

# RisiKOOP

Il gioco strategico per la conquista del mondo

Matteo Caruso, Matteo Ceccarelli, Franceso Sacripante

26 luglio 2025

# Indice

<b>1</b>	<b>Analisi</b>	<b>2</b>
1.1	Descrizione e requisiti . . . . .	2
1.2	Modello del Dominio . . . . .	3
<b>2</b>	<b>Design</b>	<b>5</b>
2.1	Architettura . . . . .	5
2.2	Design dettagliato . . . . .	5
2.2.1	Matteo Caruso . . . . .	5
2.2.2	Matteo Ceccarelli . . . . .	7
2.2.3	Francesco Sacripante . . . . .	7
<b>3</b>	<b>Sviluppo</b>	<b>11</b>
3.1	Testing automatizzato . . . . .	11
3.2	Note di sviluppo . . . . .	11
<b>4</b>	<b>Commenti finali</b>	<b>12</b>
4.1	Autovalutazione e lavori futuri . . . . .	12
4.2	Difficoltà incontrate e commenti per i docenti . . . . .	12
<b>A</b>	<b>Guida utente</b>	<b>13</b>
A.1	Avviare la partita . . . . .	13
A.2	Rinforzi iniziali . . . . .	13
A.2.1	Giocare le combo . . . . .	13
A.3	Attaccare . . . . .	13
A.4	Spostamento finale . . . . .	13
<b>A</b>	<b>Esercitazioni di laboratorio</b>	<b>14</b>
A.1	matteo.caruso7@studio.unibo.it . . . . .	14
A.2	matteo.ceccarelli@studio.unibo.it . . . . .	14
A.3	francesco.sacripante@studio.unibo.it . . . . .	14

# Capitolo 1

## Analisi

### 1.1 Descrizione e requisiti

Il software mira a replicare il gioco Risiko, un gioco da tavolo di strategia a turni dove ogni giocatore controlla una squadra di unità allo scopo di completare un obiettivo determinato da una Carta Obiettivo pescata a inizio partita. Questa richiederà di conquistare dei continenti, annientare un'altra armata oppure conquistare un certo numero di territori. Il gioco inizia spartendo tutti i territori tra i giocatori e dà dei territori iniziali con cui rinforzarli. Ogni turno, il giocatore otterrà vari carri armati da posizionare sui suoi territori. Potrà poi attaccare territori adiacenti ai propri. Se riesce a conquistare almeno uno stato otterrà una Carta Territorio, utilizzabile per giocare combo al fine di ottenere ulteriori unità nei successivi turni. Infine avrà l'opportunità di spostare delle unità fra i suoi territori.

#### Tipi di Combo

Le combo sono sempre tris di carte territorio, ognuna ricompensa un certo numero di unità:

- 3 cannoni: 4 unità.
- 3 fanti: 6 unità.
- 3 cavalieri: 8 unità.
- Un fante, un cannone e un cavaliere: 10 unità. <sup>1</sup>
- Un Jolly e due carte uguali: 12 unità.

---

<sup>1</sup>Non è possibile sostituire una delle carte con un Jolly in questa combo.

### **Requisiti funzionali**

- Il software dovrà permettere di giocare a una semplice versione di Risiko.
- Ogni giocatore ha una sua Carta Obiettivo e varie Carte Territorio.
- L'attacco avviene tramite il tiro di dadi, il cui confronto ne determinerà l'esito.

### **Requisiti non funzionali**

- La mappa è selezionabile, scelta dai giocatori a inizio partita.
- I giocatori dovranno poter nascondere le proprie Carte Obiettivo e Territorio agli altri giocatori.

## **1.2 Modello del Dominio**

Il gioco inizia con la selezione dei giocatori, del loro colore e della mappa. Successivamente vengono assegnati i territori, ed è chiesto ai giocatori di posizionare le loro unità rimanenti in quei territori. Ora inizia il game-loop del gioco, che si ripete fino a quando un giocatore non vince:

- Fase di rinforzo.
- Fase di attacco.
- Fase di spostamento finale.

