“Ingegneria del Software”

2023-2024

Docente: Prof. Angelo Furfaro, Studente: Stefano Scarcelli - 230668
<scrsfn02t15d086u@studenti.unical.it>

Documento Finale (D3) | 2024-07-14

Table of Contents

A. Stato dell’Arte . . . 2
A.1 Web application: . . . 2
A.2 PC & Game console: . . . 2
A.3 Mobile game: . . . 3
B. Raffinamento dei Requisiti . . . 4
B.1 Servizi (con prioritizzazione) . . . 4
B.2 Requisiti non Funzionali . . . 5
B.3 Scenari d’uso dettagliati . . . 5
B.4 Requisiti esclusi . . . 5
B.5 Assunzioni . . . 6
B.6 Use Case Diagrams . . . 7
C. Architettura Software . . . 8
C.1 La visione statica del sistema: Diagramma dei Componenti . . . 8
C.2 La visione dinamica dell’architettura software: Sequence Diagram . . . 9
D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS) . . . 11
E. Scelte Progettuali (Design Decisions) . . . 12
E.1 Definizione delle entità della simulazione . . . 12
E.2 Gestione delle relazioni tra entità della simulazione (Mediator) . . . 13
E.3 Costruzione del tabellone (Strategy) . . . 13
E.4 Gestione del main loop della simulazione . . . 13
E.5 Comunicazione GUI-Simulazione . . . 14
F. Progettazione di Basso Livello . . . 15
F.1 Simulazione . . . 15
F.2 GUI . . . 15
G. Spiegare come il progetto soddisfa i requisiti funzionali (RF) e quelli non funzionali (RNF) . . . 17
G.1 RF . . . 17
G.2 RNF . . . 18
Appendice. Prototipo . . . 19

Table 1. **List of Challenging/Risky Requirements or Tasks**

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Impostazione del *progetto* con sistema di automazione di **build**, **test** e gestore delle **dipendenze**	17/06/2024	18/06/2024	Tentativo di uso di **Maven**, poi scartato per via di errori nella gestione della dependency di *JavaFX*, e scelta della soluzione tramite **Gradle**
Esecuzione delle *azioni* delle varie celle sui **giocatori** permettendo *catene di eventi*	04/07/2024	06/07/2024	Spezzettamento della logica dei turni in funzioni *richiamate a catena* e uso del pattern **Mediator** per determinare il tipo di *azione* da eseguire tramite gli oggetti `Cell` mantenendo la responsabilità dell'esecuzione all'oggetto `Match`
Gestione evento di *scarto di una carta* da parte del **giocatore** nel **mazzo**	06/07/2024	06/07/2024	Uso del pattern **Mediator** precedentemente implementato per gestire la *notifica* da parte di un **giocatore** dell'*inserimento nel fondo del mazzo* di una **carta**
Gestione costruzione del **tabellone** in più modi *differenti*	09/07/2024	11/07/2024	Uso del pattern **Strategy** con la definizione di diversi *oggetti* (*strategie*) per la costruzione del **tabellone**

A. Stato dell'Arte

Analizzate sistemi esistenti, prendete spunto da cio' che esiste.

A.1 Web application:

- **Snakes and Ladders** (Play on CrazyGames)
  - Presenza di due modalità con rappresentazione grafica differente;
  - Possibilità di giocare **Player VS Player** (tutti contro tutti) o **Player VS Computer** (1 VS Computer);
  - Possibilità di giocare da **2** a **6** *giocatori*;
  - Animazione lancio di dadi (pre renderizzata, non in real time);
  - Indicatore di *turno*;
  - Piccola schermata di spiegazione delle **regole di gioco**;
- **Play Snakes and Ladders** (Calculators.org)
  - Grafica stilizzata;
  - Modalità **Single Player** e **Multi Player**;
  - Possibilità di giocare da **2** a **4** *giocatori*;
  - Effetti sonori;
  - **Leader board**;
  - Indicatore di *turno*;
- **Snakes And Ladders** (Playonlinedicegames.com)
  - Possibilità di giocare **Player VS Player** (1 VS 1) o **Player VS Computer** (1 VS Computer);
  - Interfaccia semplice;
  - Animazioni di turno rapide;
- **Snakes And Ladders** (Atlantic Lottery)
  - Gioco d'azzardo;

A.2 PC & Game console:

- **Snakes & Ladders+ : Board Game** (Link)
  - Tema marino;
  - Possibilità di giocare **Player VS Player** (1 VS 1) o **Player VS Computer** (1 VS Computer);
  - Diversi *tabelloni*;
  - Spiegazione esaustiva delle **regole di gioco**;
- **Snake and Ladder Game for Windows 10** (Link)
  - Giocabile da **2** a *più giocatori* (non specificato nella descrizione);

    - Tabellone da **100** *caselle*;
    - Possibilità di scegliere il **colore** del *tabellone* tra vari preset;

A.3 Mobile game:

- **Snakes and Ladders** (Link)
    - Vari **temi** e **tipologie** di *tabelloni*;
    - Possibilità di giocare da **2** a **4** *giocatori*;
    - Possibilità di selezionare *individualmente* per pedina se *escluderla*, inserirla come **giocatore** o inserirla come **computer**;
- **Snakes and Ladders Board Games** (Link)
    - Grafica 3D;
    - Gioco a **livelli**;
    - Il *tabellone* è sostituito con un **percorso** di una *mappa a tema*;

B. Raffinamento dei Requisiti

A partire dai servizi minimali richiesti, raffinate la descrizione dei servizi offerti dal vostro applicativo. Descrivete anche I requisiti non funzionali.

