Relazione del progetto di Programmazione ad Oggetti

“Ludo Party”

19 febbraio 2024

Team di sviluppo:

Colavita Ilaria

Menghi Massimo

Pucci Mamone Francesco

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il software realizzato per il progetto si pone l'obiettivo di realizzare una versione digitale del gioco "Ludo", conosciuto anche come "Non ti arrabbiare", con l'aggiunta di booster e malus in stile party-game.

Il gioco è composto da un tabellone con 4 grandi caselle agli angoli (di colori diversi), che fanno da casa per ogni giocatore, e da percorsi composti da celle bianche e colorate su cui le pedine si muovono. La prima cella colorata, adiacente alla casa del giocatore, fa da inizio del percorso per ogni sua pedina (Figura 1.1). All'interno di ogni casa ci sono 4 pedine che possono muoversi lungo le caselle bianche (e successivamente lungo quelle colorate). Potranno partecipare 2 o 4 giocatori che, a turno, lanceranno un dado e faranno muovere le pedine lungo le celle (in senso orario) con un numero di passi pari a quelli indicati dal dado lanciato. Percorrendo le varie caselle (bianche), il giocatore potrà raccogliere delle monete e, fermandosi su altre (girgie) indicate con "*S*" (*shop*), potrà accedere al negozio per acquistare dei booster e dei malus. Terminato il suo giro lungo le caselle bianche, percorrerà le caselle del proprio colore, fino a raggiungere il centro del tabellone.

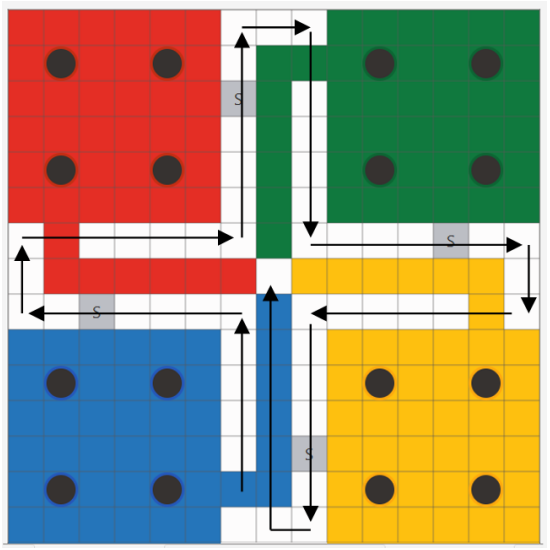


Figura 1.1: Tabellone del gioco, con case, percorso bianco e percorsi "safe"

Requisiti funzionali

- Il gioco dovrà permettere all'utente di inserire il proprio nome e successivamente scegliere il numero di giocatori della partita (due o quattro). Nel caso si scelgano due giocatori, uno sarà l'utente e l'altro sarà un giocatore pilotato dal computer; se i giocatori scelti sono pari a quattro, uno sarà l'utente e i restanti tre saranno giocatori controllati dal computer.
- Una volta iniziata la partita, il primo giocatore (quello dell'utente) dovrà tirare il proprio dado e, cliccando su una delle proprie pedine, farà la sua mossa nel tabellone muovendo la pedina scelta di un numero di celle pari al numero indicato dal dado appena lanciato, terminando il suo turno (nel gioco originale, per poter far uscire una pedina dalla propria casa, è necessario ottenere un sei; in "ludo party", al primo turno di ogni giocatore, il dado viene forzato ad ottenere tale numero per permettere a tutti di avere almeno una pedina sul tabellone).
- Il turno passa al giocatore successivo ("computer") che, scelta una sua pedina randomicamente, farà la sua mossa con la stessa modalità del giocatore "utente" e così fino a che uno dei giocatori non riesce a portare tutte le sue pedine al centro del tabellone, guadagnando la vittoria.
- I giocatori muovono le pedine in senso orario lungo le celle di colore bianco, finché non raggiungono le rispettive celle "*safe*" (le celle dello stesso colore della casa del giocatore). Una volta percorse queste ultime celle, la pedina raggiungerà il centro del tabellone. Quando una pedina raggiunge una cella in cui è già posizionata una pedina avversaria, può mangiarla e farla tornare nella sua casa, nella posizione iniziale. Nelle celle "*safe*" però, tutte le pedine sono salve, ovvero non possono essere mangiate, indipendentemente dal colore della cella "*safe*".
- Vince la partita il primo giocatore che fa arrivare tutte le proprie pedine alla casella finale, situata al centro del tabellone.
- Durante la partita, è possibile raccogliere delle monete (nascoste sotto le celle) che andranno ad incrementare la quantità di *ludollari* del giocatore. I soldi raccolti lungo il percorso permettono di acquistare dei bonus o dei malus nello shop.
- Quando una pedina raggiunge una cella "*shop*", viene abilitato il negozio e il giocatore ha la possibilità di aggiungere al proprio inventario degli oggetti monouso che può usare per avere dei benefici o per mettere in difficoltà gli avversari.

Requisiti non funzionali

L'applicazione dovrà:

- mantenere traccia su file dei punteggi dei diversi giocatori

1.2 Analisi e modello del dominio

Il software sarà in grado di modellare il dominio di una partita, generando le entità utili (giocatori, pedine, dadi, partita, tabellone, celle e shop in cui acquistare bonus e malus). Terrà inoltre traccia delle monete raccolte, che saranno utilizzate per salvare il punteggio del giocatore che guadagna la vittoria.

