Sommario

Analisi
1.1 Requisiti

L'applicazione Re:Dungeon prende ispirazione dal gioco di ruolo Pokémon Mistery Dungeon per creare un Dungeon Game con mappe generate proceduralmente ed un gameplay turn-based. Il gioco è composto da infiniti piani generati casualmente in cui il giocatore avrà il compito di trovare l'uscita sconfiggendo nemici e raccogliendo gemme per aumentare il proprio punteggio finale che verrà mostrato al termine del gioco.

Requisiti funzionali

- Il videogioco mostra un menu principale al lancio. Da questo menù sarà possibile selezionare la dimensione della finestra, selezionare una difficoltà (che modifica il numero massimo di nemici presenti contemporaneamente e la grandezza della mappa) e aggiustare il volume per la musica e gli effetti sonori.
  Al lancio del gioco verrà generato un primo piano casuale. Ogni piano contiene un'uscita che porterà il giocatore al piano successivo. Quando il giocatore sale di piano verrà generata una nuova mappa casuale. Il giocatore incontrerà diversi nemici su ogni piano, alcuni già presenti alla creazione altri generati ad ogni numero fissato di passi compiuti.
  I nemici aumenteranno di livello e di potenza con l'aumentare del livello di gioco e del giocatore.
  Il giocatore riceverà esperienza sconfiggendo i nemici e salendo di piano e raggiunto un certo ammontare di esperienza salirà di livello aumentando le proprie statistiche.
  Ogni quattro piani l'uscita sarà bloccata ed il giocatore dovrà raccogliere una chiave posizionata casualmente nel labirinto per poter salire di piano.
  Ogni cinque piani invece il giocatore si ritroverà su un piano speciale in cui dovrà affrontare un boss con meccaniche differenti rispetto allo scontro con i nemici comuni. Il gioco termina quando il giocatore muore ed alla fine verrà mostrato un punteggio che sarà tanto più elevato quanti più nemici sono stati sconfitti, quanti piani sono stati superati e quante gemme sono state raccolte.
- Sui piani potranno esserci alcune tile speciali che quando calpestate dal player sortiranno un particolare effetto. Queste tile possono essere: una pozione che riempie completamente la vita del personaggio, una gemma che fornisce un grande bonus al punteggio, la chiave per superare il piano nel caso in cui l'uscita sia chiusa o delle gemme finte totalmente identiche a quelle vere ma con effetti nefasti per il giocatore.
- Il giocatore potrà effettuare due tipi di attacco. Un colpo ravvicinato che colpisce il nemico di fronte a lui od una magia con utilizzi limitati per piano che danneggia gravemente tutti i nemici nelle vicinanze.
- Il giocatore potrà controllare le proprie statistiche, il numero di gemme in proprio possesso, il numero di magie disponibili ancora per quel piano e l'eventuale possesso della chiave tramite un Heads Up Display.

Requisiti non funzionali

- Il gioco dovrà rimanere fluido graficamente e generando gli elementi di gioco senza causare delay o bug.

