librerie:

* **java.awt**: Per la gestione grafica e il disegno degli elementi.
* **javax.imageio.ImageIO**: Per il caricamento delle immagini.
* **java.util.Random**: Per generare posizioni e valori casuali.

**5. Esempio d'Uso**

java

Copia codice

// Creazione di una nuova pozione

Pozione pozione = new Pozione();

// Disegno della pozione in un pannello grafico

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

pozione.disegna(g);

}

// Verifica se la pozione è attiva e raccoglimento

if (pozione.isAttivo()) {

pozione.raccogli();

System.out.println("Pozione raccolta! Valore Vita: " + pozione.getValoreVita());

}

**6. Potenziali Estensioni**

* **Rimozione Temporizzata**: Implementare una durata limitata per la pozione.
* **Effetti Grafici**: Aggiungere animazioni visive quando la pozione viene raccolta.
* **Tipi Diversi di Pozioni**: Estendere la classe con sottoclassi per creare tipi di pozioni (es. pozioni di velocità, potenziamenti).

**7. Gestione Errori**

* Se l'immagine non viene caricata correttamente:
  + Stampa un messaggio di errore nella console.
  + Disegna un **fallback grafico** (croce gialla).

**8. Diagramma UML (Semplificato)**

sql

Copia codice

+-------------------------+

| Pozione |

+-------------------------+

| - posizioneX: int |

| - posizioneY: int |

| - valoreVita: int |

| - dimensione: int |

| - attivo: boolean |

| - immagine: Image |

| - random: Random |

+-------------------------+

| + Pozione() |

| + disegna(Graphics g) |

| + raccogli() |

| + isAttivo(): boolean |

| + getPosizioneX(): int |

| + getPosizioneY(): int |

| + getValoreVita(): int |

| + getDimensione(): int |

+-------------------------+

CONTROLLER

**Classe GameController**

**Manuale Tecnico - Classe GameController**

**1. Descrizione della Classe**

La classe **GameController** appartiene al **Controller** dell'architettura MVC e gestisce l'intera **logica del gioco**. Si occupa di:

* Gestire l'input dell'utente (movimento del personaggio e sparo dei proiettili).
* Aggiornare lo stato del gioco (movimenti, collisioni, spawn di nemici e pozioni).
* Coordinare il **Model** (personaggio, nemici, proiettili, pozioni) e la **View** (rendering grafico).
* Implementare meccaniche come **pausa**, **game over** e gestione del punteggio.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | JFrame | finestraGioco | Finestra principale del gioco. |
| private | GiocoPanel | giocoPanel | Pannello principale dove vengono disegnati gli oggetti. |
| private | HUD | hud | Pannello per visualizzare vita e punteggio. |
| private | Timer | timer | Loop di gioco basato su un timer Swing (≈ 60 FPS). |
| private | Personaggio | personaggio | Istanza del personaggio giocante. |
| private | ArrayList<Nemico> | nemici | Lista dei nemici presenti sullo schermo. |
| private | ArrayList<Proiettile> | proiettili | Lista dei proiettili sparati dal giocatore. |
| private | ArrayList<Pozione> | pozioni | Lista delle pozioni presenti sullo schermo. |
| private | int | punteggio | Punteggio del giocatore. |
| private | boolean | inPausa | Stato del gioco: pausa o attivo. |
| private | PausePanel | pausePanel | Pannello di pausa. |
| private | Variabili di controllo | muoviSu, ... | Variabili per il movimento continuo del personaggio. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| GameController() | Inizializza la finestra, le entità del gioco e avvia il **loop principale**. |

**Metodi Principali**

**Gestione Generale del Gioco**

| **Metodo** | **Descrizione** |
| --- | --- |
| private void inizializzaGioco() | Inizializza la finestra di gioco, le entità e i pannelli grafici. |
| private void avviaLoopDiGioco() | Avvia il loop principale del gioco utilizzando un **Timer** Swing (60 FPS). |
| private void aggiornaGioco() | Aggiorna la logica del gioco: movimento, collisioni, spawn nemici/pozioni. |
| private void incrementaPunteggio(int valore) | Incrementa il punteggio del giocatore. |

**Gestione Input Utente**

| **Metodo** | **Descrizione** |
| --- | --- |
| private void configuraMovimentiPersonaggio() | Configura l'input da tastiera e mouse per muovere il personaggio e sparare. |
| Listener **Tasti** | W, A, S, D per il movimento continuo. |
| Listener **Mouse** | Spara proiettili nella direzione del cursore al clic. |
| Listener **Barra Spaziatrice** | Spara proiettili verso destra. |

