Relazione per il progetto “Monopoli”

Pecorelli Margherita
Tarsitano Laura
Vairo Giuseppe

3 Febbraio 2016

Capitolo 1

Analisi

1.1 Requisiti

Il software che vi proponiamo, mira alla costruzione di una versione digitale del MONOPOLI.
Il Monopoli è un gioco da tavolo il cui scopo è trarre profitto attraverso attività economiche, come comprare, vendere e affittare proprietà, cercando di diventare il MONOPOLISTA.

Requisiti

- Tale versione digitale del Monopoli permetterà ai giocatori (da 2 a 6) di muovere le rispettive pedine sulla tavola in base al numero fatto lanciando i 2 dadi
- Si potrà comprare, vendere e affittare
- Verrà gestita una Banca che effettuerà varie operazioni economiche
- Verrà gestita la pesca dai mazzi previsti
- Verranno gestiti i turni dei giocatori (sia reali che eventualmente virtuali)
- Ci sarà la possibilità di scegliere la versione del gioco tra quelle proposte
- Il gioco potrà essere terminato qunado sarà rimasto un unico giocatore oppure a decisione di tutti igiocatori tramite un tasto

1.2 Analisi e modello del dominio

Ad ogni partita si dovrà essere in grado di far scegliere all'utente la versione desiderata, e ciò dovrà comportare anche la scelta tra le varie versioni di pedine, caselle, edifici, proprietà (tra cui anche i rispettivi contratti), denaro, mazzi..
Ciascun giocatore (Player) sarà identificato con una pedina (Pawn) assegnata in automatico e durante il prorpio turno dovrà essere in grado di compiere diverse azioni (Action) tra cui lanciare i dadi, comprare/vendere terreni (Ownership) , costruire edifici (Building) .. Altre azioni invece saranno obbligatorie, come ad esempio pagare le tasse, pagare l'affitto..
La Banca (Bank) dovrà occuparsi di tutti gli aspetti economici del gioco tra cui la vendita di proprietà, incassare le multe, concedere ipoteche..
Le interazioni tra le entità principali del problema sono riportate in Figura 1.1.

Una delle difficoltà sarà quella di riuscire a fare diverse versioni del gioco, senza avere del riuso di codice. Ciò richiederà di astrarre il più possibile tutti i concetti dalle rispettive implementazioni.
Un'altra grossa difficoltà sarà quella di far prendere decisioni intelligenti agli avversari virtuali, ovvero quelli gestiti dal computer (come ad esempio prendere decisioni su quali terreni comprare).

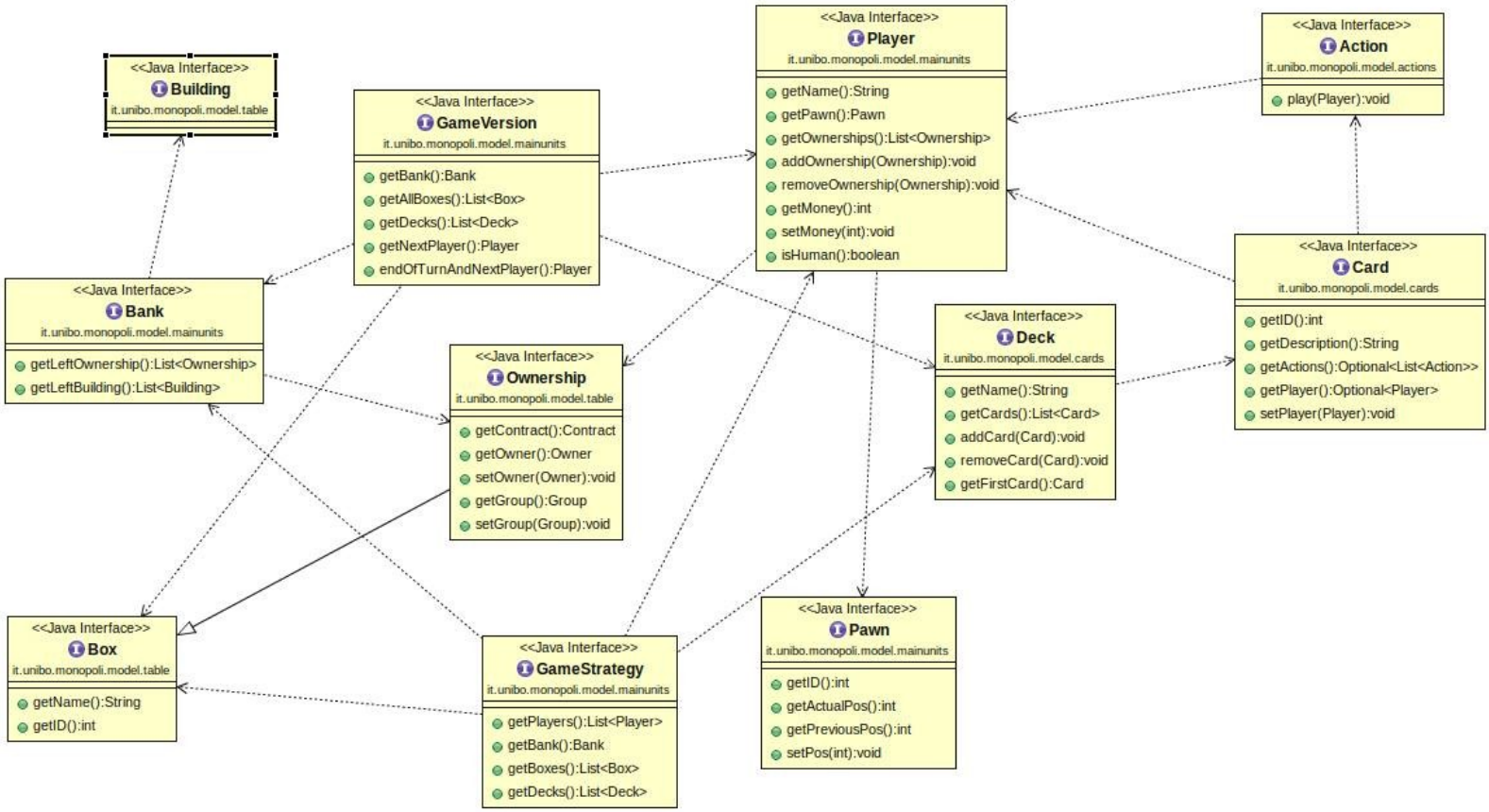


Figura 1.1

Questo UML rappresenta le interfacce delle entità principali del modello del gioco

