RELAZIONE PER LO SVILUPPO DELL'APPLICATIVO "TALISMAN"

Abtin Saadat, Alberto Arduini, Alice Girolomini, Enrico Maria Montanari

25 febbraio 2021

Capitolo 1: Analisi

1.1 Requisiti

Il team si pone come obiettivo lo sviluppo dell'applicativo "Talisman", ovvero un software che permette di giocare ad una versione semplificate del medesimo gioco da tavolo.

Requisiti funzionali

- L'applicativo dovrà permette di effettuare partite tra più giocatori umani, e ogni giocatore potrà scegliere un personaggio tra quelli disponibili prima dell'inizio della partita. Il numero massimo dei giocatori sarà, di conseguenza, uguale al numero dei personaggi disponibili;

- L'applicativo dovrà poter gestire un tabellone come quello del gioco da tavolo. In particolare, dovrà essere un tabellone diviso in tre sezioni: esterna, intermedia e interna, più una cella centrale, la casella della corona. Ogni sezione sarà composta da delle caselle, che conterranno le azioni che il giocatore dovrà compiere quando vi si posiziona sopra;

- L'applicativo dovrà poter permettere ai giocatori di spostarsi sul tabellone con le stesse modalità del gioco da tavolo, ovvero spostandosi di un numero di celle date dal lancio di un dado;

- Si dovranno gestire le carte similmente al gioco da tavolo, ovvero saranno presenti diverse tipologie di mazzi: Adventure, Talisman e Shop. Il mazzo Adventure è il mazzo "principale", che contiene tutti i tipi di carte (nemici, follower, oggetti) e da cui di solito si pesca durante il gioco; il mazzo Talisman contiene solo carte oggetto talismano, che servono per andare sull'ultima casella e vincere il gioco; il mazzo Shop è un mazzo da cui si prendono le carte in caso di acquisti durante la partita, se ad esempio lo permette una casella oppure una carta. Quando viene pescata una carta essa dovrà, seguendo le regole del gioco, essere prima posizionata sul tabellone, e poi in base al tipo di carta, si effettueranno azioni diverse, come combattere un nemico o, se è un oggetto, l'utente potrà decidere se prenderla o lasciarla sulla casella.

- Si dovranno gestire i combattimenti in modo simile a quelli del gioco da tavolo, ovvero quando un giocatore si posiziona su una cella che presenta un nemico, allora esso lo dovrà combattere facendo un tiro di dado sulla statistica che corrisponde al tipo di nemico, in quanto i nemici possono essere di tipo Forza o di tipo Astuzia, corrispondenti alle statistiche Strength e Craft;

- Saranno gestiti i combattimenti tra giocatori. In particolare, se un giocatore si posiziona su una casella dove sono presenti altri giocatori, esso potrà scegliere di sfidarli, utilizzando il medesimo sistema dei combattimenti con i nemici;

- Quando, durante un combattimento, un personaggio termina i punti vita, allora quel giocatore dovrà essere reimpostato come ad inizio partita;

- Non dovranno essere considerate alcune meccaniche avanzate, come gli incantesimi o i trofei, perché non essenziali allo svolgimento di una normale partita;
- Non sarà gestita la meccanica della magia del comando, in quanto gli incantesimi non sono gestiti. Quindi la partita sarà conclusa quando un giocatore giungerà alla cella della corona.
- Dovranno essere automatizzate quelle azioni che non richiedono interazione dell'utente, come spostare le pedine sul tabellone;

Requisiti non funzionali

- L'applicativo dovrà fornire un'interfaccia grafica che rispetti, in versione semplificata, gli elementi utilizzati durante una partita del gioco, come il tabellone, le pedine o le carte;

1.2 Analisi e modello del dominio

Il sistema gestirà, come entità di base, i giocatori e i loro personaggi. Questi rappresenteranno gli utenti durante una partita. Dei personaggi non sono state considerate le abilità speciali, ovvero azioni che il giocatore può effettuare in determinati momenti della partita, ma solo i valori univoci delle statistiche.

Ogni personaggio possiede un inventario, in cui sono memorizzate le carte che il giocatore ha raccolto durante la partita. Esse possono essere di più tipi, e ognuna ha un effetto che corrisponde all'attivazione di un'azione. I personaggi hanno anche un insieme di follower, ovvero carte che rappresentano bonus dati al personaggio. Non sono gestite le carte di tipi incantesimo, per semplificare la struttura del gioco in quando non sono essenziali per lo svolgimento di una partita.

I personaggi possono anche avere una quest attiva, che viene ottenuta, a discrezione dell'utente, posizionandosi sulla casella appropriata. Esse indicano un obiettivo da raggiungere e, quando vengono completate, forniscono un talismano al giocatore, se ce ne sono, più lo riportano alla casella dove è stata ottenuta la quest.

Anche il tabellone viene integrato nelle entità, in quanto è legato molto strettamente alle carte e ai personaggi. Questo perché, oltre ad ovviamente dover visualizzare le pedine dei personaggi nelle celle corrispondenti, dovrà anche essere mostrata la carta che è eventualmente presente su una casella e dovranno essere eseguite le azioni descritte sulle caselle sul giocatore che ci si posiziona.

Inoltre, un altro tipo di entità importante che viene gestita dal sistema sono i nemici. Questi possono essere di due tipi: Forza o Astuzia, e in base al tipo dovranno essere combattuti dal giocatore in modo diverso, ovvero utilizzando le statistiche appropriate. Dai combattimenti è stata rimossa la funzionalità di ritirata, in quanto essa è basata sugli incantesimi, anch'essi non gestiti. Un altro aspetto che non è stato considerato per motivi di tempo, ma che potrebbe essere aggiunto con facilità, sono i trofei.

Capitolo 2: Design

2.1 Architettura

L'applicazione sfrutta il pattern MVC, in particolare seguendo la sua derivazione MVP, ovvero model view presenter, in quanto si è voluto separare il più possibile il Model dalla View, facendole interagire direttamente solo quando necessario, lasciando la sincronizzazione e l'aggiornamento di esse ai controller.

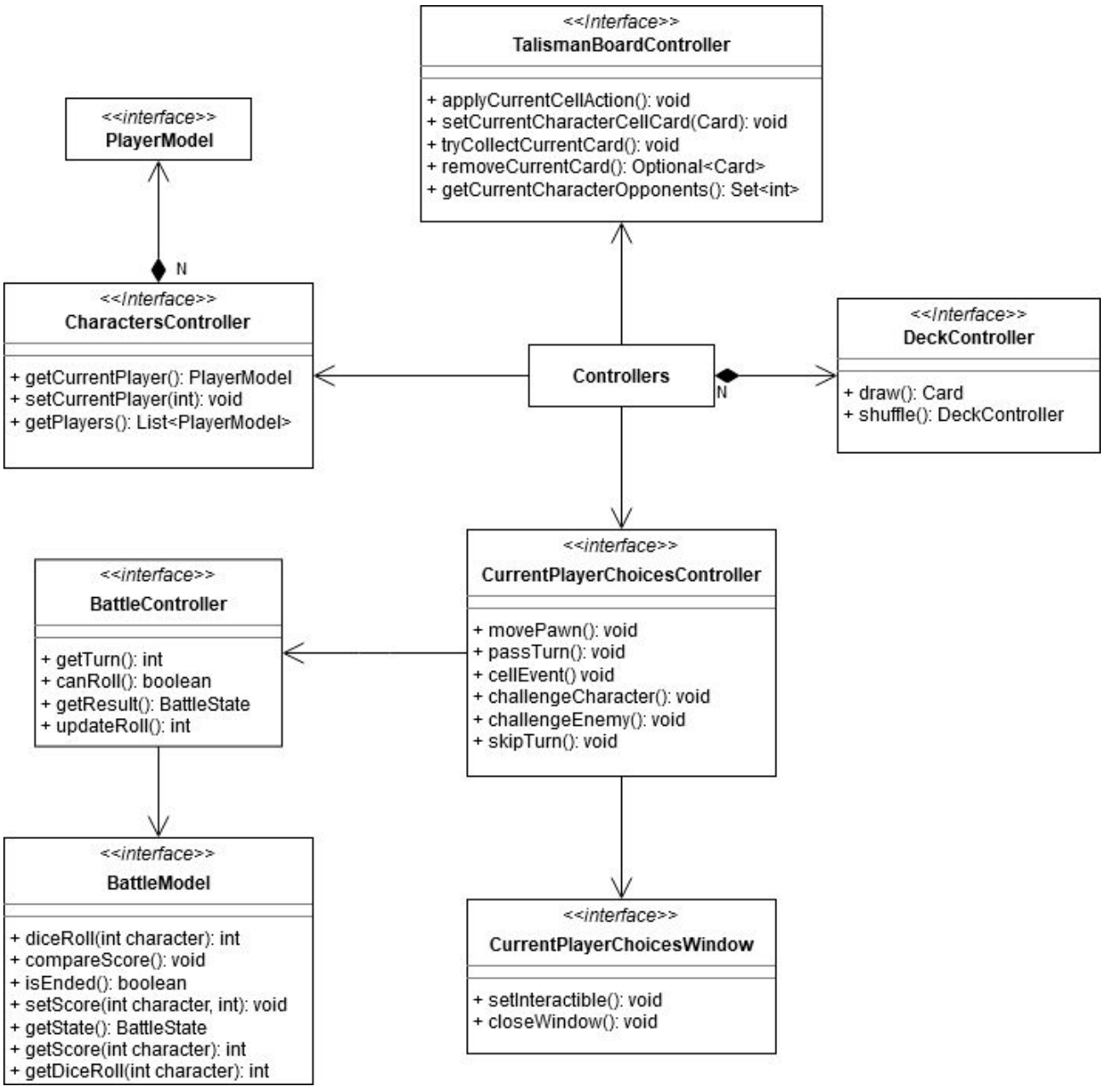
Gli input dell'utente sono forniti al sistema tramite alcune View che modellano, in modo grafico, le principali fasi del gioco. Esse sono lo stato normale del turno, dove il giocatore, dopo essersi spostato, può scegliere l'azione da eseguire, e lo stato di battaglia, dove un giocatore si sta scontrando con un nemico o un altro giocatore. Per alcuni tipi di azioni particolari sono presenti ulteriori View, mostrate solo quando opportuno, che permettono all'utente di decidere ulteriori opzioni eventualmente richieste dal sistema come, ad esempio, una casella che prevede la scelta tra più azioni.

