Relazione del progetto "coffeBreak"

Grazia Bochdanovits de Kavna
Alessandro Rebosio
Filippo Riccioti

10 luglio 2025

Indice

Capitolo 1

Analisi

1.1 Descrizione e requisiti

Il software da noi sviluppato è una riproduzione del celebre videogioco arcade *Donkey Kong*, pubblicato per la prima volta nel 1981 da Nintendo.

Il protagonista è Mario, il cui obiettivo è salvare Pauline, rapita dal gigantesco gorilla Donkey Kong. Il giocatore prende il controllo di Mario e lo guida in un platform a livelli fissi, caratterizzato da strutture complesse composte da piattaforme, scale e numerosi ostacoli da superare. All'inizio di ogni partita, Mario dispone di tre vite, ogni volta che entra in contatto con un nemico, come barili rotolanti o fiamme, perde una vita e viene riposizionato all'inizio del livello corrente. La partita continua finché ci sono vite disponibili; una volta esaurite, compare la schermata di *Game Over*, che riporta l'utente al menu principale, da cui è possibile avviare una nuova partita. Il gameplay si basa su una combinazione di tempismo e precisione nei movimenti. Mario può spostarsi verso sinistra o destra, saltare per evitare i nemici e utilizzare le scale per muoversi verticalmente tra le piattaforme. L'obiettivo di ogni livello è raggiungere la cima della struttura, dove si trovano Donkey Kong e Pauline. I quattro livelli predefiniti si ripetono ciclicamente ma presentano una difficoltà crescente: con il progredire del gioco, gli ostacoli si muovono più rapidamente, i nemici diventano meno prevedibili e i percorsi sempre più complessi. Un aspetto centrale del gioco è il sistema di punteggio, che premia il giocatore per varie azioni compiute durante la partita. I punti vengono assegnati per ogni ostacolo evitato, nemico eliminato o oggetto bonus raccolto, come le monete e il martello, posizionati in punti chiave del livello. Un ruolo particolare è riservato al martello, un potenziamento temporaneo che consente a Mario di distruggere barili e nemici per alcuni secondi, offrendo un vantaggio momentaneo e un'opportunità per accumulare punti extra.

Requisiti funzionali

- **Controlli di gioco:** il giocatore deve poter controllare il personaggio principale tramite input direzionali: camminata (destra/sinistra), salto e salita/discesa scale.
- **Gestione dei livelli e ostacoli:** il sistema deve generare quattro livelli di gioco in sequenza e gestire la dinamica degli ostacoli, incluso il lancio periodico dei barili da parte di Donkey Kong.
- **Rilevamento e gestione collisioni:** il sistema deve rilevare le collisione tra il personaggio e gli ostacoli/nemici, e applicare le conseguenze previste (es. perdita di vita o bonus).
- **Sistema di punteggio e condizioni di gioco:** il sistema deve calcolare il punteggio in base a: salti sui barili, raccolta di oggetti bonue ed eliminazione di nemici con il martello. Inoltre, deve rilevare la *condizione di vittoria* quando raggiunge Paulina, e la *condizione di GameOver* quando esaurisce le vite.
- **Interfaccia utente:** il sistema deve fornire un'interfaccia che mostri in tempo reale il punteggio e le vite rimanenti.
- **Menù principale:** il sistema deve offrire un menù iniziale con opzioni per avviare una nuova partita e visualizzare i comandi di gioco.

Requisiti non funzionali

- Il sistema deve garantire una risposta fluida e immediata ai comandi del giocatore.
- Il software deve essere eseguibile su diverse piattaforme desktop, tra cui Windows, macOS e Linux.
- Il sistema deve adattarsi correttamente a diverse risoluzioni video standard, mantenendo proporzioni e leggibilità.