Capitolo 2

Designi

2.1 Architettura

Per la realizzazione di questo progetto abbiamo deciso di utilizzare il patter architetturale MVC diviso nel modo in cui segue:

- **Model**: si occupa del dominio del modello. In questa parte vengono definite tutte le entità del gioco, le varie interazioni che devono avere fra loro, le regole e le azioni possibili. Inoltre è qui che si cercherà di rendere il tutto estendibile ad altre versioni del gioco, quindi si proverà ad astrarre lo schema base dall'implementazione della versione proposta.
- **View**: si occupa della visualizzazione grafica dei vari componenti e dell'inizializzazione del controller con la versione scelta dall'utente da richiamare nel model. La GUI dovrà visualizzare il tabellone del gioco, la banca e tutti i giocaoti con le varie pedine, i contratti e i soldi. Inoltre verrà gestito lo spostamento delle pedine sulle varie caselle.
- **Controller**: si occupa della parte strategica del gioco, quindi è colui che prende decisioni sulle prossime mosse consentite dando la possibilità ai giocatori “umani” di poter scegliere tra mirate azioni. Inoltre è colui che si occupa della gestione dei giocatori “virtuali” e delle notifiche da mandare alla view nel caso in cui quest'ultima fosse presente. Si è cercato di costruirlo senza dipendenza dalla view (che viene solo notificata in caso fosse presente); infatti il gioco può essere eseguito anche in assenza di quest'ultima in quanto il controller dispone di tutte le azioni necessarie al proseguimento della partita. Ognuna di queste è rappresentata da un metodo che, quando richiamato, ricalcola tutte le azioni consentite, in base allo stato dell'oggetto e non permetterà di svolgere quelle non contenute tra queste. Inoltre il controller ha il compito di richiamare il model per modificare lo stato degli oggetti, ma senza che quest'ultimo debba notificare nulla.

