INGEGNERIA DEL SOFTWARE: MONOPOLY

Documentazione del progetto “Elaborato 2014/2015”

Università degli Studi di Brescia

Dipartimento di Ingegneria dell'Informazione

Via Branze 38

25231- Brescia

Relazione dell'elaborato, parte I

Autori

Nome: Falleti Davide
Matricola: 85973

Nome: Mele Fabrizio
Matricola: 85971

Nome: Cordioli Francesco
Matricola: 86538

Sommario

Ingegneria del Software

Monopoly

Introduzione

Il progetto ha l'obbiettivo di realizzare tramite un processo di sviluppo incrementale un'applicazione software che consenta di giocare a Monopoly.

Accesso ai dati statici: il database

La prima scelta progettuale affrontata dal gruppo è stata come conservare le informazioni sulle caselle, le carte ed i giocatori; la scelta è ricaduta su un sistema **DBMS.** Conservare le informazioni in un database è una maniera efficace di evitare l'hardcoding; in questa maniera possiamo consultare e modificare l'elenco delle caselle, delle carte imprevisti e probabilità, senza dover mettere mano al codice sorgente (e senza dover rigenerare l'eseguibile). Inoltre l'utilizzo di un database permette di prevedere la futura aggiunta di funzionalità, come ad esempio un menù di configurazione, un certo numero di giocatori "abituali" predefiniti.

Nello specifico si è scelto di utilizzare un database **SQLite**; questo DBMS è stato scelto per la sua portabilità e per la sua semplicità. Non ci è sembrato il caso di implementare soluzioni con MySQL o PostgreSQL, vista la quantità minima di informazioni da immagazzinare nel database.

Il primo passo nella progettazione della struttura delle classi è stato prevedere la creazione di **un'interfaccia** Database-Applicazione, o meglio un *wrapper*, a cui l'applicazione delegasse tutti i compiti di recupero e modifica di dati nel database. L'impiego di una classe wrapper ci ha permesso di ridurre al minimo la dipendenza del pacchetto software dalla piattaforma di gestione dei dati, permettendo in futuro di cambiarla, ad esempio impiegando dei file XML, effettuando modifiche solo nel codice del wrapper. Di fatto il wrapper è una classe con due metodi pubblici: uno per le caselle, uno per le carte probabilità e imprevisti. A ciascuno di questi metodi spetta anche l'interpretazione dei dati testuali presenti nel database; al chiamante vengono ritornati degli oggetti *ready-to-use.*

Release 1: Il modello Tabellone

Dopo aver definito i metodi di accesso ai dati il gruppo si è rivolto ad uno dei problemi principali del progetto: la modellizzazione del oggetto fisico "Tabellone" in un insieme di classi. La linea progettuale che abbiamo cercato di mantenere è stata quella di non discostare mai il software dal modello fisico, salvo ovvie eccezioni dovute al passaggio dal mondo reale al mondo della programmazione orientata agli oggetti.

Il Tabellone

Il tabellone è stato modellizzato come oggetto Tabellone, contenente un array di oggetti Casella. Qualsiasi altra classe avesse avuto bisogno di interfacciarsi con il tabellone virtuale avrebbe dovuto dialogare solo e direttamente con il Tabellone stesso. Nella prima release questo utilizzo è stato limitato semplicemente al susseguirsi dei giocatori attorno al tabellone. Per permettere agli oggetti dipendenti di gestire in maniera efficace la partita è stato necessario fare in modo che allo spostamento del giocatore "oltre" l'ultima casella del tabellone conseguisse una sua immediata ricomparsa sulla prima casella, a simulare la circolarità delle caselle. Questa richiesta è stata implementata operando una semplice sottrazione nel caso l'intero che rappresenta la casella di destinazione fosse maggiore del numero totale di caselle, es: casellaCorrente = 35; avanzamento = 10; casellaFutura = ~~45~~ 45-40 = 5.

Inoltre dal metodo di avanzamento è stata estratta l'operazione di spostamento vero e proprio, in previsione di un futuro accesso diretto nell'evenienza di esigenze di gioco (i.e.: carte che impongono l'attraversamento del tabellone per arrivare a caselle specifiche).

La caratteristica principale della classe Casella a questo punto dello sviluppo era la conservazione del suo **stato**, ovvero la memorizzazione dei giocatori presenti nell'istante corrente sulla casella stessa. Questa scelta è dovuta alla linea progettuale perseguita, ovvero l'adesione, per quanto possibile, della realtà virtuale alla realtà fisica. L'interpretazione di questa direttiva limitatamente allo stazionamento scaturisce dall'osservazione, banale ma necessaria, che le pedine fisiche sono poste fisicamente sulle singole caselle.

Il turno di gioco

La classe principale dell'applicazione è Monopoly. In essa si trova l'interfaccia verso l'utente, consistente in un menù testuale. Dopo l'inserimento dei giocatori è possibile dare il via ad una partita. Il comando "gioca" presente nel menù, avvia una serie di azioni:

1. Viene verificata la presenza di almeno 2 giocatori, in caso contrario si ritorna al menù principale
2. Vengono mischiati i giocatori
3. Viene creata una nuova Partita:
    3.1. Viene creato un nuovo Tabellone:
        3.1.1. Tabellone chiede al DBManager l'elenco delle Caselle
    3.2. Vengono posizionati tutti i giocatori sulla casella 0

A questo punto inizia la fase di turnazione: all'interno di un ciclo viene chiamato ripetutamente il metodo Partita.turno(). Il metodo di turno risiede nella Partita per separare la logica dell'interfaccia, appartenente a Monopoly, dalla logica di gioco, appartenente a Partita.

All'interno del turno:

1. Viene chiesto al giocatore di lanciare i dadi
    1.1. In caso di dadi doppi si memorizza l'accaduto in una variabile booleana e si incrementa un contatore

1.2. Nel caso il tiro di dadi doppi appena effettuato sia stato il terzo, si sposta direttamente il giocatore in prigione
2. Calcolato lo spostamento vengono comunicati al Tabellone l'identità del Giocatore e l'entità dello spostamento.
   2.1. Il Tabellone rimuove il Giocatore dalla Casella corrente e lo posiziona nella nuova Casella.
3. Se il giocatore ha effettuato un tiro di dadi doppi si ritorna al punto 1.

Il lancio dei dadi

Il lancio dei dadi è un'azione che spetta al Giocatore; per questo motivo ciascun giocatore è stato dotato di un metodo lanciaDadi(), che si appoggia sul metodo lanciaDadi(int, int) della classe di utilità Util. L'esito del lancio viene ritornato come array di interi, in maniera da poter essere mostrato dall'interfaccia utente come due numeri separati.

Release 2: Prigione e capitale dei giocatori

Tra i requisiti della seconda release vi era un primo abbozzo di gestione del capitale dei giocatori, cioè l'assegnamento di un capitale iniziale, il "rifornimento" al passaggio dalla casella 0 («Via!»), e il pagamento delle due tasse fisse presenti nel gioco. Inoltre la release prevedeva la definizione di una casella speciale, «In Prigione!», che inviasse i giocatori stazionanti su di essa nella casella «Prigione», e la gestione del fallimento finanziario dei partecipanti.

Modifica del modello Tabellone

Freschi dal completamento della release 1 abbiamo subito affrontato il miglioramento del nostro modello di tabellone: qualcuno, nel diagramma delle classi, avrebbe dovuto accorgersi della sosta di un giocatore su una delle tre caselle che avrebbero dato luogo a degli effetti; e sempre qualcuno avrebbe dovuto accorgersi di quando un giocatore fosse passato dal Via. È subito apparsa chiaramente la necessità di modificare il meccanismo di spostamento casella per casella attorno al tabellone e di stazionamento su una casella. Una nuova osservazione di una partita reale di Monopoli ci ha fatto notare che ciascuno di noi usava "appoggiare" la propria pedina su ciascuna casella attraversata, per poi depositarla sulla casella di arrivo. Da qui l'ispirazione: il Tabellone virtuale di questa release avrebbe "appoggiato" la pedina su ciascuna Casella attraversata, e infine "depositato" la stessa sulla casella di arrivo. L'oggetto Partita, che è la classe che gestisce tutte le meccaniche di gioco, si sarebbe accorta dei transiti dal Via e delle fermate sulle tasse o sulla casella «In Prigione».

Per l'implementazione di questo nuovo modello ci siamo affidati al collaudato **design pattern Observer**: uno di noi, infatti, aveva già avuto esperienze di programmazione di applicazioni per Android, dove l'utilizzo di questo pattern è comune.

Nella nostra implementazione del pattern Observer:

— il ruolo di Subject sarebbe andato alla Casella, che avrebbe implementato due metodi, *hop* e *stop*, chiamati al momento del passaggio o della sosta dal Tabellone. Ogni casella avrebbe poi conservato in un attributo l'oggetto observer, implementante **l'interfaccia** MovementListener.
— il ruolo di Observer sarebbe andato alla Partita, che implementando l'interfaccia MovementListener avrebbe ereditato i metodi *onHop* e *onStop*, in cui avrebbe poi svolto determinate azioni a seconda della casella di transito o di sosta.

La registrazione dell'observer presso i vari subject avverrà nella fase di setup della partita; per questo è venuta comoda la creazione di un metodo Tabellone.setMovementListener che si occupasse di propagare il Listener ricevuto a tutte le Caselle.

La Banca

La classe Banca è stata sviluppata come raccolta di metodi statici: la non-necessità di contabilizzare entrate ed uscite e il suo patrimonio illimitato ci ha spinto ad interpretare la Banca come una raccolta di operazioni a cui la Partita si affida per completare versamenti, prelievi e trasferimenti. Di fatto, essendo i capitali dei giocatori semplicemente dei numeri interi, i metodi statici della Banca eseguono semplicemente somme o sottrazioni per conto della Partita.

Separare la logica "bancaria" dalla logica principale di gioco è stato fatto anche pensando ad una futura estensione della prima: sarà eventualmente possibile modificare solo la Banca senza intaccare la compatibilità con la Partita. Per esempio nel caso si voglia in futuro implementare una modalità di gioco in cui la Banca trattiene una commissione sui trasferimenti da giocatore a giocatore.

Il Fallimento

Per implementare la funzionalità di fallimento dei giocatori in caso di impossibilità di pagare il dovuto alla banca si è scelto di sfruttare il meccanismo di Exception proprio di Java. Abbiamo creato FallimentoException, sottoclasse di Exception, che sarebbe stata poi lanciata dai metodi di Banca nel caso di una richiesta di deposito o trasferimento superiore alle possibilità finanziare del giocatore. La "cattura" dell'eccezione sarebbe avvenuta da parte della classe Monopoly, cioè la classe principale dell'applicazione, a cui sarebbe spettata la rimozione dall'elenco dei giocatori del Giocatore fallito.

