# Prova Finale di Ingegneria del Software: Requisiti

*Gian Enrico Conti, Niccolò Izzo*

# Contents

# 1 Introduzione

Il progetto consiste nello sviluppo di una versione software del gioco da tavolo *Sagrada*.
Il progetto finale dovrà includere:

- diagramma UML iniziale dell'applicazione (ad alto livello)
- diagrammi UML finali che mostrino come è stato progettato il software, i diagrammi non dovranno essere generati a partire dal codice sorgente del progetto utilizzando tool automatici
- implementazione funzionante del gioco conforme alle regole del gioco e alle specifiche presenti in questo documento
- codice sorgente dell'implementazione
- codice sorgente dei test di unità

**Data di consegna: Domenica 08 Luglio 2018 23:59:59 CEST**
**Data di valutazione: 11,12,13 Luglio 2018**

Le modalità di valutazione, aule e orari, verranno comunicate successivamente.

# 2 Requisiti di Progetto

I requisiti del progetto si dividono in due gruppi:

I **Requisiti game-specific** che riguardano le regole e le meccaniche del gioco

II **Requisiti game-agnostic** che riguardano aspetti di design, tecnologici o implementativi

## 2.1 Requisiti Game-specific

Le regole del gioco sono descritte nei file `sagrada_regole_ITA.pdf`, caricati su BeeP. I nomi di classi, interfacce e variabili dovranno essere in lingua inglese.
Per la valutazione (vedi Tabella 1) si fa riferimento a due possibili set di regole: le regole semplificate e le regole complete.

- **Regole Semplificate**: Si considerano solo le **Carte Utensile** dalla numero 1 alla numero 6 inclusa.
- **Regole Complete**: Si considerano tutte le **Carte Utensile**, dalla numero 1 alla numero 12 inclusa.

Entrambi i set di regole fanno riferimento a tutte le meccaniche di gioco descritte nel manuale, eccetto quelle incluse nel paragrafo **Regole per il Gioco in Solitaria**, che costituiscono una funzionalità avanzata.
Ogni giocatore è identificato da un *nickname* che viene impostato lato client e deve essere univoco in ogni partita. L'univocità del *nickname* deve essere garantita dal server in fase di accettazione del giocatore.

## 2.2 Requisiti Game-agnostic

In questa sezione vengono presentati i requisiti tecnici dell'applicazione.
Il progetto consiste nell'implementazione di un **sistema distribuito** composto da un *singolo* **server** in grado di gestire una partita alla volta e *multipli* **client** (uno per giocatore) che possono partecipare ad una sola partita alla volta. Si richiede l'utilizzo del pattern **MVC** (Model-View-Controller) per progettare l'intero sistema.

#### 2.2.1 Server

Di seguito la lista dei requisiti tecnici per il lato server.

- Deve implementare le regole del gioco utilizzando *JavaSE*.
- Deve essere istanziato una sola volta.
- Deve gestire una partita alla volta.
- Nel caso in cui venga implementata la comunicazione client-server *sia* via socket *sia* via RMI, deve poter supportare partite in cui i giocatori utilizzano tecnologie diverse.

#### 2.2.2 Client

Di seguito la lista dei requisiti tecnici per il lato client.

- Deve essere implementato con *JavaSE* ed essere instanziabile piú volte (una per giocatore).
- L'interfaccia grafica deve essere implementata mediante JavaFX.
- Nel caso in cui venga implementata la comunicazione client-server *sia* via Socket *sia* via RMI, all'avvio deve permettere al giocatore di selezionare la tecnologia da utilizzare.
- Nel caso in cui venga implementata sia un'interfaccia testuale (CLI) che un'interfaccia grafica (GUI), all'avvio, deve permettere al giocatore di selezionare il tipo di interfaccia da utilizzare.

#### 2.2.3 Avvio della partita

Si assume che ogni giocatore che voglia partecipare ad una partita conosca l'indirizzo IP o lo URL del server. Quando un giocatore si connette:

- Se non ci sono partite in fase di avvio, viene creata una nuova partita, altrimenti l'utente entra automaticamente a far parte della partita in fase di avvio.
- Se c'è una partita in fase di avvio, il giocatore viene automaticamente aggiunto alla partita.
- La partita inizia non appena si raggiungono i 4 giocatori. Quando 2 giocatori si connettono a una partita viene inizializzato un timer di N secondi, caricato da un file di configurazione presente lato server o specificato tramite parametri *command line*. Se non si raggiungono 4 giocatori entro il timeout, la partita inizia comunque con il numero di giocatori raggiunto, a patto che questo sia >= 2. Se prima del timeout il numero di giocatori in attesa scende sotto i due, il timer viene resettato.