B.1 Servizi (con prioritizzazione)

*Descrivete in **dettaglio** i servizi offerti dal vostro Sistema, insieme a quelli che ritenete siano le soluzioni concettuali necessarie. In questa fase, non fate riferimento ad alcuna tecnologia specifica. Se volete, intervistate stakeholder e collezionate dati dal web o da altre sorgenti. Dovete acquisire una conoscenza avanzata dei problemi associate ai vostri servizi. Assegnate un ID a ciascun servizio. Prioritizzate inoltre i servizi in base a due scale: Importanza alta, media, bassa. Complessità alta, media, bassa.*

1. *Impostazione della partita da simulare:*
    a. Interfaccia di setup (**Importanza** *media*, **Complessità** *bassa*)
        i. Impostazione dimensione tabellone
        ii. Impostazione posizione delle caselle
    b. Salvare/caricare setup (**Importanza** *bassa*, **Complessità** *media*)
2. *Simulazione della partita:*
    a. Avanzamento automatico e manuale dei turni (**Importanza** *alta*, **Complessità** *bassa*)
    b. Lancio dei dai (**Importanza** *alta*, **Complessità** *media*)
        i. Dadi doppi o singoli
        ii. Regola del dado singolo
        iii. Regola del doppio sei
    c. Regole di movimento delle pedine (**Importanza** *alta*, **Complessità** *alta*)
        i. Caselle "Scale & Serpenti"
        ii. Caselle "Sosta"
        iii. Caselle "Premio"
        iv. Caselle "Pesca una Carta"
    d. Mazzo di carte (**Importanza** *media*, **Complessità** *alta*)
        i. Carte standard
        ii. Carte extra
3. *Visualizzazione dei turni simulati:*
    a. Rappresentazione del tabellone (**Importanza** *media*, **Complessità** *alta*)
    b. Rappresentazione delle pedine sul tabellone (**Importanza** *media*, **Complessità** *bassa*)
4. *Visualizzazione dei risultati della partita:*
    a. Vincitore della partita (**Importanza** *alta*, **Complessità** *bassa*)

b. Classifica finale (**Importanza** *bassa*, **Complessità** *bassa*)

Table 2. Importanza/Complessità dei servizi

	Complessità			
Importanza		**Bassa**	**Media**	**Alta**
	Bassa	4.b	1.b	
	Media	1.a, 3.b		2.d, 3.a
	Alta	2.a, 4.a	2.b	2.c

B.2 Requisiti non Funzionali

Elencare i requisiti non funzionali più' importanti per il vostro Sistema.

1. *Interfaccia grafica* (**GUI**):
   a. Menu principale
   b. Interfaccia di simulazione
2. *Regole:*
   a. Numero di dadi
   b. Tipologia dei dadi (numero di facce)
3. *Simulazione della partita:*
   a. Simulazione partite su thread separato
4. *Termine simulazione:*
   a. Salvare log della simulazione

B.3 Scenari d'uso dettagliati

Descrivere gli scenari più comuni, più interessanti, o più complicati d'uso dei vostri servizi.

- **Analisi di dati in vari scenari**:
  - Analizzare quanto il *valore dei dadi* di un giocatore contribuisce sul suo *tasso di vincita*
  - Analizzare come cambiano la *durata di ogni partita* al *variare delle regole*
  - Analizzare quali sono le *caselle più visitate* dai vari giocatori
- **Gioco**:
  - Eseguire *partite con un gruppo di giocatori*
  - Giocare a *prevedere quale pedina vincerà* la partita

B.4 Requisiti esclusi

Descrivere i servizi eventualmente i esclusi, e spiegare il perché

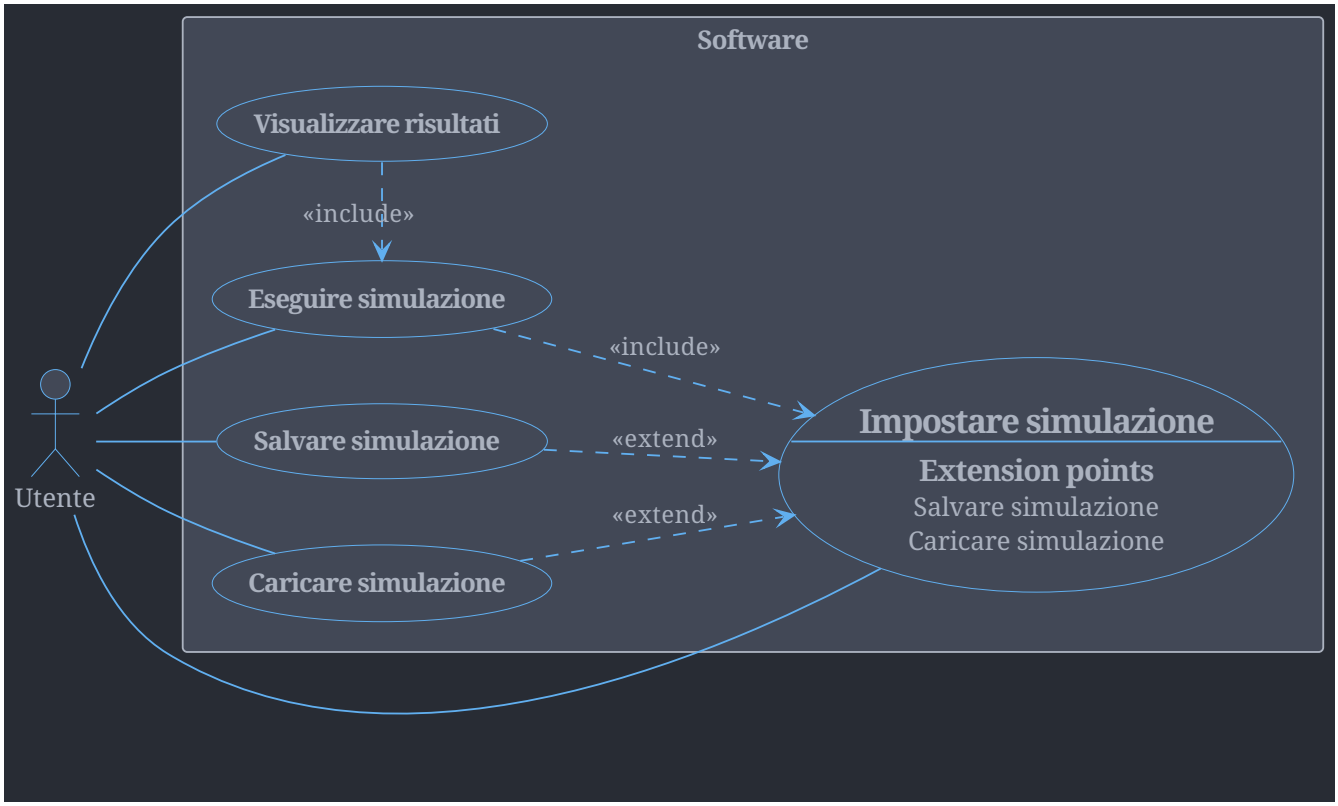
1. *Animazioni 3D:*
    a. **Animazioni lancio dadi in 3D** (*Richiede l'uso di un engine 3D portando solo un miglioramento visivo, fuori dallo scopo del software*)
    b. **Animazioni mazzo di carte in 3D** (*Richiede l'uso di un engine 3D portando solo un miglioramento visivo, fuori dallo scopo del software*)
2. *Audio*
    a. **Effetti sonori** (*Richiedono la creazione o licenza di effetti sonori portando solo un miglioramento visivo, fuori dallo scopo del software*)
3. *Simulazione:*
    a. **Simulazione singola partita in multithread** (*Non è richiesto eseguire simulazioni a velocità elevate per una singola partita, in più aumenta esponenzialmente la complessità del software potenzialmente senza alcun beneficio tangibile*)
    b. **Simulazione di più partite rapide con diverse impostazioni con gestione in coda** (*Scenario estremo non di uso comune che può essere facilmente bypassato aspettando il termine delle varie simulazioni*)
4. *Piattaforme di distribuzione:*
    a. **Piattaforme mobile** (*Software principalmente simulatilo, uso in mobilità poco utile*)
    b. **Piattaforme web** (*Implementazione web app fuori dagli scopi del progetto*)
5. *Interazioni & Integrazioni:*
    a. **API** (*Potenzialmente utile per automatization di task e integrazione in altri software ma fuori dallo scopo del progetto*)
    b. **Supporto a mod o plugin** (*Potenzialmente utile per aggiungere funzionalità extra in modo rapido da parte dell'utente ma aumento della complessità e del rischio di introduzione di bug o vulnerabilità, con conseguente aumento della complessita per il supporto post rilascio*)

