Relazione JBubbleBobble

numero di matricola: 2104150

corso: presenza MZ

nome e cognome: Shuang Dai

composizione del gruppo: Team di 1 persona

decisioni di progettazione

- gestione del profilo utente, nickname, avatar:

  la classe JBubbleBobble dove contiene il main avvia il gioco e aggiunge un'interfaccia grafica per configurare il giocatore.

  1. Caricamento Avatar:

     Un selettore di file consente al giocatore di scegliere un'immagine come avatar dal proprio sistema. L'immagine selezionata viene ridimensionata e visualizzata in un'etichetta (JLabel). L'avatar viene salvato localmente in una directory chiamata avatars per usi futuri.

  2. Inserimento Nome Giocatore:

     Un campo di testo (JTextField) consente al giocatore di inserire il proprio nome, che viene salvato nel componente di livello (LevelComponent). Pannello Personalizzato: Un pannello (JPanel) organizza i componenti per l'inserimento del nome e il caricamento dell'avatar.

- Inizializzazione del Gioco

  vengono utilizzate tre classi principali:

  1. LevelComponent: gestisce lo stato e la logica di base del livello di gioco.
  2. GameView: si occupa della rappresentazione grafica del gioco.
  3. GameController: attraverso il metodo startGame, viene avviato il ciclo principale del gioco.

- Gestione dei livelli, shermata hai vinto, game over

  la classe Level rappresenta un singolo livello di un gioco, con molte funzionalità per gestire oggetti che interagiscono tra loro (come mostri, frutti, piattaforme, bolle) e per gestire lo stato del gioco. Gestisce tutto ciò che riguarda un livello nel gioco, come l'aggiornamento degli oggetti collidibili, la gestione delle collisioni, l'aggiunta/rimozione di oggetti, e la gestione dello stato del gioco (ad esempio, se il gioco è finito o se è pronto per passare al livello successivo, pausa e continua). Inoltre, offre metodi per interagire con gli oggetti e mantiene traccia delle azioni del giocatore e degli eventi nel gioco.

  e nella classe LevelComponente vengono gestiti le seguenti funzioni:

1. Transizioni di Livello Il metodo changeLevel permette di cambiare livello sulla base di comandi (incremento, decremento, menu principale, game over). Il metodo loadLevel costruisce dinamicamente i livelli leggendo file di configurazione.
2. Input da Tastiera Tiene traccia dello stato dei tasti (sinistra, destra, su) per muovere l'eroe o attivare azioni come sparare bolle.

- classifica e gestione del punteggio (uso di stream)

Il ranking viene gestito sempre da LevelComponente all'interno del metodo paintComponent:

1. Archiviazione della classifica:

La classifica dei punteggi viene memorizzata in una lista di mappe chiamata scoreboard. Ogni mappa contiene informazioni su un giocatore, inclusi il nome del giocatore e il punteggio. La lista scoreboard viene aggiornata durante il gioco, e quando si arriva al livello 10 (il livello del "game over") o al livello 11 (schermata di "game won"), la lista viene ordinata in ordine decrescente in base al punteggio.

2. Visualizzazione della classifica:

Quando viene raggiunto il livello 10 o 11, il codice crea una schermata con la classifica. Il codice ordina la lista scoreboard in base al punteggio dei giocatori, selezionando solo i primi 5 punteggi (con un limite di 5 giocatori per la visualizzazione). Ogni voce nella classifica include il ranking del giocatore, l'avatar (se esiste), il nome del giocatore e il punteggio.

3. Ordinamento dei punteggi:

Viene utilizzato lo stream di Java per ordinare la lista scoreboard in ordine decrescente in base al punteggio del giocatore. Il codice utilizza il metodo sorted per fare questo ordinamento e quindi raccoglie i primi 5 punteggi con limit(5).

```
scoreboard = scoreboard.stream()
                .distinct()
                .sorted((o1, o2) -> {
                    return Integer.compare((Integer) o2.get("score"),
                            (Integer) o1.get("score"));
                })
                .limit(5)
                .collect(Collectors.toList());
```

4. Visualizzazione grafica:

La classifica viene disegnata sopra un'area rettangolare che mostra i dettagli di ogni giocatore. Per ogni riga della classifica, viene disegnato:

- Il ranking del giocatore.
- L'avatar del giocatore (se disponibile).
- Il nome del giocatore.
- Il punteggio del giocatore.

