Relazione per
“Programmazione ad oggetti”
Super Peach

Maurizio Capuano

Patrick Sbrighi

Miriam Sonaglia

Eraldo Tabaku

18 Febbraio 2024

Indice

Capitolo 1

Analisi

Il gruppo si pone come obiettivo quello di realizzare una versione personalizzata del primo livello del classico gioco “Super Mario Bros” del 1985.

Nello specifico, l’adattamento prevede la sostituzione dell'ormai celebre protagonista del gioco con la figura della principessa Peach come personaggio giocabile, con conseguente adattamento dell’ambiente di gioco, allineato all’identità della protagonista e alle idee stilistiche del gruppo.

1.1 Requisiti

Requisiti funzionali

- La suddetta protagonista sarà in grado di muoversi in due direzioni e saltare a comando per potersi spostare all’interno del suo mondo bidimensionale.
- Una telecamera seguirà ogni suo spostamento orizzontale.
- Il personaggio interagirà con i blocchi di gioco, potendo distruggere e attivare alcuni di essi per raccogliere potenziamenti e collezionare oggetti.
- La generazione del livello ad inizio partita, oltre alla creazione del mondo, creerà vari tipi di nemici a movimento autonomo che costituiranno degli ostacoli al giocatore.
- Prima di poter giocare il livello, il gioco presenterà all’utente un menù di gioco iniziale che permetterà di modificare alcune opzioni.
- Il sistema di gestione di vittoria e sconfitta sarà corredato dal controllo effettuato sulle vite rimaste al giocatore e i punti collezionati.

Requisiti non funzionali

- Si prospetta che il personaggio protagonista abbia, visivamente, un movimento fluido e di facile controllo.
- Le entità di gioco interagiranno tra loro seguendo un sistema efficiente di collisione.
- L’interfaccia grafica di gioco sarà semplice e intuitiva ed avrà un aspetto visivamente gradevole.
- Il software dovrà essere quanto più efficiente in termini di utilizzo di risorse.

1.2 Analisi e modello del dominio

Super Peach è un gioco platform a scorrimento orizzontale con vista laterale. La protagonista, la principessa Peach, ha come obiettivo quello di arrivare al castello di fine livello cercando di superare vari ostacoli dati dalla struttura della mappa e dei nemici che le saranno d'intralcio.

Il personaggio di Peach può muoversi a destra o sinistra e saltare.

Proprio questa operazione di salto è l'attacco fondamentale del personaggio, in quanto, saltando sopra la testa dei nemici, li eliminerà dalla mappa. Saltando, invece, in modo da colpire un blocco da sotto, con la testa, Peach ha la possibilità di distruggerlo, in caso il blocco rappresenti dei mattoni, o ricevere potenziamenti in caso si tratti di un "Lucky Block".

L'apertura di un *Lucky Block* potrà generare differenti tipi di potenziamenti e oggetti collezionabili dalla protagonista:

- *Monete d'oro*, semplici oggetti collezionabili;
- *Funghi della crescita*, che aumentano la dimensione di Peach quando è in versione "mini" dandole la possibilità di distruggere blocchi di mattoni e poter resistere ad un attacco nemico che, però, la costringerà a tornare alla sua forma di base.
- *Stella*, che rende Peach temporaneamente invincibile permettendole di uccidere tutti i nemici al semplice contatto.
- *Funghi della vita*: restituiscono una vita al giocatore in caso ne abbia persa qualcuna durante il gioco.

Cadendo in un fossato o venendo colpita da un nemico, la protagonista perderà una vita e, nel caso in cui perdesse la sua ultima vita, la partita terminerà e verrà visualizzato un messaggio di "Game Over".

La mappa di gioco comprende, come già accennato in precedenza, la generazione di blocchi di vario tipo, ostacoli strutturali (quali tubi, fossati e scale), il disegno di uno sfondo di gioco, che comprende elementi puramente estetici, ed una quest di fine livello sul "salto della bandiera" riprodotta dal gioco originale.

Dopo aver avuto successo o aver fallito nel prendere il punteggio bonus toccando la punta della bandiera, l'arrivo all'entrata della torre posta alla fine della mappa permette di vincere la partita.

I nemici presenti nella mappa saranno tra i classici del gioco originale di Super Mario, tutti generati in punti precisi della mappa e tutti aventi pattern di movimento differenti per poter garantire diversi tipi di ostacolo alla vittoria.