Il giocatore "umano" inizierà il gioco lanciando il proprio dado, e successivamente muoverà la sua pedina nel tabellone, terminando il turno alla pressione del tasto Invio. Il turno verrà poi passato ai giocatori controllati dal computer, che autonomamente giocheranno ognuno il proprio turno. Terminato il turno dell'ultimo giocatore "computer", sarà nuovamente possibile lanciare il dado per l'utente e continuare la partita.

Gli elementi del dominio del problema sono sintetizzati in Figura 1.2.

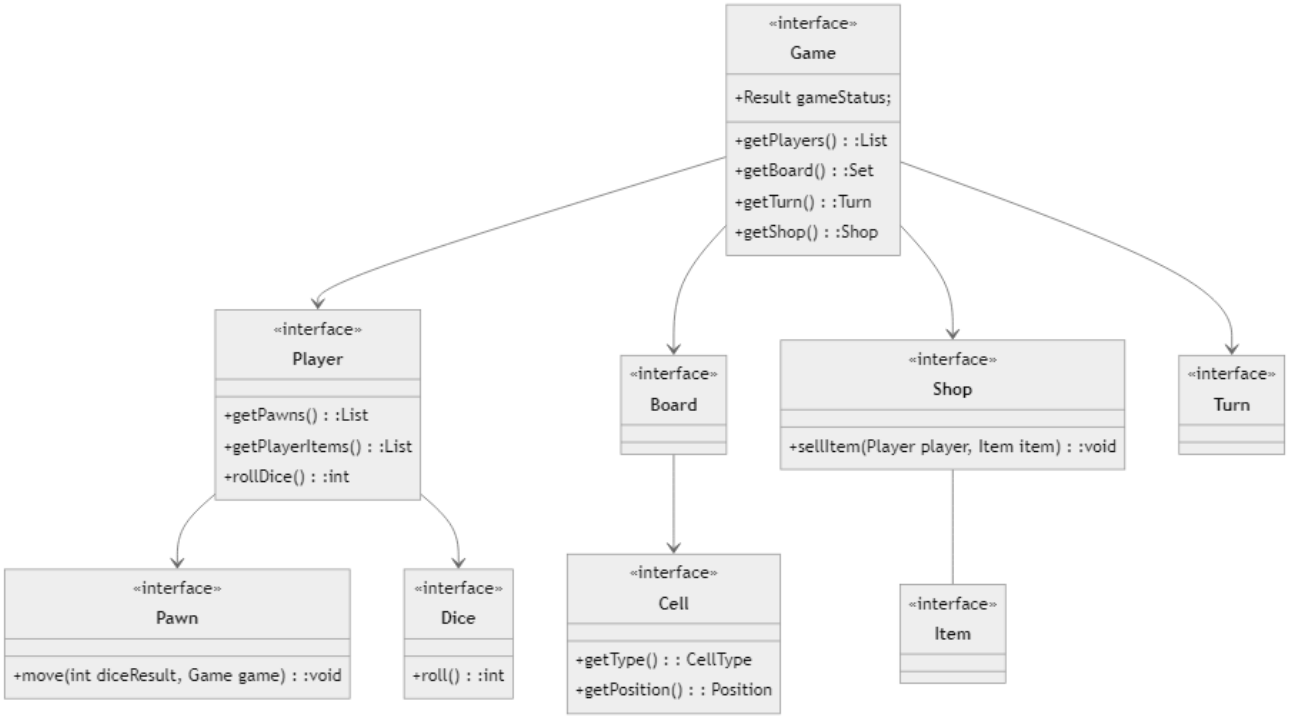


Figura 1.2: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

L’architettura di Ludo Party segue il pattern architetturale Model-View-Controller (MVC).

La BoardScene è la rappresentazione grafica del model della Board (il tabellone che contiene le pedine della partita); essa funge da interfaccia per il giocatore. Il *Game* si occupa di gestire tutti gli elementi del gioco e farli interagire fra loro. Il controller coordina model e view. La view intercetta un evento di tipo click sul dado e sulla pedina ed anche l'evento di pressione del tasto invio per passare il turno al giocatore successivo. Quando sopraggiunge un evento, la view aggiorna il model e, viceversa, quando il model viene aggiornato, si aggiorna anche la view.

I punti di ingresso che mettono in comunicazione le tre parti dell'architettura, sono: *Controller*, *BoardScene* (View), *Game* (Model) come mostrato in Figura 2.1.