Le View di input interagiscono con il sistema tramite i relativi controller, e saranno poi i controller ad interagire tra loro, aggiornando i Model e sincronizzando le proprie View. In particolare, le azioni del giocatore durante il turno sono gestite dal controller rappresentato da un'interfaccia specifica. Essa si occuperà di ricevere la maggior parte delle azioni dell'utente e di richiamare gli altri Controller principali del sistema per applicare le operazioni richieste.

Gli output dell'applicazione, invece, sono visualizzati principalmente tramite la View del tabellone, che mostra, oltre alla struttura del tabellone vero e proprio, la posizione dei giocatori ed, eventualmente, le carte presenti sulle caselle. Per alcune azioni particolari sono presenti ulteriori View che mostrano esplicitamente il risultato come, per esempio, il risultato del lancio di un dado richiesto da una casella.

La View è stata implementa in modo che siano necessarie il minor numero di modifiche ai Controllers e ai Models in caso di cambiamenti di implementazione, tramite l'utilizzo di interfacce che astraggono i loro comportamenti e la loro creazione.

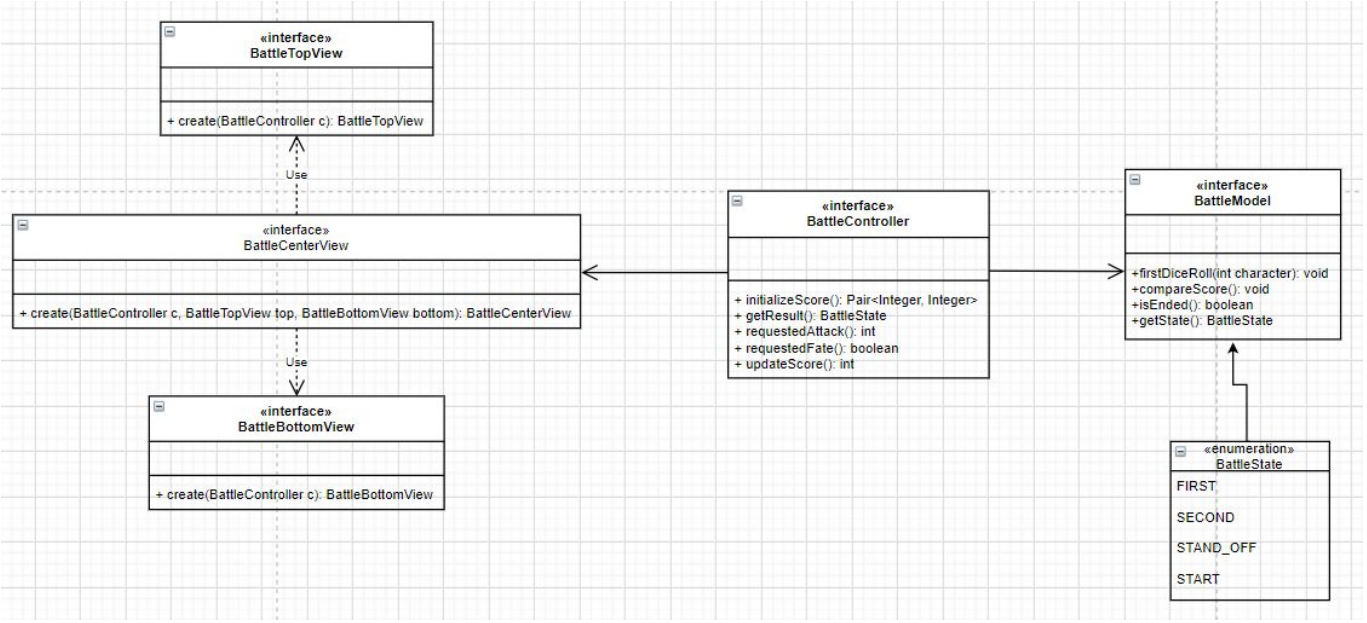
Allo stesso modo, anche buona parte del model è stato progettato allo stesso modo, per permettere un cambiamento nella gestione dei dati delle entità senza intaccare i Controller che li usano.



2.2 Design dettagliato

Alice Girolomini

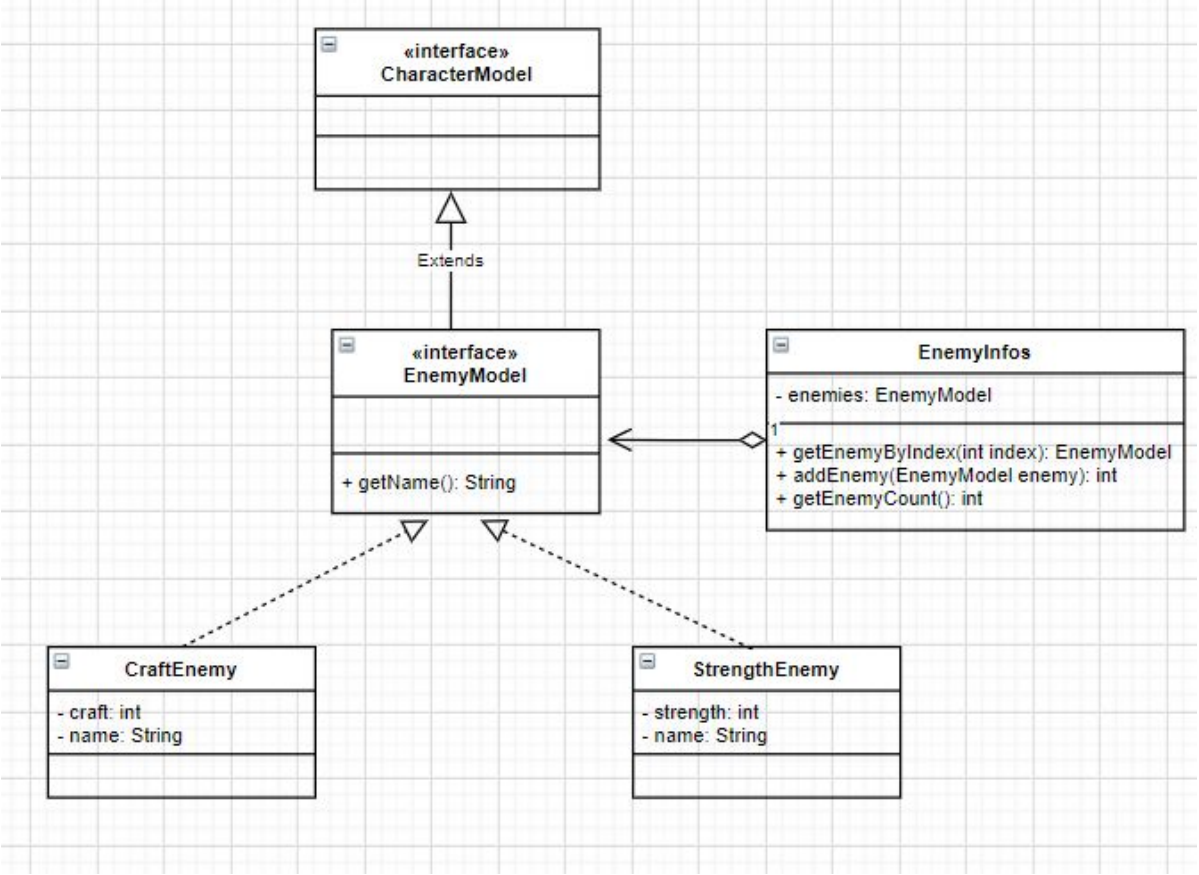
Combattimento



Ogni battaglia è gestita attraverso la relativa interfaccia grafica, controller e modello. In particolare l'interfaccia BattleModel si occupa di memorizzare le informazioni relative allo stato della battaglia quest'ultimo compito viene fatto attraverso una enumerazione BattleState che presenta i vari stati della battaglia; l'interfaccia grafica, divisa in tre parti (BattleTopView, BattleCenterView e BattleBottomView) create da BattleViewFactory, permette all'utente di compiere le azioni necessarie allo svolgimento del duello; infine BattleController viene interpellato dalla grafica per gestire gli eventi e modificare i dati contenuti nel modello.