- eroe, mostri, power up

  ogni oggetti collidibili vengono definiti da una classe astratta Collidable come la base. Gestisce variabili come la posizione, le dimensioni, la velocità e lo stato dell'oggetto (ad esempio, se sta saltando o cadendo). Contiene metodi per disegnare l'oggetto sullo schermo, rilevare collisioni con piattaforme, aggiornare la posizione e gestire la morte dell'oggetto (rimuovendolo dal livello). Alcuni comportamenti specifici, come il movimento, sono definiti nelle classi derivate che estendono Collidable.

  - ereo:

    La classe Hero gestisce la logica dell'eroe nel gioco, comprese le collisioni con piattaforme e nemici, il movimento, il tiro di bolle e l'invincibilità temporanea. Include anche la gestione delle vite e il respawn dell'eroe quando viene sconfitto.

  - mostri

    La classe Monster definisce il comportamento e le interazioni di un mostro nel gioco. Si occupa del movimento, del salto, delle collisioni con le piattaforme, dell'interazione con le bolle e le FireBubbles, e del cambiamento di direzione in modo casuale. Inoltre, implementa la gestione della "bubbled state", che permette al mostro di essere intrappolato in una bolla e di salire verso l'alto per un certo periodo.

    Ci sono tre tipi di mostri:

    1. BubbleBuster: La classe BubbleBuster è una sottoclasse della classe Monster ed è uno dei mostri più basilari nel gioco. Ha una funzionalità simile a quella dei mostri generali definiti nella classe Monster
    2. Incendo: La classe Incendo estende la classe Monster e aggiunge funzionalità per il lancio periodico di palle di fuoco. Oltre alle abilità di base di un mostro, Incendo può sparare palle di fuoco in direzione corrente a intervalli casuali compresi tra 2 e 4 secondi.
    3. Giant: è un mostro che ha un comportamento imprevedibile grazie al lancio di palle di fuoco a intervalli casuali.

  - power up:

    - nella classe Fruit ogni tipo di frutta ha un valore di punteggio diverso e può apparire sotto determinate condizioni. I valori sono 10, 500 , 3000 o un valore speciale per il personaggio Rascal.
    - Apple: aggiunge una vita
    - Banana: aggiunge 300 punti
    - FireIcon: cambia bolle a fireball

  - bolle: Le bolle vengono definite da una classe astratta Projectile, e ci sono due tipi: le bolle normali e fireball

  - adozione di Java Swing [2] per la GUI

    Per il design di UI ho utilizzato principalmenteJava Swing [2].

Design Pattern

- Observer Observable

  La classe LevelComponent implementa il pattern Observer Observable per gestire la comunicazione tra il livello del gioco e gli altri componenti.

  funziona in modo seguente:

  - Osservatori (Observer): La classe LevelComponent tiene traccia di una lista di oggetti che implementano l'interfaccia GameObserver. Questi osservatori sono aggiunti alla lista tramite il metodo addObserver(GameObserver observer) e possono essere rimossi con removeObserver(GameObserver observer).

    ```
    // GameObserver
    private List<GameObserver> observers = new ArrayList<>();

    // add observer
    public void addObserver(GameObserver observer) {
        observers.add(observer);
    }
    // remove observer
    public void removeObserver(GameObserver observer) {
        observers.remove(observer);
    }
    ```

  - Notifica (notifyObservers): Ogni volta che lo stato del gioco cambia in modo significativo (ad esempio, quando il livello del gioco cambia o viene raggiunta una condizione di vittoria/perdita), il metodo notifyObservers() viene invocato. Questo metodo attraversa la lista degli osservatori e li aggiorna chiamando il metodo updateGameState() su ciascuno di essi. Gli osservatori possono così reagire al cambiamento dello stato (ad esempio, aggiornando l'interfaccia utente, gestendo la logica del gioco, ecc.).