B.5 Assunzioni

Documenta brevemente, in questa sezione, le ipotesi/decisioni sui requisiti più rilevanti che hai dovuto prendere durante il tuo progetto

- Il *software* è pensato come uno **strumento di simulazione** per tanto *l'interazione dell'utente* durate una partita è ridotto al minimo;
- Il *software* ha come obbiettivo solo quello di **ricavare i dati** dalle simulazione, non quello di aiutare l'utente ad *interpretarli*, per tanto la visualizzazione di essi potrebbe risultare parziale e non esaustiva. Per questo motivo **l'esportazione dei dati** risulta essere una funzionalità *relativamente importante* per l'obbiettivo richiesto;
- Il *software* è pensato per essere eseguito su **piattaforma PC** con *hardware relativamente moderno* senza ottimizzazioni estensive sia dal punto di vista della *simulazione* che della *GUI*;
- Si riserva che qualsiasi dei Requisiti esclusi possano essere aggiunti in **versioni future** del *software*;

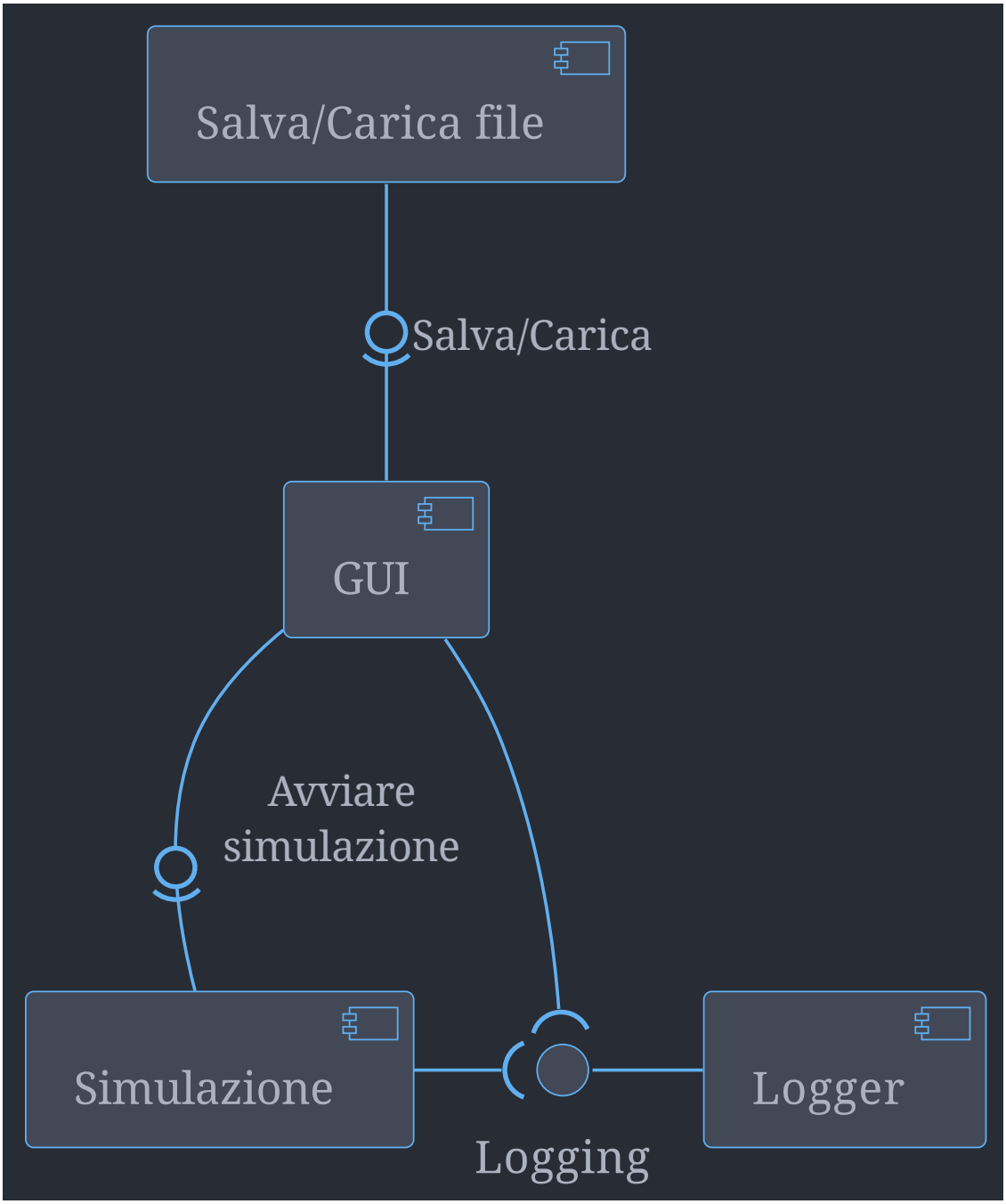
B.6 Use Case Diagrams



C. Architettura Software

SE PERTINENTE, riporta qui sia la vista statica che dinamica della progettazione del tuo sistema, in termini di diagramma dei componenti e i relativi diagrammi di sequenza.

C.1 La visione statica del sistema: Diagramma dei Componenti



C.2 La visione dinamica dell’architettura software: Sequence Diagram

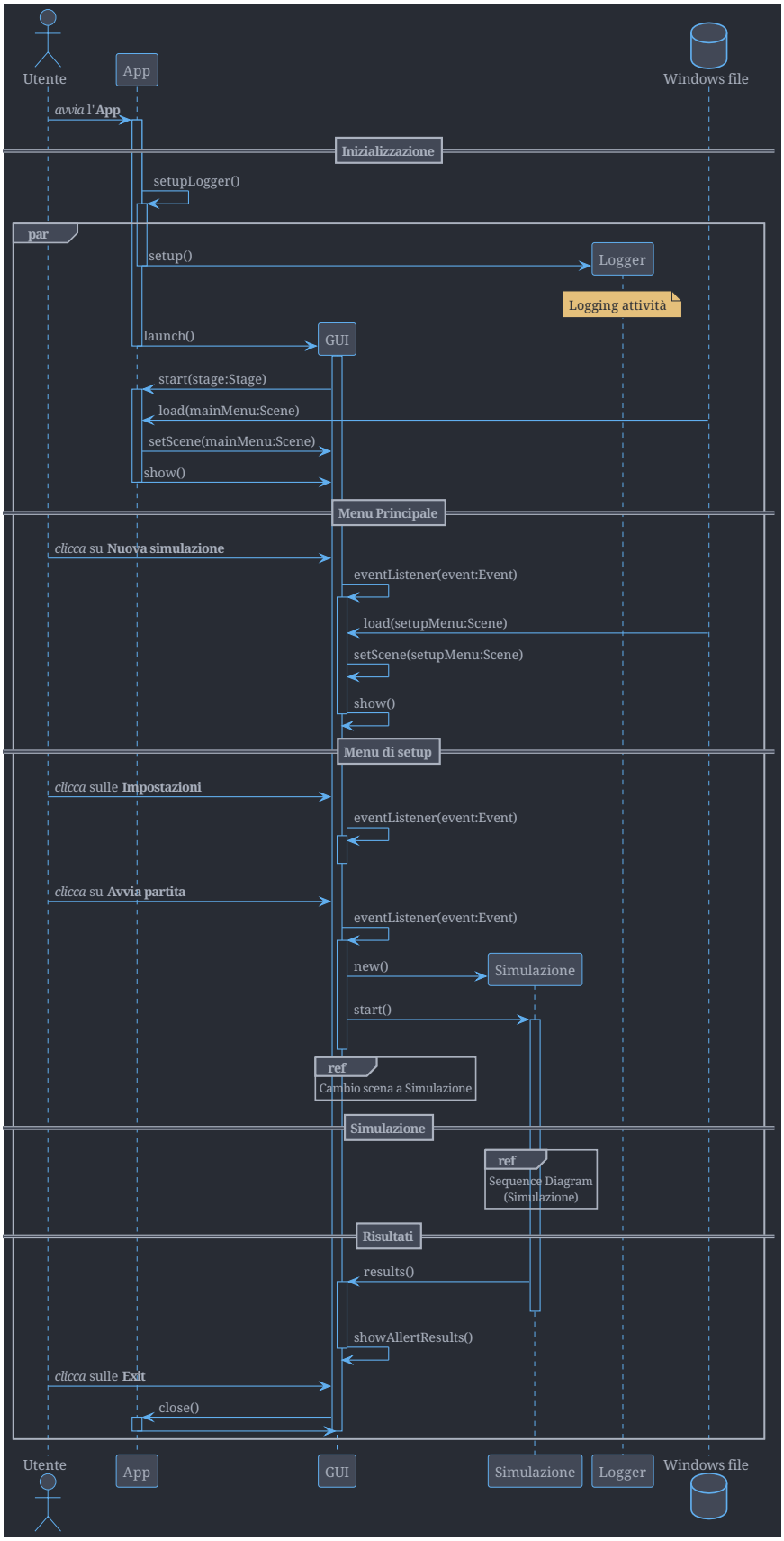


Figure 1. Sequence Diagram (GUI)