1.2 Analisi e modello del dominio

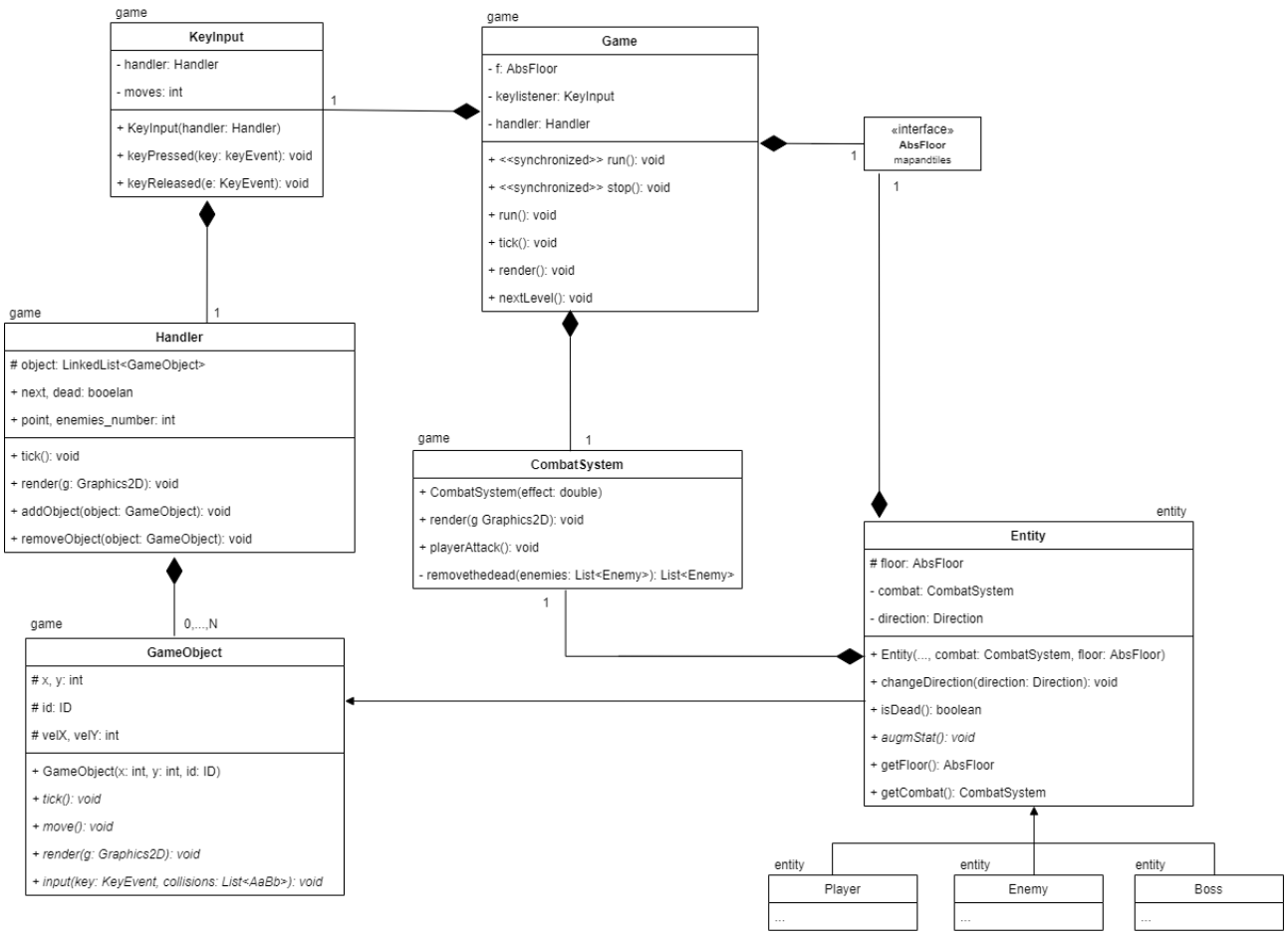

Figura 1) Struttura del dominio applicativo del gioco

Il modello del dominio applicativo del gioco è strutturato come mostrato in figura 1.
Iniziando il gioco dal menù verrà creato un nuovo Game. Un Game contiene il piano corrente, il personaggio principale ed una serie di osservatori che monitorano lo stato del gioco. Su un piano saranno presenti, oltre che al player, una serie di nemici e di oggetti/trappole. Il personaggio principale potrà attaccare i nemici che, quando sconfitti, forniranno esperienza al giocatore e verranno rimossi dal gioco. Il giocatore potrà raccogliere gli oggetti o attivare le trappole semplicemente camminando su di essi. Sarà inoltre presente una scala di uscita che porterà il giocatore al piano successivo quando ci si posizionerà sopra. Ogni cinque piani il giocatore affronterà il boss su un piano ridotto di dimensioni. Alla creazione il boss genererà una serie di attacchi magici che rimarranno sul terreno muovendosi casualmente ed infliggeranno grave danno al personaggio. Per sconfiggere il boss il giocatore dovrà raccogliere un certo numero di gemme presenti sul piano per indebolirlo. Sconfitto il boss, esso genererà un'uscita nella posizione in cui è morto e conferirà un ammontare di esperienza al personaggio. Sulla mappa il player non potrà muoversi dove sono presenti nemici o muri ma solo su tiles libere e contrassegnate come attraversabili. Ogni volta che il giocatore premerà un tasto valido da tastiera, l'osservatore keyinput muoverà consistentemente tutti gli elementi di gioco. Una delle difficoltà maggiori riscontrate sarà quella di far muovere consistentemente le unità in stile turn-based mantenendo un feel dinamico per il gioco.

Design
2.1 Architettura

Re:Dungeon è stato implementato utilizzando il pattern architetturale Entity-Component-System (ECS), pattern molto comune per lo sviluppo di videogiochi. Nel nostro caso ogni oggetto di gioco (o entity) è rappresentato tramite un'estensione della classe GameObject. I GameObject hanno un id che ne identifica il tipo specifico (e.g. Player, Floor, Enemy, Boss), un metodo per il rendering del loro sprite nella finestra di gioco, un metodo per ricevere input da tastiera, un metodo per l'eventuale movimento dell'oggetto ed uno per l'update di alcuni stati dell'oggetto a runtime (oltre a metodi e campi specifici che dipendono dal tipo di entità).
All'inizio del gioco e durante esso verranno create una serie di entità che necessitano di un solido sistema di controllo per garantire azioni consistenti ed una buona reattività del sistema agli eventi.
Per gestire tutte le entità create e le interazioni tra esse è stato implementato il pattern Observer. In particolare, Game possiede tre diversi Observer che prendono il ruolo dei Systems del modello:

- KeyInput che riceve input da tastiera e richiama, scorrendo una lista, il metodo input di tutti i GameObject facendo effettuare un'azione a tutte le entità ogni volta che il giocatore premerà un tasto valido.
- Handler che ha il compito di scorrere a runtime su tutte le entità controllando la presenza di condizioni particolari tramite tick() (per esempio se un'entità è morta o se il player si trova sull'uscita) e di istruire il motore grafico per disegnare l'entità correttamente tramite render.
- CombatSystem contiene riferimento a tutte le entity in grado di combattere e gestisce la funzionalità di combattimento, la vita delle entità ed il conferimento dell'esperienza.

Il gioco è quindi reattivo, visto che ad ogni input da tastiera corrisponde un'azione del personaggio principale e delle altre entità. Il gioco viene notificato a run-time dagli observer di cambi di stato come il passaggio di piano. Quando il gioco viene notificato del passaggio di piano sostituisce il piano corrente e ne genera uno nuovo tramite FloorFactory(). Utilizzando questa architettura è possibile aggiungere un numero enorme di nuovi oggetti di gioco, anche di tipologie diverse, semplicemente riutilizzando e adattando tramite composition gli oggetti già presenti.