La prigione

Vista la modifica al modello del Tabellone implementare lo spostamento in prigione del giocatore capitato sulla casella «In Prigione!» (30) è stato del tutto simile all'implementazione del pagamento delle tasse: allo stop sulla casella 30 la Partita avrebbe chiesto al Tabellone di spostare direttamente, con un metodo apposito che non esegue gli "hop", il giocatore nella casella 10 («Prigione»).

Release 3: Proprietà ed affitti

Le proprietà

La terza release è stata incentrata sull'implementazione del meccanismo di compravendita delle proprietà e del pagamento degli affitti.

Il primo passo è stato aggiungere le funzionalità delle proprietà al modello Tabellone. Per conservare il modello Tabellone+Caselle è stato aggiunto un nuovo livello di astrazione al livello Caselle: la classe Proprietà rappresenta un'entità astratta, una generalizzazione delle classi Stazione, Società di Servizi e Terreno. In questa maniera abbiamo potuto assegnare a ciascuna Casella un oggetto Proprietà che rappresentasse la Stazione, il Terreno o la Società "stampata" sulla casella fisica.

Nel codice questo si è tradotto in Proprietà come superclasse di Stazione, Terreno e SocietàServizi; in ciascuna sottoclasse è stato fatto l'override dei metodi opportuni per implementare il calcolo dell'affitto di ciascun tipo di proprietà.

L'acquisto delle proprietà

Ad ogni stop del giocatore su di una Casella si controlla che oggetto Proprietà è collegato alla Casella: in caso questo sia nullo si assume che il giocatore si è fermato su una casella che non prevede acquisti o pagamenti, quindi non si fa nulla; in caso la Proprietà esista per quella Casella, si procede a verificare se questa abbia già un Giocatore proprietario:

- se sì si chiama il metodo calcolaAffitto della Proprietà, lasciando che sia il polimorfismo delle classi a fare il lavoro sporco. Se il giocatore proprietario è il giocatore stesso, ovviamente, non si fa pagare nessun affitto.
- se no, si esegue subito l'acquisto della Proprietà: se i fondi non sono sufficienti verrà catturata all'interno del metodo un'eccezione di tipo FallimentoException, verrà comunicata l'impossibilità di completare la transazione per fondi sufficienti, e si continuerà con il funzionamento corretto.

Il calcolo e il pagamento degli affitti

Per ciascun tipo di proprietà vi è una maniera diversa di calcolare l'affitto:

- Per le Stazioni l'affitto è quello standard: il 10% del valore della proprietà, percentuale che rimane invariata anche nel caso il giocatore possieda più di una stazione.
- Per i Terreni l'affitto è standard, con la differenza che raddoppia nel caso il giocatore possieda tutti i Terreni di quel colore.
- Per le Società di Servizi l'affitto è calcolato in maniera diversa: il numero ottenuto dal lancio di dadi moltiplicato per 4, in caso si possieda solo una società, o per 10, in caso se ne possiedano due.

Questa distinzione tra i diversi tipi di Proprietà è stata implementata impiegando il polimorfismo: la classe Partita chiamera sempre il metodo calcolaAffitto() della Proprietà, che sarà diverso a seconda del tipo di Proprietà assegnato al momento della costruzione del Tabellone.

Alla classe Giocatore sono stati aggiunti metodi con valore di ritorno booleano per verificare il possesso di due società di servizi o di tutti i terreni di un colore. Il colore è stato rappresentato con un enumerazione.

Per riuscire a calcolare agevolmente l'affitto di Terreni e Stazioni è stato salvato, all'interno dell'oggetto Giocatore, l'ultimo lancio da lui effettuato, in maniera da poterlo recuperare velocemente durante i calcoli dell'affitto.

Nel caso un giocatore non abbia fondi sufficienti viene sollevata, all'interno del metodo Banca.trasferimento(Giocatore, Giocatore, int), una FallimentoException; per specificare che il fallimento in questione deriva da una transazione da giocatore a giocatore viene aggiuno, all'eccezione, un collegamento al Giocatore che dovrà essere rimborsato dalla Banca della somma non pagata dal Giocatore fallito. La FallimentoException verrà intercettata, prima di risalire fino alla classe Monopoly, nel metodo Partita.turno(), dove la Partita si occuperà del rimborso e di rimuovere il Giocatore fallito dalle Proprietà possedute. Risalita alla classe Monopoly l'eccezione verrà fermata e verrà rimosso dal gioco il Giocatore fallito.

Release 4: Probabilità e Imprevisti

Nella release 4 si sono implementate le carte probabilità ed imprevisti. Per distinguere una Casella normale da una Casella Probabilità o Imprevisto sono state create due sottoclassi di Casella, CasellaProbabilità e CasellaImprevisto, che differiscono da Casella e tra loro solo per il nome.

I dati relativi alle carte sono stati immagazzinati nel database: a ciascuna carta corrisponde un'identificatore numerico ed una descrizione. L'identificatore numerico verrà impiegato dalla Partita per decidere che azioni eseguire al momento della pesca della carta.

Carte

Le Probabilità e gli Imprevisti sono conservati in Carte, un mazzo virtuale creato e mischiato ad inizio partita. Nel momento dello stop del Giocatore su una CasellaProbabilità o CasellaImprevisti la Partita chiede al mazzo di pescare una carta per conto del Giocatore. Il mazzo utilizza poi il pattern Observer, in maniera del tutto analoga al meccanismo di hop/stop, per notificare la Partita dell'avvenuta pesca e dell'identificatore della carta pescata.

La partita utilizzerà, nei metodi onProbabilità() e onImprevisto(), questo identificatore per determinare quali azioni compiere.

A scopo esemplificativo, si include il codice eseguito nel caso si peschi la carta probabilità #8, la cui descrizione recita: “È il vostro compleanno: ogni giocatore vi regala 25€”.

```
case 8:
    for(Giocatore giocatore: giocatori)
        if(!giocatore.equals(g))
            Banca.trasferimento(giocatore, g, PROB_8_REGALO);
```

Listato 1, esempio di Probabilità

Release 5: la prigione

L’aggiunta delle funzionalità di prigione è stata limitata ad alcune modifiche alle classi Partita e Giocatore.

Nella classe Giocatore è stato aggiunta una variabile per definire lo stato in prigione/libero e relativi getter e setter.

Nella classe Partita è stato modificato il metodo turno() in maniera da impedire l’inizio del turno al Giocatore in prigione fino al pagamento della cauzione, definita mediante costante Partita.CAUZIONE_PRIGIONE.

Durante lo sviluppo di quest’ultima release abbiamo osservato, con grande soddisfazione, che l’approccio orientato agli sviluppi futuri utilizzato durante le release precedenti ci ha permesso di implementare una nuova funzione con modifiche minime al codice.

Strumenti applicativi utilizzati

Linguaggio di programmazione:	Java
Ambiente di sviluppo:	Eclipse
Strumenti di controllo versione:	Git+GitHub

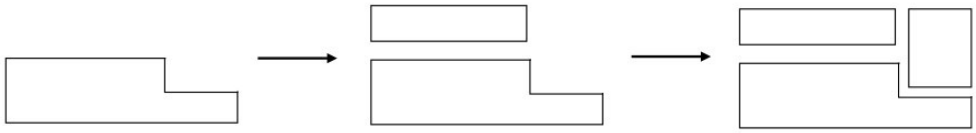
La documentazione è stata redatta utilizzando JavaDoc.

Modello di processo

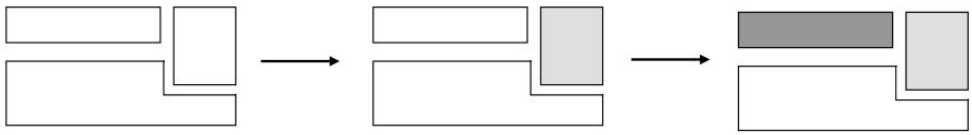
Il modello di processo adottato per lo sviluppo del software è stato il modello Incrementale/Iterativo. Questo modello consente di creare più versioni del software,ognuna delle quali integra un maggior numero di requisiti rispetto alla versione precedente

Le varie fasi possono essere rappresentate da questo modello:

Approccio incrementale:



Approccio Iterativo:



Manuale d’uso

Installazione

L’applicazione è stata scritta nel linguaggio di programmazione OO Java, quindi è eseguibile suqualsiasi calcolatore che abbia installato JRE (*Java Runtime Enviroment*).

Esecuzione

Per l’avvio dell’applicazione (.jar) eseguire i seguenti passaggi:

Aprire il Terminale di sistema (Shell di Unix o Prompt di Windows, a seconda del Sistema Operativo)

- Portarsi nella cartella in cui è presente il file **Progetto_Exe.jar**
- Eseguire il file .jar mediante il comando "*java –jar Progetto_Exe.jar*"

A questo punto, se non vengono visualizzati errori a causa di una errata versione di JRE, dovrebbe presentarsi questa situazione

Comandi

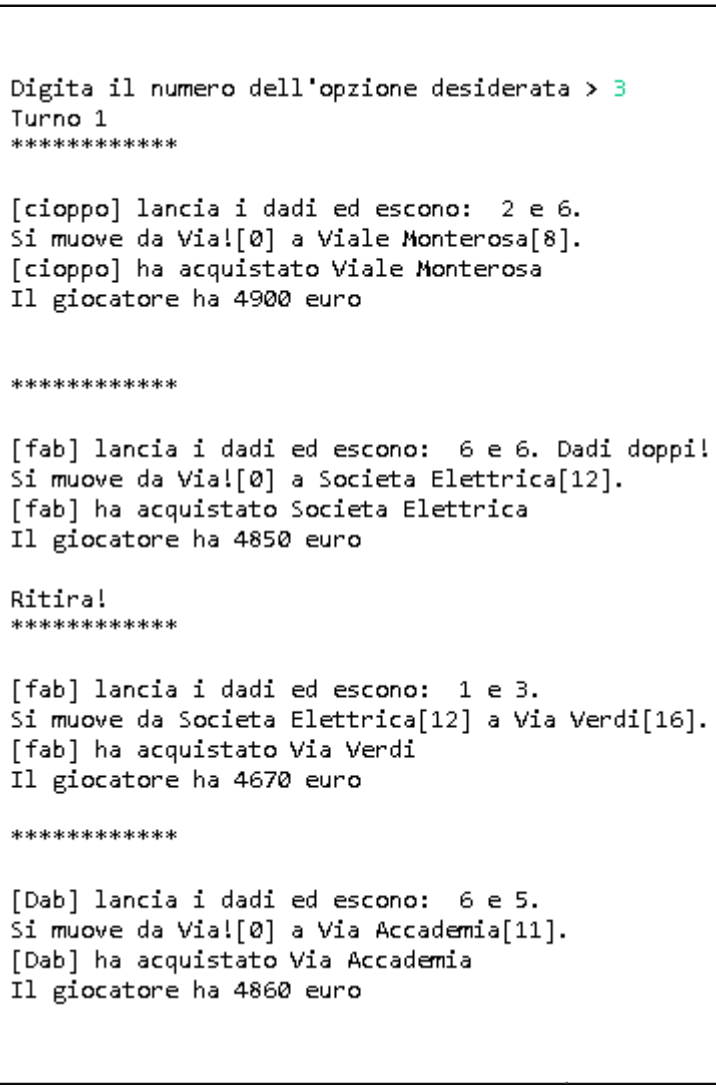
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~ BENVENUTO NEL GIOCO DEL MONOPOLY ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