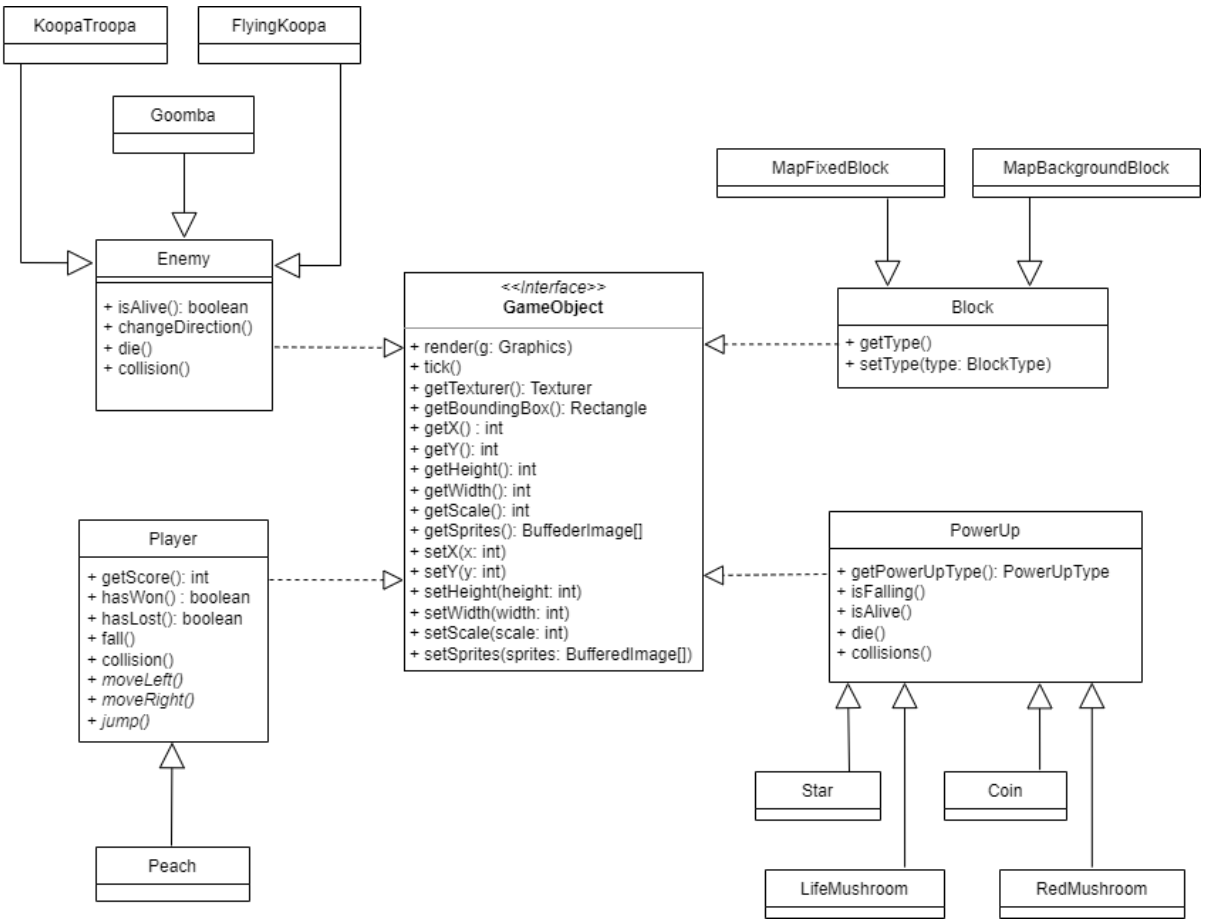

Figura 1.1 schema UML dell'analisi del problema, rappresentazione della la struttura del dominio con relative estensioni

Capitolo 2

Design

2.1 Architettura

L'architettura del software Super Peach segue, nel complesso, il pattern architetturale MVC.

Nello specifico, la parte del *Controller* si occupa di gestire gli elementi del *Model* attraverso l'implementazione di "handlers" e la gestione degli eventi dei singoli input da tastiera.

La componente *View* del progetto, concentrata perlopiù nel package "graphics", si occupa della visualizzazione a schermo di tutte le interazioni tra utente e gioco o tra entità di gioco ed entità di gioco. Effimeri stralci di *View* sono anche individuabili all'interno della classe d'avvio Game e all'interno dei gestori degli elementi del dominio poiché ad ogni ciclo di "game loop" va effettuato il rendering di ogni singola entità per aggiornare le loro posizioni.

L'utilizzo di questo tipo di architettura permette l'implementazione di una quantità arbitraria di entità di gioco. Sebbene sia molto semplice aggiungerne, è ovviamente necessario mettere mano all'interno di elementi del *Controller* in quanto sono questi a gestirne l'effettiva presenza all'interno del gioco.

