“Ingegneria del Software”
2021-2022

Docente: Prof. Angelo Furfaro

<Scale E Serpenti>

Data	<12/09/2022>
Documento	Documento Finale – D3

Team Members		
Nome e Cognome	**Matricola**	**E-mail address**
Andrea Voci	**200612**	**vcondr00a14m208e@studenti.unical.it**

Sommario

List of Challenging/Risky Requirements or Tasks

Challenging Task	**Date the task is identified**	**Date the challenge is resolved**	**Explanation of how the challenge has been managed**
Mazzo da gioco ed eventi	20/08/2022	25/08/2022	Il problema si è presentato durante l'implementazione delle carte da gioco e degli eventi che essi scaturivano. Perciò ho suddiviso il problema in in due sottoproblemi: 1. Tipi di carte pescabili; 2. Gli eventi per tipi di carta; Per il primo ho usato due path Command con cui prendo la richiesta di pescare la carta; per il secondo ho utilizzato Mediator per far combaciare l'evento alla carta pescata.
Giocatore sottoposto a condizioni variabili	20/08/2022	27/08/2022	Il problema si è presentato nel momento in cui il giocatore si trova in una cella e deve comportarsi in maniera diversa a seconda del tipo di cella (normale, scala, serpente, ecc.). Quindi ho utilizzato il path State, ma nel momento in cui usciva da uno stato di stop doveva poter ritornare al suo stato precedente, e ciò mi ha portato ad utilizzare il path Memento.
Salvataggio file	24/08/2022	31/08/2022	Ho inserito un pop-up dove consiglio di salvare direttamente le impostazioni appena personalizzate così che il file potrà poi essere caricato e letto in maniera corretta dall'applicazione, che avvierà direttamente la partita con le impostazioni definite in esso.

A. Stato dell'Arte List of Challenging/Risky Requirements or Tasks

Scale e serpenti nasce in Inghilterra per emulare il famoso gioco dell'oca, ed è un gioco il cui esito della partita è dettato tutto dal lancio dei dadi.
Si stabiliscono delle regole di base:
si imposta la grandezza del tabellone di gioco, fornendo il numero di righe e colonne, e si sceglie il numero di giocatori (1-2) e il numero di dadi (1-2); ogni lancio fa spostare il giocatore di una certa quantità; una cella (sulla quale il giocatore si posiziona e finisce il suo turno) che può essere vuota o speciale:

- **Serpente**: questa cella in realtà si distingue tra Testa e Coda ma sono collegate fra di loro per il semplice fatto che se il giocatore finisce sulla Testa, allora si ritroverà a percorrere il serpente fino alla sua coda e, quindi, a scendere di posizione;
- **Scala**: come nel Serpente, anche essa è composta da una Fine e una Cima e permetterà al giocatore finito sulla Fine di salire la scala fino alla Cima e, quindi, di saltare in avanti di posizione;
- **Sosta**: queste celle bloccano il giocatore per un tot di turni a seconda del tipo di cella, ovvero "panchina" (stop per un turno) e "locanda" (stop per 3 turni);
- **Premio**: queste celle, appunto, donano al giocatore dei premi per continuare ad avanzare di tot celle, in particolare la cella premio "Dadi" permetterà al giocatore di rilanciare i dadi e la cella premio "Molla" permetterà al giocatore di avanzare in avanti del numero di dadi precedente;
- **Pesca**: questa cella permette di pescare delle carte da un mazzo, utilizzarle al momento e rimetterle in fondo e le loro caratteristiche sono le stesse delle celle Sosta e Premi, con l'aggiunta di una carta Divieto di Sosta che permetterà di evitare la stop su una panchina o su una locanda;

Un'altra regola è che nel momento in cui un giocatore si trova nelle posizioni finali e lancia i dadi, per arrivare alla posizione finale deve effettuare un numero esatto nel lancio perché se il numero supera la dimensione massima il giocatore dovrà tornare indietro della differenza.
Il gioco puo' essere giocato nella sua versione base, con scale e serpenti, oppure con l'aggiunta delle altre celle speciali.

B. Raffinamento dei Requisiti

Il cliente ha effettuato una serie di richieste, tra cui la possibilità di simulare una partita osservandola e con la possibilità di mandare avanti le mosse, salvare e caricare una configurazione, impostare a proprio piacimento il numero di righe, colonne, dadi, scale, serpenti e quali celle speciali avere nel tabellone di gioco.

