Hollow Men

Relazione Progetto OOP 2015

Capacci Giulia
Cerami Alessia
Giordano Andrea
Giancola Pierluigi

Indice:

1 Analisi

1.1 Requisiti

Il software mira alla costruzione di un videogioco del tipo Action RPG in due dimensioni ambientato all'interno di un castello (Dungeon).
Per Action RPG si intende un videogioco nel quale l'utente controlla un personaggio che combatte contro dei nemici in maniera dinamica, ovvero non a turni.

Requisiti Concordati:

- Il videogioco si occuperà di far muovere l'utente tra le stanze dei vari piani che compongono il Dungeon, guidandolo verso nemici di potenza sempre maggiore.
  Ogni stanza, se contiene delle porte, è ben difesa dai nemici mentre, nelle altre, si potranno trovare tesori incustoditi.
  Le porte permettono all'utente di proseguire verso le stanze successive ma solo se sconfigge i nemici in quella corrente; inoltre solo una delle porte permetterà di continuare l'esplorazione.

- All'ultima stanza di un piano bisognerà affrontare un nemico molto potente per poter accedere al piano successivo

- Quando ogni piano è stato esplorato e conquistato il personaggio ha completato la sua missione

- Il videogioco permette all'utente di accumulare esperienza e oro sconfiggendo i nemici e prendendo i vari tesori; in seguito all'aumento di esperienza l'eroe acquisirà degli Stat Point (spendibili sulle statistiche) e degli Skill Point (spendibili sullo Skill Tree, spiegato in seguito)

- Il videogioco permetterà all'utente di accumulare ulteriore esperienza e oro completando alcune missioni (Achievement, spiegato in seguito)

- Il videogioco permetterà all'utente di scegliere una classe del personaggio che influirà sulle sue statistiche iniziali, sugli Achievement assegnati e sullo Skill Tree e sulle Spell

- L'esplorazione di un piano termina nel momento in cui :
  - Il personaggio termina i punti vita
  - Il tempo a disposizione per l'esplorazione termina
  - Il personaggio raggiunge il piano successivo

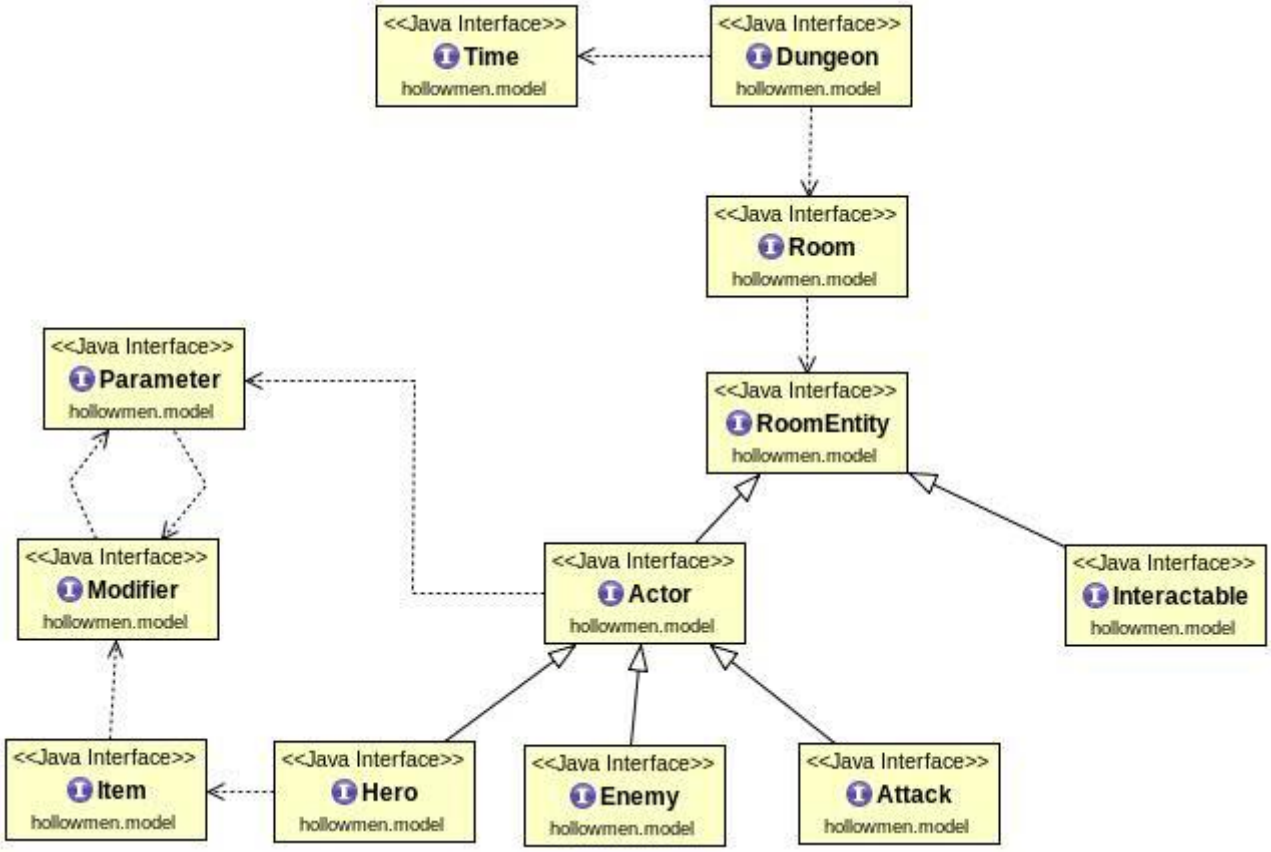
- Il videogioco permetterà all'utente di poter personalizzare il proprio personaggio:
  - equipaggiandolo dei diversi oggetti che troverà durante l'esplorazione o di quelli acquistati allo shop,
  - decidendo la classe del personaggio
  - decidendo come distribuire gli Stat Point e gli Skill Point

1.2 Analisi e modello del dominio

Il videogioco ha come elemento principale il Dungeon che dovrà accedere alle varie stanze (Room) che contengono le differenti RoomEntity.
Ogni RoomEntity può essere:

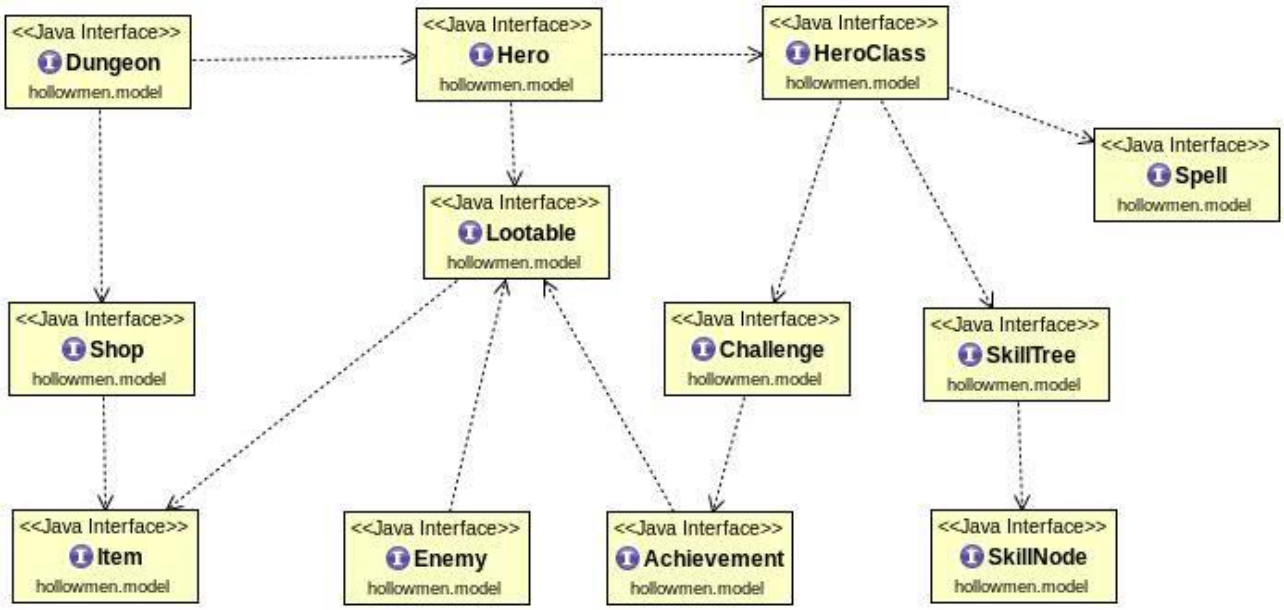
- un'entità interagibile (Interactable), ovvero che non si può muovere e con la quale l'utente può interagire muovendo l'Hero sopra esse; esempi: porte e tesori;
- un'entità attiva (Actor), ovvero un'entità che sarà in grado di poter combattere con le altre, inoltre si è voluto separare un Actor dagli attacchi per permettere la creazione di attacchi più elaborati.

Ogni Actor ha diversi parametri (Parameter). Per Parameter si intendono oggetti che hanno un valore base e devono tenere traccia delle modifiche apportate a tal valore. Le modifiche avvengono tramite oggetti Modifier che possono “attaccarsi” e “staccarsi” da uno specifico Parameter.
Gli Item utilizzano i Modifier per applicarli ai Parameter dell'Hero.
Il Dungeon dovrà tenere traccia anche del tempo che passa per notificare al giocatore un possibile game over per time out.



schema UML riassuntivo

Per l'aspetto che riguarda la personalizzazione dell'Hero.
Il Dungeon fornirà uno Shop dal quale l'Hero potrà acquistare Item con l'oro accumulato.
L'Hero sarà in grado di accumulare esperienza, oro e, con la giusta dose di fortuna, anche Item tramite oggetti Lootable che verranno generati da Enemy e Achievement.
La classe dell'Hero definisce lo SkillTree (ovvero l'insieme di SkillNode), le Challenge (insieme degli Achievement), gli oggetti che può equipaggiare, i Parameter di base e le Spell che potrà utilizzare.
Nella prima versione del software fornita ci si è concentrati maggiormente sul funzionamento base dell'applicazione tralasciando tutti gli aspetti riguardanti la HeroClass in quanto ogni suo aspetto sarebbe stato frutto di uno studio approfondito che avrebbe impattato pesantemente il monte ore.
Una delle sfide più impegnative sarà il rilevamento delle collisioni da parte del Dungeon per permettere le giuste interazioni tra le entità in gioco.

