UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

NEW MONOPOLY

Progetto a cura di
Christian Pozzoli, *845195*
Emanuele Papa, *844888*

Università degli Studi di Milano Bicocca
Progetto di Ingegneria del Software
Anno Accademico 2021-2022

Introduzione

I problemi principali di Monopoly

Monopoly è un gioco da tavolo basato a turni in cui da 2 a 6 giocatori dovranno acquistare, scambiare e vendere proprietà, costruire case e hotel e raccogliere i soldi dell'alloggio dai giocatori avversari muovendosi sul tabellone di gioco.
L'obiettivo del giocatore sarà quello di portare gli avversari in bancarotta e rimanere l'ultimo in gioco.
Tuttavia, il gioco del Monopoly ha una serie di limitazioni riguardanti l'economia della vita reale.
In primo luogo il tabellone di gioco fisso: il giocatore dovrà dunque muoversi su di esso in una sequenza di spazi prestabiliti e dovrà pagare/guadagnare la stessa quantità di soldi per tutto il resto del gioco.
In secondo luogo è assente un sistema di ricompense in gioco: nel pagare gli affitti il giocatore non ottiene alcun punto da poter integrare con la valuta del gioco per poi comprare proprietà e pagare altri affitti successivamente.
Lo scopo di questo progetto è quello di sviluppare una versione altamente configurabile del gioco del Monopoly che possa ovviare queste limitazioni.

Descrizione Progetto - New Monopoly

New Monopoly è una versione elettronica altamente configurabile basata sul gioco del Monopoly.
Lo scopo del gioco è quello di portare i giocatori avversari in bancarotta e di rimanere l'unico giocatore in controllo dell'economia.
Il progetto riguarda la realizzazione di un gioco client-server basato a turni ad alta personalizzazione giocabile dagli utenti tramite pagine Web.
Il gioco implementa tutte le funzionalità base della versione standard del Monopoly (vedere le regole a monopoly-rules.pdf).
Il gioco inoltre permetterà all'utente di:

- Abilitare, in un determinato lasso di tempo, la variazione casuale dei prezzi (acquisto, costruzione case/hotel e ipoteca), affitti, tassi e interessi di una sequenza casuale di caselle.

- Abilitare la possibilità di aggiungere un sistema di ricompense tra giocatori basato su programmi di fidelizzazione realmente esistenti.

- Modificare il numero massimo di giocatori possibile della versione standard del gioco.

- Partecipare a una partita già in corso.

- Modificare il livello di difficoltà del gioco su 3 diversi livelli (facile, medio, difficile). È anche possibile scegliere un livello di difficoltà personalizzato configurato dall'utente.

Componenti software utilizzate

Java

Il progetto è per gran parte scritto utilizzando linguaggio di programmazione Java, nello specifico Java 11, in modo da poter assicurare compatibilità con vari strumenti software e librerie a disposizione.

Java Script

L'approccio client-server per lo sviluppo dell'applicativo prevede di creare un'interfaccia grafica affinché l'utente ci possa interagire. Questo ha reso necessario sfruttare JavaScript in modo da rendere le pagine web dinamiche e interattive.

JUnit

Framework utilizzato per le fasi di testing. Rende disponibile una serie di librerie e tool per il testing. Le fasi di test sono state effettuate per ogni feature implementata durante tutto lo sviluppo (ad esclusione dello sviluppo front-end).

Jacoco

Plug-in Maven, permette la generazione di report sulla coverage del codice, in modo da mantenere monitorata la copertura dei test durante lo sviluppo e favorire meccanismi di continuous integration.

Maven

Diffuso build system per Java, permette l'automazione di processi di build, test e integrazione di framework e librerie.

Spring

Nello specifico il progetto Spring boot, gestisce meccanismi utili di dependency injection e viene sfruttato come base per progetti client-server e per automatizzarne le configurazioni. In particolare incorpora Tomcat e Jetty sfruttati dal framework Vaadin per la creazione di server web.

Git

Software diffuso di versionamento del codice, come repository è stata utilizzata la piattaforma github.

SonarCloud

Versione cloud di SonarQube, strumento per verificare la qualità del codice, coverage e smell. Offre una dashboard con un report completo sulla qualità del progetto.

Vaadin

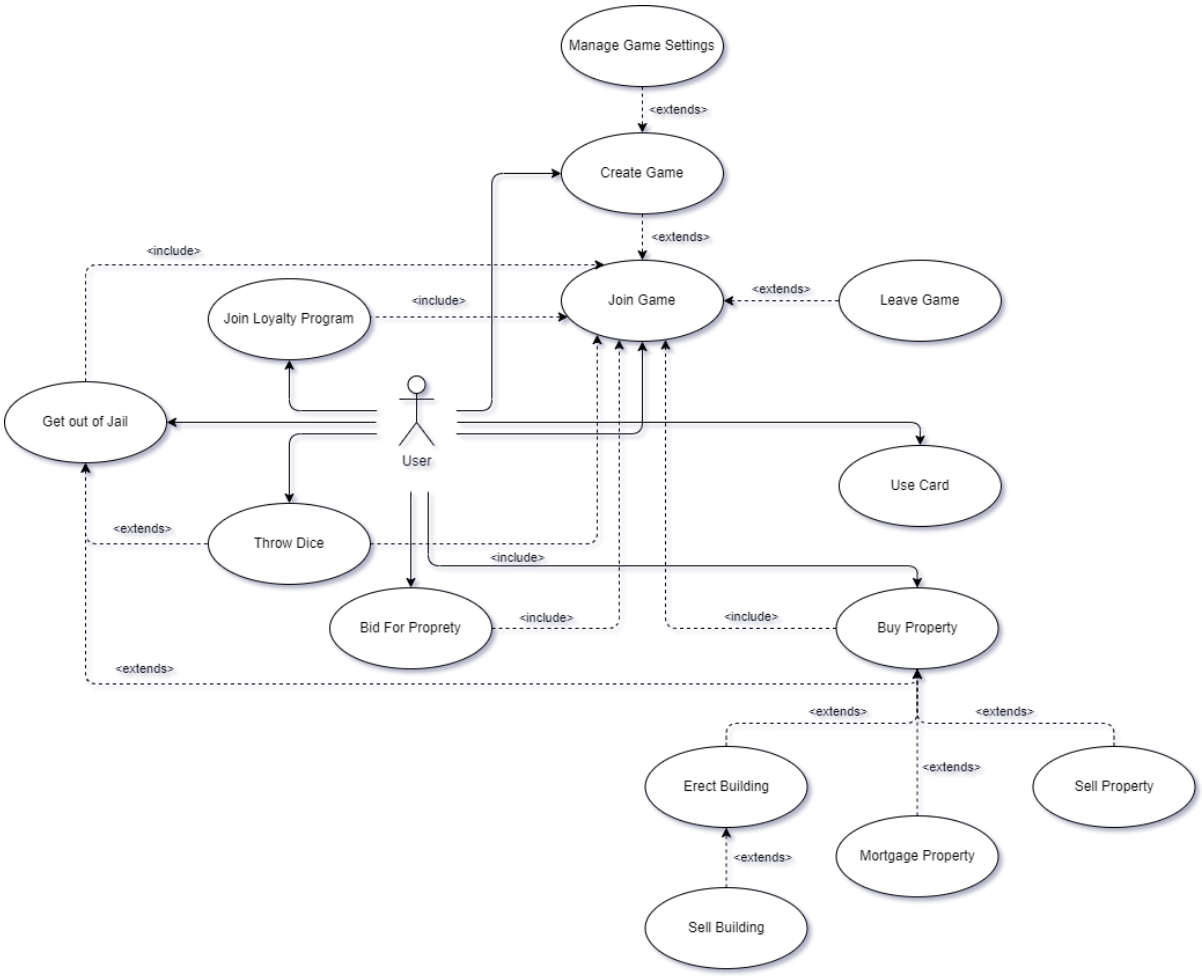
Framework che offre la possibilità di sviluppare applicazioni client-server sfruttando Java come intera code base, sia per il back-end che per il front-end.