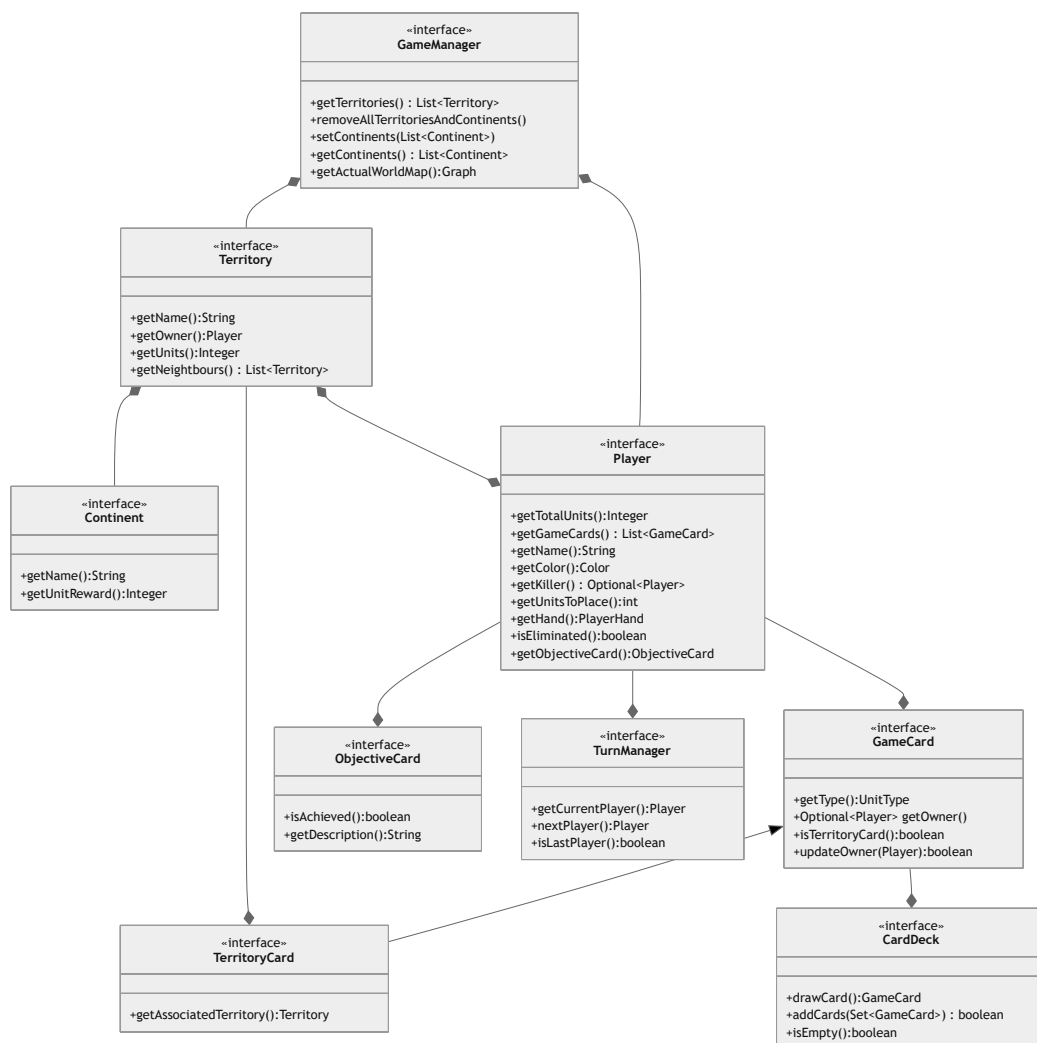


Figura 1.1: UML del modello svolto dopo l'analisi dei requisiti.