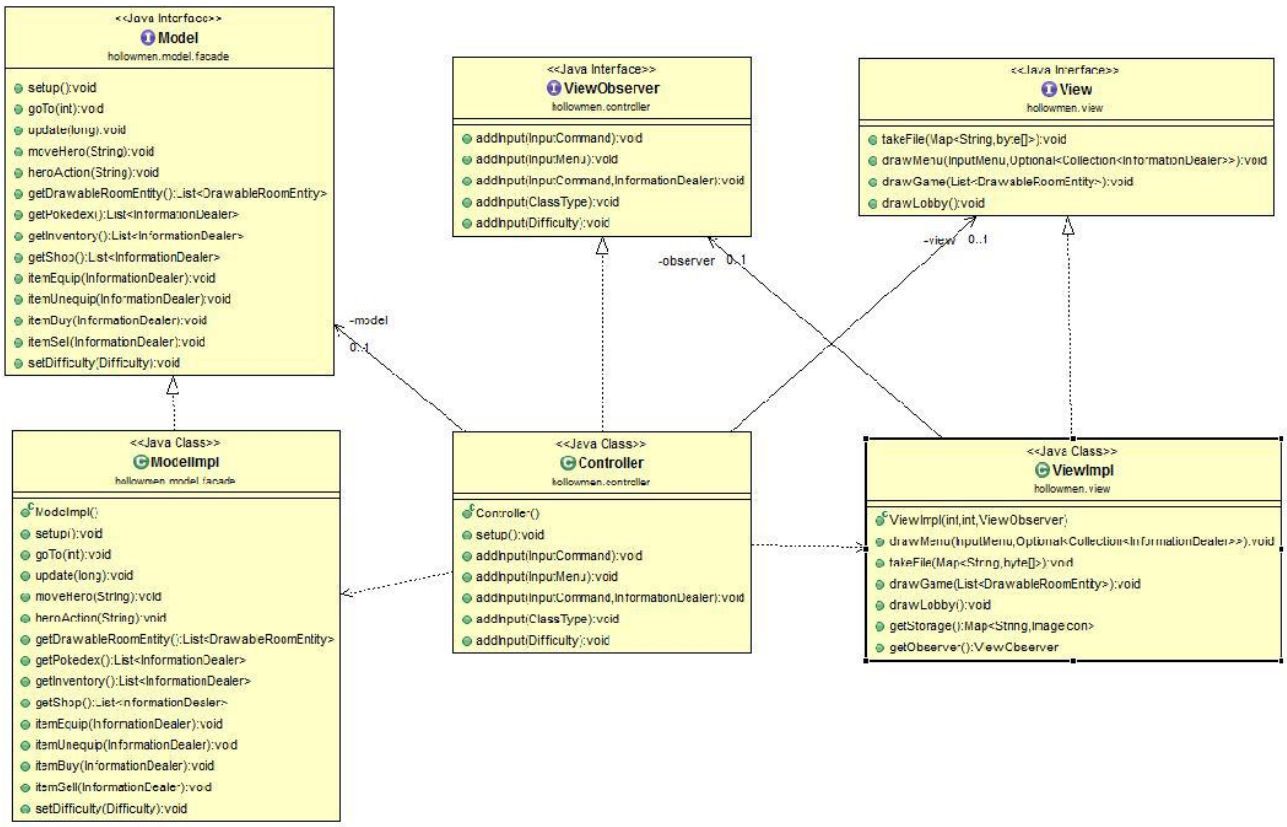


schema UML riassuntivo

2 Design

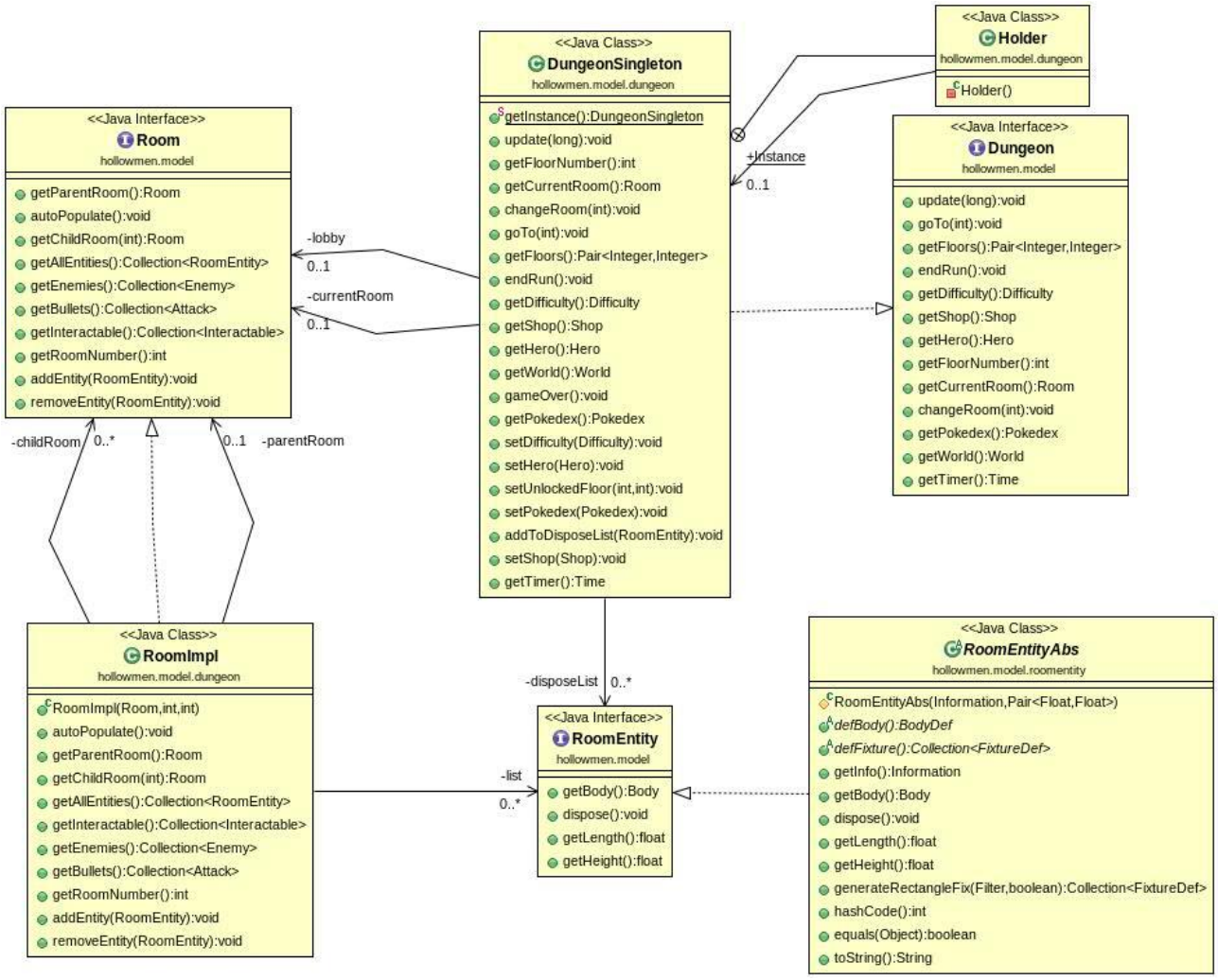
2.1 Architettura

Il design utilizzato è quello MVC (Model,View,Controller).
Questo tipo di design permette una netta suddivisione dell'operato in quanto le 3 parti lavorano su elementi distinti e indipendenti.
Il Model si occupa di definire i vari elementi di gioco e fornire dei metodi per modificarli, la View si occupa di gestire la rappresentazione grafica, la parte di interazione con il Controller e la parte dell'audio, mentre il Controller si pone come intermediario tra View e Model mappando gli input.
Ne segue che, la sostituzione della View con un'altra UI non causa alcuna modifica al Model e al Controller purché questa rispetti i vincoli dettati dalle interfacce.



2.2 Design dettagliato

Giancola Pierluigi



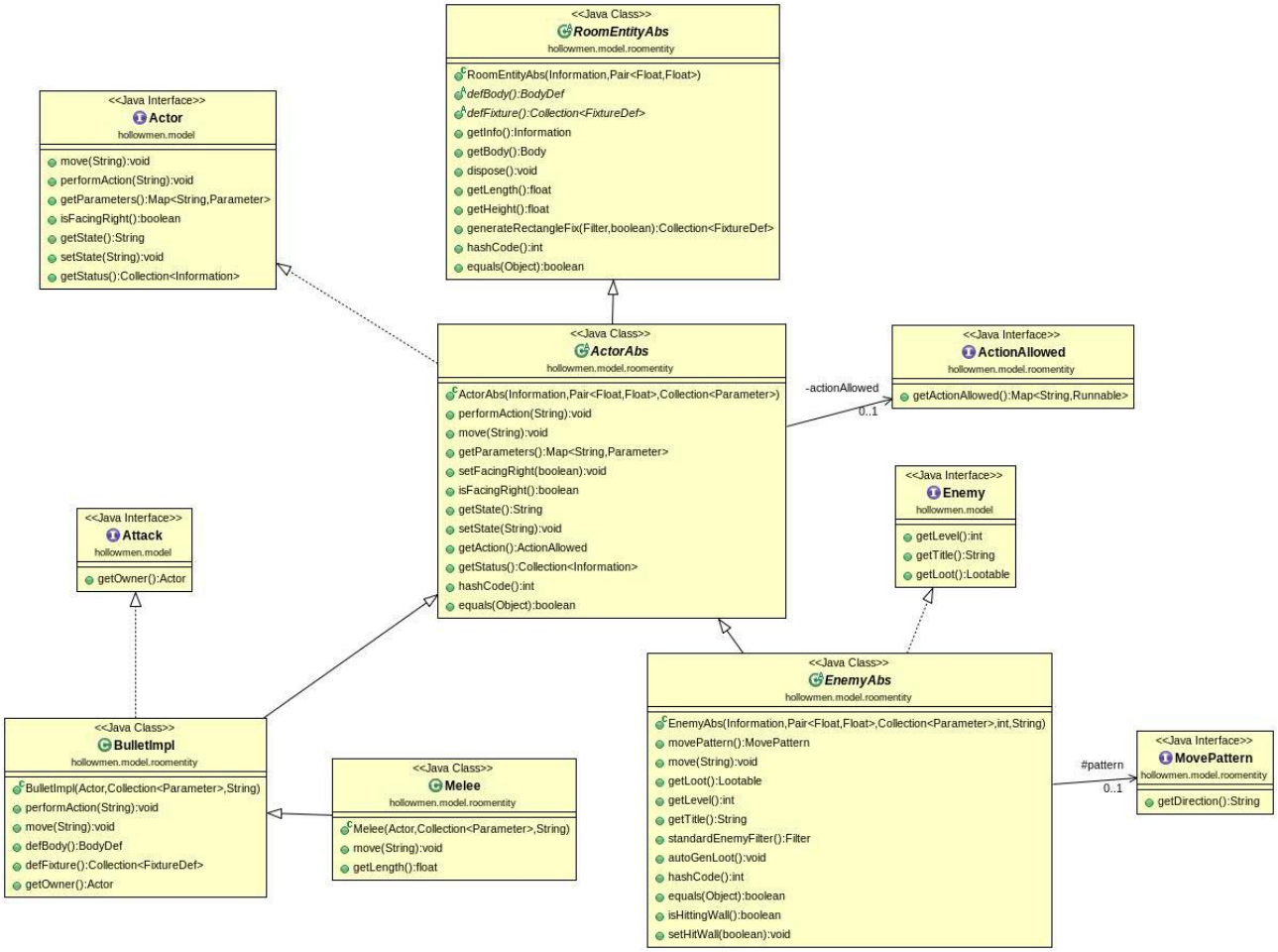
Ho deciso di implementare il Dungeon utilizzando il pattern Singleton, poichè non più di un Dungeon doveva essere inizializzato.
Inoltre la scelta di questo pattern ha permesso di rendere la classe raggiungibile in qualunque punto del codice semplificando notevolmente le interazioni all'interno del modello.
Non volendo lasciare dei setter sull'interfaccia e non potendo avere un costruttore con parametri, il DungeonSingleton fornisce dei setter per inizializzare alcune variabili di gioco.

Essendo un punto centrale del modello, ho affidato ad esso il compito di gestire le collisioni utilizzando una libreria esterna (Box2D, approfondito nel

punto 3.3 di questo documento).

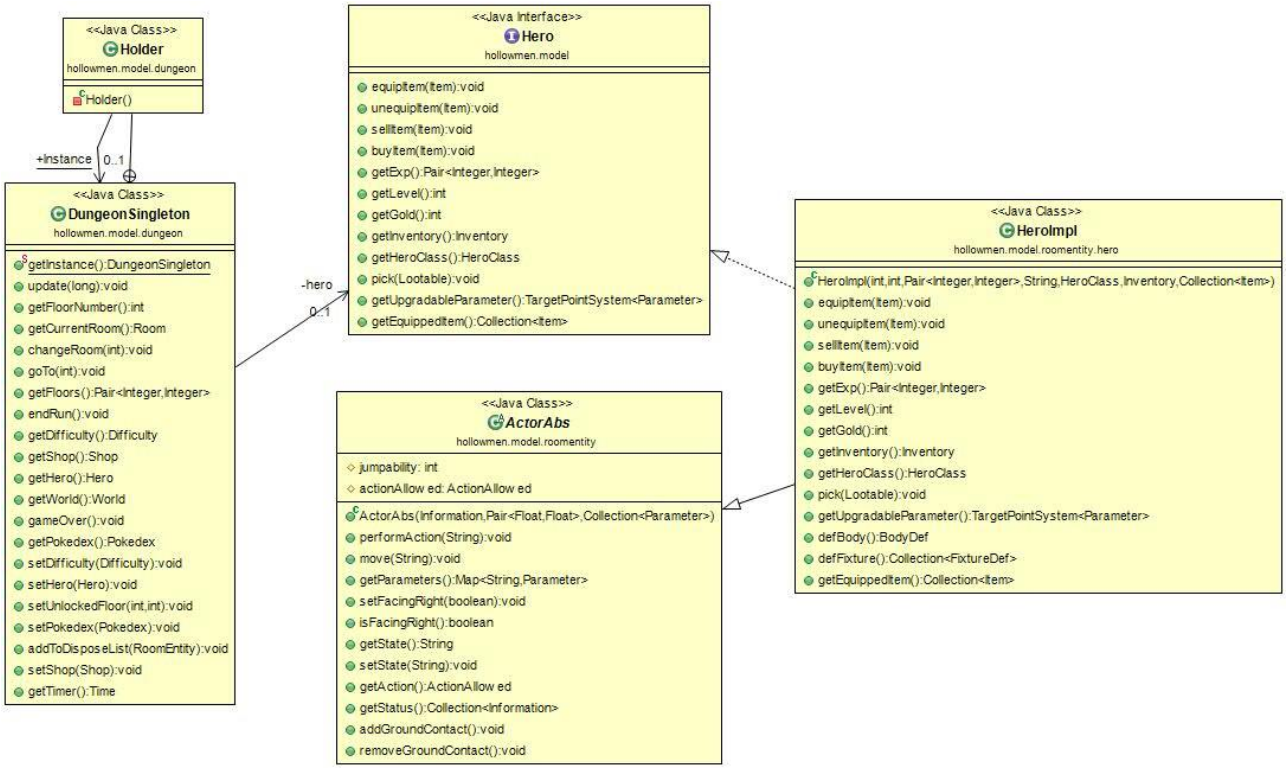
Le Room utilizzano il pattern Compose per tener traccia delle altre visitate in precedenza: questo permette al DungeonSingleton di doversi interfacciare sempre e soltanto ad un'unica Room esonerandolo dal doverne tener traccia in una mappa. Per questo motivo il DungeonSingleton ha da default una Room completamente vuota (Lobby) che serve per creare la prima stanza nel momento in cui si chiama il metodo goTo().