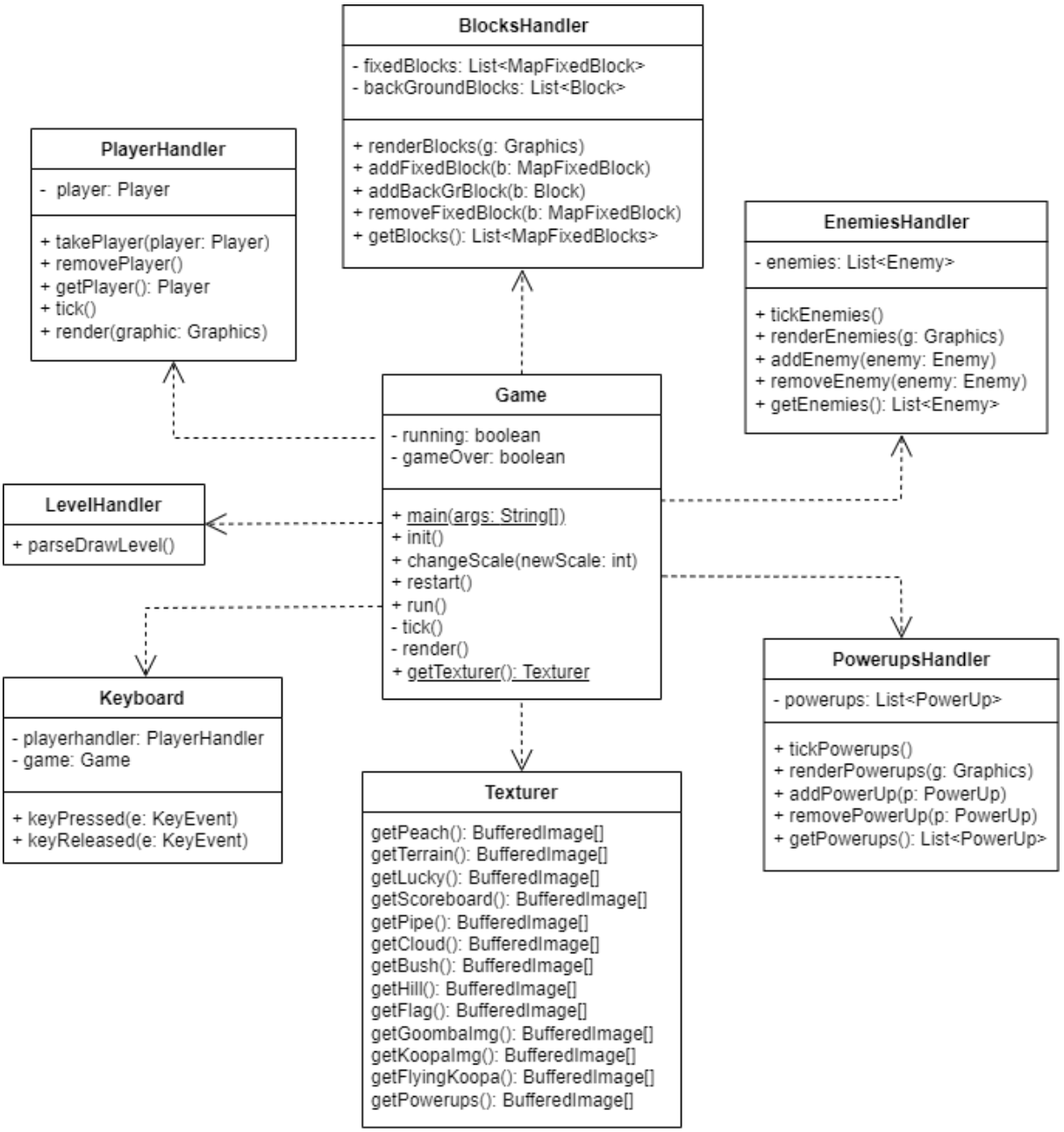

Figura 2.1 Schema UML architetturale del software. La classe Game permette alle classi Handler di gestire tutti gli oggetti di gioco e le mette in comunicazione tra loro. Keyboard si occupa di gestire le interazioni indotte dagli input da tastiera e Texturer carica nel progetto tutti gli sprites necessari per permettere ai singoli oggetti di disegnarsi con lo sprite giusto.

2.2 Design dettagliato

Maurizio Capuano

Il mio compito all'interno del gruppo costituiva la creazione, gestione ed implementazione di ogni elemento fisso facente parte della mappa di gioco.
Ho inoltre costruito ed implementato la classe di gestione del Thread di gioco "*Game*", tramite la quale avviene la costruzione di tutte le componenti del software, e della grafica di gioco una volta superato il menù principale e aver iniziato a giocare.

Nella progettazione, una significativa parte di quella che adesso è la View non era stata presa in considerazione. Forse frutto della nostra totale inesperienza sulle interfacce grafiche, sono riuscito solo in un secondo momento a capire come fare il setting delle varie componenti grafiche e gestirne la comunicazione all'interno dell'architettura, a partire dal retrieve dei file nelle risorse del progetto fino ad arrivare all'effettiva costruzione degli elementi grafici, visibili a schermo con i corretti adattamenti.
Di seguito, il processo creativo nel dettaglio suddiviso nei singoli problemi affrontati e risolti:

Problema: estensibilità dei blocchi (Blocks) di gioco per permettere a futuri eventuali cambiamenti di poter aggiungere altri elementi dello stesso tipo ma dal comportamento diverso.

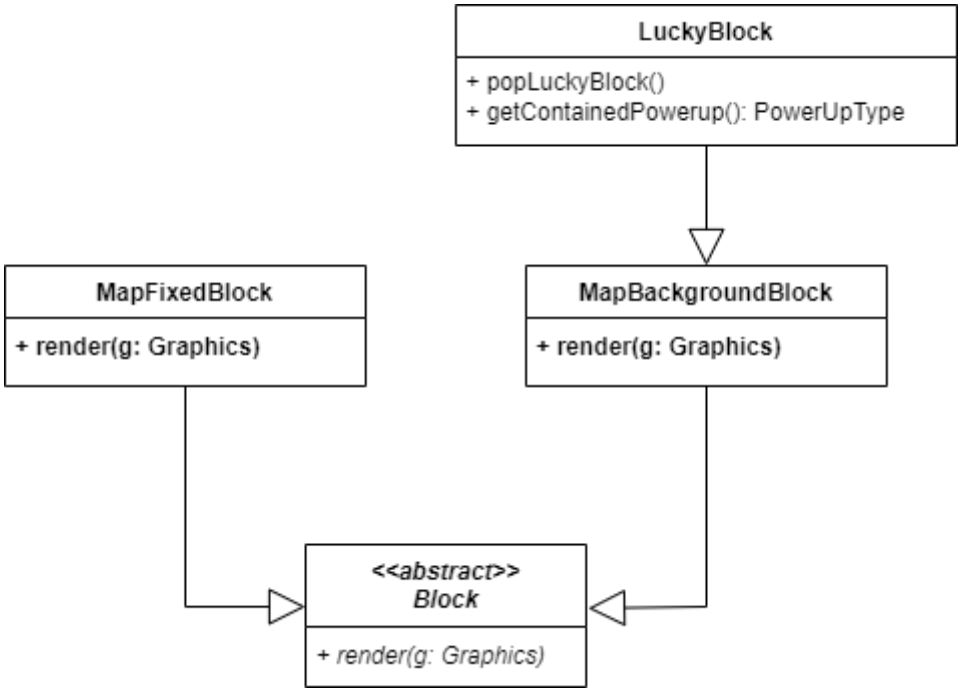

Figura 2.2.1: Rappresentazione UML di una componente del model dell'applicazione dove viene applicato il pattern Template Method alla gerarchia del tipo di blocchi.

Soluzione: grande classico quando si parla di estensione di classi, Template method è il pattern di design che mi ha permesso di realizzare quello di cui avevo bisogno potendo

delegare alle classi figlie di Block la definizione del loro comportamento nel rendering che non potrà poi essere ridefinito in ulteriori classi figlie (come ad esempio LuckyBlock che aggiunge solo funzionalità senza cambiare quelle esistenti).

Problema: non voglio delegare alla classe Game la gestione delle collezioni che conterranno i miei elementi della mappa.