--------------------------------
Menu' Monopoly
--------------------------------
1       Inserisci Nuovo Giocatore
2       Stampa Elenco Giocatori
3       Gioca
4       Rimuovi Giocatore
5       Reset

0       Esci

Digita il numero dell'opzione desiderata >
```

Nome	Inserimento nuovo giocatore
Numero	1
Funzione	L'applicazione permette di inserire il nome di un nuovo giocatore.
Immagini	`Digita il numero dell'opzione desiderata > 1` `Inserisci un nuovo giocatore > Dab`
Note	Se è già stato inserito il numero massimo di giocatori, il programma impedisce l'inserimento di un nuovo giocatore e lo comunica all'utente.

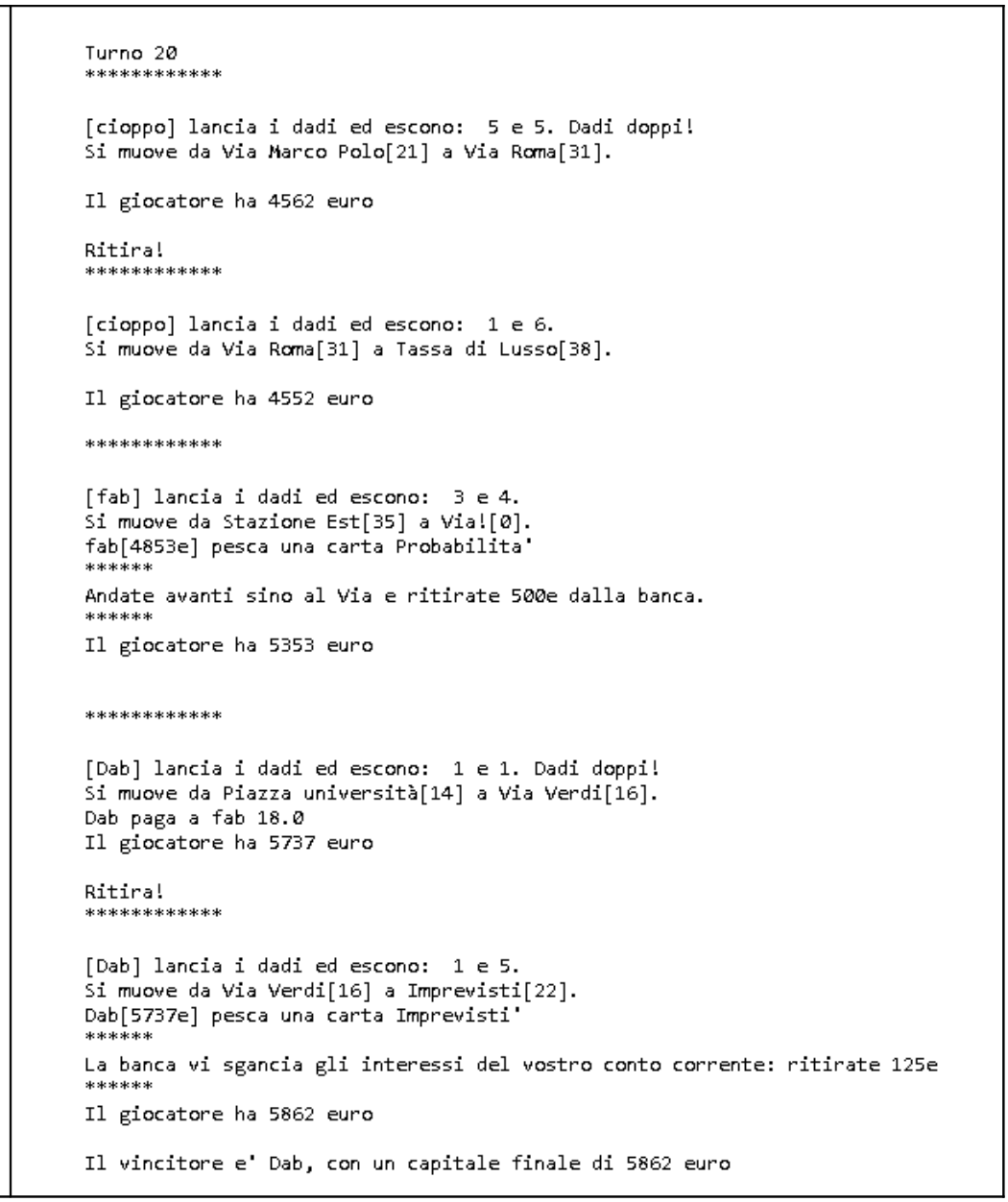
Nome	Stampa elenco giocatori
Numero	2
Funzione	L'applicazione permette di visualizzare l'elenco di tutti i giocatori che sono stati inseriti. caricato!il!modello!nel!sistema!

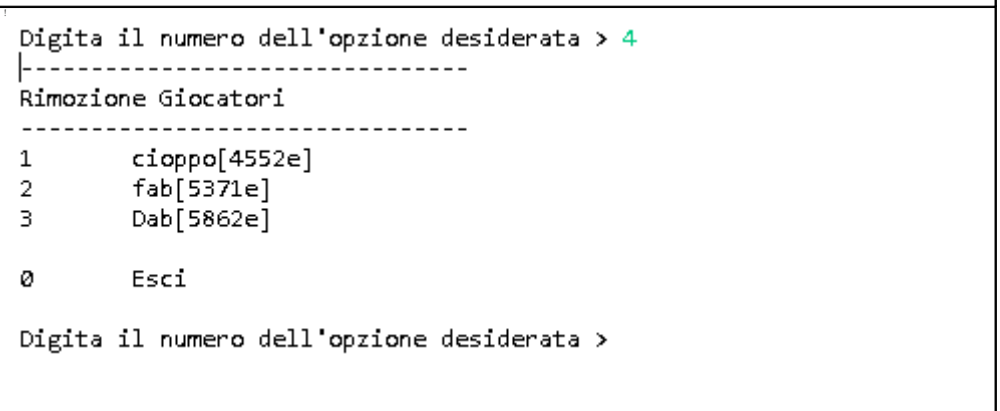
Immagini	`Digita il numero dell'opzione desiderata > 2`
	`1 - Dab[0e]`
	`2 - fab[0e]`
	`3 - cioppo[0e]`

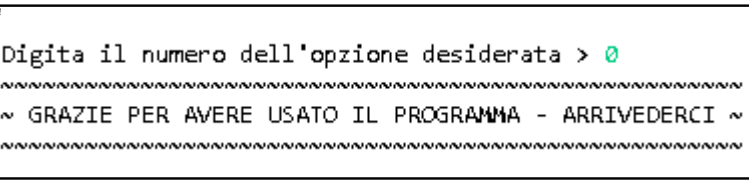
Note	Se non ci sono giocatori inseriti, il programma lo comunica all'utente.

Nome	Gioca
Numero	3
Funzione	L'applicazione permette di giocare una partita a Monopoly, permettendo l'avanzamento dei turni tramite la pressione del tasto invio
Immagini	

```
Digita il numero dell'opzione desiderata > 3
Turno 1
************

[cioppo] lancia i dadi ed escono:  2 e 6.
Si muove da Via![0] a Viale Monterosa[8].
[cioppo] ha acquistato Viale Monterosa
Il giocatore ha 4900 euro


************

[fab] lancia i dadi ed escono:  6 e 6. Dadi doppi!
Si muove da Via![0] a Societa Elettrica[12].
[fab] ha acquistato Societa Elettrica
Il giocatore ha 4850 euro

Ritira!
************

[fab] lancia i dadi ed escono:  1 e 3.
Si muove da Societa Elettrica[12] a Via Verdi[16].
[fab] ha acquistato Via Verdi
Il giocatore ha 4670 euro