# Capitolo 2

## Design

### 2.1 Architettura

L'architettura del software è basata su un pattern Model-View-Controller (MVC). L'entry point dell'applicazione è il **Controller**, che si occupa di avviare il model, che implementa **GameManager**, e le view registrate, che implementano **RisikoView**. La separazione permette di aggiungere facilmente altre **RisikoView** se necessario.

Le fasi di gioco sono state modellate con una *State Machine*, un paradigma di programmazione che permette di dividere il sistema in varie sotto-fasi, detti anche *stati*. Ogni fase ha la propria logica diversa da quella di tutte le altre. Nel programma si riconoscono perché implementano **GamePhase**. Questo paradigma favorisce il *Single Responsibility Principle*, siccome ogni fase è responsabile internamente della propria gestione. Un'altro vantaggio risiede nella chiarezza con cui descrive in che punto dell'esecuzione si trova l'applicazione, siccome può essere in una sola fase.

### 2.2 Design dettagliato

#### 2.2.1 Matteo Caruso

##### Validare le combo di carte

**Problema** Bisogna validare vari tipi di combo di carte, ognuna con requisiti diversi. Inoltre ogni combo ricompensa il giocatore con un numero di unità diverso.

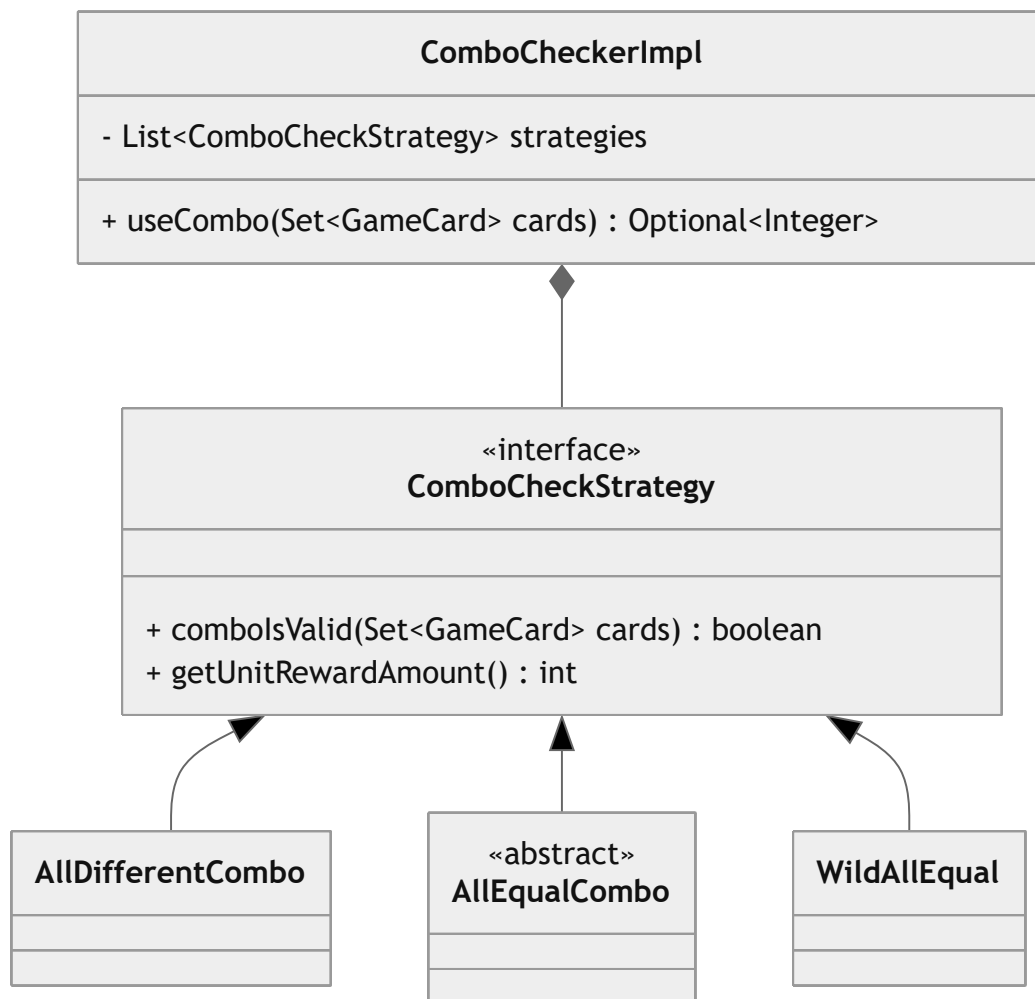


Figura 2.1: UML del pattern Strategy per la validazione delle combo.

**Soluzione** La validazione delle combo usa il pattern *Strategy*, in cui ogni validatore di combo è una strategia diversa.

Il pattern *Strategy* è più adatto rispetto al pattern *Template Method*, siccome ogni validatore di combo differisce molto dagli altri, fatta eccezione dei validatori *AllEqualCombo*.

#### Validare le *AllEqualCombo*

**Problema** Bisogna validare le combo di carte dove hanno tutte lo stesso tipo.

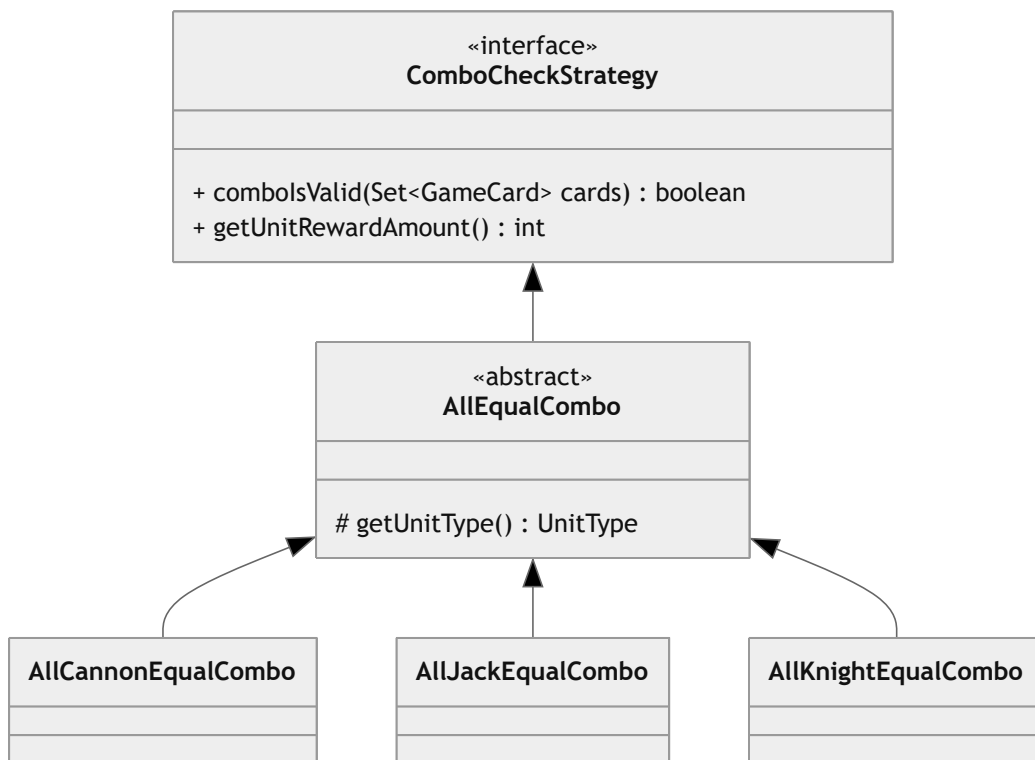


Figura 2.2: UML del pattern Template Method per la validazione delle combo di carte con tipo uguale.