B.1 Servizi (con prioritizzazione)

- **Alta**
  - Gestione delle impostazioni di gioco per un'esperienza più variegata;
  - Visonamento della partita in modalità passiva o attiva (mandando avanti le mosse manualmente);
  - Posizionamento corretto delle celle speciali senza sovrapposizioni;
- **Media**
  - Le celle speciali richieste;
  - Controllo dei vincoli che ogni oggetto di gioco, utilizzato o non, comporta al gioco;
- **Bassa**
  - Salvataggio e caricamento di configurazioni di gioco su file;
  - Possibilità di poter scegliere la posizione dei serpenti e delle scale;

B.2 Requisiti non Funzionali

- Usabilità: l'interfaccia è stata resa semplice per essere usata da un qualsiasi tipo di utente;
- Robustezza: controllo di tutti i possibili errori che un utente potrebbe commettere durante la configurazione della partita o del caricamento di un file errato;
- Riusabilità: la struttura è stata impostata in modo modulare per permettere un aggiornamento dell'applicazione con nuove versioni del gioco o un semplice miglioramento dell'interfaccia;
- Manutenibilità: come spiegato prima, è possibile aggiornare l'applicazione con aggiunte di nuove celle speciali o una possibile modifica della struttura;
- Correttezza: il gioco rispetta tutte le regole e caratteristiche del documento di specifica dei requisiti;

B.3 Scenari d'uso dettagliati

Il sistema mette a disposizione una completa caratterizzazione del tabellone di gioco e della sua grandezza oltre a poter osservare in maniera passiva e attiva la partita.

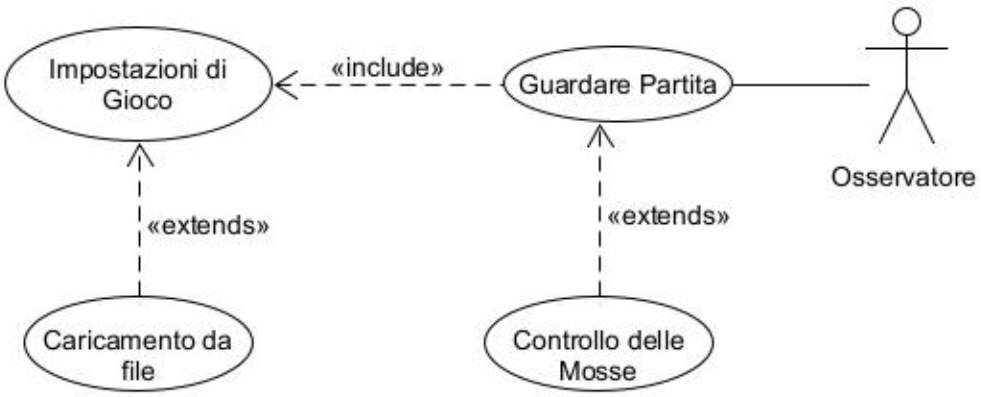
B.4 Excluded Requirements

La possibilità di scegliere le posizioni delle scale e dei serpenti per mancanza di tempo in vista della data di consegna;

B.5 Assunzioni

Le partite potranno avere un massimo di 4 giocatori, 3 dadi, 10 righe e colonne, e un numero massimo di scale e serpenti proporzionato al numero di righe e colonne per via della loro occupazione di 2 celle per singolo.