**Gestione Nemici e Pozioni**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| private void generaNuovoNemico() | Genera un nuovo nemico ai bordi della finestra in una posizione casuale. |
| private void generaNuovaPozione() | Genera una nuova pozione in una posizione casuale sulla mappa. |

**Collision Detection**

| **Metodo** | **Descrizione** |
| --- | --- |
| private boolean nemicoCollisione(Personaggio, Nemico) | Controlla la collisione tra il personaggio e un nemico. |
| private boolean pozioniCollisione(Personaggio, Pozione) | Controlla la collisione tra il personaggio e una pozione. |

**Gestione Stati del Gioco**

| **Metodo** | **Descrizione** |
| --- | --- |
| private void mettiInPausa() | Mette in pausa il gioco e mostra il pannello di pausa. |
| private void riprendiGioco() | Riprende il gioco dallo stato di pausa. |
| private void mostraGameOverPanel() | Termina il gioco e mostra il pannello di **Game Over**. |

**3. Dettagli Tecnici**

* **Loop di Gioco**:
  + Implementato con un **Timer Swing** che aggiorna lo stato del gioco a circa **60 FPS** (16 ms per frame).
* **Gestione Input**:
  + Usa **InputMap** e **ActionMap** per gestire l'input da tastiera (movimento) e mouse (sparo).
  + Tasti:
    - **W, A, S, D** per il movimento.
    - **Barra Spaziatrice** per sparare.
    - **ESC** per mettere in pausa.
* **Spawn di Nemici e Pozioni**:
  + Nemici vengono generati ai bordi della finestra a intervalli regolari.
  + Pozioni vengono generate in posizioni casuali ogni **15 secondi**.
* **Collision Detection**:
  + Basata su **rettangoli di collisione** (hitbox) calcolati dalle coordinate e dimensioni degli oggetti.
* **Gestione Stati**:
  + **Pausa**: Ferma il timer e mostra un pannello interattivo.
  + **Game Over**: Termina il gioco, mostra il punteggio finale e fornisce opzioni per **riprovare** o tornare al **menu principale**.

**4. Dipendenze**

La classe dipende da:

* **Model**: Personaggio, Proiettile, Nemico, Pozione.
* **View**: GiocoPanel, HUD, PausePanel, GameOverPanel, MenuPrincipale.
* **Swing**: Per la gestione dell'interfaccia grafica (JFrame, Timer, InputMap).

**5. Esempio d'Uso**

java

Copia codice

public class Main {

public static void main(String[] args) {

new GameController(); // Avvia il gioco

}

}

**6. Potenziali Estensioni**

* **Aggiunta di livelli**: Spawn dinamico di nemici e potenziamenti con difficoltà crescente.
* **Aggiunta di power-up**: Proiettili multipli, velocità potenziata, scudi protettivi.
* **Salvataggio del punteggio**: Memorizzare il punteggio più alto (High Score).
* **Effetti audio**: Integrare suoni per sparo, collisioni e game over.

VIEW

**Classe ControlliPanel**

**Manuale Tecnico - Classe ControlliPanel**

**1. Descrizione della Classe**

**La classe ControlliPanel appartiene al View del progetto e fornisce un'interfaccia grafica che mostra all'utente i controlli del gioco. Questa schermata include:**

* **Un titolo chiaro e centrato.**
* **Una lista dettagliata dei controlli del gioco.**
* **Un bottone interattivo per tornare al menu principale.**

**La classe utilizza componenti di Swing per il rendering grafico, come JLabel, JTextArea, e JButton.**

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| **private** | **JLabel** | **titolo** | **Titolo del pannello.** |
| **private** | **JTextArea** | **controlli** | **Testo che elenca i controlli del gioco.** |
| **private** | **JButton** | **btnTornaIndietro** | **Bottone per tornare al menu principale.** |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| **ControlliPanel(ActionListener tornaIndietroListener)** | **Inizializza e configura i componenti grafici della schermata dei controlli.** |

**Metodi Principali**

| **Metodo** | **Descrizione** |
| --- | --- |
| **N/A** | **La classe è auto-contenuta nel costruttore e non include metodi pubblici.** |