**Soluzione** Qui è possibile usare il pattern *Template Method*, dove la classe astratta `AllEqualCombo` definisce il template method `comboIsValid` e l'operazione primitiva `getUnitType` <sup>1</sup>.

Gli lascia anche la responsabilità di implementare `getUnitRewardAmount`. Le classi che estendono questa classe astratta sono `AllCannonEqualCombo`, `AllJackEqualCombo` e `AllKnightEqualCombo`, che implementano l'operazione primitiva sopracitata.

## 2.2.2 Matteo Ceccarelli

## 2.2.3 Francesco Sacripante

### Creazione della logica delle prime due fasi

**Problema** Le prime due fasi hanno due logiche diverse tra loro e tra il resto del gioco

<sup>1</sup>Restituisce `UnitType`, un enumeratore che rappresenta i semi delle carte.



**Soluzione** Scomporre il controller in diversi dipi di controller.

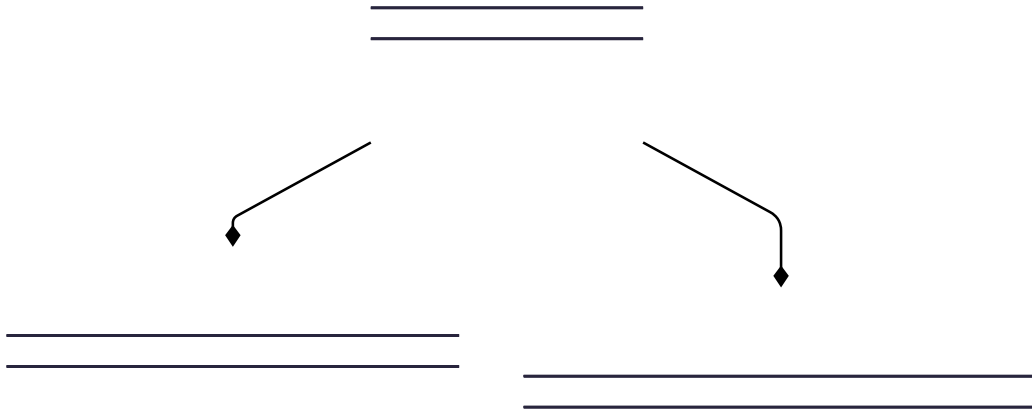


Figura 2.3: UML dei vari controller per le prime due fasi di gioco.

Risiko è un software molto legato alla visualizzazione del gioco, quindi per favorire il *Separation of Concerns*, il controller è diviso in sotto-controller: `DataAddingContrller` permette di impostare giocatori e la mappa, mentre `DataRetrieveController` favorisce l'ottenimento di informazioni quali il giocatore corrente. Infine `TurnManager` permette di gestire le fasi del gioco.

### **Aggiornamento dei colori dei territori della view**

**Problema** Durante il gioco, se un giocatore conquista un territorio, questo deve istantaneamente cambiare colore nel colore del nuovo proprietario.