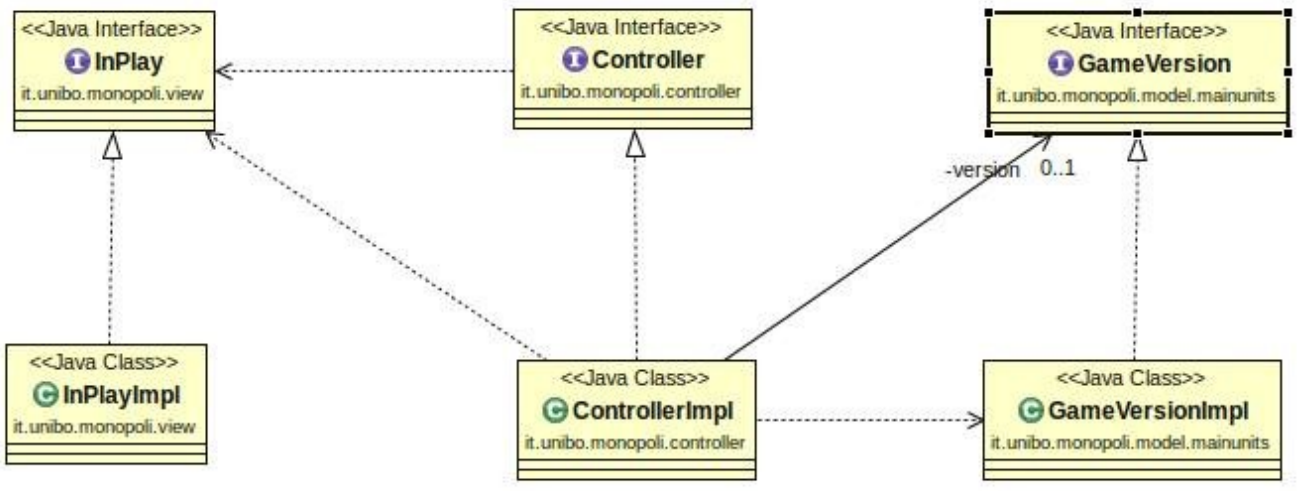


Diagramma principale del nostro UML

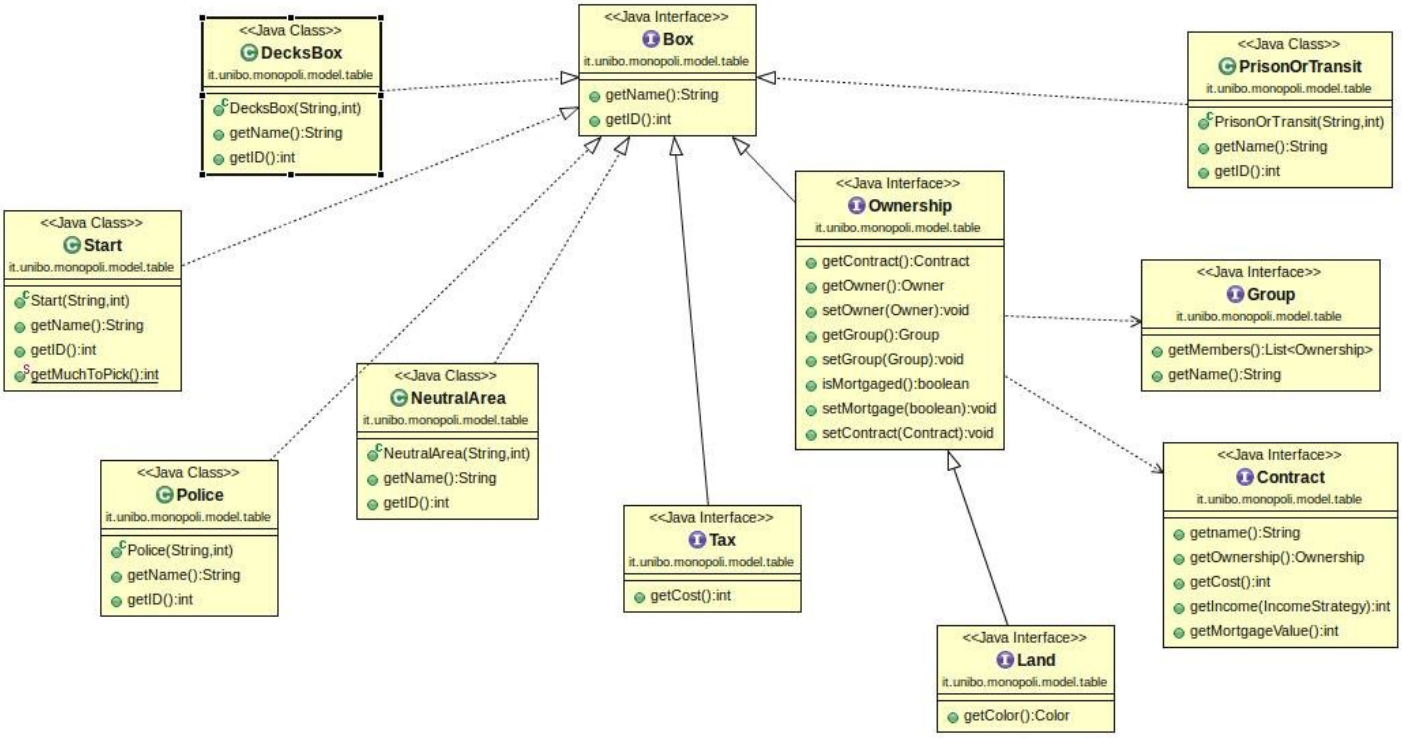
2.2 Design dettagliato

2.2.1 Model

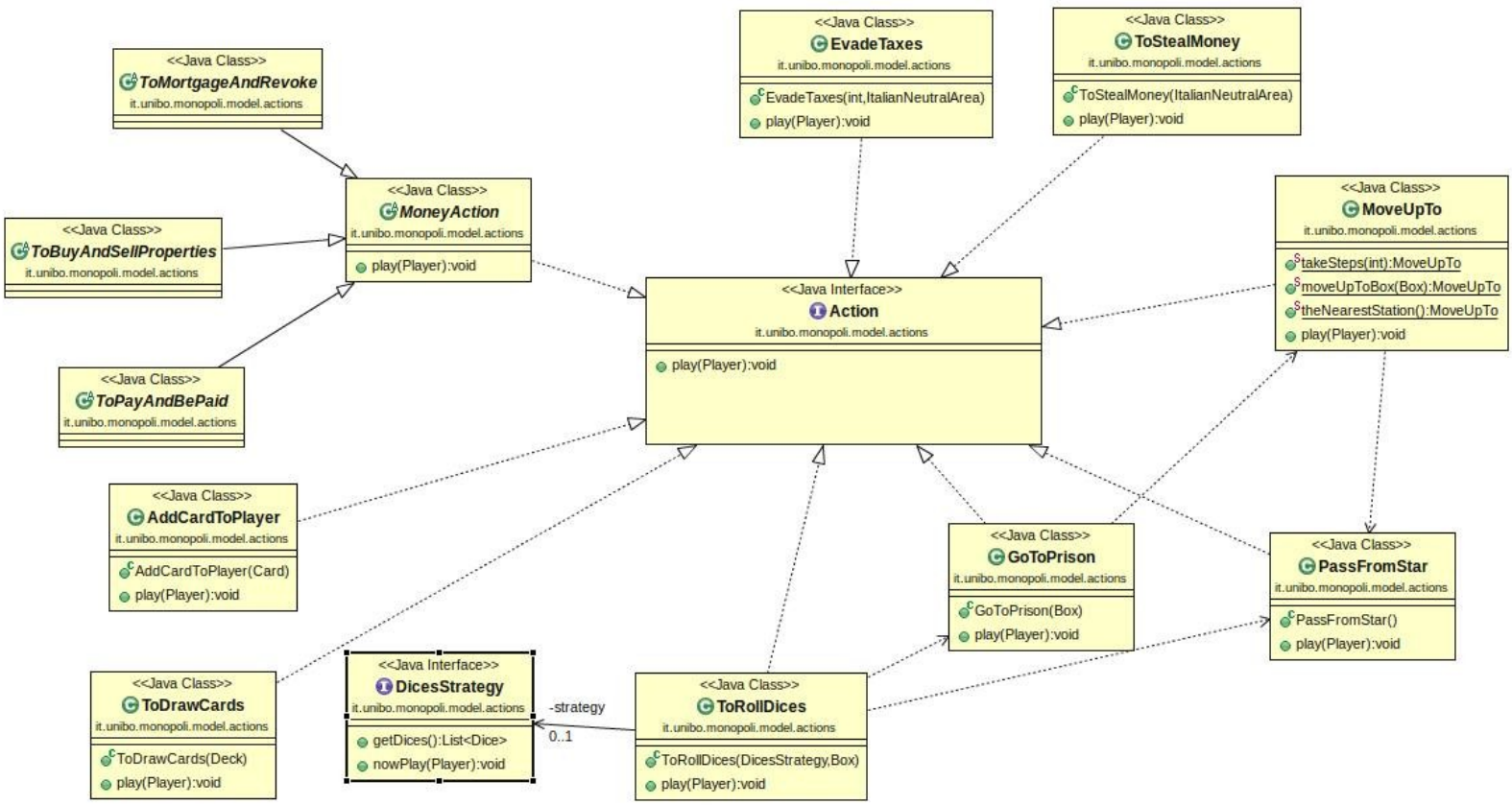
In questa parte del progetto si è cercato di rendere tutto il più estendibile possibile in modo tale da poter aggiungere diverse versioni del gioco senza dover ri-implementare tutto da capo (sperando che sia effettivamente così). Per fare ciò, sono state realizzate delle interfacce, ognuna per una singola entità del gioco (come la banca, i giocatori, le pedine..) , con dei metodi “essenziali” per il gioco stesso che però possono essere implementati a piacimento dalle sotto-classi.
Oltre a ciò, un'altra importante decisione (a mio parere) è stata fare un'unica interfaccia principale “Action” da cui estendono tutte le azioni. Quest'interfaccia è una “FunctionInterface”, infatti ha un unico metodo “play” che svolge l'azione specifica. In questo modo, nelle versioni future, se si dovessero aggiungere delle azioni da far compiere ai giocatori, basterà passarle tra le azioni e fare per ognuna il “.play()”.
Un altro aspetto positivo è il modo in cui sono state pensate le carte da gioco (“Cards”); queste possono essere pescate dai mazzi previsti dalla versione classica (quella implementata da noi), ma possono anche essere “possedute” da un giocatore. Quindi si potrà prevedere anche una versione in cui ogni giocatore oltre ai contratti e ai soldi “ha in mano” anche delle carte da gioco da usare a suo piacimento (come per esempio le carte di Bang).

