

**Prova Finale di Ingegneria Del Software**

**Anno Accademico 2021-2022**

**Requisiti**

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Requisiti di Progetto</b>	<b>3</b>
2.1	Requisiti Game-specific . . . . .	3
2.2	Requisiti Game-agnostic . . . . .	4
2.2.1	Server . . . . .	4
2.2.2	Client . . . . .	4
2.3	Funzionalità Avanzate . . . . .	4
<b>3</b>	<b>Valutazione</b>	<b>5</b>

## 1 Introduzione

Il progetto consiste nello sviluppo di una versione software del gioco da tavolo *Eriantys*.

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 potranno 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;
- documentazione del protocollo di comunicazione tra client e server;
- documenti di peer review (uno relativo alla prima peer review e uno relativo alla seconda);
- codice sorgente dell'implementazione;
- codice sorgente dei test di unità.

**Data di consegna: venerdì 01 luglio 2022 23:59:59 CEST**

**Data di valutazione: da fissare (a partire da lunedì 4 Luglio 2022)**

Modalità di valutazione, aule e orari, verranno comunicati 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 nel file `eriantys-rules.pdf`, caricato su WeBeeP. In caso di discordanza tra tale file ed eventuali traduzioni in altre lingue, si faccia riferimento alla versione italiana.

I nomi di classi, interfacce, le variabili, ed in generale tutti gli identificativi nel codice dovranno essere in lingua inglese. Sempre in inglese dovranno essere anche i commenti nel codice e la documentazione tecnica (JavaDoc). La lingua dell'interfaccia utente dell'applicazione può essere scelta liberamente tra inglese e italiano.

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 considerino solo le regole per lo svolgimento di partite da 2 o da 3 giocatori; è esclusa la variante per esperti (e quindi sono escluse le carte personaggio).
- **Regole Complete:** si considerino le regole per lo svolgimento di partite da 2 o da 3 giocatori, includendo la variante per esperti (si richiede l'implementazione di almeno 8 diverse carte personaggio, a scelta del gruppo).

Ogni giocatore è identificato da un *nickname* che viene impostato lato client. Tale nickname deve essere univoco, nel senso che non possono esserci due utenti collegati contemporaneamente con lo stesso *nickname*. 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 al fine di gestire una singola partita (o più partite nel caso in cui venga implementata la funzionalità avanzata "partite multiple").
- Consente ai vari giocatori di svolgere i propri turni tramite le istanze del lato client, connesse via socket TCP-IP.

### 2.2.2 Client

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

- Deve essere implementato con *JavaSE* ed essere istanziabile più volte (una per giocatore), anche sulla stessa macchina.
- L'interfaccia grafica deve essere implementata mediante Swing o JavaFX.
- 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.

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.
- Il giocatore che crea la partita sceglie il numero di giocatori che ne fanno parte.
- Se c'è una partita in fase di avvio, il giocatore viene automaticamente aggiunto alla partita.
- La partita inizia non appena si raggiunge il numero di giocatori atteso (in base alla scelta effettuata dal primo giocatore in fase di creazione della partita).

È necessario gestire sia il caso in cui i giocatori escano dalla partita, sia il caso in cui cada la connessione di rete. In entrambi i casi la partita dovrà terminare (anche se in fase di avvio) e tutti i giocatori verranno notificati.

## 2.3 Funzionalità Avanzate

Le funzionalità avanzate sono requisiti **facoltativi** da implementare al fine di incrementare il punteggio in fase di valutazione. Queste funzionalità vengono valutate solo se i requisiti game-specific e game-agnostic presentati nelle sezioni precedenti sono stati implementati in maniera sufficiente. Le funzionalità avanzate implementabili sono:

- **Carte personaggio:** Implementare tutte e 12 le carte personaggio.
- **Partita a 4 giocatori:** Realizzare il progetto in modo che sia possibile giocare partite da 2, 3 oppure 4 giocatori.
- **Partite multiple:** Realizzare il *server* in modo che possa gestire più partite **contemporaneamente**. Ai fini dell'implementazione di questa funzionalità aggiuntiva, le regole precedentemente specificate in merito alla creazione delle partite possono essere modificate in base alle esigenze implementative o di interfaccia utente.
- **Persistenza:** Lo stato di una partita deve essere salvato su disco, in modo che la partita possa riprendere anche a seguito dell'interruzione dell'esecuzione del server. Per riprendere una partita, i giocatori si devono ricollegare al server utilizzando gli stessi *nickname* una volta che questo sia tornato attivo. Si assume che il server non interrompa la propria esecuzione durante il salvataggio su disco e che il disco costituisca una memoria totalmente affidabile.
- **Resilienza alle disconnessioni:** I giocatori disconnessi possono ricollegarsi in seguito e continuare la partita. Mentre un giocatore non è collegato, il gioco continua saltando i turni di quel giocatore. Se rimane attivo un solo giocatore, il gioco viene sospeso fino a che non si ricollega almeno un altro giocatore oppure scade un timeout che decreta la vittoria dell'unico giocatore rimasto connesso.

### 3 Valutazione

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

Requisiti Soddisfatti	Voto Massimo
Regole Semplificate + CLI + Socket	18
Regole Complete + CLI + Socket	21
Regole Complete + CLI + GUI + Socket	24
Regole Complete + CLI + GUI + Socket + 1 FA	27
Regole Complete + CLI + GUI + Socket + 2 FA	30
Regole Complete + CLI + GUI + Socket + 3 FA	30L

Tabella 1: Tabella di valutazione (FA=Funzionalità avanzata)

Di seguito si riportano gli aspetti che vengono valutati e contribuiscono alla definizione del punteggio finale:

- 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, la mancanza di codice ripetuto e metodi di eccessiva lunghezza.
- La *qualità della documentazione*, con riferimento ai commenti JavaDoc in inglese presenti nel codice, alla documentazione aggiuntiva riguardante il protocollo di comunicazione, e ai documenti delle peer review.
- 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.