**3. Dettagli Tecnici**

* **Layout:**
  + **Utilizza un BorderLayout per organizzare il pannello in tre sezioni:**
    - **Nord: Titolo con font grande e bordi per il margine superiore e inferiore.**
    - **Centro: Testo descrittivo dei controlli centrato usando un GridBagLayout.**
    - **Sud: Bottone "Torna al Menu" centrato all'interno di un pannello trasparente.**
* **Aspetto Grafico:**
  + **Colori:**
    - **Sfondo: Grigio scuro (45, 45, 45).**
    - **Testo e titolo: Giallo per risaltare sullo sfondo.**
  + **Font:**
    - **Utilizza Garamond per un aspetto elegante.**
    - **Dimensioni:**
      * **Titolo: 36px, grassetto.**
      * **Testo dei controlli: 24px, grassetto.**
  + **Trasparenza:**
    - **Il testo e i pannelli centrali sono resi trasparenti usando setOpaque(false).**
* **Interattività:**
  + **Il bottone "Torna al Menu":**
    - **Esegue un'azione definita dall'ActionListener passato al costruttore.**
    - **Configurato con uno sfondo grigio e testo bianco.**

**4. Dipendenze**

**La classe dipende dalle seguenti librerie:**

* **javax.swing: Per i componenti grafici come JPanel, JLabel, JTextArea, e JButton.**
* **java.awt: Per layout e configurazioni grafiche.**
* **java.awt.event.ActionListener: Per gestire l'input utente tramite il bottone.**

**5. Esempio d'Uso**

**java**

**Copia codice**

**// Creazione del pannello dei controlli con ritorno al menu principale**

**ControlliPanel controlliPanel = new ControlliPanel(e -> {**

**// Logica per tornare al menu principale**

**System.out.println("Torno al menu principale!");**

**});**

**// Aggiunta del pannello a una finestra principale**

**JFrame finestra = new JFrame("Controlli");**

**finestra.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**finestra.setSize(800, 600);**

**finestra.add(controlliPanel);**

**finestra.setVisible(true);**

**6. Potenziali Estensioni**

* **Aggiunta di immagini esplicative: Inserire icone o schermate che mostrano i controlli visivamente.**
* **Scorrimento: Implementare uno JScrollPane per gestire liste di controlli più lunghe.**
* **Transizioni Animate: Aggiungere effetti di transizione per rendere l'interfaccia più dinamica.**

**7. Diagramma UML (Semplificato)**

**diff**

**Copia codice**

**+--------------------------+**

**| ControlliPanel |**

**+--------------------------+**

**| - titolo: JLabel |**

**| - controlli: JTextArea |**

**| - btnTornaIndietro: JButton |**

**+--------------------------+**

**| + ControlliPanel(ActionListener) |**

**+--------------------------+**

**Classe GameOverPanel**

**Manuale Tecnico - Classe GameOverPanel**

**1. Descrizione della Classe**

La classe **GameOverPanel** appartiene al **View** del progetto e rappresenta la schermata di **Game Over**. Essa fornisce:

* Un'immagine di **sfondo**.
* Un **titolo centrale** "Game Over".
* Visualizzazione del **punteggio finale**.
* Due pulsanti interattivi:
  + **"Gioca Ancora"** per riavviare la partita.
  + **"Menu Principale"** per tornare al menu principale.
* Un **messaggio finale** di ringraziamento.

La classe utilizza **Swing** per il rendering grafico e implementa una logica semplice per mostrare componenti personalizzati con layout eleganti.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | Image | sfondo | Immagine di sfondo della schermata di Game Over. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| GameOverPanel(int punteggio, ActionListener giocaAncora, ActionListener tornaAlMenu) | Inizializza i componenti della schermata di Game Over e imposta le azioni dei pulsanti. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| protected void paintComponent(Graphics g) | Disegna l'immagine di sfondo adattata alle dimensioni del pannello. |

**3. Dettagli Tecnici**

* **Layout**:
  + Utilizza un **BorderLayout** per suddividere il pannello in:
    - **Nord**: Titolo "Game Over".
    - **Centro**: Pannello contenente il punteggio finale e i pulsanti.
    - **Sud**: Messaggio finale "Grazie per aver giocato!".
* **Sfondo Personalizzato**:
  + Carica un'immagine di sfondo utilizzando **ImageIO**.
  + L'immagine viene adattata automaticamente alle dimensioni del pannello con il metodo **paintComponent**.
* **Componenti**:
  + **Titolo**:
    - Font: Garamond, **Bold**, dimensione **100px**.
    - Colore: **Giallo**.
  + **Punteggio Finale**:
    - Visualizza il punteggio passato come parametro.
    - Font: Garamond, **Plain**, dimensione **30px**.
    - Colore: **Bianco**.
  + **Pulsanti**:
    - **"Gioca Ancora"** e **"Menu Principale"**.
    - Font: Garamond, **Bold**, dimensione **28px**.
    - Dimensione: **300x70px**.
    - Sfondo: **Grigio** con testo **Giallo**.
  + **Footer**:
    - Messaggio finale con font **Italic** e dimensione **20px**.
* **Interattività**:
  + I pulsanti ricevono degli ActionListener passati come parametri al costruttore, permettendo di:
    - **Riavviare il gioco**.
    - **Tornare al menu principale**.