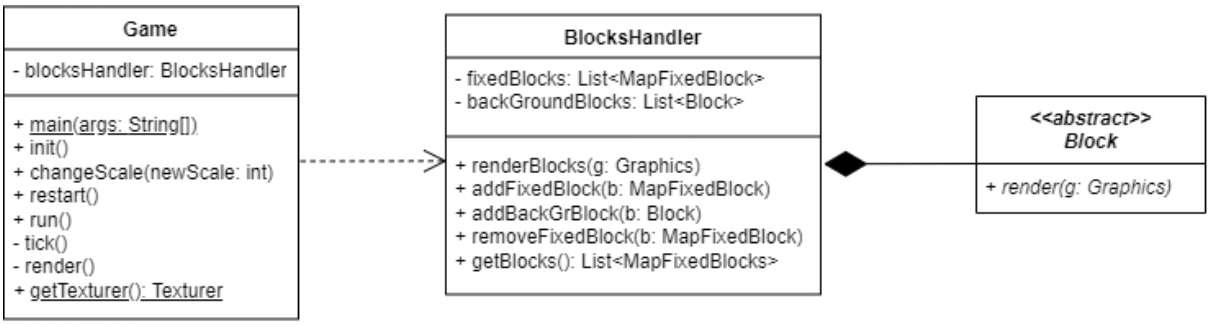

Figura 2.2.2: Rappresentazione UML di una componente del controller dell'applicazione dove viene applicato una sorta di Mediator pattern.

Soluzione: Game al suo interno non conterrà una collezione di blocchi, avrà invece un riferimento ad un oggetto mediatore che si occuperà di manipolare le diverse collezioni.
Dettaglio puramente implementativo: la collezione di blocchi di sfondo è definita su oggetti del tipo della classe astratta poiché è considerata la possibilità che alcuni blocchi (codificati dal parser del livello nella classe LevelHandler) non siano raggiungibili dal personaggio e quindi, per alleggerire il carico di controlli effettuati sui blocchi, alcuni blocchi che appaiono come FixedBlocks sono in realtà blocchi di sfondo con i quali non è possibile interagire.

Miriam Sonaglia

Ho contribuito a questo progetto tramite l'implementazione e la gestione dei power ups all'interno della mappa.
Mi sono, inoltre, occupata dell'implementazione dell'interfaccia grafica e sonora del menu e delle relative opzioni disponibili ovvero: il tasto che permette di iniziare a giocare, la scelta delle musiche e delle dimensioni della GUI e, infine, il tasto di uscita.

L'implementazione dei compiti a me assegnati non è stata immediata e mi sono imbattuta in diverse problematiche che elencherò di seguito assieme alle relative soluzioni trovate.

Problema: gestione delle collisioni con il mondo.
Soluzione: ho usufruito di un BlocksHandler, così da avere una lista con tutti i blocchi presenti nella mappa. In questo modo, ad ogni tick di gioco, ho l'opportunità di controllare, tramite i metodi getBounds, cosa il mio power up stia toccando e dove si stia muovendo. Ciò mi ha quindi permesso di gestire anche il movimento e la morte dei miei power ups tramite i metodi updateCoords() e die().

Problema: corretta applicazione del principio DRY durante la programmazione. In entrambe le parti di cui mi sono occupata era molto semplice, infatti, ripetere delle stesse parti di codice.
Soluzione, PowerUps: ho racchiuso tutti i metodi che potevano essere comuni ai singoli power up in una classe madre astratta, *PowerUp*. In questo modo, ho potuto sia evitare di scrivere delle classi pressoché identiche, sia facilitare il riuso di questa parte di codice semplificando l'eventuale creazione di ulteriori power ups.

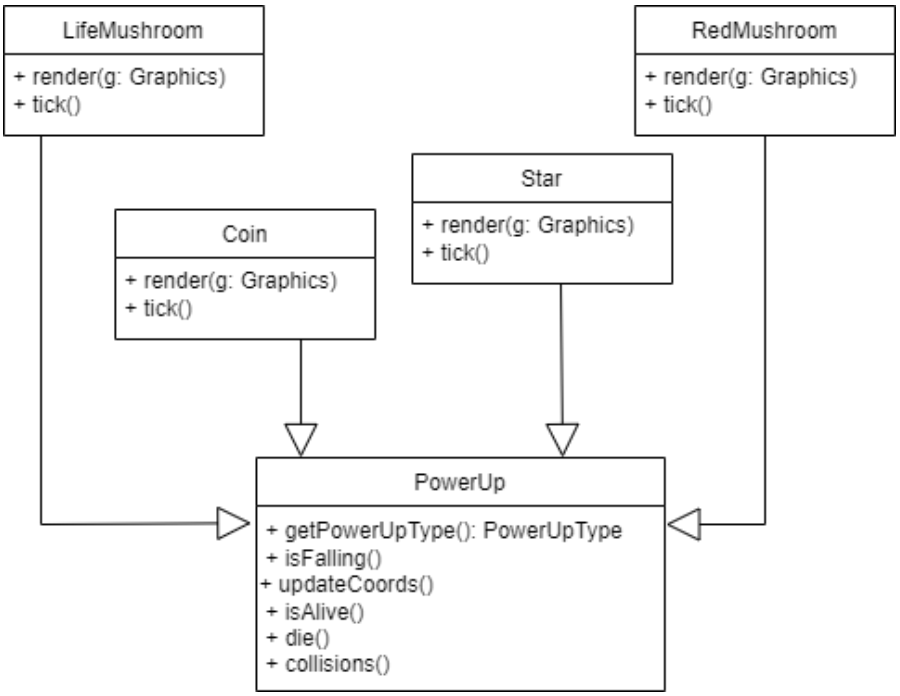

Figura 2.2.3: Rappresentazione UML della classe madre PowerUp estesa dalle quattro classi figlie.