************

[Dab] lancia i dadi ed escono:  6 e 5.
Si muove da Via![0] a Via Accademia[11].
[Dab] ha acquistato Via Accademia
Il giocatore ha 4860 euro
```

Immagini	

Nome	Rimuovi giocatore
Numero	4
Funzione	L'applicazione permette di rimuovere un giocatore, presentando la lista dei giocatori precedentemente inseriti
Immagini	
Note	I valori da inserire possono variare rispetto a quelli presentati nell'immagine, a seconda delle opzioni disponibili.

Nome	Esci
Numero	0
Funzione	L'applicazione si chiude.
Immagini	
Note	Per la riesecuzione del programma seguire nuovamente la procedura di esecuzione.

Release 1

Diagramma UML delle classi

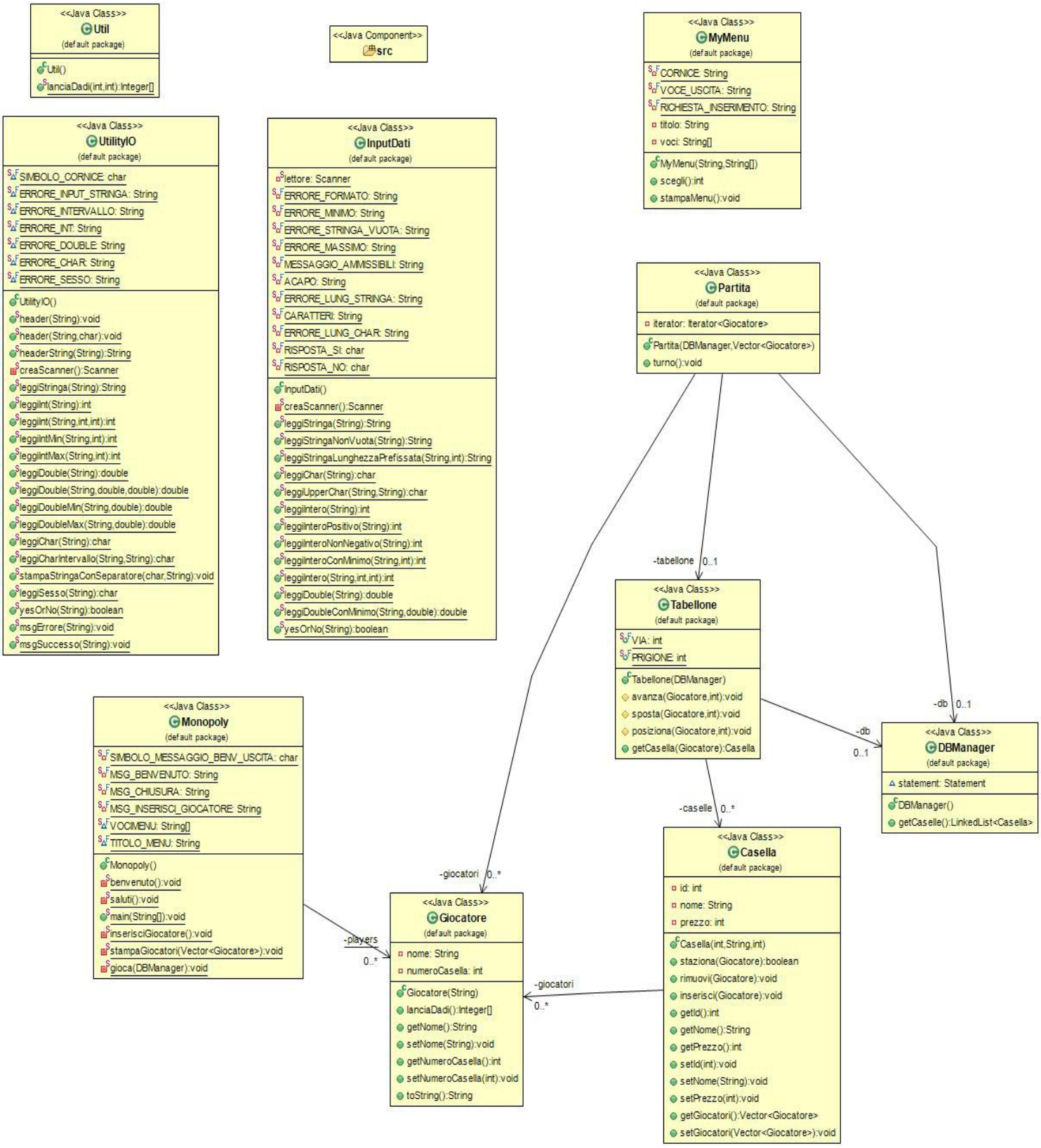
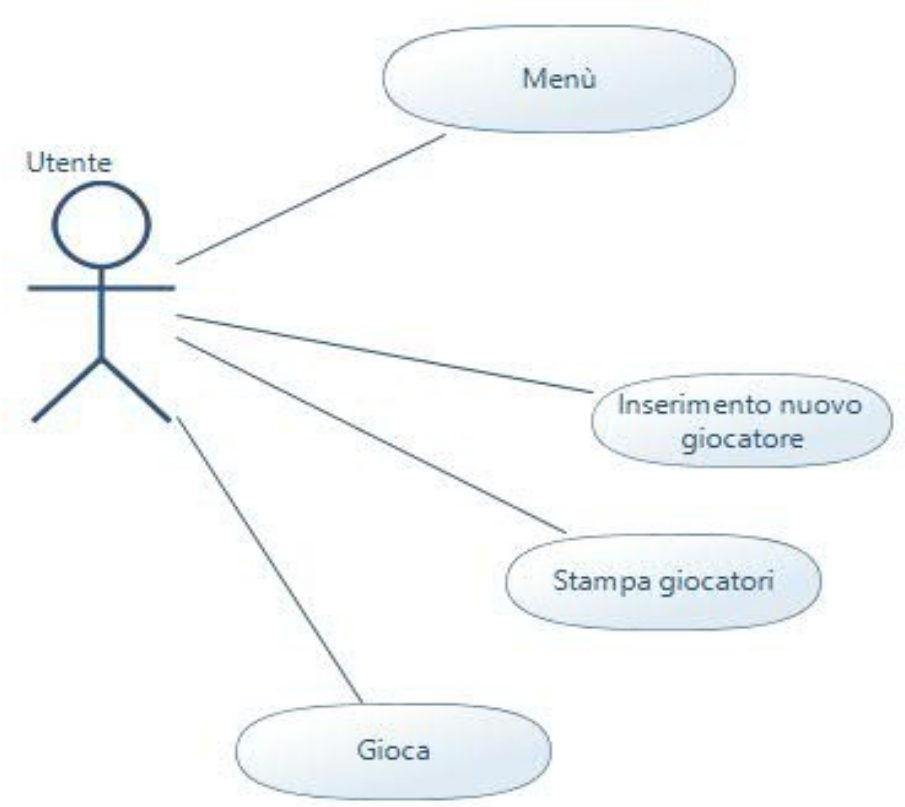


Diagramma UML Casi d'uso



Casi d'uso

Nome	Menu Principale
Attore	Utente
Scenario Principale	1. L'applicazione presenta il menu di scelta 2. L'utente sceglie l'azione da svolgere 3. L'utente ha scelto di uscire dall'applicazione 4. L'applicazione viene terminata.
Scenario Alternativo	3.a L'utente ha scelto un'operazione da svolgere L'applicazione svolge l'operazione selezionata, torna al punto 1!
Scenario Alternativo	3.a L'utente digita un input errato Viene visualizzato un messaggio d'errore, Torna al punto 1!

Gestione del menu':

Nome	Inserimento nuovo giocatore
Attore	Utente
Scenario Principale	1. L'utente sceglie di inserire un nuovo giocatore 2. Il sistema fa inserire all'utente il nome del giocatore 3. Il sistema controlla che non ci siano più di 6 giocatori
Scenario Alternativo	1.a L'utente decide di inserire un nuovo giocatore Torna al punto 2
Scenario Alternativo	3.a L'utente inserisce più di 6 giocatori Il sistema rileva che ci sono più di 6 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Nome	Stampa giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di stampare a video la lista dei giocatori 2. Il sistema stampa la lista dei giocatori inseriti

Nome	Gioca
Attore	Utente
Scenario Principale	1. L'utente sceglie di iniziare la partita 2. Il sistema controlla che i giocatori siano più di 2 3. Il sistema svolge la partita
Scenario Alternativo	2.a L'utente inserisce meno di 2 giocatori Il sistema rileva che ci sono meno di 2 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Release 2

Diagramma UML delle classi

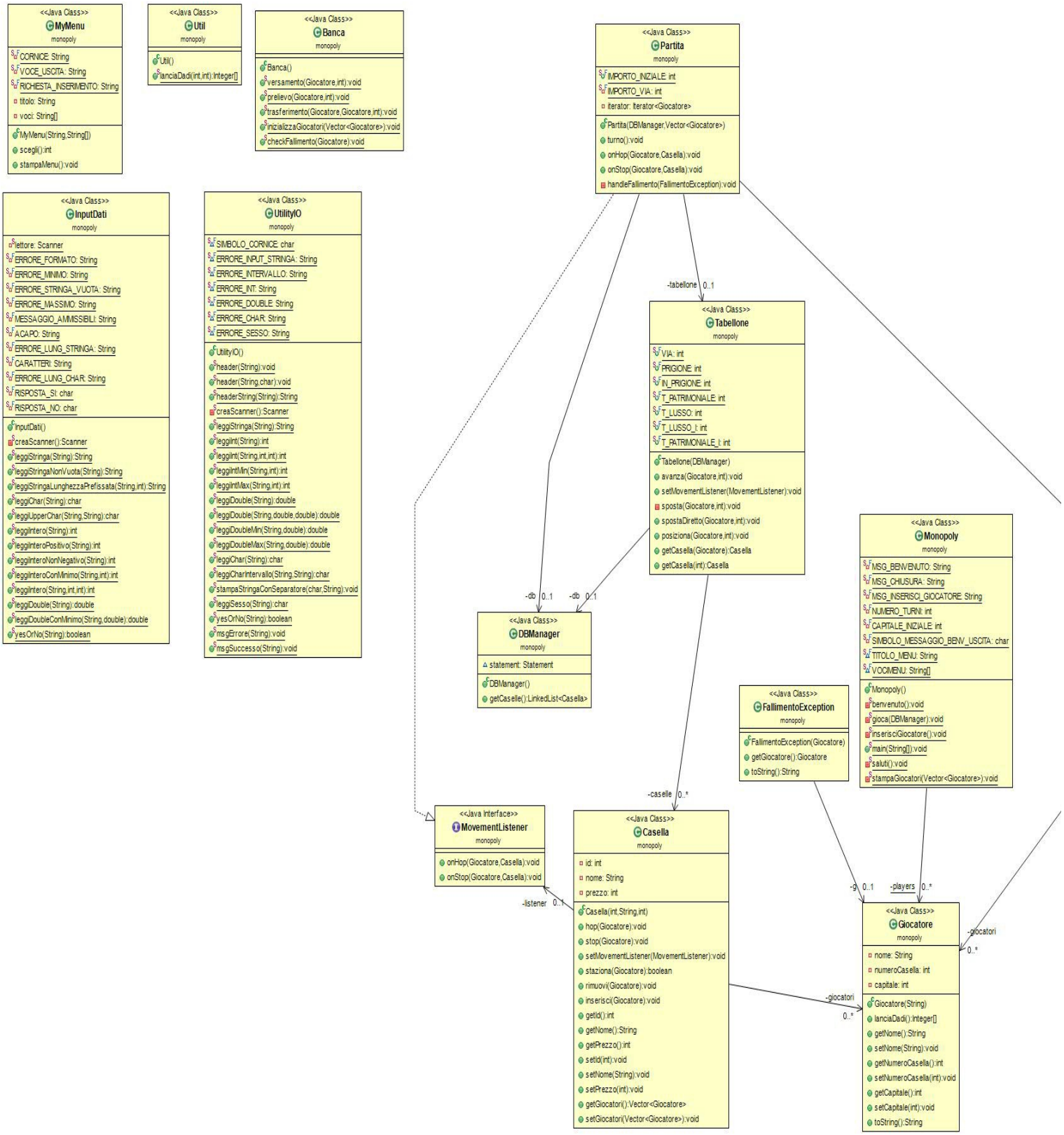
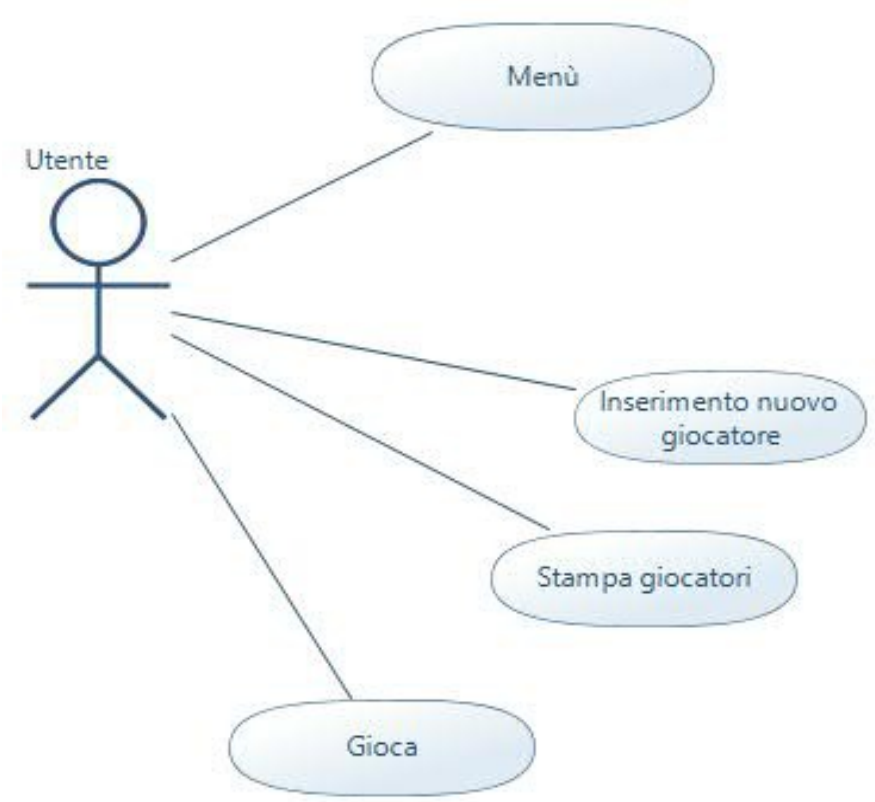


Diagramma UML Casi d'uso



Casi d'uso

Nome	Menu Principale
Attore	Utente
Scenario Principale	1. L'applicazione presenta il menu di scelta 2. L'utente sceglie l'azione da svolgere 3. L'utente ha scelto di uscire dall'applicazione L'applicazione viene terminata.
Scenario Alternativo	3.a L'utente ha scelto un'operazione da svolgere L'applicazio n e svolge l'operazione selezionata Torna al punto 1
Scenario Alternativo	3.a L'utente digita un input errato Viene visualizzato un messaggio d'errore, torna al punto 1

Gestione del menù:

Nome	Inserimento nuovo giocatore
Attore	Utente

Scenario Principale	1. L'utente sceglie di inserire un nuovo giocatore 2. Il sistema fa inserire all'utente il nome del giocatore 3. Il sistema controlla che non ci siano più di 6 giocatori
Scenario Alternativo	1.a L'utente decide di inserire un nuovo giocatore Torna al punto 2
Scenario Alternativo	3.a L'utente inserisce più di 6 giocatori Il sistema rileva che ci sono più di 6 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Nome	Stampa giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di stampare a video la lista dei giocatori 2. Il sistema stampa la lista dei giocatori inseriti e il relativo capitale

Nome	Gioca
Attore	Utente
Scenario Principale	1. L'utente sceglie di iniziare la partita 2. Il sistema controlla che i giocatori siano più di 2 3. Il sistema svolge la partita 4. Il sistema termina la partita e stampa a video tutti i turni con i relativi lanci di dadi dei giocatori, i loro movimenti e i loro capitali attuali
Scenario Alternativo	2.a L'utente inserisce meno di 2 giocatori Il sistema rileva che ci sono meno di 2 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Release 3

Diagramma UML delle classi

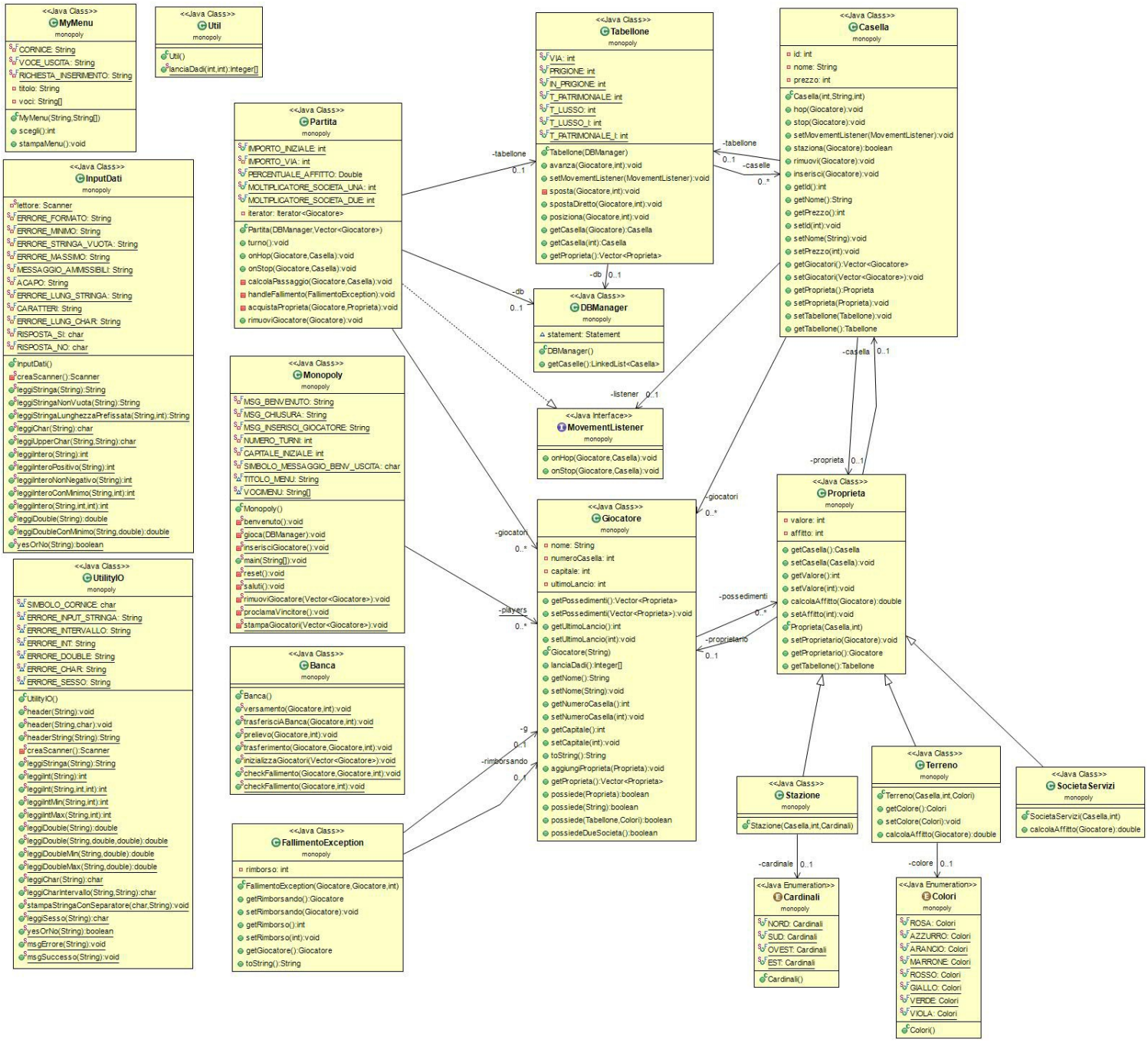
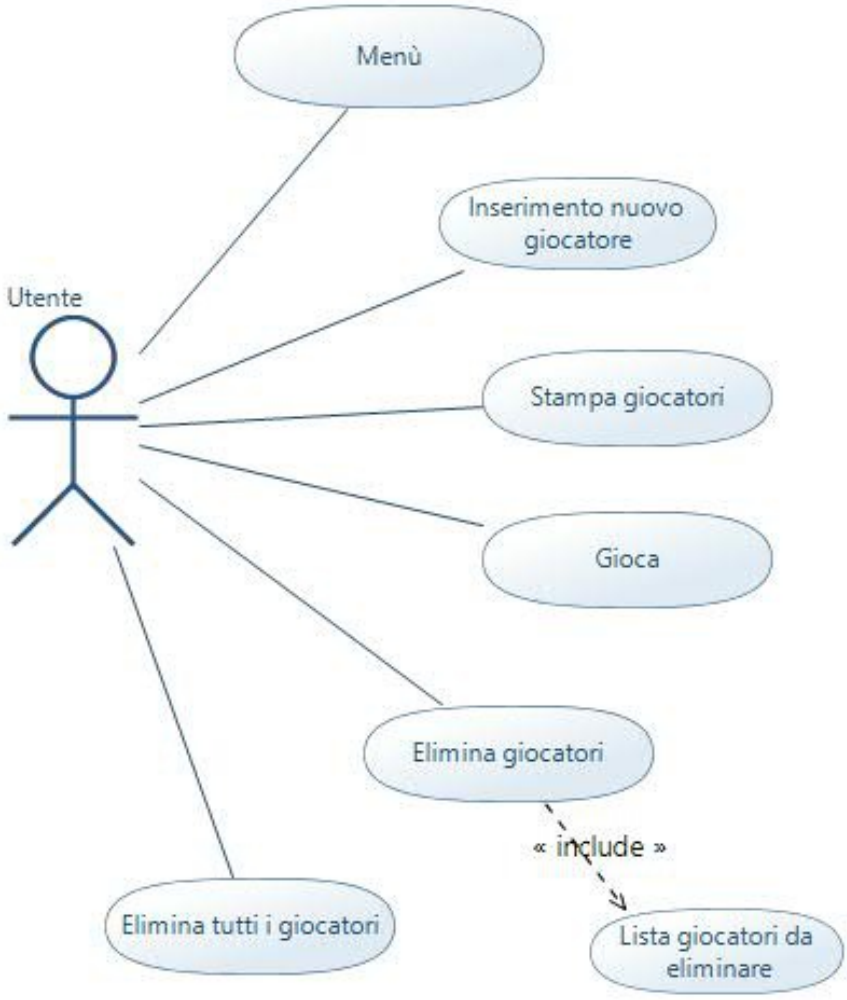


Diagramma UML casi d'uso



Casi d'uso

Nome	Menu Principale
Attore	Utente
Scenario Principale	1. L'applicazione presenta il menu di scelta 2. L'utente sceglie l'azione da svolgere 3. L'utente ha scelto di uscire dall'applicazione
Scenario Alternativo	3.a L'utente ha scelto un'operazione da svolgere L'applicazione svolge l'operazione selezionata, torna al punto 1!
Scenario Alternativo	3.a L'utente digita un input errato Viene visualizzato un messaggio d'errore Torna al punto 1!

Gestione del menù:

Nome	Inserimento nuovo giocatore
Attore	Utente
Scenario Principale	1. L'utente sceglie di inserire un nuovo giocatore 2. Il sistema fa inserire all'utente il nome del giocatore 3. Il sistema controlla che non ci siano più di 6 giocatori
Scenario Alternativo	1.a L'utente decide di inserire un nuovo giocatore Torna al punto 2
Scenario Alternativo	3.a L'utente inserisce più di 6 giocatori Il sistema rileva che ci sono più di 6 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Nome	Stampa giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di stampare a video la lista dei giocatori 2. Il sistema stampa la lista dei giocatori inseriti

Nome	Elimina giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di eliminare un giocatore 2. <<include>>" Lista dei giocatori da eliminare" 3. Il sistema elimina il giocatore scelto dall'utente