2.2 Design dettagliato

Francesco Padovani:

Package Mapandtiles:

AbsFloor è un'interfaccia che realizza le funzioni che dovranno essere richiamate dalle altre entità del gioco di cui un'istanza dell'interfaccia AbsFloor sarà un component.
Essa viene implementata nelle classi Floor e BossFloor che definiscono rispettivamente un piano semplice ed il piano del boss.
Floor e BossFloor estendono anche GameObject di cui utilizzano il metodo render, chiamato da Game tramite l'Handler, per disegnare la mappa sulla schermata di gioco. La mappa viene rappresentata tramite il campo tilestate che consiste in una HashMap che associa ad un punto una Tile. Tile è la classe che rappresenta un singolo quadrato della mappa. Contiene un Point che identifica la posizione di una tile univocamente sulla mappa, una box AABB per fare in modo che le entità possano controllare la collisione con una tile non attraversabile, lo sprite che la rappresenterà sulla schermata di gioco ed un ID che identifica la tipologia di tile: attraversabile, non attraversabile, uscita ed altri tipi speciali che sortiscono un effetto quando il personaggio le calpesta.

AbsFloor:

L'interfaccia AbsFloor fornisce dei metodi utilizzati dalle altre classi. Tra di essi vi sono placeEntity() che viene utilizzato da Game per posizionare il player ed i nemici su di una tile libera appena quando un nuovo livello viene generato ed i metodi inerenti al sistema di movimento della telecamera.
Il sistema di movimento della telecamera è implementato tramite il metodo moveCam() chiamato dal Player che regola gli offset relativi alla posizione del giocatore per renderizzare le tile nella posizione corretta sulla schermata di gioco.

Floor:

Floor rappresenta un piano generico della mappa, che dovrà essere composto da un numero semi-casuale di stanze collegate da corridoi in modo che ogni stanza sia raggiungibile.
Floor viene costruito da Game passandogli come parametri il livello attuale del gioco, la grandezza della mappa e la grandezza della finestra di gioco tramite FloorFactory.
Alla creazione la classe Floor chiama il metodo floorGenner() per generare la mappa. FloorGenner() crea la mappa attraversabile chiamando roomsCreate() che tramite la classe di supporto Leaf() utilizza un algoritmo BSP per creare le stanze ed i corridoi.
L'algoritmo BSP è stato scelto per via della sua consistenza e del costo computazionale relativamente basso, il codice per l'esecuzione dell'algoritmo è stato adattato da quello presente al seguente link.
FloorGenner() successivamente crea tramite metodi privati di Floor le tile speciali come la tile pozione e la tile uscita. Infine, inizializza le tile non attraversabili e tramite l'algoritmo implementato nella classe Maputil ne determina la tipologia per assegnare loro lo sprite corretto.

BossFloor:

BossFloor è implementato analogamente a Floor con la differenza che, essendo un piano con configurazione fissa, non ha la necessità di generare una mappa casuale ma genera sempre una mappa rettangolare chiusa su tutti i lati. Un'ulteriore differenza sussiste nel metodo exitCreate() per generare l'uscita che viene chiamato dal Boss alla morte per creare un'uscita sulla posizione in cui quest'ultimo è stato sconfitto.