1.2 Modello del Dominio

Il gioco si avvia da un **menù iniziale**, che consente al giocatore di avviare una nuova partita, visualizzare i cinque punteggi più alti raggiunti nelle sessioni precedenti oppure uscire dal gioco. Al momento dell'avvio della partita, il gioco carica il **primo livello** della rotazione ispirata a *Donkey Kong* (Nintendo, 1981), e posiziona automaticamente il personaggio principale (Jumpman) nella posizione iniziale in basso a sinistra.

La **partita si sviluppa su due livelli arcade classici**, che si alternano in rotazione ad ogni completamento, ricreando il comportamento dell'originale. Ogni livello presenta una disposizione fissa di piattaforme, scale, ostacoli e nemici (barili o fiamme), con un obiettivo specifico: raggiungere la cima dello schermo evitando gli ostacoli e salvare la damigella in pericolo.

Il personaggio è controllabile tramite input da tastiera e può muoversi lateralmente, salire scale e saltare. Il gioco gestisce le collisioni con gli elementi di gioco (piattaforme, scale, ostacoli e oggetti bonus) permettendo un'esperienza coerente con l'originale.

La visuale è statica: l'intero livello è visibile in una singola schermata. Tuttavia, il personaggio può cadere oltre la parte inferiore dello schermo. In tal caso, o se viene colpito da un ostacolo, si attiva la condizione di **game over** o di perdita di una vita, mostrando il punteggio accumulato nella sessione in corso e confrontandolo con il record precedente.

Durante la partita, il **punteggio** è visibile nella parte superiore dello schermo e si aggiorna dinamicamente in base alle azioni del giocatore (es. salto dei barili, raccolta bonus, tempo residuo). È anche possibile mettere il gioco in **pausa** e riprendere la partita in qualsiasi momento.

Una volta terminata la partita (completamento dei livelli o esaurimento delle vite), viene mostrata una schermata di fine con l'opzione per tornare al menù principale, avviare una nuova partita o uscire.

Capitolo 2

Design

2.1 Architettura

L'architettura del nostro progetto adotta il pattern **Model-View-Controller**, una scelta consolidata nello sviluppo di applicazioni interattive e, in particolare, di videogiochi. Questo approccio consente una chiara separazione tra:

- **Model**: gestisce lo stato interno del gioco, comprese le entità (come Mario, Donkey Kong, i nemici, le piattaforme, ecc.), le logiche di gioco, le collisioni e il punteggio. L'organizzazione del codice segue il principio della *Single Responsibility*, assegnando a ciascuna classe un compito ben definito.

- **Controller**: gestisce il *game loop* e riceve gli input effettivi da tastiera, che vengono poi inoltrati al Model per essere elaborati come azioni di gioco. Il Controller si occupa quindi di aggiornare, a ogni frame, sia il Model che la View, coordinando l'interazione tra logica di gioco e presentazione visiva.

- **View**: si occupa di tutte le componenti visuali e delle interfacce di gioco. Disegna lo stato corrente durante i vari *game states* (menu, in-game, pausa, game-over), avvalendosi di un *Render Manager* per il disegno delle entità durante il gameplay.

Nel caso specifico di *CoffeeBreak*, ispirato a *Donkey Kong*, questa struttura ha permesso di progettare un'architettura modulare, facilmente estendibile e manutenibile. Inoltre, l'adozione del pattern MVC ha facilitato il testing delle singole componenti e reso possibile il lavoro in parallelo tra i membri del team, migliorando l'efficienza della collaborazione.

2.2 Design dettagliato

2.2.1 Alessandro Rebosio

Gestione degli stati del gioco

Problema Nei videogiochi, è comune avere diversi stati di esecuzione, come *menu*, *in-game*, *pausa* e *game-over*. Una gestione non strutturata di questi stati può portare a codice confuso, con numerosi controlli condizionali sparsi e difficoltà nel mantenimento e nell'estensione del comportamento del gioco. Inoltre, diventa complesso isolare la logica specifica di ciascuno stato, con il rischio di introdurre bug quando si modifica o aggiunge un nuovo stato.