Nome	Elimina tutti i giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di eliminare tutti i giocatori 2. Il sistema elimina tutti i giocatori inseriti

Nome	Gioca
Attore	Utente

Scenario Principale	1. L'utente sceglie di iniziare la partita 2. Il sistema controlla che i giocatori siano più di 2 3. Il sistema svolge la partita 4. Il sistema termina la partita e stampa a video tutti i turni con i relativi lanci di dadi dei giocatori, le proprietà acquistate, i loro movimenti, i capitali attuali. 5. Il sistema dichiara il fallimento di un giocatore 6. Il sistema dichiara il vincitore Il sistema rileva che un giocatore non ha capitale sufficiente a pagare l'affitto all'avversario e lo dichiara fallito facendolo uscire dalla partita.
Scenario Alternativo	2.a L'utente inserisce meno di 2 giocatori Il sistema rileva che ci sono meno di 2 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Release 4

Diagramma UML delle classi

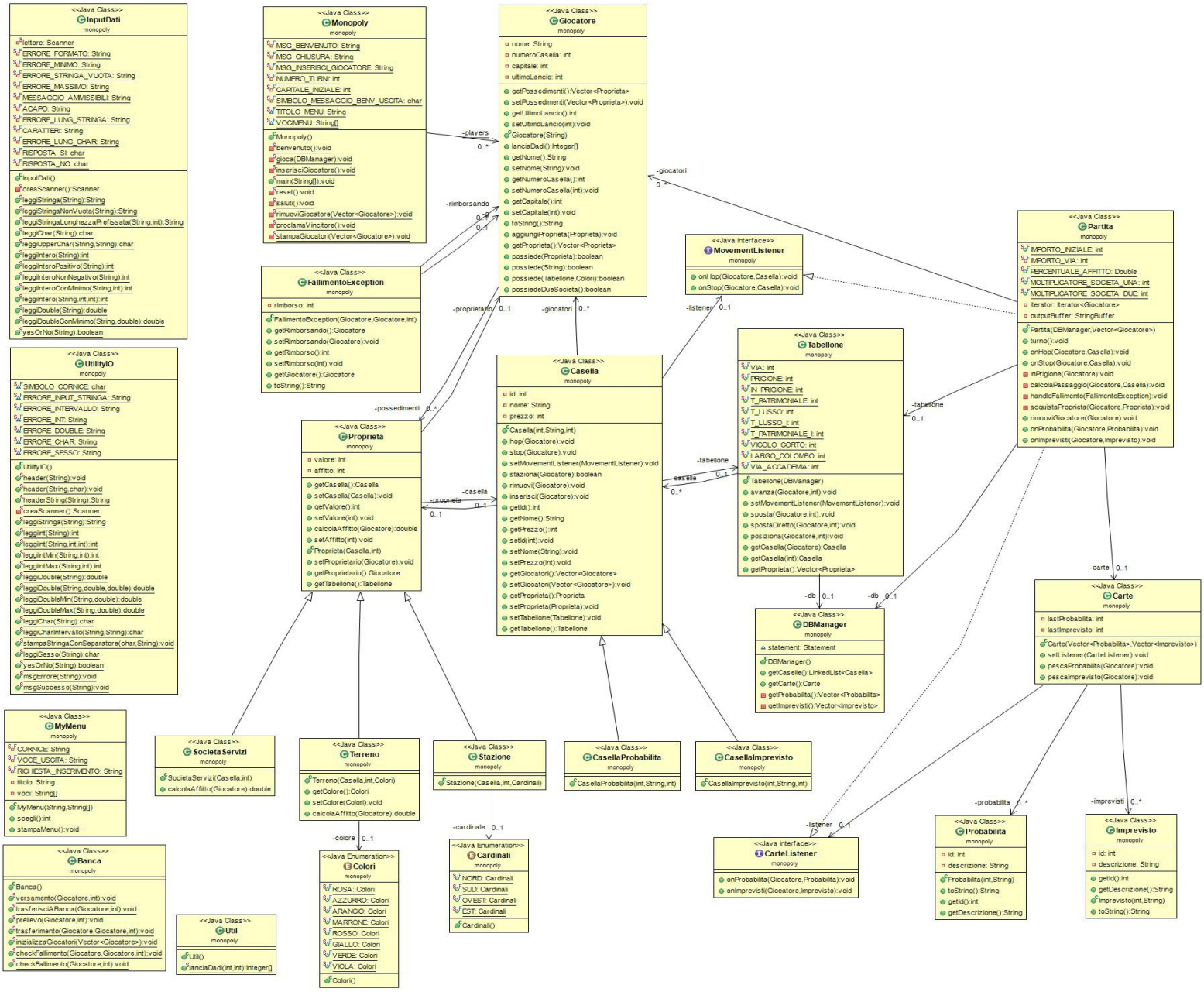
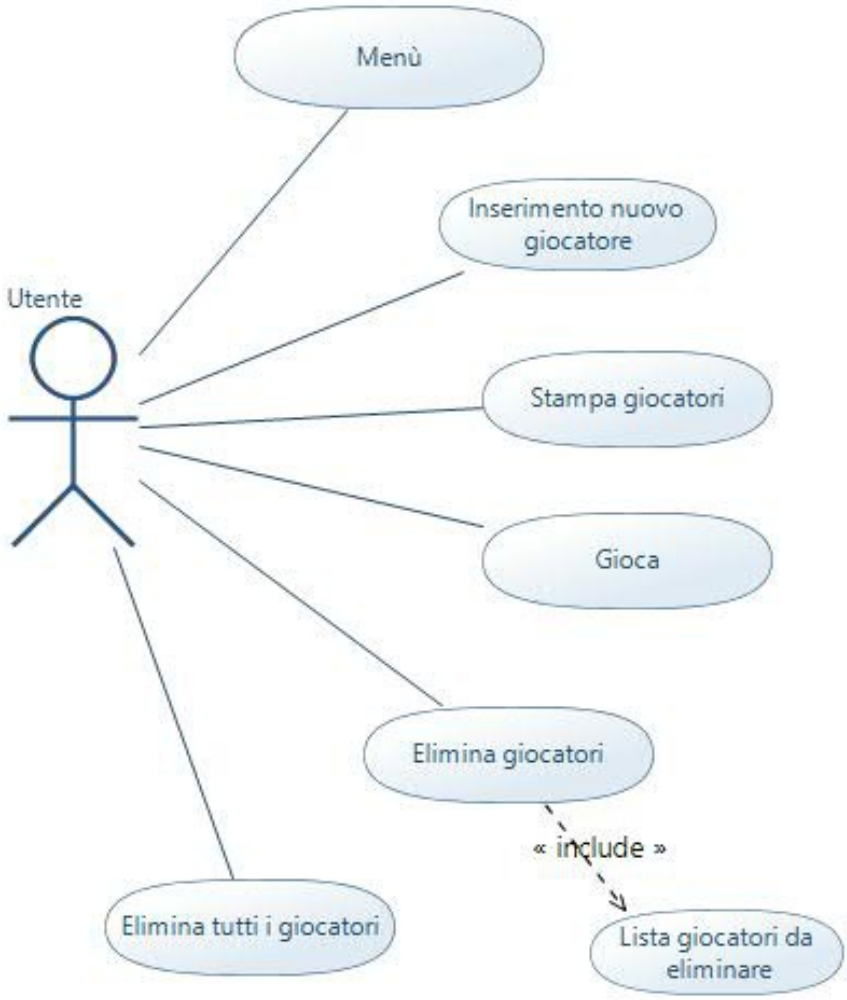


Diagramma UML casi d'uso



Casi d'uso

Nome	Menu Principale
Attore	Utente
Scenario Principale	1. L'applicazione presenta il menu di scelta 2. L'utente sceglie l'azione da svolgere 3. L'utente ha scelto di uscire dall'applicazione
Scenario Alternativo	3.a L'utente ha scelto un'operazione da svolgere L'applicazione svolge l'operazione selezionata, torna al punto 1!
Scenario Alternativo	3.a L'utente digita un input errato Viene visualizzato un messaggio d'errore Torna al punto 1!

Gestione del menu':

Nome	Inserimento nuovo giocatore
Attore	Utente
Scenario Principale	1. L'utente sceglie di inserire un nuovo giocatore 2. Il sistema fa inserire all'utente il nome del giocatore 3. Il sistema controlla che non ci siano più di 6 giocatori Postcondizione: i nomi dei giocatori vengono memorizzati nel sistema.
Scenario Alternativo	1.a L'utente decide di inserire un nuovo giocatore Torna al punto 2
Scenario Alternativo	3.a L'utente inserisce più di 6 giocatori Il sistema rileva che ci sono più di 6 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Nome	Stampa giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di stampare a video la lista dei giocatori 2. Il sistema stampa la lista dei giocatori inseriti

Nome	Elimina giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di eliminare un giocatore 2. <<include>>" Lista dei giocatori da eliminare" 3. Il sistema elimina il giocatore scelto dall'utente

Nome	Elimina tutti i giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di eliminare tutti i giocatori 2. Il sistema elimina tutti i giocatori inseriti

Nome	Gioca
Attore	Utente

Scenario Principale	1. L'utente sceglie di iniziare la partita 2. Il sistema controlla che i giocatori siano più di 2 3. Il sistema svolge la partita 4. Il sistema termina la partita e stampa a video tutti i turni con i relativi lanci di dadi dei giocatori, le proprietà acquistate, i loro movimenti, i capitali attuali. 5. Il sistema dichiara il fallimento di un giocatore 6. Il sistema dichiara il vincitore Il sistema rileva che un giocatore non ha capitale sufficiente a pagare l'affitto all'avversario e lo dichiara fallito facendolo uscire dalla partita.
Scenario Alternativo	2.a L'utente inserisce meno di 2 giocatori Il sistema rileva che ci sono meno di 2 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Release 5

Diagramma UML delle classi

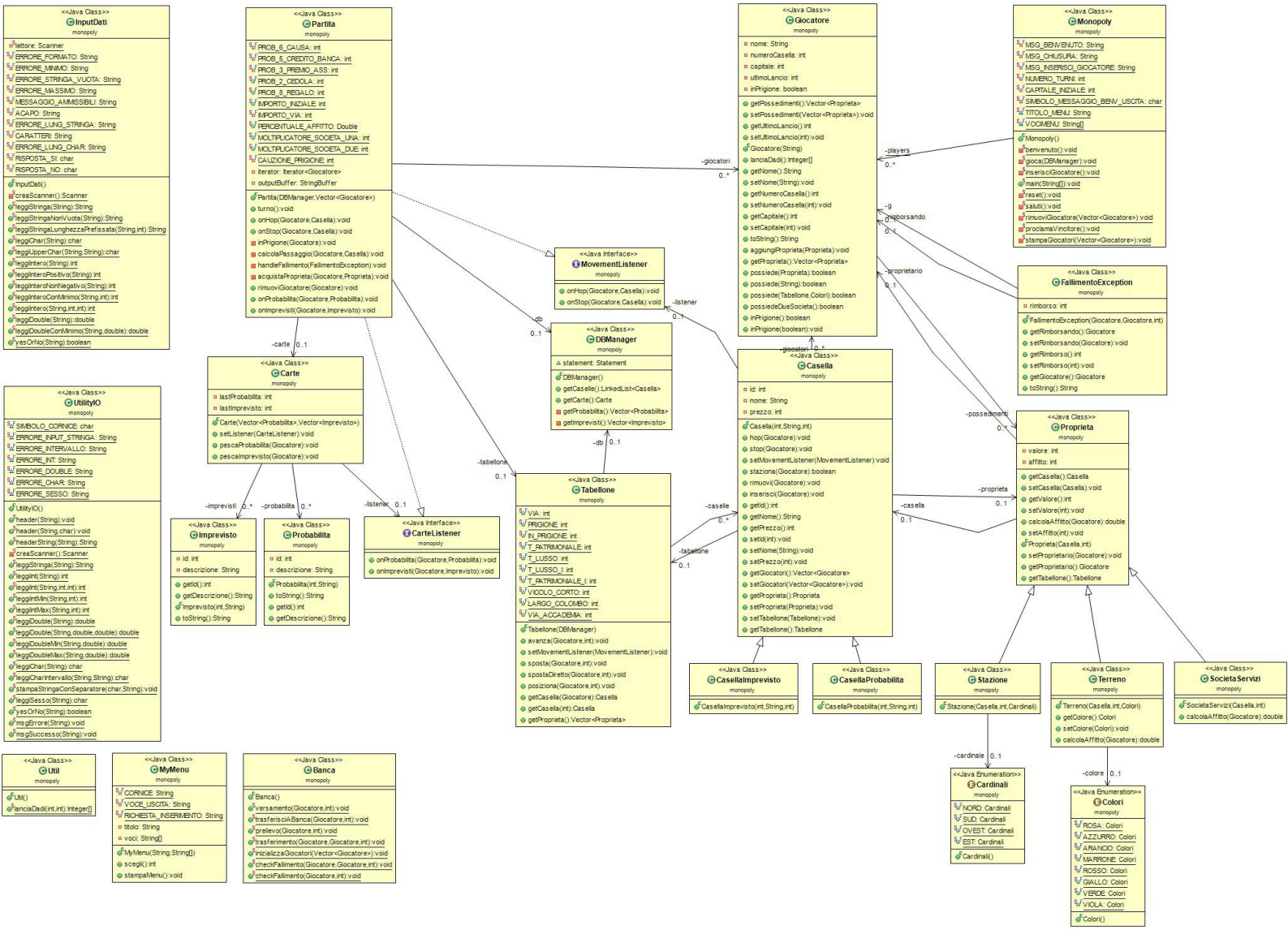
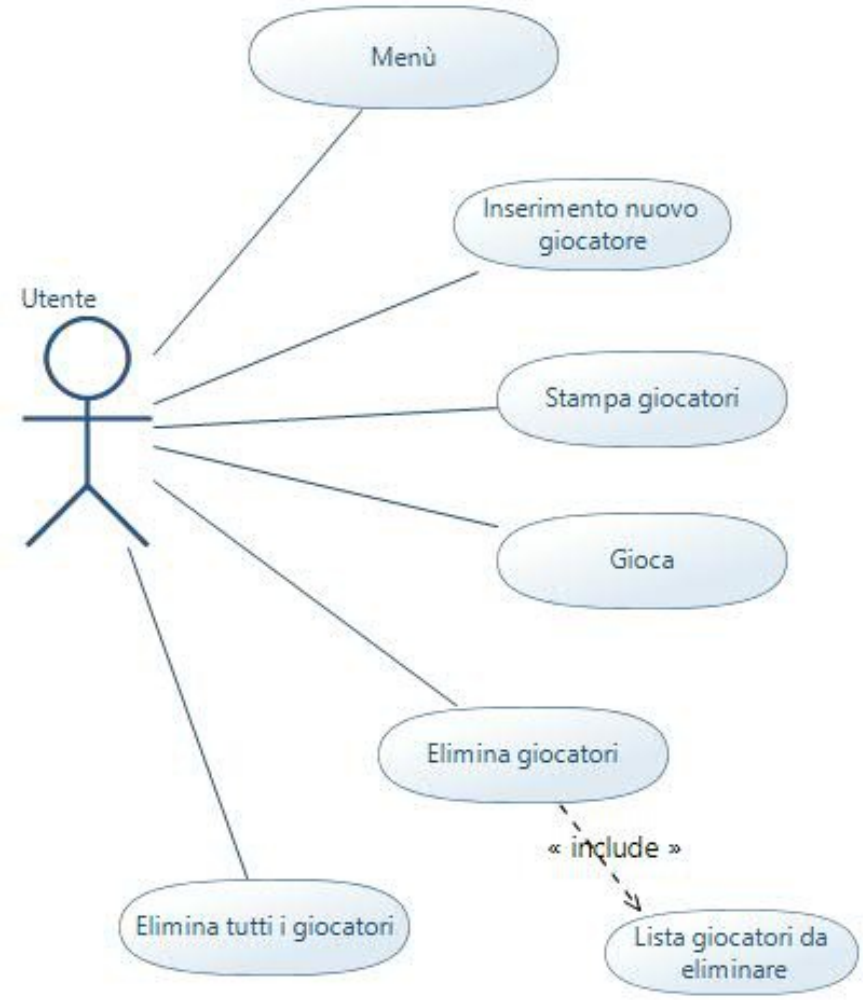


Diagramma UML casi d'uso



Casi d'uso

Nome	Menu Principale
Attore	Utente
Scenario Principale	1. L'applicazione presenta il menu di scelta 2. L'utente sceglie l'azione da svolgere 3. L'utente ha scelto di uscire dall'applicazione
Scenario Alternativo	3.a L'utente ha scelto un'operazione da svolgere L'applicazione svolge l'operazione selezionata, torna al punto 1!
Scenario Alternativo	3.a L'utente digita un input errato Viene visualizzato un messaggio d'errore Torna al punto 1!

Gestione del menu':

Nome	Inserimento nuovo giocatore

Attore	Utente
Scenario Principale	1. L'utente sceglie di inserire un nuovo giocatore 2. Il sistema fa inserire all'utente il nome del giocatore 3. Il sistema controlla che non ci siano più di 6 giocatori Postcondizione: i nomi dei giocatori vengono memorizzati nel sistema.
Scenario Alternativo	1.a L'utente decide di inserire un nuovo giocatore Torna al punto 2
Scenario Alternativo	3.a L'utente inserisce più di 6 giocatori Il sistema rileva che ci sono più di 6 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Nome	Stampa giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di stampare a video la lista dei giocatori 2. Il sistema stampa la lista dei giocatori inseriti

Nome	Elimina giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di eliminare un giocatore 2. <<include>>" Lista dei giocatori da eliminare" 3. Il sistema elimina il giocatore scelto dall'utente

Nome	Elimina tutti i giocatori
Attore	Utente
Scenario Principale	1. L'utente sceglie di eliminare tutti i giocatori 2. Il sistema elimina tutti i giocatori inseriti

Nome	Gioca
Attore	Utente

Scenario Principale	1. L'utente sceglie di iniziare la partita 2. Il sistema controlla che i giocatori siano più di 2 3. Il sistema svolge la partita 4. Il sistema termina la partita e stampa a video tutti i turni con i relativi lanci di dadi dei giocatori, le proprietà acquistate, i loro movimenti, i capitali attuali. 5. Il sistema dichiara il fallimento di un giocatore 6. Il sistema dichiara il vincitore Il sistema rileva che un giocatore non ha capitale sufficiente a pagare l'affitto all'avversario e lo dichiara fallito facendolo uscire dalla partita.