La RoomEntityAbs utilizza il pattern Template Constructor ovvero crea il Body (entità della libreria Box2D, approfondito nel punto 3.3 di questo documento) all'interno del costruttore utilizzando i due metodi astratti.

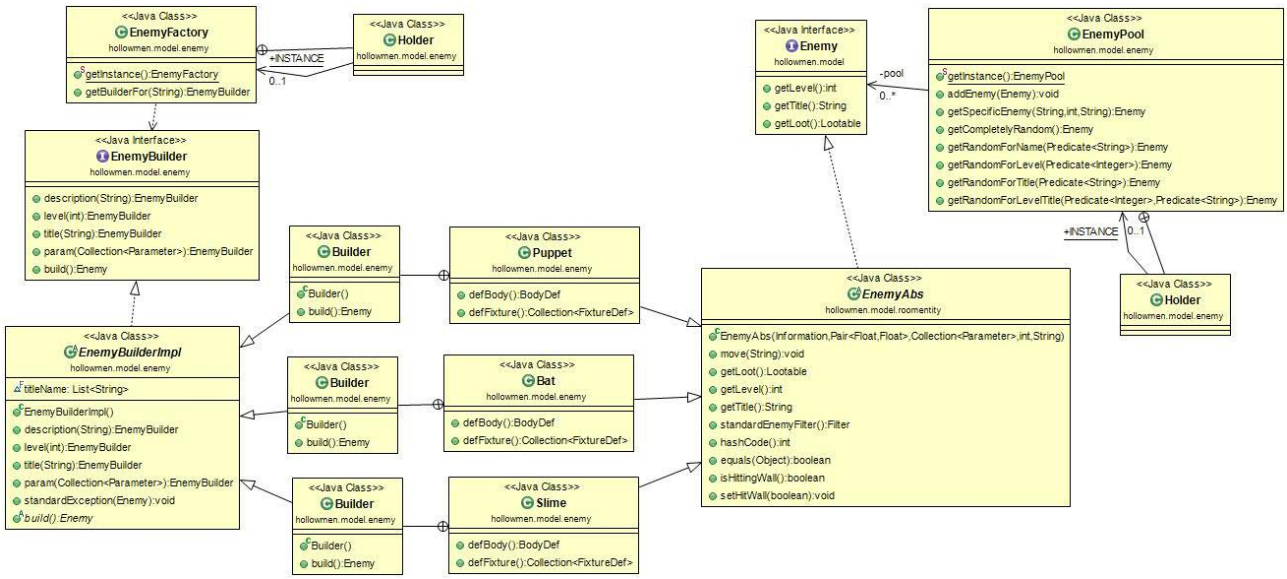


Schema UML riassuntivo degli Actor, degli Enemy e degli Attack, da notare l'utilizzo di due Strategy ActionAllowed e MovePattern.
Il primo serve per definire le operazioni da fare in base alla String presa in

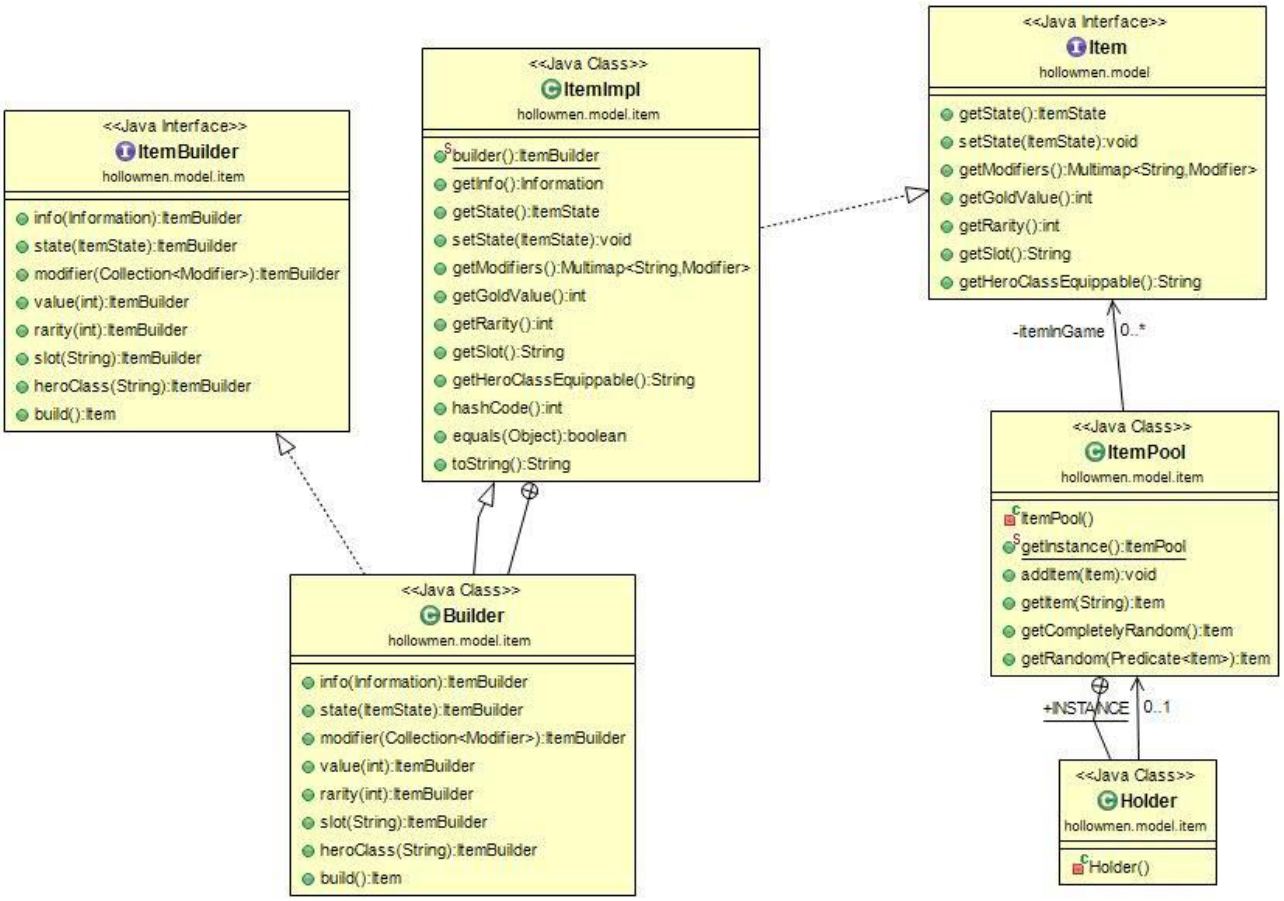
input dal metodo performAction(), il secondo serve a dare una direzione automatica degli Enemy.



Schema UML che mostra la classe dell'Hero, HeroImpl.



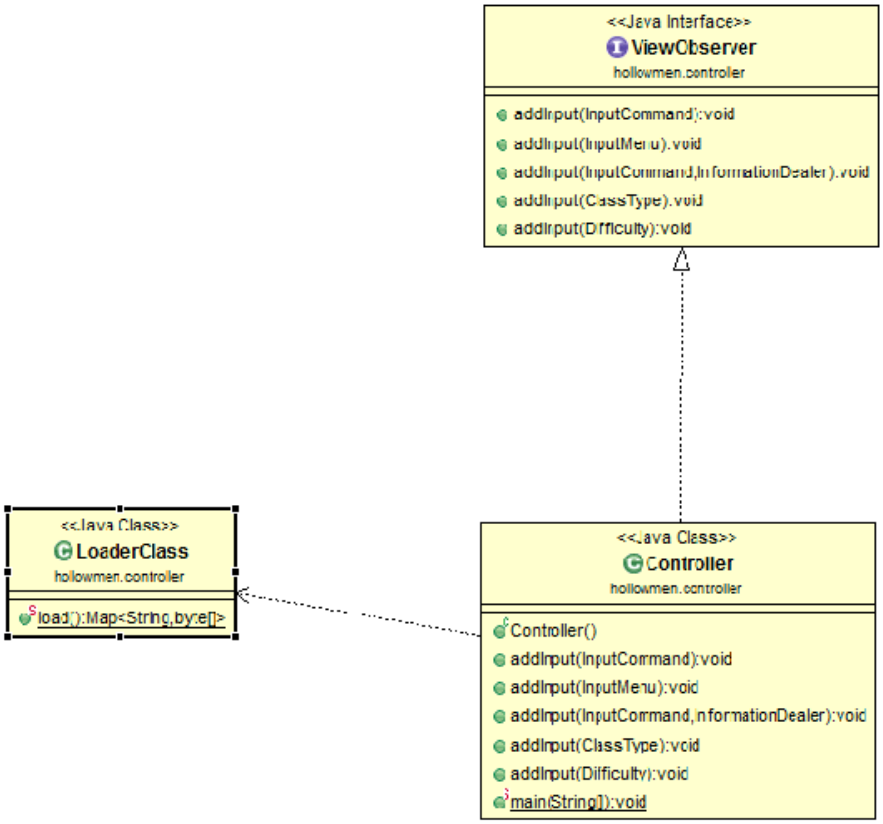
Schema UML che mostra i diversi tipi di Enemy implementati, da notare l'utilizzo del pattern Abstract Factory Builder in grado di tornare un EnemyBuilder relativo all'Enemy che si vuole creare specificandone il nome. EnemyPool mantiene le informazioni per costruire ogni Enemy.



Schema UML riassuntivo degli Item, ItemPool mantiene le informazioni per la costruzione degli Item

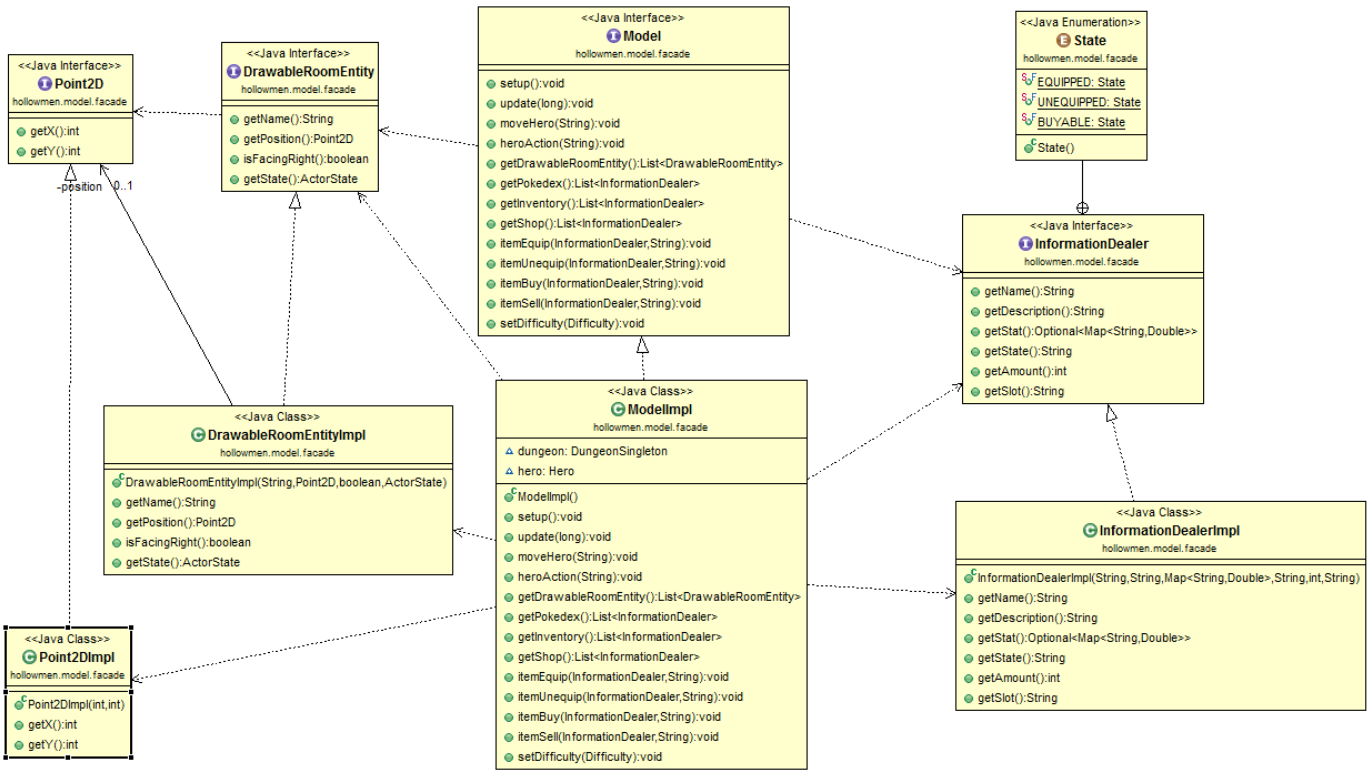
Giordano Andrea

Il controller si occupa della comunicazione fra Model e View, della gestione di I/O di files, Input dell'utente e del gameloop. Si sarebbe dovuto occupare della gestione dei files di salvataggio e settings, ma ciò non è stato possibile. La classe Controller è usata per il caricamento di tutte le immagini necessarie alla View.
Queste verranno caricate attraverso la LoaderClass e poi passate alla View. Inoltre la classe Controller è la quella in cui si trova il gameloop e il loop di lettura di input mentre ci si trova nel menù. Il loop di lettura input serve per leggere gli input per spostarsi tra i menù, il gameloop serve invece per far girare il gioco, andando a chiamare gli update del model e comunicando alla View cosa disegnare.



