Ingegneria del Software A.A. 2016/2017 Esame 2017-08-29

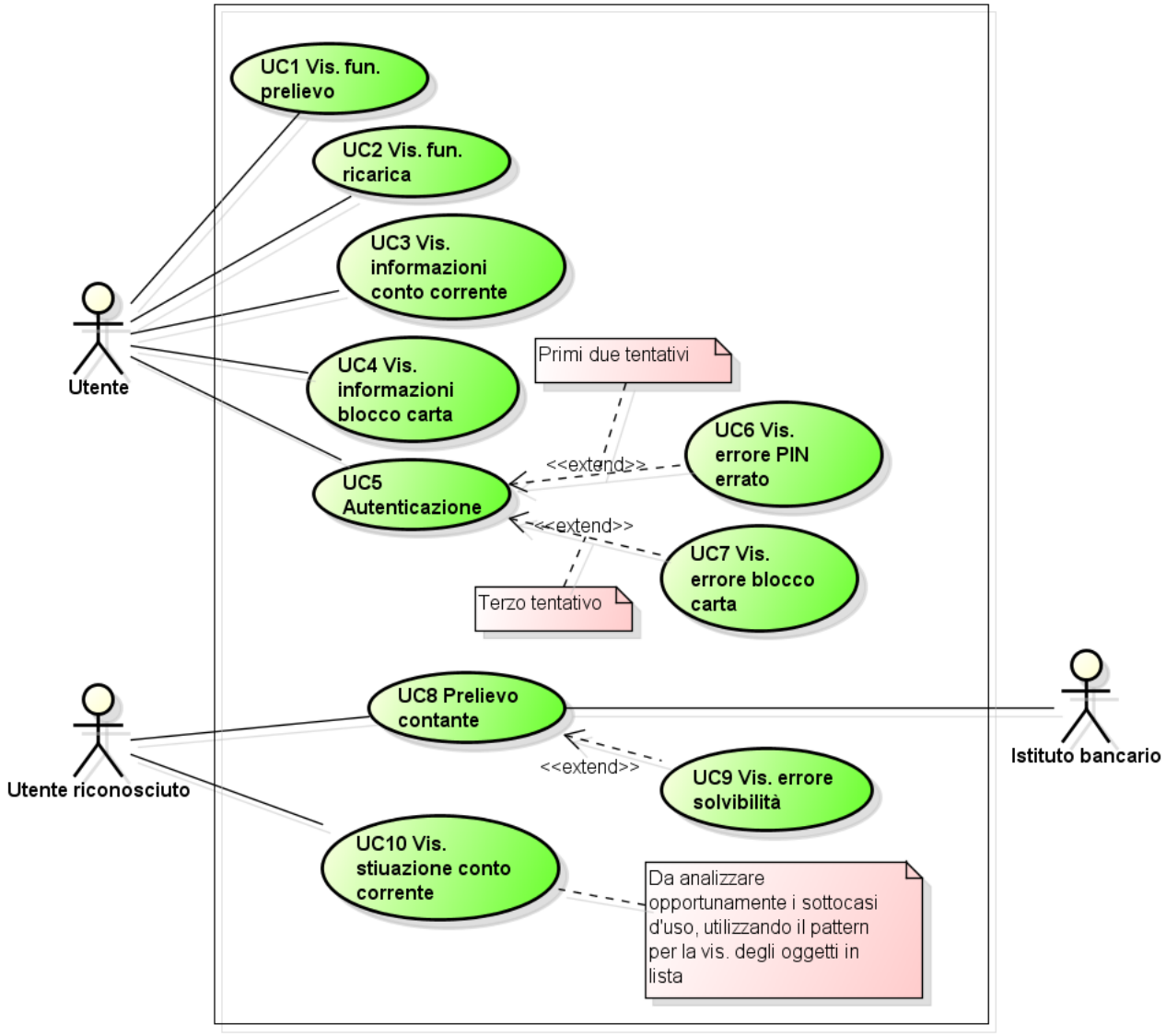
Esercizio 1 (6 punti)