Soluzione, CustomButton: durante la creazione dell'interfaccia, mi sono resa conto che stavo utilizzando degli stessi metodi per creare dei JButton, ho così scelto di creare una classe generica CustomButton che facilita l'implementazione di un qualsiasi button nella classe PeachMenu.

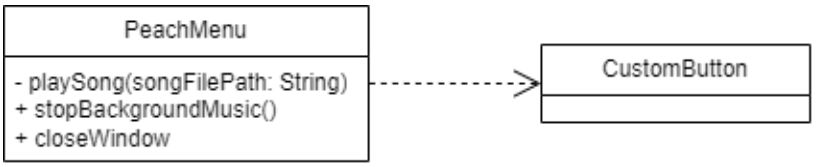

Figura 2.2.4: Rappresentazione UML della classe CustomButton utilizzata da PeachMenu

Eraldo Tabaku

Personalmente, mi sono occupato dell'implementazione dei nemici all'interno del gioco, dei loro movimenti, della loro corretta gestione delle collisioni con l'ambiente e di tutta la parte grafica annessa ad essi, ovvero tutto ciò che riguarda la gestione, l'aggiornamento e il caricamento dei vari sprite necessari. Per rispettare il pattern MVC, ho affidato la gestione del Control all'EnemiesHandler, il quale possiede come unico campo un ArrayList di nemici che vengono gestiti dall'handler stesso. Nel corso del progetto ho cercato di tenere a mente tutti i principi di buona programmazione per produrre del codice semplice (KISS principle) e cercando di produrre del codice propenso ad una eventuale estensione futura.
In conclusione, mi sono occupato della scrittura di buona parte delle classi di Testing presenti nel progetto.

Problema: Tutti i nemici hanno caratteristiche simili tra di loro e differiscono per pochi comportamenti, quindi sarebbe necessario rendere più conciso il codice rispettando anche il principio DRY.

Soluzione: Per risolvere il problema ho utilizzato un pattern di programmazione *template method* creando, così, una classe astratta Enemy che implementa l'interfaccia GameObject, dalla quale ereditano tutte le diverse categorie di nemici e specificano le loro peculiarità mediante i metodi render() e tick() di cui viene fatto un Override.
E' stato possibile definire tutti i metodi in comune ai nemici nella classe astratta Enemy in maniera da rispettare il principio DRY; questa realizzazione permette, inoltre, un facile ampliamento del codice rendendo semplice l'aggiunta di nuove categorie di nemici grazie all'ereditarietà, infatti, durante il processo di sviluppo iniziale vi era solamente pianificato l'inserimento di due tipologie di nemici (KoopaTroopa e Goomba) ma grazie alla progettazione fatta mi è stato semplice inserire anche la terza categoria di nemici (FlyingKoopa) senza troppi problemi.

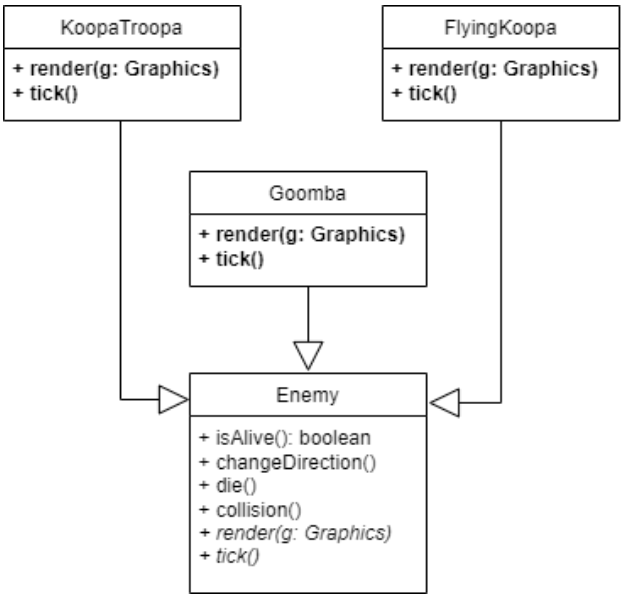

Figura 2.2.5 che mostra l'implementazione del pattern Template Method

Problema: Come si potrebbe procedere per generare i nemici sulla mappa?

Soluzione: Per risolvere il problema ho pensato che avrei potuto generare i nemici all'interno della mappa assieme alla mappa stessa. All'interno della classe LevelHandler, tramite il metodo parseDrawImage(), un file png di pixel colorati viene letto per intero come se fosse una matrice e, in base ai valori RGB, vengono associati i singoli sprite e, di conseguenza, vengono aggiunti i nemici istanziati all'interno dell'EnemiesHandler.

Problema: Come gestisco la comunicazione tra Game e i nemici?

Soluzione: Ciò viene gestito grazie all'EnemiesHandler che implementa una sorta di *Mediator Pattern* per cui viene gestita la comunicazione tra Game e i singoli nemici mediante la classe di mezzo.

Patrick Sbrighi

Io mi sono occupato della realizzazione di Peach, il personaggio principale, di come farla muovere in base ai tasti premuti dell'utente, di come rilevare quali tasti sono stati premuti e di farla interagire con tutti gli oggetti presenti nel gioco. Nel dettaglio, ho realizzato le classi Player e Peach, per la modellazione del personaggio, e Keyboard, per la gestione dell'interazione con l'utente.

Inoltre, per rispettare MVC e le specifiche prese in gruppo, ho realizzato la classe PlayerHandler che al suo interno contiene il giocatore. Nell'implementare questi elementi mi sono trovato di fronte a problemi per me nuovi e che non mi aspettavo di dover risolvere perchè non mi era mai capitato di realizzare un progetto di questo tipo e di queste dimensioni. Ho comunque cercato per tutta la scrittura del codice di utilizzare i concetti imparati a lezione e di rispettare i consigli per migliorare il codice dati in classe.