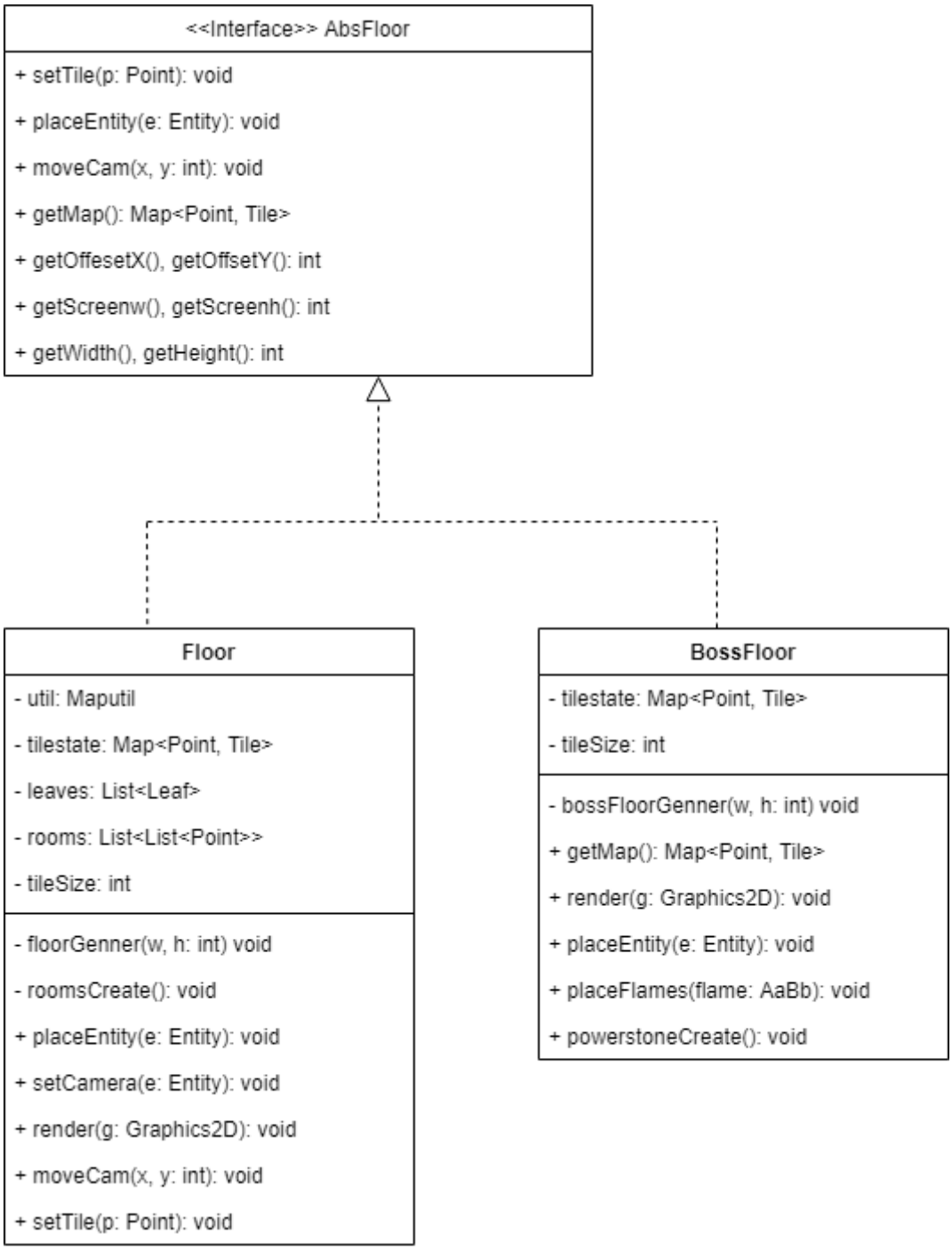

Figura 2) Struttura principale della parte di Francesco Padovani

Luigi Incarnato:

Package Entity e CombatSystem:

Entity è una classe che estende GameObject e rappresenta le entità dotate di una propria vita come il personaggio principale, i nemici ed il boss. Entity contiene tutti quei parametri e metodi comuni a tutti gli "esseri viventi" del gioco: un contatore degli hp, le statistiche di combattimento, una box AABB per la gestione delle collisioni, una componente Floor, i controlli della direzione ed il check per la morte tramite il metodo isDead che ritorna un boolean. Le Entity possiedono inoltre una SpriteSheet in cui sono presenti tutte le immagini che li rappresentano graficamente e permettono di eseguire animazioni. Le entity sono in grado di muoversi ogni qual volta viene registrato un input da tastiera tramite il metodo input ereditato da GameObject ed evocato tramite il KeyListener durante il gioco.

Enemy:

La Classe Enemy estende Entity e rappresenta i nemici comuni. Gli Enemy possiedono, oltre ai parametri comuni a tutte le Entity, una componente Player che permette loro di vedere le informazioni del personaggio principale. I nemici si muovono infatti cercando di raggiungere la posizione del giocatore ogni volta che esso esegue un'azione. I nemici controllano prima di muoversi se la posizione in cui stanno per andare sia valida o se il personaggio principale sia a distanza di attacco. Nel secondo caso, invece di compiere un movimento, attaccano il giocatore. I nemici vengono creati durante la partita tramite una EnemyFactory e collocati in una posizione valida tramite il metodo placeEntity di Floor. La resa grafica dei nemici viene gestita dall'Handler, gli attacchi eseguiti e subiti da CombatSystem ed il loro movimento da KeyInput. Tramite questi osservatori i nemici riescono ad avere un comportamento autonomo e vengono gestiti facilmente a prescindere dal numero. Quando i nemici vengono creati le loro statistiche di base vengono aumentate relativamente al piano in cui ci si trova e al livello del player tramite il metodo augmStat.

Boss:

Il Boss estende anch'esso Entity e si comporta in modo analogo ad Enemy. Il Boss possiede una lista di box AABB che rappresentano l'attacco magico. Quando il boss viene creato genera infatti un numero pseudocasuale di attacchi magici che vengono posizionati casualmente sulla mappa. Questi attacchi magici infliggono danno al giocatore quando le loro box e quella del player collidono. Il movimento di queste box è casuale di una posizione per volta ed il loro sprite viene renderizzato tramite il metodo render del boss che ne conosce la posizione.

CombatSystem:

CombatSystem è una classe del package game che ha il compito di monitorare le entità impegnate in combattimento. Ogni Entity ed il Game posseggono un CombatSystem. La funzione render di Game chiama la funzione render di CombatSystem per poter disegnare le animazioni degli attacchi quando eseguiti. CombatSystem contiene una lista di tutti i nemici, il boss (se presente) ed il player. Quando il giocatore o un nemico attaccano CombatSystem esegue le operazioni di assegnamento del danno, rispettivamente tramite i metodi playerDamage ed enemyDamage. CombatSystem rileva tramite il Player se esso sta eseguendo un attacco normale o l'attacco magico e tramite playerMagicAttack e playerAttack controlla se il giocatore stia colpendo un nemico. I metodi relativi al danno eseguono anche un check dopo l'assegnazione del danno per verificare se il player o il nemico siano morti. Nel caso in cui un nemico sia morto viene aumentata l'esperienza del personaggio. CombatSystem si occupa anche delle meccaniche dello scontro con il boss: assegna il danno sia del boss fisico che delle magie da lui create e diminuisce tramite lowerBossStats le statistiche del Boss quando il giocatore raccoglie una gemma del potere.