Descrizione

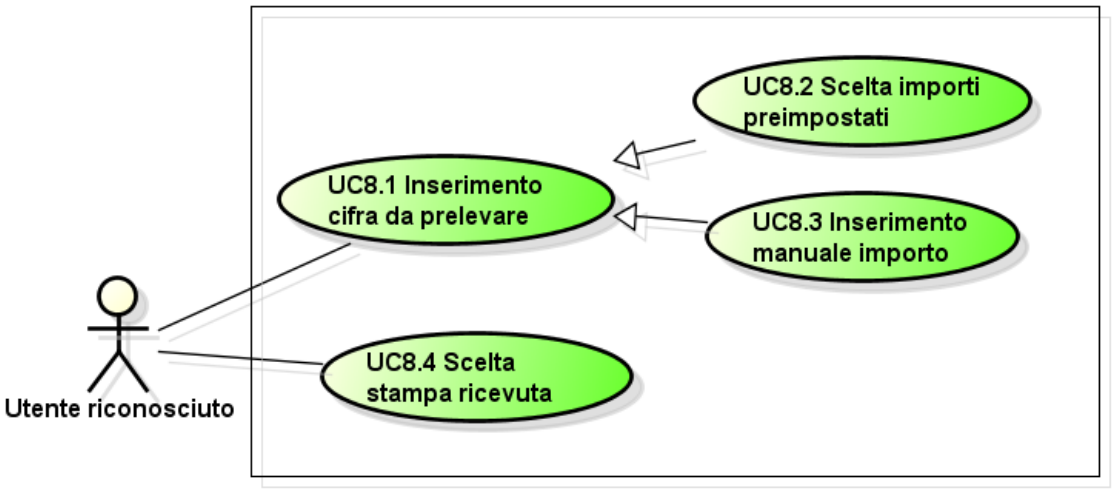
Lo sportello ATM (“bancomat”) può essere visto come una semplice interfaccia esposta verso un utente. Inizialmente, essa presenta funzionalità quali il prelievo, la ricarica telefonica, la visualizzazione della situazione del conto corrente aperto presso quell’istituto, e le istruzioni per bloccare la propria carta di debito/credito in caso di furto o smarrimento. L’accesso a tali funzionalità richiede l’inserimento di un PIN di 5 cifre. Fino a due errori di immissione del PIN in una sessione di utilizzo, l’interfaccia visualizza un messaggio di errore. Al terzo errore, il sistema blocca la tessera bancomat dell’utente, visualizzando la notifica corrispondente. Il prelievo di contante richiede all’utente di specificare la cifra da erogare: è possibile scegliere tra proposte preimpostate oppure inserire un altro importo usando un apposito tastierino. L’ATM dialoga poi con l’istituto bancario di appartenenza dell’utente per effettuare verifiche di solvibilità dell’importo. In caso di risposta negativa, l’interfaccia ATM visualizza un opportuno messaggio di errore. Contestualmente al prelievo, l’utente può richiedere la stampa di una ricevuta. La funzionalità di visualizzazione della situazione del proprio conto corrente, mostra a video gli ultimi 10 movimenti effettuati su di esso. Per ciascun movimento in lista, la funzionalità visualizza l’importo, il beneficiario (per le uscite) o l’ordinante (per le entrate), e la data di esecuzione del versamento.Si utilizzino i diagrammi dei casi d’uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d’uso individuati.

Soluzione

I casi d’uso indicati come “da analizzare” devono essere opportunamente riportati su un diagramma dei casi d’uso.



Il caso d'uso UC8 individua i seguenti sotto-casi d'uso.



Esercizio 2 (7 punti)