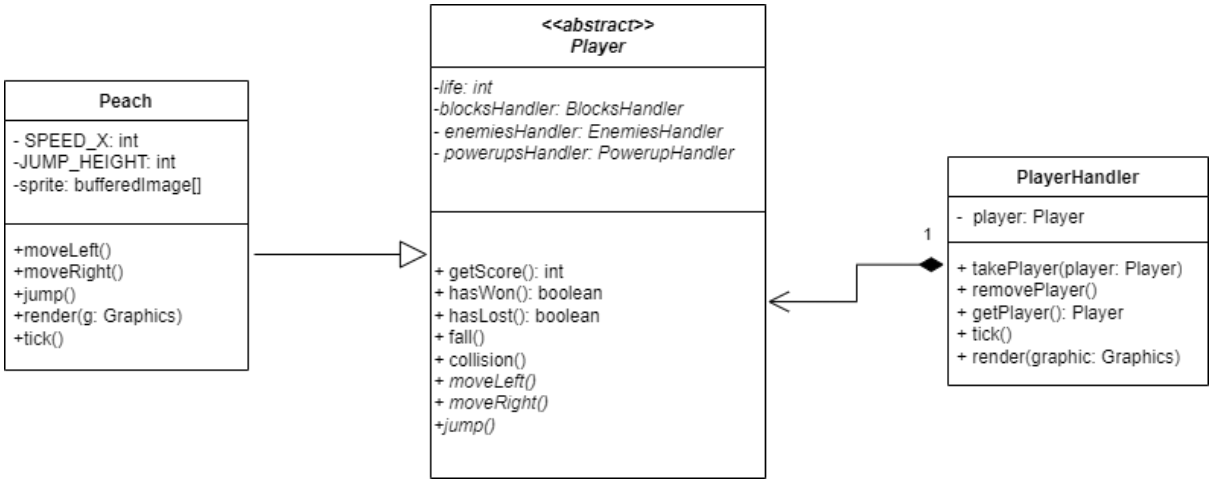

Figura 2.2.6 mostra l'implementazione del Template Method tra Player e Peach, inoltre possiamo notare una sorta di Mediator Pattern applicato a PlayerHandler

Problema: Rendere possibile estendere il gioco in futuro implementando nuovi personaggi con caratteristiche diverse
Soluzione: Per risolvere questo problema ho utilizzato il pattern Template Method. Ho quindi implementato una classe astratta Player che gestisce tutti i comportamenti generali che può avere un player, come, per esempio il conteggio dei punti, quello delle vite, la gravità e le collisioni con il resto del mondo, mentre, gli elementi che hanno caratteristiche diverse, vengono definiti come abstract dentro player, ma implementati dentro la classe Peach. In questo modo è possibile avere sprite diversi per ogni personaggio ed implementare diverse specifiche per saltare e muoversi. Infine, ereditando metodi comuni dalla classe player permette di non riscrivere inutilmente del codice (DRY).

Problema: Come sapere se il personaggio sta collidendo con qualche elemento del gioco
Soluzione: Uno dei problemi principali è stato controllare se il giocatore stesse toccando o no un qualche oggetto all'interno del mondo, riconoscere quale tipo di oggetto è stato toccato e con quale parte del corpo. Per risolvere questo problema mi è tornato molto utile l'aver scelto, insieme ai miei compagni, di implementare un handler per ogni oggetto del gioco. In questo modo, la classe Player ha a sua disposizione tutti gli elementi e con essi può controllare se il giocatore ne sta toccando qualcuno, attuando un diverso comportamento in base al tipo di elemento e al modo in cui esso è stato toccato.

Figura 2.2.7 mostra la relazione tra Game e Keyboard

Problema: Come gestire le interazioni tra giocatore e gioco
Soluzione: Ho implementato una classe Keyboard che si occupa unicamente di rilevare eventuali click sulla tastiera e di far svolgere i diversi comportamenti al player. Questa classe è stata aggiunta come KeyListener dentro la classe Game (implementata da Maurizio Capuano) potendo così collegare la rilevazione degli eventi generati dalla tastiera con il gioco effettivo.

Problema: Ad ogni click dell'utente il personaggio deve fare ciò che gli è stato chiesto
Soluzioni: Per permettere ciò ho beneficiato ancora una volta degli handler degli oggetti di gioco, in questo caso PlayerHandler, così ho potuto utilizzare i diversi metodi implementati dalla classe Player per permettere al giocatore di muoversi, scegliendo in base al tasto premuto dall'utente. Possiamo definire il tutto come una sorta di Mediator Pattern.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il testing automatizzato del progetto è stato svolto utilizzando la suite di testing JUnit.
I test realizzati mirano a consentire la verifica della mancanza di regressione nelle funzionalità a fronte di eventuali aggiornamenti futuri. Nello specifico, i test creati cercano di garantire il corretto funzionamento di Model e Controller.

- *BlocksHandlerTest*: viene utilizzata per testare la corretta aggiunta e rimozione di blocchi gestiti dal BlockHandler.
- *BlockTest*: utilizzata per testare la corretta creazione dei blocchi di gioco (fixed o background) e il metodo per la restituzione del Rectangle bound appartenente ai singoli blocchi.
- *EnemiesHandlerTest*: verifica l'aggiunta di nemici, la loro rimozione, il tick() di tutti i nemici gestiti dall'EnemyHandler e la corretta esecuzione degli eventi di morte dei nemici, con conseguente rimozione dalla lista dell'EnemyHandler.
- *EnemyTest*: utilizzata per testare la creazione dei nemici, la corretta collisione dei nemici coi death block e con i fixed block
- *PlayerHandlerTest*: verifica l'aggiunta e la rimozione del player dal PlayerHandler.
- *PlayerTest*: verifica la corretta creazione e rimozione di un Player.
- *PowerupsHandlerTest*: verifica l'aggiunta e la rimozione dei Powerups nella lista del PowerupHandler.
- *PowerupsTest*: testa creazione, distruzione e collisioni coi FixedBlock dei Powerups.