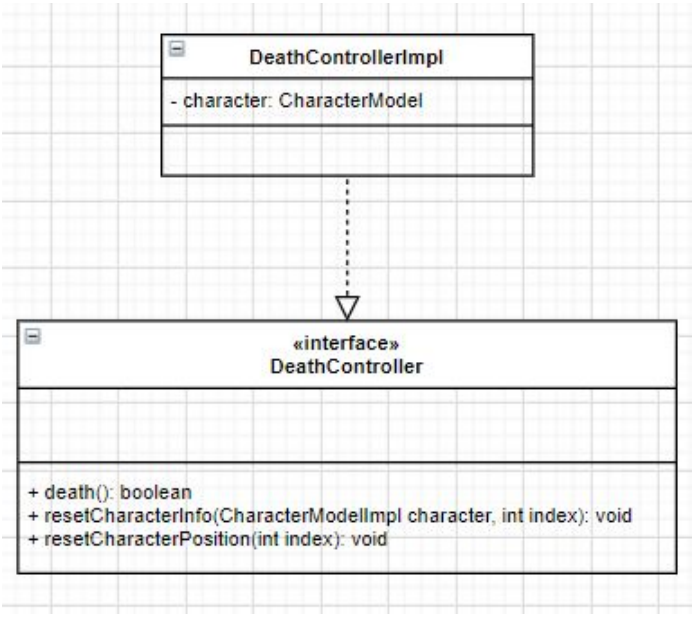
Nemici

Durante la battaglia le carte nemico vengono rappresentate attraverso l'interfaccia EnemyModel che grazie all'estensione dell'interfaccia dei personaggi permette di trattare in modo più generico gli avversari sia che siano personaggi sia che siano nemici. Abbiamo due implementazioni che sono necessarie al fine di distinguere i nemici che utilizzano forza dai nemici che utilizzano astuzia. Per tenere traccia dei possibili nemici e accedere alle loro informazioni più facilmente viene utilizzata la classe statica EnemyInfos.



Gestione morti

Per quanto riguarda la gestione delle morti dei personaggi è l'interfaccia DeathController che si occupa di verificare la vita residua del personaggio sconfitto nell'ultima battaglia e di seguito resetterà le informazioni del personaggio e lo riporterà alla casella iniziale, avvalendosi dell'aiuto degli altri controller.



Death controller viene utilizzato da BattleController sul personaggio sconfitto alla fine di ogni battaglia.

Abtin Saadat

Carte e mazzi

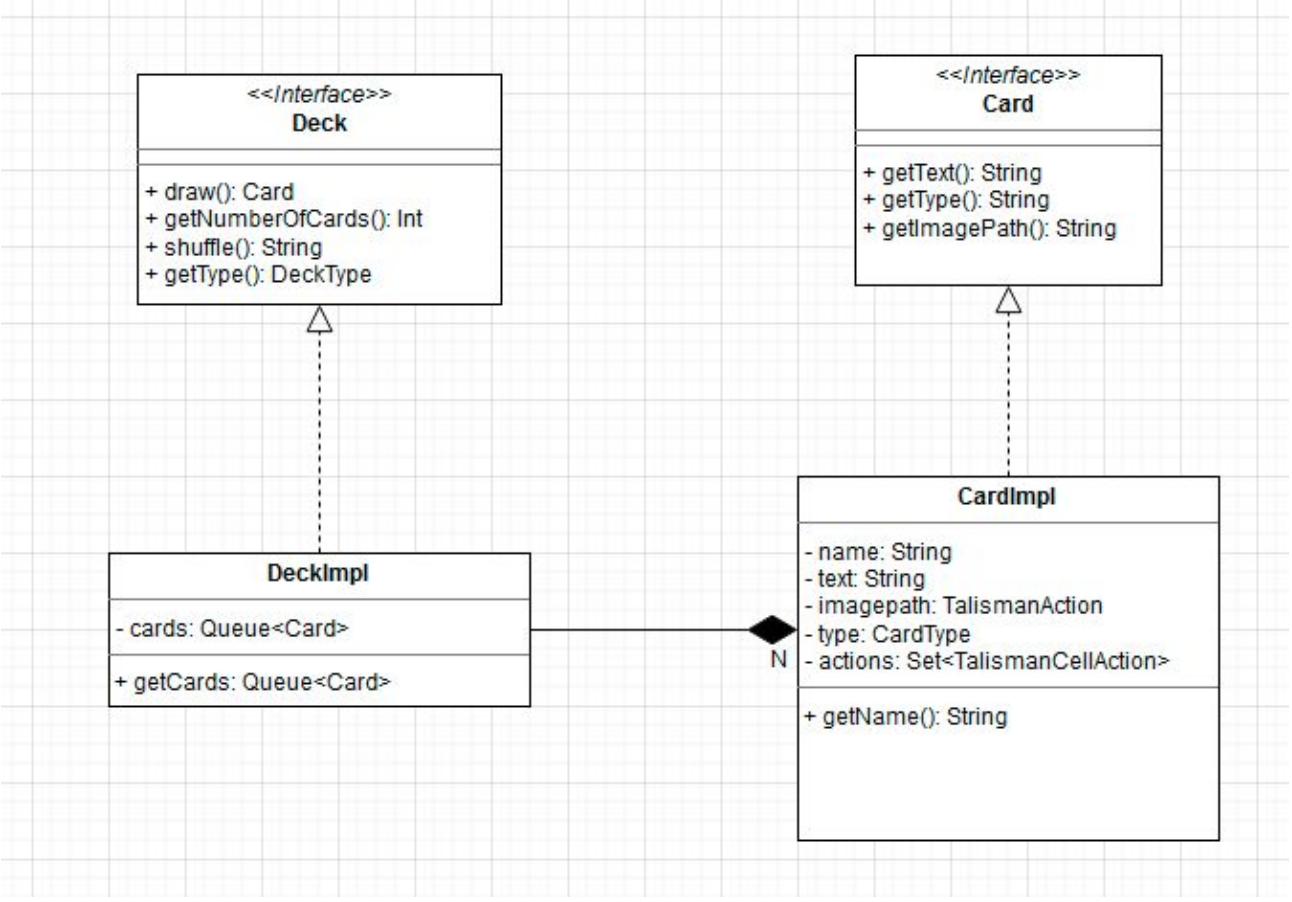
Nel gioco sono presenti vari tipi di carte, che a loro volta compongono vari tipi di mazzi.

C'è un'interfaccia Card che modella la singola carta e un'interfaccia Deck che modella un mazzo, e le loro relative implementazioni. Sono presenti due enum che modellano i tipi di carte (object, enemy o follower) e i tipi di deck (adventure, talisman e shop). Vi è anche presente una Factory che serve per generare i mazzi.

La classe Card contiene nome, descrizione, percorso dell'immagine della carta e lista di azioni, e i metodi get per ottenere queste informazioni.