Soluzione Per affrontare il problema, abbiamo adottato lo **State Pattern**, che permette di rappresentare ogni stato del gioco come un oggetto distinto, con comportamento e logica propri. Ogni stato implementa una comune interfaccia, garantendo uniformità nell'aggiornamento e nel rendering. Il `GameModel` mantiene il riferimento allo stato corrente. Questo approccio migliora la modularità del codice, rende più facile aggiungere o modificare stati in futuro e contribuisce a una migliore organizzazione del flusso di esecuzione.

Gestione dei collezionabili

Problema Nel gioco sono presenti diversi tipi di oggetti collezionabili (ad esempio monete, power-up), che condividono comportamenti comuni come la raccolta, l'attivazione di effetti e la rimozione dalla scena. Implementare queste funzionalità separatamente per ogni tipo di collezionabile porta a duplicazione di codice e rende difficile mantenere coerenza e gestire eventuali modifiche comuni.

Soluzione Per risolvere questo problema, abbiamo adottato il **Template Method**, definendo una classe base astratta per i collezionabili che implementa lo scheletro generale del processo di raccolta, lasciando alle sottoclassi il compito di definire i dettagli specifici (ad esempio l'effetto attivato al momento della raccolta). Questo approccio permette di riutilizzare il codice comune e garantisce una struttura chiara e uniforme, facilitando l'estensione con nuovi tipi di collezionabili senza modificare il comportamento di base.

TODO

Problema

Soluzione

2.2.2 Grazia Bochdanovits de Kavna

Gestione degli stati di Mario

Problema Nel gioco, il personaggio di Mario può trovarsi in diversi stati con comportamenti distinti, come lo stato normale e lo stato con il martello. Se questi venissero gestiti tramite controlli condizionali direttamente nella classe di Mario, il codice violerebbe l'Open/Closed principle, diventando rigido e difficile da estendere a nuove abilità future.

Soluzione Per gestire in maniera flessibile gli stati di Mario viene implementato lo **State Pattern**, definendo un'interfaccia *CharaterState* che dichiara i comportamenti comuni a tutti gli stati, lasciando alle sottoclassi l'implementazione di quelli specifici (come ad esempio la possibilità o no di arrampicarsi sulle scale). In questo modo, aggiungere nuovi stati diventa semplice e non rischiede modifiche a quelli già implementati, rendendo il codice più modulare, estendibile e facile da mantenere.

TODO

Problema

Soluzione

TODO

Problema

Soluzione

Capitolo 3

Sviluppo

3.1 Testing automatizzato

In questo progetto abbiamo implementato dei test unitari con JUnit 5 per tutte le classi principali. I test progettati assicurano la verifica automatica delle funzionalità fondamentali del software. Di seguito vengono riportati alcuni esempi di test implementati.

Common

- `TestBoundingBox`: verifica la corretta creazione e il corretto funzionamento delle dimensioni delle entità.
- `TestResourceLoader`: viene verificato il giusto caricamento delle risorse del gioco.
- `TestPosition`: testa la gestione delle posizioni nello spazio di gioco.
- `TestVector`: verifica le operazioni sui vettori.

Controller

- `TestInputManager`: controlla la gestione degli input da tastiera.
- `TestGameController`: testa la logica principale di controllo del gioco.

Repository

- `TestScoreRepository`: verifica la corretta gestione della persistenza e del recupero dei punteggi nella repository.