**4. Dipendenze**

La classe dipende dalle seguenti librerie:

* **javax.swing**: Per componenti grafici come JPanel, JLabel, JButton.
* **java.awt**: Per layout, disegno grafico e gestione dei colori.
* **javax.imageio.ImageIO**: Per caricare l'immagine di sfondo.

**5. Esempio d'Uso**

java

Copia codice

// Creazione del pannello Game Over con punteggio e azioni

GameOverPanel gameOverPanel = new GameOverPanel(

12345, // Punteggio finale

e -> { System.out.println("Riavvio il gioco!"); }, // Azione per "Gioca Ancora"

e -> { System.out.println("Torno al menu principale!"); } // Azione per "Menu Principale"

);

// Aggiunta del pannello a una finestra principale

JFrame finestra = new JFrame("Game Over");

finestra.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

finestra.setSize(1280, 720);

finestra.add(gameOverPanel);

finestra.setVisible(true);

**6. Potenziali Estensioni**

* **Effetti di Transizione**: Aggiungere animazioni al caricamento del pannello (es. dissolvenza).
* **Musica di sottofondo**: Integrare una musica di Game Over usando Clip di Java Sound.
* **Pulsante "Esci dal Gioco"**: Aggiungere un'opzione per chiudere completamente l'applicazione.

**7. Gestione Errori**

* **Caricamento Immagine**:
  + Se l'immagine di sfondo non viene trovata, viene stampato un messaggio di errore e il pannello continua a funzionare con un eventuale sfondo di default (es. colore scuro).

**8. Diagramma UML (Semplificato)**

diff

Copia codice

+--------------------------+

| GameOverPanel |

+--------------------------+

| - sfondo: Image |

+--------------------------+

| + GameOverPanel(...) |

| + paintComponent(Graphics g) |

+--------------------------+

**Classe GiocoPanel**

**1. Descrizione della Classe**

La classe **GiocoPanel** appartiene al **View** del progetto e rappresenta il **pannello principale** dove viene disegnato l'intero stato del gioco. Essa include:

* Il **personaggio** controllato dal giocatore.
* I **nemici** attivi sullo schermo.
* I **proiettili** sparati dal giocatore.
* Le **pozioni** presenti nella mappa.
* Uno **sfondo grafico** che copre l'intera finestra.

La classe utilizza il metodo paintComponent di **Swing** per aggiornare e disegnare in modo continuo tutti gli elementi grafici sulla mappa.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | Personaggio | personaggio | Riferimento al personaggio principale. |
| private | ArrayList<Nemico> | nemici | Lista dei nemici attualmente attivi. |
| private | ArrayList<Proiettile> | proiettili | Lista dei proiettili sparati dal personaggio. |
| private | ArrayList<Pozione> | pozioni | Lista delle pozioni visibili sulla mappa. |
| private | Image | sfondoGioco | Immagine di sfondo per il pannello di gioco. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| GiocoPanel(Personaggio personaggio, ArrayList<Nemico> nemici, ArrayList<Proiettile> proiettili, ArrayList<Pozione> pozioni) | Inizializza il pannello con il personaggio, nemici, proiettili e pozioni. |

**Metodi Principali**

**Disegno Grafico**

| **Metodo** | **Descrizione** |
| --- | --- |
| protected void paintComponent(Graphics g) | Ridisegna lo stato del gioco: sfondo, personaggio, nemici, proiettili e pozioni. |

**3. Dettagli Tecnici**

* **Sfondo Grafico**:
  + Lo sfondo viene caricato utilizzando **ImageIO**.
  + Se lo sfondo è presente, viene ridimensionato per adattarsi automaticamente alla dimensione del pannello.
* **Disegno degli Elementi**:
  + **Personaggio**:
    - Viene disegnato utilizzando il metodo **disegna** della classe Personaggio.
  + **Nemici**:
    - Solo i nemici con stato **attivo** vengono disegnati.
  + **Proiettili**:
    - I proiettili attivi vengono disegnati uno ad uno.
  + **Pozioni**:
    - Le pozioni attive vengono disegnate sullo schermo.
* **Ottimizzazione**:
  + Il metodo paintComponent chiama super.paintComponent(g) per pulire l'area di disegno prima di ridisegnare i nuovi elementi.
  + Ogni oggetto viene controllato tramite il metodo isAttivo per disegnare solo quelli che sono visibili.