Commenti Finali

Autovalutazione e progetti futuri

Maurizio Capuano

È innegabile che prendere parte ad un progetto di gruppo comporti delle responsabilità che non ci si immagina nemmeno di affrontare quando si crea qualcosa da soli, e la portata del lavoro, sebbene facessi fatica ad immaginarlo, aumenta in maniera notevole.
Ma lavorare in team è una skill che si acquisisce con tempo ed esperienza, e lavorare così tanto tempo e con tale immersione a questo progetto credo abbia significativamente migliorato tutte, e dico tutte, le mie soft skills sul team working.
Non avevo chissà quale esperienza nemmeno con i sistemi di controllo versione, e cimentarmi così, di petto, nella gestione di una repository condivisa è stato abbastanza faticoso durante l'intero percorso, ma anche questo, devo sinceramente ammettere, mi ha insegnato molte cose sul campo pratico che probabilmente non avrei mai incontrato a meno di un esperienza di questo tipo.
Sulla progettazione mi sento di dire che forse ho ancora qualcosa da imparare, parlo sia personalmente sia a nome dell'intero gruppo con cui ho trascorso questa tortuosa esperienza, ma non intendo dire che ciò che è stato fatto dovrebbe essere buttato, sono fiero del risultato ottenuto perché non mi sono mai avvicinato a qualcosa di simile con così tanta efficienza e pulizia in termini di sviluppo. Forse sarebbe anche carino portare avanti il risultato per migliorarlo ancora, renderlo più scalabile e portabile inserendo parecchie interfacce e risolvendo lacune architetturali, ma sicuramente non riguarderà il futuro prossimo, ho già speso moltissimo tempo ed energie e devo recuperare alcune cose che ho lasciato indietro (accademicamente parlando).
Ma ripeto, sebbene potrebbe non sembrare dalle ultime parole, sono molto contento dei risultati ottenuti, del rapporto instaurato con i miei compagni ed i significativi miglioramenti che tutto ciò ha portato alla mia persona e alle mie competenze.
Spero di poter presentare degnamente il gran lavoro svolto quando saremo convocati a discussione per poter chiudere in bellezza questo percorso che è stato ricco di fatica e scoraggiamenti, ma anche di miglioramenti e soddisfazioni.

Patrick Sbrighi

Questa è stata la prima volta in cui mi sono trovato a lavorare ad un progetto di queste dimensioni in team. Mi sono realmente reso conto del ruolo fondamentale che ha la collaborazione tra membri per ottenere un risultato finale e quanto siano importanti alcune fasi di progettazione che prima schivavo o svolgevo in modo superficiale, con poca attenzione ai dettagli. Infatti ho imparato che una approfondita analisi del problema, come quella che è stata fatta in questo caso, permette di agevolare di tantissimo il lavoro in singolo, potendo portare avanti le proprie soluzioni sapendo già cosa si andrà a fare dopo e come interagire con gli altri oggetti del progetto, evitando anche di dover continuamente mettersi d'accordo con gli altri membri del gruppo rischiando di dover cambiare le proprie soluzioni. Inoltre ho appreso come, soprattutto nei progetti di gruppo, utilizzare uno strumento di controllo versione come GIT sia fondamentale e veramente comodo, permettendo di mantenere versioni stabili del progetto mentre si continua a sviluppare.
All'interno di questo gruppo mi sono soprattutto occupato della realizzazione di tutto quello che riguarda il personaggio principale e delle interazioni con gli altri componenti del gioco e con l'utente. Inizialmente non è stato facilissimo per me capire come implementare le mie parti perchè spesso tendo a scoraggiarmi e perdere fiducia in me stesso quando non riesco a trovare una soluzione nell'immediato ad un problema nuovo. Credo però di aver svolto correttamente tutte le parti a me assegnate, anche quelle che mi hanno dato più problemi, come le collisioni del personaggio e il salto. Infine spero di essere migliorato come programmatore, ma la cosa principale che mi rimarrà di questo progetto sarà l'aver acquisito più sicurezza in me per risolvere problemi mai affrontati prima.

Miriam Sonaglia

Non avendo mai programmato in team, il progetto proposto mi ha sicuramente messa di fronte ad una sfida. Mi sono dovuta da subito interfacciare con altre persone e cooperare con loro al fine di una funzionale realizzazione finale del lavoro. Sicuramente è stata un'esperienza positiva poiché, per quanto, a volte, possa essere complicata e faticosa la coordinazione tra membri del gruppo, è utile avere uno scambio di idee e pareri che possano aiutare in momenti di difficoltà. Nel mio caso è stato di fondamentale importanza avere delle persone con cui confrontarmi e con cui chiarire i miei dubbi; avendo iniziato a programmare all'università ancora non sono ancora molto sicura delle mie capacità e, spesso, tendo a bloccarmi quando mi trovo di fronte a cose nuove o particolarmente impegnative.
Ho, inoltre, imparato a riconoscere ed apprezzare l'utilità di avere un'idea di progettazione e suddivisione dei compiti già da prima di iniziare a programmare, infatti, l'analisi a priori delle eventuali problematiche del lavoro ha velocizzato molto il lavoro che, se svolto senza precedente analisi, sarebbe stato molto più oneroso e difficile da implementare e gestire.
Questo progetto, infine, mi ha aiutata ad acquisire praticità con GIT che, inizialmente, mi sembrava molto complicato e intricato da usare ma che, in seguito, si è rivelato molto utile e funzionale al fine della realizzazione di un progetto di tale portata poiché permette ad ogni membro del gruppo di accedere ad una versione aggiornata del progetto ma, allo stesso tempo, anche di programmare in aree di lavoro separate senza andare ad intaccare in alcun modo il lavoro degli altri.
Per quanto inizialmente abbia riscontrato delle difficoltà, questo progetto mi ha aiutata ad acquisire maggiore sicurezza nello svolgere i miei compiti e lavorare in gruppo ha alleggerito molto le mie preoccupazioni poiché sapevo di poter fare affidamento su qualcuno.