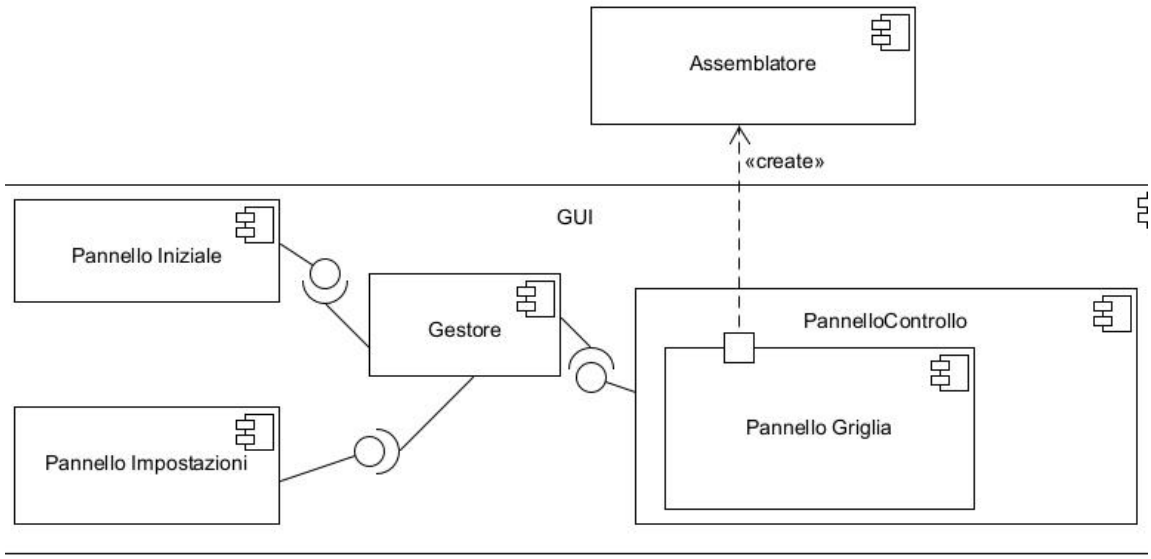
B.6 Use Case Diagrams



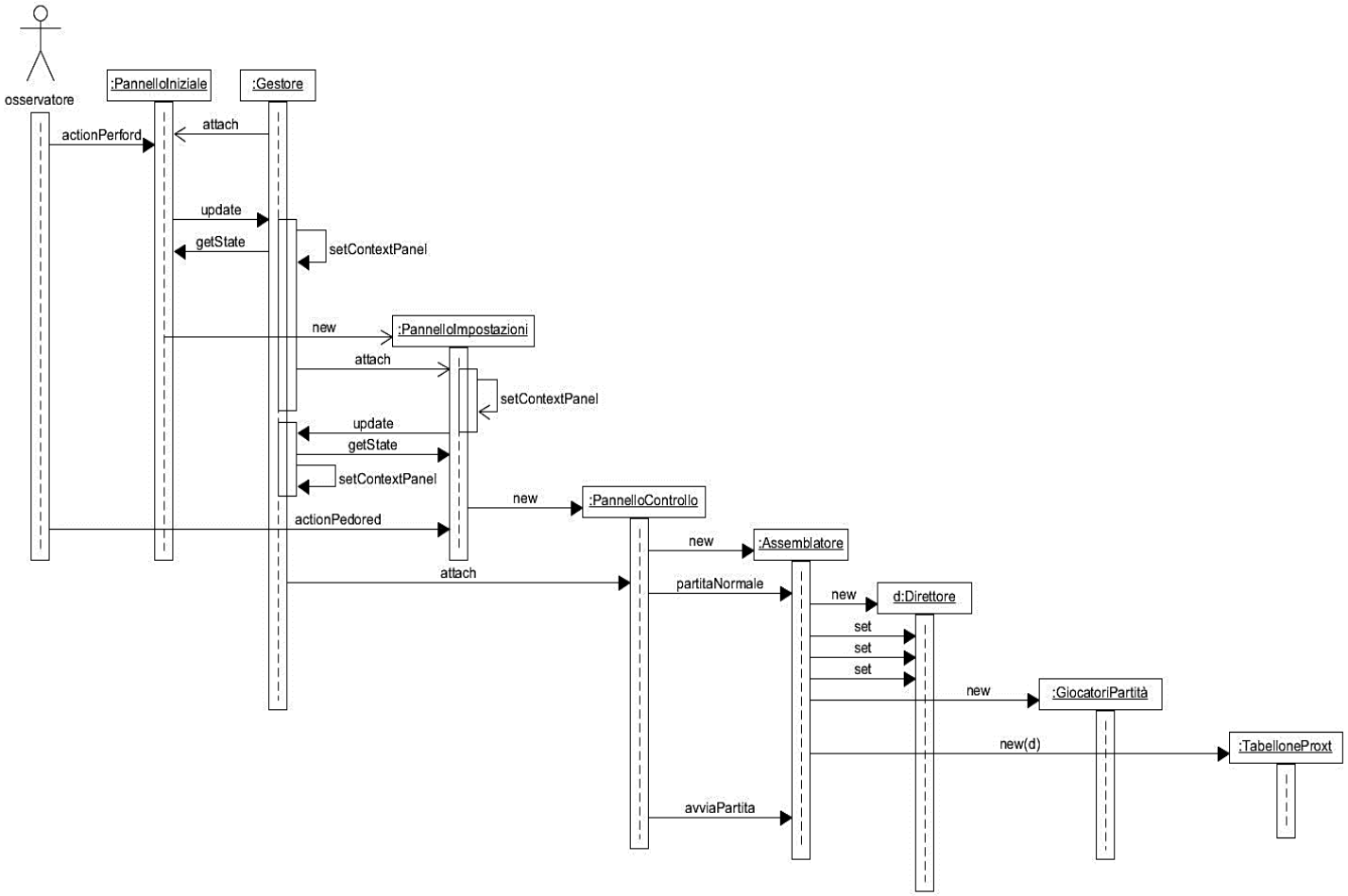
1. ***Guardare Partita***
   a. Il cliente specifica le varie Impostazioni di Gioco e le conferma;
   b. Il sistema viene configurato e avviato;
   c. Il cliente osserva la partita e il vincitore;
2. ***Controllo delle Mosse (external)***
   a. Il cliente fa avanzare la partita controllandone le mosse;
3. ***Impostazioni di Gioco***
   a. Il cliente definisce il numero di righe e colonne;
   b. Il cliente definisce il numero di scale e serpenti;
   c. Il cliente definisce il numero di giocatori e dadi;
   d. Il cliente definisce quali regole potrà avere il gioco (celle speciali e la possibilità di controllare le mosse);
   e. Il sistema riceve le impostazioni;
4. ***Caricamento da file (external)***
   a. Il cliente decide di caricare le impostazioni personalizzate da un file;