Scenario Alternativo	2.a L'utente inserisce meno di 2 giocatori Il sistema rileva che ci sono meno di 2 giocatori e stampa a video un messaggio d'errore Torna al menù principale

Testing

Test Suite	1
Software Testato	Menu Principale.
Versione	1
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Risultato	Il Menu Principale risponde correttamente ai comandi dell'utente.
Autore	Cordioli Francesco.
Data	09/03/2015

Test Case	1
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver* Main *Oracle:* Main *Stub:*
Dati in ingresso	0
Uscite attese	Uscita dal programma con relativo messaggio di uscita.
Uscite effettive	Le stesse.
Risultato'	PASS! Il menu principale risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	2
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver* Main *Oracle:* Main *Stub:*
Dati in ingresso	1
Uscite attese	Inserimento e salvataggio di un nuovo giocatore.
Uscite effettive	Le stesse.

Risultato'	PASS! Il menu principale risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	3
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver* Main *Oracle:* Main *Stub:*
Dati in ingresso	2
Uscite attese	Visualizzazione dei giocatori inseriti dall'utente.
Uscite effettive	Le stesse.
Risultato'	PASS! Il menu principale risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	4
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver* Main *Oracle:* Main *Stub:*
Dati in ingresso	3
Uscite attese	Visualizzazione della partita giocata con il numero della faccia dei dadi tirati e spostamento del giocatore.
Uscite effettive	Le stesse.
Risultato'	PASS! Il menu principale risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	5

Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver:* Main *Oracle:* Main *Stub:*
Dati in ingresso	4
Uscite attese	Messaggio di errore e ritorno al menu principale.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	E' stato dato in ingresso un'opzione del menù che non esiste.

Test Suite	2
Software Testato	Inserimento dei giocatori.
Versione	1
Obiettivo	Verificare il corretto inserimento dei giocatori e il numero massimo di essi.
Risultato	L'inserimento risponde correttamente ai comandi dell'utente.
Autore	Cordioli Francesco.
Data	09/03/2015

Test Case	1
Obiettivo	Verificare il corretto inserimento dei dati.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	"Dab".
Uscite attese	Salvataggio del nome.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.

Osservazioni	Se vengono inseriti più di 6 giocatori viene stampato a video un messaggio che informa l'utente che è stato raggiunto il numero massimo di giocatori.

Test Case	2
Obiettivo	Verificare il corretto inserimento dei dati
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Messaggio d'errore.
Uscite effettive	Salvataggio del nome.
Risultato'	FAIL! Il programma non risponde correttamente ai comandi dell'utente.
Osservazioni	Manca il controllo sul mancato inserimento del nome del giocatore.

Test Case	3
Obiettivo	Verificare il corretto inserimento dei dati
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Messaggio d'errore.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Sistemato il controllo sull'inserimento del nome.

Test Suite	3
Software Testato	Visualizzazione dei giocatori.

Versione	1
Obiettivo	Verificare la corretta visualizzazione della lista dei giocatori inseriti dall'utente.
Risultato	La visualizzazione risponde correttamente ai comandi dell'utente.
Autore	Cordioli Francesco.
Data	09/03/2015

Test Case	1
Obiettivo	Verificare la corretta visualizzazione della lista dei giocatori.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Lista dei nomi.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Suite	4
Software Testato	Gioco-Partita.
Versione	1
Obiettivo	Verificare il corretto funzionamento del gioco.
Risultato	Il gioco funziona come da consegna.
Autore	Cordioli Francesco.
Data	09/03/2015

Test Case	1
Obiettivo	Verificare il corretto funzionamento del gioco.

Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Lista di 20 turni con varie descrizioni di dadi e posizioni aggiornate.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Il programma da messaggio di errore se sono stati inseriti meno di 2 giocatori.

Test Case	2
Obiettivo	Verificare che tutti i giocatori inizino la partita dalla casella VIA!
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	I giocatori si trovano tutti sulla casella VIA!
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	3
Obiettivo	Verificare che dopo 3 volte che il giocatore ottiene lo stesso punteggio per entrambi i dadi finisce in prigione.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.

Uscite attese	Il giocatore dopo 3 tiri uguali finisce in prigione.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	4
Obiettivo	Verificare che una volta che il giocatore arriva alla casella 40 riparta dalla casella VIA!
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Alla conclusione di ogni giro il conteggio della posizione del giocatore riparte da 0.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.

Test Case	5
Obiettivo	Verificare che una partita termina dopo 20 turni.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Termine della partita.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.

Osservazioni	

Test Suite	5
Software Testato	Gioco-partita con l'interazione della banca.
Versione	1