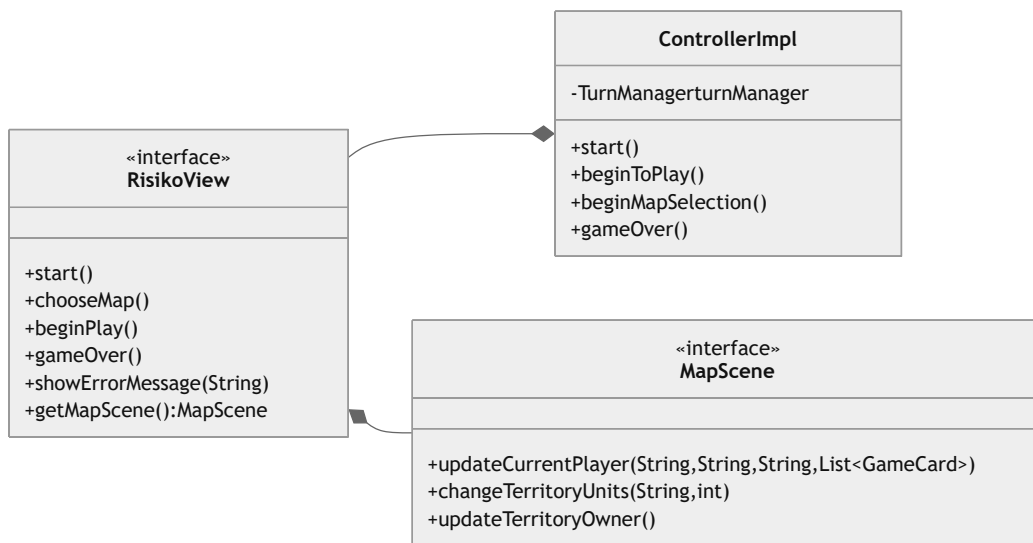


Figura 2.4: UML del observer pattern usato per aggiornare gli elementi delle view.

**Soluzione** Usare un'interfaccia dedicata agli aggiornamenti della view. Ogni volta che al model succede una cosa, il controller chiama i metodi dell'interfaccia per aggiornare la view. Il pattern usato qui è una sorta di observer pattern, dove l'observer è la view che viene "notificata" quando un territorio cambia proprietà e il soggetto è il territorio.

### Accorpamento di detentori di oggetti

**Problema** Alcuni oggetti condividono la detenzione ed alcuni tipi di operazioni su alcuni tipi di oggetti posseduti. Ci sono oggetti che detengono più territori o più giocatori.

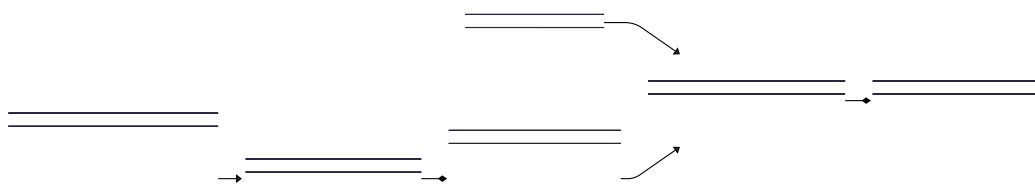


Figura 2.5: UML degli holders del gioco.

**Soluzione** Si creano delle interfacce in cui si accorpano tutte le operazione in comune riguardante un certo insieme di oggetti.

# Capitolo 3

## Sviluppo

### 3.1 Testing automatizzato

Il testing automatizzato è stato realizzato tramite JUnit, focalizzato principalmente sul model, come l'inserimento della mappa, la gestione dei giocatori, la validazione delle combo di carte, e sulla gestione delle fasi di gioco. L'interfaccia grafica è stata testata manualmente durante lo sviluppo del software.

### 3.2 Note di sviluppo

## Capitolo 4

### Commenti finali

4.1 Autovalutazione e lavori futuri

4.2 Difficoltà incontrate e commenti per i docenti

# Appendice A

## Guida utente

A.1 Avviare la partita

A.2 Rinfori iniziali

A.2.1 Giocare le combo

A.3 Attaccare

A.4 Spostamento finale

# Appendice A

## Esercitazioni di laboratorio

### A.1 `matteo.caruso7@studio.unibo.it`

- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=178723#p247198>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=179154#p247764>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=180101#p248784>
- Laboratorio 11: <https://virtuale.unibo.it/mod/forum/discuss.php?d=181206#p250854>

### A.2 `matteo.ceccarelli@studio.unibo.it`

- Laboratorio XX: <https://virtuale.unibo.it>

### A.3 `franceso.sacripante@studio.unibo.it`

- Laboratorio XX: <https://virtuale.unibo.it>