An Adapted Monopoly Game

progetto-monopoly-1-gruppo-monopoly-1

Appello gennaio 2021

Ernesto Gallucci 830689
Giulio Riggio 844901
Alessandro Rosa 801460
Raffaele Cerizza 845512

Indice

1 Introduzione

Il testo della traccia prevede che: The goal of this project is to develop a turn-based, client-server NewMonopoly game with high configurability, in which users play the game through a web site. The game is web-based.
Dovendo imparare da zero come poter fare un progetto web-based e client-server abbiamo deciso di utilizzare Node.js per tre motivi:

1. Node.js è veloce ed è orientato al web per piccole applicazioni.
2. Un membro del gruppo lo utilizzava da qualche mese.
3. Per poter imparare una tecnologia sempre più importante in ambiente web.

Node.js è una libreria e un ambiente runtime multipiattaforma. Esso è utilizzato per la creazione di applicazioni JavaScript lato server. Inoltre è un software gratuito e open source. Node.js utilizza un modello basato su eventi per affrontare la scalabilità e consentire l'utilizzo di librerie JavaScript che aiutano a semplificare la codifica. Alcune differenze tra Java e Node.js:

- Java è compilato e convertito in bytecode.
- Node.js interpreta ed esegue JavaScript code.

Insieme a Node.js abbiamo utilizzato il framework web React. Esso è una libreria JavaScript per la creazione di interfacce utente. Pertanto può essere utilizzato come base nello sviluppo di applicazioni a pagina singola o mobile. React è tra i framework web più sofisticati, decide all'ultimo secondo se eseguire il codice sul server o sul client, una logica intelligente prenderà la decisione "al volo" in base al carico o alla RAM. React è comunemente object-oriented (everything derives from React.Component), ma è raro avere più di un livello di subclass. Il codice di React è costituito da entità denominate componenti. Queste componenti possono essere classi o funzioni. Pertanto nel seguito il termine "componente" verrà utilizzato anche per indicare una "classe".
In React è necessario adottare strategie diverse per ottenere high cohesion e low coupling perché è necessario partire dal vincolo di React di avere delle componenti incluse in altre componenti e dal dover usare JavaScript e HTML.
Durante l'implementazione abbiamo riscontrato alcune difficoltà dovute, in gran parte, alle differenze con Java:

- E' necessario separare la UI in una gerarchia di componenti, in cui esistono componenti che si comportano da contenitori per altri componenti.
- I componenti devono essere aggiornati per poter vedere i cambiamenti settando lo stato.
- In React, la condivisione dello stato tra componenti si ottiene spostando lo stato verso il più vicino antenato comune dei componenti che ne hanno bisogno.

- La gerarchia di componenti è dall'alto verso il basso.
- E' stato necessario capire come passare i parametri alla classi e funzioni e componenti tramite props.

Per rendere il nostro monopoly più bello e più vicino agli standard di internet abbiamo arricchito il progetto utilizzando il Material-UI di Google (https://material-ui.com/). Esso è una libreria di componenti semplice e personalizzabile che permette di creare applicazioni React più veloci, più belle e più accessibili.
Per la realizzazione del nostro monopoly abbiamo assunto che tutti i giocatori che partecipano ad una partita accedano al gioco dallo stesso luogo utilizzando un solo computer. L'idea è quella di soddisfare le esigenze di un gruppo di amici, che vogliono giocare a monopoly, ma che non hanno il gioco da tavolo. Per superare questo problema possono accedere al nostro sito e giocare attraverso il computer come se fosse il gioco vero e proprio. Inoltre abbiamo cercato, per quanto i tempi ristretti c'è l'hanno permesso, di superare tutti i limiti del monopoly standard che vengono espressi nella traccia del progetto.
Alla fine possiamo dire di aver imparato una nuova tecnologia diversa da Java.

Il branch da considerare per la consegna è il **branch master** del repository Git che ci è stato assegnato.

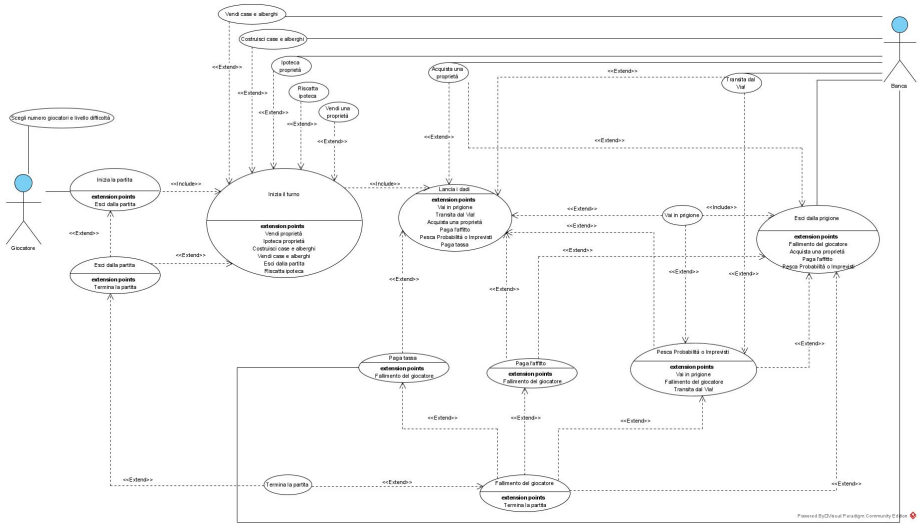
2 Requisiti

Questi sono i requisiti che sono emersi dalla traccia del progetto relativo al monopoly:

- L'applicazione deve possedere tutte le funzionalità chiave di un monopoly standard.
- Gli utenti devono poter giocare tramite un sito web.
- L'applicazione deve essere client-server.
- L'applicazione deve essere altamente configurabile.
- L'applicazione deve consentire ai giocatori di guadagnare dei punti premio che possono essere usati in alternativa o in combinazione ai soldi per svolgere pagamenti nel gioco (affitti, tasse ecc....).
- L'applicazione dovrà possedere un sistema, basato sulla lealtà, che permetta l'uso di valute virtuali seguendo le regole economiche delle alleanze internazionali che esistono attualmente.
- L'applicazione dovrà consentire ai giocatori di scegliere tra tre diversi livelli di difficoltà (facile, medio e difficile). Ogni livello si distinguerà dagli altri attraverso regole e configurazioni diverse.

3 Analisi

3.1 Diagramma dei casi d'uso



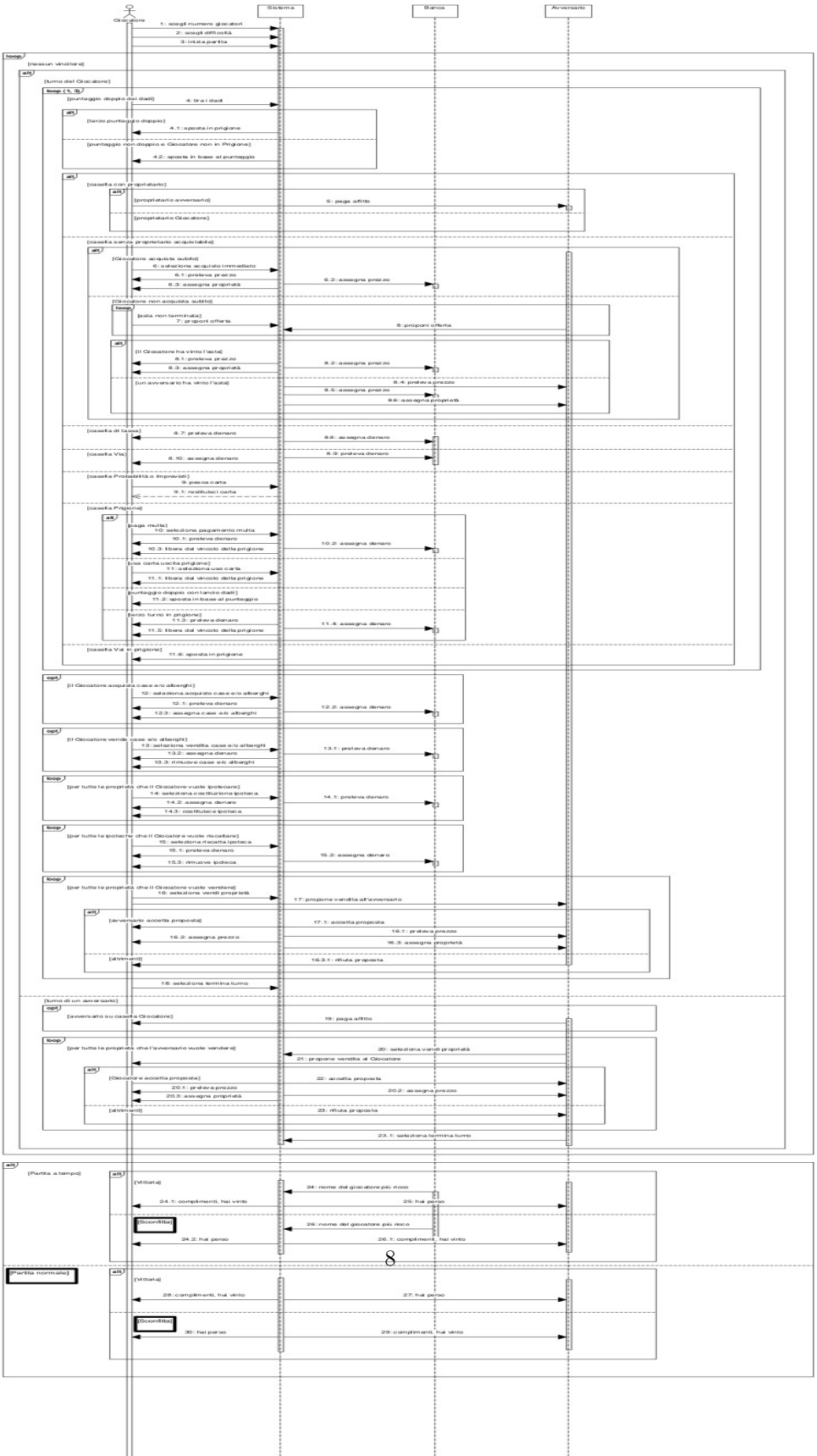
3.2 Descrizione dei casi d'uso

I casi d'uso indicati nel diagramma dei casi d'uso riportato al paragrafo precedente sono stati descritti in forma dettagliata. Tuttavia per evitare di rendere la presente documentazione eccessivamente lunga si è deciso di riportare qui di seguito solo i nomi dei casi d'uso identificati. In ogni caso le descrizioni dettagliate di tutti questi casi d'uso si trovano nella cartella Descrizione-Casi-dUso. In particolare sono stati identificati i seguenti casi d'uso:

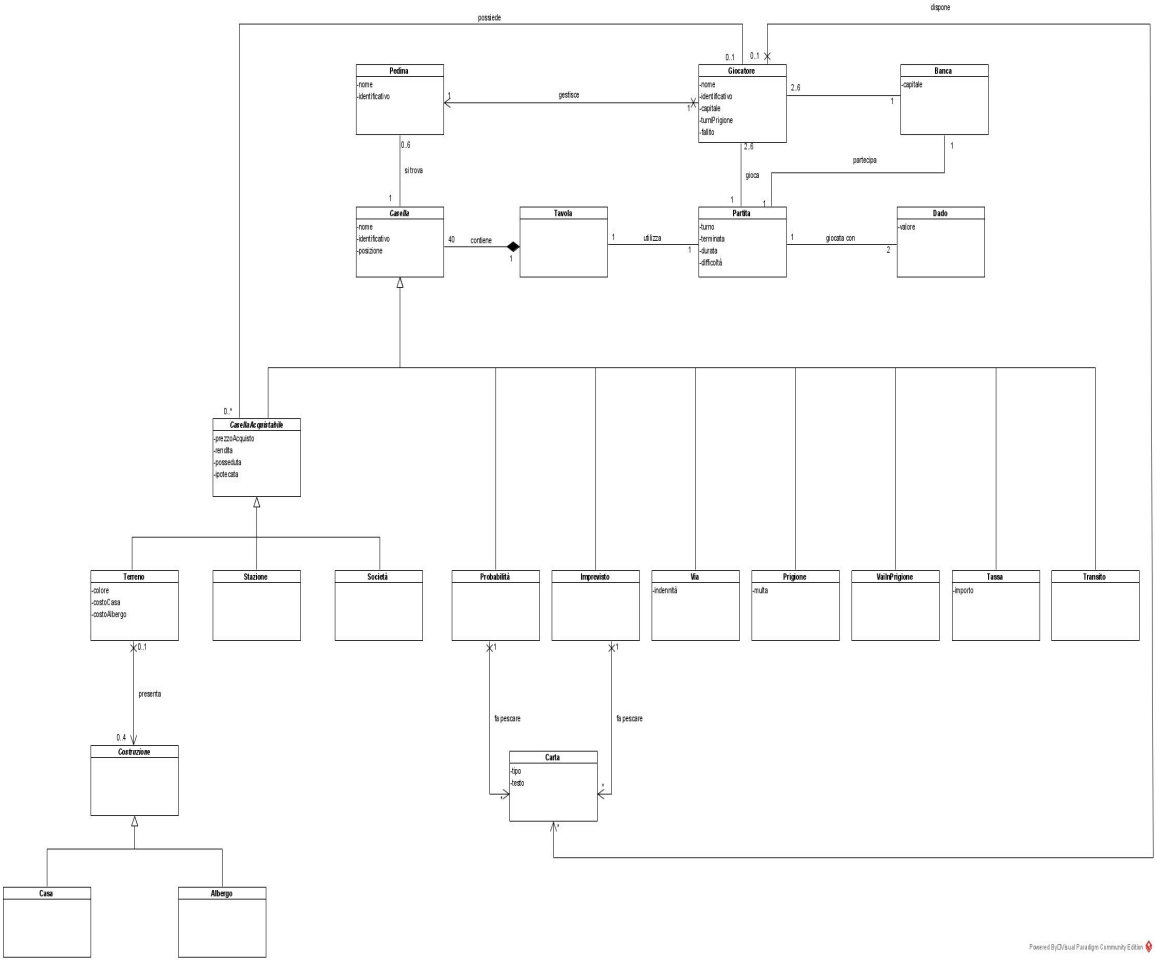
1. UC1 - Scegli numero giocatori e livello difficoltà
2. UC2 - Inizia la partita
3. UC3 – Paga tassa
4. UC4 - Paga l'affitto
5. UC5 - Pesca probabilità o imprevisti
6. UC6 – Ipoteca proprietà
7. UC7 - Riscatta ipoteca
8. UC8 - Esci dalla prigione
9. UC9 – Transita dal Via
10. UC10 - Acquista una proprietà

11. UC11 - Lancia dadi
12. UC12 – Termina la partita
13. UC13 - Inizia il turno
14. UC14 - Esci dalla prigione
15. UC15 – Vai in prigione
16. UC16 - Costruisci case e alberghi
17. UC17 - Vendi case e alberghi
18. UC18 – Fallimento del giocatore
19. UC19 - Vendi una proprietà