Model

- `TestMario`: controlla che il movimento e le funzionalità del personaggio principale avvengano correttamente.
- `TestLivesManager`: controlla la gestione delle vite del personaggio principale.
- `TestNormalState`: verifica il comportamento base del personaggio nello stato normale.
- `TestWithHammerState`: verifica l'attivazione del potere alla raccolta del martello, e la sua durata.
- `TestPlatform`: controlla il comportamento delle piattaforme normal e breackable.
- `TestLadder`: verifica il funzionamento delle scale.
- `TestGameTank`: verifica il funzionamento della tanica d'olio.
- `TestPauline`: testa il comportamento dell'ostaggio.
- `TestDonkeyKong`: verifica la logica del lancio dei barili e il comportamento dell'antagonista.
- `TestBarrel`: controlla il movimento e le collisioni dei barili.
- `TestFire`: verifica il comportamento delle fiamme.
- `TestCollectible`: testa la raccolta degli oggetti collezionabili (monete, martelli) e il loro effetto.
- `TestEntry:` verifica la corretta gestione delle singole voci della classifica.
- `TestGameLeaderBoard:` verifica la gestione e il salvataggio dei punteggi nella leaderboard.
- `TestScore:` controlla la gestione e il calcolo dei punteggi.
- `TestBonus:` verifica la logica dei bonus di gioco.
- `TestInGameModelState:` testa lo stato del modello durante la partita.
- `TestMenuModelState:` verifica il comportamento del modello nel menu principale.
- `TestPauseModelState:` controlla la gestione dello stato di pausa.

3.2 Note di sviluppo

3.2.1 Alessandro Rebosio

Progettazione con `generics`

Utilizzati i generics per definire interfacce e classi parametrizzate. Permalink
`https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/3a264084eabd86bac19f9978d9c2aba0c1cbf624/src/main/java/it/unibo/coffebreak/api/common/State.java#L14`

Utilizzo di `Stream`

Utilizzati di frequente, soprattutto per il controllo sulle entità. Permalink di un esempio
`https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/3a264084eabd86bac19f9978d9c2aba0c1cbf624/src/main/java/it/unibo/coffebreak/impl/model/physics/collision/GameCollision.java#L54-L67`

Utilizzo di lambda expressions

Utilizzati di frequente, nel caricamento delle risorse. Permalink di un esempio:
`https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/3a264084eabd86bac19f9978d9c2aba0c1cbf624/src/main/java/it/unibo/coffebreak/impl/common/ResourceLoader.java#L99`

Gestione degli `Optional`

Usato per gestire valori che potrebbero essere assenti. Permalink di un esempio
`https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/3a264084eabd86bac19f9978d9c2aba0c1cbf624/src/main/java/it/unibo/coffebreak/impl/model/states/ingame/InGameModelState.java#L57-L61`

3.2.2 Grazia Bochdanovits de Kavna

Utilizzo di `Stream`

Utilizzati di frequente. Permalink di un esempio
https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/926d3f1b985c9bba79c5f766ac0070b04fd6185f/src/main/java/it/unibo/coffebreak/impl/view/states/ingame/InGameView.java#L62-L65

Gestione degli `Optional`

Usato per gestire valori che potrebbero essere assenti. Permalink di un esempio
https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/926d3f1b985c9bba79c5f766ac0070b04fd6185f/src/main/java/it/unibo/coffebreak/impl/view/render/GameRenderManager.java#L82-L87

Utilizzo di lambda expressions

Utilizzati di frequente, soprattuto nel controllo delle collisioni fra entità. Permalink di un esempio:
https://github.com/alessandrorebosio/OOP24-coffeBreak/blob/926d3f1b985c9bba79c5f766ac0070b04fd6185f/src/main/java/it/unibo/coffebreak/impl/model/entities/mario/Mario.java#L173-L186

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Alessandro Rebosio

Dopo un'attenta analisi iniziale, ho iniziato subito a scrivere le classi principali del progetto. Tuttavia, man mano che procedevo nello sviluppo, mi rendevo conto di come alcune parti potessero essere migliorate, e dove possibile ho effettuato operazioni di refactoring. Sono consapevole che ci siano ancora margini di miglioramento, ma questa esperienza mi ha fatto comprendere quanto l'utilizzo dei pattern sia utile per organizzare meglio il lavoro e ridurre la complessità complessiva.