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Risultato	Il risultato finale corrisponde alla consegna data.
Autore	Cordioli Francesco.
Data	10/03/2015

Test Case	1
Obiettivo	Verificare che ad ogni giocatore sia assegnata la cifra di 5000€.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Ogni giocatore ha un capitale di 5000€.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	2
Obiettivo	Verificare che ad ogni passaggio dal via il giocatore riceva 500€.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*

Dati in ingresso	Nessuno.
Uscite attese	Ogni utente riceve 500€ dalla banca.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	3
Obiettivo	Verificare che dalla casella 30 ("IN PRIGIONE") venga spostato alla casella 10 ("PRIGIONE/TRANSITO") senza ricevere i 500€ dalla banca per il passaggio dal via.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Passaggio dalla casella 30 alla casella 10 e i soldi devono rimanere gli stessi.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	4
Obiettivo	Verificare il pagamento di 250€ alla banca per aver sostato sulla casella TASSA PATRIMONIALE.

Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore ha 250€ in meno.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	5
Obiettivo	Verificare il pagamento di 10€ alla banca per aver sostato sulla casella TASSA DEL LUSSO.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore ha 10€ in meno.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	6
Obiettivo	Verificare il fallimento di un giocatore e l'uscita di esso dalla partita.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*

Dati in ingresso	Nessuno.
Uscite attese	Il giocatore esce dalla partita.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	7
Obiettivo	Verificare il vincitore della partita giocata.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Messaggio a video con il nome del vincitore.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Il giocatore ha vinto perché o tutti gli altri giocatori hanno fallito oppure è quello più ricco.

Test Suite	6
Software Testato	Menu Principale.
Versione	1
Obiettivo	Verificare il corretto funzionamento delle due voci aggiunte al menù principale.
Risultato	Il Menu Principale risponde correttamente ai comandi dell'utente.
Autore	Falleti Davide.
Data	20/03/2015

Test Case	1
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver* Main *Oracle:* Main *Stub:*
Dati in ingresso	4
Uscite attese	Visualizzazione di una lista di giocatori per l'eliminazione.
Uscite effettive	Le stesse.
Risultato'	PASS! Il menu principale risponde correttamente ai comandi dell'utente.
Osservazioni	Visualizzo una lista di giocatori da eliminare; per poter scegliere quale eliminare premo il numero della lista corrispondente al giocatore.

Test Case	2
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Ambiente	*Driver* Main *Oracle:* Main *Stub:*
Dati in ingresso	5
Uscite attese	Eliminazione di tutti i giocatori.
Uscite effettive	Le stesse.
Risultato'	PASS! Il menu principale risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Suite	7
Software Testato	Gioco-partita con l'interazione della banca e con il valore su ogni casella.

Versione	1
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Risultato	Il risultato finale corrisponde alla consegna data.
Autore	Falleti Davide.
Data	20/03/2015

Test Case	1
Obiettivo	Verificare che ogni casella abbia un valore in euro.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Ogni casella ha il suo costo.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	2
Obiettivo	Verificare che quando un giocare arriva su una casella che non è ancora stata acquista-ta, la acquista. automaticamente.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore acquista automaticamente la proprietà.
Uscite effettive	Le stesse.

Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	3
Obiettivo	Verificare che il giocare acquisti la proprietà solo se ha denaro sufficiente.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Messaggio di fondi insufficienti e non avviene l'acquisto della proprietà.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	4
Obiettivo	Verificare che quando un giocatore arriva su un terreno o una stazione di proprietà di un altro giocatore, deve pagare a quest'ultimo il 10% del valore della casella.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Pagamento del 10% della casella all'avversario.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.

Osservazioni	

Test Case	5
Obiettivo	Verificare che quando un giocatore arriva su una società di proprietà di un altro giocatore, l'affitto deve essere 4 volte il numero ottenuto con i dadi.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	L'affitto pagato (in €) all'avversario è dato dal valore dei dadi moltiplicato per 4.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Questo avviene se l'avversario possiede una sola società di servizi.

Test Case	6
Obiettivo	Verificare che quando un giocatore arriva su una società di proprietà di un altro giocatore, l'affitto deve essere 10 volte il numero ottenuto con i dadi.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	L'affitto pagato (in €) all'avversario è dato dal valore dei dadi moltiplicato per 10.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Questo avviene se l'avversario possiede entrambe le società di servizi.

Test Case	7
Obiettivo	Verificare che se il giocatore possiede tutti i terreni di un certo colore, l'affitto raddoppia.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	L'affitto pagato all'avversario è doppio.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	8