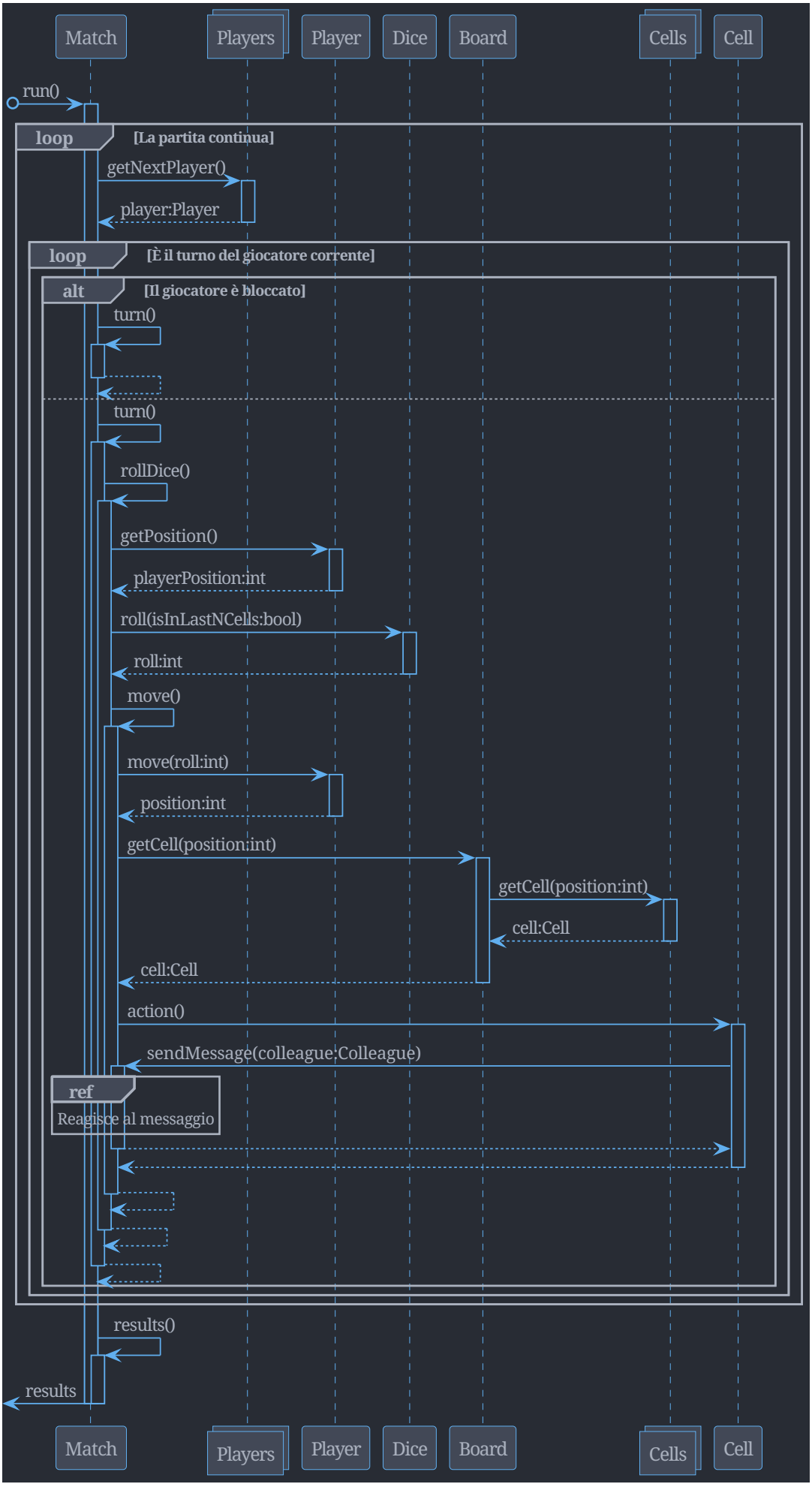


Figure 2. Sequence Diagram (Simulazione)

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

Definite le sorgenti di dati a voi necessarie per realizzare I servizi di cui sopra. Modellate tali dati tramite un ER o similari. Specificate se e quali di tali dati sono gia' forniti da applicativi esistenti.

Il sistema non fa uso di DBMS.

E. Scelte Progettuali (Design Decisions)

Documenta qui le 5 decisioni progettuali più importanti che hai dovuto prendere. È possibile utilizzare sia una specifica testuale che schematica.

E.1 Definizione delle entità della simulazione

Come prima cosa ho deciso di partire dalla definizione delle *entità* che rappresenteranno la **partita**, sia dal punto di vista delle *responsabilità* che da quello della distribuzione delle varie *proprietà* (impostazioni) della **partita**.

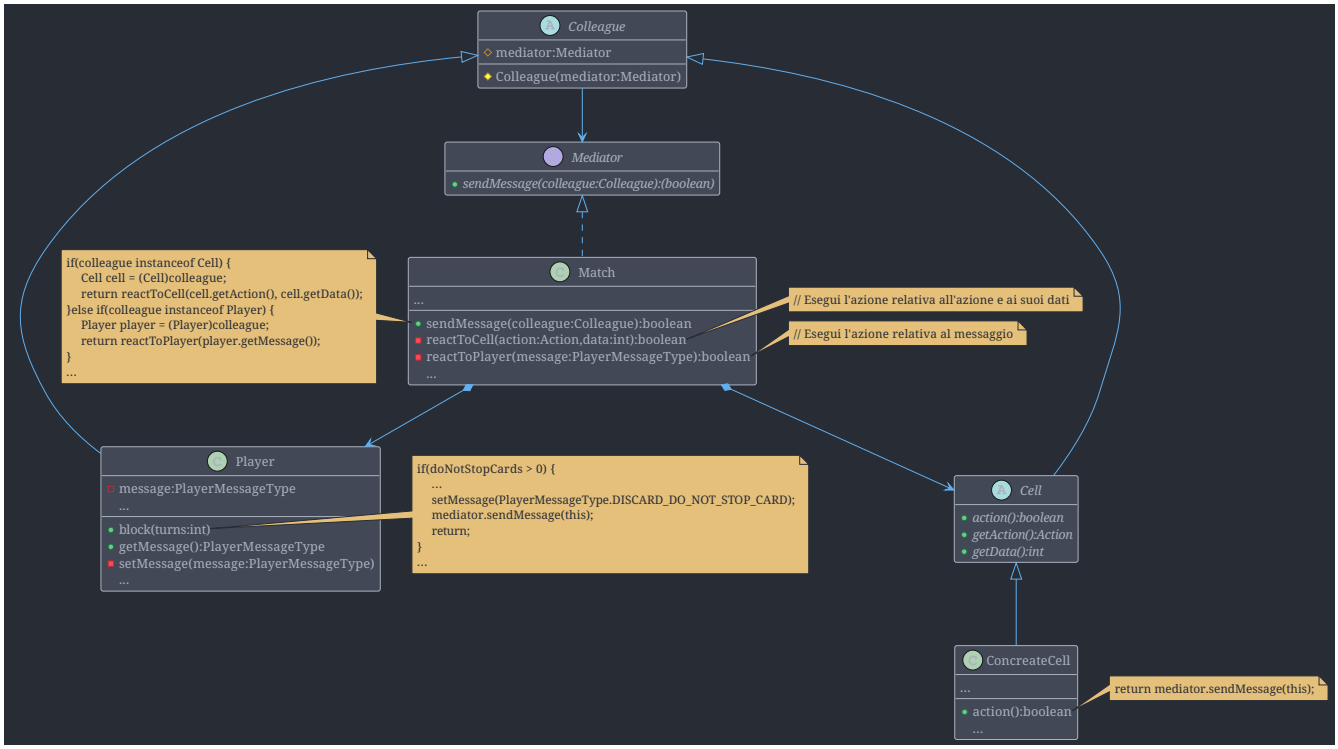
Ho optato per una struttura *simil-albero*, con un entità principale (**Match**) che racchiude tutte le altre principali (con anche esse che a loro volta possono essere composte da altre entità). In essa sarà presenta la logica principale della **simulazione** (main loop) con le proprietà che definiscono lo stato attuale della **simulazione** e che gestiscono il flusso di essa.