C. Architettura Software

C.1 The static view of the system: Component Diagram



C.2 The dynamic view of the software architecture: Sequence Diagram



Le partite potranno avere un massimo di 4 giocatori, 4 dadi, 10 righe e colonne, e un numero massimo di scale e serpenti proporzionato al numero di righe e colonne per via della loro occupazione di 2 celle per singolo.

D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

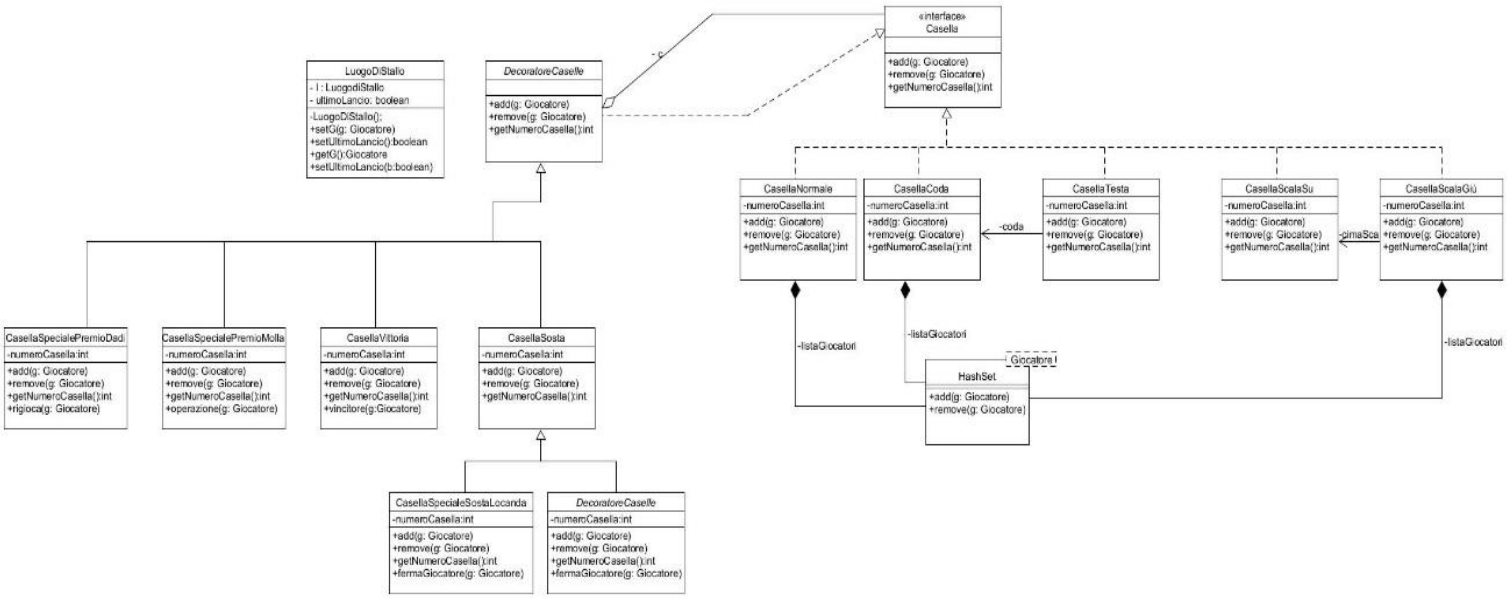
Il cliente potrà salvare le impostazoni su file creati tramite strutture java presenti nel sistema.