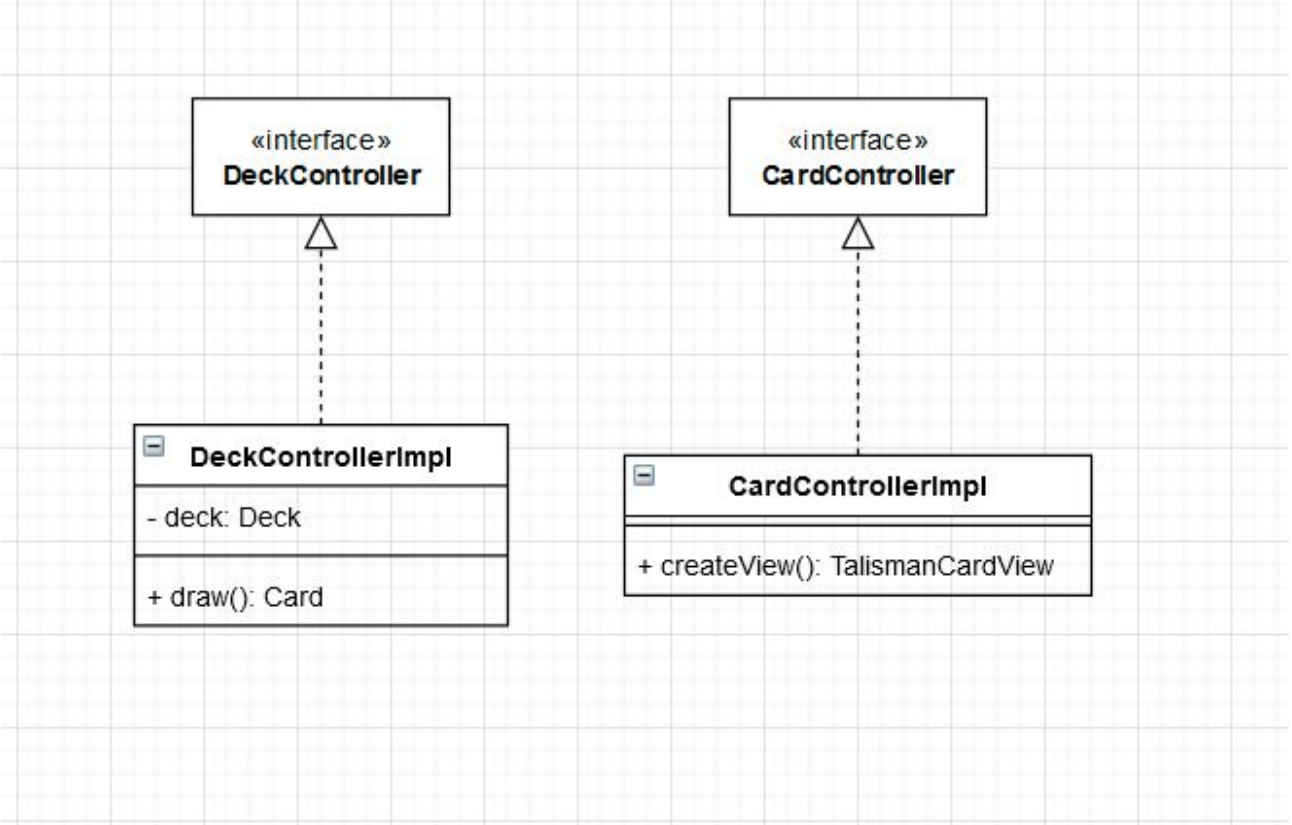
La classe Deck contiene una lista di carte, implementata come Queue, un DeckType che indica il tipo di deck e dai metodi get, shuffle, e draw.

La Factory crea i mazzi partendo dal DeckType e usando metodi statici all'interno di essa per creare le carte da aggiungere al mazzo. Il metodo shuffle (che richiama il metodo shuffle di Collections) non viene chiamato all'interno della Factory, anche se ai fini del gioco non ci sono casi di mazzi non mischiati, per separare model da controller il più possibile.

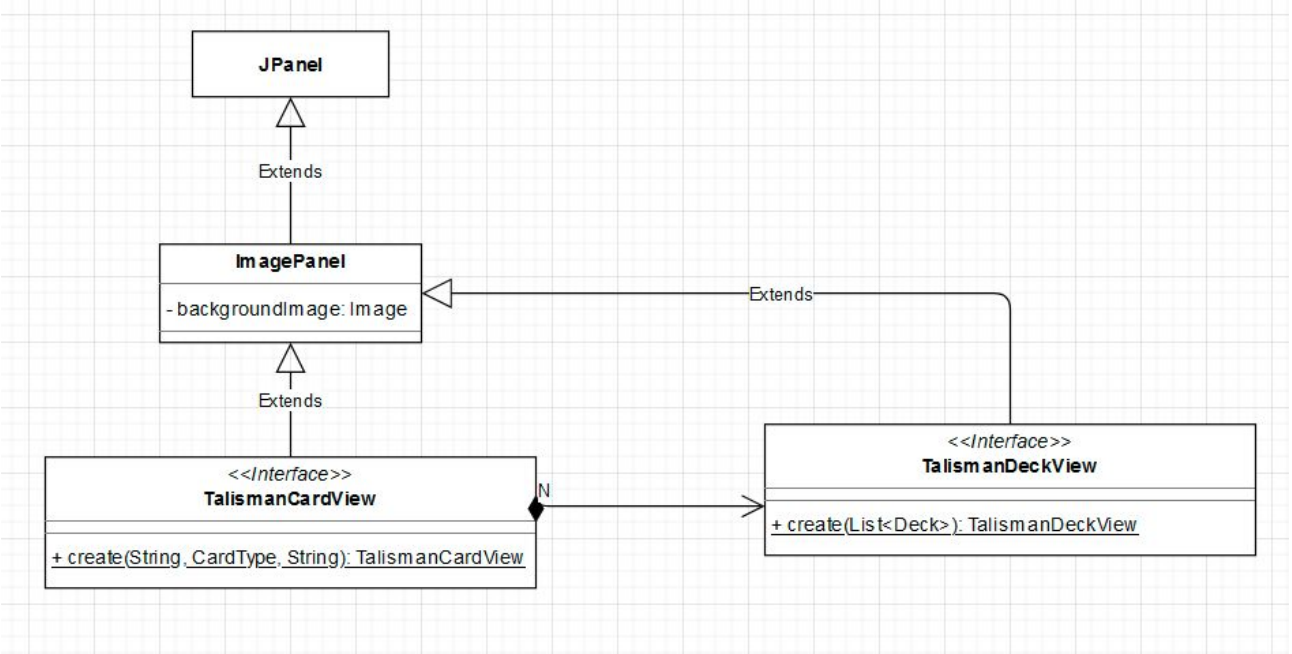


Ogni istanza di deck ha il suo controller associato (DeckController), che fa da wrapper al model del deck e permette la comunicazione con la view e gli altri controller.

Il CardController invece contiene un singolo metodo statico che prende come parametro il model della carta e ritorna la view.



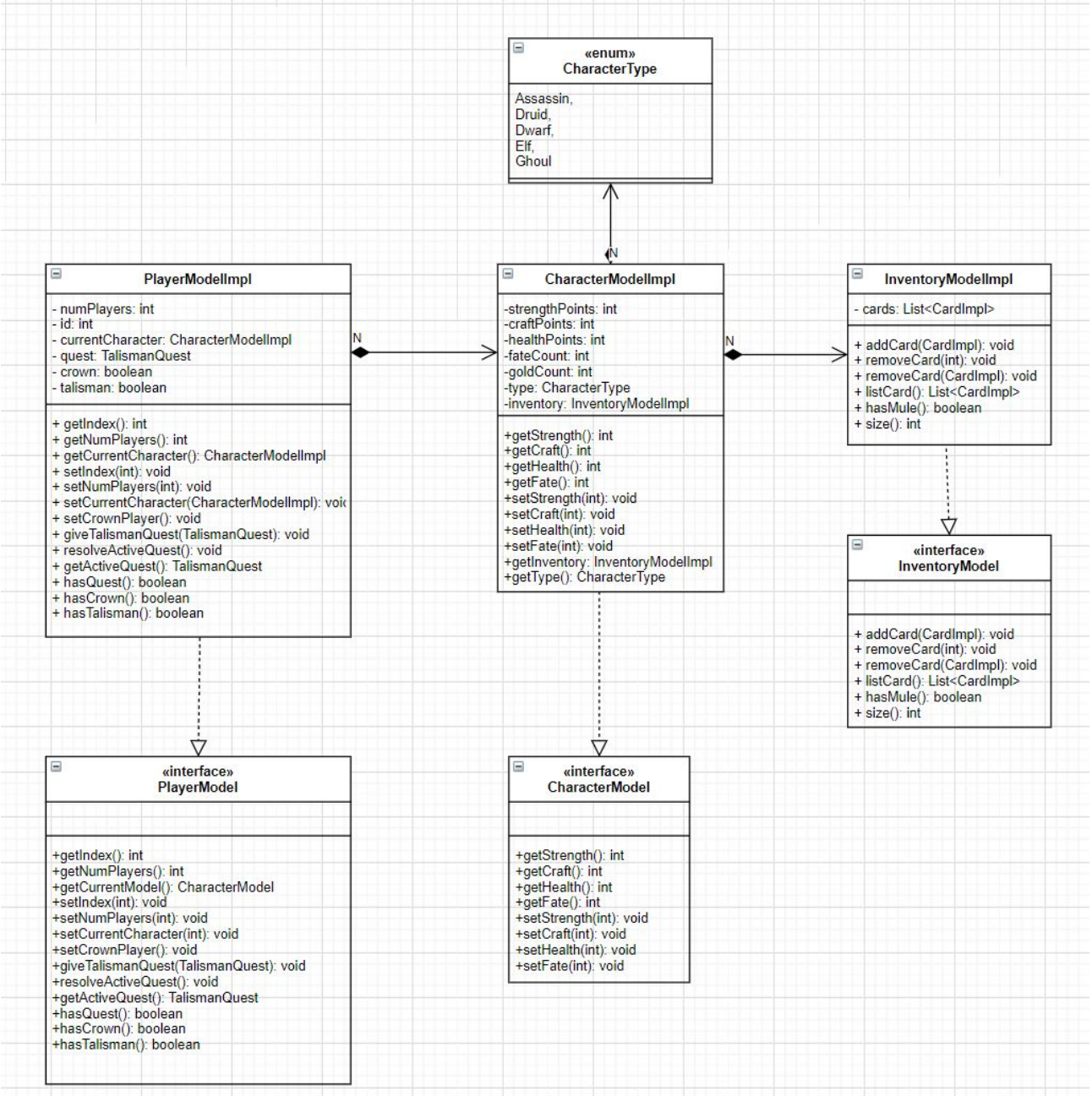
La View delle carte è strutturata similmente al model, c'è la view della singola carta e la view di un mazzo (o lista di carte di un giocatore) che è composta da una lista di view delle carte che si possono scorrere tramite due bottoni.