Successivamente ho suddiviso la **partita** nelle varie *sotto entità* principali:

- **Giocatore** (*Player*)
  - Contiene tutte le informazioni relative ad un singolo giocatore tra cui: **Posizione sul tabellone**, **Numero di turni per cui è bloccato**, **Carte in possesso**
  - In più è responsabile per il suo **spostamento**, gestione dello **stato di blocco** e uso delle **carte in possesso**
- **Tabellone** (*Board*)
  - Contiene la collezione delle **Celle**, altra entità che definisce le *azioni* che un **giocatore** deve eseguire quando vi si trova sopra
  - Ho deciso di implementarla come *classe assestante*, invece che semplice *collezione*, per permettere la definizione di funzioni specifiche aggiuntive in futuro
- **Dadi** (*Dice*)
  - Contiene tutte le informazioni relative al **lancio dei dadi** tra cui, oltre alla definizione dei dadi stessi (estremamente flessibile sia sul *numero di facce* che *numero di dadi*), anche le informazioni relative alle regole che **influenzano il lancio dei dadi**
- **Mazzo di carte** (*Deck*)
  - Anche essa definisce una collezione di **Carte**, oggetti definiti tramite il pattern **SINGLETON* *(implementati tramite *enumerator*) che rappresentano il tipo della carta
  - Anche essa come il **Tabellone** è come *classe assestante*, invece che semplice *collezione*, per poter implementare la gestione diretta delle carte come: **mescolamento delle carte alla creazione** e **pesca con scarto automatico** delle carte così dette *istantanee*
  - Le **Carte** in se non forniscono alcuna direttiva relativa alla'azione da eseguire, è la **Partita** a gestire le azioni che le carte definiscono

E.2 Gestione delle relazioni tra entità della simulazione (Mediator)

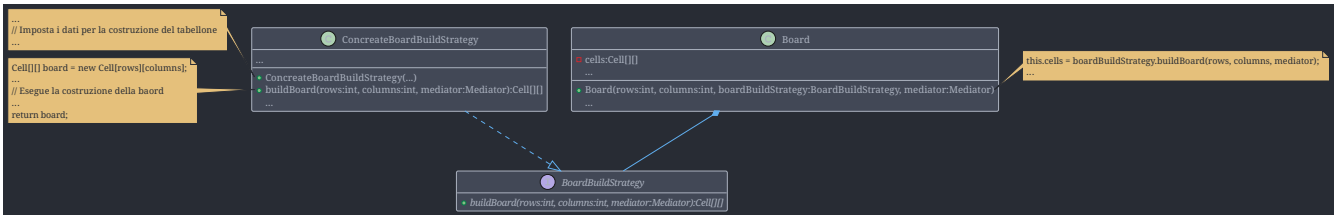
Per la gestione delle **interazioni** tra le entità di *pari livello* (cioè le *entità* che hanno lo stesso *padre*), per limitare al minimo le dipendenze, ho optato per l'uso del design pattern ***MEDIATOR***.



E.3 Costruzione del tabellone (Strategy)

Per la costruzione del **tabellone** ho scelto di usare il patter ***STRATEGY***, definendo un oggetto per la costruzione costruzione delle varie **celle** nelle varie posizioni (*matrice* di **celle**).

Il motivo di questa scelta è legata alla possibilità di poter definire, in maniera indipendente dal **tabellone**, vari modi di costruirlo (ad esempio, random, manuale o anche un misto di varie strategie) permettendo di modificare le due funzionalità in maniera indipendente.



E.4 Gestione del main loop della simulazione

Per quanto riguarda il **main loop** implementato nel metodo *run()* di **Match**, una soluzione potrebbe essere quella di usare il pattern ***STATE***, ma ho optato per un'implementazione classica (usando metodi richiamati a cascata) in quanto la logica generale *non dovrebbe cambiare in modo prevalente* in futuro e soprattutto l'implementazione richiederebbe molto *tempo* e *spezzetterebbe* il codice in maniera eccessiva.