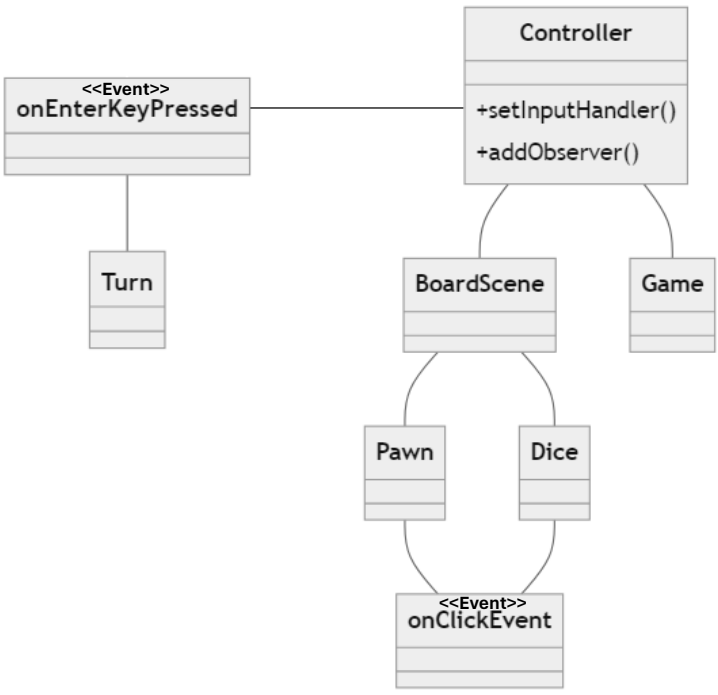


Figura 2.1: Schema UML architetturale di Ludo Party

Il model e il controller sono stati realizzati per essere quasi del tutto indipendenti dalla view, rendendone facile un riutilizzo ed un eventuale sostituzione dell'attuale libreria grafica con un'altra.

2.2 Design dettagliato

A. Colavita

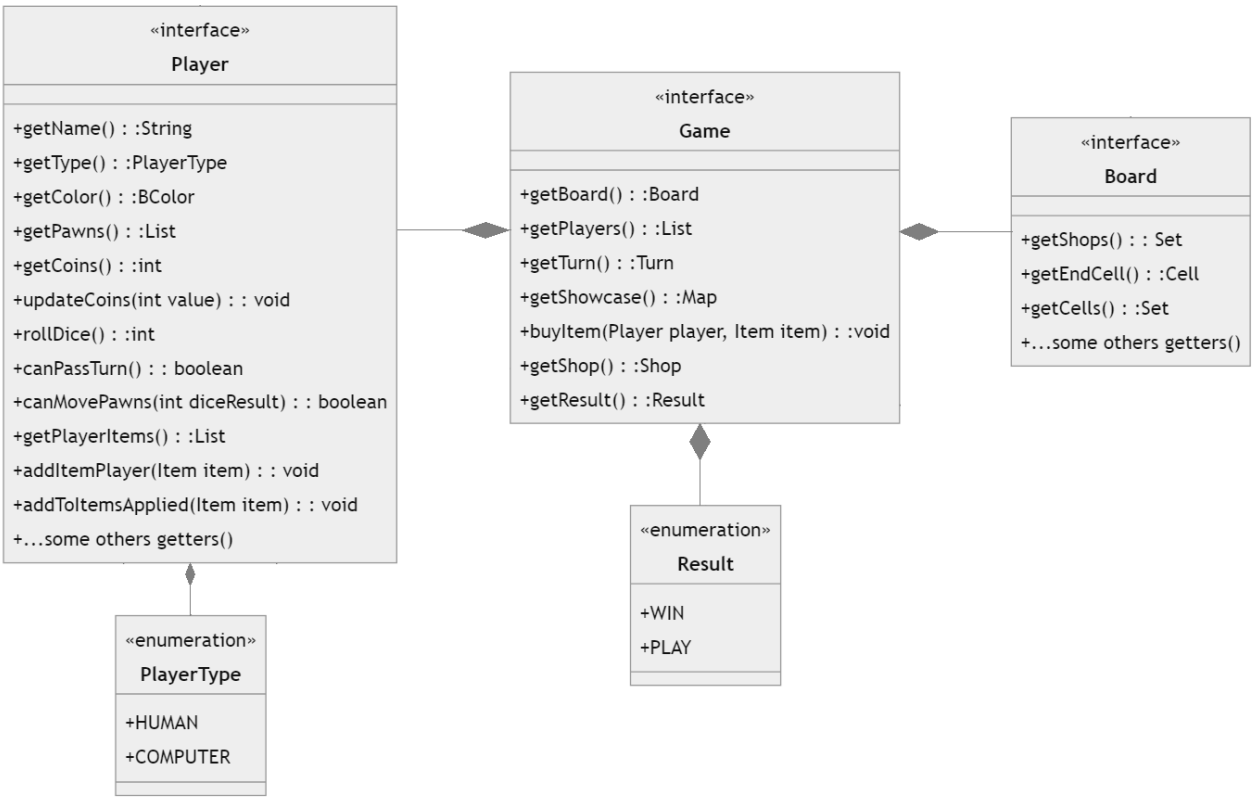


Figura 2.2: Schema UML del model del Player, del Game e della Board

Problema Il gioco necessita di un tabellone contenente le pedine che possono essere mosse da un giocatore. Ogni giocatore ha una sua casa e un colore e, quando è il suo turno, lancia il proprio dado e muove una delle sue pedine di un numero di celle pari al numero indicato dal dado. Lungo il percorso, le pedine raccolgono delle monete che serviranno al giocatore per poter acquistare degli item dallo *shop* (i bonus/malus) e che faranno da punteggio in caso di vittoria. Quando uno dei giocatori guadagna la vittoria, può salvare il proprio punteggio su un file.