Essendo il mio primo progetto strutturato, ho cercato fin da subito di scrivere codice il più possibile chiaro, modulare ed espandibile, in modo da facilitarne la manutenzione e l'evoluzione futura.

All'interno del gruppo ho ricoperto diversi ruoli, anche se mi sono concentrato in particolare sullo sviluppo del Controller. Ho anche definito una base per la View, per poi lasciarne lo sviluppo ai miei colleghi. Questo mi ha dato modo di sperimentare la sincronizzazione e l'interazione tra le diverse componenti dell'architettura, aiutandomi a comprendere più a fondo le dinamiche di progettazione in un contesto MVC.

4.2 Difficoltà incontrate e commenti per i docenti

Questa sezione, **opzionale**, può essere utilizzata per segnalare ai docenti eventuali problemi o difficoltà incontrate nel corso o nello svolgimento del progetto, può essere vista come una seconda possibilità di valutare il corso

(dopo quella offerta dalle rilevazioni della didattica) avendo anche conoscenza delle modalità e delle difficoltà collegate all'esame, cosa impossibile da fare usando le valutazioni in aula per ovvie ragioni. È possibile che alcuni dei commenti forniti vengano utilizzati per migliorare il corso in futuro: sebbene non andrà a vostro beneficio, potreste fare un favore ai vostri futuri colleghi. Ovviamente *il contenuto della sezione non impatterà il voto finale.*

Appendice A

Guida utente

A.1 Menu Principale

All'avvio dell'applicazione, l'unico strumento a disposizione sarà la tastiera. Nel menu principale sarà possibile visualizzare la leaderboard e selezionare le opzioni disponibili tramite le frecce `SU` e `GIÙ`. Per confermare la selezione si utilizza il tasto `ENTER`.

A.2 Durante il Gioco

Entrati in gioco, si parte dal primo livello: lo scopo è raggiungere la principessa evitando i nemici e raccogliendo power-up.

Per giocare basta usare la tastiera: ogni `freccia` muove il personaggio nella rispettiva direzione, la `barra spaziatrice` fa saltare, mentre le frecce `SU` e `GIÙ` funzionano solo in corrispondenza di una scala. In qualsiasi momento, premendo il tasto `ESC` puoi mettere il gioco in pausa.

A.3 Menu di Pausa

Nel menu di pausa si naviga tra le opzioni disponibili con le frecce `SU` e `GIÙ`, e si seleziona l'opzione desiderata con `ENTER`.

A.4 Game Over

Quando il personaggio perde tutte le vite, si entra nella schermata di Game Over. In questa schermata l'unica azione possibile è premere `ENTER` per tornare al menu principale.

Appendice B

Esercitazioni di laboratorio

B.1 alessandro.rebosio@studio.unibo.it

- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=177162#p246059`
- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=179154#p248324`
- Laboratorio 10: `https://virtuale.unibo.it/mod/forum/discuss.php?d=180101#p249805`
- Laboratorio 11: `https://virtuale.unibo.it/mod/forum/discuss.php?d=181206#p250995`

Bibliografia

[Cla25] ClassicGaming.cc. Donkey kong - sounds. Online resource, 2025. Archivio sonoro dei suoni originali dell'arcade Donkey Kong (Nintendo, 1981), utilizzato per integrare effetti sonori autentici nel progetto. Include salti, cadute, rumori ambientali e suoni di interazione.

[Goo25] Google Fonts. Press start 2p — google fonts. Font disponibile tramite Google Fonts, 2025. Font bitmap ispirato alla tipografia dei videogiochi arcade degli anni '80, usato per i testi e i punteggi nel progetto. Garantisce coerenza stilistica con l'estetica retro.

[The25] The Spriters Resource. Donkey kong (arcade) sprites. Online sprite archive, 2025. Collezione completa degli sprite estratti dal gioco originale Donkey Kong. Utilizzati per ricostruire graficamente il gameplay e l'interfaccia in stile retro. Include animazioni di Mario, Donkey Kong, ostacoli e oggetti.