I patter utilizzarti nella parte di model sono:

- Strategy viene usato:
  - nella versione del gioco (GameVersion vuole una GameStrategy) così che il controller possa richiamare sempre gli stessi metodi di GameVersion ma che la risposta sia diversa in base alla versione scelta
  - nei dadi da tirare (ToRollDice) abbiamo implementato due strategie: quella classica ha solo due dadi e se fa doppio può tirare 2 volte oppure, se è in prigione, uscire senza dover aspettare 3 turni; quella italiana invece ha tre dadi e se fa triplo va direttamente in prigione (mentre se fa doppio è uguale a quella classica)
- Template Method viene usato:
  - in MoneyAction definisco un metodo astratto che viene richiamato nel metodo principale e che svolge azioni diverse a seconda di chi lo implementa. Anche le classi astratte ToPayAndToBePaid, ToMortgageAndRevoke e ToBuyAndSellProperties usano la stessa tattica.
- Static Factory viene usato:
  - in MoveUpTo che ha diversi metodi che ritornano un'instanza di questa classe ma che la implementano in maniera diversa a seconda di come ci si debba “muovere”. Ogni metodo ha una nome che spiega abbastanza bene la “modalità di movimento” che si sta per adottare.
  - anche in ToBuyProperties e ToSellProperties sono presenti dei metodi che tornano un'instanza della classe inizializzata in maniera differente a seconda di quale metodo si è scelto (quello per comprare una proprietà o un edificio). Anche qui i metodi hanno dei nomi significativi in base all'azione desiderata.
- Builder viene usato:
  - in ClassicLandContract che per essere inizializzato ha bisogno di diversi parametri in input quasi tutti dello stesso tipo, quindi per evitare di confondersi, ho preferito utilizzare un builder anch'esso con metodi aventi nomi specifici per il parametro che vanno ad inserire.



Questo UML rappresenta tutte le interfacce principali adottate per costruire le varie caselle del tablellone.



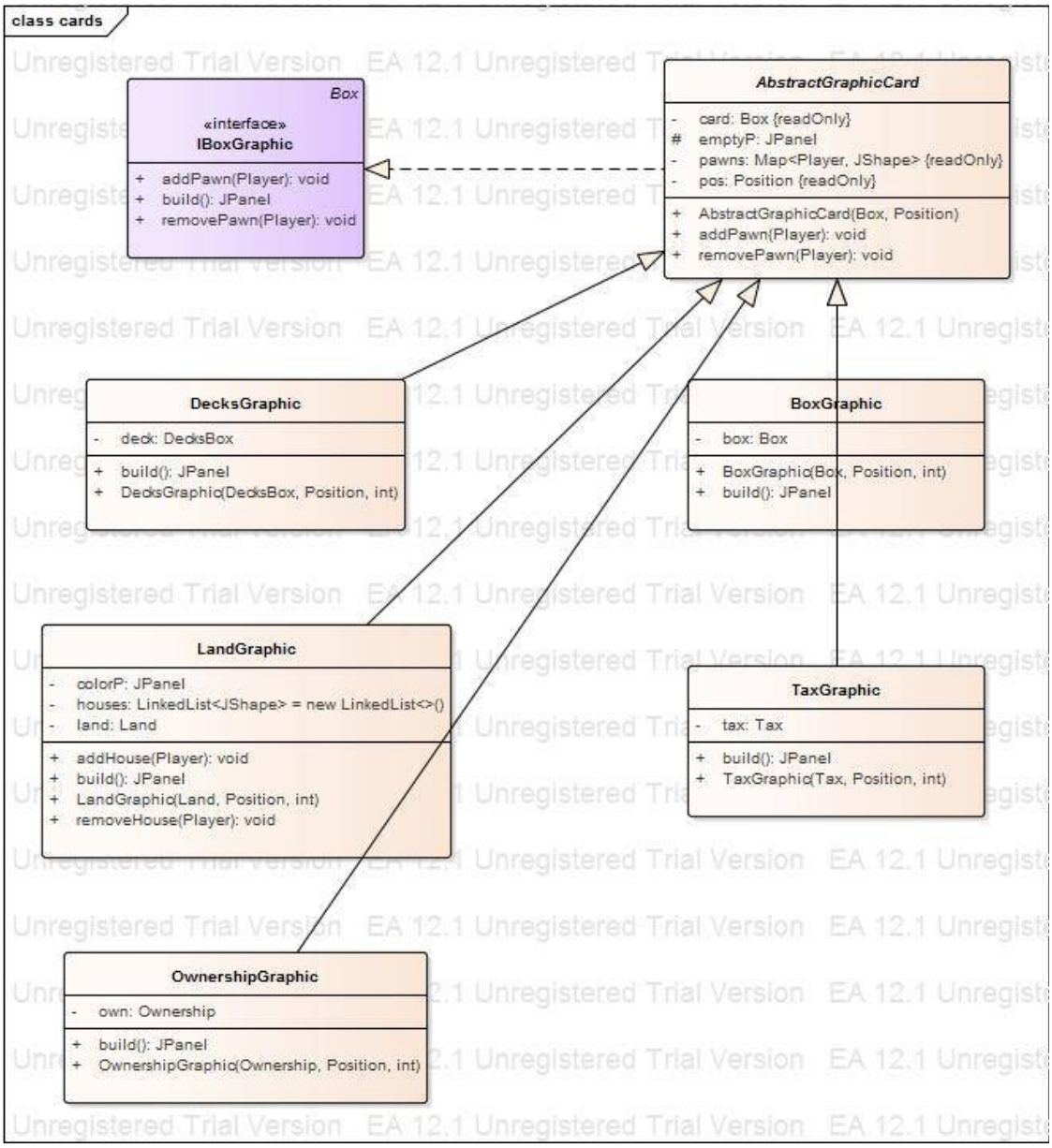
Questo UML rappresenta tutte le interfacce principali adottate per le azioni che i giocatori devono/possono svolgere