Obiettivo	Verificare il fallimento di un giocatore.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore esce dal gioco.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Questo avviene se un giocatore non possiede un capitale sufficiente per pagare per intero l'affitto dovuto a un altro giocatore.

Test Case	9
Obiettivo	Verificare che se un giocatore si sposta su una casella di un altro giocatore ma non ha i soldi per pagare l'affitto, fallisce dando il suo capitale al giocatore che possiede la proprietà e quest'ultimo riceve la differenza del valore dovuto dalla banca.

Test Case	9
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	L'avversario riceve i soldi dal giocatore che ha fallito e riceve il resto dei soldi dalla banca.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	10
Obiettivo	Verificare che le proprietà del giocare fallito ritornino acquistabili.
Ambiente	*Driver* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Le proprietà tornano acquistabili.
Uscite effettive	Le stesse.
Risultato	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Suite	8
Software Testato	Gioco-partita con l'aggiunta delle PROBABILITA' e degli IMPREVISTI.
Versione	1

Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Risultato	Il risultato finale corrisponde alla consegna data.
Autore	Falleti Davide.
Data	27/03/2015

Test Case	1
Obiettivo	Verificare che all'inizio di ogni partita ciascuno dei due mazzi (IMPREVISTI e PROBABILITA') venga mescolato casualmente.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Mazzo mescolato casualmente.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	2
Obiettivo	Verificare che il giocatore quando arriva su una casella di probabilità o imprevisti, peschi la carta in cima al mazzo.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore pesca la prima carta del mazzo.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	3

Obiettivo	Verificare che il giocatore esegua l'operazione scritta sulla carta di probabilità o imprevisti.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore si comporterà in modo diverso in base all'operazione scritta sulla carta.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	4
Obiettivo	Verificare che il giocatore depositi la carta in fondo al mazzo.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore mette la carta in fondo al mazzo.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	5
Obiettivo	Verificare il fallimento di un giocatore.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore fallisce ed esce dalla partita.
Uscite effettive	Le stesse.

Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	Un giocatore fallisce quando non possiede un capitale sufficiente da eseguire l'azione scritta sulla carta di probabilità o di imprevisti.

Test Case	6
Obiettivo	Verificare che se un giocatore fallisce, l'eventuale resto che dovrebbe dare all'avversario venga fornito dalla banca.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	L'avversario riceve una certa quantità di denaro dalla banca.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	L'eventuale resto è dovuto dal fatto che il giocatore non ha soldi per eseguire l'azione scritta sulla carta di probabilità o imprevisti e fallendo cede tutto il suo capitale all'avversario a cui è dovuto.

Test Suite	9
Software Testato	Gioco-partita con l'aggiunta di alcune regole per la gestione della PRIGIONE.
Versione	1
Obiettivo	Verificare il corretto funzionamento del Menu Principale.
Risultato	Il risultato finale corrisponde alla consegna data.
Autore	Cordioli Francesco.
Data	03/04/2015

Test Case	1
Obiettivo	Verificare che il giocatore finisca in prigione quando pesca una carta di PROBABILITA' O IMPREVISITI che gli indica di andare in prigione.

Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore entra in prigione.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	2
Obiettivo	Verificare che il giocatore finisca in prigione quando, tirando i dadi, esce per tre volte dadi doppi.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore finisce in prigione.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	3
Obiettivo	Verificare che, al turno successivo, prima di tirare i dadi, il giocatore paghi 50€ alla banca per uscire di prigione.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore esce di prigione e la banca ha 50€ in più.

Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	4
Obiettivo	Verificare che se un giocatore è in prigione e non possiede un capitale sufficiente per uscirne, fallisce.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore fallisce.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	5
Obiettivo	Verificare che tutte le proprietà del giocatore fallito ritornino acquistabili.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Le proprietà sono riacquistabili.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	6
Obiettivo	Verificare che tutto il capitale del giocatore fallito venga ceduto alla banca.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	La banca ha in più il denaro del giocatore fallito.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	

Test Case	7
Obiettivo	Verificare che il giocatore fallito esca dalla partita.
Ambiente	*Driver:* Terminal_in_out *Oracle:* Main *Stub:*
Dati in ingresso	Nessuno.
Uscite attese	Il giocatore non c'è più nella partita.
Uscite effettive	Le stesse.
Risultato'	PASS! Il programma risponde correttamente ai comandi dell'utente.
Osservazioni	