Questa è la parte di interfacciamento con il Model. Ci sono alcune interfacce che vengono poi utilizzate dalla View poiché hanno solo metodi getter (Point2D, DrawableRoomEntity, InformationDealer).

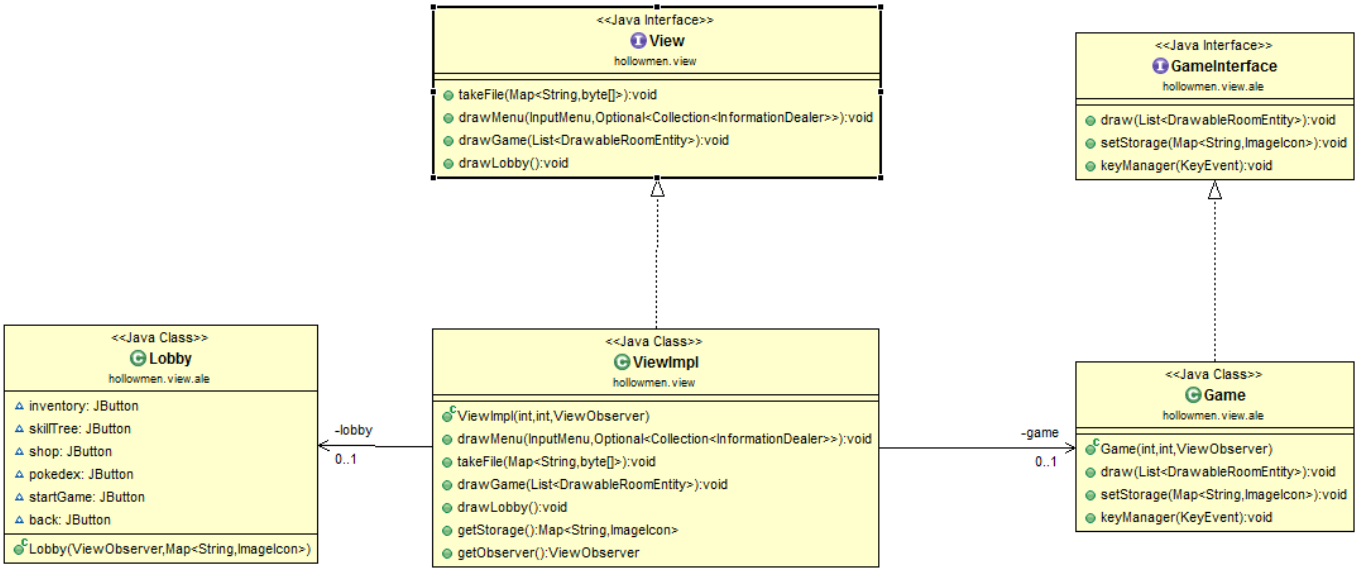
L'interfaccia Model contiene tutti i metodi necessari per la gestione degli input dell'utente e per estrarre informazioni dal model per passarle alla View.

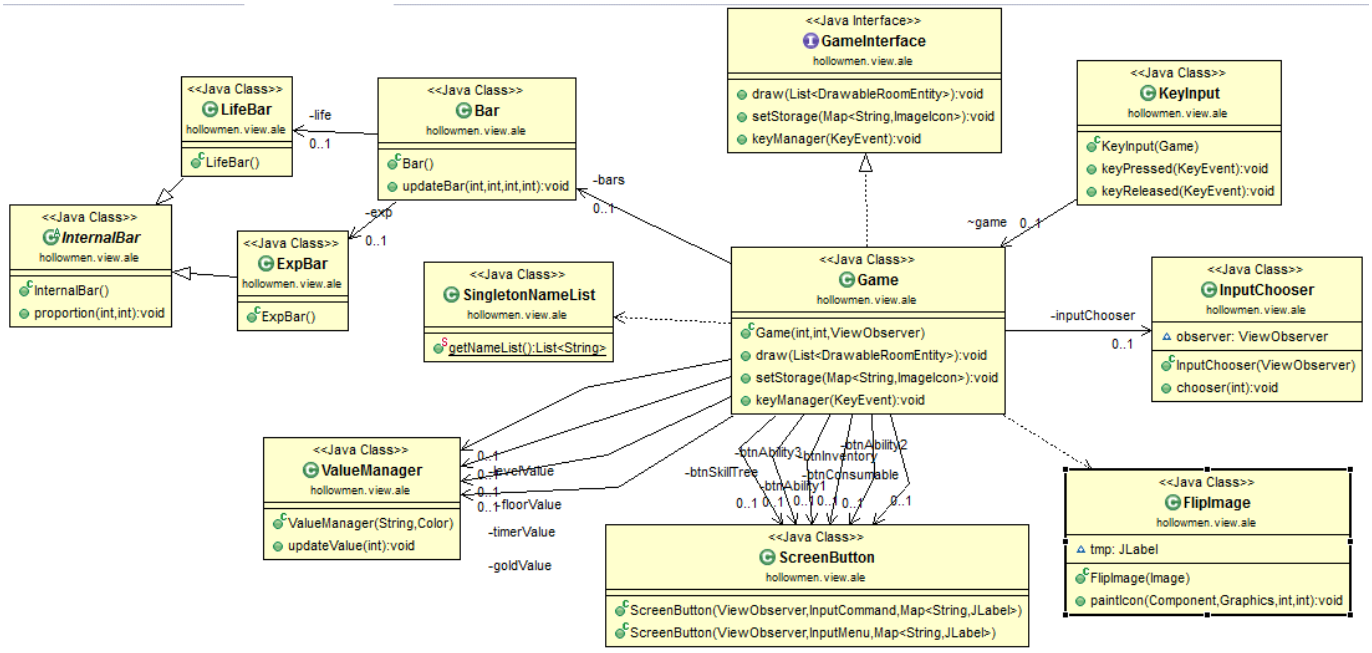


Pattern utilizzati:
-Singleton: Per la LoaderClass che viene usata per caricare le immagini;
-Observer: La classe Controller implementa l'interfaccia ViewObserver, così da mettere a disposizione della View alcuni metodi.
-Facade: L'interfaccia del Model con la quale il Controller comunica è una maschera per coprire il modello che sta sotto.

Cerami Alessia

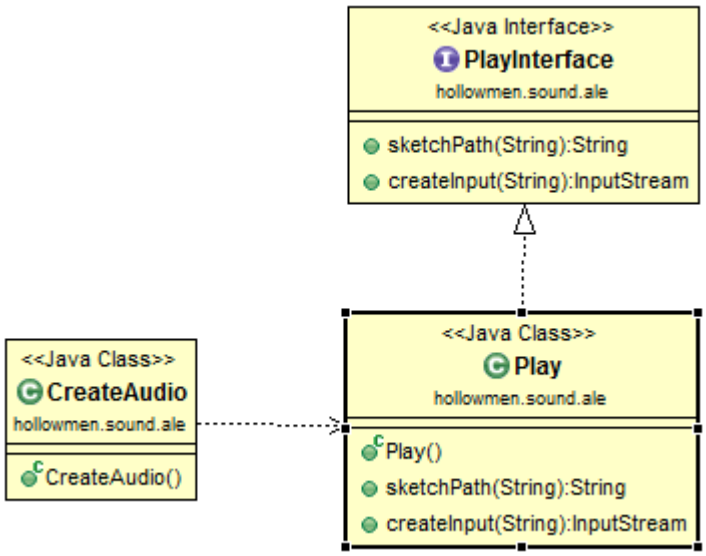
La View riceve dal Controller tutte le immagini necessarie a mostrare in fase di running il gioco. Tali immagini le vengono passate attraverso il metodo *takeFile(), all'interno della classe ViewImpl.* La View si occupa inoltre di notificare al Controller quali input sono stati premuti dall'utente nel momento in cui gioca.
Essa è stata suddivisa in più classi.
La classe *ViewImpl* implementa tutti i metodi che sono stati dichiarati nell'interfaccia View, dalla quale estende, e che permette la comunicazione fra View e Controller.
Tutti i componenti creati all'interno delle varie classi non hanno un layout in particolare ma vengono disposti settando manualmente la loro dimensione e la loro posizione all'interno del pannello a cui essi sono stati assegnati.





La classe *FlipImage* è stata usata per creare le immagini di verso opposto a quelle presenti nello storage da cui carico le immagini ad inizio partita.
La classe *SingletonNameList* è stata creata per contenere una lista unica delle immagini da mostrare durante il gioco.
Per quanto riguarda la lettura degli input ho utilizzato un mio KeyEventDispatcher tramite cui leggo un input utente che verrà poi notificato al Controller.

Pattern utilizzati:
- Singleton: Usato per la generazione di un'unica istanza della classe in cui è implementato (SingletonNameList, contiene i nomi delle immagini che utilizzo);
-Listener: Usato per "ascoltare" tutti gli input che il giocatore ha premuto e che verranno subito notificati al Controller attraverso il ViewObserver.

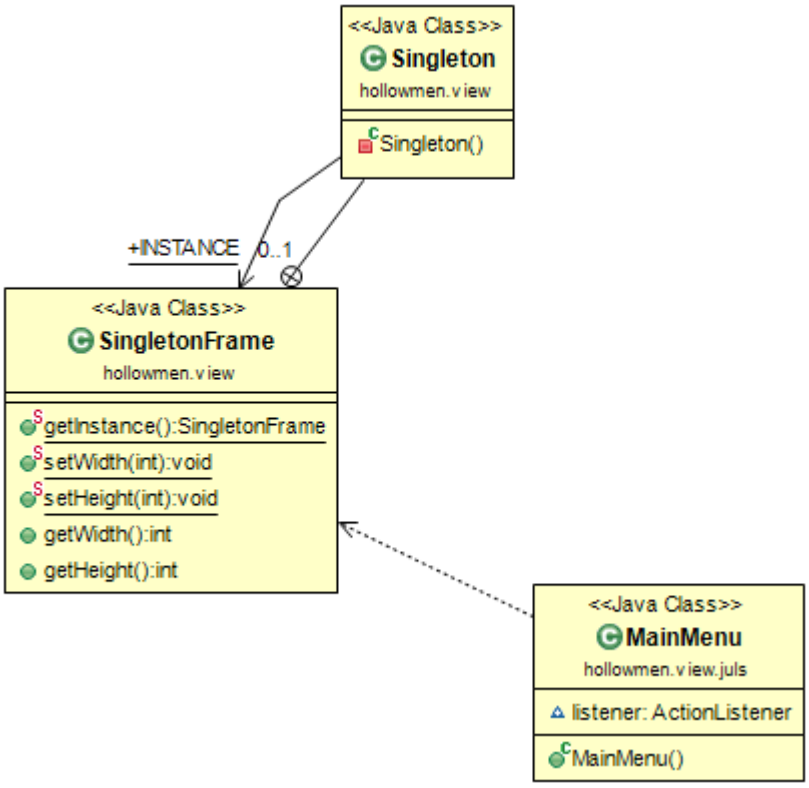


Il package *Sound* è costituito dalla classe Play che implementa l'interfaccia *PlayInterface()*.
Tale classe offre la possibilità di caricare un clip audio attraverso il metodo *createInput()* che ritorna un BufferedInputSream.
Permette anche di trasformare una stringa in una path assoluta attraverso il metodo *SketchPath()*.
Il clip audio creato viene poi usato all'interno della classe *CreateAudio()*, in cui attraverso il costruttore viene mandato in loop. Ciò avviene attraverso la chiamata al metodo *loop*().
(L'unico fallo della libreria è il gap di un secondo che intercorre tra la fine della canzone e l'inizio della stessa.)