Enrico Maria Montanari

Gestione giocatori

Nel nostro progetto i giocatori sono rappresentati da una classe specifica, la quale svolge la funzione di tramite per collegare tutte le caratteristiche tipiche di un giocatore, come ad esempio l'inventario o le statistiche del suo personaggio giocante. In figura è rappresentato lo schema UML dei giocatori.



In questa prima figura possiamo già identificare le principali classi che gestiscono i giocatori.

Troviamo la classe PlayerModel, dove vengono memorizzati tutti i dati più relativi alla persona fisica che gioca, piuttosto che al personaggio. In questa classe, infatti, vengono memorizzati tutti quei dati riguardanti al giocatore che sicuramente esistono solo univocamente, come ad esempio la presenza o meno di un talismano nell'inventario di un giocatore.

Al contrario, nella classe CharacterModel, abbiamo tutti i dati riguardanti direttamente il personaggio interpretato dal nostro giocatore, come ad esempio la vita, i soldi, ecc..
Aggregati a questa classe troviamo un enum chiamato CharacterType e la classe modello dell'inventario del personaggio, dove vengono memorizzate tutte le carte in possesso al giocatore.

Nella classe riguardante l'inventario del personaggio troviamo metodi e funzioni propriamente creati per facilitare il più possibile la gestione delle carte del personaggio.

L'enum CharacterType, invece, serve per identificare la classe di partenza del personaggio, così che, dovesse il personaggio morire durante il gioco, noi poi possiamo risalire alla classe originaria del personaggio facilmente.

Per concludere questa prima parte, esiste anche un Controller chiamato CharacterController. Tramite questa classe è possibile accedere in un qualsiasi momento a tutti i giocatori e personaggi in gioco, assieme ad alcune comode funzioni per gestire la partita, come ad esempio sapere chi è al momento alla corona del comando, anche se ora come ora inutilizzata per via delle condizioni di vittoria semplificate.

Altro aspetto importante riguardante i personaggi sono le quest, queste consentono ai giocatori di ottenere i talismani e poter così accedere all'area finale di gioco.

Nel nostro progetto le quest vengono richiamate da una classe TalismanAction ma comunque gestite principalmente all'interno della classe PlayerModel.

Come è possibile notare nell'UML sopra le quest vengono identificate con TalismanQuest, questa infatti è una classe astratta base, la quale poi viene ereditata dalle classi specifiche per la tipologia di quest, quali: DeliverObject, KillEnemy, TakePlayerLife.

Mentre le ultime due classi non hanno nulla di più se non un metodo toString(), come nella classe parente, DeliverObject possiede invece due metodi per gestire l'oggetto da consegnare in una determinata cella del tabellone, questo può essere scelto casualmente o meno. L'oggetto da consegnare è poi descritto tramite un enum, chiamato QuestObjectType.

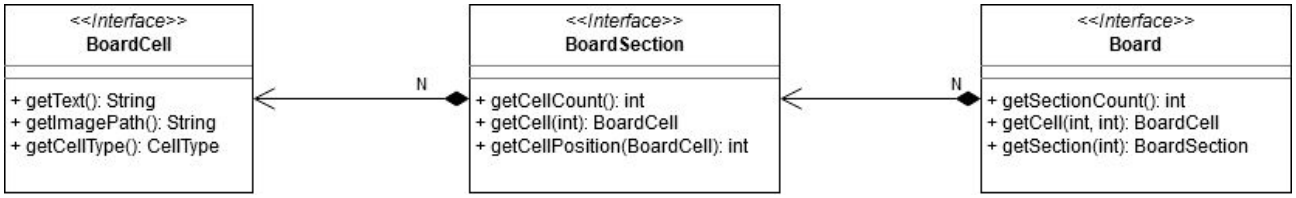
Infine troviamo un sistema di enum che implementano tutti un'interfaccia comune Character, con lo scopo di fornire un metodo separato per risalire ad alcuni dati statici, piuttosto che usare dei file di testo. In particolare qui si trovano tutti i dati delle classi di default dei personaggi, per es. quanta vita ha un personaggio ad inizio gioco. Come citato prima questi enum potrebbero essere benissimo sostituiti da una forma di memorizzazione dati più consona, come un file di testo o ancora meglio un database, soprattutto se il gioco dovesse avere più di 5 classi di partenza.

Sia nelle classi riguardanti le quest che i personaggi di default, sono presenti metodi, altrettanto statici e rimpiazzabili da un file di testo, con funzionalità estetiche, al momento non utilizzati, come ad esempio la descrizione del personaggio.

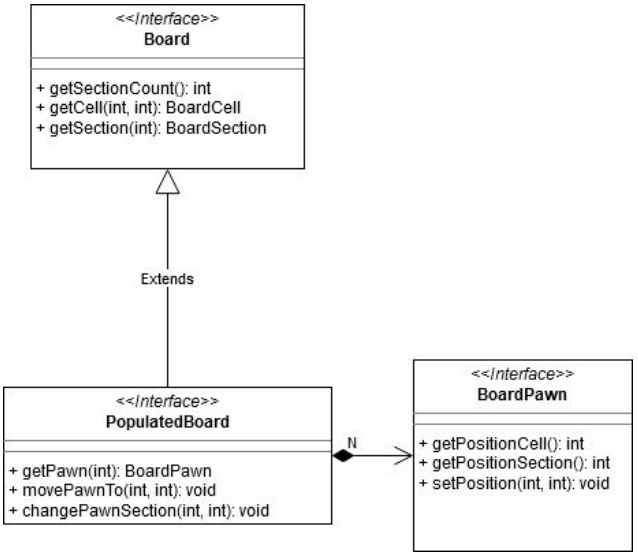
Alberto Arduini

Tabellone

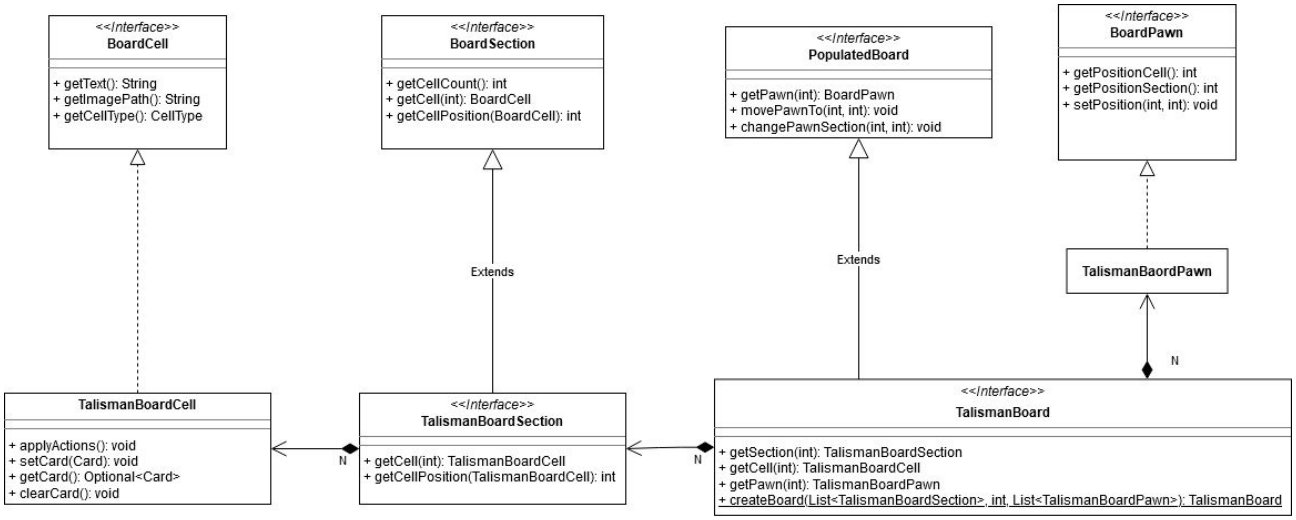
Il tabellone del gioco è stato progettato cercando di modellare per primo un generico tabellone da gioco composto da più sezioni e in cui ogni sezione è composta da più celle, tramite le interfacce Board, BoardSection e BoardCell, come in figura.



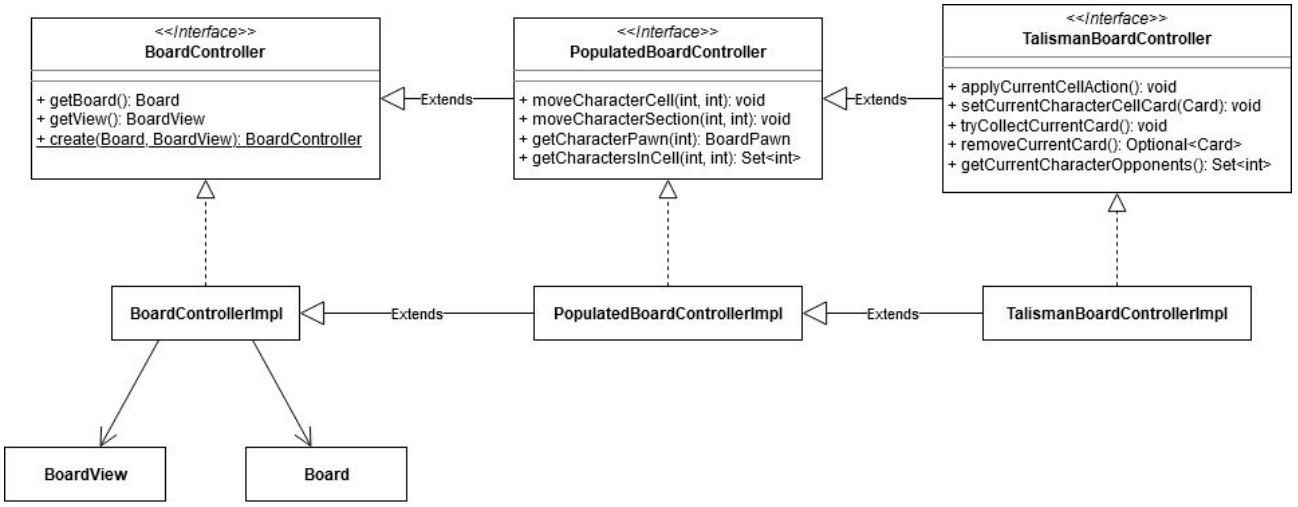
Ad esse è poi aggiunta la gestione delle pedine, sempre in modo che sia indipendente dal gioco che si sta modellando, tramite le interfacce PopulatedBoard e BoardPawn. Si è pensato di lasciare la memorizzazione della posizione delle pedine all'interno della loro interfaccia, mentre il tabellone si occuperà solo di informarle quando sono spostate, senza gestire direttamente le posizioni.



Infine, sono poi aggiunte le interfacce e le implementazioni specifiche per Talisman, tramite TalismanBoardCell, TalismanBoardSection, TalismanBoard e TalismanBoardPawn. Esse vanno ad aggiungere quelle funzionalità specifiche al gioco stesso, come le azioni o le carte sulle celle. Tutte le componenti del Model specifico del tabellone forniscono un metodo statico create che, seguendo il pattern del factory method, creano il Model appropriato senza dover conoscerne l'implementazione.



Il tabellone viene poi gestito da un Controller apposito. Anch'esso è stato progettato prima per gestire un tabellone generico, di un qualunque gioco, e poi ne è fornita una versione specifica per talisman. Esso si occuperà di ricevere le operazioni da effettuare sul tabellone, e sarà poi lui a modificare il Model e mantenere allineata la View, senza la necessità di un intervento esterno, in modo da non vincolare altre componenti del sistema sul tipo di Models o di Views utilizzati. Inoltre, come nel Model del tabellone, è fornito un metodo statico create che, seguendo il pattern del factory method, crea un controller dati i suoi Model e View.

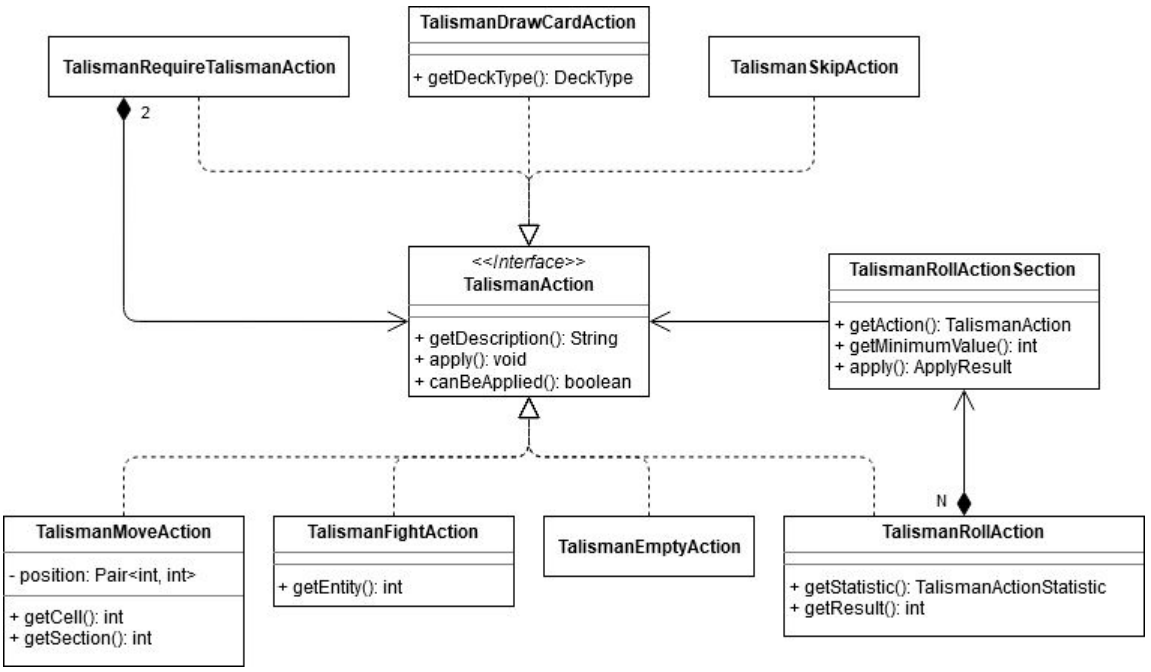


Azioni

Questa parte di progetto è stata progettata in comune, in quanto queste azioni sono utilizzate sia dalle carte che dalle caselle del tabellone.

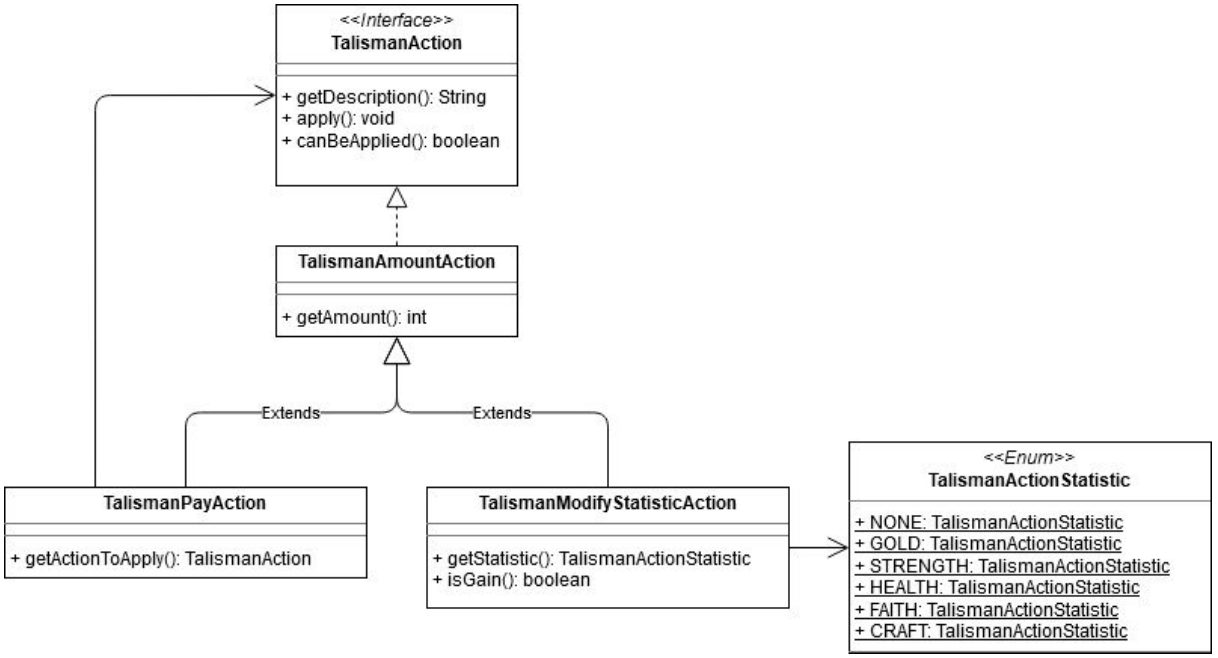
Il Model delle azioni è strutturato partendo da un'interfaccia che modella le operazioni essenziali che permettono di eseguire le azioni sul giocatore corrente, ovvero quello che ha il turno, e di visualizzarle correttamente in una view.

Da essa poi sono create le varie implementazioni delle azioni, e ogni tipo di azione descrive cosa essa deve fare al giocatore.

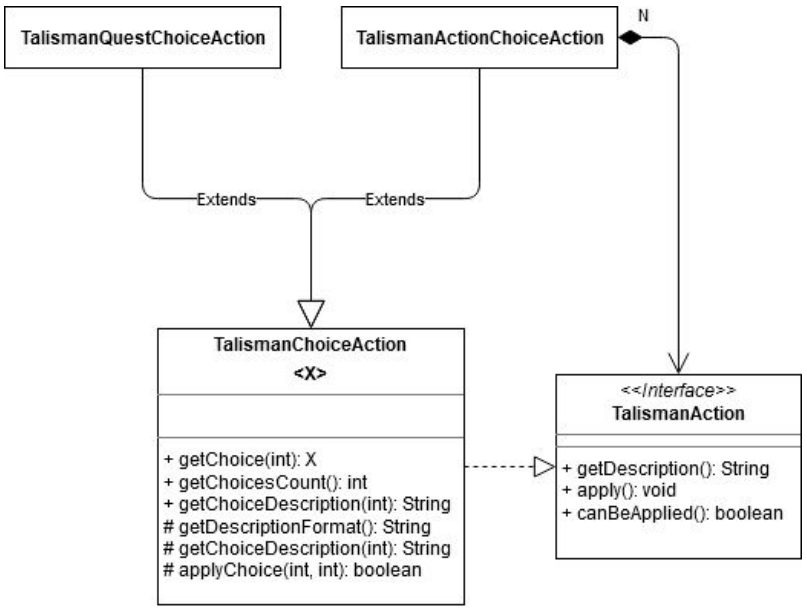


In più sono stati considerati alcuni tipi di azioni che, essendo accomunati da alcune funzionalità o caratteristiche, generalizzano un aspetto del loro comportamento. In particolare esse sono le azioni che richiedono un valore (di minimo, di massimo, etc.) e quelle che permettono una scelta da parte dell'utente. Esse sono state sviluppate come illustrato nelle figure successive:

- Azioni con valore:



- Azioni con scelta dell'utente:



Capitolo 3: Sviluppo

3.1 Testing automatizzato

Il testing dell'applicativo è fatto sulle principali componenti del sistema, in particolare su alcuni Model e Controller. I test sono fatto tramite la libreria JUnit 5 e, in particolare, tramite la API jupiter che essa fornisce.

Metodi di test

- testBattleModel: verifica le funzioni base del modello battaglia ovvero il lancio del dado dei diversi personaggi e la condizione di vincita.
- testBattleController: verifica il corretto funzionamento del controller tra cui la gestione dei turni degli sfidanti, l'utilizzo di monete fato e la somma dei punteggi .
- testDeathController: esegue un test sulla fine della battaglia, la morte del personaggio, il reset alla posizione iniziale e il reset delle informazioni del personaggio assegnato al giocatore.
- testMoveAction: esegue il test dell'azione presente sulle caselle del tabellone che deve spostare il personaggio.
- testPayAction: esegue un test che controlla il funzionamento dell'azione che richiede al giocatore di pagare per eseguire un'ulteriore azione.
- testGiveItemAction: esegue il test dell'azione che fornisce al personaggio una carta da un mazzo.
- testRequireTalismanAction: controlla il funzionamento dell'azione che richiede al giocatore di possedere un talismano.
- testModifyStatisticAction: controlla il funzionamento dell'azione che modifica le statistiche del giocatore, sia in modo positivo che negativo.
- testMovePawn: controlla che la pedina di un giocatore sia spostata in modo corretto, sia tramite il model che tramite il controller.

3.2 Metodologia di lavoro

- Alice Girolomini: realizzazione della grafica, modello e controller della battaglia, modello dei nemici, controller per la gestione morti delle morti dei personaggi, controller e grafica del personaggio corrente, grafica per la scelta degli avversari.
- Abtin Saadat: realizzazione della grafica, modello e controller delle carte e mazzi, Factory per la generazione di carte e mazzi, realizzazione della view scorrevole della lista di carte del giocatore dentro il pannello del turno del giocatore.
- Enrico Maria Montanari: realizzazione del modello, controller e factory dei giocatori e personaggi interpretati, realizzazione delle quest portate dai giocatori e dei dati di default per i personaggi
- Alberto Arduini: realizzazione del tabellone, in particolare dei suoi modelli, controller e interfaccia utente, realizzazione delle azioni delle caselle e delle carte progettate in comune, realizzazione delle grafiche e dei modelli del menu iniziale.

Le parti dell'applicativo realizzate e progettate in comune sono le azioni delle caselle e delle carte, perché i loro tipi e il funzionamento sono i medesimi, e la gestione delle azioni che il giocatore può effettuare durante il proprio turno, in quando esse ricoprono funzionalità sviluppate da tutti i membri.

Per fare queste integrazioni si è preferito prima realizzare le parti dei singoli membri, in modo da poter sviluppare le funzionalità nel modo più aderente possibile alla progettazione, e di adattare poi le interfacce fornite dalle parti di ognuno perché potessero essere integrate correttamente, fornendo metodi aggiuntivi o modificando quelli esistenti.

L'applicativo è stato sviluppato utilizzando il DVCS in modo essenziale. La repository utilizzata è stata infatti organizzata con due branch: una branch "dev", ossia quella attivamente in sviluppo, dove sono stati fatti i commit dai membri del gruppo, e una "master", che contiene la versione dell'applicativo stabile.

3.3 Note di sviluppo

Alice Girolomini

- Lambda
- Stream

Alberto Arduini

- Classi con generici;
- Optional;
- Lambda expression;
- Stream;

Abtin Saadat

- Lambda

Capitolo 4: Commenti finali

4.1 Autovalutazione e lavori futuri

Alice Girolomini

La realizzazione del progetto è stata un'esperienza istruttiva ma anche impegnativa soprattutto per quanto riguarda la gestione del tempo e la necessità di comunicare con gli altri componenti del gruppo. Nel complesso posso ritenermi in parte soddisfatta del mio percorso, in quanto ho visto una crescita nella comprensione e nell'utilizzo della materia, d'altro canto però avrei potuto organizzare in maniera più efficiente il lavoro se avessi chiesto prima consigli in quanto è il primo lavoro di grandi dimensioni che realizzo. Sicuramente questo progetto mi ha permesso di capire il modo migliore di lavorare per progetti futuri.

Enrico Maria Montanari

Portare a termine questo progetto è stato sicuramente soddisfacente, soprattutto considerate le mie disponibilità, purtroppo, di tempo. Ho imparato a gestirmi al meglio i ritagli di tempo e che la comunicazione è essenziale, non per forza via testo o vocale ma anche solo con i commenti nel codice, in modo che quando c'era la possibilità si potesse lavorare il più velocemente possibile.

Sicuramente avremmo potuto organizzarci meglio se avessimo pensato ad una scaletta temporale con delle deadlines, magari una prima fase di brainstorming tutti assieme, però considerando com'è andata sono decisamente soddisfatto.

Sono sicuro che se ci fosse una seconda occasione, tutti noi lavoreremmo con una marcia in più.