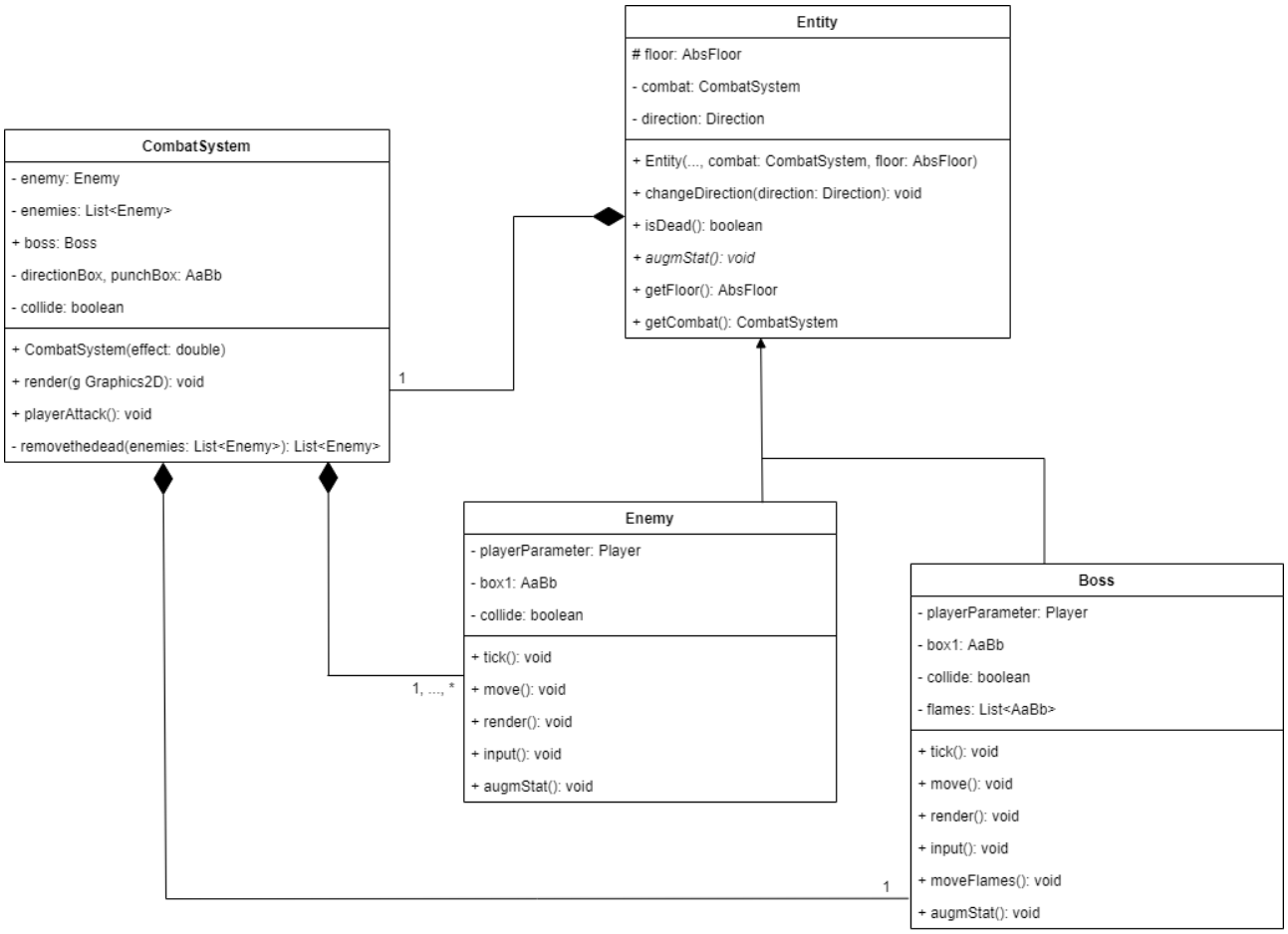

Figura 3) Struttura principale della parte di Luigi Incarnato

Matteo Vanni:

Package Entity e Game:

Come già precedentemente spiegato

Player:

La classe Player estende Entity e rappresenta appunto l'utente giocante. Il player oltre alle statistiche generate nel costruttore, dategli dalla classe padre, ha dei parametri per l'esperienza del giocatore, il numero di attacchi magici e un inventario che permette di tenere traccia delle gemme e delle chiavi ottenute, e degli attacchi magici restanti.

Per far muovere il player e le sue sprite il programma fa uso di *move()*, *render()*, *input()* e *tick()* ereditate da Entity

La funzione *move()* aiuta a muovere il personaggio, impostando le velocità in base alla direzione presa, facendo i controlli per le tile "speciali", quali scale, gemme, chiavi, pozioni e trappole, e sul muovere la telecamera

La funzione *render()* serve a disegnare la barra della salute del giocatore, scrivere il livello del player per tenerne traccia mentre si gioca e generare le barre di salute ed esperienza

Input() si occupa di controllare l'hitbox del player qualora si scontrasse contro un muro o un'entità dimodoché da farlo fermare invece che proseguire in quella direzione, in più serve per gestire la direzione in cui si muove il giocatore, per poter aggiornare la sprite, e l'inizio del combattimento generando le sprite del combattimento nella direzione in cui si sta guardando

La funzione *tick()* permette di generare le sprite del personaggio in base alla direzione prendendo la posizione vettoriale dalle spritesheet sorgenti

Ovviamente il giocatore avrà delle funzioni per poter impostare il numero di magie, il piano su cui è stato generato e l'esperienza.
Quest'ultima una volta raggiunto o superato il limite farà salire di livello il giocatore, *levelUp()* permette di gestire l'overflow dell'esperienza in modo che la sconfitta di un nemico potente porti al passaggio di più livelli;
ovviamente salendo di livello il giocatore ottiene un aumento sulle statistiche determinato da *augmStat(),* ereditato da Entity, che fa salire le statistiche in modo completamente casuale, e aumentando anche il numero di incantesimi possibili ogni 5 livelli, in modo da poter ottenere diverse situazioni di gioco e un certo livello di sfida

KeyInput:

Classe che estende la classe astratta KeyAdpter è utilizzata come listener sulle azioni dell'utente per l'aggiornamento di azione delle entità in gioco.
KeyInput comprende un handler per permettere di gestire gli input da tastiera e un contatore di passi per la generazione di nemici dopo un numero predefinito di passi fatti dal player, definiti dalla difficoltà impostata.
La classe ha due metodi che gestiscono la pressione e il rilascio dei tasti:
il primo *keyPressed()* controlla alla pressione del tasto se il player sta controllando l'inventario, si sta muovendoo sta attaccando dimodoché tutti gli enemy presenti in quel momento aggiornino le loro azioni di

conseguenza “inseguendo” il giocatore se si sta muovendo o attaccando, oppure attaccandolo se il player non si sta muovendo o se sono abbastanza vicini, permette inoltre di notificare all’handler le collisioni delle entità tra di loro o contro i muri;
la seconda funzione *keyReleased()* si attiva una volta che si è smesso di premere un tasto azzerando le velocità di tutte le entità e fermando ogni azione stessero facendo anche a causa della natura turn-based del gioco