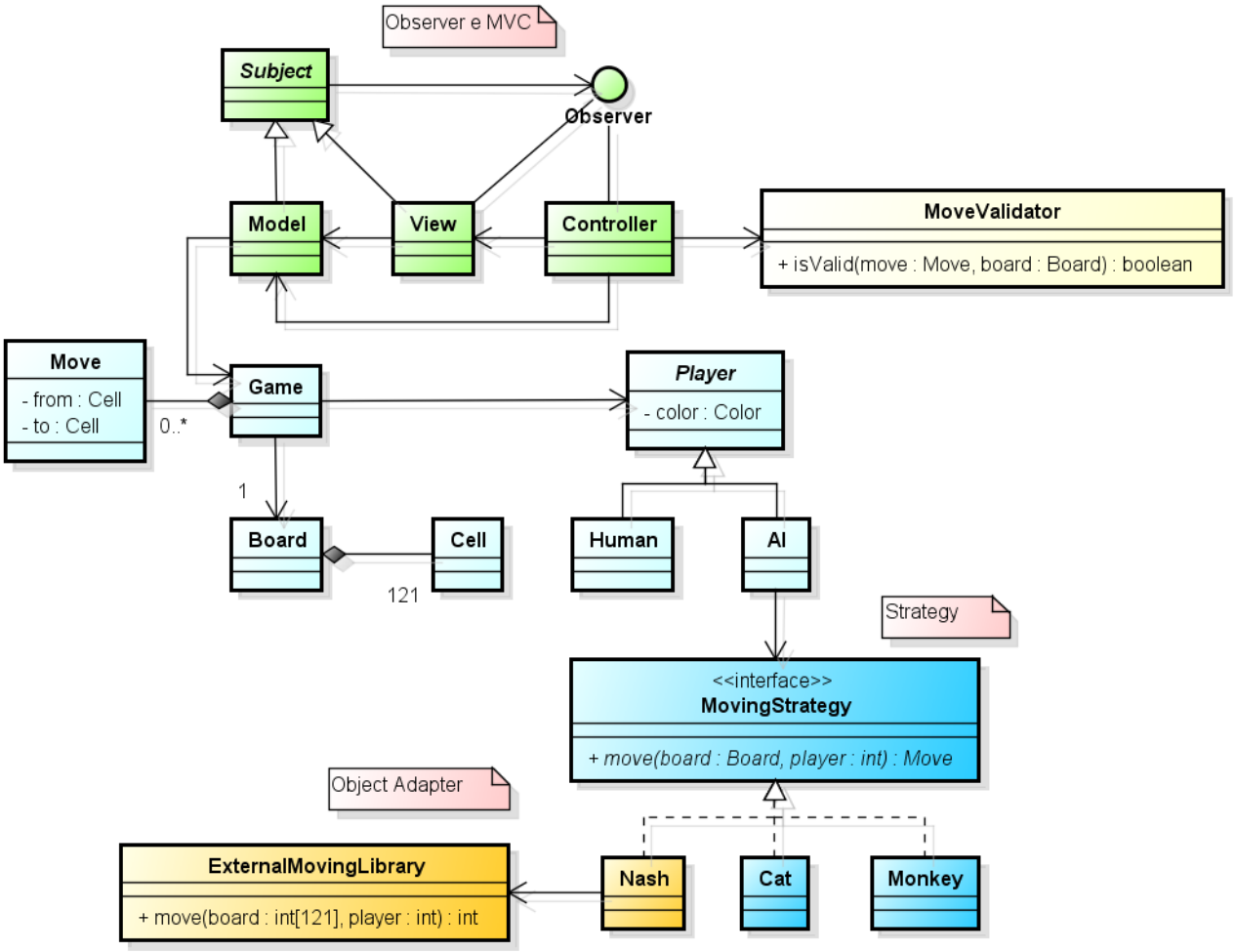
Descrizione

Hex è un gioco su scacchiera con celle esagonali re-inventato dal famoso matematico John F. Nash quando era studente a Princeton. La scacchiera ha forma di rombo con 11 celle per lato e pedine di due colori, uno per giocatore. All'inizio del gioco, ciascuno dei due lati opposti del rombo contiene pedine dello stesso colore, in modo tale che ogni giocatore occupi lati non contigui del rombo. Durante il gioco, ciascun giocatore, a turno, posiziona le proprie pedine sulle scacchiera, cercando di costruire un percorso continuo che unisca i due lati del proprio colore. Si immagini di dover progettare questo sistema di gioco, fornendo all'utente la possibilità di giocare contro un altro umano o contro una intelligenza artificiale (AI) di tipologia *monkey*, *cat*, *nash*, corrispondenti ad abilità crescenti. Tali tipologie di AI utilizzano una rappresentazione della scacchiera a matrice di celle, di tipo `Cell`. Per implementare il livello *nash* si desidera utilizzare una libreria esterna pre-esistente che espone un metodo `move(board: int[121], player: int): int`, che data una scacchiera linearizzata e l'indicazione del giocatore che deve muovere, ritorna la cella in cui porre la pedina. Ogni mossa deve essere validata opportunamente, in modo da garantire il rispetto delle regole di gioco. Per gli utenti umani, il sistema deve disporre di una interfaccia grafica che permetta loro di scegliere visivamente le proprie mosse.