Capacci Giulia

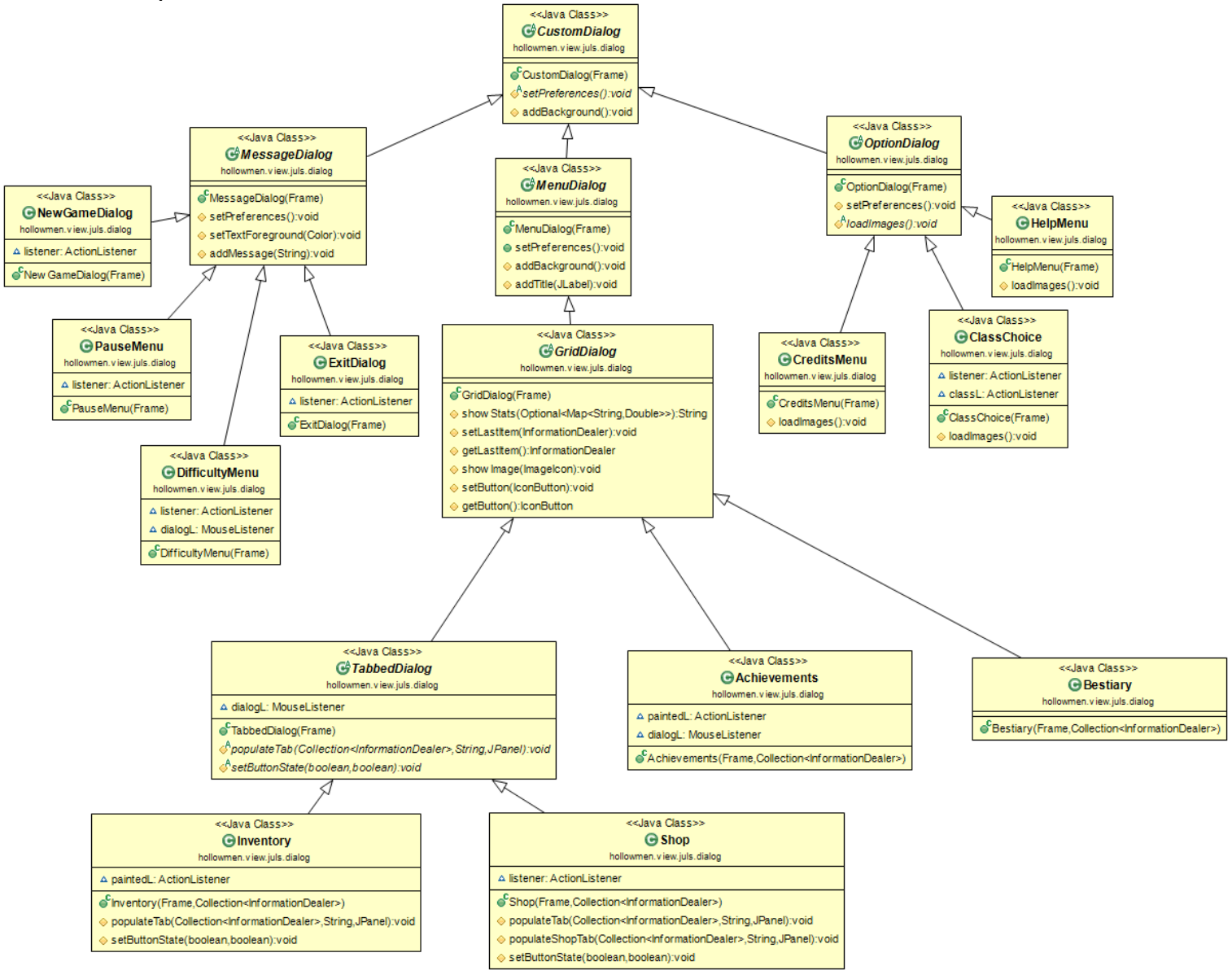
Poiché una delle esigenze del software era quella di disporre di un numero piuttosto elevato di menù, il primo scoglio da affrontare è stato quello della gestione di tutte le finestre. Fin da subito ho cestinato anche solo la considerazione di utilizzare più frames e ho provveduto alla creazione di una singola istanza di JFrame adottando il pattern Singleton. Il motivo principale di questa scelta risiede nella necessità (per ogni menù) di avere accesso allo stesso frame, infatti, se ne avessi utilizzati di diversi avrei dovuto allocare molte più risorse e sprecare memoria. Disponendo invece di una serie di JDialog personalizzate facenti riferimento all'unico frame disponibile evito

questo inconveniente e rendo l'applicazione più "user friendly" e facilmente navigabile. Nello specifico, la realizzazione del Singleton è stata basata sul principio della "lazy initialization", quindi la sua creazione viene messa in atto solo al momento del primo utilizzo. Inizialmente il pensiero era quello di creare una versione thread-safe tramite la parola chiave "synchronized" ma poi, essendomi resa conto della sua inefficienza (= overhead alle successive chiamate del getter), ho optato per una sincronizzazione implicita: nella classe che implementa il singleton è inclusa una classe-contenitore avente, come attributo statico, un'istanza del singleton stesso che fa sì che il primo accesso a tale attributo venga effettuato durante l'inizializzazione della classe-contenitore, e quindi sempre in modo serializzato.

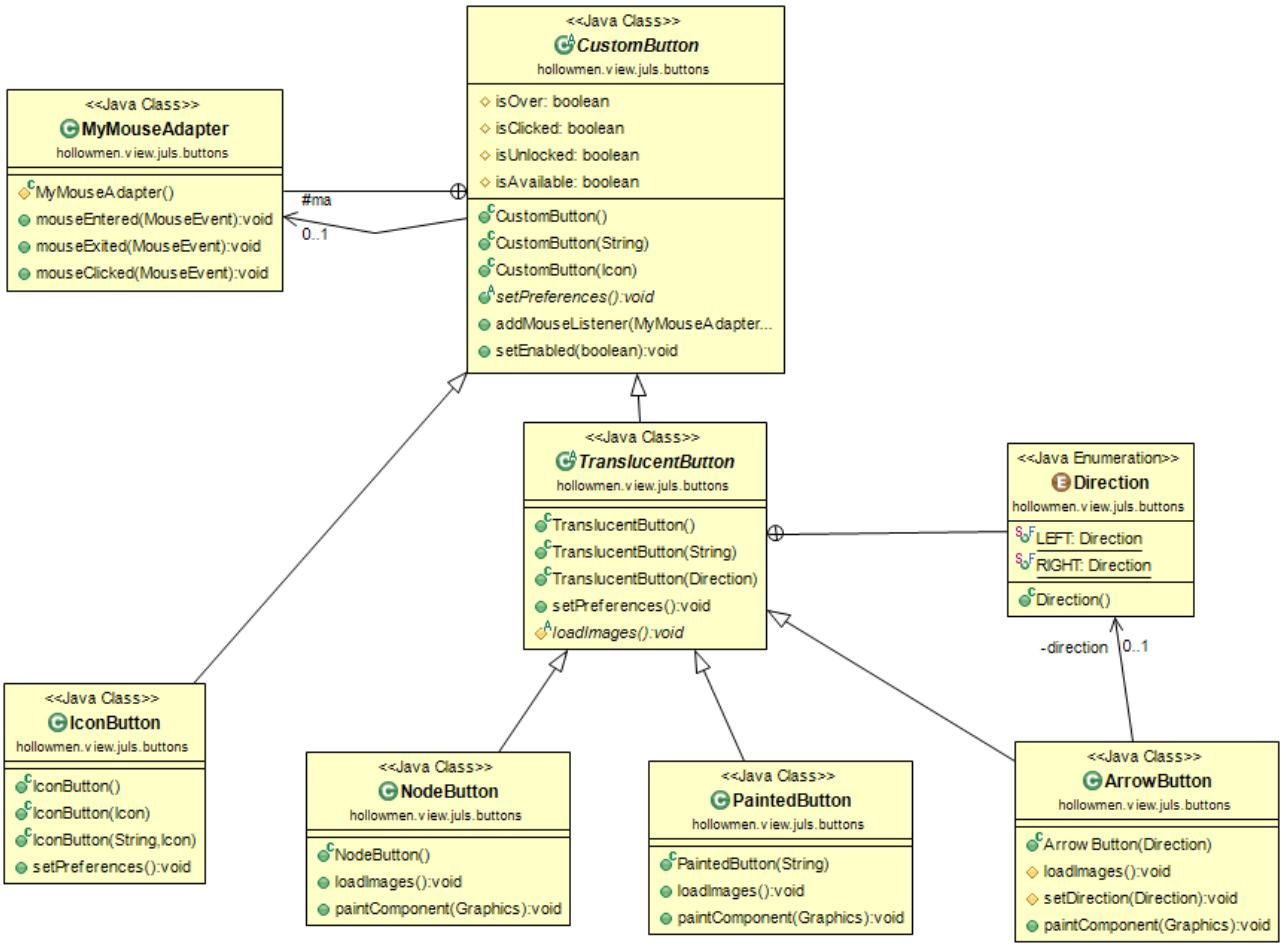


Per la creazione dei diversi menù di gioco, ho deciso di “sfruttare” l’ereditarietà della programmazione ad oggetti riutilizzando in modo vantaggioso il codice: tramite una gerarchia di classi astratte e concrete posso costruire una qualsiasi nuova finestra basandomi sulle caratteristiche che essa deve avere, quindi anche nel futuro sarà possibile aggiungere un nuovo menù semplicemente estendendo una delle classi astratte di sotto rappresentate.
Tutte le “foglie” di questo albero/UML corrispondono ad uno dei menù.
Più nello specifico, la differenziazione al primo livello:

- MessageDialog: rappresenta un menù semplice, di piccole dimensioni, dotato di pochi bottoni ed un testo;
- MenuDialog: rappresenta un menù più complesso che permette all’utente di compiere scelte importanti al livello di “utilizzo del software”, più precisamente permette di effettuare operazioni sugli elementi caratteristici del gioco (items, mobs, etc) che andranno poi ad influire sul proseguimento della campagna;
- OptionDialog: rappresenta un menù di medie dimensioni costituito prevalentemente da informazioni consultabili dall’utente.



Anche per i bottoni ho adottato la stessa strategia:



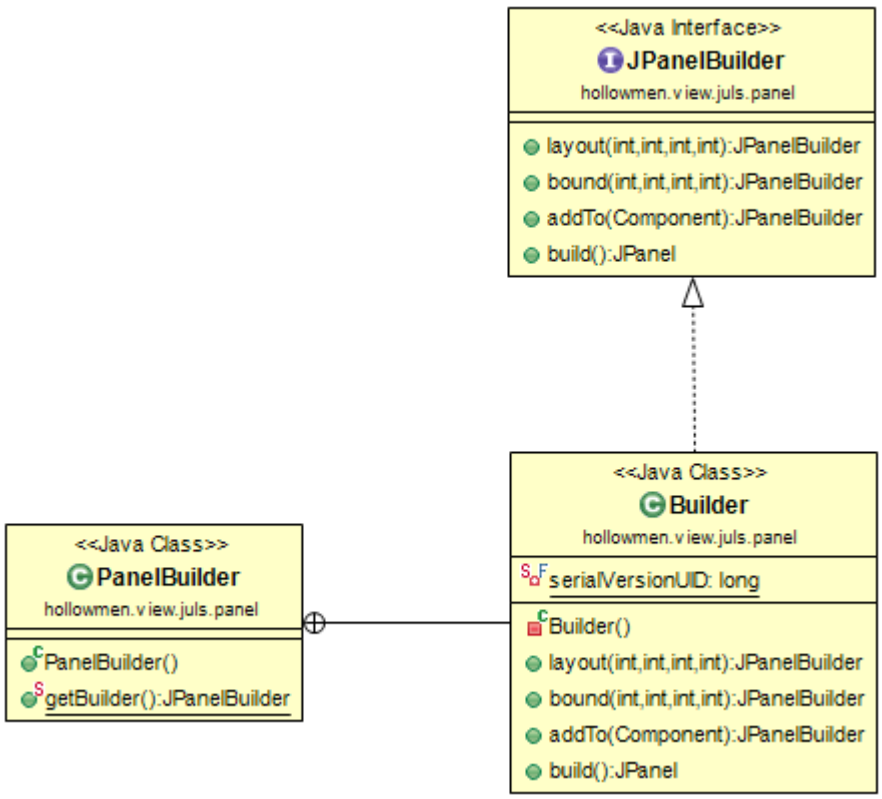
I bottoni personalizzati si differenziano per tipologia e possono essere chiamati in qualsiasi punto del codice.

- PaintedButton: rappresenta il bottone principale dei menù (più o meno l'equivalente del classico JButton, ma con un look&feel decisamente più accattivante), è privo di opacità, quindi permette di intravedere lo sfondo posto dietro ad esso;
- ArrowButton: rappresenta la classica "freccia"; per il momento sono disponibili solo quella destra e quella sinistra, ma in futuro potranno esserne aggiunte altre in base alle esigenze e con pochissimo sforzo;
- NodeButton: rappresenta il bottone classico per uno SkillTree. La classe al momento è presente, ma poiché non è stato possibile implementare

il menù non viene utilizzata (anche se le probabilità che venga aggiunto in futuro sono alte).

- IconButton: rappresenta il bottone con icona, viene utilizzato soprattutto all'interno di Inventario, Shop e Bestiario.

Procedendo nell'implementazione, mi sono accorta di dover disporre di un numero elevato di JPanel aventi tutti delle caratteristiche comuni (layout, opacità, principio di collocazione), così è sorta l'idea di utilizzare un Builder Pattern per la creazione di questo tipo di oggetti:



L'interfaccia JPanelBuilder contiene i metodi che dovranno essere implementati dalla classe innestata Builder contenuta nella classe PanelBuilder. E' possibile ottenere uno di questi oggetti semplicemente chiamando per primo il metodo getBuilder() sulla classe PanelBuilder e per ultimo il metodo build().

3 Sviluppo

3.1 Testing

Giancola Pierluigi

Sono state sottoposte a Testing:

- Le classi Pool (Enemy e Item) e il Cloner; nello specifico si è guardato se un oggetto aggiunto quando richiesto veniva effettivamente clonato;
- Le classi Builder (Enemy e Item); nello specifico si sono controllate le diverse eccezioni che lanciano al momento di input non validi;
- I metodi operazionali dell'Hero sugli Item; è stato testato il corretto indirizzamento dei modifier presenti negli Item sui parameter dell'Hero e il loro effettivo valore finale;
- La registrazione di eventi sulla classe TimerSingleton; è stato testato la corretta scadenza e applicazione delle funzioni Consumer, tuttavia durante la fase di testing manuale si è scoperto un bug che il test non aveva trovato, risolto in seguito.