E. Scelte Progettuali (Design Decisions)

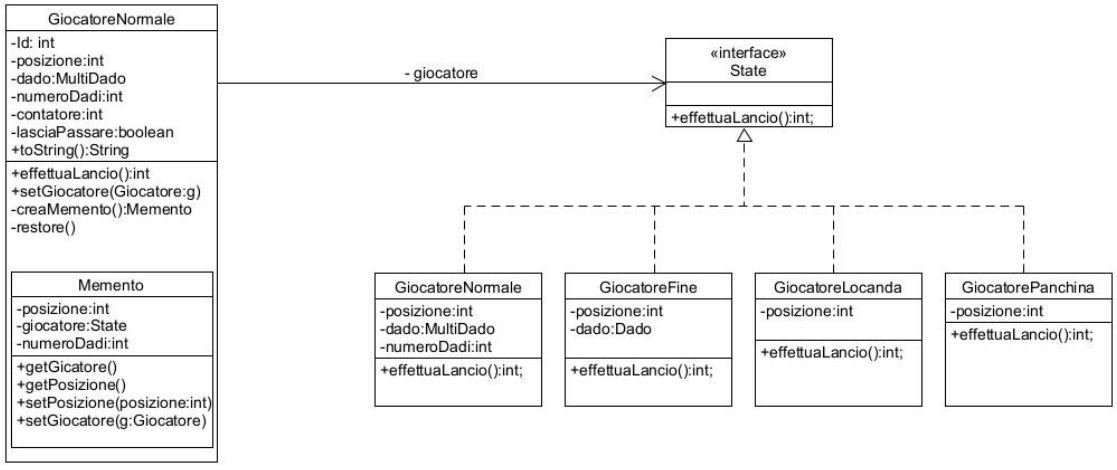
1. **State e Memento**: durante la partita capita che un giocatore venga sottoposto a cambi di stato, per via di alcune condizioni presenti in partita, e per questo ho utilizzato il pattern State; il pattern Memento l'ho utilizzato per riportare il giocatore ad uno stato precedente a seguito della fine della condizione;
2. **Decorator e Adapter**: per realizzare le caselle ho implementato un Set per poter contenere tutti i giocatori all'interno della casella stessa e, quindi, l'invocazione del metodo add sulla casella implica l'invocazione del metodo add anche sul Set; il Decorator è stato utilizzato per poter assegnare alle caselle una specifica caratteristica, nulla o speciale che sia, per poter gestire meglio le loro funzionalità;
3. **Command e Mediator**: per utilizzare la casella speciale Pesca ho dovuto implementare la classe Carte per effettuare l'operazione di mischia e ho utilizzato una classe Pescatore, che sarà l'oggetto Command, per effettuare le operazioni di pesca e posa; la classe Mazzo, che effettua l'operazione di pesca direttamente da Pescatore, passerà la carta pescata ad un oggetto Evento che, tramite Mediator, scaturirà l'evento di quel tipo di carta;
4. **Proxy e Builder**: per costruire il tabellone (classe TabelloneConcreto) mi sono affidato ad un'interfaccia Builder e, quindi, ad un BuilderConcreto per inserire tutti gli oggetti e le caratteristiche nelle varie celle; ho poi usato una classe TabellaProxy per effettuare in maniera sicura le operazioni di aggiunta e rimozione dei giocatori nella tabella (invocandole sul TabelloneConcreto);
5. **Singleton**: tra i vari oggetti di gioco troviamo il Dado e il Mazzo e sono tutti degli oggetti Singleton, così come il Luogo di Stallo, e ciò implica che avrò una sola istanza di questi oggetti che verranno riutilizzati durante la partita;
6. **Facade**: la classe Assemblatore funge da vero e proprio oggetto di tipo Facade gestendo e raggruppando al suo interno le classi del sottosistema;