Jackson

Libreria dedicata alla deserializzazione/serializzazione di file JSON.

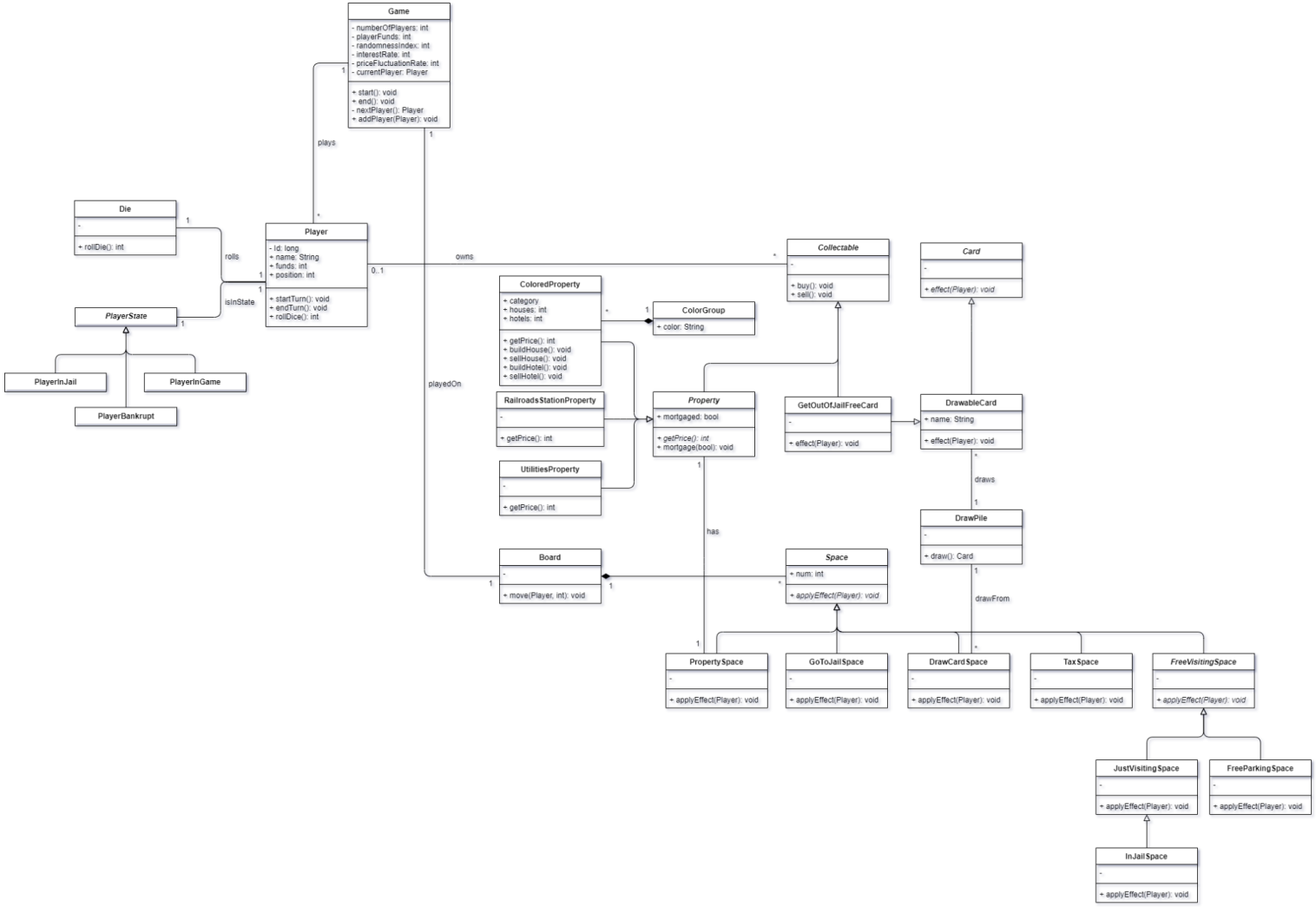
Diagrammi

Diagramma dei casi d’uso



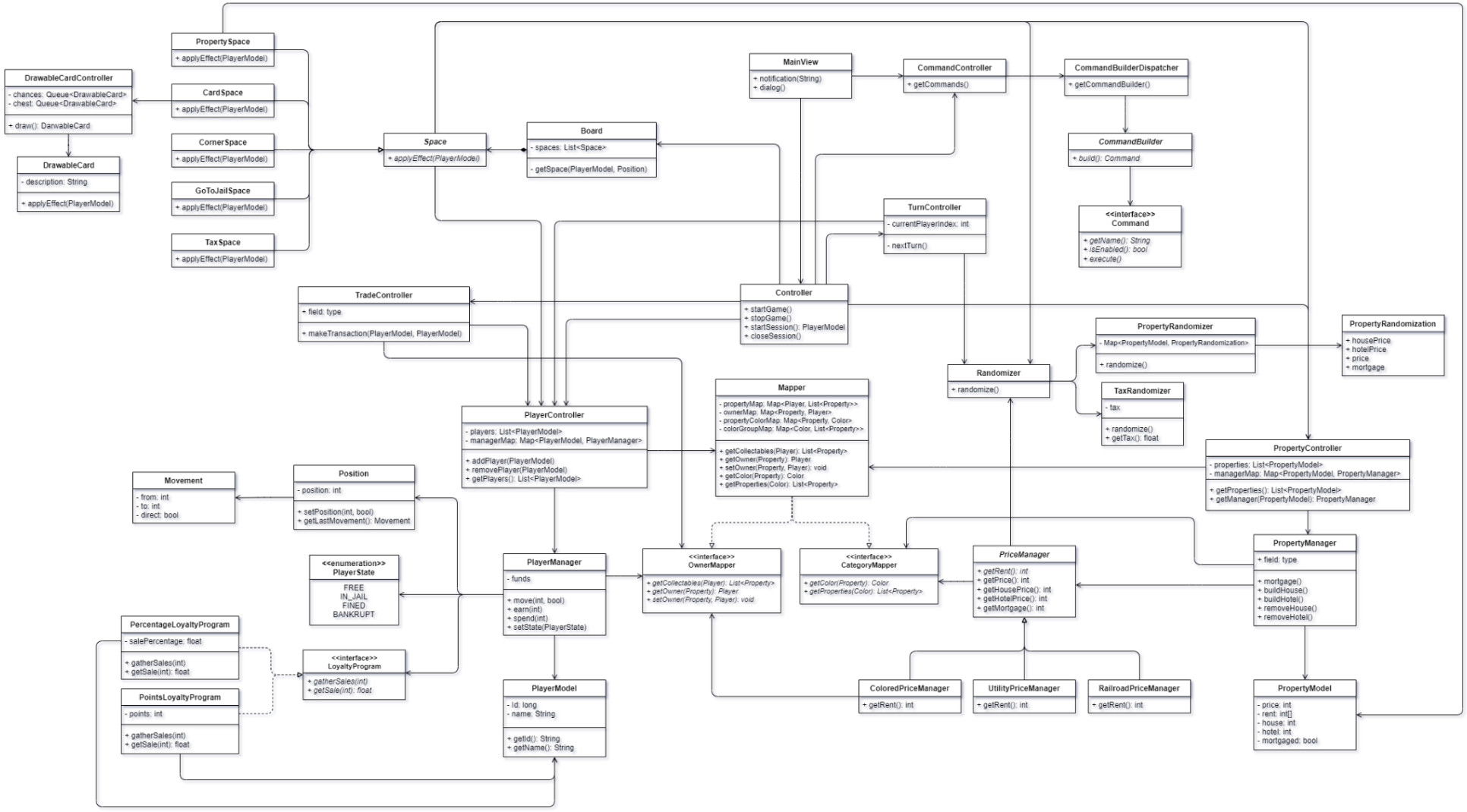