Sono stati effettuati Test Manuali per:

- Corretto movimento delle varie Entità all'interno delle stanze, corrette collisioni tra esse, non automatizzati perché l'effettiva posizione di un'entità all'interno della stanza dipendeva da calcoli fatti dalla libreria esterna Box2D, così come le collisioni.
  Tuttavia si sono effettuati test preliminari al fine di capire le potenzialità della libreria.

Giordano Andrea

Per quanto mi riguarda, non sono riuscito a fare testing automatizzato poiché la mia parte non riguardava calcoli aritmetici ma gestione dell'applicazione. La mia forma di testing è stata quella di far provare ai membri del gruppo l'applicazione in modo da notare un'anomalia nel funzionamento. Quando questa veniva riscontrata, mi veniva riferito e io provvedevo a cercare una soluzione al problema.

Cerami Alessia

Per quanto mi riguarda, non ho fatto dei testing automatizzati, ma per testare il corretto funzionamento della mia parte ho avviato più volte l'applicazione in modo da correggere i vari bug man mano che li trovavo.

Capacci Giulia

Per ciò che riguarda il testing della mia parte di lavoro, non ho utilizzato una suite specifica poiché grazie ad un semplicissimo metodo main() posto all'interno della classe MainMenu ho potuto monitorare costantemente l'aspetto dell'applicazione.

3.2 Metodologia di lavoro

Giancola Pierluigi

Il mio ruolo all'interno del gruppo è stato quello di sviluppare il Dungeon e tutte le entità al suo interno, senza sviluppare un interfacciamento con il Controller che ho affidato al collega Giordano.
Ho trovato utile il DVCS per quando ho dovuto affrontare due modifiche piuttosto importanti.

Giordano Andrea

All'interno del gruppo, la parte che mi spettava era quella del Controller. Per vari motivi mi sono ritrovato a fare anche la parte di interfacciamento(Facade) del Model.
La parte più complicata è stata quella di assemblare le varie parti poiché abbiamo riscontrato molti problemi in ognuna di esse. Il mio compito, a questo punto, è stato abbastanza delicato perché dovevo capire da dove sorgevano i vari problemi che poi andavano risolti.

Cerami Alessia

All'interno del gruppo, la parte che mi spettava era quella di fornire un'interfaccia grafica del progetto, almeno per ciò che riguarda le 2 schermate di gioco (Lobby e Game), permettendo il caricamento dei vari componenti del gioco.
La mia parte prevedeva anche la riproduzione di un clip audio, cosa che mi è stata resa possibile usando la libreria Minim.

Capacci Giulia

In questo progetto mi sono cimentata nell'implementazione dei menù e delle utility di gioco. Trovo molto interessante ed affascinante tutto ciò che riguarda l'aspetto grafico di un software, per questo motivo ho voluto approfondire le librerie AWT e Swing di Java.
Riassumo ora, brevemente, i menù da me implementati:

- MainMenu
- NewGameDialog
- DifficultyMenu
- ClassChoiceMenu
- PauseMenu
- CreditsMenu
- HelpMenu
- ExitDialog
- Inventory
- Shop
- Bestiary

Generale

Si è deciso, per semplicità, di utilizzare un'unica linea di sviluppo su cui lavorare. Questo ha ridotto al minimo i merge conflict.

Per integrare le parti di codice sviluppate separatamente si è convenuto di utilizzare delle interfacce per ognuno dei macro componenti (Modello,

Controller e View), tuttavia a pochi giorni dalla fine e dopo un consulto con uno dei docenti, ci si è accorti che l'analisi del modello e le relative interfacce non erano ben pensate. Ciò ha costretto il gruppo a dover modificare le interfacce causando diversi problemi.

3.3 Note di sviluppo

Giancola Pierluigi

Uno degli aspetti sicuramente più complicati è stato quello di gestire le varie collisioni che avvengono nel modello.
La libreria impiegata è Box2D, una libreria usata per le simulazioni fisiche. Inizialmente volevo tenere ben separati i codici della libreria dai miei usando il pattern Facade, tuttavia questo si è rivelato un vero e proprio bagno di sangue portandomi alla decisione di integrare la libreria al Dungeon.
Ho utilizzato anche la libreria Google Guava per sfruttare le Multimap.

Giordano Andrea

Il codice è quasi privo di commenti e probabilmente non molto efficiente. Ci sono alcune parti di codice ripetute o poco performanti che saranno corrette.

Capacci Giulia

Mi sono occupata anche dell'aspetto prettamente "grafico" del software. Poiché questa parte non è, ovviamente, oggetto di valutazione non ho ritenuto necessario contemplarla nel monte ore. Ho utilizzato il software Open Source *Gimp*[1] per tutto ciò che riguarda sprites e background del videogioco. Dichiaro che il background del MainMenu non è opera mia, ma è un'immagine presa dal web. Per le altre immagini ho preso ispirazione da numerosi videogiochi, tra cui *FinalFantasy*[2], *KingdomHeats*[3] e *DungeonEternal*[4]. I titoli dei menù sono stati generati tramite il sito *CoolText*[5]. La canzone in sottofondo è una versione a 8 bit di "*Oh Glory*" del gruppo "*Panic! at the Disco*"[6] reperita dal sito *SoundCloud*[7].

([1]): https://www.gimp.org/
([2]): https://it.wikipedia.org/wiki/Final_Fantasy
([3]): https://it.wikipedia.org/wiki/Kingdom_Hearts
([4]): http://spritedatabase.net/file/10370
([5]): https://cooltext.com/
([6]): https://it.wikipedia.org/wiki/Panic!_at_the_Disco
([7]): https://soundcloud.com/thegreenguitar/8-bit-oh-glory-panic-at-the

4 Commenti Finali

4.1 Autovalutazione e lavori futuri

Giancola Pierluigi

Sono molto soddisfatto della parte svolta da me. Penso di essere riuscito a cavarmela egregiamente nonostante la mole di lavoro e il non poter far girare l'applicazione prima dell'ultimo giorno.
Nonostante abbia testato il funzionamento di poche classi la maggior parte dei miei bug erano derivati da una comprensione sbagliata di ciò che faceva Box2D nella libreria.
Penso di aver impiegato molti design pattern e questo ha aiutato davvero molto il mio codice a non esplodere.
Durante il corso di questo progetto ho imparato ad amare gli stream e la parte della programmazione funzionale di Java, le classi che sono riuscite meglio sono l'ExceptionThrower e il TimerSingleton, ampiamente utilizzate e con le quali mi sono trovato davvero bene, anche i Pool sono state delle classi davvero utili.
Finisco questo progetto fiero del lavoro svolto e sapendo di aver imparato davvero molto.

Giordano Andrea

Nel complesso sono molto insoddisfatto del mio lavoro poiché frettoloso e fatto male. Fino all'ultima settimana di lavoro non era ancora stato implementato l'interfacciamento con il Model e ciò mi ha penalizzato molto, in quanto non sapevo in che modo poter comunicare con esso.
Mi è stato chiesto di implementare tale interfacciamento da parte del Model, data la sua enorme mole di lavoro. Io pensavo di poterci riuscire e così ho dato la mia disponibilità. Questo è stato però un errore perché mi ha portato via parte del tempo e quindi non sono più riuscito a completare la parte di gestione del salvataggio. Ciò è stato per me un problema poiché ho perso molte ore di lavoro per provare a gestire il salvataggio dei file, ma senza però

riuscire a finire di implementarlo.
All'interno del gruppo ho fornito sostegno ad una parte della View (Alessia Cerami), aiutandola nella correzione della trascrizione di codice, e svolto la parte del Model volta all'interfacciamento con il Controller oltre che all'implementazione del Controller stesso. Il fatto di essermi dedicato a cose non strettamente inerenti al Controller ha sottratto tempo all'implementazione del Controller stesso.
Per quanto riguarda le future modifiche, sarà aggiunta la gestione del salvataggio (La libreria è stata già aggiunta al progetto e studiata).
Il codice va migliorato poiché pochi sono i pattern utilizzati. Inoltre, alcune classi saranno divise in classi più piccole.

Cerami Alessia

Per quanto riguarda la trascrittura del codice, ho dovuto chiedere degli aiuti ad Andrea Giordano, in quanto ho riscontrato dei problemi. Mi ha dovuto spiegare anche a livello teorico come funzionavano certe cose.
Nel complesso ho sempre puntato ad una creazione di classi riusabili piuttosto che ad una proliferazione di classi molto simili tra loro.
In questo modo credo di aver fornito una buona base nel caso in cui, nel futuro, si vogliano apportare delle modifiche.
Lavorare in gruppo ha avuto i suoi aspetti positivi e negativi. Il dover sottostare al codice altrui certamente era una cosa alla quale non ero abituata, dal momento in cui questo è stata la prima volta che lavoravo in un progetto di gruppo.
Per quanto riguarda le future modifiche, la classe della lobby verrà modificata in modo da permettere l'interazione con degli oggetti muovendo l'eroe come se si fosse in gioco in una stanza senza nemici.
Anche la classe della creazione del clip audio sarà migliorata. Infatti, verranno aggiunti nuovi audio in modo da avere una distinzione, anche dal punto di vista uditivo, tra i vari menù e il gioco in sé. Verranno anche aggiunti i suoni per gli attacchi, la selezione delle abilità etc.