Soluzione Per la parte di model, ho realizzato le interfacce per il Player, il Game e la Board, con le relative implementazioni (Figura 2.2) Ho creato poi la parte grafica dei Player che permette di aggiornare l'immagine del dado lanciato e la label contenente la

quantità di ludollari raccolti; per coordinare view e model ho utilizzato il pattern Observer (Figura 2.3).

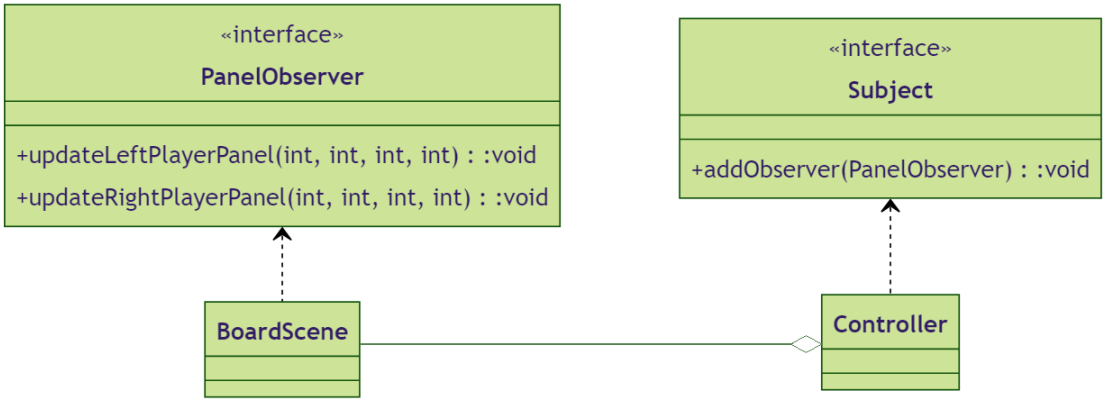


Figura 2.3: Schema UML del pattern Observer

La creazione dei pannelli a destra e a sinistra del tabellone è stata realizzata con una classe astratta da estendere come mostrato in Figura 2.4. Ciò permette di evitare la ripetizione di codice nell'implementazione dei due pannelli.

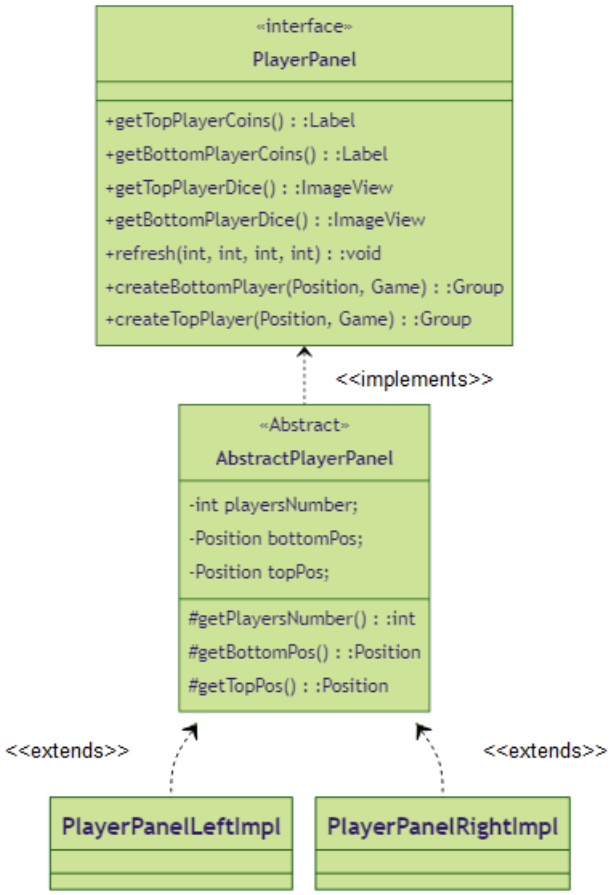


Figura 2.4: Schema UML del PlayerPanel e relative implementazioni

Il salvataggio dei punteggi viene effettuato su un file tramite la classe ScoreManager che è stata realizzata attraverso il pattern Singleton, scelto poiché permette di controllare un potenziale accesso concorrente ad una risorsa condivisa (Figura 2.5).

