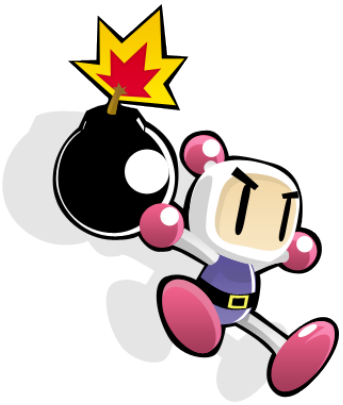
Progetto Saga: BomberMan

1) Il gioco

1.1) Bomberman



Il giocatore deve guidare Bomberman attraverso il suo sforzo di superare gli infiniti livelli del labirinto sotterraneo e risalire in superficie al fine di diventare umano. Per riuscirci il giocatore dovrà completare due obiettivi in ogni livello: distruggere tutti i nemici, e trovare l'uscita del livello, che sarà accessibile solamente una volta che tutti i nemici saranno sconfitti, solamente piazzando bombe. Oltre a questo, ogni piano contiene Power-Ups (potenziamenti) che possono migliorare le abilità di Bomberman, come la sua probabilità di sopravvivenza.
I potenziamenti possono essere raccolti per aumentare sia il numero di bombe che è possibile piazzare allo stesso momento, sia il raggio d'azione delle esplosioni, sia il numero di vite di Bomberman; sia aggiungere nuove abilità, come quella di poter passare oltre le bombe o calciarle vie.

1.2) I nemici

In questo remake ci sono cinque differenti nemici, ognuno diverso dall'altro: si muovono con velocità differenti, si spostano in modo differente, alcuni vagano senza meta, altri invece attraversano le pareti senza alcuno sforzo. Sono molto impegnativi da sconfiggere, ma una volta piazzata la bomba nel posto giusto il gioco è fatto!

Barom

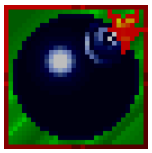
Onil

Minvo

Pontan

Maron

1.3) i Power-Up



Aumenta di una unità la quantità di bombe che possibile piazzare allo stesso tempo.

Permette a BomberMan di calciare le bombe lontano da sé. Un'ottima arma contro i nemici fluttuanti.

Permette a BomberMan di maneggiare le bombe con cura, ora può passarci sopra senza venirne ostacolato.

Diminuisce di una unità il raggio d'azione delle esplosioni.

Aumenta di una unità il raggio d'azione delle esplosioni.

BomberMan guadagna una vita extra.

La velocità di BomberMan diminuisce.

La velocità di BomberMan aumenta.

Il tempo a disposizione per completare il livello diminuisce di 30 secondi.

Il tempo a disposizione per completare il livello aumenta di 30 secondi.

2) Game Design

La struttura del gioco è principalmente suddivisa in due parti: la parte **logica** e la parte **grafica**.

2.1) Delegation pattern

Il componente principale del gioco è la classe *Bomberman.java,* questa ha il compito fondere i due meccanismi di gioco, aggiornando la parte logica, e mostrando, allo stesso tempo, a schermo la relativa parte grafica ad ogni fotogramma.
Le varie schermate di gioco si alternano e susseguono tra loro grazie all'uso del Delegation Pattern: la classe di gioco contiene i riferimenti a tutte le schermate che implementano l'interfaccia UIScreen, le quali hanno la responsabilità di aggiornare il loro stato ed intercettare l'input dell'utente da tastiera in modo da poterlo guidare attraverso le varie fasi di gioco. La classe Bomberman riceve tutti gli input (sia interni che esterni) e li delega alla UIScreen correntemente attiva; avremo così che ad ogni aggiornamento chiamato dalla classe PlayN i metodi init(), update(int delta) e paint(float alpha) saranno delegati alla corrispettiva schermata, che li delegherà a sua volta alla componente di gioco più appropriata.
Ogni azione che coinvolge passaggio da una schermata alla successiva è così gestito dalla classe principale, l'unica a possedere i riferimenti alle altre schermate ed i metodi di accesso alle istanze. Nel momento in cui il programma inizia, il gioco delega il caricamento di tutte le risorse alla classe loadingScreen; a caricamento terminato questa chiamerà un metodo della classe Bomberman che delegherà il controllo alla schermata iniziale di gioco, la quale a sua volta attenderà l'input dell'utente per cominciare una nuova partita. Quando l'input è fornito la schermata iniziale chiamerà a sua volta un altro metodo della classe Bomberman "startNewRound", che si occuperà della preparazione del nuovo round, delegando il controllo alla classe GameScreen. Il gioco può iniziare.