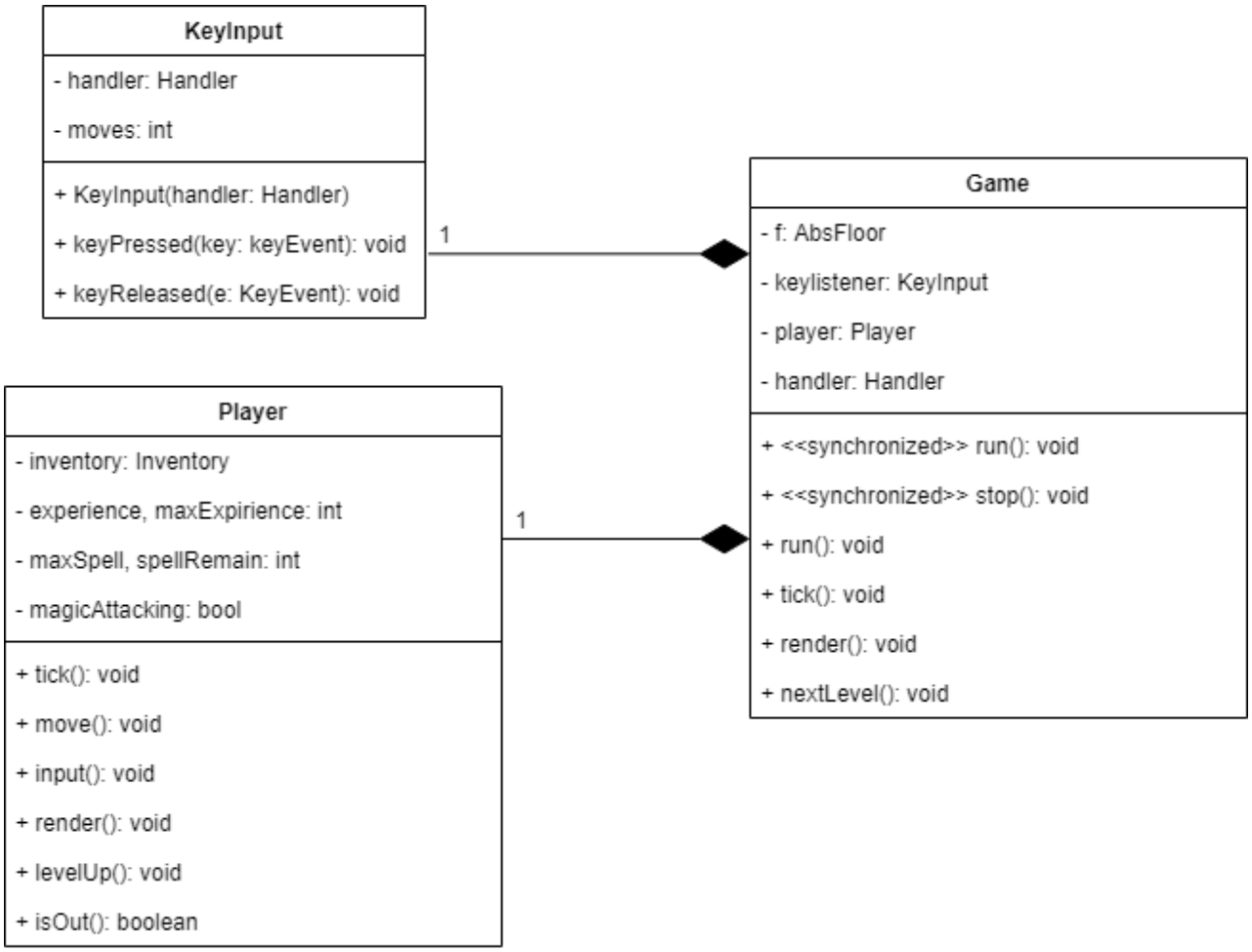

Figura 4) Struttura principale della parte di Matteo Vanni

Leroy Fabbri:

Package Menu:

Menu:

Menu rappresenta la schermata principale che viene visualizzata all'avvio del gioco. Si presenta inizialmente come una finestra di dimensione 960x760, al cui interno sono presenti 4 pulsanti: Inizia gioco, Tutorial, Opzioni e Esci.
"Inizia gioco" crea una nuova istanza di Game, che viene avviata appena si clicca sul pulsante;
"Tutorial" apre un nuovo pannello con all'interno la legenda dei comandi di gioco, al fine di conoscere i pulsanti per effettuare i tipi di attacchi e aprire l'HUD del gioco; "Opzioni" apre il pannello delle impostazioni, in cui si potrà cambiare la dimensione della finestra del menu e del gioco tramite un menu a tendina nelle seguenti dimensioni: "960x760","1280x720", "1440x900" e "1920x1080". Tutti i pulsanti e le immagini di sfondo si adatteranno al cambiamento della dimensione della finestra. Il secondo a menu a tendina, invece, regola la difficoltà del gioco, che consiste non solo nella variazione delle dimensioni della mappa di gioco, ma regola anche la quantità di nemici su un piano e la velocità di generazione di nuovi nemici nel piano. Le difficoltà sono rispettivamente: "Facile", "Normale" e "Difficile".