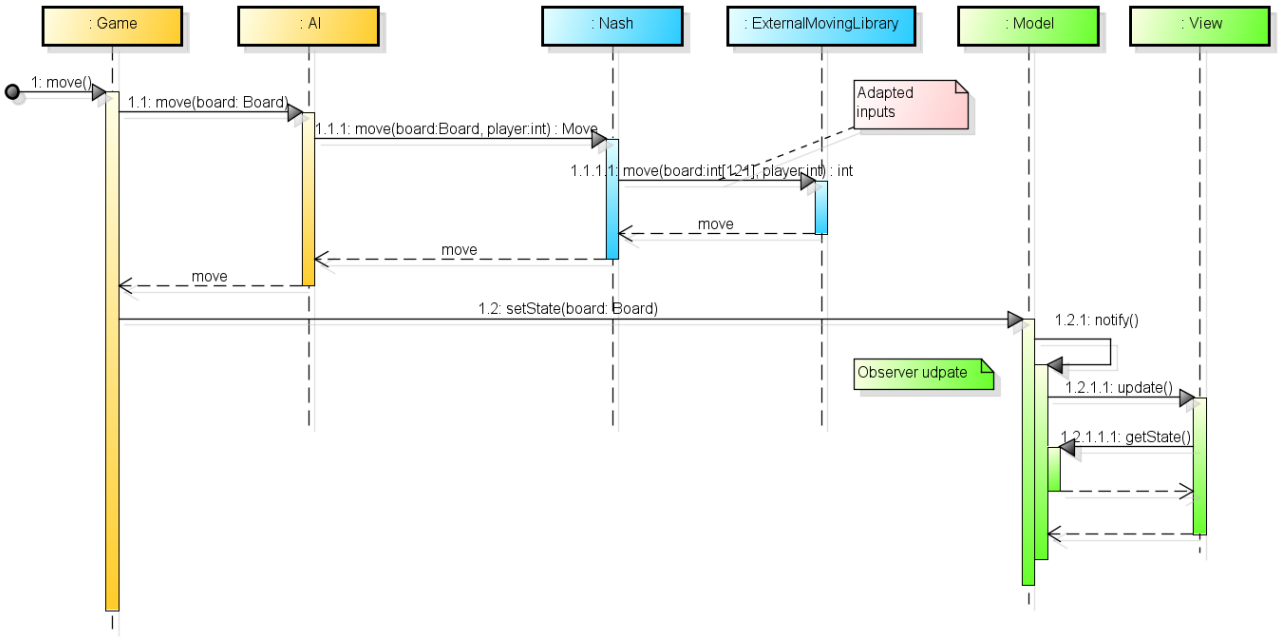
Si modelli il sistema descritto mediante un diagramma delle classi che utilizzi opportuni design pattern. Mediante un diagramma di sequenza, si descrivano poi le azioni corrispondenti alla mossa da parte di un'AI di tipo nash, fino al posizionamento della pedina scelta sulla scacchiera.

Soluzione

La soluzione prevede l'utilizzo dei seguenti design pattern: Observer, MVC, Strategy e Object Adapter.



Un esempio di diagramma di sequenza che modella le invocazioni necessarie per completare una mossa proveniente dall'AI di tipo Nash il seguente.



Esercizio 3 (3 punti)

Descrizione

L'utilizzo del paradigma di programmazione reactive punta a ridurre il numero di operazioni sincrone (bloccanti) utilizzate, ritenendo che l'impiego di operazioni asincrone sfrutti meglio (senza attese) le risorse di calcolo a disposizione del programma. Una forma basica di questo paradigma utilizza funzioni di callback, come, per esempio, nel framework *Node.js*. Si assuma un programma reactive contenente una classe `UserRepository`, che esponga il metodo `findById(id: String): User`, che recupera da una base dati l'utente contraddistinto da un particolare id, e una classe `UserController`, che esponga il metodo `getUser(id: String): String`, che ritorna una rappresentazione in formato JSON di un utente con identificativo id.

Utilizzando un diagramma di sequenza, si modelli il recupero del JSON associato all'utente con id "`rcardin`", implementando un approccio che preveda l'invocazione di almeno una funzione di callback. Si ricordi che, in un linguaggio a oggetti, essa può essere rappresentata con un pattern Strategy, ossia come implementazione di una functional interface, cioè un'interfaccia che espone un solo metodo. È possibile aggiungere ulteriori tipi al diagramma e/o modificare le firme dei metori esistenti.

Soluzione

Una delle possibili soluzioni prevede l'utilizzo di un ulteriore livello di indirezione, ossia il tipo `UserService`. Si noti la natura asincrona dei messaggi tra il tipo controller e il tipo service.