    ```
    //notifyObservers
    private void notifyObservers() {
        for (GameObserver observer : observers) {
            observer.updateGameState();
        }
    }
    ```

  - Cambiamenti di stato: Un esempio pratico di cambiamento di stato potrebbe essere quando il giocatore supera un livello o perde tutte le vite. In questi casi, il metodo notifyObservers() viene invocato per garantire che tutti gli osservatori registrati vengano informati e possano reagire di conseguenza. Ad esempio, un osservatore potrebbe aggiornare il punteggio visualizzato nel gioco, mentre un altro potrebbe gestire la logica per passare al livello successivo o visualizzare un messaggio di vittoria.

- Strategy Design Pattern

  Il Strategy Design Pattern è un pattern comportamentale che consente di definire,in questo caso, azioni dell'eroe come muoversi, sparare bolle e sparare bolle di fuoco e di rendere intercambiabili queste implementazioni senza modificare il codice che le utilizza. Questo pattern è utile quando si ha un comportamento che può variare in base alla situazione, e in questo caso si applica perfettamente per gestire diversi comportamenti di un eroe (ad esempio, un eroe normale rispetto a un eroe invincibile).

  - Interfaccia HeroActionStrategy: Definisce un contratto per le azioni che un eroe può fare, come il movimento e il tiro di bolle. Ogni tipo di eroe che vogliamo supportare (normale, invincibile, ecc.) implementerà questa interfaccia in modo diverso.
  - Implementazioni concrete (NormalHeroAction e InvincibleHeroAction): NormalHeroAction definisce come un eroe normale esegue le azioni (muoversi, sparare bolle). InvincibleHeroAction, invece, definisce come un eroe invincibile esegue le azioni.

  Infatti la classe GameController utilizza il Strategy Pattern per cambiare il comportamento dell'eroe in base al suo stato.

MVC

- Model:

  La parte del modello riguarda lo stato del gioco, il comportamento del giocatore, dei nemici e di altri elementi del gioco.

  Classe Level: gestisce lo stato del livello di gioco. Mantiene i dati relativi ai vari livelli, come lo stato e il comportamento dei giocatori, dei nemici, delle bolle, ecc. Ad esempio, il progresso del livello, lo stato corrente dei nemici, il completamento del livello, ecc. Gestisce anche il progresso del gioco, come la creazione dei nemici e l'aggiornamento degli obiettivi del livello.

  Classe Hero: definisce le proprietà e i comportamenti del personaggio giocatore. Proprietà: come la salute del giocatore, il punteggio, lo stato attuale (come in piedi, saltando, sparando bolle, ecc.). Comportamenti: come il movimento del giocatore, il salto, il lancio delle bolle, ecc.

- View La vista si occupa di visualizzare i dati dal modello e di presentare l'interfaccia utente. La parte della vista si occupa di disegnare la schermata del gioco e di mostrare il feedback delle azioni del giocatore.

  Classe GameView: si occupa di disegnare l'interfaccia di gioco, visualizzando i personaggi, i nemici, le bolle, il punteggio, ecc.

  Questa classe preleva i dati dal modello, come il punteggio del giocatore e lo stato attuale del gioco. Utilizza Java Swing per disegnare questi dati e trasformarli in effetti visivi. Crea la finestra di gioco, imposta i pulsanti, mostra il menu e altri elementi dell'interfaccia. Aggiorna la schermata in modo che il giocatore possa vedere lo stato del gioco (ad esempio, il movimento del personaggio, la scomparsa dei nemici, il rimbalzo delle bolle, ecc.).

- Controller Il controllore gestisce gli input dell'utente e si occupa di coordinare l'interazione tra il modello e la vista.

Classe GameController: è il controllore principale del gioco, che gestisce gli input dell'utente e aggiorna il modello e la vista. Riceve gli input da tastiera (come le frecce direzionali, la barra spaziatrice, ecc.) e aggiorna lo stato del gioco in base a questi input. Aggiorna i dati nel modello (come la posizione del giocatore, lo stato delle bolle, ecc.) e notifica la vista per aggiornare la schermata.

Classe BubbleKeyListener: implementa un ascoltatore di eventi da tastiera per catturare gli input dell'utente. Questa classe si occupa di ascoltare gli eventi della tastiera, come il movimento del personaggio con le frecce direzionali o il lancio delle bolle con la barra spaziatrice. Risponde agli input da tastiera aggiornando lo stato del modello (come la posizione del personaggio, il lancio delle bolle) e assicura che la vista venga aggiornata.