Nelle opzioni sono inoltre presenti due Slider, il primo per la regolazione del volume della musica, mentre il secondo regola il volume degli effetti sonori all'interno della partita. È possibile testare il volume degli effetti prima di entrare in partita con l'apposito pulsante "Suono" situato a sinistra dello slider. Infine è presente il pulsante per tornare alla schermata principale.

Hud:

Hud è una classe che estende GameObject e mostra varie informazioni riguardanti la partita attuale.
In partita, l'HUD è inizialmente nascosto, ma può essere reso visibile tenendo premuto il tasto Q.
Al rilascio del tasto, l'HUD scompare, in modo da non ostruire la vista della mappa di gioco al giocatore, e nascondere potenziali nemici alla vista.
All'interno dell'HUD sono presenti: il piano del dungeon che si sta affrontando al momento, contrassegnato dall'icona delle scale, una gemma, che rappresenta il numero di gemme raccolte durante la partita (le trappole, viste sulla mappa attraverso la stessa icona, non aumenteranno il counter delle gemme totali raccolte), la salute attuale e la salute massima del personaggio, contrassegnate dal cuore rosso, l'attacco e la difesa del personaggio, identificati rispettivamente da una spada e uno scudo. È presente anche un contatore che segna il numero di volte che si possono usare gli attacchi magici nel piano attuale.

ImagePanel:

ImagePanel è una classe che estende JPanel. La funzione principale, ImagePanel, crea una versione modificata di JPanel, che, chiedendo in input un'immagine, la mostrerà come sfondo. La funzione setImage prende in input l'immagine, che all'interno del menu viene ridimensionata all'interno della finestra.

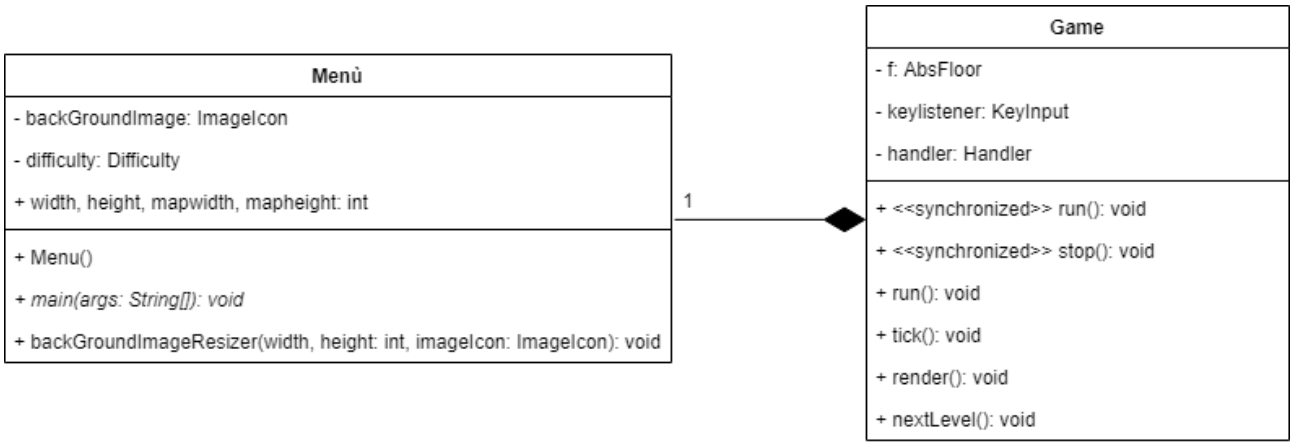

Figura 5) Struttura principale della parte di Leroy Fabbri

Sviluppo

3.1 Testing automatizzato

È stata realizzata una classe chiamata ReDungeonTest in cui vengono effettuati test automatizzati tramite JUnit.

- bossTest() verifica il corretto assegnamento delle statistiche e il corretto scaling del Boss.
- PlayerTest() verifica il corretto scaling delle statistiche del player, la loro corretta inizializzazione, il corretto funzionamento degli attacchi e del movimento tramite input da tastiera simulati, il corretto conteggio degli utilizzi dell'attacco magico, il corretto sistema di leveling tramite esperienza e la corretta assegnazione dello stato Dead.
- EnemyTest() verifica che le statistiche dei nemici scalino correttamente, che assegnino la giusta quantità di esperienza e che muoiano.
- CombatSystemTest() verifica il corretto funzionamento del sistema di combattimento, dell'assegnazione del danno sia ai nemici comuni che al boss.

3.2 Metodologia di lavoro

Il progetto è stato sviluppato utilizzando il DVCS Git ed il gruppo ha lavorato su di un repository condiviso da remoto.

Francesco Padovani:

Ho realizzato individualmente tutte le classi del package mapandtiles che si occupa della generazione e della gestione della mappa di gioco:

- AbsFloor
- Tile
- Leaf
- Floor
- BossFloor
- Maputil
- FloorFactory
- E le enum Corner e TileType

Ho inoltre realizzato la classe AABB per la realizzazione di una box per il controllo delle collisioni delle entità e la classe Inventory che implementa un sistema di inventario per il giocatore.