**4. Dipendenze**

La classe dipende dalle seguenti librerie e classi:

* **javax.swing**: Per l'implementazione del pannello grafico.
* **java.awt**: Per il disegno grafico tramite Graphics e gestione delle immagini.
* **javax.imageio.ImageIO**: Per il caricamento dell'immagine di sfondo.
* **Model**: Personaggio, Nemico, Proiettile, Pozione.

**5. Esempio d'Uso**

java

Copia codice

// Creazione delle entità di gioco

Personaggio player = new Personaggio(100, 100);

ArrayList<Nemico> nemici = new ArrayList<>();

ArrayList<Proiettile> proiettili = new ArrayList<>();

ArrayList<Pozione> pozioni = new ArrayList<>();

// Creazione del pannello di gioco

GiocoPanel giocoPanel = new GiocoPanel(player, nemici, proiettili, pozioni);

// Aggiunta del pannello a una finestra principale

JFrame finestra = new JFrame("Survival Game");

finestra.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

finestra.setSize(1280, 720);

finestra.add(giocoPanel);

finestra.setVisible(true);

**6. Potenziali Estensioni**

* **Effetti Grafici**:
  + Aggiungere animazioni per i proiettili o nemici eliminati.
* **Livelli Multipli**:
  + Caricare sfondi diversi in base al livello attuale.
* **FPS Counter**:
  + Mostrare il numero di frame al secondo per monitorare le prestazioni.
* **Effetti di Overlay**:
  + Visualizzare effetti grafici come esplosioni o danni.

**7. Gestione Errori**

* **Caricamento Immagine di Sfondo**:
  + Se il caricamento fallisce, viene stampato un messaggio di errore in console e il gioco continua senza lo sfondo.

**8. Diagramma UML (Semplificato)**

yaml

Copia codice

+--------------------------+

| GiocoPanel |

+--------------------------+

| - personaggio: Personaggio |

| - nemici: ArrayList<Nemico>|

| - proiettili: ArrayList<Proiettile> |

| - pozioni: ArrayList<Pozione> |

| - sfondoGioco: Image |

+--------------------------+

| + GiocoPanel(...) |

| + paintComponent(Graphics g) |

+--------------------------+

**Classe HUD**

**1. Descrizione della Classe**

La classe **HUD** appartiene al **View** del progetto e rappresenta l'**interfaccia utente** che visualizza le informazioni vitali del gioco:

* **Barra della vita** del personaggio.
* **Punteggio attuale** del giocatore.

L'HUD viene aggiornato dinamicamente per riflettere lo stato corrente del personaggio e del punteggio.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | Personaggio | personaggio | Riferimento all'istanza del personaggio. |
| private | JLabel | lblVita | Etichetta per visualizzare i punti vita (HP). |
| private | JProgressBar | barraVita | Barra progressiva per rappresentare la vita. |
| private | JLabel | lblScore | Etichetta per visualizzare il punteggio attuale. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| HUD(Personaggio personaggio, int punteggio) | Inizializza il pannello HUD con vita e punteggio del personaggio. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| public void aggiornaHUD(int punteggio) | Aggiorna il valore della barra vita e del punteggio in tempo reale. |

**3. Dettagli Tecnici**

* **Componenti Grafici**:
  1. **Barra della Vita**:
     + Realizzata con JProgressBar per visualizzare il valore corrente della vita.
     + Configurata con:
       - **Range**: da 0 a vitaMassima.
       - **Valore**: impostato dinamicamente con setValue.
       - **Colori**:
         * **Foreground**: Giallo (barra progressiva).
         * **Background**: Nero.
  2. **Etichette**:
     + **lblVita**:
       - Mostra il testo: HP: X/Y (vita corrente e massima).
       - Font: Garamond, **Bold**, 20px.
       - Colore: **Bianco**.
     + **lblScore**:
       - Mostra il testo: Score: X (punteggio attuale).
       - Font: Garamond, **Bold**, 20px.
       - Colore: **Bianco**.
* **Layout**:
  1. Utilizza un **FlowLayout** con allineamento a sinistra e spaziatura personalizzata.
  2. Il colore di sfondo è **nero trasparente** (0, 0, 0, 128) per dare un effetto moderno e visibile senza coprire troppo il gameplay.
* **Aggiornamento dell'HUD**:
  1. Il metodo **aggiornaHUD** viene chiamato nel loop di gioco per aggiornare:
     + **Etichetta della vita** (lblVita).
     + **Barra della vita** (barraVita).
     + **Punteggio** (lblScore).

**4. Dipendenze**

La classe dipende dalle seguenti librerie e classi:

* **javax.swing**: Per i componenti grafici come JPanel, JLabel e JProgressBar.
* **java.awt**: Per layout e gestione dei colori.
* **Model**: Personaggio.

**5. Esempio d'Uso**

java

Copia codice

// Creazione del personaggio

Personaggio player = new Personaggio();

// Inizializzazione dell'HUD con punteggio iniziale

HUD hud = new HUD(player, 0);

// Aggiornamento dell'HUD in base al gioco

hud.aggiornaHUD(100); // Aggiorna il punteggio e la vita attuale

// Aggiunta dell'HUD a una finestra principale

JFrame finestra = new JFrame("Survival Game HUD");

finestra.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

finestra.setSize(1280, 720);

finestra.add(hud, BorderLayout.NORTH);

finestra.setVisible(true);

**6. Potenziali Estensioni**

* **Colori Dinamici della Barra Vita**:
  + Cambiare il colore della barra vita in base al valore corrente (es. rosso per valori critici).
* **Timer del Gioco**:
  + Aggiungere un'etichetta per visualizzare un timer o il tempo rimanente.
* **Pannello Informazioni Extra**:
  + Mostrare ulteriori dettagli come armi, power-up o livello corrente.

**7. Diagramma UML (Semplificato)**

sql

Copia codice

+--------------------------+

| HUD |

+--------------------------+

| - personaggio: Personaggio |

| - lblVita: JLabel |

| - barraVita: JProgressBar|

| - lblScore: JLabel |

+--------------------------+

| + HUD(Personaggio, int) |

| + aggiornaHUD(int) |

+--------------------------+

**Classe MenuPrincipale**

**Manuale Tecnico - Classe MenuPrincipale**

**1. Descrizione della Classe**

La classe **MenuPrincipale** appartiene al **View** del progetto e rappresenta la **schermata principale** del gioco. Contiene:

* Un'immagine di **sfondo** per il menu.
* Il **titolo del gioco** ben visibile.
* **Tre pulsanti** interattivi:
  + **Gioca**: Avvia il gioco principale.
  + **Controlli**: Mostra una finestra con i comandi di gioco.
  + **Esci**: Chiude l'applicazione.
* Un **footer informativo** con il nome dello sviluppatore e un collegamento al profilo LinkedIn.

La classe utilizza componenti **Swing** con layout ben organizzati per un'interfaccia chiara e intuitiva.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | Image | sfondo | Immagine di sfondo per il menu principale. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| MenuPrincipale() | Configura la finestra principale, carica lo sfondo e inizializza i componenti del menu. |

**Metodi Principali**

| **Firma del Metodo** | **Descrizione** |
| --- | --- |
| private JButton creaBottone(...) | Crea un pulsante personalizzato con testo, dimensioni e azione specifica. |
| private void mostraControlli() | Mostra una nuova finestra con il pannello dei controlli del gioco. |
| protected void paintComponent(Graphics g) | Disegna l'immagine di sfondo ridimensionata (tramite pannello interno). |

**3. Dettagli Tecnici**

* **Sfondo Grafico**:
  + Caricato con **ImageIO**.
  + Viene disegnato tramite un pannello personalizzato che sovrascrive il metodo paintComponent.
* **Componenti**:
  + **Titolo**:
    - Testo: **"Survival Game"**.
    - Font: Garamond, **Bold**, 72px.
    - Colore: **Giallo**.
  + **Pulsanti**:
    - **"Gioca"**: Avvia il gioco tramite GameController.
    - **"Controlli"**: Mostra una finestra con il **ControlliPanel**.
    - **"Esci"**: Chiude l'applicazione con System.exit(0).
    - Layout: **GridBagLayout** per centrare i pulsanti.
    - Dimensioni: **300x80px** con colore di sfondo **grigio scuro** e testo **giallo**.
  + **Footer**:
    - Testo: "Gioco realizzato da Stefano Bruni | linkedin.com/in/stefano-bruni93".
    - Font: Garamond, **Italic**, 18px.
    - Colore: **Giallo**.
* **Interattività**:
  + Il pulsante **"Gioca"** avvia il gioco chiudendo il menu e creando un'istanza di GameController.
  + Il pulsante **"Controlli"** apre una nuova finestra che mostra il pannello ControlliPanel.
  + Il pulsante **"Esci"** termina il programma.

**4. Dipendenze**

La classe dipende dalle seguenti librerie e classi:

* **javax.swing**: Per la creazione della finestra e dei componenti grafici.
* **java.awt**: Per layout e gestione dei colori.
* **javax.imageio.ImageIO**: Per il caricamento delle immagini.
* **Controller**: GameController.
* **View**: ControlliPanel.

**5. Esempio d'Uso**

java

Copia codice

public class Main {

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new MenuPrincipale());

}

}

**6. Potenziali Estensioni**

* **Effetti di Transizione**:
  + Aggiungere animazioni tra il menu e il gioco.
* **Musica di sottofondo**:
  + Integrare una musica per il menu principale con la libreria **Java Sound API**.
* **Immagine Alternativa**:
  + Caricare uno sfondo diverso in base alla difficoltà scelta.
* **Pulsanti per le Opzioni**:
  + Aggiungere un pulsante per configurare impostazioni come volume e controlli.

**7. Gestione Errori**

* **Caricamento Immagine**:
  + Se l'immagine di sfondo non viene trovata, stampa un messaggio di errore in console.

**8. Diagramma UML (Semplificato)**

diff

Copia codice

+--------------------------+

| MenuPrincipale |

+--------------------------+

| - sfondo: Image |

+--------------------------+

| + MenuPrincipale() |

| - creaBottone(...) |

| - mostraControlli() |

+--------------------------+

**Classe PausePanel**

**Manuale Tecnico - Classe PausePanel**

**1. Descrizione della Classe**

La classe **PausePanel** appartiene al **View** del progetto e fornisce un **pannello di pausa** semi-trasparente che viene visualizzato quando il gioco viene messo in pausa. Il pannello mostra:

* Un **titolo**: "Gioco in Pausa".
* Il **punteggio corrente** del giocatore.
* Due **pulsanti interattivi**:
  + **Riprendi**: Riprende il gioco.
  + **Torna al Menu Principale**: Termina il gioco attuale e torna al menu principale.

La classe utilizza **Swing** per la creazione dei componenti grafici e il **GridBagLayout** per organizzare i componenti in modo centrato.

**2. Struttura della Classe**

**Attributi**

| **Modificatore** | **Tipo** | **Nome** | **Descrizione** |
| --- | --- | --- | --- |
| private | JLabel | lblPausa | Etichetta del titolo "Gioco in Pausa". |
| private | JLabel | lblPunteggio | Etichetta per mostrare il punteggio corrente. |
| private | JButton | btnRiprendi | Pulsante per riprendere il gioco. |
| private | JButton | btnTornaMenu | Pulsante per tornare al menu principale. |

**Costruttori**

| **Metodo** | **Descrizione** |
| --- | --- |
| PausePanel(int punteggio, ActionListener riprendiListener, ActionListener tornaAlMenu) | Inizializza il pannello con titolo, punteggio e pulsanti interattivi. |

**3. Dettagli Tecnici**

* **Aspetto Grafico**:
  + **Sfondo**: Semi-trasparente (nero con opacità 200).
  + **Testo e Font**:
    - Titolo: Garamond, **Bold**, dimensione **36px**, colore **giallo**.
    - Punteggio: Garamond, **Plain**, dimensione **24px**, colore **bianco**.
    - Pulsanti: Font **Garamond**, dimensione **20px**.
  + **Pulsanti**:
    - Sfondo: Grigio scuro tendente al nero (45, 45, 45).
    - Testo: Giallo.
    - Dimensione uniforme: **300x50px**.
* **Layout**:
  + Utilizza un **GridBagLayout** per centrare tutti i componenti verticalmente e orizzontalmente.
  + GridBagConstraints è configurato per mantenere la spaziatura e allineare gli elementi.
* **Funzionalità Interattive**:
  + **Pulsante Riprendi**:
    - Esegue un'azione personalizzata passata tramite ActionListener (es. riprendere il timer di gioco).
  + **Pulsante Torna al Menu Principale**:
    - Chiude la finestra corrente usando SwingUtilities.getWindowAncestor(this).dispose() e avvia una nuova istanza di MenuPrincipale.

**4. Dipendenze**

La classe dipende dalle seguenti librerie e classi:

* **javax.swing**: Per i componenti grafici come JPanel, JLabel, JButton.
* **java.awt**: Per layout, gestione dei colori e componenti grafici.
* **View**: MenuPrincipale (per tornare al menu principale).

**5. Esempio d'Uso**

java

Copia codice

// Creazione del pannello pausa

PausePanel pausePanel = new PausePanel(

1500, // Punteggio corrente

e -> System.out.println("Gioco ripreso!"), // Azione per "Riprendi"

e -> System.out.println("Torno al menu principale.") // Azione per "Menu Principale"

);

// Aggiunta del pannello a una finestra di test

JFrame finestra = new JFrame("Pausa");

finestra.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

finestra.setSize(1280, 720);

finestra.add(pausePanel);

finestra.setVisible(true);

**6. Potenziali Estensioni**

* **Transizione Animata**:
  + Aggiungere effetti di dissolvenza o animazioni quando il pannello viene visualizzato.
* **Opzioni di Gioco**:
  + Integrare pulsanti per modificare impostazioni come volume o difficoltà.
* **Timer**:
  + Mostrare un timer di pausa per informare il giocatore del tempo passato in pausa.

**7. Diagramma UML (Semplificato)**

diff

Copia codice

+--------------------------+

| PausePanel |

+--------------------------+

| - lblPausa: JLabel |

| - lblPunteggio: JLabel |

| - btnRiprendi: JButton |

| - btnTornaMenu: JButton |

+--------------------------+

| + PausePanel(...) |

+--------------------------+

**8. Gestione Errori**

* La classe non presenta criticità specifiche. Se vengono passati null come ActionListener, il comportamento dei pulsanti sarà inattivo.

MAIN

**Classe Main**

**Manuale Tecnico - Classe Main**

**1. Descrizione della Classe**

La classe **Main** rappresenta il **punto di ingresso principale** del gioco. La sua responsabilità è avviare l'applicazione creando e visualizzando il **menu principale** utilizzando il thread dedicato all'interfaccia grafica (**Swing Event Dispatch Thread**).

**2. Struttura della Classe**

**Metodi**

| **Modificatore** | **Tipo di Ritorno** | **Nome** | **Parametri** | **Descrizione** |
| --- | --- | --- | --- | --- |
| public | void | main | String[] args | Metodo principale che avvia l'applicazione. |

**3. Dettagli Tecnici**

* **Esecuzione Sicura con Swing**:
  + Il metodo **SwingUtilities.invokeLater** viene utilizzato per assicurarsi che la creazione e visualizzazione dell'interfaccia grafica avvenga sul **Swing Event Dispatch Thread**. Questo è il **modo corretto e sicuro** per avviare componenti Swing.
* **Avvio del Menu Principale**:
  + Crea un'istanza della classe **MenuPrincipale** che rappresenta la **schermata iniziale** del gioco.
  + Il metodo setVisible(true) rende visibile il menu all'utente.

**4. Dipendenze**

La classe dipende da:

* **View**:
  + **MenuPrincipale**: Punto di ingresso per il gioco (menu principale).
* **javax.swing.SwingUtilities**: Utilizzato per eseguire l'interfaccia utente in modo sicuro sul **EDT (Event Dispatch Thread)**.

**5. Esempio d'Uso**

La classe Main non necessita di input esterni e viene eseguita direttamente:

bash

Copia codice

javac main/Main.java

java main.Main

**6. Diagramma UML (Semplificato)**

css

Copia codice

+--------------------------+

| Main |

+--------------------------+

| + main(String[] args) |

+--------------------------+

**7. Dettagli Aggiuntivi**

* **Importanza di invokeLater**:
  + SwingUtilities.invokeLater garantisce che il codice UI venga eseguito nel thread corretto, prevenendo possibili problemi di **concorrenza** e **interruzione** dell'interfaccia grafica.
* **Scalabilità**:
  + La classe è facilmente espandibile per includere altre operazioni di inizializzazione, come il caricamento delle risorse o configurazioni di gioco.

**8. Conclusione**

La classe **Main** serve come entry-point minimale ma efficiente, avviando il gioco tramite il **MenuPrincipale**. Questo approccio segue le migliori pratiche di sviluppo con **Swing**.