2.2.2 View

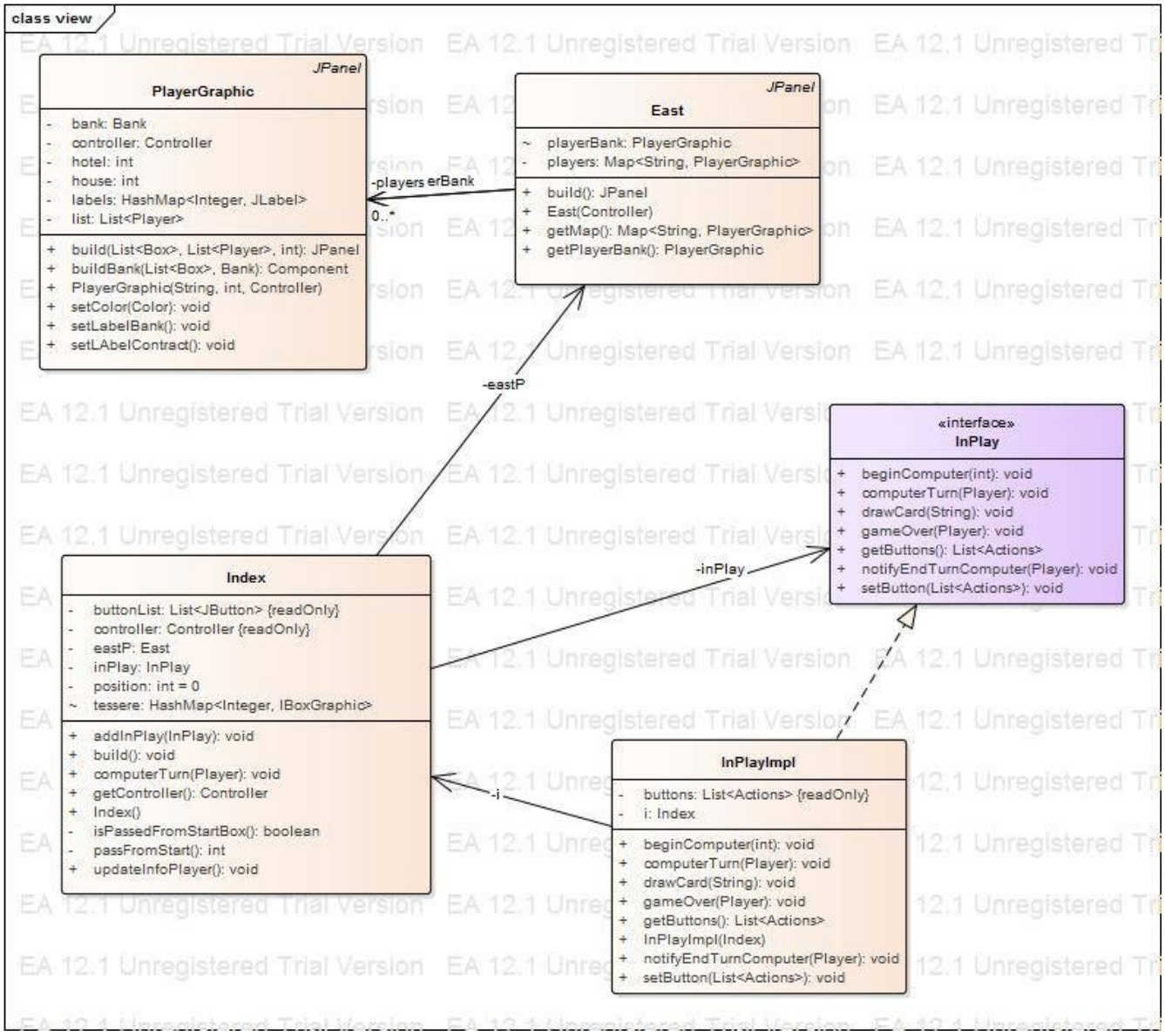
La view si occupa di visualizzare attraverso un'interfaccia grafica gli aspetti principali del gioco: Tabellone, informazioni dei giocatori, contratti e soldi posseduti.
Inoltre al lancio del gioco dà la possibilità di scegliere la versione (tra quelle disponibili), e di aggiungere i giocatori specificando il nome e la tipologia (umano o computer).
Per la creazione dell'interfaccia grafica si è utilizzata la libreria Swing e i suoi componenti e per organizzare la suddivisione del frame principale si ci è ispirati al gioco "Monopoli" raggiungibile al seguente link --> http://www.giochi.com/game/monopoli.
Nella fase di analisi e progettazione della view si è cercato di separare gli aspetti principali del gioco da quelli superflui e di secondaria importanza.
La progettazione della view è partita dalla classe principale, ovvero quella che contiene il tabellone e tutte le informazioni dei giocatori: la classe Index.
La classe Index è un JFrame con Layout impostato a BorderLayout organizzato in questo modo:

1. Nella sezione SOUTH c'è un panel contenente dei JButton che si attivano all'occorrenza per svolgere le azioni disponibili
2. Nella sezione EST troviamo le informazioni relative ai giocatori e alla banca; il panel della banca si trova fisso nel NORTH mentre i giocatori vengono creati dinamicamente, e in base al numero vengono creati i panel necessari e visualizzati in uno ScrollPane così da poter visualizzare tutte le informazioni necessarie senza necessità di doverle ridurre.
   Le informazioni che vengono visualizzate riguardano il nome, il valore dei soldi e delle proprietà(Ownership) possedute. Questo avviene attraverso delle label che hanno lo stesso colore del contratto visualizzato nel tabellone così da poter rendere il tutto di maggiore comprensione per l'utente.
3. Nel CENTER troviamo il tabellone vero e proprio, la plancia da gioco.
   E' un panel con Layout impostato a GridBagLayout dove le tessere vengono inserite soltanto nella cornice del tabellone.

L'entità fondamentale sulla quale si è posta maggiore attenzione è stata quella della singola tessera del tabellone.
Dopo aver individuato 5 tipi di tessere (Decks, Box, Land, Tax, Ownership) si sono raccolti tutti gli aspetti comuni delle tessere in una classe astratta AbstractGraphic da cui tutte le varie tipologie estendono, e ogni singola classe in base alla sua tipologia ha una struttura diversa.
Questa decisione è stata presa in modo tale che se in un futuro si voglia aggiungere un nuovo tipo di tessera basterà estendere dalla classe astratta per avere tutte le stesse funzionalità delle altre.
Nel Frame principale ogni singola sezione viene creata da una classe specializzata solo per quello in modo che se ci sono da fare modifiche in futuro basterà intervenire sulle singole componenti e non sul frame principale.
La view comunque comunica col controller, passando i giocatori, la loro tipologia e la versione di gioco scelta. Dopodiche il gioco è pronto a partire.



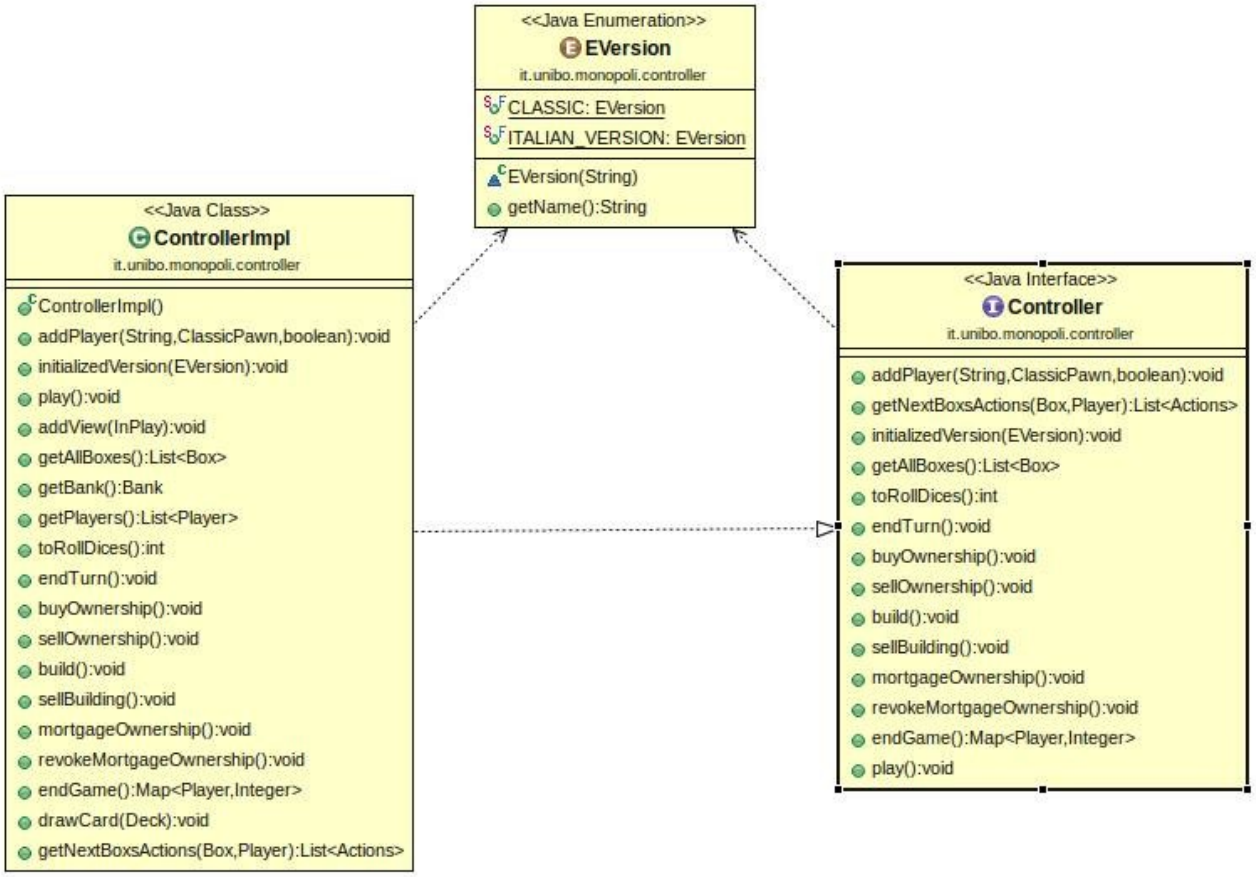
Questo UML rappresenta le classi adottate nella costruzione delle tessere del tabellone di gioco.