F. Progettazione di Basso Livello

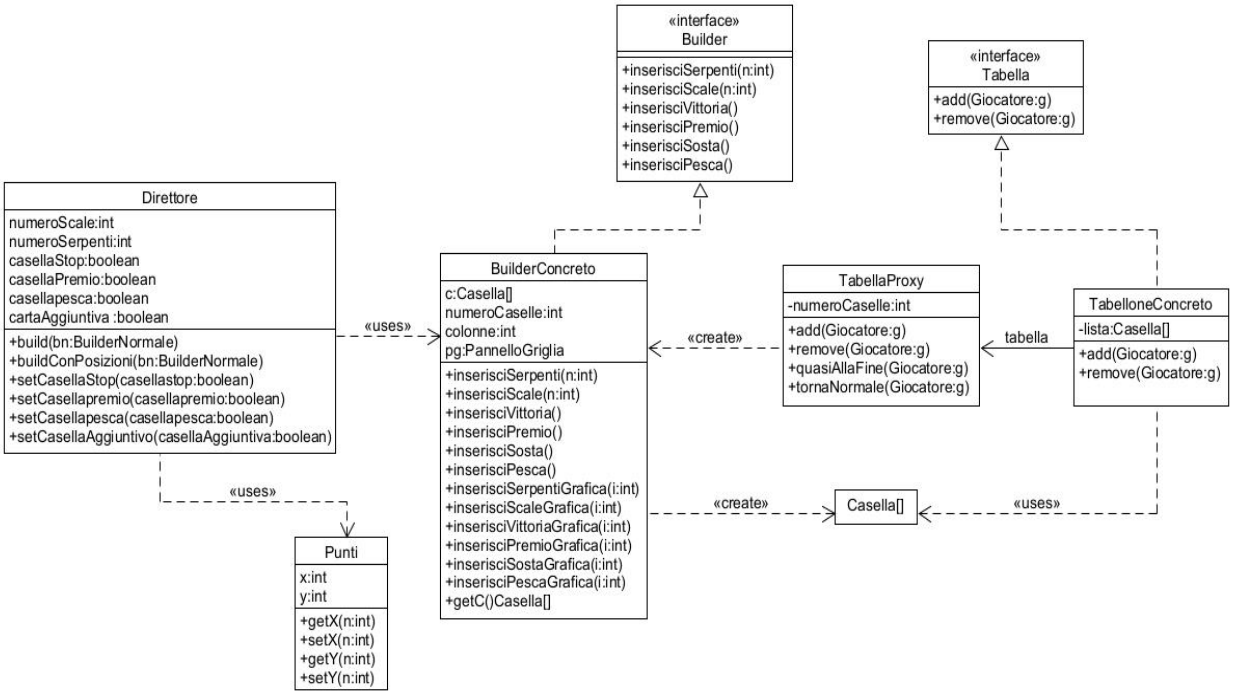
Package caselle:



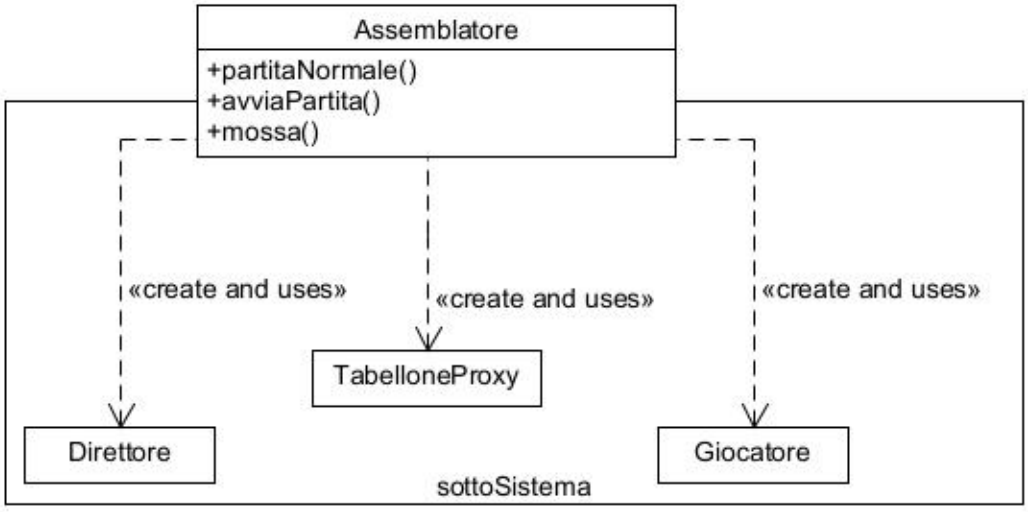
Package giocatori:



Package tabella:



Package sottoSistema:



G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)

FRs:

1. Utilizzando un Builder e un Direttore l'applicazione è in grado di leggere i valori forniti dal cliente al momento della personalizzazione delle impostazioni di gioco e creare la partita con le caratteristiche appena scelte. Questo grazie al Direttore che legge i valori e li invoca all'interno del Builder;
2. Una volta spuntata la casella check per il Controllo Mosse, verrà a crearsi un pulsante nel PannelloControllo che invocherà il metodo mossa nella classe Assemblatore in modo tale da monitorare la partita in maniera attiva. In caso di mancata spunta della casella, l'Assemblatore provvederà ad effettuare la mossa ogni 2.5 s (tramite Thread);
3. La creazione del tabellone di gioco avviene tramite Builder, che andrà a supervisionare le posizioni delle celle e le loro caratteristiche in modo da evitare conflitti con quelle speciali; il supervisionamento dei giocatori (per non farli andare fuori struttura) spetta alla TabellaProxy, che, dopo tutti i vari controlli, aggiungerà i giocatori nella posizione corretta;
4. Ogni cella rispetta la sua gerarchia, da CellaVuota e CellaVittoria, e le loro operazioni verranno svolte separatamente dal Direttore per garantire una corretta distinzione tra esse;
5. In un futuro aggiornamento della versione dell'applicazione verrà aggiunta la possibilità di scegliere la posizione di scale e serpenti, che verrà letta e interpretata dal Direttore per costruire il nuovo tabellone personalizzato tramite Builder;

NFRs:

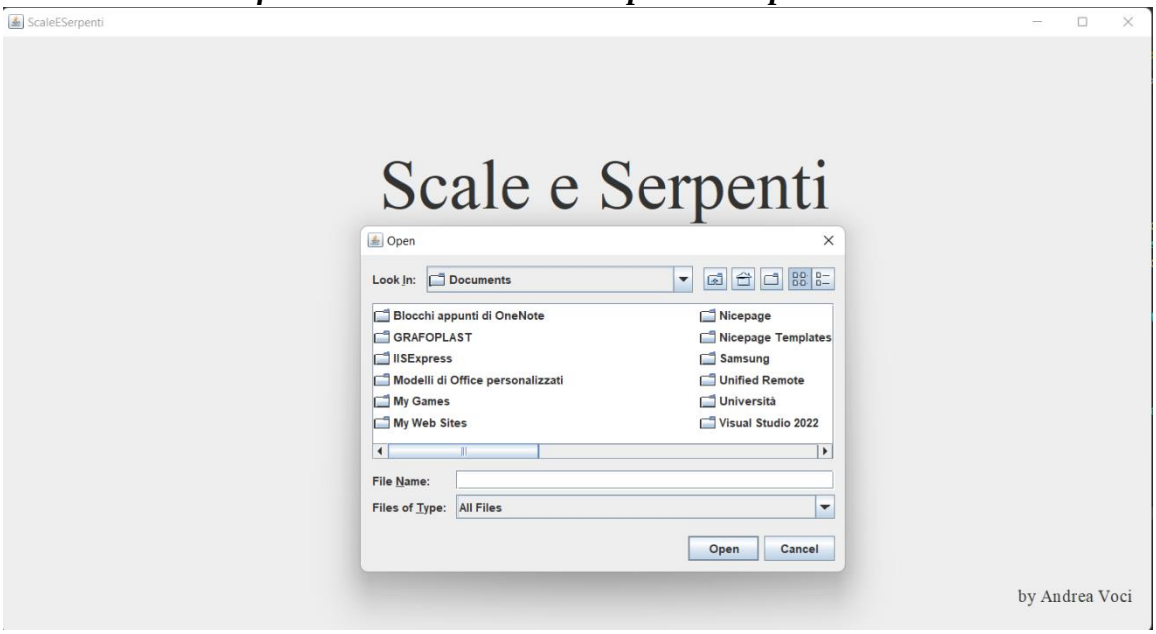
L'utilizzo di diversi Design Path ha reso il sistema modulare e quindi più semplice da gestire, soprattutto in caso di malfunzionameno o correzione di una specifica classe. Ciò porta ad avere, quindi, una buona Riusabilità, Riparabilità ed Evolvibilità. Il sistema risulta conforme a quello richiesto dal cliente risultando di conseguenza Corretto ed Usabile oltre che Robusto per via dell'inserimento di alcune implementazioni e controlli aggiuntivi;

