Prova Finale Ingegneria del Software

Diagramma UML del Modello di Gioco (Codex)

Descrizione della modellazione

Gruppo AM-52, referente: Bordignon Livio

3-26-2024

CONTENTS

PANORAMICA MODELLO

Le classi definite per la modellazione del gioco (*Codex*) sono state costruite allo scopo di riprodurre la funzionalità del gioco stesso, in modo il più possibile indipendente dall'implemetazione della *view*. In particolare, tutte le classi espongono un'interfaccia pubblica che permette ai vari componenti del gioco di comunicare tra loro, consentendo di sviluppare la funzionalità globale del gioco stesso, ma tenendo completamente nascosta l'implememtaizone interna della propria specifica funzionalità. Inoltre, le classi principali, che saranno gestite dal *controller*, offrono una serie di metodi pubblici che permettono al *controller* di inviare messaggi al modello e, a sua volta, ricevere informazioni sullo stato del gioco.

Seguendo questo approcio, alcuni elementi del gioco non affatto sono modellati nel codice del modello, come ad esempio la lavagna segna-punti e le pedine: essi non sono parte della funzionalità del gioco, ma solo uno strumento visivo per segnalare il punteggio di ogni giocatore. Nel modello, è sufficiente avere l'informazione del conteggio dei punti di ogni giocatore. Sarà la *view* ad implememtare la parte grafica (o testuale) per la visualizzazione. Anche le carte di gioco, nel modello, implementano solo la propria funzionalità dal punto di vista della dianamica di gioco (ad esempio assegnazione punti, risorse visibile, angoli visibili, ecc...) ma non implementano nessuna proprietà grafica. Sarà sempre compito del codice della *view* implementare la garfica (o il testo) da visualizzare. Con la medesima prospetiva, anche le carte obiettivo non sono modellate come carte (cioè non fano parte della gerarchia della classe base Card), in quanto dal punto di vista della loro funzionalità non sono delle carte da gioco (non si posizionano sull'area di gioco, non si pescano, non hanno risorse visibili, non hanno angoli, ecc...), anche se nel codice della *view* esse saranno visualizzate con l'immagine di una carta. In altre parole, la classi che saranno sviluppate nella view saranno classi diverse da quelle definite nel model. Questa totale separazione e indipendenza consente una più facile manutenzione e test del codice ed eventuali restyling grafici del gioco senza impatti sulla funzionalità.

INTERFACCIA CON IL CONTROLLER

Le due classi, di più alto livello, che sono gestite dal controller sono (vedi Figura 1):

- GameLobby: modella la lobby pre-partita, in cui si raccolgono i giocatori prima di iniziare una nuova partita. Ad essa è delegata la funzionalità di definire il numero di giocatori e di controllare l'univocità del nickname di ogni giocatore
- GameManager: gestisce la stato del gioco, esponendo al controller i metodi per le azioni di gioco di ogni gocatore e per ricavare le informazioni sullo stato del gioco stesso

Il flusso logico, per l'avvio di una partita, consiste nella creazione della lobby da parte del giocatore che vuole creare una partita, il quale definisce anche il numero massimo di giocatori (da due a quattro). Successivi giocatori che si connettono vanno a riempire la lobby, specificando il proprio nickname. Quando la lobby è completa, il *controller* instanzia un oggetto GameManager, il quale al suo interno crea tutti gli oggetti necessari allo svolgimento del gioco e gestisce la dinamica della partita. Il *controller*, monitorando l'istanza della classe GameManager, può gestire la comunicazione tra *model* e *view*.