#### 2.2.4 Corso della partita

Il server consente ai vari giocatori di svolgere i propri turni secondo le regole del gioco. È necessario gestire sia il caso in cui i giocatori escano dalla partita, sia il caso in cui cada la connessione di rete.

- Ogni giocatore ha un tempo predefinito per eseguire le mosse (caricato da file di configurazione o parametro *command line*). Il server attende per il periodo di cui sopra, dopo il quale sospende il giocatore, i.e. il giocatore non esegue mosse ma viene comunque considerato nel conteggio dei punti.
- Tutti i giocatori vengono notificati della mancanza di un giocatore
- Il gioco continua, saltando i giri del giocatore sospeso
- Il giocatore può riconnettersi e continuare il gioco.

Se in ogni momento dovesse rimanere un solo giocatore in una partita in corso,la partita termina con la vittoria di quel giocatore.

#### 2.2.5 Funzionalità Avanzate

Le funzionalità avanzate sono requisiti **facoltativi** da implementare al fine di incrementare il punteggio in fase di valutazione. Si noti che queste funzionalità vengono valutate solo se i requisiti game-specific e game-agonistic presentati in Sezione 2 sono stati implementati in maniera sufficiente (vedi Sezione 3). I requisiti avanzati da implementabili sono:

- **Single Player**: Questa funzionalità avanzata consiste nell'implementazione delle regole descritte nel paragrafo **Regole per il Gioco in Solitaria** del manuale di gioco. Questa modalità di gioco sarà selezionabile dai *client* all'avvio, durante la selezione delle opzioni del client.
- **Persistenza**: Realizzare un sistema di gestione degli utenti che permetta ai giocatori di salvare i loro progressi sul server corrente. Questo sistema deve conservare per ciascuno le statistiche di gioco (numero di vittorie, tempo di gioco, numero di sconfitte, etc.) e tenere traccia della classifica globale di tutti i giocatori ordinata per numero di vittorie. Queste statistiche devono essere visualizzate alla fine di ogni partita. Inoltre, per ogni parita si desidera memorizzare, in un frame laterale, la sequenza di mosse effettuate dai vari giocatori in ordine di occorrenza, in modo che sia possibile riprodurre una partita già giocata.
- **Carte Schema Dinamiche**: Realizzare un sistema di caricamento da file di mappe di gioco personalizzate. I client devono fare *rendering* di queste mappe personalizzate in maniera vettoriale.
- **Partite Multiple**: Realizzare il *server* in modo che posssa gestire più partite **contemporaneamente**, dopo la procedura di creazione della prima partita, i giocatori che accederanno al server verranno gestiti in una *sala d'attesa* per creare una seconda partita e così via.

## 3 Valutazione

In Tabella 1 sono riportati i punteggi **massimi** ottenibili in base ai requisiti implementati.

| **Requisiti Soddisfatti** | **Voto Massimo** |
|---|---|
| Regole Semplificate + CLI + RMI | 18 |
| Regole Complete + CLI + RMI | 20 |
| Regole Complete + CLI + Socket | 21 |
| Regole Complete + CLI + Socket + 1 F.A. | 22 |
| Regole Complete + GUI + (RMI o Socket) + 1 F.A. | 24 |
| Regole Complete + GUI + RMI + Socket + 1 F.A. | 27 |
| Regole Complete + CLI + GUI + RMI + Socket + 1 F.A. | 30 |
| Regole Complete + CLI + GUI + RMI + Socket + 2 F.A. | 30L |

Table 1: Tabella di valutazione

Di seguito si riportano i criteri di valutazione:

- La qualità della *progettazione* con particolare riferimento ad un uso appropriato di interfacce, ereditarietà, composizione tra classi, uso dei design pattern (statici, di comunicazione e architetturali) e divisione delle responsabilità.
- La *stabilità* dell'implementazione e la *conformità alle specifiche*.
- La *leggibilità* del codice scritto, con particolare riferimento a nomi di variabili/metodi/classi/package, all'inserimento di commenti in inglese e documentazione JavaDoc in inglese, la mancanza di codice ripetuto e metodi di eccessiva lunghezza.

- L'*efficacia* e la *copertura* dei casi di test, il nome e i commenti di ogni test dovranno chiaramente specificare le funzionalità testate e i componenti coinvolti.
- *L'utilizzo* degli strumenti (IntelliJ IDEA, Git, Maven, ...).
- *L'autonomia*, *l'impegno* e la *comunicazione* (con i responsabili e all'interno del gruppo) durante tutte le fasi del progetto.