Cliccare sull’immagine per aprire il link

Diagramma delle classi a livello di dominio



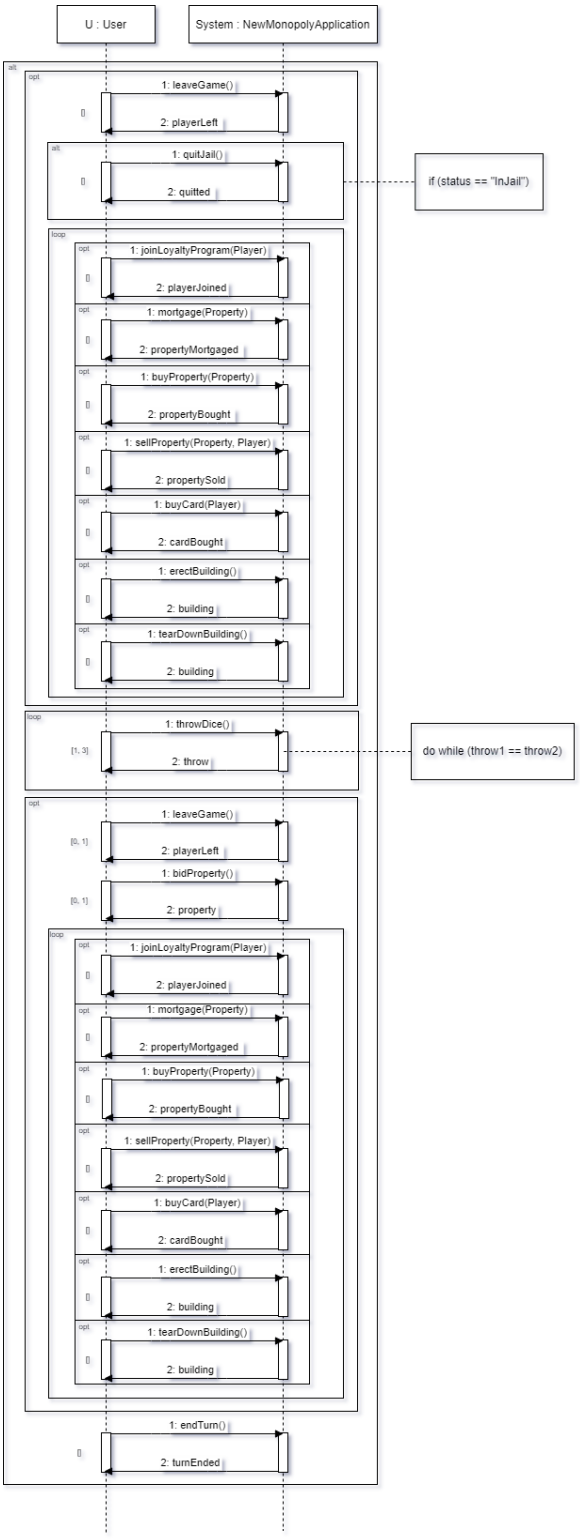
Cliccare sull'immagine per aprire il link