MODELLAZIONE DEL GIOCO

La funzionalità del gioco è costrutita con una serie di classi, ciascuna delle quali incapsula una funzionalità ben precisa. Le principali classi sono le seguenti:

- Player: modella un giocatore che partecipa alla partita, mantenendo e gestendo al suo intero le proprietà del giocatore, come la propria mano (tre carte), il punteggio, ecc... La classe Player al suo interno usa altre classi base per gestire il comportamento del giocatore
- Deck: modella un mazzo di carte (iniziali, oro o risorse), offrendo la funzionalità di mescolare, pescare, ecc...
- PlayingBoard: modella l'area di gioco di ciascun giocatore, con la possibilità di posizionare una carta, ottenere informazioni sulle risorse e gli oggetti visibili, ecc...

- Card: classe base per le carte da gioco, che sono le carte iniziali, le carte oro e le carte risorse. Una carta è un oggetto che ha due faccie (frontale e posteriore), che può essere posizionata nell'area di gioco
- CardFace: rappresenta una faccia di una carta da gioco, che ha alcune caratteristiche come gli angoli visibili, le risorse/oggetti visibili, un punteggio che si ottiene posizionando la carta sull'area di gioco, ecc...
- Objective: rappresenta un obiettivo di gioco, che fornisce al giocatore un puntaggio a fine partita

CLASSE PLAYER

La classe Player (vedi Figura 2) rappresenta un giocatore. Al suo interno crea e mantiene (come campi privati) tutti i componenti che servono per gestire un giocatore, come l'area di gioco (PlayingBoard), il proprio obiettivo segreto (Objective), una lista di carte (KingdomCard) che rappresenta la sua mano, un contatore del punteggio, ecc... I metodi pubblici consentono alla classe GameManager di gestire il giocatore durante lo svolgimanto della partita, comandando le azioni quali posizionare una carta nell'area di gioco, pescare una carta, ecc...

MODELLAZIONE DELLE CARTE DA GIOCO

Come accennato in precedenza, nel modello di gioco, una carta rappresenta un oggetto che può essere posizionato sull'area di gioco di un giocatore, pertanto ricadono in questa categoria le carte iniziali, le carte risorsa e le carte oro, ma non le carte obiettivo. Ogni carta ha associato un identificatore numerico (int) univoco. Esso costituisce l'informazione trasferita tra *model* e *view* (tramite *il controller*), per identificare una carta. La classe base di ogni carta è la classe astratta Card (vedi Figura 3), che offre un piccolo insieme di funzionalità comuni a tutte le carte, come ottenere l'id della carta le sue due faccie. Le carte sono poi suddivise in due sottoclassi, StarterCard (carte iniziali) e KingdomCard (carte risorse e oro). Le carte iniziali sono particolari, in quanto un giocatore ha solo una carta iniziale, che deve essere posizionata all'inizio della partita. Le carte risorse e oro fanno pate della famiglia Kingdom card, che rappresenta la famiglia di carte da gioco effettive, ognuna con un proprio regno (kingdom) associato. Dalla classe KingdomCard derivano infatti le specifiche classi GoldCard e ResourceCard che rappresentano le carte oro e risorsa. Tutte le classi della gerarchia sono definite come classi astratte, ma le classi StarterCard, GoldCard e ResourceCard hanno dei metodi statici per ottenere le istanze delle carte da gioco:

- *getCards()*: ritorna la lista completa delle carte
- *getCardsWithId()*: ritorna la carta di cui è specificato il proprio id

La vera funzionalità di ogni carta è legata alla faccia della carta, dato che un gioctaore può posizionare ogni carta nell'area di gioco scegliendone la faccia visibile. La gerarchia di classi che modella la faccia della carta ha una classe base astrattta comune chiamata CardFace (vedi Figura 4). Questa classe offre l'interfaccia base della faccia della carta, cioè la presenza degli angoli visibili o coperti, per collegare tra loro le carte sull'area di gioco, e la presenza di risorse e/o oggetti visiblili. A partire dalla classe base la gerarchia si divide in StarterCardFace e KingdomCardFace, cioè anche la faccia delle carte è divisa tra le carte iniziali e le carte da gioco vere e proprie (risorse e oro). Infatti la carta iniziale ha un comportamento speciale: essa può essere sempre e solo posizionata come prima carta sull'area di gioco e non porta punti al giocatore. Le carte da gioco, invece, possono avere dei vincoli di posizionamento (risorse necessarie) e possono far ottenere dei punti al giocatore, quando posizionate nell'area di gioco. Inoltre, la carta iniziali non è mai considerata nel calcolo degli obiettivi legati alla disposizione delle carte. La classe KingdomCardFace definisce i seguenti metodi astratti, che sono implementati dalle sottoclassi in base alla propria specifica funzionalità:

- *boolean canPlace(BoardInfo board)*: verifica se la carta può essere posizionata sull'area di gioco, cioè se eventuali risorse richieste sono disponibili sull'aera di gioco stessa
- *int gainedPoints(BoardInfo board, int linkedCorners)*: calcola il puntegio ottenuto dal giocatore, quando la carta è posizionata sulla propria area di gioco.

MODELLAZIONE DELL'AREA DI GIOCO

L'area di gioco di ciascun giocatore è modellata con una classe chiamata PlayingBoard, con il supporto di una classe BoardSlot (vedi Figura 5). Quest'ultima rappresenta una generica casella dell'area di gioco, su cui potrebbe essere posizionata una carta da gioco. Ogni casella è semplicemente definita da una coppia di indici interi, che definiscono le coordinate orizzontale e verticale della casella. La casella (0, 0) è la casella di origine, in cui può essere solo posizionata la carta iniziale. Infatti la classe PlayingBoard definisce un costruttore pubblico che accetta come parametro la faccia della carta iniziale e quindi inizializza l'istanza dell'area di gioco con tale faccia posizionata in posizione (0, 0).

La classe PlayingBoard offre una serie di metodi pubblici per posizionare una nuova carta, richiedere l'elenco di posizioni disponibili per posizionare una nuova carta (solo carte kingdom sono accettate), richiedere l'elenco di tutte le carte già posizionate, le risorse visibili, gli oggetti visibili. Ogni volta che una nuova carta è aggiunta, attraverso il metodo placeCard(), lo stato interno della classe viene aggiornato. Molti metodi sono definiti in un'interfaccia BoardInfo, che espone solo dei metodi osservatori. In questo modo:

- un riferimento di tipo BlayingBoard può essere passato come riferimento di tipo BoardInfo, in modo che tale riferimento non possa essere utilizzato da chi lo riceve per modificare l'area di gioco
- la calsse PlayingBoard, tramite il metodo getInfo(), crea un'oggetto immutabile che implementa l'interfaccia BoardInfo, che può essere usato per ottenere informazioni sullo stato dell'area di gioco, con la sicurezza che esso è immutabile (è una fotografia dell'area di gioco).

CLASSI RANDOMDEALER<T> E DECK

La classe RandomDealer<T> è una classe generica che può essere tipizzata con qualsiasi tipo di dato. Offre un costruttore pubblico che accetta una List<T>, i cui elementi venogno tutti copiati (*shallow copy*) in una lista interna. A quel punto gli elementi della lista interna possono essere richiesti uno ad uno, con il metodo getNextItem(). Appena viene creata un'istanza della classe RandomDealer<T>, la lista al suo interno è ordinata come la lista di origine passata al costruttore. É possibile mescolare in modo casuale l'ordine degli elementi invocando il metodo init(). Il metodo hasNext() indica se si possono richiedere altri elementi e il metedo getItemsCount() ritorna il numero di elementi che possono essere ancora richiesti. In qualsiasi momento, è possibile invocare il metodo init() per rimescolare tutti gli elementi e rimetterli nuovamente a disposizione. É disponibile un overload del costruttore che accetta un seme (un numero intero), in modo che l'invocazione del metodo init() rimescoli la lista sempre allo stesso modo.

La classe Deck non è altro che una wrapper class che contiene al suo interno un'istanza RandomDealer<T> (vedi Figura 6), configurata per gestire una lista di carte. La classe Deck mette a disposizione alcuni metodi statici per ottenere le istanze dei mazzi per le carte iniziali, risorsa e oro. I metodi esposti dalla classe Deck, per pescare una carta, mescoare le carte, ecc... delegano la relativa azione all'istanza RandomDealer<T> mantenuta all'interno della classe Deck.

Per la distribuzione delle carte obiettivo ad inizio gioco, la classe GameManager fa uso di un'istanza della classe RandomDealer<Objective>.

CLASSI OBJECTIVE E PATTERNFINDER

La classe Objective rappresenta un obiettivo del gioco, che fornisce punti addizionali al giocatore a fine partita (punti bonus). Essa offre un metodo pubblico calculatePoints() che riceve in ingresso un unico parametro di tipo BoardInfo e calcola il punteggio ottenuto dall'obiettivo. Per il calcolo dei punti, la classe Objective mantiene al suo interno un'istanza della classe PatternFinder (vedi Figura 7). Questa classe rappresenta la logica legata all'obiettivo (per esempio la diagonale, le coppie di oggetti, ecc...) e offre un metodo pubblico findPatterns() che calcola quanti obiettivi sono stati raggiunti sull'area di gioco. In pratica la classe Objective delega alla classe PatternFinder il calcolo di quanti obiettivi siano stati ragiunti e poi calcola i punti ottenuti dal giocatore semplicemente moltiplicando tale numero per il punteggio bonus del singolo obiettivo.

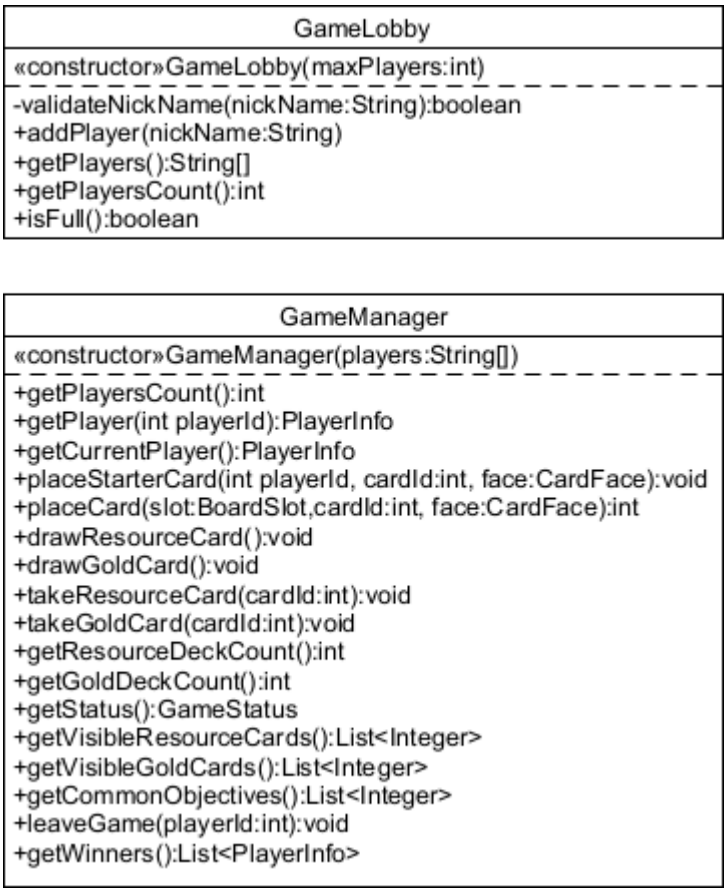

Figura 1 – **Classi GameManager e GameLobby.**

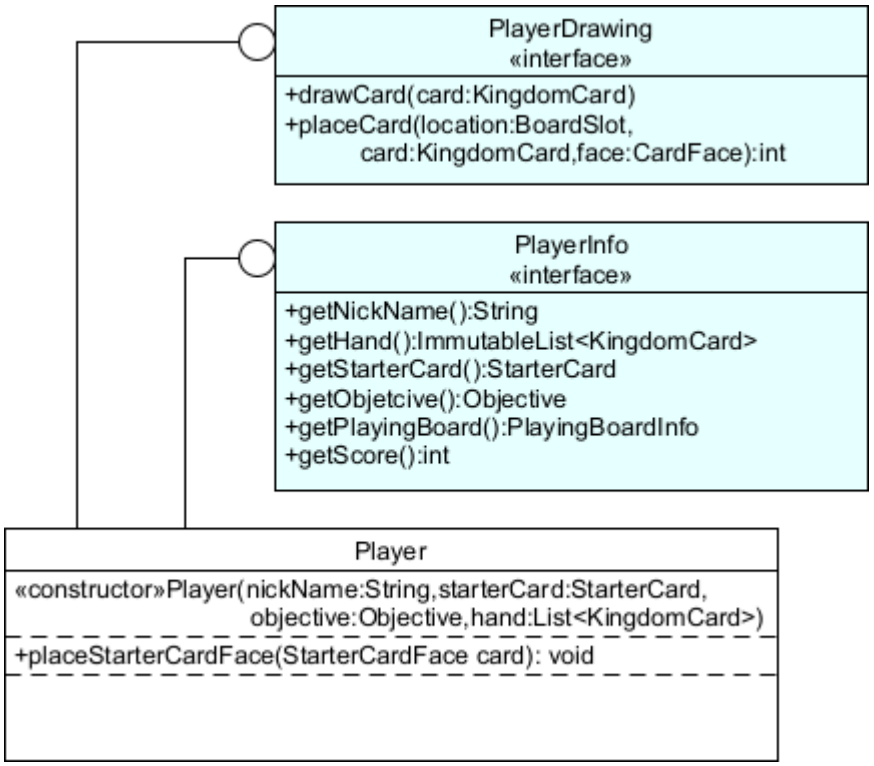

Figura 2 – ***Classe Player.***

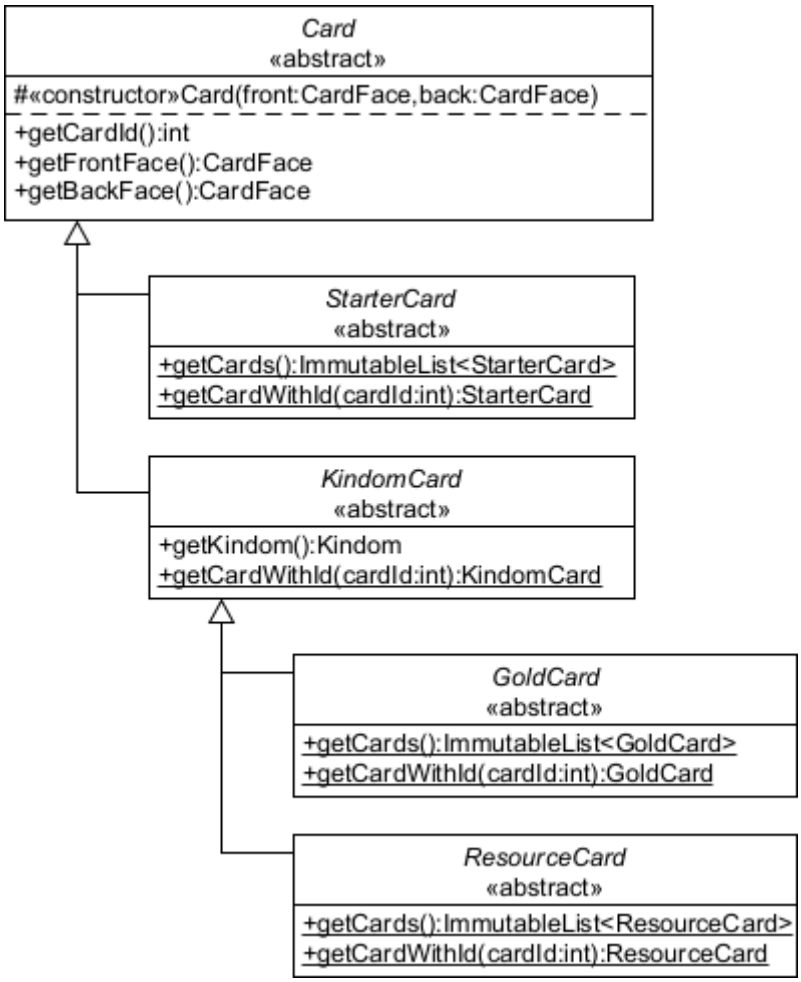

Figura 3 **– Gerarchia di classi per modellare le carte di gioco.**

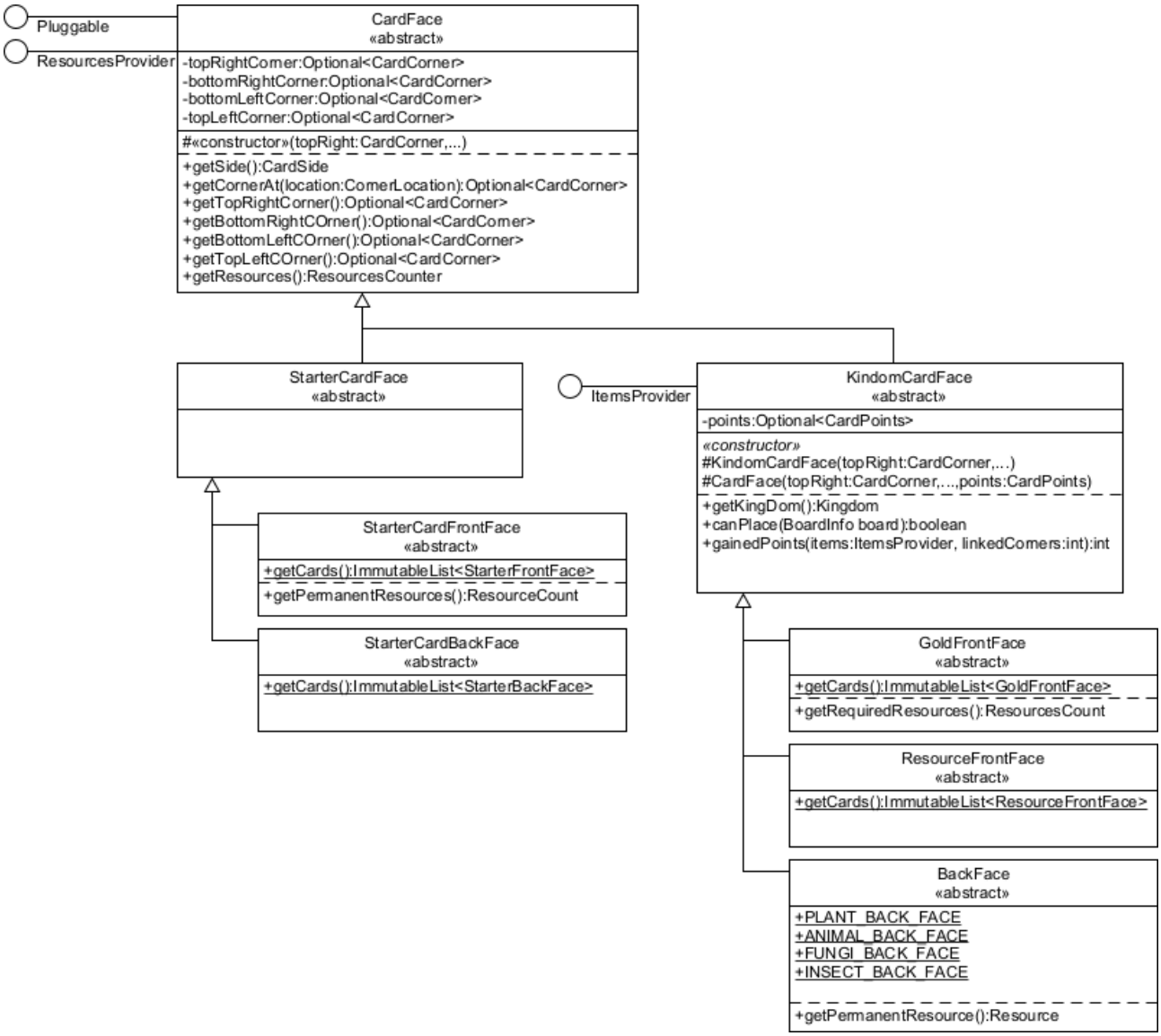

Figura 4 – **Gerarchia delle classi che modellano la faccia di una carta.**

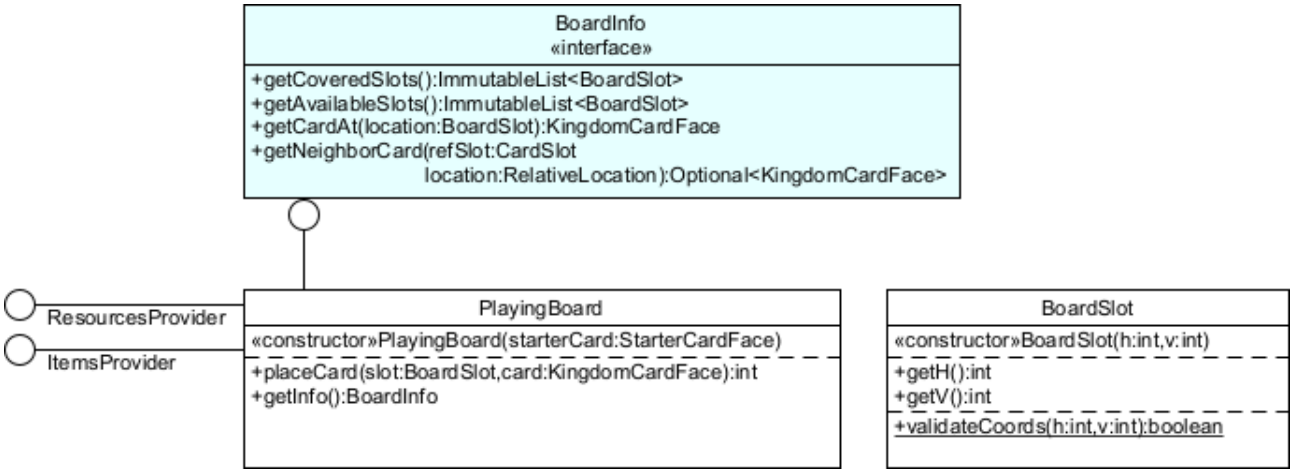

Figura 5 – **Classi PlayingBoard e BoardInfo.**

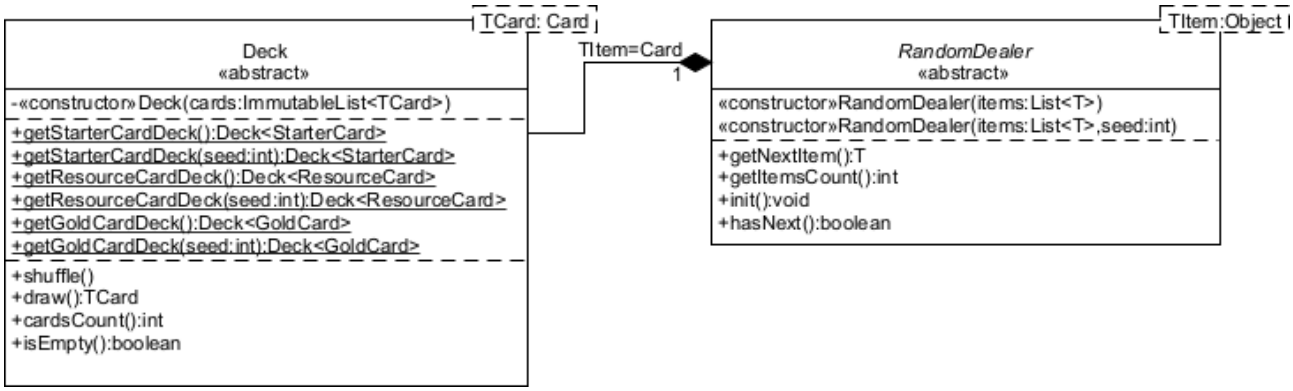

Figura 6 – **Classi RandomDealer e Deck.**

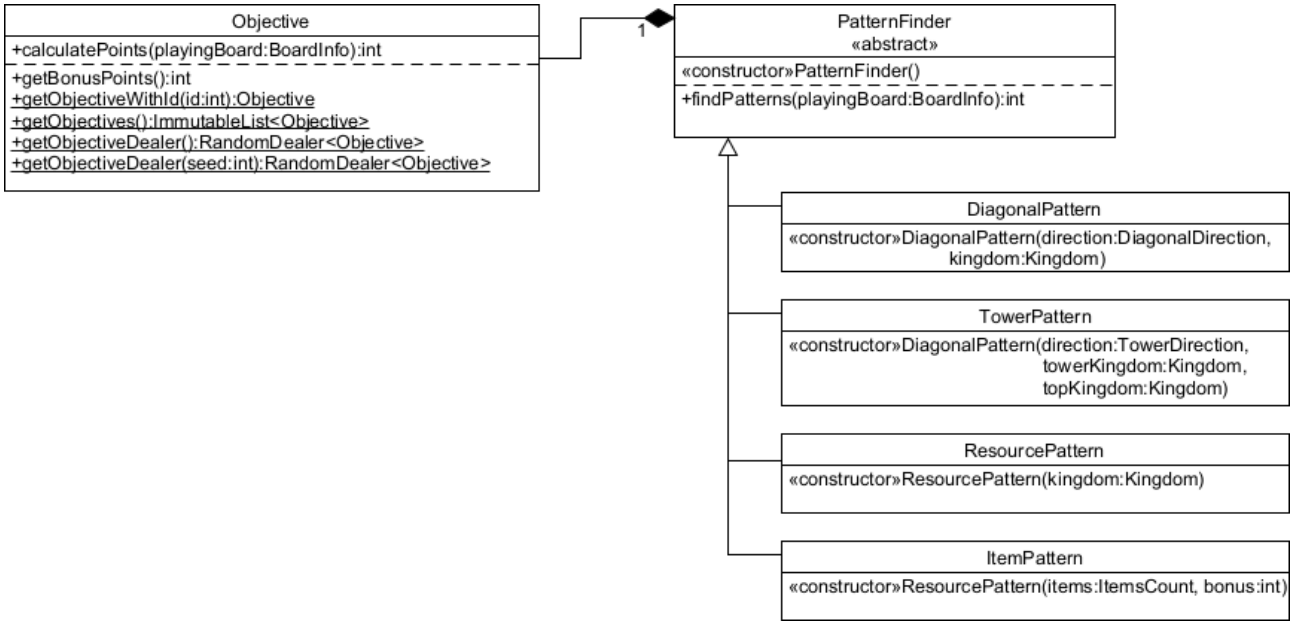

Figura 7 – **Classi Objetcive e PatternFinder**