2.2) Scissione tra parte logica e grafica

Mentre le schermate di pausa, caricamento, vittoria, e sconfitta si occupano unicamente di mostrare semplici messaggi, la classe GameScreen rappresenta il punto di accesso alla logica del gioco vero e proprio.
Il loop di gioco è diviso in due parti: il sistema logico e l'interfaccia grafica sviluppata attraverso l'uso del Decorator Pattern; la parte logica si occupa di gestire per intero lo stato delle entità e del mondo di gioco, gestire i movimenti e le collisioni tra i corpi tramite l'uso della libreria di simulazione fisica open-source Jbox2D. La parte grafica incapsula i meccanismi logici ereditando, nel caso di classi istanziate programmaticamente, le classi stesse, e decorando con il Decorator Pattern quelle classi che sono invece istanziate a tempo di esecuzione (bombe, fiamme e power-up).
Estendere le classi logiche con sottoclassi contenenti metodi grafici è un'ottima strategia che permette di tenere sotto completo controllo le chiamate che si susseguono autonomamente nel mondo logico, catturarle e gestirle in quello grafico senza alterarle nel processo, e senza che la logica ne sia minimamente consapevole.
Purtroppo non è possibile applicare lo stesso metodo nel caso di entità che vengono istanziate dal mondo logico stesso a tempo di esecuzione. Non avendo nessuna possibilità di controllo su questo, l'unica alternativa è quella di usare un decoratore: nel momento in cui viene piazzata una bomba, ad esempio, il mondo logico si occupa di istanziarla e gestirla; per far si che

questo evento si perpetri fino al livello grafico bisogna far sì che il metodo (protetto) che si occupa della gestione della bomba e della relativa esplosione ("handleExplosion") ritorni, seppur non sia necessario ai fini della logica, la lista di entità logiche instanziate a per quella specifica chiamata. Ora, come precedentemente detto, il nostro LogicWorld è esteso da una classe decoratrice chiamata DecoratedLogicWorld che ha così piena facoltà di intercettare l'output di questo metodo eseguendo un Overriding dello stesso.
In questo modo abbiamo prelevato le entità generate, abbiamo instanziato dei decorator appropriati per ognuna di queste (FireDecorator, BombDecorator e PowerUpDecorator) e ceduto nuovamente il controllo alla parte logica. Ovviamente questo meccanismo è stato ripetuto per tutte le altri componenti come la gestione del terreno e delle entità.

2.3) Tipi di Entità

Nel gioco sono presenti vari tipi di entità:

- Entity: semplice entità astratta, presenta uno stato senza avere un corpo fisico
- DynamicPhysicsEntity: un'estensione di Entity che aggiunge le funzionalità dinamiche di jbox2d come il movimento nello spazio, ad esclusione della gestione delle collisioni
- DynamicPhysicsEntity e l'interfaccia PhysicsEntity.HasContactListener: questa interfaccia fornisce i metodi base per la gestione delle collisioni alle entità dinamiche.