Alberto Arduini

Realizzare questo progetto è stato istruttivo soprattutto per quanto riguarda l'organizzazione dei membri e del lavoro proprio e degli altri, soprattutto per quanto riguarda il fornire supporto ai membri che ne hanno avuto bisogno e per il cercare di realizzare codice facilmente interpretabile e utilizzabile da altri, permettendo loro una più facile integrazione con la mia parte.

Un punto su cui avrei preferito una migliore organizzazione è stato l'utilizzo del DVCS, in quanto è stato utilizzato in modo molto semplice, e quindi non sono state sfruttate funzionalità che avrebbero agevolato lo sviluppo e l'organizzazione della repository.

Sicuramente il progetto potrebbe essere continuato, considerando che alcuni aspetti più specifici del gioco non sono stati gestiti per mancanza di tempo e altri sono stati semplificati. Quindi, un'eventuale espansione dell'applicazione potrebbe essere quella di completare prima le componenti già esistenti, sistemando le interfacce grafiche e fornendo un'esperienza utente più gradevole, e poi passare all'aggiunta di altre funzionalità assenti, partendo da quelle considerate opzionali nella proposta di progetto e poi passando a quelle non considerate per mancanza di tempo.

Abtin Saadat

Questo progetto mi è piaciuto molto perché di solito quando si studia si fa sempre poca pratica, invece qua abbiamo fatto qualcosa di pratico che si avvicina molto al mondo del lavoro (quasi un mezzo tirocinio). Non avendo mai fatto progetti di questo calibro sulla programmazione mi ha forzato a creare una routine mia di lavoro, ogni giorno vedevo i commit nuovi degli altri e

comunicavamo ogni tot i nostri progressi su Whatsapp (purtroppo non ci siamo mai visti dal vivo per il progetto) e nel caso che cosa ci serviva dalle altre persone (raramente dato che le parti erano abbastanza indipendenti tra di loro, tranne per la fine dove le varie parti dovevano comunicare insieme).

Io mi sono occupato delle carte, e molte difficoltà che ho riscontrato erano nella fase di progettazione, ad esempio parte della view mia è stata progettata male e ci ho perso molto tempo, dovendola rifare. Ho usato anche molto del tempo in fase di progettazione per fare codice più coeso possibile con quello di altri, e cercare di scrivere meno codice doppio possibile. Ad esempio le azioni delle carte e delle caselle sono molto simili, quindi siamo riusciti a generalizzarle.

Vado fiero del fatto che il progetto è abbastanza espandibile, ad esempio nel caso di aggiunta di nuovi tipi di mazzi o di carte è molto facile da implementarle, anche perché la nostra versione di Talisman non è completa e si possono aggiungere molte meccaniche.

Appendice A: Guida utente

Avvio di una partita

Come prima finestra si apre un menù con i tasti start game e quit game che permettono di, rispettivamente, iniziare una nuova partita o uscire dal gioco.

Alla pressione di start game si apre una nuova finestra, dove ogni giocatore può scegliere il suo personaggio. Ogni giocatore che vuole partecipare deve premere su uno dei personaggi rimasti disponibili, finché tutti non hanno scelto oppure sono terminati i personaggi.

Dopodiché, premendo su start game, viene iniziata la partita vera e propria e viene aperta la finestra principale del gioco, ovvero il tabellone, insieme alla finestra che permette al giocatore che ha il turno di effettuare le azioni desiderate.

Interfaccia del tabellone

Essa mostra il tabellone del gioco, in particolare sono sempre mostrate le caselle e le pedine dei giocatori.

Spostando il mouse sopra una casella viene mostrato il suo nome e le azioni da compiere quando ci si posiziona con la pedina sulla casella stessa.

Inoltre, se, sulla casella su cui è posizionato il giocatore che ha il turno, è presente una carta, allora essa sarà mostrata sempre passando il mouse sulla casella. In quest'ultimo caso, cliccando con il tasto sinistro sulla casella, la carta rimarrà visibile a schermo, e sarà possibile interagire con i comandi che permettono di, in caso sia possibile, raccogliere la carta per posizionarla nel proprio inventario. Compiendo questa azione si ottiene il bonus specificato dalla carta.

Turno di gioco

La sequenza di un turno è la seguente:

1. Viene chiesto di tirare il dado per spostarsi, premendo sul bottone che presenta l'immagine di un dado;

2. Viene chiesto di spostare la pedina del numero di celle risultante dal lancio precedente in senso orario, premendo sul tasto “Move”;
3. Dopo lo spostamento, prima di lasciare all’utente la scelta dell’azione da compiere, vengono fatte alcune azioni automaticamente:
    a. Se sulla casella di destinazione è presente una carta mostro, allora viene iniziato il combattimento con esso (Vedi: Combattimento);
    b. Se la casella è quella finale, ovvero la casella centrale con la corona, allora il gioco termina e il giocatore è il vincitore;
4. L’utente potrà poi scegliere di effettuare una di due azioni:
    a. Eseguire l’azione descritta sulla cella premendo il tasto “Event”. Potrebbe essere mostrata a schermo, se necessaria, una ulteriore finestra che permetterà, tramite una serie di pulsanti, di scegliere un l’opzione tra quelle indicate dalla casella.
    b. Nel caso in cui sono presenti altri giocatori sulla medesima casella di destinazione, sfidare uno di essi premendo il tasto “Challenge” (Vedi: Combattimento). Verrà mostrata una finestra in cui saranno elencati i nomi dei personaggi disponibili per il duello. L’utente dovrà inserire il numero assegnato al personaggio che desidera combattere;
5. Dopo aver completato le azioni precedenti, l’utente potrà passare il turno al giocatore successivo premendo il tasto “Pass”;
    a. Quando viene passato il turno, se il giocatore possiede una quest attiva, viene controllato che essa sia completa. Nel caso lo fosse viene terminata consegnando un Talismano al giocatore e riportandolo alla casella dove ha ottenuto la quest;

Combattimento

Esso viene svolto all’interno della finestra della battaglia. La sequenza di completamento di un combattimento è la seguente:

1. Il primo giocatore effettua il lancio del dado, premendo sul tasto del dado, per capire quale sarà il proprio punteggio di attacco. Esso sarà mostrato nella parte inferiore della finestra, a fianco al pulsante;
2. Nel caso l’utente non sia soddisfatto del risultato esso può utilizzare un proprio punto “fato” per azzerare il risultato del dado, premendo il pulsante corrispondente, ovvero quello con l’icona gialla;
    a. Questo può essere fatto una sola volta per giocatore per combattimento e finché esso possiede punti fato;
    b. Se viene utilizzato il fato, allora sarà necessario fare un nuovo tiro del dado, come descritto nel punto 1;
3. Si passerà poi il turno all’altro personaggio con il tasto attacco, ovvero quello con l’icona blu. Dopo di chè il primo giocatore non potrà più cambiare il proprio punteggio.
4. Il secondo personaggio esegue le stesse operazioni dei punti 1, 2 e 3, nello stesso ordine;
    a. Se il secondo personaggio è un mostro e non un giocatore, sarà comunque richiesto eseguire manualmente le operazioni, ad eccezione dell’utilizzo del fato, in quando i mostri non ne possiedono. Questo perché si è voluto riprendere come, anche nel gioco da tavolo fisico, il giocatore effettua i lanci dei dadi per i nemici;

5. Quando entrambi hanno confermato l'attacco, il personaggio con il punteggio finale più alto vince il combattimento;
    a. Se il personaggio perdente è un giocatore allora esso perderà un punto vita (Vedi: Morte dei personaggi);
    b. Se il personaggio perdente è un mostro e il combattimento è cominciato per via della sua carta mostro pescata o presente sulla cella dove il personaggio si è posizionato, allora la sua carta verrà scartata;

Morte dei personaggi

Quando un personaggi termina i punti vita, allora il suo giocatore ottiene un nuovo personaggio, perdendo le carte nel proprio inventario e tornando alle statistiche iniziali, e deve ricominciare a giocare dall'inizio.

Appendice B: Esercitazioni di laboratorio

B.0.1 Alberto Arduini

Non ho seguito il laboratorio del corso, in quanto lo avevo già frequentato il precedente anno accademico.

B.0.2 Alice Girolomini

Non sono state svolte le esercitazioni in quanto ho frequentato il laboratorio nel precedente anno accademico.

B.0.3 Abtin Saadat

Non ho seguito il corso di questo anno perché avevo già seguito quello dell'anno scorso insieme al laboratorio e alle esercitazioni.

B.0.4 Enrico Maria Montanari

Non ho seguito il corso di laboratorio quest'anno per impegni di lavoro.

Bibliografia

Regolamento Base Talisman

https://www.giochiuniti.it/wp-content/themes/giochiuniti/files-supporto/Talisman/TALISMAN_Regole_Base_Regolamento-web.pdf