Questo UML rappresenta la classe principale Index che è connessa all'interfaccia utilizzata dal controller quando deve notificare alla view.

2.2.3 Controller

Il Controller è stato implementato in maniera tale da permettere il suo utilizzo anche in mancanza della View, quindi basterebbe richiamare singolarmente le azioni possibili per giocare senza la GUI. Infatti il Controller può eseguire solo ed esclusivamente le azioni consentite, che sono calcolate in base all'azione precedentemente svolta, salva e aggiorna internamente tutte le modifiche dovute alle varie azioni. Il Controller aggiunge ed inizializza tramite un apposito metodo i player, quest'ultimi richiamano tutte le informazioni necessarie in base alla versione usata, dopo di che richiama diversi metodi in base all'utilizzatore: - se l'utilizzatore è un computer calcola in automatico tutte le azioni possibili e in base a determinati criteri sceglie l'azione da compiere,se e quando compierla ,ed infine finisce il suo turno; - se invece l'utilizzatore è un giocatore “reale” è quest'ultimo che deve richiamare una tra le possibili azioni da eseguire; Con lo stesso metodo il Controller fornisce le azioni possibili alla View ,che sa cosi cosa poter richiamare e cosa invece non può eseguire, inoltre manda dei notify a quest'ultima per modificarne determinati stati. La fine del gioco avviene: - richiamando una determinata azione , che è sempre attiva, quindi in qualsiasi momento si può decidere di finire una partita,restituendone i giocatori ,o se uno, il vincitore. - se rimane solo un giocatore, restituendone la stampa. Il Controller è stato pensato in maniera tale che ad esempio un'aggiunta di un'azione comporti un piccolissimo dispendio di energia,in quanto basterebbe aggiungerla tra le azioni e renderla un'azione consentita solo nel caso scelto.



Questo UML rappresenta le classi principali del controller

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il test JUnit del model si trova in model.mainunits “TestModel”. Qui si è cercato di testare le principali azioni e regole del gioco stesso. Le sottoparti del test sono separate da una riga e sono precedute da un commento in cui si indica quali classi sono state testate.

Il test JUnit del controller è stato effettuato per verificare se effettivamente senza la presenza della view venivano svolte con successo solo le azioni possibili,sono stati effettuati due test, uno nel caso vado tutto a buon fine, e l'altro causando eccezioni.

3.1 Metodologia di lavoro

Ci siamo trovati inizialmente per discutere del progetto e per definire le linee guida del lavoro di ognuno di noi, dopo di chè abbiamo svolto separatamente ognuno la propria parte.
La suddivisione del lavoro si è attenuta a quanto dichiarato precedentemente:

- Model: Margherita Pecorelli
- View: Laura Tarsitano
- Controller: Giuseppe Vairo

Durante lo svolgimento autonomo delle diverse parti del progetto, siamo sempre rimasti in comunicazione soprattutto tramite BitBucket.
Quando le tre parti erano quasi totalmente operative, abbiamo iniziato anche a vederci di persona per unire il tutto. Durante quest'ultima fase, non eravamo più totalmente divisi sui tre fronti, ma si è cercato di risolvere in collaborazione tutti i problemi verificatisi. Quest'ultimi sono nati soprattutto quando si è andato ad unire il codice per testarlo nella sua completezza. Ci siamo infatti accorti che non sempre avevamo capito appieno il lavoro svolto dai compagni o la piena correttezza del codice. Inoltre ci siamo ritrovati a dover rivalutare la nostra suddivisione del MVC in quanto il model e il controller non erano ben distinti l'uno dall'altro.
In fase di sviluppo si è utilizzato molto BitBucket con Mercurial come DVCS per salvare e notificare agli altri ogni modifica importante fatta. In particolare al controller è stato molto utile per capire come implementare le dinamiche del gioco.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Secondo noi questo non è un “buon” progetto ma è una base solida per una possibile futura implementazione. Purtroppo anche le “poche” ore a disposizione e la poca competenza in materia non ci hanno permesso di fare tutto ciò che avremmo voluto.
Il team si era infatti riproposto, se ci fosse stato abbastanza tempo, di fare una seconda versione del gioco. L'implementazione più interessante, sarebbe stata quella descritta nel punto 2.2.1 (Model) con anche le carte da gioco (tipo Bang). Purtroppo non è stato possibile realizzarla, ma per evitare che ci fosse un'unica versione si è deciso, oltre a quella “Classic” di implementare anche quella “Italiana”. Sostanzialmente le regole e le classi utilizzate sono le stesse, ma in quella italiana c'è una strategia di “RollDice” differente e si sono aggiunte le azioni “ToStealMoney” e to “EvadeTaxes”.
Non siamo molto soddisfatti di questa seconda implementazione in quanto non c'è stato il tempo necessario da dedicare alla realizzazione della nostra idea iniziale (quella con le carte da gioco).
Sicuramente, un progetto per il fututo sarebbe realizzare questa seconda versione e magari anche altre di diversi temi (da Game of Thrones a Nightmare before Christmas).