2.4) Nemici

I nemici e la classe Enemy sono estensioni della classe DynamicPhysicsEntity ed implementano l'interfaccia HasContactListener per la gestione delle collisioni; ogni particolare nemico è poi esplicato nella specifica classe che ne definisce il tipo di movimento ed i peculiari dettagli come la velocità ed il comportamento in seguito alla collisione con corpi esterni.
I nemici si muovono sul terreno in maniera completamente autonoma, secondo lo schema dettato dall'interfaccia MoveStrategy che rappresenta lo Strategy Pattern dei tipi di movimento.
Per i cinque nemici sopra elencati sono state definite le seguinti strategie:

- Strategy1 descrive una strategia di movimento pseudo-casuale basata sull'esplorazione della mappa passando per le caselle libere senza possibilmente percorrere due volte la stessa via.
- Strategy2 descrive una strategia di movimento che permette ai nemici che la adottano di passare attraverso ogni tipo di muro mantenendo la stessa direzione per un numero fissato di caselle prima di svoltare, se possible, in una direzione prima non intrapresa.

2.5) Power-Up

I power-up e la classe PowerUp estendono la classe DynamicPhysiscsEntity, ed implementano un metodo specializzato "apply(Bomber)" il quale, presa un'entità di Bomberman in input, applicano su di esso gli effetti (positivi e non) dello specifico potenziamento.
I power-up sono contenuti all'interno delle caselle di gioco "BrickBlock", le quali una volta frantumante con una bomba, lo espongono all'esterno istanziandolo nel LogicWorld.
L'assegnazione di uno specifico power-up ad una singola cella avviene nel momento della generazione del livello: attraverso l'enumerazione PowerUpEnum (che contiene un

parametro in grado di estrarre un power-up per percentuale di probabilità) è possibile ottenere un riferimento all' enum del power-up scelto, per poi istanziarlo nel momento più opportuno attraverso il relativo metodo presente nella enumerazione.

2.6) Collisioni

Tutte le entità in gioco sono gestite dalla classe EntityEngine associata al LogicWorld. Questa classe contiene i metodi responsabili di aggiungere e rimuovere le entità dal motore logico, aggiornare il loro stato, e gestire le collisioni tra le quelle che implementano HasContactListener grazie all'interfaccia ContactListener di jbox2d.
Analogamente nella parte grafica questo meccanismo si riflette nella classe DecoratedEntityEngine, associata al DecoratedLogicWorld. Questa classe si allo stesso modo si occupa di delegare i metodi di aggiornamento e disegno alle singole entità decorate, le quali, controllate dal valore dell' Enum "State" delle relative entità logiche, assumeranno un aspetto grafico differente.

2.7) Modellazione grafica

La grafica di gioco si sviluppa su diversi livelli:

- il livello di sfondo
- il livello del terreno di gioco
- il livello in primo piano
- il livello degli Sprite e delle animazioni (che usufruiscono di accelerazione hardware)
- il livello in sovraimpressione

Questi sono raccolti in una classe con metodi statici che ha il compito di fungere da intermediaria tra tutte le componenti che desiderano disegnare a schermo: il LayerManager. L'aspetto e le animazioni delle varie entità sono invece gestite dalle classi Sprite e SpritePlayer. Uno Sprite rappresenta una singola animazione in tutti i suoi aspetti (numero di frame, grandezza, tempo di ogni frame, ciclo di riproduzione), mentre lo SpritePlayer è responsabile della corretta riproduzione degli Sprite a schermo: ad ogni update registra la quantità di tempo trascorsa dalla precedente chiamata ed aggiorna, se necessario, il fotogramma correntemente mostrato; è inoltre in grado di mandare in loop un'animazione.

2.8) Stringhe localizzate, costanti e risorse

Tutte le stringhe che fanno riferimento a risorse e file su disco sono situate all'esterno del codice, nella file assets/text/game.properties, Mentre le costati di gioco sono raccolte nella classe Constants.java. Tra le costanti troviamo una lista di tutte le risorse che è necessario caricare prima di iniziare un round, queste saranno prelevate dal DecoratedEntityEngine e inizializzate nei primi momenti subito successivi all'avvio.
I riferimenti alle risorse specifiche di ogni singola classe grafica sono salvati in campi privati statici di ogni decoratore, così da risparmiare memoria.

3) Schema delle classi

è possibile trovare lo schema delle classi all'interno del file .zip