E.5 Comunicazione GUI-Simulazione

Con l'implementazione della **GUI** e della **Simulazione** tramite thead indipendente, ho dovuto determinare un modo per far comunicare i due *componenti* e ho così individuato 3 principali strategie:

- Uso di **monitor** e passaggio di dati tramite *metodi* e *variabili*
  - Implementare dei metodi da usare per far comunicare il thread della **GUI** con quello della **Simulazione** sincronizzandoli appropriatamente
  - + Efficiente
  - - Implementazione complessa
  - - Rischio di introdurre bug
  - - Introduzione di dipendenze
- Uso di **task**
  - Particolari oggetti forniti dal framework **JavaFX** che permettono di eseguire operazioni in *background* per interagire con la **GUI**
  - + Strumento molto flessibile
  - + Sicuro ed efficiente
  - - Richiede una re-implementazione del main loop
- Uso di **Platform.runLater(...)**
  - Particolare classe fornita dal framework **JavaFX** che permette il passaggio di un *Runnable* di eseguire operazioni sulla **GUI** in maniera automatica
  - + Semplice e veloce da implementare
  - + Sicuro e rapido nell'esecuzione
  - - Non ottimale per operazioni lunghe e complesse

Per la gestione degli *eventi* legati alla **simulazione**, la mia scelta è ricaduta sull'ultima opzione, per via della *semplicità* d'uso e la possibilità di definire **singole funzioni** che eseguono *semplici operazioni* (simile a dei messaggi) in maniera disconnessa dal **main loop thread**, che può operare in modo indipendente (la classe **Platform** implementa un lista di richieste permettendo l'accumulo e lo smaltimento di esse in maniera automatica da parte della **GUI**).

Per la gestione dell'avanzamento *manuale* o *automatico* della **simulazione** ho invece optato per la prima opzione, in quanto l'uso di **Platform.runLater(...)** non garantisce l'istantaneità dell'esecuzione dell'azione, fondamentale per una ottima responsività della **GUI**.

F. Progettazione di Basso Livello

F.1 Simulazione

L'implementazione della simulazione si basa sull'esecuzione del thread **Match** che implementa nel metodo `run()` il *main loop* della partita. In questa classe sono definite tutte le condizioni e le operazioni da eseguire per permettere alla simulazione di procedere. Le operazioni sono gestite dalla classe **Matach** che interagisce con tutte le altre entità che la compongono e si pone da intermediario (***MEDIATOR***) tra di esse.

Ogni operazione è racchiusa in un metodo separato che, a cascata, chiama i metodi successivi per procedere con la simulazione.

Il *main loop* si compone da 3 fasi principali:

1. Individuazione del giocatore che deve prendere turno
    a. Questa fase è semplicemente gestita da un indice che viene incrementato ogni volta che un giocatore termina il proprio turno e quando anche l'ultimo giocatore ha terminato il turno questo viene riportato a 0, ripetendo le turnazioni
2. Esecuzione del turno
    a. Il turno è composto a sua volta da 4 fasi:
        i. Verifica se il giocatore è bloccato (è l'entità **Player** a tenere traccia di ciò)
        ii. Lanciare i dadi (eseguendo il metodo `roll(boolean isBoardLastNCells)` e passando come parametro la se è verificata la condizione della regola ***Dado Singolo***)
        iii. Movimento del giocatore (muovendo il **Player** e verificando che il giocatore non abbia vinto o sia uscito fuori dal tabellone, nel primo caso si ritorna terminando la partita, mentre nel secondo caso sposta il giocatore all'inizio del tabellone del movimento residuo)
        iv. Esecuzione dell'azione della cella (che tramite il patter ***MEDIATOR*** segnala all'oggetto **Match** di eseguire l'operazione richiesta sul giocatore corrente)
3. Terminazione del turno
    a. Si verificano le condizioni se il turno del giocatore è finito e deve eseguire un'altro turno
    b. E si verificano le condizioni se la partita è terminata con un vincitore

Nel *main loop* sono in più presenti diversi *interrupt check* in modo da terminare il loop non appena esso viene interrotto (per esempio dalla chiusura della **GUI**).

F.2 GUI

Le varie schermate della **GUI** sono state implementate tramite l'uso del software **Scene Builder** e per ognuna di esse è stato definito un `Controller`, una classe speciale che inizializza i vari componenti della scena e ne gestisce le interazioni tramite degli `EventListener`.

In più è il controller della schermata di setup a costruire ed avviare direttamente l'oggetto **Match**

relativo alla partita da simulare. In questo modo esso ne possiede un riferimento che può passare al controller relativo alla schermata di simulazione che può usare per comunicare con esso.

L'implementazione dei controller segue le linee guida descritte nella documentazione del framework.

G. Spiegare come il progetto soddisfa i requisiti funzionali (RF) e quelli non funzionali (RNF)

Riporta in questa sezione in che modo la progettazione architettonica e di basso livello prodotta soddisfa gli RF e gli RNF.

G.1 RF

1. *Impostazione della partita da simulare:*
    a. Interfaccia di setup
        i. Impostazione dimensione tabellone (Da un minimo di 3 ad un massimo di 15 per dimensione)
        ii. Impostazione posizione delle caselle (Possibilità di scegliere il **BoardStrategyBuilder**)
    b. Salvare/caricare setup (Salvataggio e caricamento dei setup tramite file *YAML*)
2. *Simulazione della partita:*
    a. Avanzamento automatico e manuale dei turni (Checkbox che permette di scegliere se abilitare l'avanzamento automatico, o se usare il tasto di avanzamento manuale che si disabilita in avanzamento automatico)
    b. Lancio dei dai
        i. Dadi doppi o singoli (Possibilità di scegliere tra 1 o 2 dadi)
        ii. Regola del dado singolo (Checkbox che si disabilita se è stato scelto 1 dado solo)
        iii. Regola del doppio sei (Checkbox che si disabilita se è stato scelto 1 dado solo)
    c. Regole di movimento delle pedine
        i. Caselle "Scale & Serpenti" (Checkbox)
        ii. Caselle "Sosta" (Checkbox)
        iii. Caselle "Premio" (Checkbox)
        iv. Caselle "Pesca una Carta" (Checkbox)
    d. Mazzo di carte
        i. Carte standard (Abilitata automaticamente se la Checkbox "Pesca una Carta" è attiva)
        ii. Carte extra (Checkbox che si disabilita se le carte sono disabilitate)
3. *Visualizzazione dei turni simulati:*
    a. Rappresentazione del tabellone (Tabellone rappresentato tramite griglia con numero cella e tipologia)
    b. Rappresentazione delle pedine sul tabellone (All'interno della cella tramite un numero colorato)
4. *Visualizzazione dei risultati della partita:*

a. Vincitore della partita (Popup a termine della simulazione)
b. Classifica finale (Possibilità di consulate il tabellone al temine della partita o tramite log)