Appendix. Prototype

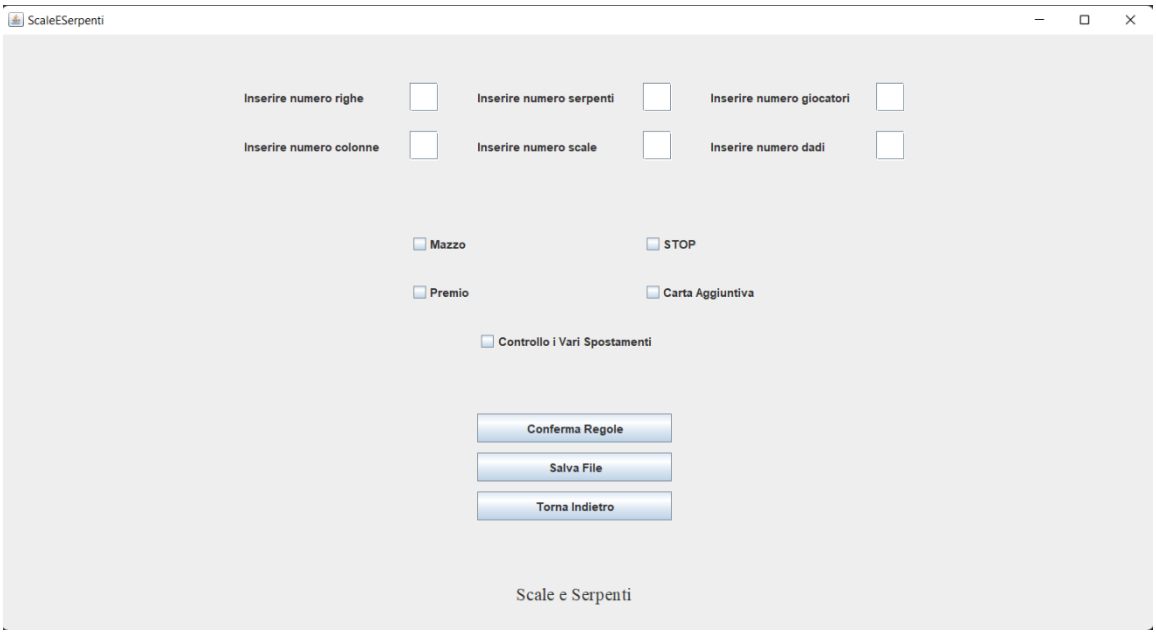
Di seguito vengono mostrati degli screenshots dell'applicazione:

Menù principale:

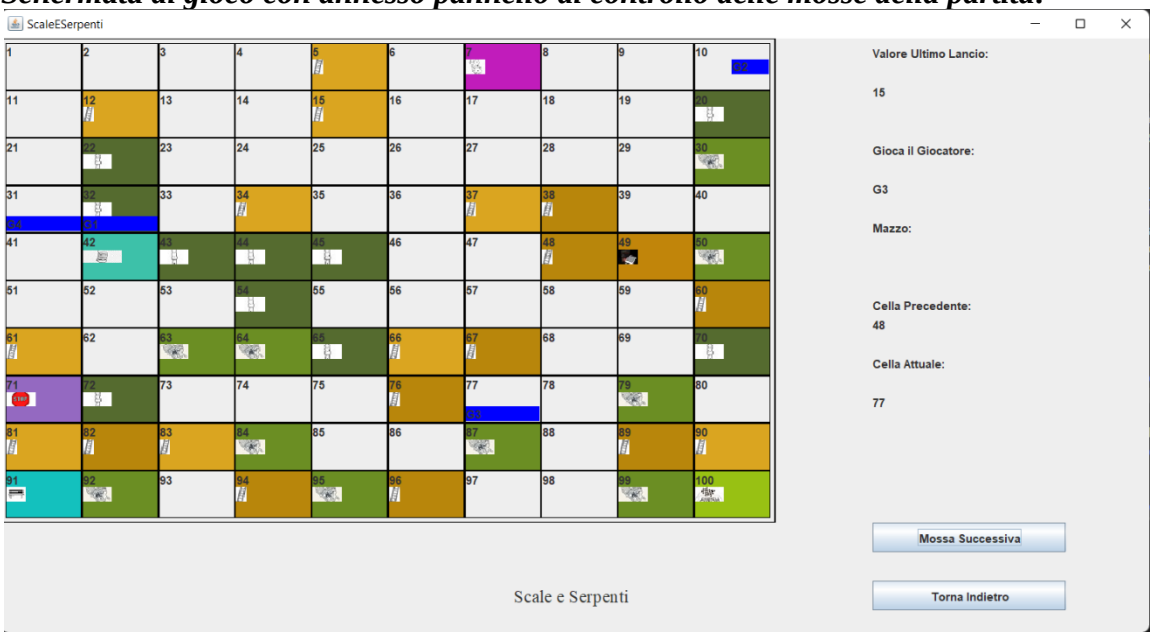
Caricamento di un file esterno contenente le impostazioni personalizzate:



Schermata per il setup del tabellone di gioco, con possibilità di salvarlo in formato file:



Schermata di gioco con annesso pannello di controllo delle mosse della partita:



Schermata di fine partita con pop-up che annuncia il vincitore: