

# RisikOOP

Il gioco strategico per la conquista del mondo

Matteo Caruso, Matteo Ceccarelli, Franceso Sacripante

29 luglio 2025

# Indice

<b>1</b>	<b>Analisi</b>	<b>3</b>
1.1	Descrizione e requisiti . . . . .	3
1.2	Modello del dominio . . . . .	4
<b>2</b>	<b>Design</b>	<b>6</b>
2.1	Architettura . . . . .	6
2.2	Design dettagliato . . . . .	8
2.2.1	Matteo Caruso . . . . .	8
2.2.2	Matteo Ceccarelli . . . . .	10
2.2.3	Francesco Sacripante . . . . .	12
<b>3</b>	<b>Sviluppo</b>	<b>15</b>
3.1	Testing automatizzato . . . . .	15
3.2	Note di sviluppo . . . . .	15
3.2.1	Matteo Caruso . . . . .	15
3.2.2	Matteo Ceccarelli . . . . .	16
3.2.3	Francesco Sacripante . . . . .	16
<b>4</b>	<b>Commenti finali</b>	<b>18</b>
4.1	Autovalutazione e lavori futuri . . . . .	18
4.1.1	Matteo Caruso . . . . .	18
4.1.2	Matteo Ceccarelli . . . . .	18
4.1.3	Francesco Sacripante . . . . .	19
<b>A</b>	<b>Guida utente</b>	<b>20</b>
A.1	Avviare la partita . . . . .	20
A.1.1	Inserire i giocatori . . . . .	20
A.1.2	Scegliere la mappa . . . . .	21
A.2	Rinforzi iniziali . . . . .	22
A.3	Schermata di gioco . . . . .	23
A.3.1	Tipi di obiettivi . . . . .	24

A.4	Fase di rinforzi . . . . .	24
A.4.1	Giocare le combo . . . . .	25
A.4.2	Aggiungere unità . . . . .	26
A.5	Fase di attacco . . . . .	27
A.6	Fase di spostamento strategico . . . . .	29
A.7	Vittoria . . . . .	30
<b>B</b>	<b>Esercitazioni di laboratorio</b>	<b>32</b>
B.1	matteo.caruso7@studio.unibo.it . . . . .	32
B.2	franceso.sacripante@studio.unibo.it . . . . .	32

# 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 ad 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 delle unità con cui rinforzarli. All'inizio di ogni turno, il giocatore otterrà varie unità da posizionare sui suoi territori. Potrà poi attaccare territori adiacenti ai propri. Se riesce a conquistare almeno un territorio 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. La mappa contiene vari territori e i collegamenti tra di essi. Ogni territorio, a cui è associato una Carta Territorio, fa parte di un continente. Vengono assegnati i territori ad ogni giocatore, 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, completando il compito descritto dalla sua Carta Obiettivo:

- Fase di rinforzo.
- Fase di attacco.
- Fase di spostamento strategico.

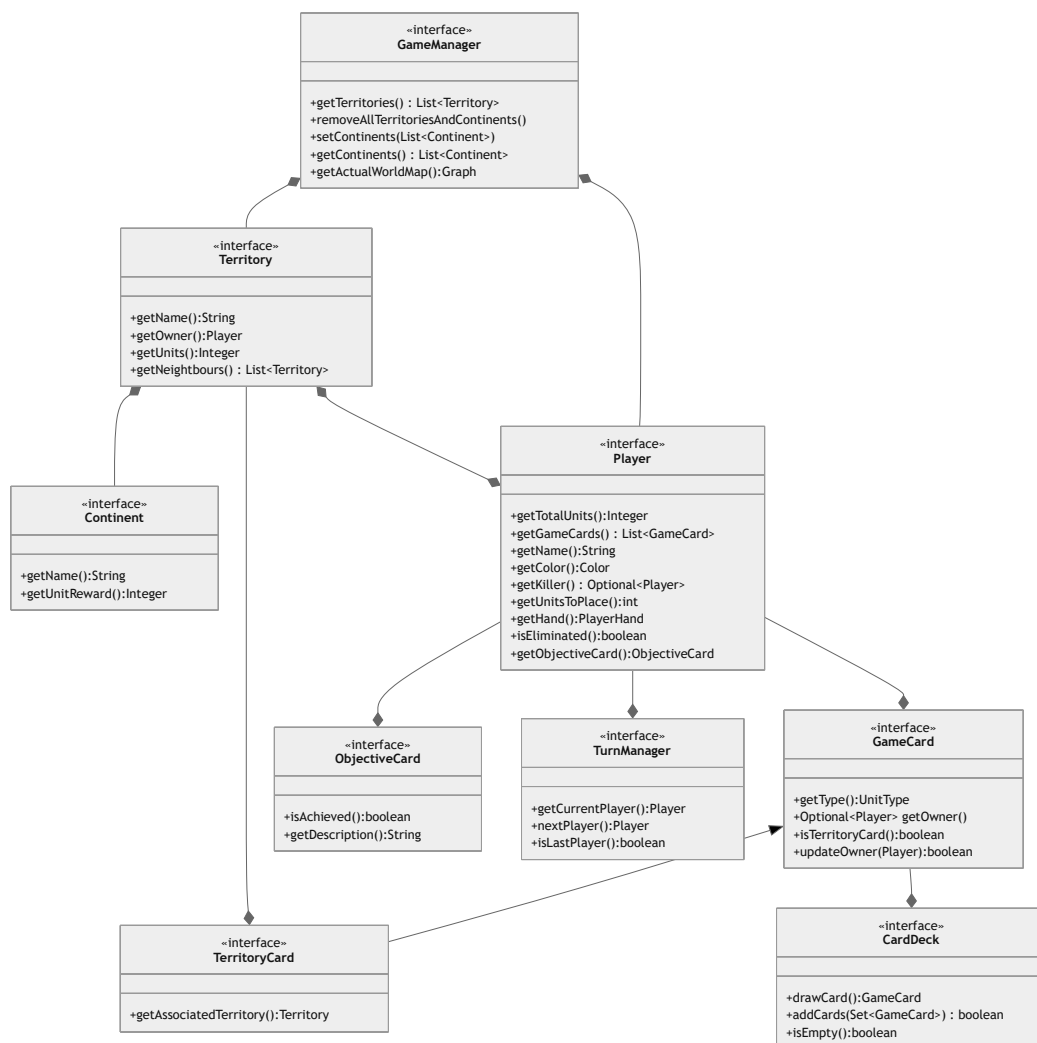


Figura 1.1: UML del modello svolto dopo l'analisi dei requisiti.

# 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, senza compromettere la logica del model e del controller.

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 della propria gestione. Un'altro vantaggio risiede nella chiarezza con cui descrive in che momento dell'esecuzione si trova l'applicazione, in quanto può esservi una sola fase alla volta.

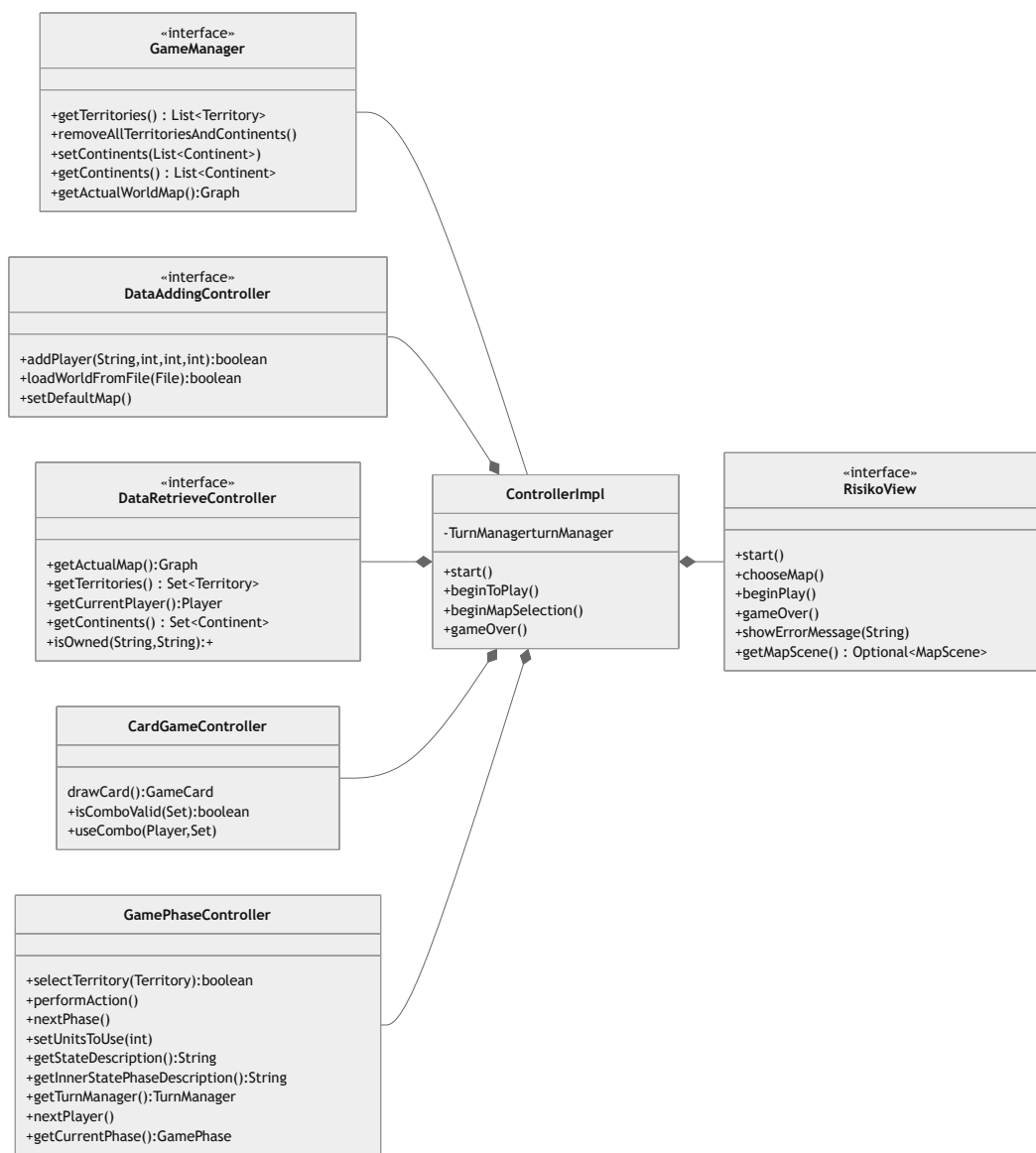


Figura 2.1: UML degli entry-point dell'architettura MVC.



## 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.

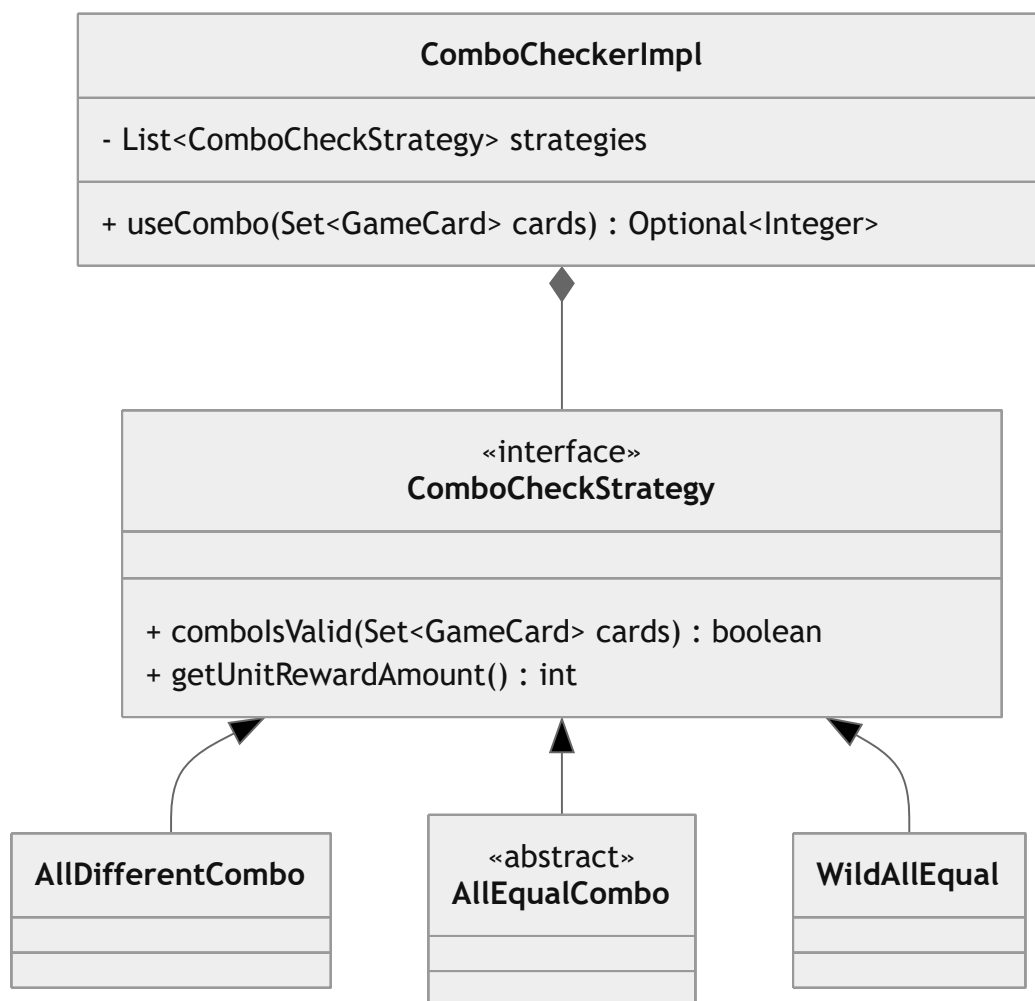


Figura 2.2: UML del pattern Strategy per la validazione delle combo.

**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*, poiché ogni validatore di combo differisce molto dagli altri, fatta eccezione dei validatori `AllEqualCombo`.

### Validare le `AllEqualCombo`

**Problema** Bisogna validare le combo di carte con stesso seme.

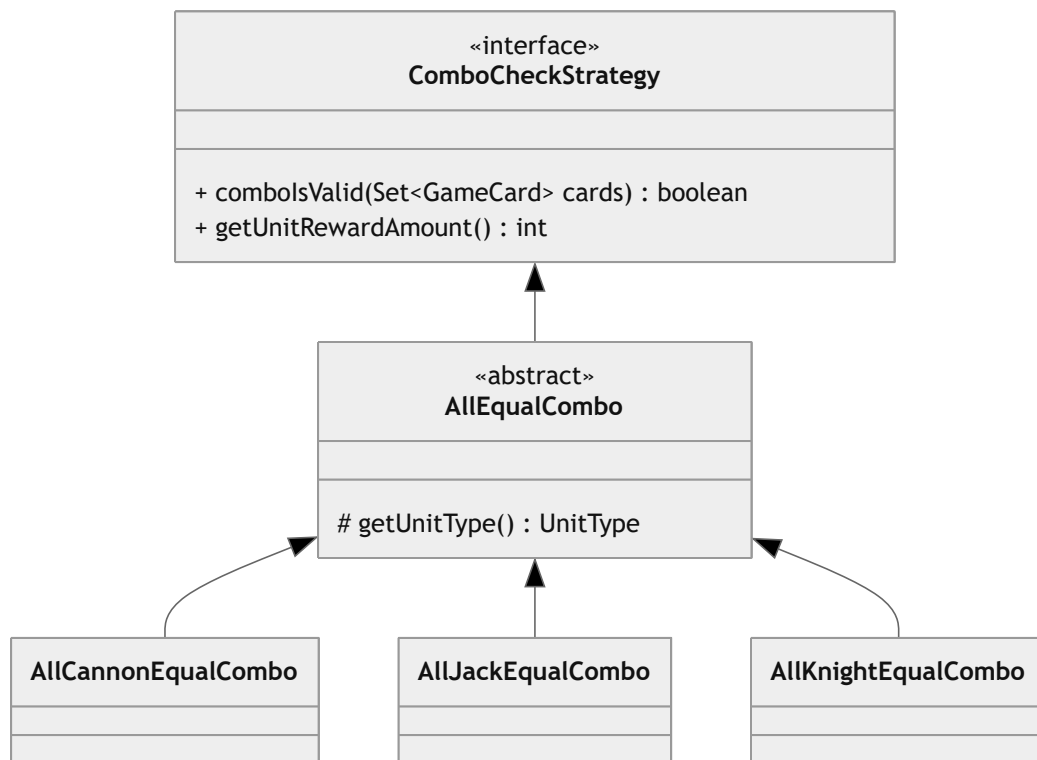


Figura 2.3: UML del pattern *Template Method* per la validazione delle combo di carte con tipo uguale.

**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.

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

## 2.2.2 Matteo Ceccarelli

### Gestione del mazzo di carte obiettivo

**Problema** Non essendo la mappa fissa, è necessario realizzare delle carte obiettivo che si adattassero dinamicamente alla configurazione della mappa selezionata.

**Soluzione** Ho usato il pattern *Template method*.

Creo una classe astratta `AbstractObjectiveCardBuilder` che incorpora la logica di creazione delle carte esponendo i metodi astratti primitivi `buildDescription` e `buildSpecification`, che generano le carte mediante il metodo template `createCard`.

Ho preferito l'utilizzo del *Template Method* rispetto al *Pattern Strategy* perché mi permetteva di isolare meglio gli elementi che differiscono tra le varie tipologie, isolando meglio le classi che ereditano da `AbstractObjectiveCardBuilder`.

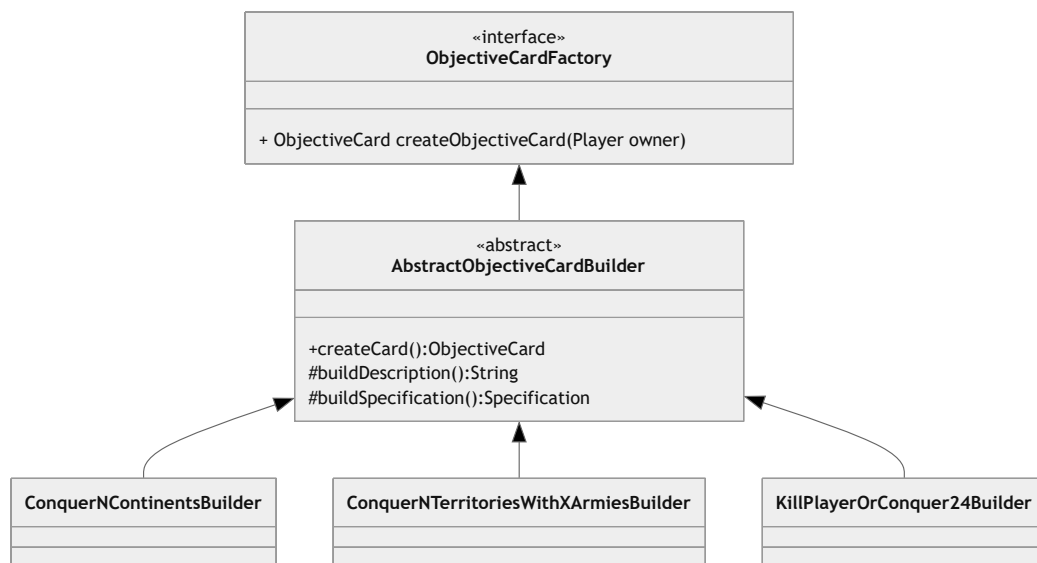


Figura 2.4: UML del pattern Template Method per la gestione delle carte obiettivo.

### Composizione delle condizioni di vittoria

**Problema** Avere un modo semplice, espressivo e componibile per definire e controllare la condizione di vittoria delle Carte Obiettivo, che spesso condividono aspetti di logiche comuni, come ad esempio conquista  $x$  territori o conquista almeno  $n$  territori con almeno  $m$  truppe.

**Soluzione** Lo *Specification Pattern* mi permette di rispondere a questa esigenza e, mediante la combinazione logica di tante piccole condizioni atomiche, posso realizzare espressioni logiche più o meno complesse. La parte centrale del pattern risiede nell'interfaccia funzionale **Specification** che espone i metodi **and**, **or**, **not** responsabili della composizione delle varie specifiche. I vantaggi introdotti da questo pattern mi hanno portato a preferire questa soluzione, all'alternative di *Strategy* che non si prestava al riutilizzo di logiche comuni, come ad esempio conquista  $n$  territori. La scelta dello *Specification Pattern* si è rivelata dunque la più adatta per astrarre la logica di verifica dell'obiettivo dalla costruzione della carta, massimizzando il riuso e mantenendo il design pulito e modulare.

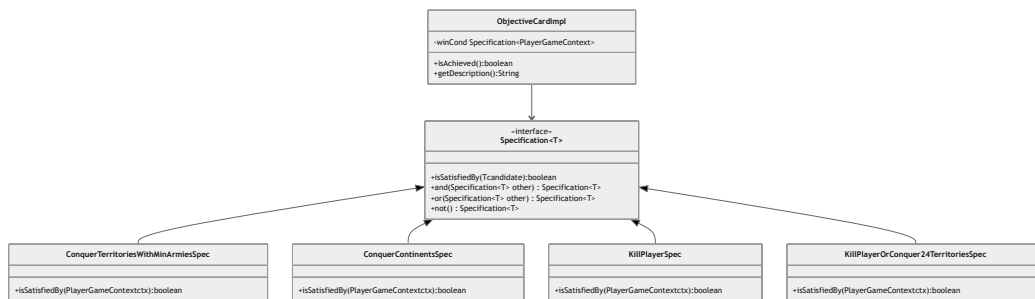


Figura 2.5: UML dello Specification Pattern per la gestione delle condizioni di vittoria.

## Gestione delle fasi di gioco

**Problema** RisikOOP è un gioco a turni in cui, in base alla fase corrente, gli stessi eventi della GUI (ad es. **selectTerritory** o **performAction**) devono produrre comportamenti diversi.

**Soluzione** Si è adottato il pattern *Strategy*, modellando ogni fase di gioco come una strategia indipendente, aderente all'interfaccia base **GamePhase**. In più, dato che non tutti gli eventi interessano ogni fase del gioco, si è scelto di creare tante piccole interfacce ognuna responsabile di incapsulare la logica di un'azione specifica. Così facendo ogni fase implementa unicamente le interfacce delle azioni corrispondenti.

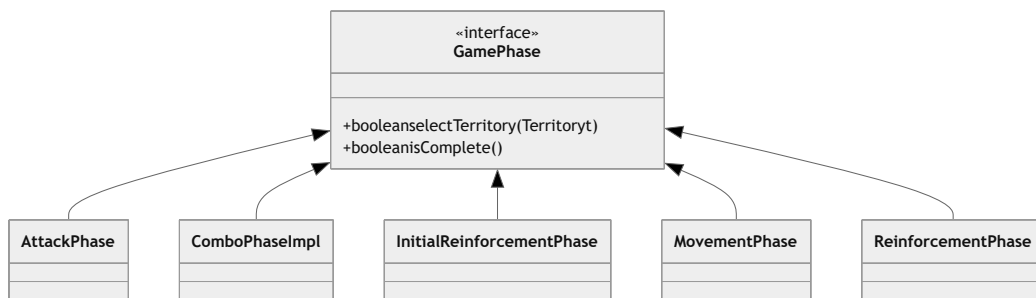


Figura 2.6: UML del pattern Strategy per la gestione delle fasi di gioco.

### 2.2.3 Francesco Sacripante

#### Creazione della logica delle prime due fasi

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

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

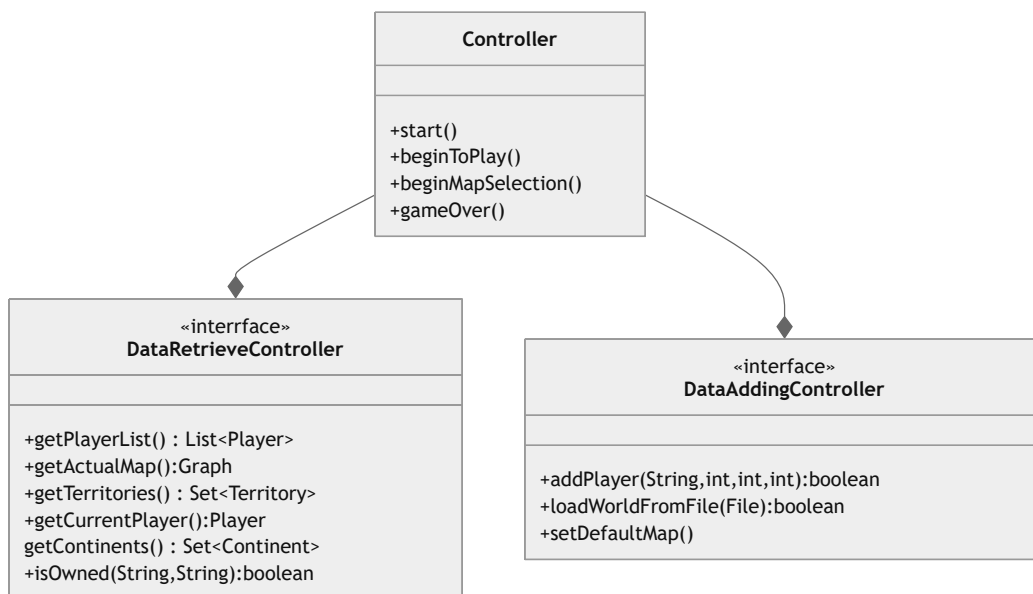


Figura 2.7: UML dei vari controller per le prime due fasi di gioco.

Risiko è un software molto legato alla visualizzazione del gioco, quindi per favorire il *Separation of Concerns*, il controller è diviso in sotto-controller: `DataAddingController` permette di impostare i giocatori e la mappa, mentre `DataRetrieveController` favorisce l'ottenimento di informazioni quali

il giocatore corrente. In questo caso è stato usato il *Delegate Pattern*, dove si creano delle classi per ottenere e aggiungere dati nelle prime due 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, mostrando quello del nuovo proprietario.

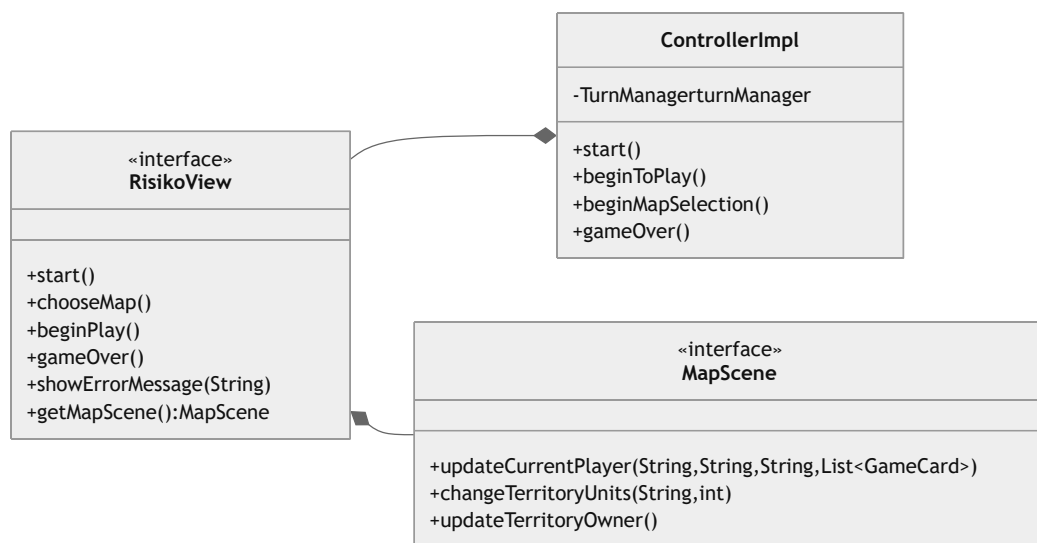


Figura 2.8: UML del observer pattern usato per aggiornare gli elementi delle view.

**Soluzione** Usare un'interfaccia dedicata agli aggiornamenti della View. Ogni volta che c'è una variazione al model, il controller chiama i metodi dell'interfaccia per aggiornare la View. Il patter usato quì è una sorta di *Observer*, 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.

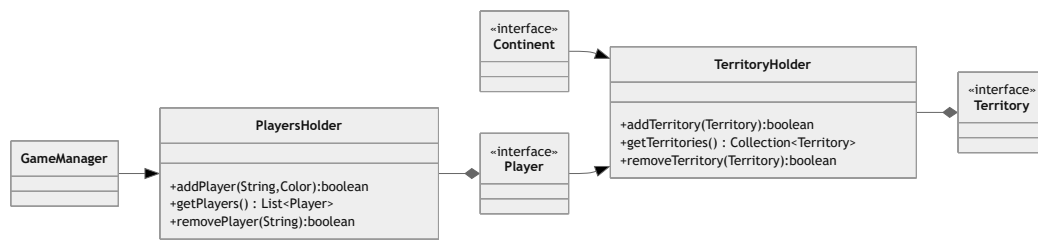


Figura 2.9: UML degli holders del gioco.

**Soluzione** Si creano delle interfacce in cui si accorpano tutte le operazioni in comune riguardanti 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 la gestione delle fasi di gioco. L'interfaccia grafica è stata testata manualmente durante lo sviluppo del software.

### 3.2 Note di sviluppo

#### 3.2.1 Matteo Caruso

##### Utilizzo della libreria GraphStream

Permalink <https://github.com/CarusoMatteo/Risik00P/blob/ef7453b6b504f2af3d8dbcf8706244942db86c6f/src/main/java/it/unibo/risikoop/view/implementations/scenes/mapscene/MapJPanel.java#L57-L59>

##### Utilizzo di Stream e Method References

Utilizzata in vari punti. Un esempio è: <https://github.com/CarusoMatteo/Risik00P/blob/ef7453b6b504f2af3d8dbcf8706244942db86c6f/src/main/java/it/unibo/risikoop/model/implementations/gamecards/combo/WildAllEqualCombo.java#L47-L54>



### 3.2.2 Matteo Ceccarelli

#### Generics avanzati

Permalink <https://github.com/CarusoMatteo/Risik00P/blob/5033629c3594c141715ed421e7aa94e77a9c043/src/main/java/it/unibo/risikoop/model/interfaces/Specification.java#L11>

#### Lambda expressions

Permalink <https://github.com/CarusoMatteo/Risik00P/blob/5033629c3594c141715ed421e7aa94e77a9c043/src/main/java/it/unibo/risikoop/model/interfaces/Specification.java#L38C5-L40C6>

#### Stream

Permalink <https://github.com/CarusoMatteo/Risik00P/blob/5033629c3594c141715ed421e7aa94e77a9c043/src/main/java/it/unibo/risikoop/model/implementations/TurnManagerImpl.java#L49C9-L50C58> Permalink <https://github.com/CarusoMatteo/Risik00P/blob/5033629c3594c141715ed421e7aa94e77a9c043/src/main/java/it/unibo/risikoop/model/implementations/gamecards/objectivecards/ConquerNContinetsBuilder.java#L42C9-L47C31>

#### Reflection Class<T>

Permalink <https://github.com/CarusoMatteo/Risik00P/blob/5033629c3594c141715ed421e7aa94e77a9c043/src/main/java/it/unibo/risikoop/controller/implementations/GamePhaseControllerImpl.java#L235C5-L240C6>

### 3.2.3 Francesco Sacripante

#### Utilizzo degli stream

Per gestire e far eseguire delle operazioni su una lista: <https://github.com/CarusoMatteo/Risik00P/blob/ee602038cf4972b80a3f30fb006dca5d9ab2ccb1/src/main/java/it/unibo/risikoop/model/interfaces/holder/TerritoryHolder.java#L41>

#### Utilizzo della libreria esterna GraphStream

Per la gestione della mappa di gioco dinamica: <https://github.com/CarusoMatteo/Risik00P/blob/ee602038cf4972b80a3f30fb006dca5d9ab2ccb1/src/main/java/it/unibo/risikoop/model/implementations/TerritoryImpl.java#L75>

## Utilizzo della libreria esterna Jackson

Per il caricamento della mappa da file: <https://github.com/CarusoMatteo/Risik00P/blob/ee602038cf4972b80a3f30fb006dca5d9ab2ccb1/src/main/java/it/unibo/risikoop/controller/implementations/DataAddingControllerImpl.java#L46>

# Capitolo 4

## Commenti finali

### 4.1 Autovalutazione e lavori futuri

#### 4.1.1 Matteo Caruso

Malgrado l'avvio stentato dello sviluppo del progetto, grazie alla collaborazione dei miei compagni, sono riuscito a trovare la giusta motivazione per portarlo a termine con buon risultato. Sono fiero di come ho gestito la validazione delle combo, riuscendo a comporre due pattern. La costruzione della schermata principale è stata appagante da ideare per l'adattamento alle proporzioni dello schermo, nonostante non aspiri a lavorare con le interfacce grafiche. È stata interessante la condivisione del lavoro, anche se è stato talvolta difficile sincronizzare la collaborazione a causa dei periodi di maggior attività diversi dei membri del gruppo. La divisione del lavoro è stata equa, nonostante Risiko sia molto incentrato sulla visualizzazione del gioco. Questo progetto è stata un'esperienza di apprendimento proficua, nonché la mia prima esperienza di un grande lavoro in gruppo. Credo che questa esperienza tornerà utile in futuro.

#### 4.1.2 Matteo Ceccarelli

Credo che la parte meglio riuscita del progetto sia stata l'implementazione delle carte obiettivo. È stata infatti la sezione in cui ho potuto maggiormente cimentarmi nel design, sperimentando soluzioni che ritengo valide, facilmente estendibili e manutenibili. Ho trovato estremamente interessante osservare concretamente come i diversi pattern architettonici possano essere combinati tra loro in modo armonico.

La parte che mi ha coinvolto di meno è stata invece la gestione delle fasi di gioco tramite macchina a stati. Pur ritenendo questa scelta la più

adatta in termini di chiarezza, leggibilità, logica e mantenibilità del codice – e rappresentando un approccio tipico del mondo industriale nel quale lavoro da anni – l’ho trovata meno stimolante, nonostante in questo progetto l’abbia rielaborata in un’ottica orientata agli oggetti.

Per quanto riguarda il lavoro in gruppo, lo considero l’esperienza più formativa. Sebbene programmi da diversi anni, raramente mi era capitato di affrontare lo sviluppo collaborativo e simultaneo del codice. Il coordinamento con i colleghi è stato senza dubbio uno degli aspetti più arricchenti del progetto.

Un altro elemento di grande utilità è stato l’approccio a Git e GitHub: strumenti che non avevo mai utilizzato prima e che ho trovato estremamente pratici ed efficaci nella gestione condivisa del codice.

Ritengo che il mio contributo all’interno del gruppo sia stato equilibrato. Sono consapevole che avrei potuto dare di più, ma per esigenze lavorative non sempre mi è stato possibile partecipare al progetto con continuità.

### **4.1.3 Francesco Sacripante**

Purtroppo vi è stata una fase di pre-progetto più lunga del previsto, in cui cercavamo di strutturare almeno lo scheletro del progetto affinché tutti potessero lavorare in parallelo senza pestare i piedi agli altri. E’ stata forse una fase troppo lunga, cosa che ci è costata l’iniziare tardi il progetto in sé, però questo ci ha permesso di finire il progetto in sole tre settimane. Purtroppo non mi sento di aver usato codice avanzato come gli altri componenti, però vado fiero di essere la prima persona ad aver pensato al concetto di macchina a stati (anche se poi l’idea non è stata sviluppata e implementata da me). Un’altra cosa che mi è piaciuta è l’essere riuscito a coordinare l’avanzamento del progetto ed anche il come sia il prodotto finito, migliorabile ma molto meglio rispetto alle aspettative iniziali. È stato anche molto soddisfacente l’essere riuscito ad implementare la lettura da file json delle varie custom map, siccome l’idea iniziale era di usare un file di testo normale. E sono comunque soddisfatto di aver strutturato il progetto in modo tale da rendere più facile una possibile futura implementazione di file di salvataggio di una partita, in quanto basterebbe serializzare il game manager ed il turn manager nel loro stato. In generale, contando che all’inizio la realizzazione del progetto sembrava lontanissima, essere arrivato alla fine con questo risultato mi rassicura e rallegra decisamente, inoltre sono contento di come si è lavorato in gruppo ed anche delle cose che ho fatto a livello di codice (lo scheletro).

# Appendice A

## Guida utente

### A.1 Avviare la partita

Prima di iniziare la partita bisogna definire i giocatori e la mappa su cui giocare.

#### A.1.1 Inserire i giocatori

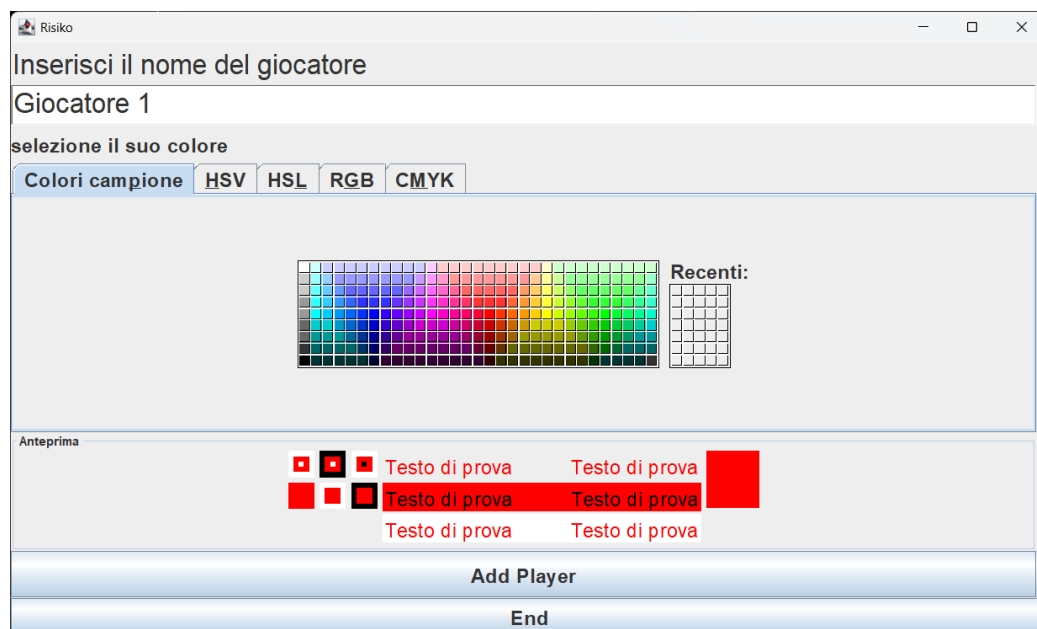


Figura A.1: Schermata di aggiunta giocatori.

Per inserire un giocatore, scrivere il suo nome e selezionare il suo colore. Infine cliccare sul pulsante **Add Player**. Ripetere il procedimento per ogni giocatore (almeno due), ricordando che sia il nome che il colore devono essere univoci.

Per concludere la scelta del giocatore, premere il pulsante **End**.

### A.1.2 Scegliere la mappa

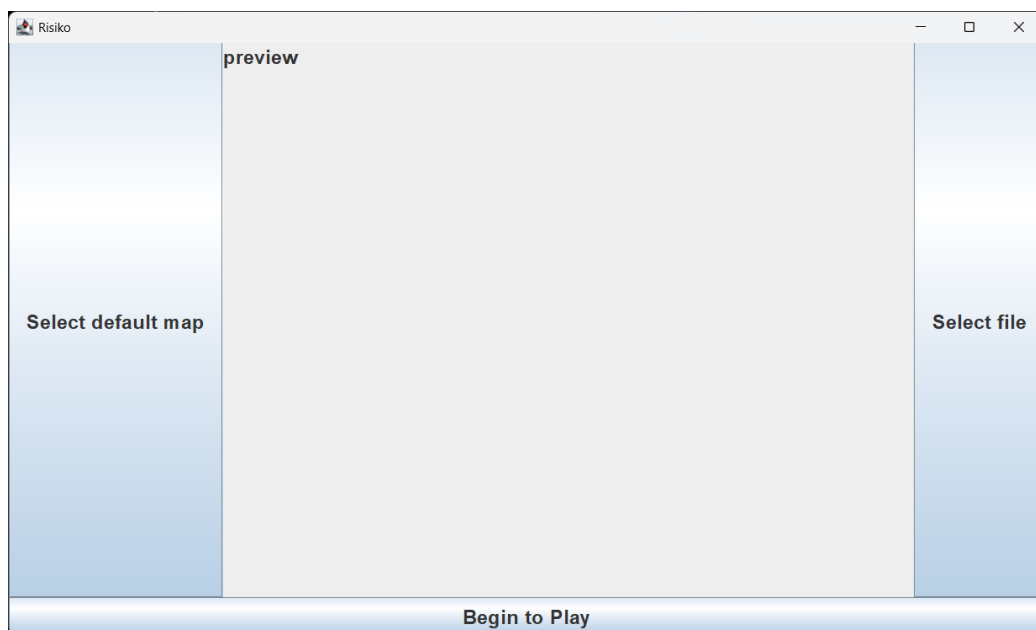


Figura A.2: Schermata di selezione mappa.

Cliccare il pulsante **Select default map** per usare la mappa di default, altrimenti cliccare su **Select file** per scegliere un file **json** contenente i dati della mappa. In caso di successo, verrà visualizzata l'anteprima della mappa selezionata.

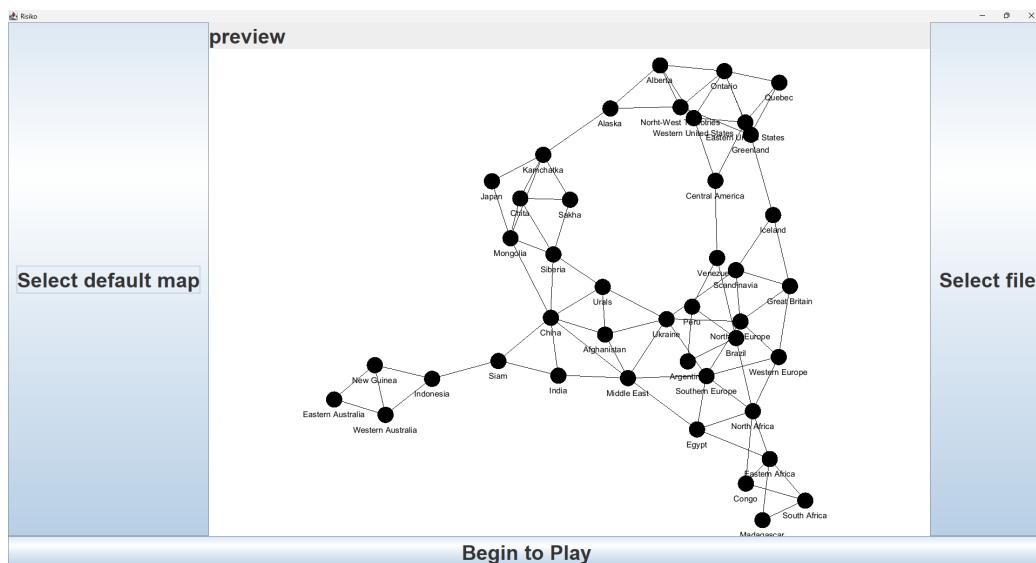


Figura A.3: Anteprima della mappa di default selezionata.

Una volta completata la scelta, cliccare sul pulsante **Begin to Play** per iniziare la partita.

## A.2 Rinforzi iniziali

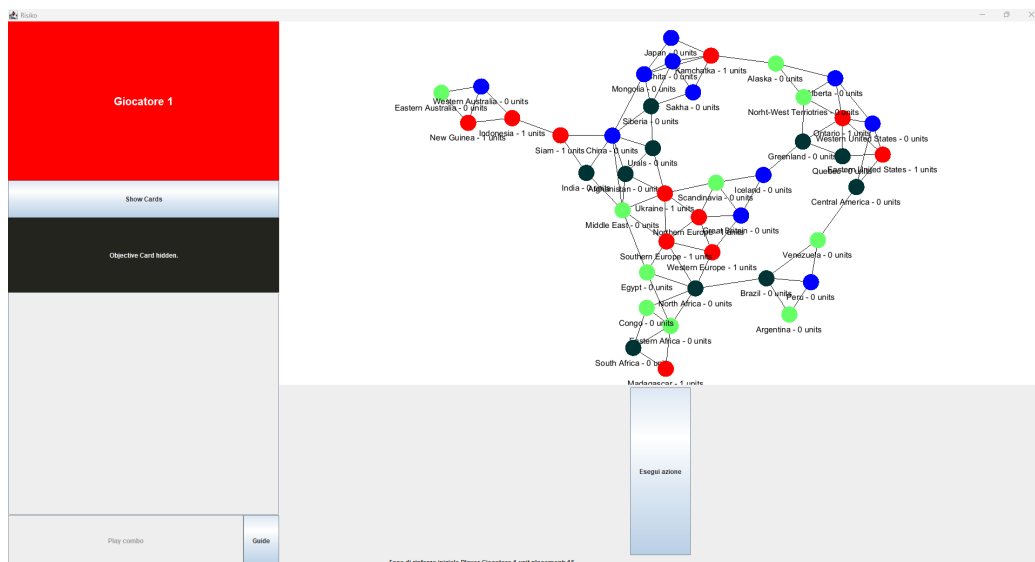


Figura A.4: Schermata di gioco iniziale.

I territori sono assegnati randomicamente ad ogni giocatore e viene posizionata un'unità su ogni territorio. Ora a turno i giocatori posizionano le unità a loro rimaste. Il contatore è mostrato nel pannello inferiore. Cliccare su un territorio in possesso a quel giocatore per rinforzarlo fino a quando sono state posizionate tutte le unità. Infine cliccare sul pulsante **Esegui azione** per permettere al giocatore successivo di rinforzare i propri territori.

### Spostare e selezionare i nodi del grafo

È possibile trascinare i nodi del grafo per spostarli in caso di sovrapposizione. Se non è possibile spostare o cliccare i nodi del grafo, impostare il ridimensionamento dello schermo a 100%. È un bug noto con la libreria GraphStream che stiamo utilizzando (Fonti: StackOverflow, Github).

## A.3 Schermata di gioco

La fase di preparazione è stata completata. D'ora in avanti si svolgerà il turno di gioco normale fino alla vittoria di un giocatore. La mappa è mostrata in alto a destra, dove il colore del territorio rappresenta il giocatore che lo possiede. Il pannello a sinistra mostra informazioni relative al giocatore corrente. È possibile visualizzare o nascondere i dati privati, ovvero le carte e l'obiettivo, cliccando sul pulsante **Show Cards** o **Hide Cards**.

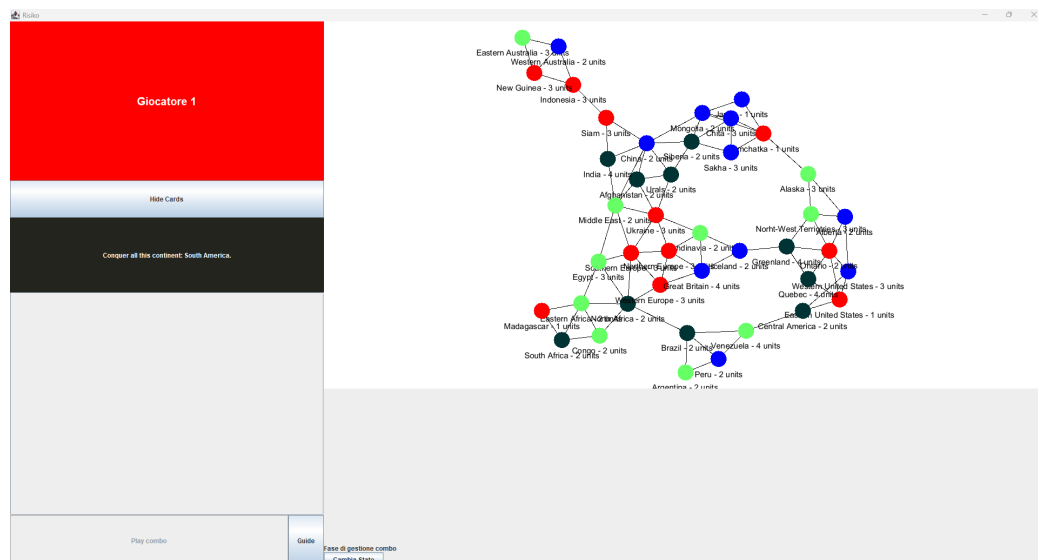


Figura A.5: Schermata di gioco con carte visibili.



È possibile visualizzare la lista di continenti e territori cliccando sul pulsante **Guide**.