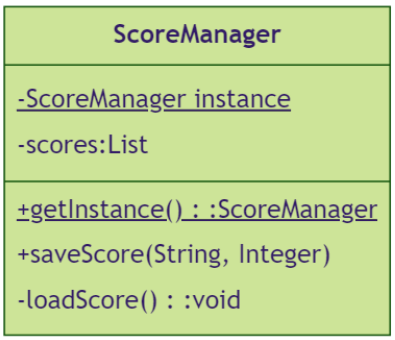
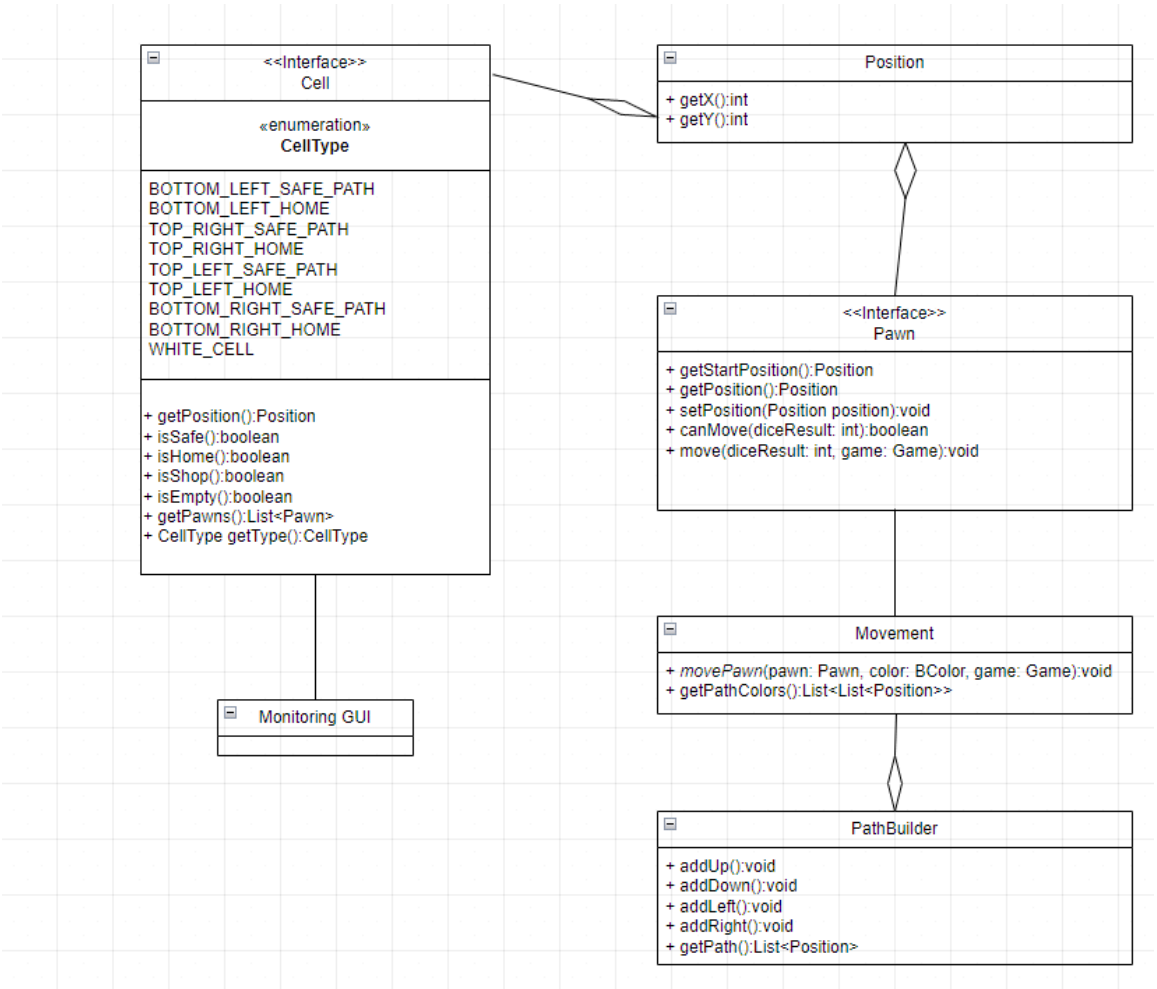


Figura 2.5: Schema UML del pattern Singleton

B. Menghi

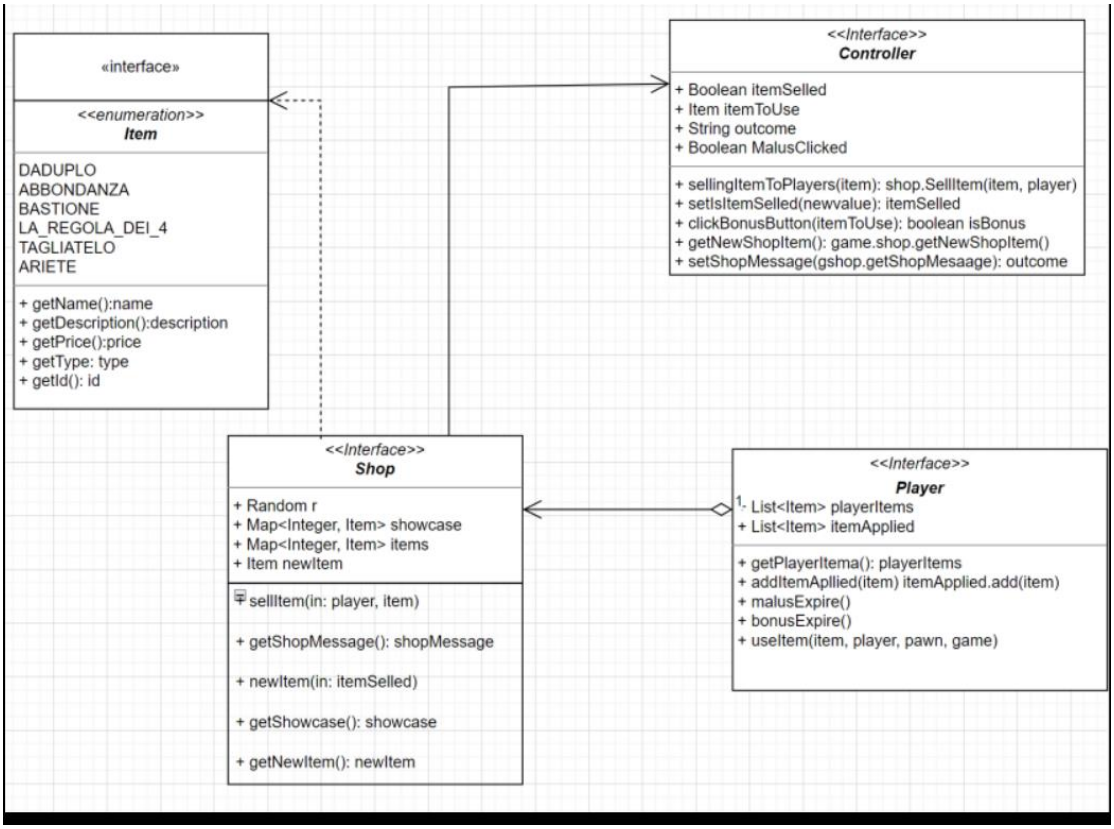


PROBLEMA: Nella board del gioco “Ludo-Party”, si deve avere la possibilità di muovere le pedine, in un determinato "percorso" deciso dal colore del player, il quale ha il controllo

delle sue 4 pedine. I due o i quattro percorsi dei player hanno in comune più celle, queste ultime sono identificate da una posizione, e in caso una pedina raggiunge la pedina di un altro giocatore, se da sola e non popolante una cella "safe" la pedina dell'altro giocatore viene riportata alla sua home, ma non nel caso abbia bastione, come power-up.

SOLUZIONE: A livello di patterns non ho utilizzato molti design patterns, ho fatto uso di una classe "PathBuilder" per appunto costruire un percorso immutabile delle pedine per ogni giocatore, ho fatto uso di una classe di utility Movement (quest'ultimo non è da ritenersi un pattern vero e proprio), per gestire il movimento delle pedine nella board, la cui parte è fondamentale per il funzionamento corretto dell'applicazione.

C. Pucci Mamone



PROBLEMA: A differenza della versione fisica del gioco qui abbiamo stabilito di creare degli item con cui i giocatori potevano interagire in maniera diversificata al semplice table-game. La prima sfida è stata ideare gli oggetti da zero, in modo tale che fossero compatibili con la programmazione e l'obbiettivi dello studio di questo corso.

Oltre alla creazione andavano ideati dei sistemi di "distribuzione" e mantenimento di questi ultimi, il quale mi portava spesso ad avere una porzione dello sviluppo dipendente da altre classi.

Dopo ciò si doveva andare a modificare tutte quelle che erano le possibili interazioni degli Item con il resto del programma.

SOLUZIONE: Sono dalla creazione di Item come classe specifica con le proprie caratteristiche, queste ultime andavano ad interagire in maniera particolare con lo shop (distribuzione) e l'inventory dei player (mantenimento) oltre all'utilizzo di semplici liste per l'aggiunta o la rimozione degli oggetti applicati allo specifico giocatore.

le interazioni andavo a toccare singolarmente il loro campo specifico (gli item che modificavano specificamente il roll dei dadi).

Capitolo 3

Sviluppo

3.1 Testing automatizzato

I test sono stati effettuati sul modello utilizzando JUnit.

Colavita:

- test su Player
- test su Board

Pucci Mamone

- test su Item

3.2 Metodologia di lavoro

Dopo la fase di progettazione, sono state definite le principali interfacce e successivamente ognuno ha realizzato il proprio codice.

Suddivisione del lavoro iniziale:

Colavita: gestione dei giocatori, dei punteggi; gestione delle case contenenti le pedine ad inizio partita

Piolanti: gestione della turnistica e dell'ordine delle azioni che i giocatori devono compiere nel proprio turno; gestione dei dadi

Pucci Mamone: creazione dei booster/malus e gestione di essi attraverso lo shop e le interazioni con il dado, i giocatori e le pedine

Menghi: gestione delle celle da percorrere dalle pedine di ciascun giocatore

Piolanti ha deciso di abbandonare il progetto. La sua parte è stata fatta in collaborazione, una parte da lui e la restante dagli altri membri del team.

A. Colavita

Ho realizzato la parte relativa ai giocatori ma non era ancora disponibile la parte di shop e di interazione con gli item (bonus/malus), che sono stati aggiunti solo successivamente. Mi sono poi occupata di realizzare la gestione della partita e la creazione della board su cui posizionare le celle di tipo “casa” dei giocatori. La gestione dei punteggi era una parte piuttosto scollegata dal gioco per cui non c’erano particolari dipendenze.

La parte di gestione dei turni e dei dadi, mancando Piolanti, è ricaduta un po' su tutti i restanti componenti, rendendo il controller una parte realizzata in comune.

È stato utilizzato il DVCS Git, lavorando principalmente sul branch main e a volte su branch secondari, effettuando poi il merge sul main. È stato effettuato anche qualche cherrypick da un branch secondario al main.

B. Menghi

Mi è stato chiesto di gestire le pedine, le loro interazioni nelle o con le celle e principalmente gestire il movimento delle pedine, la quale ha una lieve dipendenza in uscita nella gestione dei turni e sul funzionamento dei power-up ad esempio "Bastione", nel caso di dover controllare che la pedina nemica candidata ad essere mangiata/riportata alla home, deve prima sapere se il player proprietario ha il Bastione attivo. Quindi nel mio caso le dipendenze sono minime. Le parti chieste a me da gestire non richiedevano particolare attenzione al riuso tramite interfacce e a livello architetturale, la mia parte è risultata piuttosto scorrevole, e non ho riscontrato dimenticanze significative nell'implementazione della mia parte.

C. Pucci Mamone

La gestione degli item e dello shop erano quasi unicamente dipendenti al player, in quanto la compravendita e l'utilizzo di questi era dovuto unicamente a lui e a cosa interagiva.

Sono dovuto stare dietro ai continui cambiamenti delle varie porzioni di codice per assicurarmi che l'oggetto compisse correttamente il suo giro di interazioni, se poteva andare nell’inventario, se interagiva con il player corretto (non avversario nei casi dei bonus) e soprattutto che il suo effetto durasse solamente i turni necessari.

3.3 Note di sviluppo

A. Colavita

Si è scelto di utilizzare la libreria grafica JavaFX e la libreria log4j per il logging.

- JavaFX: permalink GitHub
- log4j: permalink GitHub

Feature avanzate del linguaggio:

- Stream: permalink GitHub
- lambda expressions: permalink GitHub

B. Menghi

È stata utilizzata la libreria JavaFX che si occupa della parte grafica della view.

Ho fatto uso dei Circle e delle loro funzionalità per traslarli nella view.

C. Pucci Mamone

È stata utilizzata la libreria JavaFX per la creazione dello shop e degli item.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

A. Colavita

La fase di progettazione non è stata troppo complicata, si sono definiti subito gli aspetti principali da modellare. La fase di sviluppo, invece, è stata un po' più complicata in quanto un gruppo da 4 non è facilmente coordinabile in mancanza di una figura apposita.

Ritengo il codice sviluppato di buona qualità, avendo utilizzato dei pattern e qualche costrutto avanzato. Sicuramente è migliorabile, rendendo meno oneroso il reperimento delle risorse e/o utilizzando maggiormente i costrutti avanzati.

B. Menghi

A livello logico/architetturale mi sono ritrovato in quello che facevo, essendo in grado di creare uno schema UML logicamente funzionante e adatto al problema imposto, ovvero nella gestione delle pedine, dei movimenti delle pedine, e delle celle, utilizzate in gran parte nella sezione view del modello MVC, mentre ho riscontrato difficoltà non indifferenti nella gestione di quelle piccole dipendenze, anche causate dall'indecisione sul chi dovesse implementare alcune piccole parti. Un'altra difficoltà è stata l'utilizzare i design patterns e diverse funzionalità avanzate a cui, nella maggior parte dei casi ho dovuto rinunciarne.

C. Pucci Mamone

Non posso definire il mio lavoro perfetto, ho dovuto più volte ritoccare il codice per renderlo meno ridondante o più semplice dove necessario, per il futuro terrò a mente come ho svolto questo progetto in particolar modo riguardo l'efficienza del tempo usato nel ragionamento e nella creazione di strutture logiche. Mi è capitato un paio di volte di dovermi fermare a ragionare più del necessario dopo aver scritto pezzi di codice, portandomi a dover rilavorare se non a ricreare da capo siccome non avevo svolto un lavoro corretto fin dall'inizio.

A Guida utente

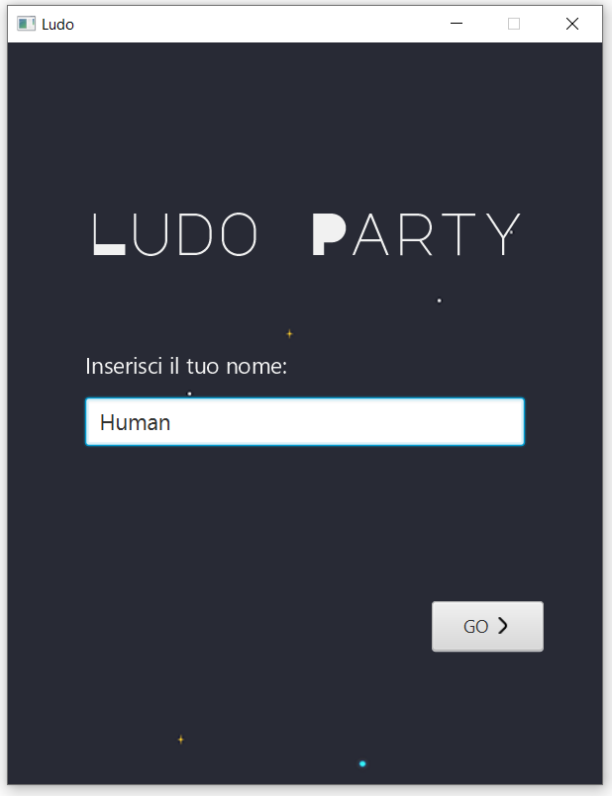


Figura A.1: Prima Schermata

Figura A.2: Seconda schermata

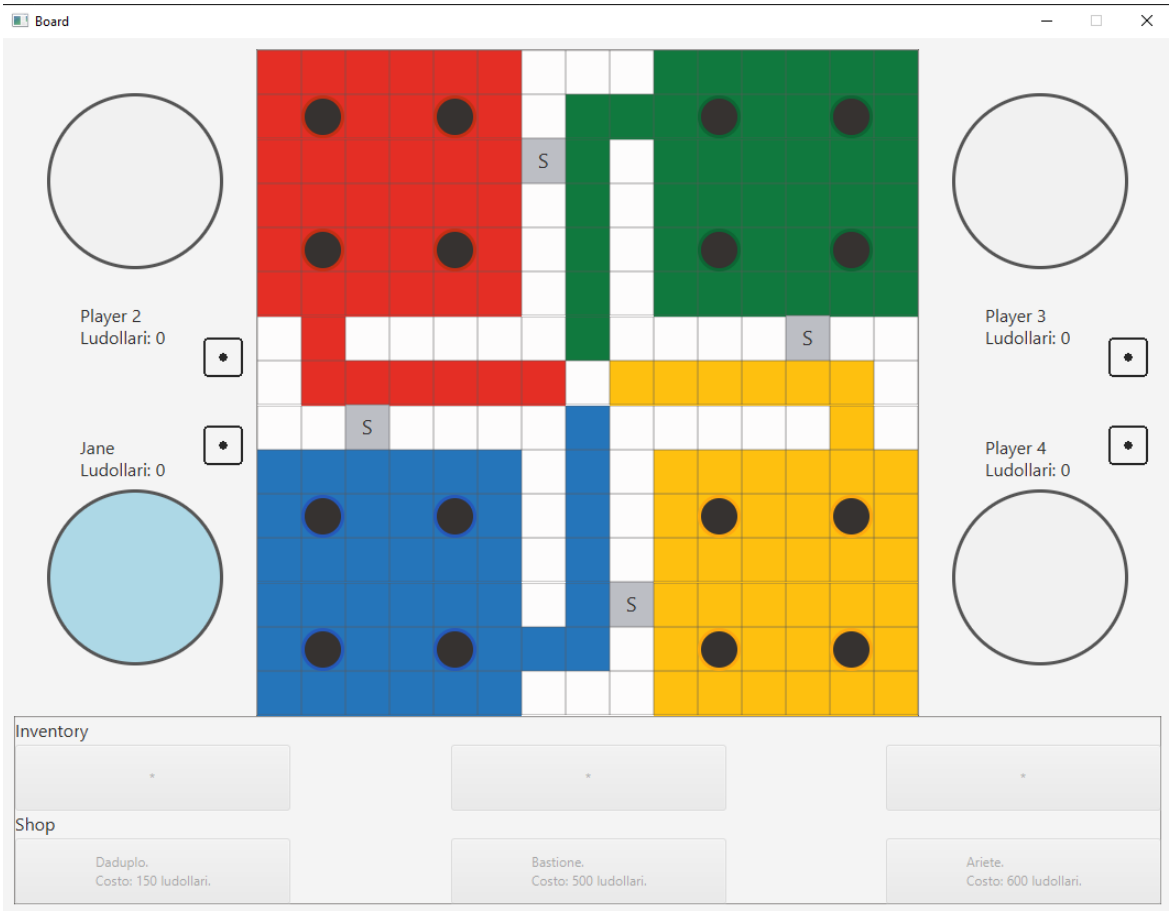


Figura A.3: Schermata di gioco

Una volta inserito il proprio nome e scelto la modalità con cui giocare (2 o 4 giocatori), l'utente può iniziare il proprio turno eseguendo le azioni sottoelencate:

- lanciare il proprio dado
- muovere una delle pedine
- premere il tasto Invio per passare il turno

La caratteristica unica di *Ludo Party* sono gli *Item*; questi possono essere usati dal giocatore dopo averli comprati nello *shop*, rappresentato graficamente dai tre bottoni inferiori interagibili unicamente quando una sua pedina si fermerà sulla casella specifica. Questi verranno aggiunti all'inventario del giocatore, i tre bottoni superiori con cui potrà interagire durante il proprio turno, quando avrà la possibilità economica o lo spazio e avrà cliccato il bottone associato all'oggetto interessato. L'inventario può contenere un massimo di tre oggetti e dovranno essere tutti diversi.

Gli item son di due categorie: *bonus* e *malus*; i primi si utilizzano a inizio del proprio turno per avere dei potenziamenti mentre i malus vanno a mettere in difficoltà gli avversari per aiutare l'utente a raggiungere la vittoria. Gli item sono utilizzabili una sola volta, dopo averli consumati spariranno dall'inventario.

Qui una lista degli Item:

Bonus

- Daduplo: al prossimo tiro di dado ne lancerai due.
- Abbondanza: i coin raccolti raddoppiano per questo turno.
- Bastione: fino al tuo prossimo turno non sei targhettabile dai malus degli avversari e le tue pedine non possono venir mangiate.

Malus

- La regola dei 4: riporta una pedina avversaria indietro di 4 caselle.
- Tagliatelo: il prossimo tiro di dado dell'avversario sarà dimezzato.
- Ariete: disattiva anticipatamente il bastione di un avversario.