Luigi Incarnato:

Ho svolto la realizzazione di tutte le classi di Entity:

- Enum Attribute e Direction
- Boss
- EnemyFactory
- Enemy
- CombatSystem

Creata la classe resourceloader per il caricamente di immagini tramite inputstream, la casse buuferedImage per la gestione delle immagini e la classe spritesheet per la gestione di immagini sprite da cui ritagliare parti ben precise del file

Leroy Fabbri:

Ho realizzato le classi del package menu, che si occupa della gestione grafica del menu principale e dell'HUD di gioco:

- Menu
- ImagePanel
- Hud
- Enum Difficulty

Matteo Vanni:

Ho realizzato le classi del package entity, game e util, che si occupano rispettivamente della gestione del player, dell'input e dei font personalizzati utilizzati nel programma

- Player
- KeyInput
- Classe di utility per l'HUD: CustomFontUtil

Creata inoltre una funzione molto semplice che ridimensiona l'immagine di background nel menu in base alla risoluzione impostata

Classi svolte in comune:

- Game: Luigi Incarnato, Leroy Fabbri e Francesco Padovani
- Handler: Luigi Incarnato e Leroy Fabbri
- KeyInput: Matteo Vanni e Luigi Incarnato
- HUD: Leroy Fabbri e Luigi Incarnato

3.3 Note di sviluppo

Francesco Padovani:

Utilizzo di Stream per effettuare controlli in maputil, nel metodo removethedead di CombatSystem per lasciare nella lista solamente i nemici ancora vivi.
Utilizzo dell'algoritmo BSP per la generazione delle mappe casuali.

Luigi Incarnato:

Utilizzo di lambda expression nei foreach di liste in CombatSystem. Utilizzo delle librerie javax per l'esecuzione degli audio di gioco e di sistema. Scrittura ed uso della classe spreadsheet per il "ritaglio" di pezzi ben precisi da un'immagine data in input, preso dal web.

Leroy Fabbri:

Uilizzo di lambda expressions per i listener sul menu.

Matteo Vanni:

Utilizzo di lambda expression nel metodo move di Player e uso di stream e deriveFont in CustomFontUtil per poter impostare il font personalizzato in gioco.

Commenti finali

Il gioco nasce prendendo spunto da Pokèmon mistery dungeon di cui utilizza l'idea di un gameplay input-based per l'esplorazione di dungeon generati casualmente e il combattimento di nemici. La resa del gioco ci soddisfa in quanto pensiamo di essere riusciti ad emulare il funzionamento concettuale di base del gioco, seppur in scala di complessità più ridotta. Il gioco si presta ad ulteriori migliorie come l'introduzione di diversi personaggi giocabili, una maggiore varietà di nemici ed un sistema di itemizzazione complesso.

4.1 Autovalutazione e lavori futuri

Francesco Padovani:

Mi ritengo contento della realizzazione della mia parte del progetto. In particolare, sono molto soddisfatto della resa grafica e delle configurazioni della mappa. Penso che il game design sia un ambito estremamente stimolante e che ha definitivamente catturato il mio interesse. Sicuramente una parte che ha creato difficoltà è stata la realizzazione dell'architettura del software che ritengo non pulitissima, complice anche la decisione di utilizzare un pattern a noi estraneo fino alla realizzazione del progetto.

Luigi Incarnato:

Sono più che soddisfatto della parte assegnatami e svolta pur essendo, per me, la prima volta nell'attuale un progetto di questo calibro. Mi son dovuto interfacciare con nuove realtà ed anche difficoltà che pian piano venivano fuori durante la scrittura del codice, pur avendo fatto meeting premeditati insieme ai miei colleghi. Sono molto soddisfatto della grafica da me "assemblata" per gli "oggetti di gioco" che si visualizzano sulla mappa. Sono contento della resa dei nemici, come inseguimento e combattimento, non sicuramente con una IA elaborata, ma comunque riescono a svolgere bene il compito di "elemento di disturbo". Come altri miei colleghi non sono sicuramente troppo contento della non troppo pulita realizzazione dell'architettura del software dei pattern da noi utilizzati. Complessivamente sono soddisfatto della resa finale del progetto e di come, anche con alcune difficoltà, si interfaccino le varie classi tra di loro.

Leroy Fabbri:

Sono contento della sezione del progetto che mi è stata assegnata, nonostante sia il primo progetto di queste dimensioni a cui prendo parte. Ho trovato molto intrigante il fatto di dover affrontare problematiche nuove che dal lato "videogiocatore" vengono spesso trascurate e a cui si dà solitamente poca importanza. L'architettura del software non è il punto che preferisco del progetto, ma sono più che contento del risultato finale. Per quanto riguarda le difficoltà incontrate, ne ho riscontrate solamente nella progettazione iniziale per i layout dei menu.

Matteo Vanni:

Sono veramente soddisfatto di ciò che ho fatto, poiché ci sono state abbastanza difficoltà essendo che è stato il primo vero lavoro di gruppo svolto con coordinazione attraverso la piattaforma git hub.
Penso che la parte più complessa a livello logico sia stata, oltre la scelta delle parti più importanti da schematizzare, rendere equilibrato lo scaling tra i nemici e il player, dimodoché non risultasse impossibile l'avanzamento. Posso infine dire che sono veramente contento nel vedere una semplice idea, rivelatasi poi un intrigante sfida sia personale che per il team, prendere vita.

Appendice
Guida Utente

Vengono utilizzati i tasti

- WASD per muoversi
- ↑↓←→ (frecce direzionali) per cambiare direzione senza muoversi
- J per effettuare un attacco singolo corpo a corpo
- K per eseguire un attacco magico che infligge danno ad area intorno al personaggio (usi limitati per ogni piano)
- Q per aprire l'HUD che mostra il livello del piano, le statistiche del giocatore e la quantità di gemme raccolte
- Per raccogliere gli oggetti basterà che il giocatore passi sopra di essi

Il giocatore ha come obbiettivo l'esplorazione di un dungeon composto da infiniti piani. Il gioco termina alla morte del personaggio principale e mostrerà il punteggio ottenuto. Si otterranno punti salendo di piano, sconfiggendo i vari nemici sparsi per il dungeon e raccogliendo le gemme disseminate su ogni piano (attenzione! non tutto è ciò che sembra). Qualche volta l'uscita potrebbe essere bloccata e sarà necessario trovare una chiave.