3.3 Diagramma di sequenza di sistema

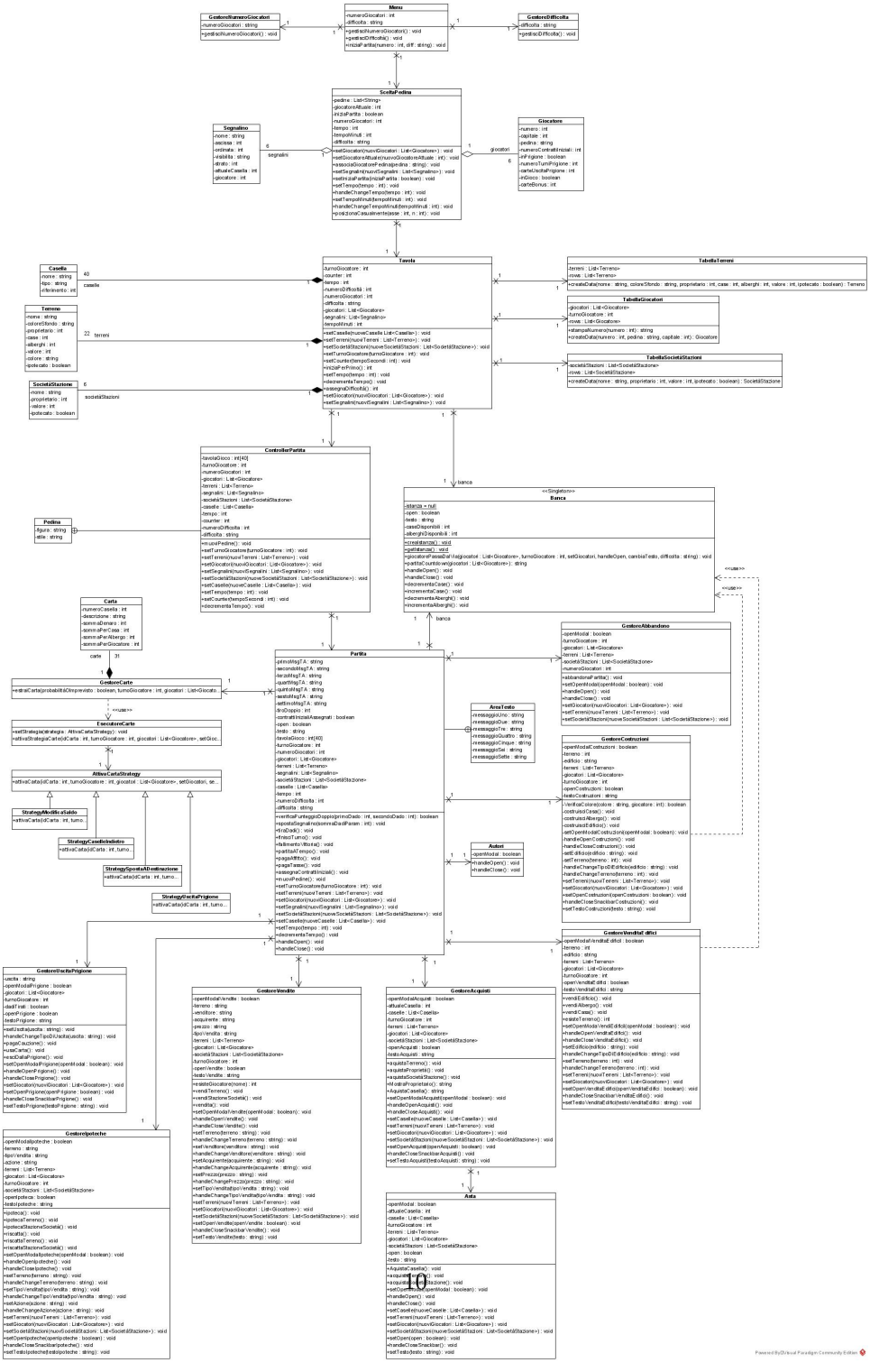


3.4 Modello di dominio



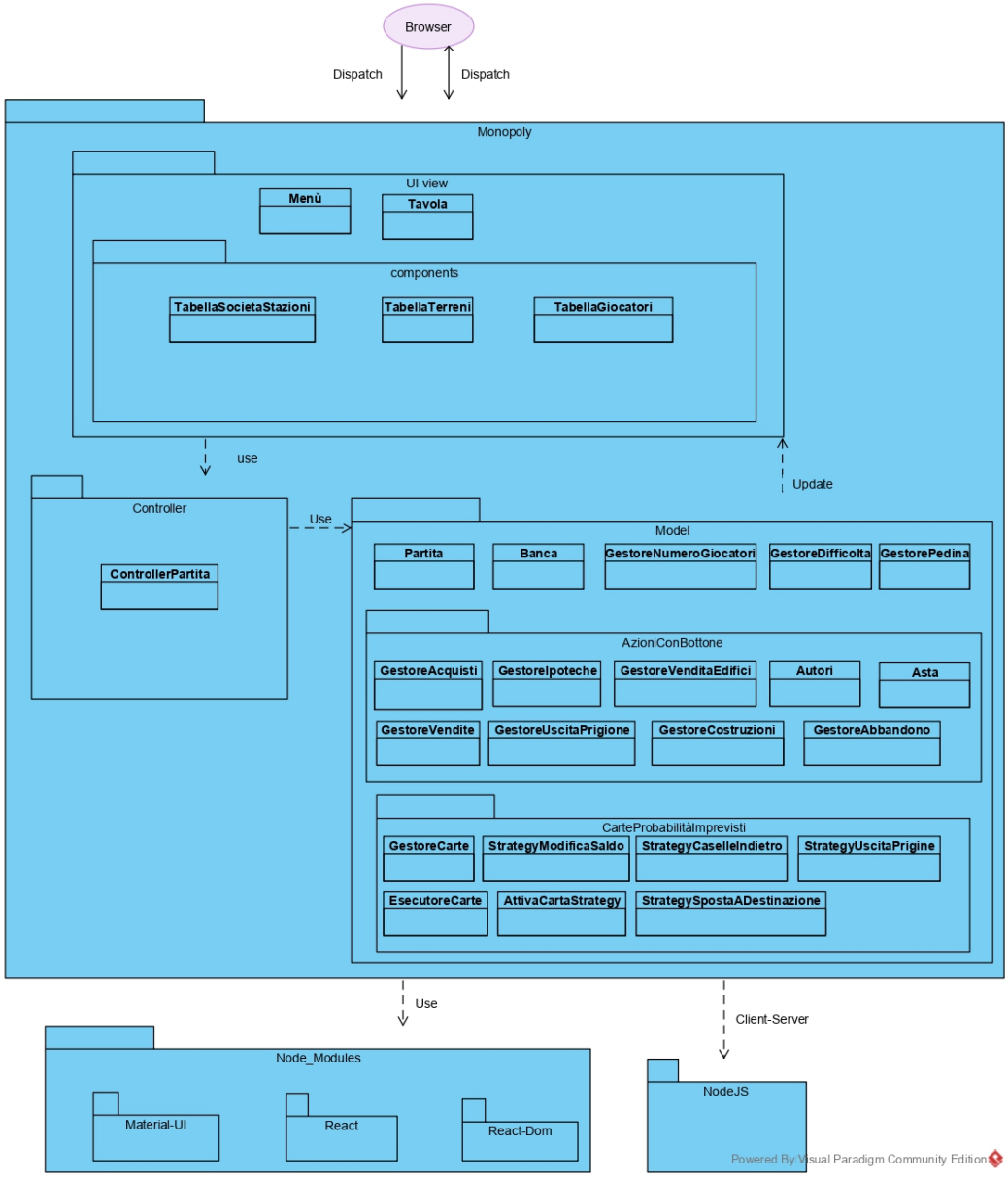
4 Progettazione

4.1 Diagramma delle classi a livello di progetto



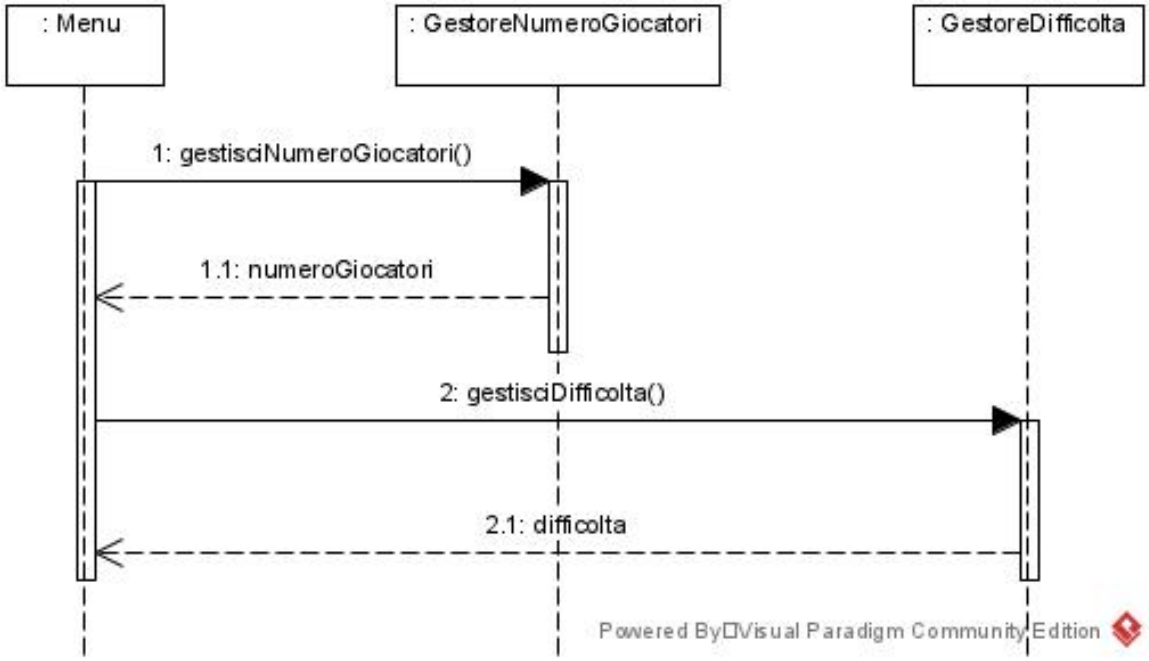
In questo paragrafo vengono illustrate brevemente alcune particolarità significative del diagramma delle classi a livello di progetto.
Anzitutto in questo diagramma sono stati riportati i tipi degli attributi nonostante l'implementazione sia avvenuta utilizzando un linguaggio debolmente tipizzato come JavaScript. In particolare sono stati specificati i tipi degli attributi in quanto nell'implementazione del codice gli attributi sono stati gestiti nella maggior parte dei casi come se avessero un tipo definito. In questo modo si sono ottenuti tra l'altro una maggiore comprensibilità del codice e una maggiore facilità nel debug.
Inoltre si noti che in alcuni casi i metodi ricevono come parametri delle funzioni. Anche questo è coerente con l'utilizzo di JavaScript.
Ancora, la classe Banca è stata realizzata utilizzando il design pattern Singleton. Per evidenziare questa particolarità si è utilizzato lo stereotipo «Singleton» accanto al nome della classe. Questa scelta è stata motivata dal fatto che il programma Visual Paradigm utilizzato per la realizzazione dei diagrammi non supporta la convenzione UML che prevede l'aggiunta del numero 1 nell'apice destro della classe per denotarne la natura di classe Singleton.
Infine si possono notare alcune associazioni di dipendenza connotate dallo stereotipo «use». Queste particolari associazioni sono state utilizzate quando la classe sorgente ha necessità di utilizzare un metodo della classe di destinazione per completare una propria operazione. Questo è coerente con la notazione UML 2.0 e con la relativa letteratura [1, p. 48].

4.2 Diagramma architettura software

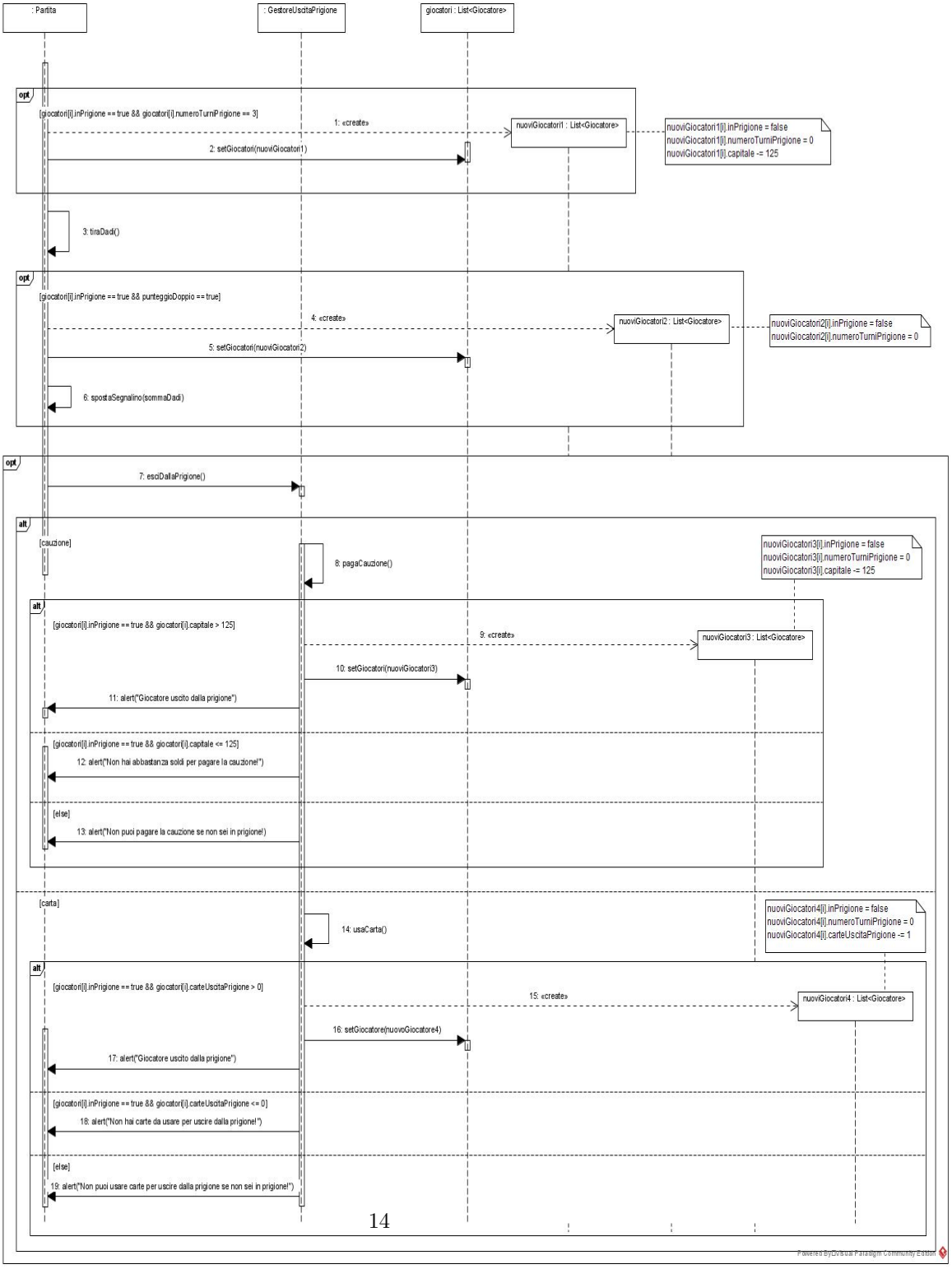


4.3 Diagrammi di sequenza

4.3.1 Diagramma di sequenza per la scelta del numero dei giocatori e della difficoltà

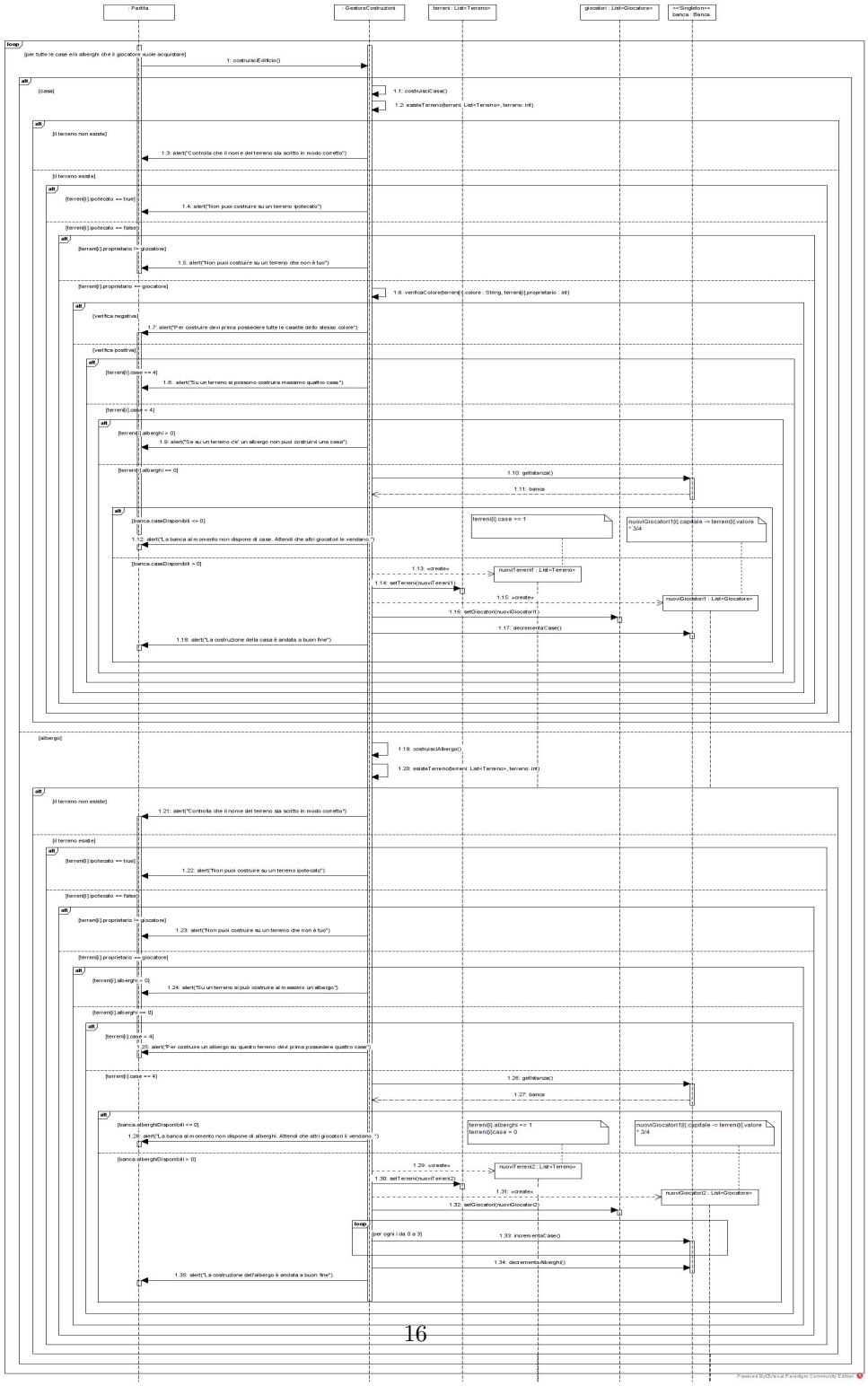


4.3.2 Diagramma di sequenza per l'uscita dalla prigione

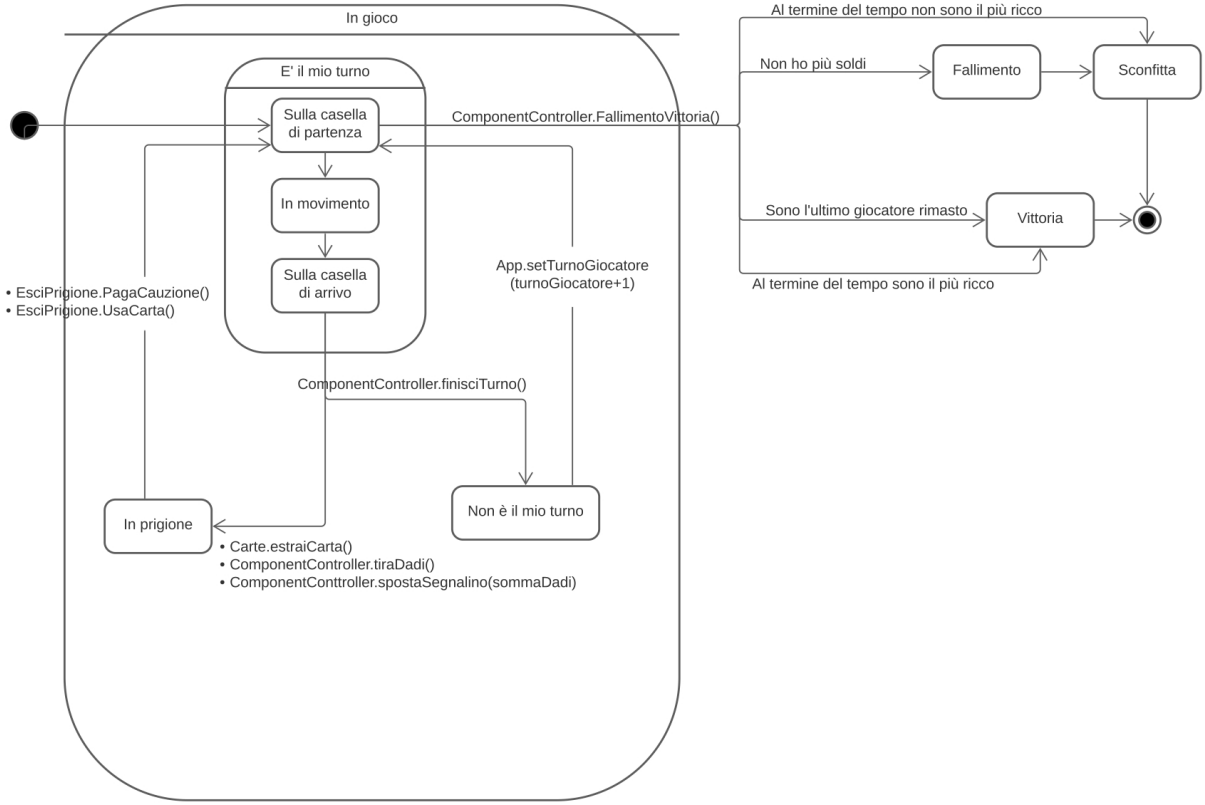


Nel diagramma di sequenza che rappresenta le interazioni che portano un giocatore a uscire di prigione sono state utilizzate alcune note. In queste note vengono indicati i valori degli attributi di alcuni oggetti a seguito del compimento di determinate operazioni. Questi valori sono determinati tramite opportuni assegnamenti. Tuttavia i diagrammi di interazione non prevedono alcuna notazione per gli assegnamenti dato che questi non rappresentano interazioni tra oggetti[3, p. 266]. Pertanto per maggiore chiarezza si è deciso di rappresentare i risultati di questi assegnamenti tramite opportune note.

4.3.3 Diagramma di sequenza per l'acquisto di case o alberghi

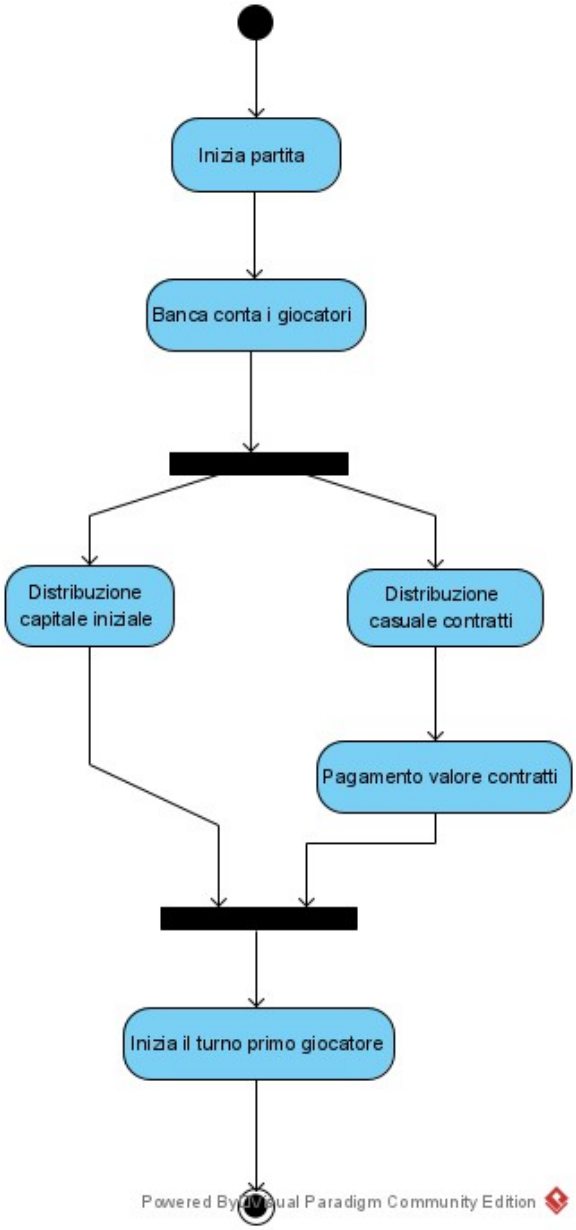


4.4 Diagramma degli stati

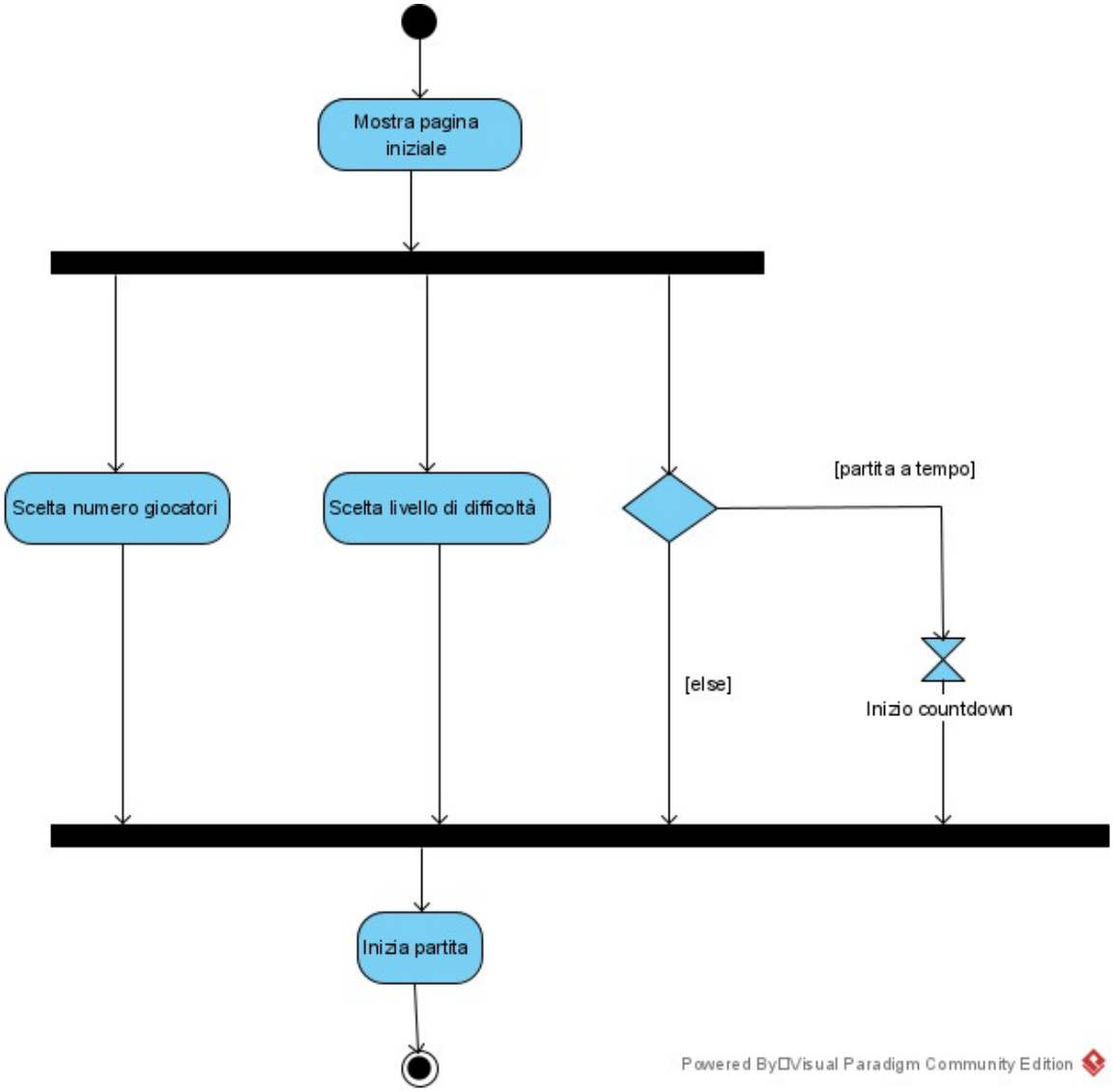


4.5 Diagrammi delle attività

4.5.1 Diagramma delle attività all'inizio della partita



4.5.2 Diagramma delle attività per le scelte iniziali



4.6 Design patterns e architectural patterns

Durante la progettazione dell'applicazione si è deciso di implementare diversi design patterns e architectural pattern. Vengono ora esposte le ragioni che hanno portato a utilizzare questi pattern e il modo in cui sono stati implementati.

4.6.1 Singleton

Il design pattern Singleton è stato utilizzato per implementare la gestione delle case e degli alberghi da parte della banca. Infatti ogni volta che un giocatore acquista o vende una casa o un albergo, la banca deve aggiornare il numero di case e alberghi disponibili. Pertanto è necessario che il numero di case e alberghi disponibili sia unico nel gioco. E questa unicità è garantita proprio dall'utilizzo del design pattern Singleton.
In particolare il design pattern Singleton è stato applicato alla componente Banca.js. Questa componente presenta gli attributi `caseDisponibili` e `alberghiDisponibili` che rappresentano rispettivamente il numero di case e il numero di alberghi che i giocatori possono ancora acquistare dalla banca. Inoltre Banca.js presenta un attributo statico `instance`, che verrà utilizzato dalla componente per restituire una propria istanza e che viene inizializzato con il valore `null`. Infine la componente Banca.js presenta i metodi tipici di una classe Singleton:

- Un metodo statico `creaIstanza()` che restituisce una istanza di Banca.js;
- Un metodo statico `getIstanza()` che restituisce l'istanza già creata di Banca.js oppure una nuova istanza se il valore di istance è ancora null.

Attraverso il design pattern Singleton è stato possibile offrire un accesso globale alla componente Banca.js. Questo consente di aggiornare il valore delle case e degli alberghi disponibili in modo univoco nel sistema. Infatti il numero di case e di alberghi disponibili tiene sempre conto degli aggiornamenti derivanti dalle operazioni dei vari giocatori; e non viene creata una "nuova" istanza di Banca.js ogni volta che un giocatore acquista o vende una casa o un albergo.
Infine si precisa che l'aggiornamento delle variabili relative alle case e agli alberghi disponibili avviene attraverso appositi metodi pubblici esposti dalla componente Banca.js.

4.6.2 Facade

Il design pattern Facade è stato utilizzato al fine di avere unico punto di accesso al sottosistema che si occupa della gestione di una partita di monopoly. In particolare questo design pattern si è tradotto nell'introduzione della componente ControllerPartita.js che fornisce un accesso unitario alle varie azioni che possono essere compiute durante una partita e che possono essere invocate dall'utente tramite una serie di bottoni presenti al centro della tavola da gioco. In questo modo si è tra l'altro garantita la separazione tra View e Model. Questo è coerente con l'applicazione del pattern architetturale Model-View-Controller. Ed è

in ogni caso coerente anche con il design pattern Facade che permette quindi di sviluppare il sottosistema Model in modo indipendente dalla View. In questo quadro è stato realizzato un facade controller [3, pp. 355-356].