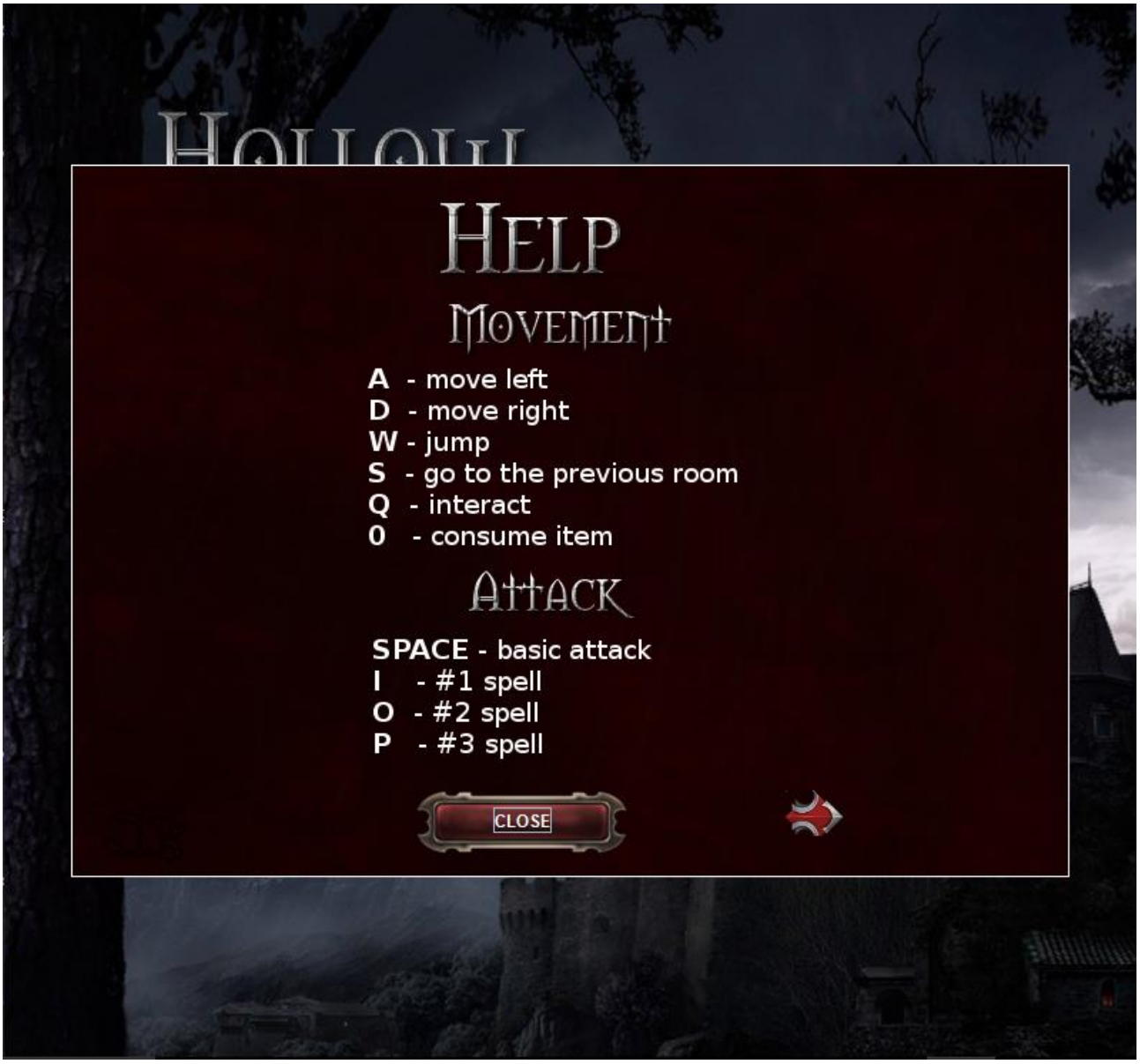
Capacci Giulia

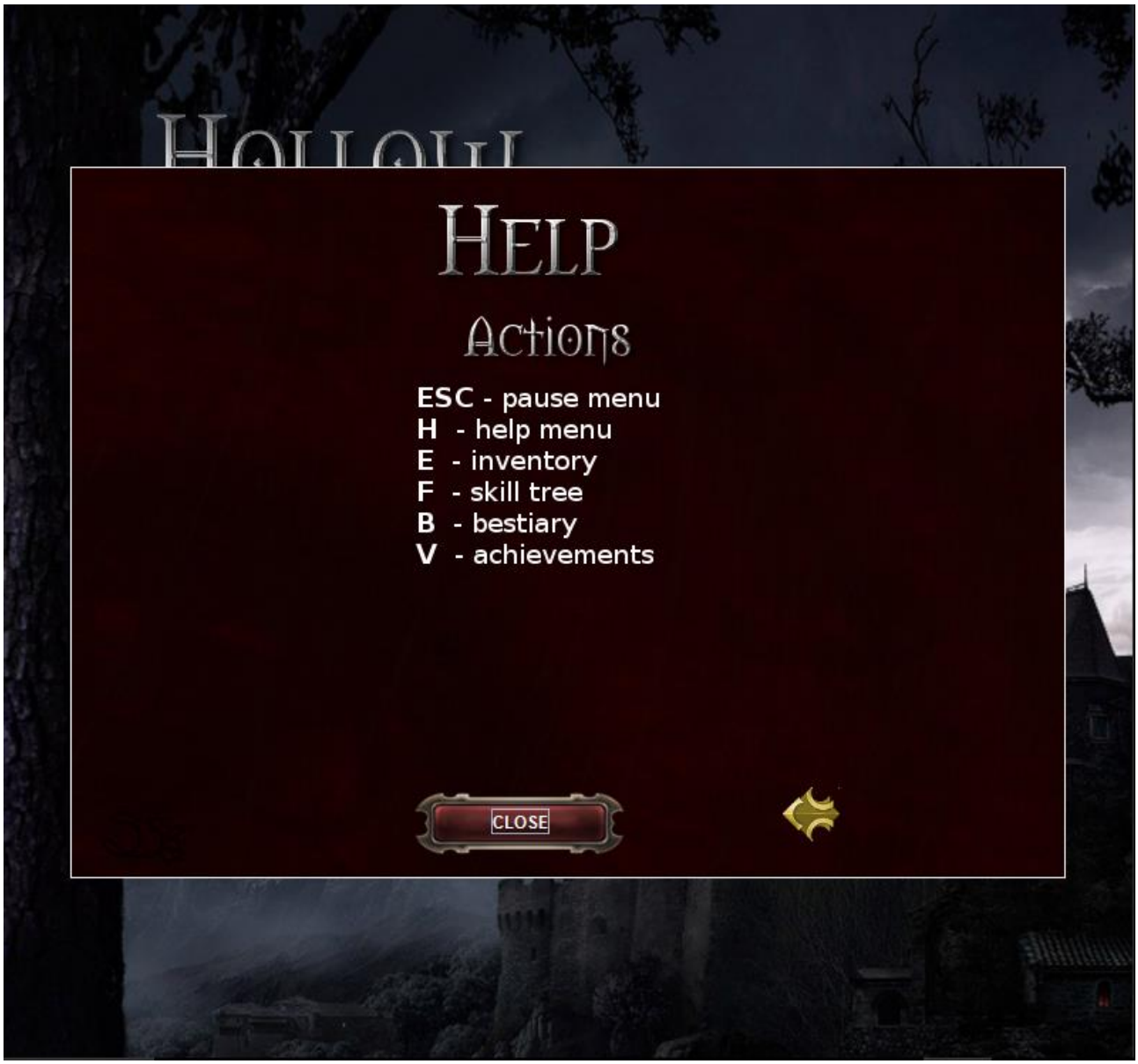
Nonostante le difficoltà riguardanti la comunicazione con alcuni elementi del gruppo, sono piuttosto soddisfatta del mio lavoro, ma allo stesso tempo sono convinta di poter migliorare alcune parti di codice. Sono orgogliosa dell'aspetto finale della mia parte di lavoro poiché è risultata corrispondere appieno alle mie aspettative. Occupandomi di questo grande progetto ho acquisito più fiducia in me stessa e nelle mie capacità e ho capito che è proprio questo l'aspetto che più mi piace e più vorrei approfondire in futuro. L'applicazione è stata pensata per essere ampliata, sicuramente mi dedicherò alla realizzazione di uno Skill Tree e di un menù dedicato alle statistiche e al Level Up dell'eroe.

A. Guida Utente

Una volta avviato, il software mostra il menù di gioco principale da cui è possibile cominciare una nuova partita, riprendere quella salvata (non disponibile in questa prima release), visualizzare una schermata di Help che riassume i comandi di gioco, leggere le informazioni relative agli sviluppatori oppure terminare la partita chiudendo l'applicazione.

Il menù di Help è molto importante per i nuovi giocatori poiché permette di conoscere tutti i comandi di gioco; consigliamo quindi di prenderne visione prima di iniziare la partita.



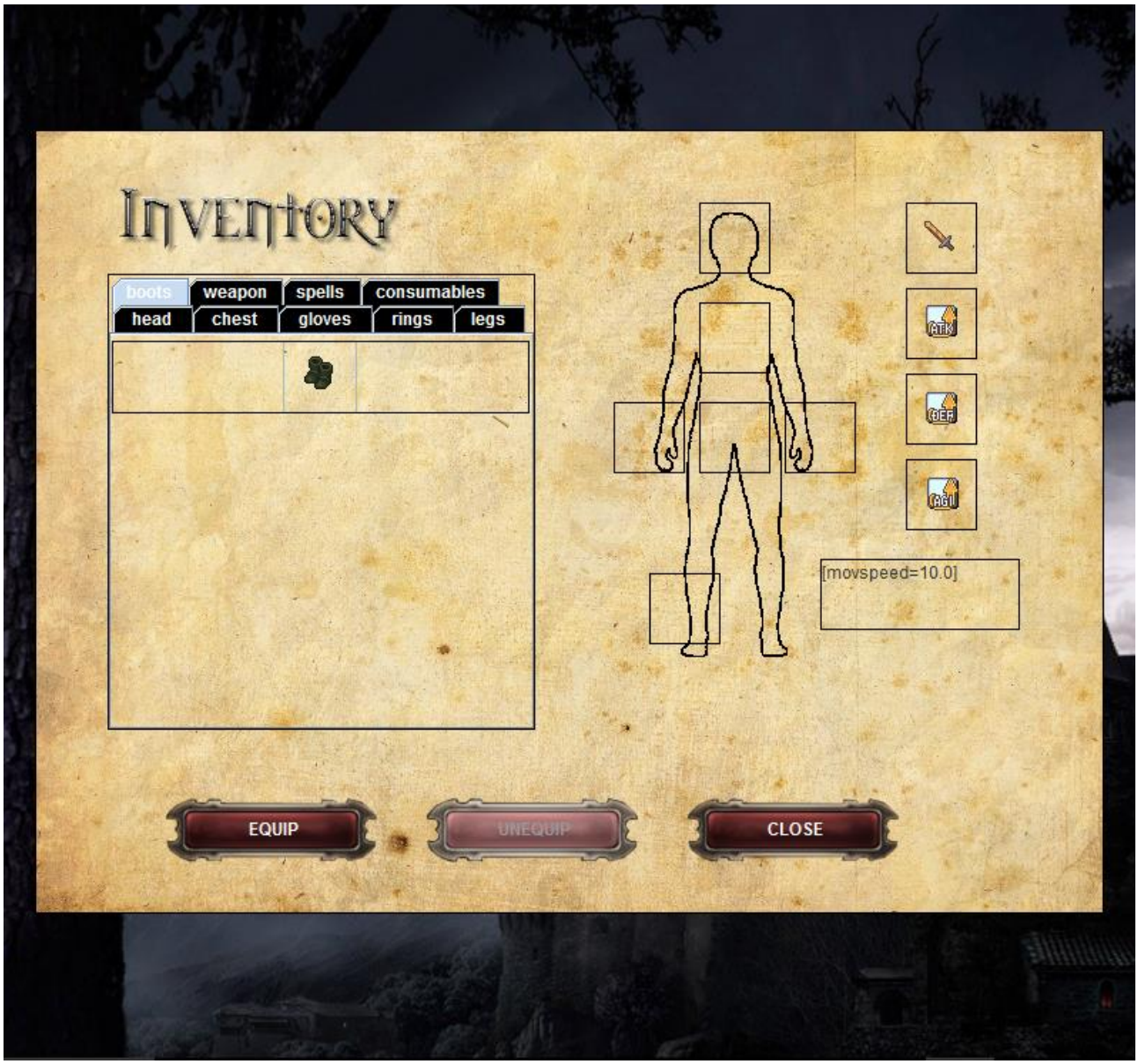


Nota: in questa release non sono presenti Skill Tree, Achievements, Spell e Consumables, per cui i relativi tasti sono disabilitati.

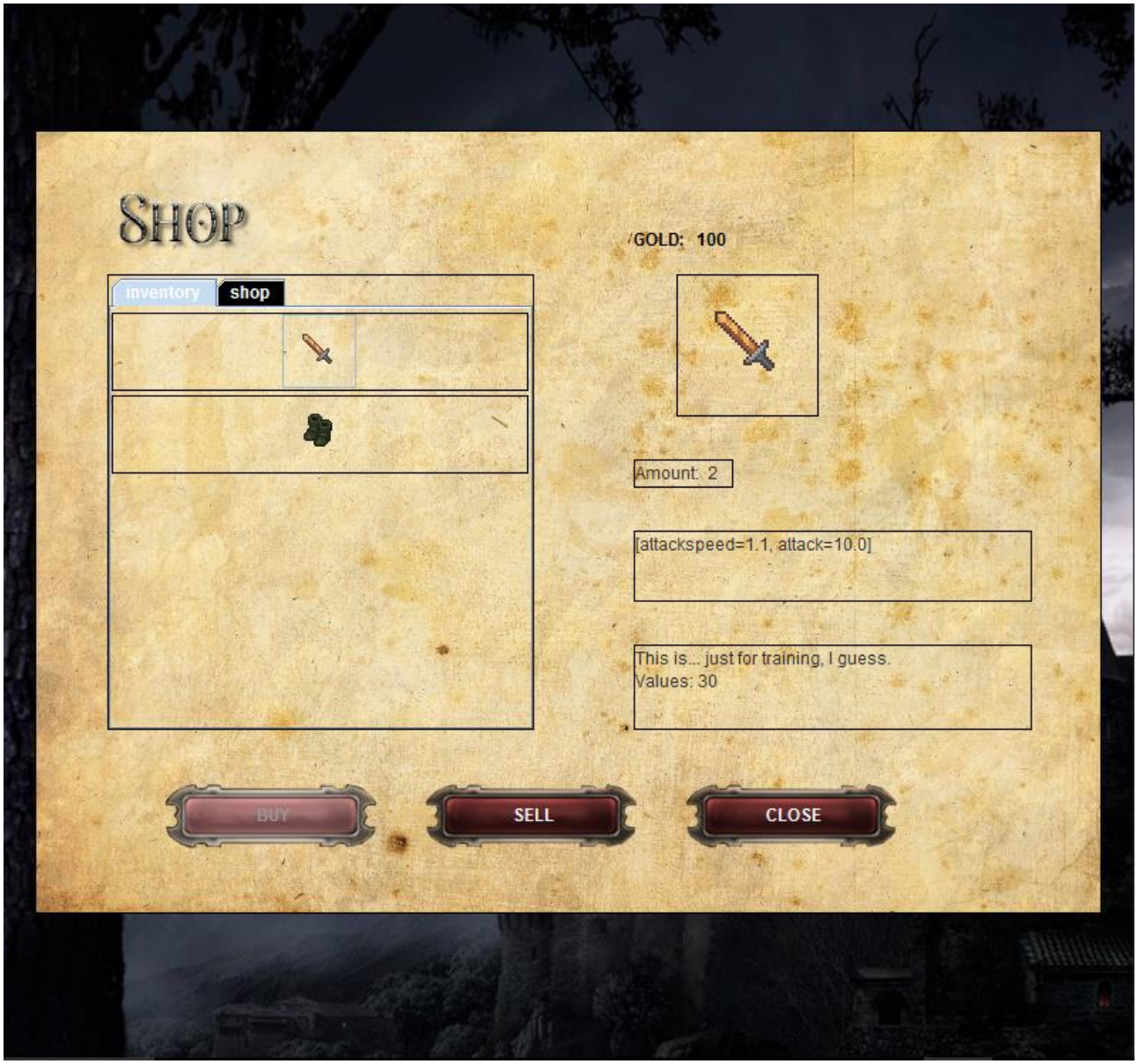
Una volta iniziata una nuova partita e selezionate classe e difficoltà ci si ritrova nella Lobby di gioco.