Margherita Pecorelli: non sono particolarmente soddisfatta del mio lavoro, avrei avuto bisogno di molte più ore per implementare ciò che avevo pensato. Nel complesso non mi sembra troppo sbagliato ciò che ho fatto, ma riguardando il tutto ora mi sono resa conto che in alcune parti faccio riuso di codice. Inoltre, mi sono ritrovata a 2 giorni dalla consegna con un dubbio atroce: io ho inizializzato tutte le ownerships, i decks, le home, gli hotel e tutto il resto in ClassicStrategy e ItalianStrategy, ma poi ho pensato che in questo modo la GameVersion era praticamente inutile (tutti i suoi metodi richiamano quelli della strategy); quindi ho iniziato a pensare che forse tutte le inizializzazioni dichiarate prima, sarebbero dovute stare in una enum. Ho deciso di non provare nemmeno a modificare il tutto, ma ancora ora non sono sicura di cosa andava fatto. Spero possiate illuminarmi.
Per quanto riguarda il lavoro di gruppo, sono soddisfatta. È stata un'esperienza interessante, impegnativa, ma anche divertente. Abbiamo potuto confrontarci e lavorare insieme, cosa che non riesce sempre benissimo nello studio individuale. Probabilmente io ero quella che cercava di spronare a tirare un po' di più, ma non mi è sembrato che ci fossere dei dislivelli molo marcati. Siamo un buon gruppo di lavoro.

Laura Tarsitano: per quanto riguarda il lavoro svolto per il progetto non sono troppo soddisfatta. Purtroppo il tempo era poco e non sono riuscita ad organizzare il lavoro per come andava effettivamente organizzato e a dedicare il tempo che meritava l'implementazione di questo progetto. All'inizio sono stata troppo frettolosa nel voler visualizzare già l'interfaccia grafica senza una progettazione precedente. Successivamente però ho organizzato il lavoro per come andava fatto, ma il poco tempo e qualche lacuna in materia hanno rallentato di molto il mio lavoro. Non sono soddisfatta dell'interfaccia grafica ottenuta, avrei voluto avere maggior tempo da dedicare ai dettagli, sia implementativi che grafici. Sicuramente se avessi organizzato meglio il lavoro sin dal principio avrei raggiunto l'obbiettivo preposto.
Per quanto riguarda il lavoro col gruppo sono invece più soddisfatta rispetto che al mio lavoro individuale. Siamo stati capaci confrontarci e supportarci, sempre tutti disponibili ad aiutare gli altri.

Assemblare le parti non è stato per niente facile, ma tutti ci siamo impegnati fino alla fine per far funzionare al meglio il programma. All'interno del gruppo non penso di aver avuto un ruolo di fondamentale importanza, se non quello di supporto.

Giuseppe Vairo: non reputo il mio lavoro il più pesante del gruppo, in quanto la mia parte consisteva nel capire il model e dare istruzioni corrette sia al model che alla view, anzi non sono soddisfatto del mio operato, anche se credo che con più tempo a disposizione sarei riuscito ad implementare meglio il tutto. Ho cercato di aiutare il più possibile il gruppo.

4.2 Difficoltà incontrate e commenti per i docenti

Margherita Pecorelli: nel totale il corso mi è piaciuto molto e secondo me viene svolto molto bene. Le uniche cose che segnalo sono:

- 12 crediti? Almeno 18 :D
- Mercurial e BitBucket: sono stati spiegati troppo velocemente e non ci si è soffermati abbastanza. Mi rendo conto che il tempo sia poco e la quantità di lavoro molta, ma io personalmente ho riscontrato parecchie difficolta nell'utilizzarli. Solo ora inizio a comprenderli abbastanza bene.

Appendice A

Guida utente

Lo scopo del gioco è diventare il MONOPOLISTA.
Ad ogni turno sarà possibile tirare i dadi e in base alla casella su cui si è capitati si potranno compiere varie azioni specifiche.
Se si è su una terra:

- se è libera la si può comprare
- se appartiene ad un altro giocatore gli deve pagare il valore di rendita. Se il giocatore possiede tutte le terre di quel gruppo (es tutti i verdi se la terra su cui si trova è verde) allora il valore di rendita diventerà doppio. Se sulla terra sono stati costruiti degli edifici, il valore di rendita dipende da quanti edifici sono stati costruiti.
- se è del giocatore e quest'ultimo ha tutte le le terre di quel gruppo allora potrà costruire un edificio: si parte con le case, una per turno, fino ad un massimo di 4; poi si potrà costruire un albergo. Se invece non ha costruito nulla, allora può ipotecarla o venderla

Se si è su una stazione o una società:

- se è libera la si può comprare
- se appartiene ad un altro giocatore gli deve pagare il valore di rendita. Se il giocatore possiede più stazioni, allora il valore di rendita dipenderà da quante ne possiede, se invece possiede tutte le società, allora il valore di rendita sarà 10 volte il numero fatto coi dadi.

Se si è su una casella rappresentanti i mazzi di carte:

- si pesca una carta e si svolge l'azione descritta

Se si è su una casella rappresentante una tassa:

- si deve pagare la tassa

Se si è sull'area neutra (il parcheggio):

- non si deve eseguire alcuna azione

Se si passa dalla prigione/transito:

- non si deve eseguire alcuna azione

Se si è sulla casella della polizia:+

- si va direttamente in prigione e ci si deve rimanere per 3 turni. Si può comunque tirare i dadi: se i numeri sono doppi, allora si può uscire. In alternativa, se si possiede la carta “Esci gratis di prigione” si può uscire anche se i dadi non hanno fatto doppio

Se si passa dal via:

- si ritirano $200

La versione italiana cambia di poco:
Se si è su una casella rappresentante una tassa:

- si paga solo il 68% circa e i soldi non vanno alla banca ma vengono nascosti nell'area neutra (parcheggio)

Se si è sull'area neutra (il parcheggio) e ci sono dei soldi nascosti:

- si rubano i soldi

Quando si tirano i dadi, se si fa doppio e si è in prigione si può uscire, altrimenti si può tirare una seconda volta. Se si fa triplo, inceve, si va direttamente in prigione.