4.6.3 Strategy

Il design pattern Strategy è stato utilizzato per implementare la gestione delle carte relative alle probabilità e agli imprevisti. Si è scelto di utilizzare questo design pattern per tre principali motivi. Il primo motivo è che questo design pattern permette di isolare i dettagli di implementazione della gestione delle carte dalle componenti che ne fanno uso: in questo modo risulta più facile modificare e mantenere il codice sia del contesto che delle strategie. Il secondo motivo è che questo design pattern rende possibile introdurre nuove strategie di gestione delle carte senza dover modificare il contesto in cui vengono utilizzate. Il terzo motivo è che questo design pattern permette di mantenere un'alta coesione e una bassa dipendenza per le componenti che hanno bisogno di interagire con le carte relative alle probabilità e agli imprevisti in quanto il contesto mantiene un unico riferimento alla superclasse delle strategie concrete.
In particolare per la realizzazione di questo design pattern: il ruolo di Client è assunto dalla componente GestoreCarte.js; il ruolo di Context è assunto dalla componente EsecutoreCarte.js; il ruolo di Strategy è assunto dalla componente AttivaCartaStrategy.js; i ruoli di ConcreteStrategy sono assunti dalle componenti StrategyModificaSaldo.js, StrategyCaselleIndietro.js, StrategySpostaADestinazione.js, StrategyUscitaPrigione.js. A questo proposito si precisa che il Client interagisce esclusivamente con il Context creando e passando a quest'ultimo una specifica ConcreteStrategy. E questo è coerente con la descrizione originale del design pattern Strategy [2, p. 317].

4.6.4 Model-View-Controller

Il pattern architetturale Model-View-Controller è stato utilizzato per separare le responsabilità dell'applicazione. In particolare:

- La View comprende le componenti che si occupano dell'interazione con l'utente e della visualizzazione delle schermate del gioco.
- Il Controller comprende la componente ControllerPartita.js che si occupa di fornire un unico punto di accesso al Model. Inoltre ControllerPartita.js si occupa di trasmettere al Model le informazioni provenienti dalla View in modo che possano essere gestite dalle opportune componenti del dominio applicativo.
- Il Model comprende le componenti che si occupano della gestione dei dati provenienti dalla View. Il Model costituisce il cuore dell'applicazione e garantisce il buon funzionamento di tutte le principali azioni di gioco.

4.7 Design principles

Per l'implementazione del progetto si è deciso di seguire il Single Responsibility Principle. Infatti le funzionalità principali del monopoly sono state incapsulate in classi separate. Questo risulta evidente dal fatto che la gestione di ogni bottone che compare al centro della tavola del monopoly risulta incapsulata in una specifica classe il cui nome presenta il prefisso "Gestore". In questo modo è stato possibile accrescere la coesione delle singole classi, nonché facilitare le operazioni di debugging e la mantenibilità.

5 Implementazione

5.1 Come è organizzata la cartella del progetto

La cartella del nostro progetto è organizzata in due parti principali: la prima è costituita dalla cartella Documenti Monopoly, la seconda è costituita dal codice del monopoly e da tutto ciò che serve al suo funzionamento. All'interno della cartella Documenti Monopoly ci sono alte quattro cartelle:

- Analisi. Questa cartella contiene tutto il materiale che abbiamo realizzato durante la fase di analisi ovvero:
  - Le descrizioni dei casi d'uso nella cartella Descrizione-Casi-dUso
  - Il diagramma dei casi d'uso
  - Il diagramma di sequenza di sistema
  - Il modello di dominio
- Progettazione. Questa cartella contiene tutto il materiale prodotto durante la fase di progettazione ovvero:
  - I diagrammi delle attività nella cartella Diagrammi delle attività
  - I diagrammi di sequenza nella cartella Diagrammi di sequenza
  - Il diagramma dell'architettura software
  - Il diagramma degli stati
  - Il diagramma delle classi a livello di progetto
- Diagrammi di Gantt. Questa cartella contiene i diagrammi di Gantt iniziale e finale.
- SonarQube e Understand. Questa cartella contiene i risultati dei test che abbiamo svolto con questi software.
- Documentazione progetto-monopoly-1-gruppo-monopoly-1. E' il file in formato PDF di questa documentazione.

5.2 Cosa abbiamo implementato

Il nostro monopoly è costituito da tre pagine. Nella prima gli utenti potranno scegliere il livello di difficoltà tra: facile, normale e difficile. Inoltre potranno selezionare il numero dei giocatori che parteciperanno, da un minimo di due a un massimo di sei. In base al numero di giocatori selezionato cambieranno sia la quantità di denaro sia il numero di contratti iniziali assegnati a ognuno. Inoltre in base al livello di difficoltà scelto ci saranno alcune differenze.

- Se il livello è facile i giocatori svolgeranno una classica partita a monopoly con una lieve aggiunta: la possibilità di guadagnare dopo ogni giro della tavola una carta bonus che permette di evitare il pagamento di affitti e tasse.
- Se il livello è medio ogni giocatore verrà posizionato casualmente sulla scacchiera. Pertanto i giocatori non partiranno dal VIA come nel livello facile. Inoltre gli affitti e le tasse saranno aumentati.
- Il livello difficile presenta le stesse caratteristiche del livello medio ma con affitti e tasse ancora più onerosi.

Dopo aver selezionato queste informazioni premendo il bottone "Inizia la partita" si passerà alla pagina successiva.
Nella seconda pagina sarà possibile scegliere se svolgere una partita a tempo e ogni giocatore dovrà scegliere la pedina che lo identificherà durante il gioco. Se i giocatori preferiscono non svolgere una partita a tempo basterà non scrivere niente nelle apposite caselle di testo. Se, invece, vogliono svolgere una partita a tempo possono scegliere tra due opzioni: svolgerla con un numero di turni prestabilito al termine del quale vincerà il giocatore più ricco, oppure impostare un countdown che conterà i secondi che mancano alla fine della partita e quando arriverà a zero stabilirà il vincitore. Come abbiamo già detto, in questa pagina ogni giocatore potrà scegliere la sua pedina. Perciò partendo dal primo giocatore ognuno dovrà cliccare sulla pedina che desidera scegliere fino ad arrivare all'ultimo giocatore. Naturalmente non è ammesso che due giocatori scelgano la stessa pedina. Quando tutti i giocatori avranno scelto la loro pedina si passerà alla terza pagina.
La terza pagina è quella in cui si può giocare. Al centro è posizionata la tavola del monopoly con le pedine che si muovono in base al risultato del lancio dei dadi. A destra della tavola è visibile una tabella che mostra i giocatori con il numero che li identifica, la loro pedina e il loro capitale. I giocatori che hanno perso, abbandonato il gioco o che non hanno mai partecipato alla partita saranno evidenziati di rosso. Invece il giocatore che stà svolgendo il turno in quel momento sarà evidenziato di verde chiaro. Il giocatore che svolgerà il turno per primo verrà scelto casualmente. A sinistra della tavola da gioco è presente una tabella che mostra le società e le stazioni con il loro nome, il proprietario, il valore e se sono ipotecate. Sotto la tavola è presente la tabella che contiene tutti i terreni. Anche in questo caso saranno visibili il nome, il proprietario, il colore, il numero di case, il numero di alberghi, il valore e se è ipotecato. Quando la

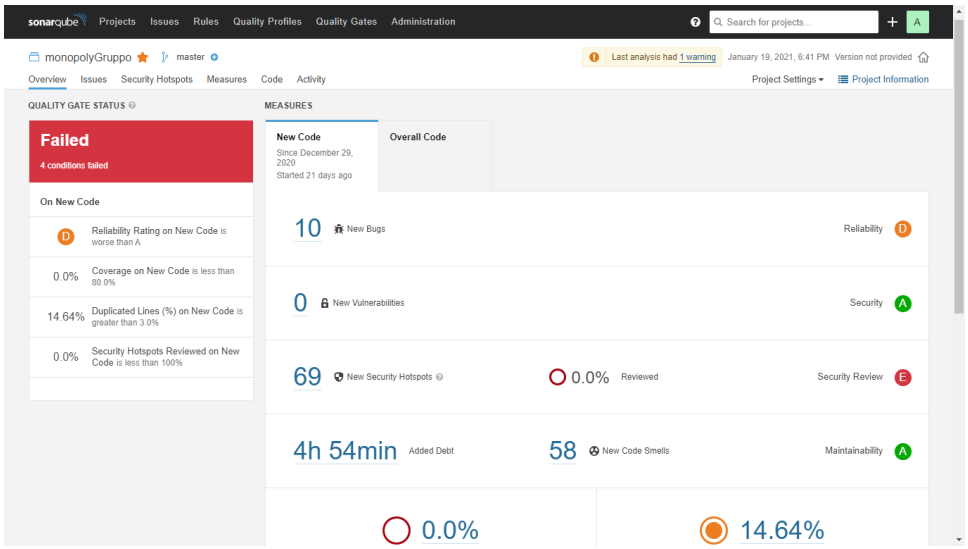
partita inizia vengono assegnati ad ogni giocatore una serie di contratti ovvero di terreni, società e stazioni.
Nello spazio vuoto al centro della tavola sono presenti una serie di bottoni che permettono all'utente di svolgere varie azioni. I tasti sono:

- Tira dadi. Permette di tirare i dadi e il risultato verrà mostrato nella finestra sottostante.
- Costruisci. Permette di aprire una finestra modale in cui inserire le informazioni per poter costruire un edificio su un terreno scelto dal giocatore.
- Vendi. Anche in questo caso si aprirà una finestra modale che permette di vendere ad altri giocatori terreni, società o stazioni.
- Finisci turno. Consente al giocatore che stà svolgendo il turno di concluderlo e fa passare il comando al giocatore successivo.
- Abbandona la partita. Permette ad un giocatore di lasciare la partita prima che sia conclusa.
- Autori. Mostra una finestra modale con i nomi degli autori di questo monopoly.
- Ipoteca. Permette di ipotecare una proprietà o di riscattare una proprietà ipotecata.
- Acquista. Permette al giocatore che stà svolgendo il turno di acquistare la proprietà su cui si trova, se non ha già un proprietario. Se il giocatore non la volesse comprare può far partire un asta che coinvolge tutti i giocatori. Quello che offre di più diventerà il proprietario della proprietà all'asta.
- Esci dalla prigione. Permette di pagare o usare una carta per uscire di prigione.
- Vendi edificio. Permette di vendere un edificio alla banca.

6 Risultati dei test

6.1 Con SonarQube

In questo paragrafo verranno illustrati i risultati dell'analisi della qualità del software ottenuti tramite il software SonarQube. In particolare per evidenziare i progressi ottenuti nella qualità dell'implementazione grazie all'impiego di questo software verranno mostrati i risultati dell'analisi di SonarQube sia su una versione avanzata (ma non definitiva) del progetto sia sulla versione definitiva. Cominciamo con i risultati dell'analisi svolta con SonarQube il 19.1.2021.

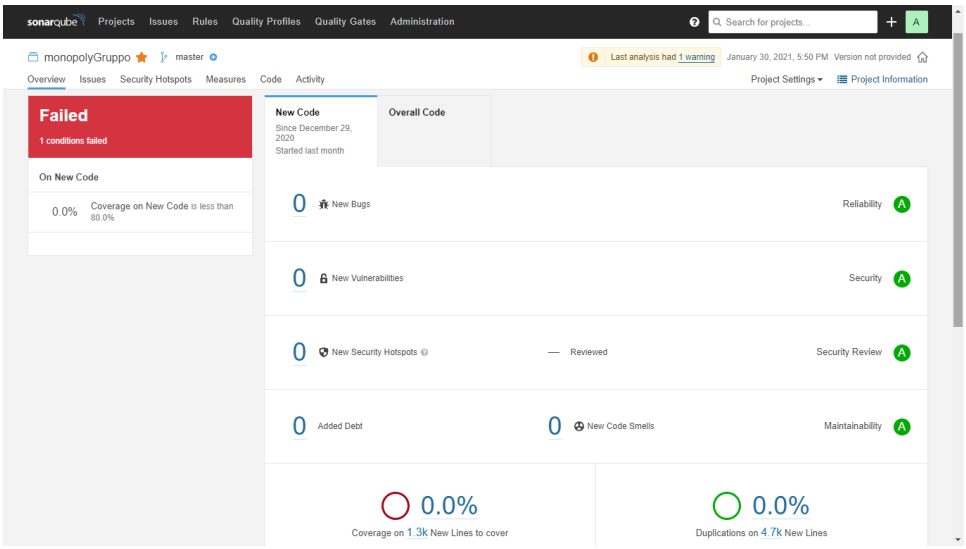


Questa analisi mostra: la presenza di un piccolo numero di bug; un elevato numero di security hotspots; circa 5 ore di debito tecnico dovute a 58 code smells; una percentuale di duplicazione del codice del 14%. In particolare:

- La maggior parte dei security hotspots hanno riguardato la presenza di alert e l'utilizzo della funzione `Math.random` per il calcolo del risultato del lancio dei dadi. Per rimuovere questi security hotspots si è proceduto come segue.
  In primo luogo si è provveduto a sostituire gli alert con alcuni Snackbars. In particolare uno Snackbar è una componente del Material-ui di React che permette di visualizzare brevi messaggi durante l'esecuzione dell'applicazione. Questi Snackbar sono particolarmente sicuri. Pertanto il loro utilizzo ha permesso di rimuovere i security hotspots relativi agli alert.
  In secondo luogo si è provveduto a sostituire l'impiego diretto della funzione `Math.random` tramite la creazione e l'impiego di una specifica funzione denominata CryptoRandom. Questa funzione è costruita in modo tale da generare numeri casuali senza incorrere in problemi di sicurezza. Pertanto anche per questo motivo tutti i security hotspots evidenziati da SonarQube sono stati rimossi dal progetto.

- Per rimuovere il codice duplicato si è proceduto a una complessa operazione di refactoring. In particolare si è fatto largo uso della tecnica di refactoring Extract Method, trasformando porzioni di codice utilizzate in più occasioni in singoli metodi che vengono chiamati opportunamente.

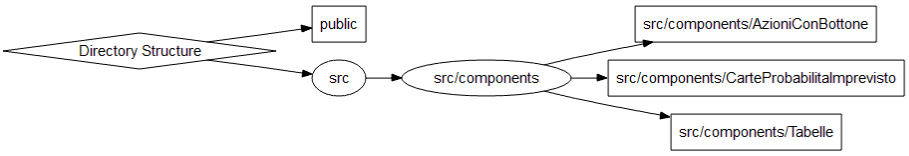
Di seguito è riportato il risultato dell'ultima analisi che abbiamo svolto:



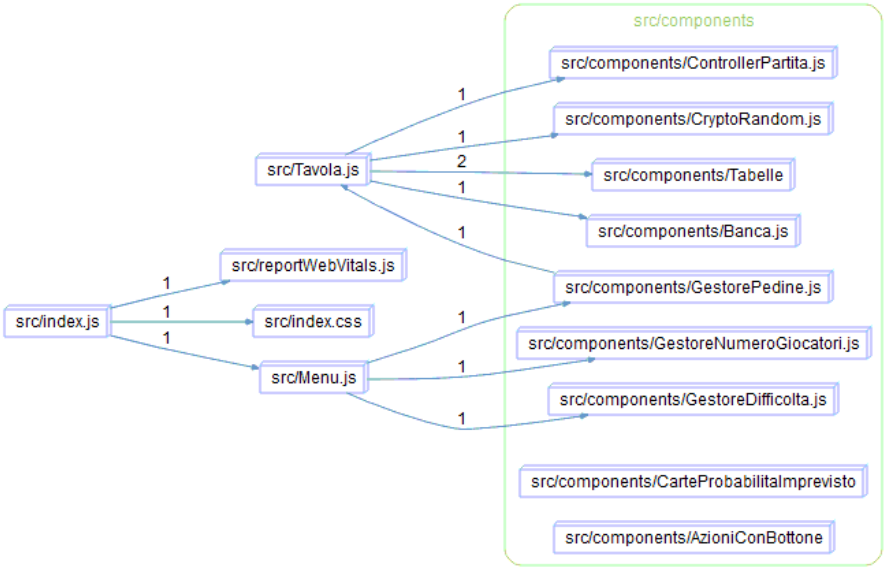
Di seguito riportiamo la project key e il provide token di questa analisi:
Project key: **monopolyGruppo**
Provide a token:
monopolyGruppo: 0debbe8ce17e971af3dafc78891a719f0ab7a550

6.2 Con Understand

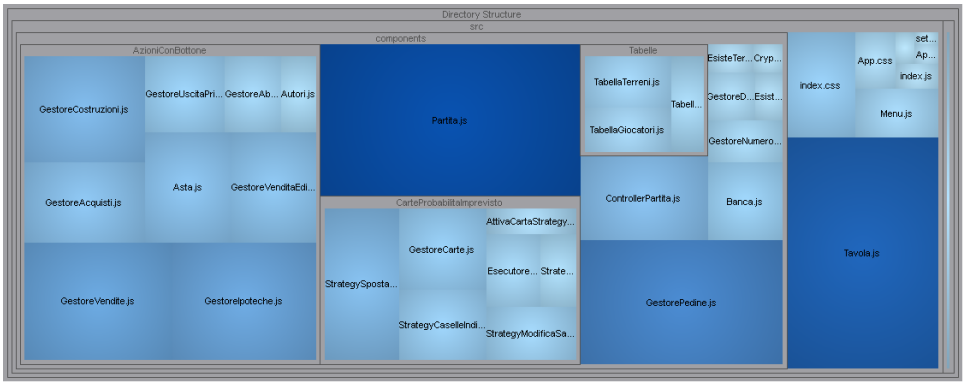
Oltre alle analisi con SonarQube abbiamo utilizzato anche il software Understand. Questo tool è particolarmente utile in quanto analizza il codice e ne riporta le metriche. Grazie a questo tool si possono creare dei grafi che rappresentano le dipendenze tra le classi o più in generale fra le varie parti di codice e individuare vari code-smell. Di seguito riportiamo i risultati dell'analisi che abbiamo svolto con questo software.
La struttura della directory del progetto:



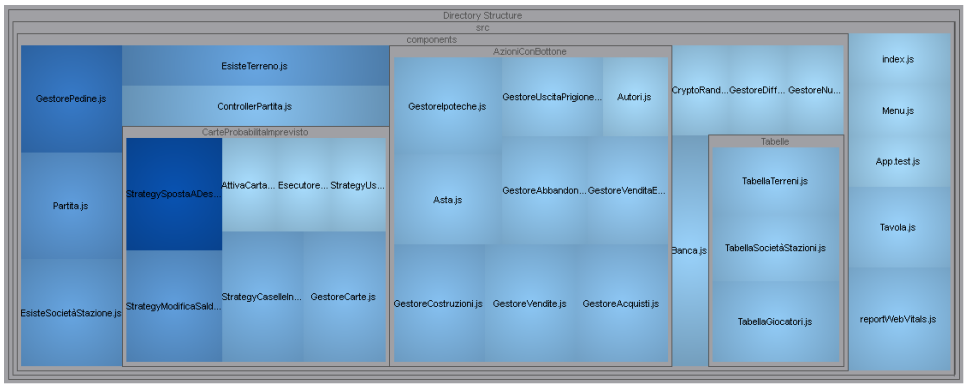
Le dipendenze:



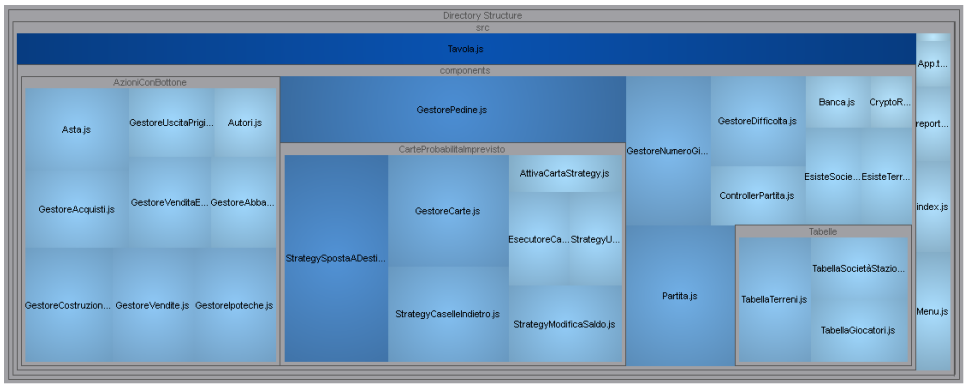
CountLine:



Cyclomatic:



Line Code:



Le criticità principali che sono emerse da quest'analisi con Understand sono:

- File lunghi. React utilizza la sintassi JSX. Questa sintassi è particolarmente verbosa, ma necessaria per la renderizzazione dei componenti. Pertanto non è stato possibile ridurre il numero di queste linee di codice che influenza in maniera decisiva la dimensione di ogni componente.
- Funzione attivaCarta in StrategySpostaADestinazione.js. Secondo Understand questa funzione risulta essere difficile da testare. Si è già provveduto a suddividere la gestione delle singole carte attraverso il design pattern Strategy. Tuttavia la gestione di alcune carte rimane complessa a causa dell'elevata complessità della logica del monopoly e delle sue componenti. Pertanto non è stato possibile intervenire ulteriormente nei tempi concessi per il completamento del progetto.

- Commenti. Un altro problema riguarda il basso numero di commenti. Abbiamo cercato di scrivere un codice molto pulito e autoesplicativo, commentando quando era necessario, per evitare di creare un codice eccessivamente lungo e farraginoso. Nonostante ciò durante le verifiche il numero di righe commentate è risultato essere troppo basso.

7 Diagrammi di Gantt

Confrontando il diagramma di Gantt iniziale e il diagramma di Gantt finale del nostro progetto si potranno notare le seguenti differenze:

1. La fase di progettazione e la fase di implementazione nel primo diagramma non si sovrapponevano. La fase di implementazione iniziava solo dopo la conclusione della fase di progettazione. Nel diagramma finale, invece, la fase di progettazione e la fase di implementazione si sovrappongono leggermente. Questa differenza è dovuta al fatto che prima di iniziare a lavorare a questo progetto non avevamo una grande conoscenza di ReactJS. Per evitare di fare una progettazione che si rivelasse inadatta, abbiamo preferito iniziare a prendere un po la mano, con questa libreria, mentre svolgevamo la fase di progettazione.

2. La prima versione del diagramma di Gantt prevedeva una fase di testing coincidente con la fase di implementazione. In particolare si era deciso di seguire la pratica dello sviluppo guidato dai test suggerita dal metodo agile Extreme Programming. Questa scelta è stata successivamente ritrattata a causa delle difficoltà riscontrate nell'apprendimento del framework React e del poco tempo a disposizione per lo sviluppo di un'applicazione complessa come quella del monopoly. Per questi motivi nel diagramma finale la fase di testing risulta più contenuta. Si precisa qui che i test sono stati eseguiti senza tuttavia predisporre la loro automazione per mancanza di tempo.

3. La scrittura della documentazione è durata un giorno in più del previsto e si è conclusa il 28/01/2021 invece che il 27/01/2021.

4. La revisione finale è stata spostata al 29/01/2021 a causa dei ritardi nella scrittura della documentazione e nella fase di testing.

5. La consegna del progetto è avvenuta il 30/01/2021 invece che il 29/01/2021. Purtroppo GanttProject non permette di inserire delle attività nei giorni del fine settimana perciò abbiamo posizionato la consegna il primo giorno utile (1/02/2021).

Comunque, nonostante le difficoltà e la necessità di rivedere le aspettative previste nel primo diagramma di Gantt, siamo riusciti a consegnare il progetto rispettando le scadenze imposte dalla traccia.

8 Guida all'istallazione

Per poter istallare il monopoly, da noi realizzato, si possono seguire due strade diverse. La prima si basa sull'assunzione che l'utente abbia a disposizione Git e che faccia il download con esso. La seconda, invece, è pensata per quegli utenti che non hanno Git sul loro computer.

8.1 Primo metodo

1. Istallazione di Node.js.
   Per poter scaricare e istallare Node.js sarà sufficiente recarsi al seguente sito (https://nodejs.org/it/) e scaricare, ed installare, la versione più adatta al proprio sistema operativo.

2. Istallazione Visual Studio Code.
   Noi abbiamo scritto la nostra applicazione utilizzando Visual Studio Code come editor di testo. Per poterlo istallare bisognerà andare al seguente sito: (https://code.visualstudio.com/download) e scaricare, ed installare, la versione più adatta al proprio sistema operativo. Bisogna dire, però, che tutte le operazioni che verranno svolte sul terminal di Visual Studio possono essere svolte anche dalla bash di Git.

3. Scaricare il monopoly.
   Per poter scaricare il nostro monopoly sarà sufficiente aprire la bash di Git nella cartella in cui vogliamo istallarlo e scrivere il comando:
   git clone https://github.com/UnimibSoftEngCourse2021/progetto-monopoly-1-gruppo-monopoly-1.git
   Quando l'operazione sarà conclusa avremo una cartella progetto-monopoly-1-gruppo-monopoly-1 con tutto il codice da noi realizzato. Però non saremo ancora pronti a partire perché mancheranno i node-modules necessari al funzionamento del monopoly.

4. Istallazione dei node-modules.
   Per poter istallare i node-modules necessari al funzionamento del monopoly il metodo più comodo e veloce è creare una nuova react-app. Per fare ciò bisogna entrare nella cartella in cui volgiamo istallare il monopoly, aprire la bash di Git e scrivere il comando:
   npx create-react-app nomeApp
   successivamente premere invio. In questo modo verrà creato un nuovo progetto React con i node-modules necessari al suo funzionamento. Quando questa operazione sarà conclusa bisognerà aprire la cartella appena creata che avrà il nome che abbiamo scelto (nel nostro esempio nomeApp) quindi bisognerà prendere la cartella node-modules e spostarla nella cartella del nostro progetto (progetto-monopoly-1-gruppo-monopoly-1). Dopo aver svolto questa operazione possiamo anche eliminare nomeApp.

5. Ultimi node-modules.
   Adesso bisognerà aprire il monopoli in Visual Studio Code.

   - Aprire Visual Studi Code.
   - Premere il tasto Explorer (in alto a sinistra sopra la lente d'ingrandimento).
   - Premere il tasto Open Folder e selezionare la cartella progetto-monopoly-1-gruppo-monopoly-1.
   - Aprire il terminal premendo View e, successivamente, Terminal in alto a sinistra.
   - Se non siamo ancora dentro entrare nella cartella del monopoly con il comando **cd progetto-monopoly-1-gruppo-monopoly-1**.
   - Istallare gli ultimi node-modules che permetteranno l'utilizzo del material design. Per fare ciò si utilizzeranno le seguenti istruzioni:
     npm i –save @material-ui/core
     npm add @material-ui/icons
     se questi comandi non dovessero funzionare bisognerebbe provare a sostituire npm con yarn.

6. Far partire il progetto.
   Adesso siamo pronti per far partire il nostro monopoly basterà scrivere il comando **npm start** e dopo pochi minuti verrà aperta una finestra del browser che ci permetterà di giocare.

8.2 Secondo metodo

1. Installazione di Node.js.
   Per poter scaricare e installare Node.js sarà sufficiente recarsi al seguente sito (https://nodejs.org/it/) e scaricare, ed installare, la versione più adatta al proprio sistema operativo.

2. Installazione dei node-modules.

   - Creare la cartella di installazione del progetto, ad esempio monopolyGruppo1.
   - Aprire cmd (Prompt dei comandi) e spostarsi nella cartella monopolyGruppo1 con il corretto path: **cd C:/. . . path.../monopolyGruppo1**.
   - Quindi, eseguire il seguente comando: **npx create-react-app monopoly**.
   - Spostarsi nella cartella monopoly con **cd monopoly**

3. Ultimi node-modules.
   Una volta all'interno dalla cartella monopoly sarà necessario eseguire questi due comandi in cmd:

   - **npm i –save @material-ui/core**

- **npm add @material-ui/icons**

4. Scaricare il monopoly.
   Per poter scaricare il nostro monopoly bisogna, innanzi tutto, andare sul branch master del nostro progetto all'indirizzo: **https://github.com/-UnimibSoftEngCourse2021/progetto-monopoly-1-gruppo-monopoly-1**. Cliccare su Code download zip, quindi, scompattare. Adesso bisognerà spostarsi nel folder scompattato (**progetto-monopoly-1-gruppo-monopoly-1-develop**) e copiare il contenuto in **C:/... path.../monopoly-Gruppo1/monopoly**. Sovrascrivere i file quando viene chiesto.

5. Far partire il progetto.
   Con il prompt dei comandi eravamo rimasti nella cartella monopoly. Per far partire il progetto basterà svolgere queste due istruzioni:

   - **cd C:/... path.../monopolyGruppo1/monopoly**
   - **npm start**

   Verrà richiesto di consentire l'accesso a Node.js, ma solo la prima volta.
   A questo punto il browser aprirà, nel giro di un minuto, che è il tempo necessario a far partire il server, il gioco.

9 Bibliografia

Riferimenti bibliografici

[1] Fowler M., UML Distilled. Guida rapida al linguaggio di modellazione standard, a cura di Baresi L., Pearson, 2018.

[2] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[3] Larman C., Applicare UML e i pattern. Analisi e progettazione orientata agli oggetti, a cura di Cabibbo L., Pearson, 2016.