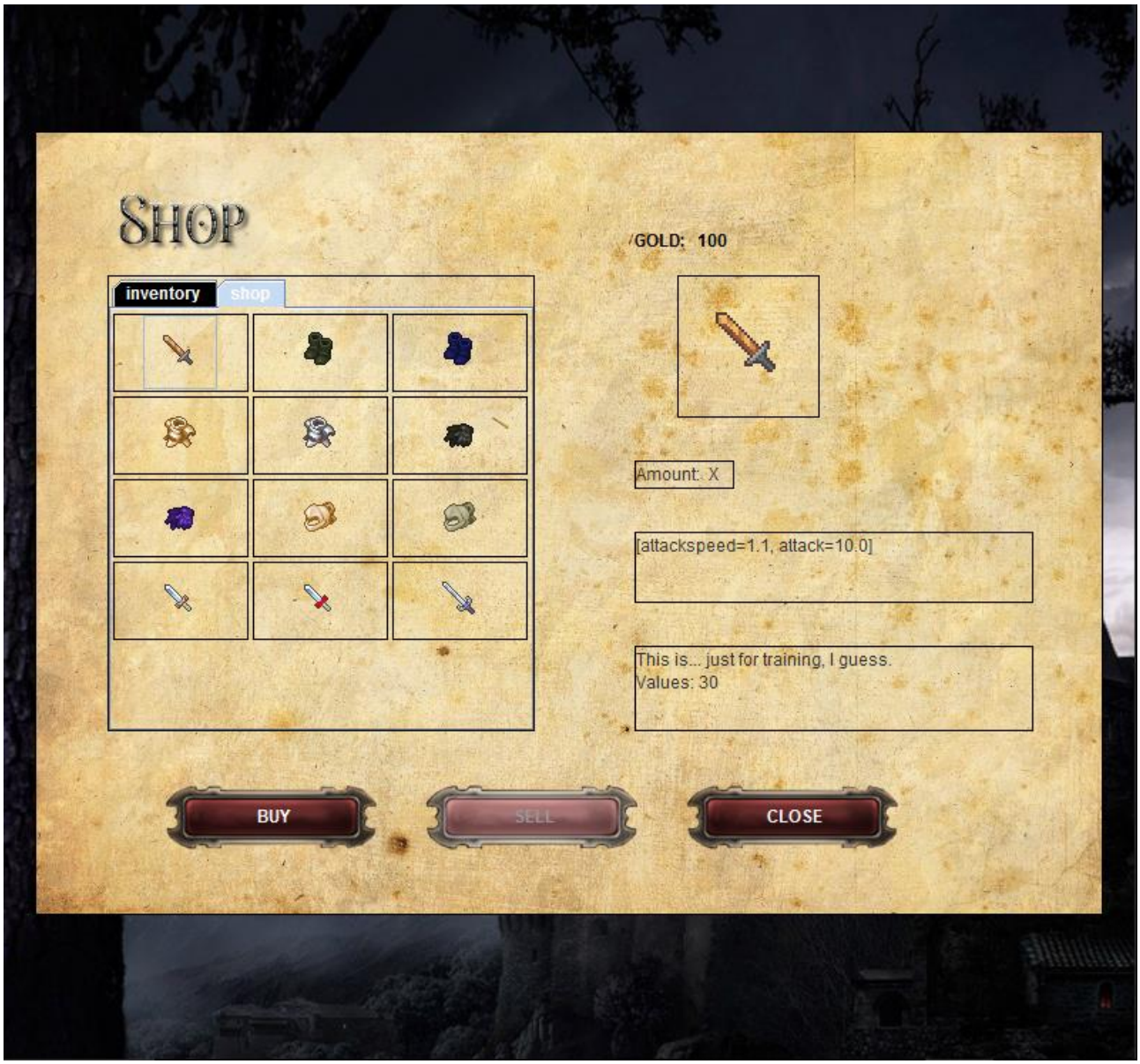
Da qui si ha accesso ai menù di utility come Inventario, Shop e Bestiario, si può tornare al menù principale oppure si può iniziare la partita vera e propria.

L’inventario permette di equipaggiare e disequipaggiare il proprio personaggio con gli oggetti trovati durante la partita oppure comprati allo shop. Per ciascun oggetto è presente un piccolo riquadro che ne riassume le statistiche.

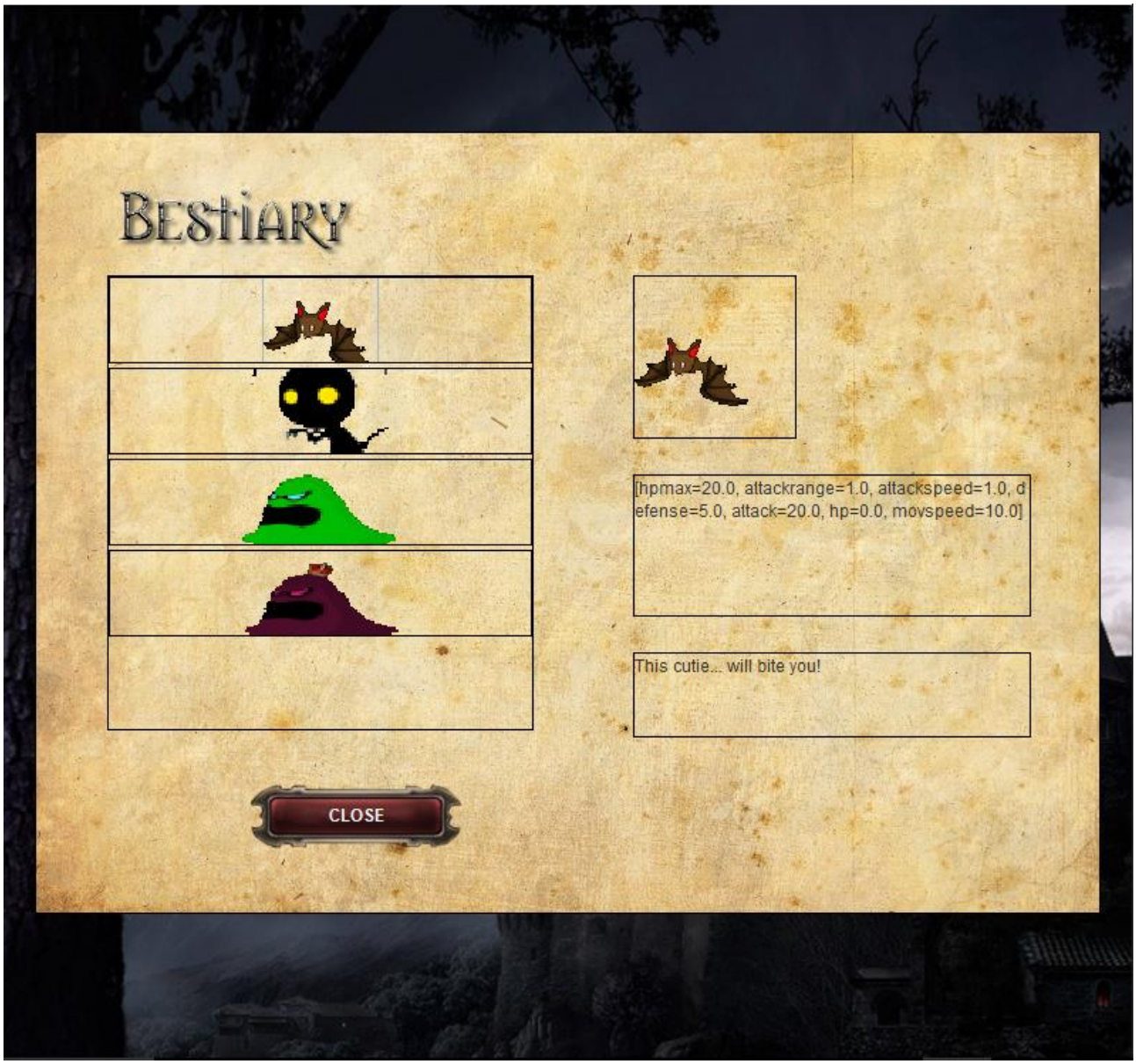


Lo Shop permette invece di vendere e/o acquistare oggetti. Selezionando un oggetto è possibile visualizzare le sue caratteristiche, tra cui quantità posseduta, statistiche e descrizione.

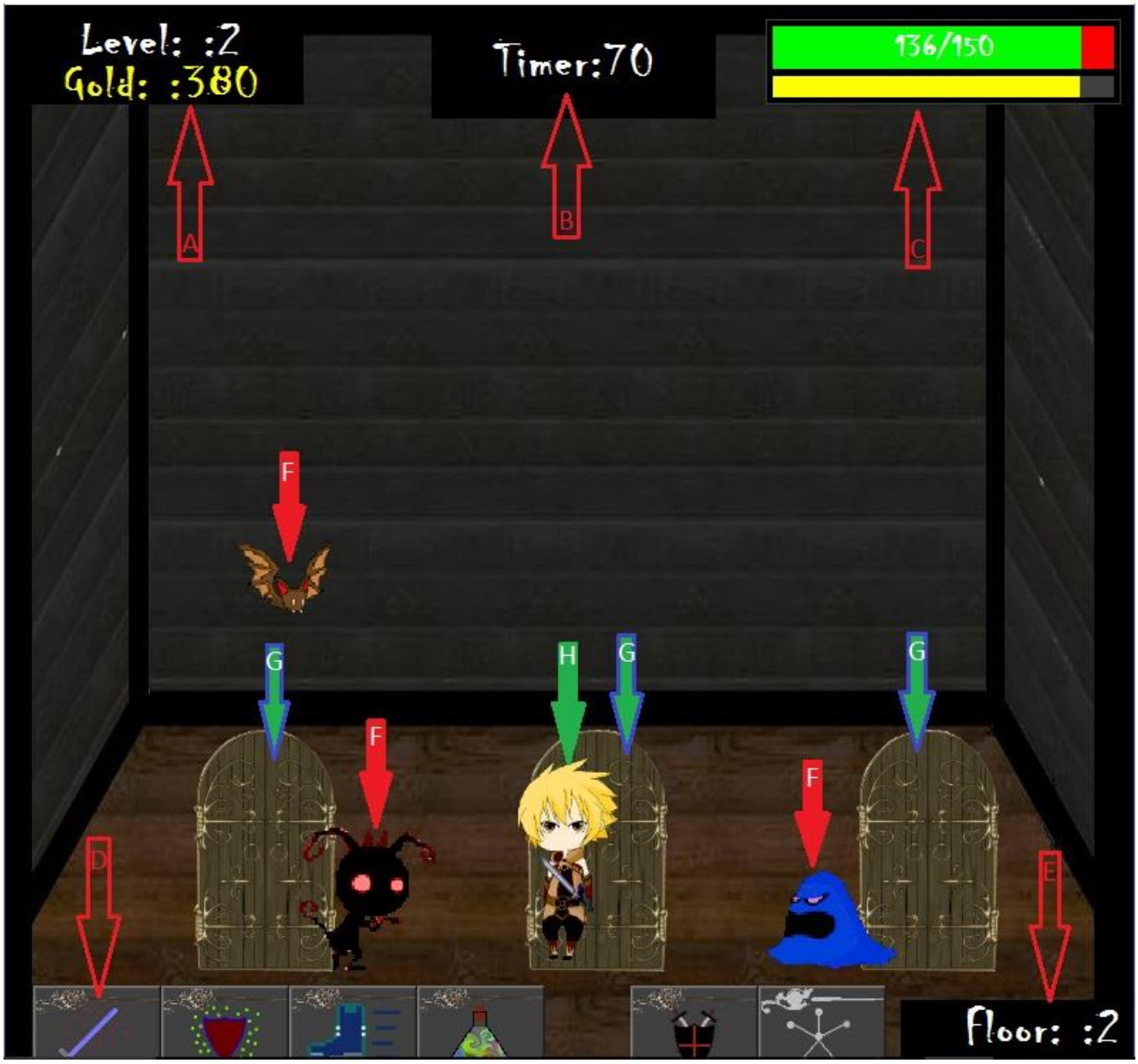



SHOP
GOLD: 100
inventory
shop
Amount: X
[attackspeed=1.1, attack=10.0]
This is... just for training, I guess.
Values: 30
BUY
SELL
CLOSE

Il Bestiario contiene informazioni relative a tutti i nemici incontrati durante la partita.



Lo scopo del gioco è quello di raggiungere l'ultimo piano del castello ripulendolo da qualsiasi traccia di Hollow Men nemico. Per poter accedere alla stanza successiva è necessario prima aver liberato quella corrente.



A) Riquadro che mostra il livello del personaggio e il gold accumulato
B) Riquadro che mostra il Timer, una volta che raggiunge lo 0 è game over
C) Riquadro che mostra la vita (barra in verde) e l'esperienza (barra in giallo)
D) Riquadro che mostra le spell e l'oggetto consumabile
E) Riquadro che mostra il piano corrente
F) Nemici
G) Porte interagibili (tasto Q quando si è davanti alla porta desiderata) permettono di avanzare tra le stanze del piano. Solo una delle porte

condurrà a una stanza con altre porte, le altre due conducono a dei tesori.

H) L'Hero, ovvero tu!

L) Questa è la zona di back presente in ogni stanza, per poter tornare alla stanza precedente ci si posiziona l'Hero sopra e si preme S

M) Treasure, contiene gold, esperienza e a volte oggetti. Per raccoglierlo si interagisce con esso (tasto Q mentre l'Hero è sopra di esso).