Eraldo Tabaku

Il progetto realizzato mi ha dato modo di avere un primo approccio più serio all'ottica di sviluppo in team, a posteriori posso dire di aver preso coscienza dell'importanza di tutte le fasi della creazione di un progetto, nello specifico ho imparato che con una fase iniziale di analisi del dominio gestita in maniera corretta, si può affrontare in maniera molto più semplice e lineare la fase di sviluppo successiva. Io e il mio gruppo abbiamo gestito in maniera discreta tutte le fasi di progettazione, anche grazie a una buona comunicazione tra tutti i componenti, in particolare abbiamo gestito tutte le operazioni per step dandoci delle scadenze periodiche al termine delle quali abbiamo discusso i problemi riscontrati cercando di risolverli insieme nel modo più efficiente possibile. Per quanto riguarda la mia parte più nello specifico, mi rendo conto soprattutto nelle fasi iniziali ho faticato a trovare delle strategie funzionanti per affrontare i problemi riscontrati, soprattutto nel lato logico-implementativo. A progetto finito posso dire di essere soddisfatto del mio operato, considerando il fatto che prima di questo corso non avevo alcuna nozione riguardo la programmazione orientata agli oggetti. L'utilizzo della piattaforma GIT come strumento d'appoggio per il lavoro in team è stato cruciale e poterne approfondire le modalità di funzionamento sarà sicuramente un qualcosa che mi sarà molto utile anche nel futuro. Tutto sommato, dopo una prima fase iniziale in "salita", penso di aver svolto un bel lavoro nel limite delle mie conoscenze anche se, ribadendo il mio discorso iniziale, mi rendo conto che avrei potuto risparmiare molte ore che ho dedicate al bug fixing semplicemente effettuando una migliore valutazione del problema e delle modalità risolutive da intraprendere. In conclusione affermo che avrei voluto avere la possibilità di poter dedicare più ore "massime" al progetto per poter implementare tante altre migliorie alle classi dei nemici, di cui mi sono occupato nel dettaglio. Mi sarebbe piaciuto riuscire a lavorare di più alle collisioni dei nemici in modo da renderle più "precise", oltre che a migliorare la loro IA in maniera da permettere loro di poter adattare i pattern di movimento alle diverse situazioni; queste idee mi saranno sicuramente di spunto per la realizzazione di progetti personali futuri, magari anche con lo stesso gruppo essendomi trovato molto bene a lavorare con loro.

Guida Utente

L'interfaccia di gioco si apre con un menu che permette di scegliere tra quattro azioni disponibili:

- *START GAME* permette di iniziare la partita
- *MUSIC* permette di scegliere fra 3 delle tracce audio da riprodurre durante l'esperienza di gioco.
  Le musiche riportate sono estrapolate da un altro gioco uscito nel 2005 che aveva anch'esso come protagonista il nostro personaggio: "Super Princess Peach".
- *GUI SCALE* permette di selezionare 4 modificatori di scala di gioco che danno l'opportunità di ridimensionare la finestra facendo rimanere invariate tutte le proporzioni interne al mondo. Si parte dalla scala "Tiny", la più piccola, a "Large", la più grande.
- *EXIT* permette di uscire dal menu, chiudendo la finestra di gioco

Cliccando sul tasto *START GAME*, il gioco si avvia facendo cadere Peach dall'alto dello schermo; il suo scopo, e, di conseguenza, anche quello del player, sarà quello di arrivare alla torre alla fine del livello.
Per poter muovere il personaggio si dovranno premere, da tastiera, i pulsanti "A" e "D", che gli permetteranno di muoversi rispettivamente verso sinistra e verso destra.
Per saltare si dovrà utilizzare il pulsante "SPACE", cliccando il pulsante due volte, Peach avrà la possibilità di fare un salto a mezz'aria.
Nel cercare di raggiungere la fine del livello, Peach sarà ostacolata da vari nemici ma potrà ucciderli saltando loro sopra.

Durante il percorso, la nostra protagonista potrà usufruire di potenziamenti che la aiuteranno nella sua impresa: i power up; questi potranno essere ottenuti colpendo dal basso, saltando, il Lucky Block (il blocco giallo avente un "punto interrogativo" disegnato) e si presenteranno in tre forme;

- ***Fungo rosa***: aumenta la dimensione di Peach (nel caso sia nella forma piccola) permettendole di distruggere i blocchi di mattoni. Le permetterà anche di sopravvivere dopo aver subito un colpo nemico e, quindi, di tornare nella sua forma piccola al posto di perdere una vita.
- ***Fungo con cuore***: fa ottenere a Peach una vita aggiuntiva.

- ***Stella***: rende Peach invincibile per un determinato lasso di tempo. In questa forma, Peach sarà invulnerabile agli attacchi nemici e avrà la possibilità di ucciderli senza dover saltar loro sopra

Il Lucky Block potrebbe generare anche un ulteriore oggetto, una ***Moneta d'oro***. Ce ne sono 7 collezionabili in totale ma non danno a Peach alcun potere speciale.
Peach, a inizio partita, ha a disposizione un massimo di 3 vite, se le dovesse terminare tutte allora il gioco terminerà e verrà visualizzata una schermata di "Game Over".
Se la nostra protagonista riuscirà nell'impresa di arrivare fino alla fine del livello (toccare la cima della bandiera conferisce al personaggio un importante bonus nel punteggio) e raggiungere la torre toccandone il portone di ingresso, allora avrà vinto la partita e verrà mostrato il punteggio ottenuto lungo il livello.
Per tornare alla schermata del menu iniziale basterà cliccare il pulsante "ESC" in qualunque momento.