G.2 RNF

1. *Interfaccia grafica* (**GUI**):
   a. Menu principale (Menu principale semplice per presentare il software)
   b. Interfaccia di simulazione (Schermata di simulazione interattiva)
2. *Regole:*
   a. Numero di dadi (Predisposizione per scelta di dadi superiore a 2)
   b. Tipologia dei dadi (Predisposizione per scelta di dadi con facce superiore a 2)
3. *Simulazione della partita:*
   a. Simulazione partite su thread separato (Implementazione della classe **Match** su thread)
4. *Termine simulazione:*
   a. Salvare log della simulazione (Uso del logger ***Logback***)

Appendice. Prototipo

Fornisci un breve rapporto sul tuo prototipo e in particolare: informazioni su ciò che hai implementato, come l'implementazione copre RF e RNF, come i prototipi dimostrano la correttezza del tuo progetto rispetto a RF e RNF. Puoi aggiungere alcuni screenshot per descrivere quanto richiesto sopra. Preparati a mostrare il tuo prototipo durante l'esame orale.

Il prototipo implementa quasi tutti i requisiti richiesti, la possibilità di impostare una partita con i vari parametri modificabili per abilitare o disabilitare le varie regole, assicurando la compatibilità di esse. Permette il salvataggio e caricamento su file delle impostazioni di una parita in formato YAML.

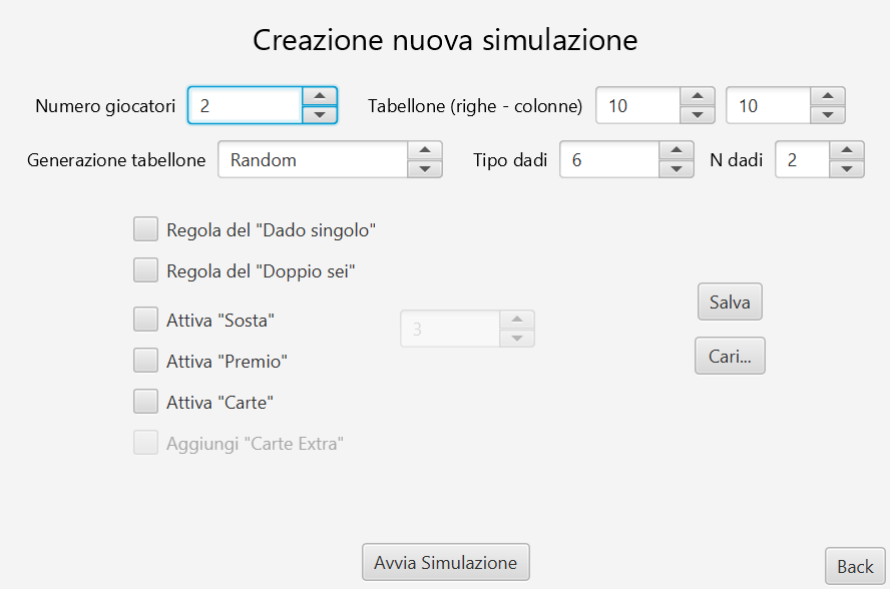


Figure 3. Menu di impostazioni della simulazione

Figure 4. Schermata di visualizzazione della partita avviata

La simulazione viene eseguita in completa autonomia, con la sola interazione richiesta dall'utente per la gestione dell'avanzamento dei turni che può essere automatico o manuale. Le varie e entità

che compongono la simulazione sono testate tramite l'uso di test unitari con l'ausilio della libreria **JUnit**. I test pur non essendo esaustivi (troppo complesso e superfluo) dimostrano la correttezza delle varie operazioni di cui ogni oggetto e responsabile, verificando tutti i componenti più critici del software. Fa eccezione il testing del *main loop* che essendo molto complesso (e principalmente basato su eventi casuali) richiederebbe un testing estremamente estensivo e complesso. La valutazione della suo correttezza e stata verificata manualmente esaminando i log, verificando che le varie operazioni vengono eseguite correttamente (stato coerente dei dati) e nell'ordine prestabilito.

La rappresentazione della simulazione sulla GUI può risultare grezza e poco rifinita ma svolge correttamente la sua funzione di visualizzazione dello stato attuale della partita, mentre i log generati durante la simulazione risultano più esaustivi mostrando lo storico completo delle operazioni eseguite e permettendo un'analisi più approfondita della simulazione stessa.

```
4071 [Thread-3] INFO  s.simulation.Match -- !Simulation started!
4075 [Thread-3] INFO  s.simulation.Match -- Round 1
5727 [JavaFX Application Thread] INFO  s.gui.ControllerMatch -- Auto run: ON
5729 [Thread-3] INFO  s.simulation.entities.Dice -- Dice: 6+4
5729 [Thread-3] INFO  s.simulation.Match -- Player 1 rolled: 10
5729 [Thread-3] INFO  s.simulation.Match -- Player 1 moved to: 10
5733 [Thread-3] INFO  s.simulation.Match -- Player 1 is on: [---]
6735 [Thread-3] INFO  s.simulation.entities.Dice -- Dice: 2+2
6735 [Thread-3] INFO  s.simulation.Match -- Player 2 rolled: 4
6735 [Thread-3] INFO  s.simulation.Match -- Player 2 moved to: 4
6735 [Thread-3] INFO  s.simulation.Match -- Player 2 is on: [---]
7738 [Thread-3] INFO  s.simulation.Match -- Round 2
7739 [Thread-3] INFO  s.simulation.entities.Dice -- Dice: 4+6
7739 [Thread-3] INFO  s.simulation.Match -- Player 1 rolled: 10
7739 [Thread-3] INFO  s.simulation.Match -- Player 1 moved to: 20
7739 [Thread-3] INFO  s.simulation.Match -- Player 1 is on: [---]
...
80895 [Thread-3] INFO  s.simulation.Match -- Player 3 rolled: 1
80895 [Thread-3] INFO  s.simulation.Match -- Player 3 moved to: 25
81496 [JavaFX Application Thread] INFO  s.gui.ControllerMatch -- Next turn (manual)
81897 [Thread-3] INFO  s.simulation.Match -- Player 3 has won!
```