Diagramma delle classi a livello di progettazione



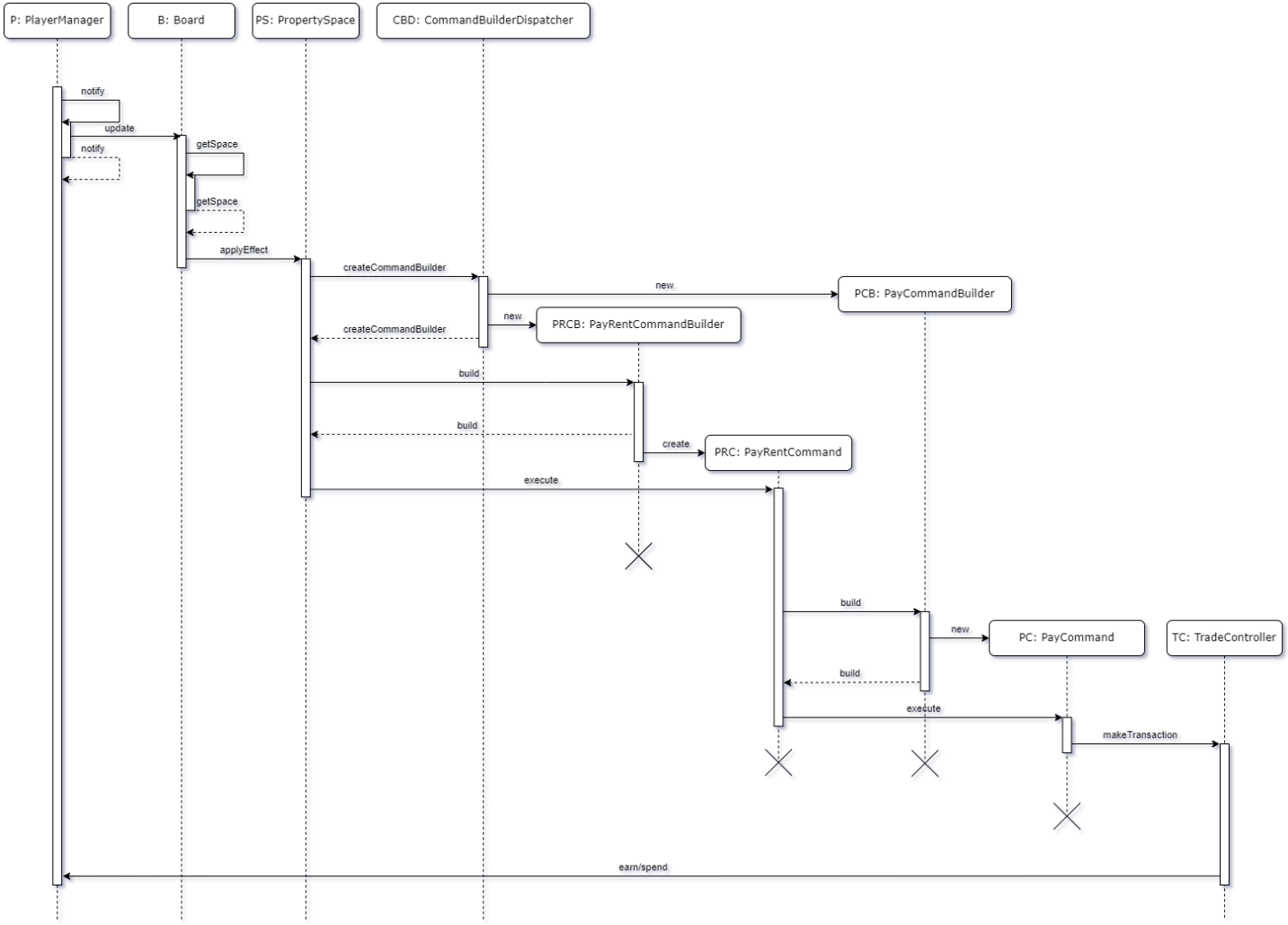
Cliccare sull'immagine per aprire il link

Diagramma SSD



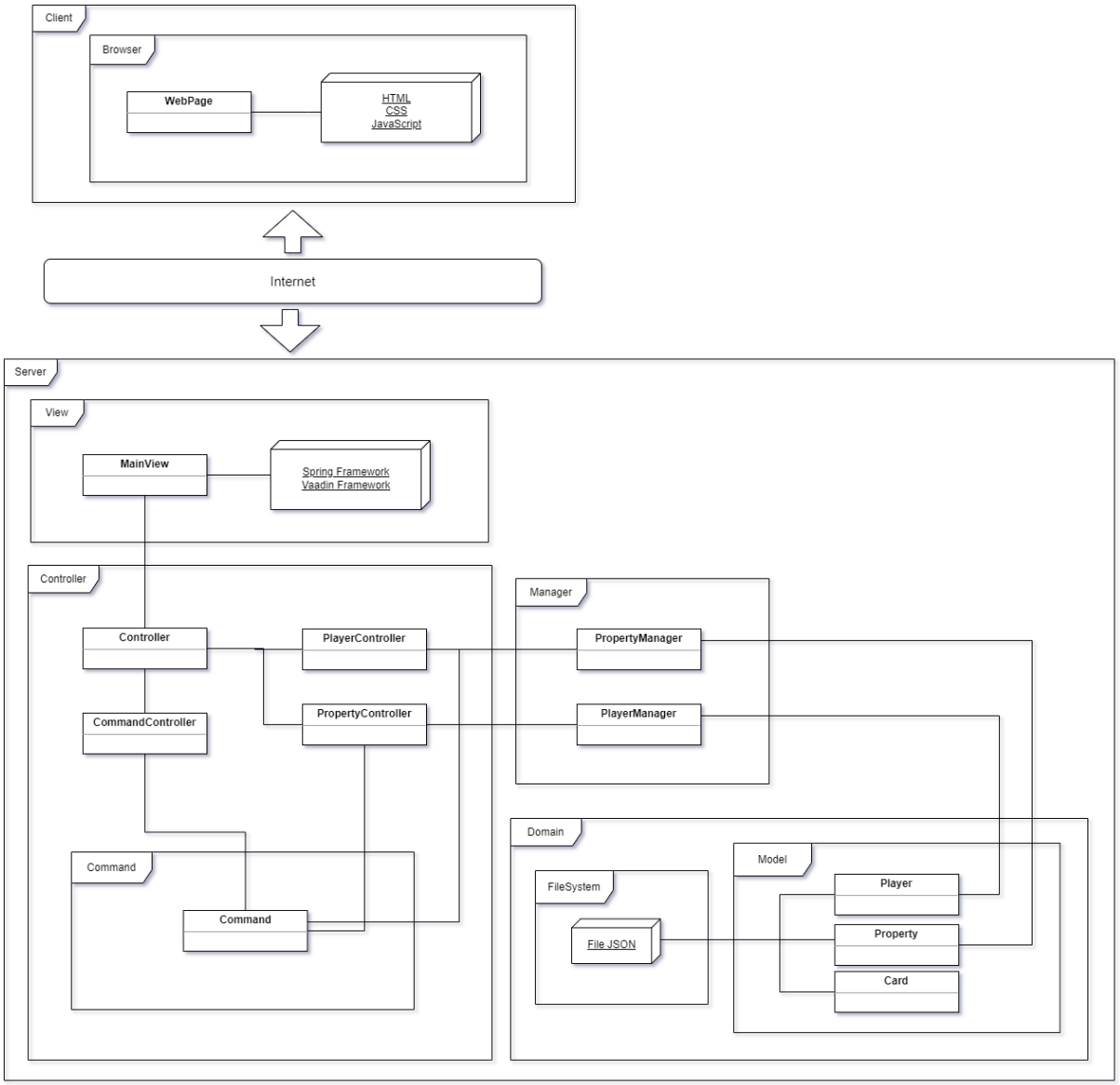
Cliccare sull'immagine per aprire il link

Diagramma di sequenza di progettazione



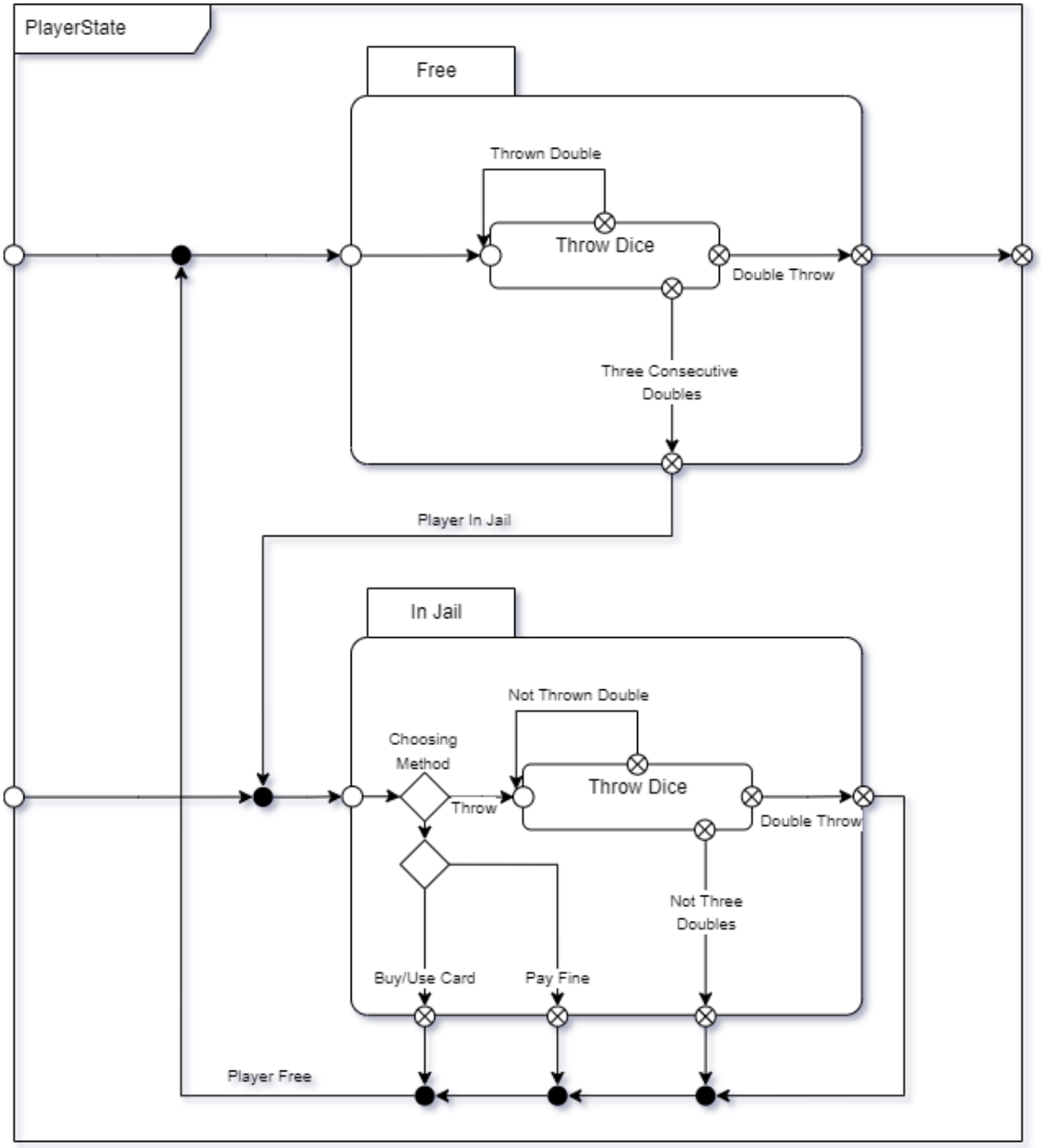
Cliccare sull'immagine per aprire il link

Diagramma dell’architettura software



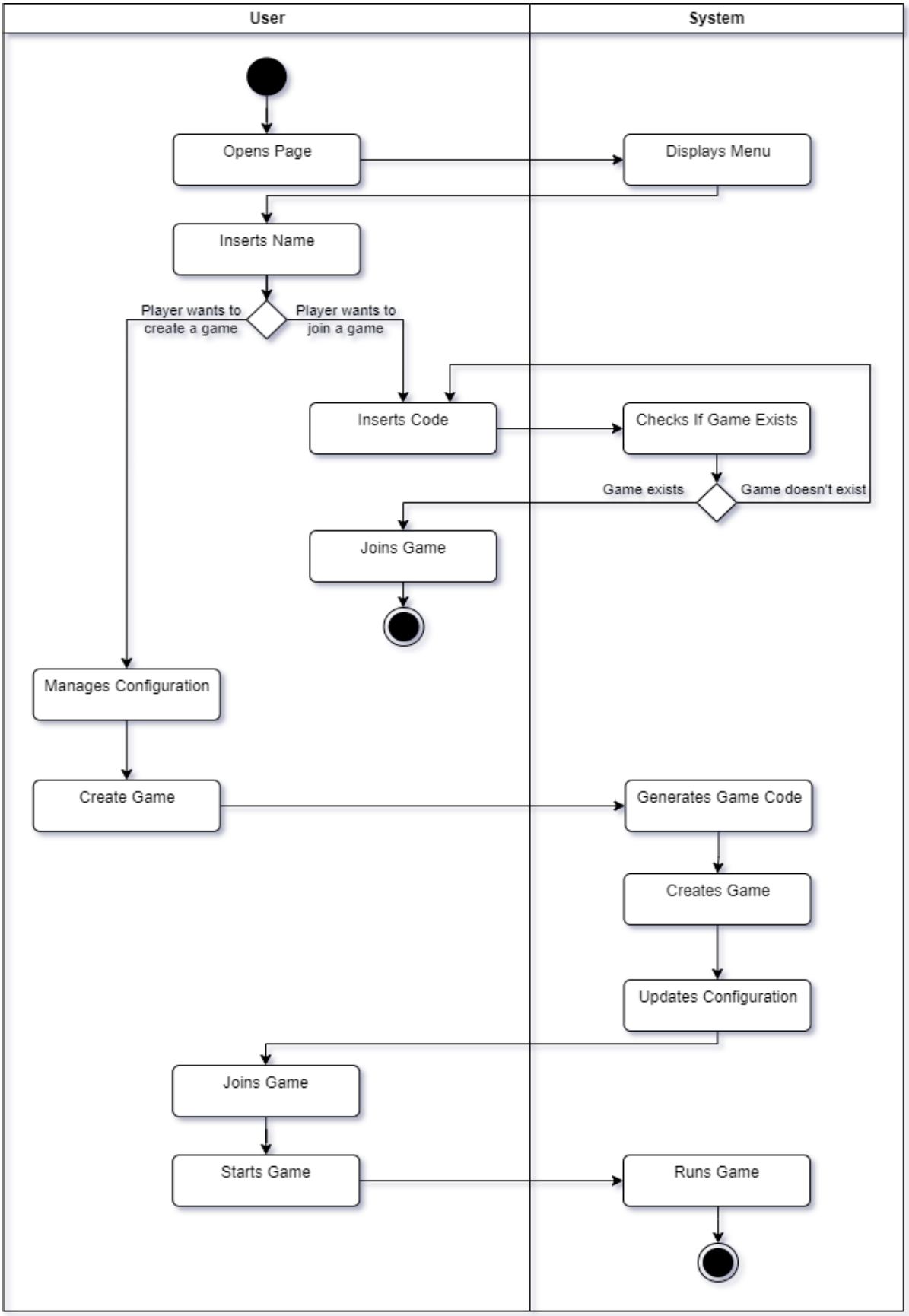
Cliccare sull’immagine per aprire il link

Diagramma a stati



Cliccare sull'immagine per aprire il link

Diagramma di attività



Cliccare sull'immagine per aprire il link

Analisi dei requisiti

Attori

- Giocatore

Requisiti funzionali

- **Configurare una partita**
  - Il giocatore, creatore della partita, può modificarne le impostazioni quali
    - numero di giocatori partecipanti
    - fondi dei giocatori
    - indice di casualità della partita
    - tasso di interesse
    - tasso di aumento/riduzione dei prezzi
    - modalità di fidelizzazione
  - Il giocatore, in alternativa, può anche scegliere tra più livelli di difficoltà dove ognuno rappresenta una combinazione delle impostazioni sopra definite

- **Creare una partita**
  - Un giocatore può comunicare al sistema l'intenzione di creare una partita
  - Il sistema crea una partita, con le impostazioni definite dall'utente, e la associa a un codice univoco che sarà utilizzato per accedervi
  - Il giocatore viene automaticamente aggiunto alla partita

- **Partecipare a una partita**
  - Un giocatore può unirsi alla lobby di gioco tramite l'utilizzo di un codice univoco fornito dal sistema alla creazione della partita
  - Se il limite dei giocatori non è raggiunto, il sistema reindirizza il giocatore all'indirizzo della partita
  - Il giocatore non può modificare le impostazioni di gioco scelte nella fase di creazione

- **Iniziare una partita**
  - Quando i giocatori sono più di uno, il giocatore che ha creato la partita può decidere di iniziarla
  - Il sistema inizia la partita

- **Uscire da una partita in corso**
  - Il giocatore può smettere di giocare e uscire dalla partita anche quando questa non è ancora terminata
  - Il sistema reindirizza il giocatore alla pagina iniziale
  - Se il giocatore alla sua uscita era l'ultimo rimasto, il sistema elimina la partita

- **Vendere proprietà**
  - Il giocatore può indicare la propria volontà di vendere le proprietà ai giocatori
  - Il giocatore interessato a vendere può decidere se vendere al miglior offerente oppure scegliere lui l'offerta che più gli aggrada.
  - Il giocatore non può selezionare proprietà con case o hotel già edificati
  - Il sistema notifica gli altri giocatori con una richiesta di scambio e reindirizza i giocatori alla pagina di transazione

- **Unirsi a un programma fedeltà**
  - Il giocatore può aderire a un programma di fedeltà di un altro giocatore per ricevere dei bonus quali
    - ► Raccolta punti: il giocatore riceve una quantità di punti in proporzione alla spesa effettuata. I punti possono essere utilizzati per ricevere uno sconto su una futura spesa. Una volta utilizzati, i punti dovranno essere riguadagnati.
    - ► Carta sconto: il giocatore riceve uno sconto per ogni spesa effettuata fino a una quantità massima consentita

- **Lanciare i dadi**
  - Il giocatore può decidere quando "lanciare" i dadi nel corso del suo turno
  - Se il lancio dei dadi porta a una coppia di valori uguali, i dadi vengono lanciati nuovamente
  - Ottenendo tre lanci con coppie uguali durante lo stesso turno, il giocatore finisce direttamente in prigione senza passare dal "Via"

- **Mettere all'asta una proprietà**
  - Quando un giocatore finisce su uno spazio con una proprietà che non è interessato ad acquistare, può decidere di metterla all'asta
  - Il sistema reindirizza tutti i giocatori alla pagina dell'asta
  - Il giocatore può partecipare all'asta da lui iniziata

- **Costruire edifici**
  - Il giocatore può acquistare, ad ogni momento del suo turno, un edificio (casa o hotel) a un prezzo prestabilito per aumentare il valore dell'affitto su una determinata proprietà
  - Per costruire un edificio, il giocatore deve essere in possesso di tutte le proprietà appartenenti allo stesso gruppo colorato
  - É possibile acquistare case solo se i terreni dello stesso colore hanno un numero di case che varia al più di uno
  - É possibile acquistare un hotel solo se i terreni appartenenti al gruppo dello stesso colore posseggono un hotel oppure quattro case ciascuno. In questa situazione, le case vengono sostituite da un hotel

- **Vendere edifici**
  - Il giocatore è libero di vendere gli edifici collocati sopra i terreni in suo possesso per riguadagnare una quantità di soldi pari alla metà del prezzo di un singolo edificio
  - É possibile vendere case solo se i terreni dello stesso colore hanno un numero di case che varia al più di uno
  - É possibile vendere hotel solo se i terreni dello stesso colore hanno quattro case oppure lo stesso numero di hotel. In questa situazione, l'hotel viene restituito alla Banca e sul terreno vengono posizionate quattro case

- **Ipotecare una proprietà**
  - Il giocatore può, in ogni momento del suo turno, ipotecare una proprietà per riguadagnare una quantità di soldi prestabilita
  - Una proprietà per essere ipotecata deve prima essere spogliata di tutti gli edifici collocati sopra il suo suolo
  - Per rimuovere l'ipoteca da una proprietà, il giocatore dovrà rifornire i soldi ricevuti dall'ipoteca più il 10% di interesse

Pattern e Design Principles applicati

MVC

Come linee guida architetturali del progetto abbiamo seguito quelle definite dal pattern MVC *(Model View Controller)*. È stato possibile ottenere disaccoppiamento tra lo strato di View e quello di Model, in modo che la view acceda al modello solo attraverso il livello di Controller. Nello specifico l'implementazione si discosta leggermente dal pattern, in quanto prevede che il livello di view acceda al controller per ottenere dei comandi da mostrare all'utente. L'utente tramite l'interazione comunica direttamente con il livello di model. La view comunque non ha alcun controllo e rimane disaccoppiata rispetto ai livelli inferiori, poiché è il livello di controller che gestisce i comandi da esporre.

SOLID

Durante lo sviluppo del progetto si è cercato di seguire i principi esposti in SOLID, cercando anche di mantenere un disaccoppiamento alto fra le componenti e alta coesione.
Nello specifico le classi sono state concepite in modo da avere responsabilità limitata in modo da non renderle monolitiche. Allo stesso modo ci si è concentrati sull'astrazione di molto concetti per permettere di applicare i pattern Liskov substitution e dependency inversion, così come seguire l'applicazione dell'interface segregation con la creazione di svariate interfacce specifiche.

Observer

Pattern applicato in gran parte a livello di view secondo gli standard MVC in modo da *iscrivere* le classi di interfacce utente all'aggiornamento degli oggetti a livello di model. Il pattern sfrutta interfacce custom observer/observable e oggetti consumer per *notificare* le view di aggiornamenti di dati rappresentati a livelli più bassi.

Chain Of Responsibility

Sfruttato nella creazione lato server di istanze degli *Space* appartenenti alla *Board*. Ogni *Space* ha come successore uno *Space* di altro tipo, in modo da creare l'istanza del tipo giusto in base al numero dello spazio che si vuole istanziare.

Factory

Utilizzato nel *CommandBuilderDispatcher* dove è possibile ottenere il builder della classe specificata per poter costruire istanze *Command.*

Builder

Pattern utilizzato nella creazione di istanze di *Command*, il pattern offre la possibilità di creare *Command* basati su specifici giocatori o proprietà alle richieste dalla view.

Command

L'implementazione di questo pattern permette la creazione di oggetti con operazioni specifiche su giocatori o proprietà in modo da essere esposte a livello di view, ma anche di essere utilizzate come componenti atomiche di altri comandi a backend.

Strategy

L'utilizzo di questo pattern avviene principalmente nelle classi *PriceManager* e *Space.* Queste ultime sono classi astratte e seguono approcci diversi in base alla categoria di proprietà (nel caso del *PriceManager*) o la categoria dello spazio stesso. Allo stesso modo anche i *Command* seguono questo approccio.

Scelte e assunzioni implementative

Data la libertà di implementazione del progetto, abbiamo sviluppato alcune scelte su determinate funzionalità.

- **Il ruolo dello "Entrepreneur"**: il limite dei partecipanti al gioco è stato aumentato permettendo ai giocatori della partita di modificarne il numero nella fase di creazione. Il creatore può decidere di impostare fino a dieci giocatori come numero massimo. La scelta di questi numeri è dovuta a un bilanciamento del gioco per cui non si vuole che certi utenti si ritrovino a giocare senza la possibilità di acquistare alcuna proprietà.
  Un utente può unirsi a una partita iniziata a meno che questa non conti già il numero massimo di giocatori ammessi.

- **Il programma di fidelizzazione**: Il giocatore che decide di partecipare al programma di fidelizzazione offerto da un altro giocatore riceverà degli sconti su tutte le proprietà di quest'ultimo.
  Abbiamo deciso di implementare due programmi selezionabili nella fase di creazione della partita: "Raccolta Punti" e "Carta Sconto"
  - La Raccolta Punti permette al giocatore iscritto di ricevere una quantità di punti basata sulle spese effettuate sulle proprietà dell'offerente e che potrà utilizzare successivamente (sempre sulle sue proprietà) per ammortizzare il prezzo degli affitti.
    Ogni giocatore può conservare fino a un massimo di 200 punti.
    Se il numero di punti è superiore alla spesa, i punti verranno sottratti per il valore di tale. Invece, se la spesa è superiore alla quantità di punti, questi verranno azzerati e il prezzo verrà ridotto per il loro numero.
  - La Carta Sconto permette al giocatore iscritto di avere uno sconto percentuale permanente su tutte le proprietà dell'offerente. Ogni volta che il giocatore effettuerà una spesa su un terreno dell'altro, la percentuale di sconto aumenterà di una porzione dell'esborso.
    Ogni giocatore può accumulare fino a uno sconto massimo del 25%.

- **La modalità di casualità**: Per rendere il gioco più dinamico abbiamo implementato un sistema di randomizzazione su molti dei suoi valori.
  In un determinato numero di turni, i prezzi, le tasse e gli interessi di una sequenza casuale di caselle vengono modificati tramite apposite formule.
  Per determinare quali caselle debbano essere variate, abbiamo introdotto un indice di randomizzazione configurabile nelle impostazioni iniziali della partita.

Possibili future modifiche

Durante lo sviluppo del progetto sono sorte alcune funzionalità per cui non abbiamo trovato tempo per essere implementate.
Alcune possibili modifiche da poter apportare sono:

- La volontà di **acquistare una proprietà** di un altro giocatore
  - Un giocatore che vuole acquistare può fare una proposta
  - Il sistema notifica il possessore della proprietà della proposta fatta

- La possibilità di **vendere una carta Probabilità/Imprevisti** “Esci Gratis di Prigione”
  - Un giocatore che vuole vendere una carta può fare una richiesta di vendita
  - Il sistema notifica tutti i giocatori della richiesta fatta

- La volontà di **acquistare una carta Probabilità/Imprevisti** “Esci Gratis di Prigione” di un altro giocatore
  - Un giocatore che vuole acquistare una carta può fare una proposta
  - Il sistema notifica il possessore della carta della proposta fatta

- L’implementazione di **nuovi programmi di fidelizzazione**
  - Carta Regalo: ogni determinato numero di pagamenti, il giocatore interessato ricevere uno sconto di una quantità prestabilita

- La **modifica dei programmi di fidelizzazione**
  - Un giocatore può aderire a un programma di fidelizzazione con un gruppo colorato di proprietà definito per ottenere sconti qualora dovesse pagare affitti/migliorare le caselle
  - Il giocatore può indicare il numero di punti da poter spendere nel programma di fidelizzazione “Raccolta Punti”

- La modifica del **numero di caselle all’interno del tabellone** dalle configurazioni della partita

- L’implementazione delle **animazioni legate al movimento della pedina** di un giocatore

- L’aggiunta di una **rappresentazione grafica delle carte** riguardanti le proprietà e le carte pescabili Probabilità e Imprevisti

- La **modifica dinamica dei valori riportati sul tabellone** di gioco in base alla randomizzazione

- Il **bilanciamento del gioco**, in modo da generare valori randomici che non sbilanciano troppo il corso della partita

- **Container Docker** per le singole partite, in quanto l'applicazione per come si presenta ora è poco scalabile. La possibilità di associare una partita a un container docker singolo aggiungerebbe scalabilità sostenendo alti valori di giocatori e partite

- Sistema di **heartbeat** in modo che il server percepisca l'inattività di un giocatore per poterlo rimuovere dalla partita. Inoltre tornerebbe anche utile alla chiusura della scheda o del browser con una partita in corso, in quanto l'ambiente web non offre un metodo efficace per notificare un evento di questo tipo

Guida per l'utente

Software required

- Java 11

- Maven

The program was tested and executed on Java 11 and executed on Maven 3.6.3 or higher versions.

How to build and run locally

- Run `mvn clean install` to let the script install dependencies needed
- To execute the application, run `mvn compile exec:java -Pproduction -Dexec.mainClass="it.monopoly.Application"`. The first time the command runs a `npm install` to install npm locally right into project directory (it might takes some minutes).
  - The application runs by default on port *8080.* To execute the application on a different port, run the above command with additional argument `-Dexec.args="--server.port=port_number"` where *`port_number`* is the port number.
- Go to a new window on your web browser and type `localhost:8080` on the URL search bar.
- To add more players, open a new window on your web browser, type `localhost:8080/unique_code` where the *`unique_code`* is the 16-characters alphanumeric code found on the lobby dialog or on the creator's webpage URL or simply click on *Participate Game* and insert the code on `localhost:8080`.

Glossario

- **Client-Server**: Sistema di architettura di rete nella quale genericamente un computer *client* o terminale si connette ad un *server* per la fruizione di un certo servizio

- **Codice univoco partita**: Codice alfanumerico utilizzato per distinguere le partite tra loro. Viene utilizzato nella path dell'URL della pagina Web per reindirizzare i giocatori alla pagina corretta

- **Entrepreneur**: Qualsiasi giocatore che decide di partecipare al gioco

- **Lobby di Gioco**: Sala di attesa in cui i giocatori esprimono la volontà di voler partecipare al gioco. Una lobby viene istanziata quando un giocatore intende creare una partita

- **Pagina Web**: Documento digitale tramite il quale sono rese disponibili all'utente le informazioni del *World Wide Web* attraverso un web browser

- **Programmi di fedeltà**: Sistema di ricompensa del giocatore basato sulle sue spese

- **Sistema**: Servizio offerto

- **Tasso di aumento/riduzione dei prezzi**: Tasso di variazione dei prezzi di una determinata casella di gioco

- **Tasso di interesse**: Valore percentuale che viene applicato a un prestito

- **Utente**: Chiunque utilizzi il sistema