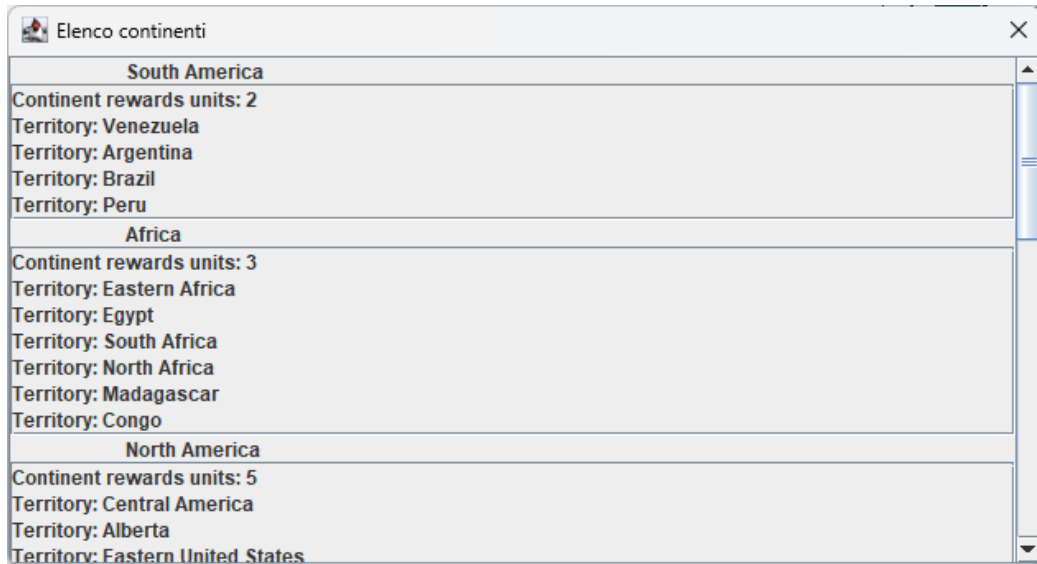


Figura A.6: Guida ai continenti e territori.

### A.3.1 Tipi di obiettivi

- Sconfiggere un determinato giocatore o conquistare 24 territori.
- Conquistare 18 territori e occuparli tutti con almeno 2 unità.
- Conquistare dei continenti, scelti in modo che la somma dei territori sia maggiore di 16.

## A.4 Fase di rinforzi

La fase di rinforzi permette di rinforzare i territori all'inizio di ogni turno. Il giocatore ottiene tante unità quanti sono i territori da lui occupati diviso tre, arrotondato per difetto. Inoltre, se il giocatore possiede tutti i territori di un continente, ottiene un bonus specifico per quel continente, visibile nella Guida.

### A.4.1 Giocare le combo

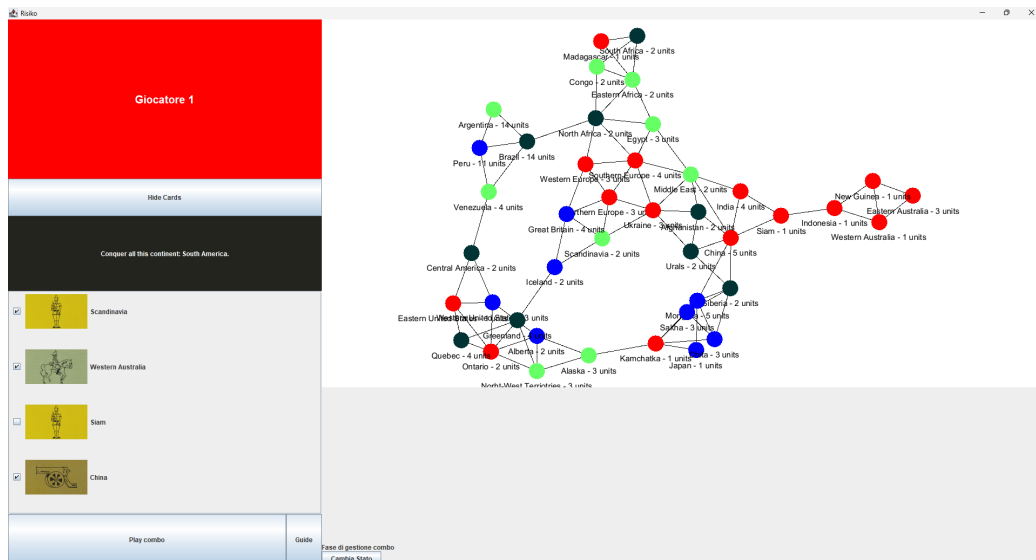


Figura A.7: Carte valide selezionate per giocare una combo.

Prima di posizionare le unità, il giocatore può giocare una o più combo di carte. Per ottenere una carta, un giocatore deve conquistare almeno un territorio durante la fase di attacco del suo turno. Ogni carta ha un seme (**Cannone**, **Fante**, **Cavaliere**) e un territorio. Nel mazzo sono presenti anche due carte **Jolly**.

Le combo utilizzabili sono le seguenti:

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

Per giocare una combo, selezionare le *checkbox* delle carte da giocare e cliccare sul pulsante **Play Combo**, che si attiverà solo se la combo è valida.

Sia che il giocatore abbia giocato una combo o meno, cliccare sul pulsante **Cambia stato** per passare alla fase di posizionamento delle unità.

### A.4.2 Aggiungere unità

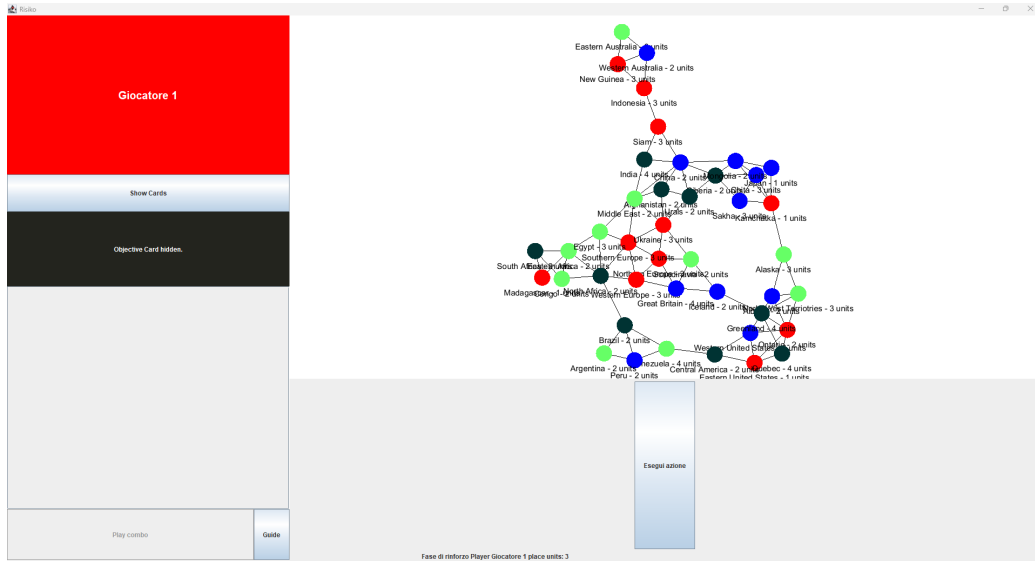


Figura A.8: Fase di rinforzo.

L'aggiunta di unità sui territori del giocatore avviene come nella fase di rinforzi iniziali, ovvero cliccando sui territori nella mappa fino ad averle posizionate tutte. Una volta posizionate tutte le unità, cliccare sul pulsante **Esegui azione** per passare alla fase di attacco.

## A.5 Fase di attacco

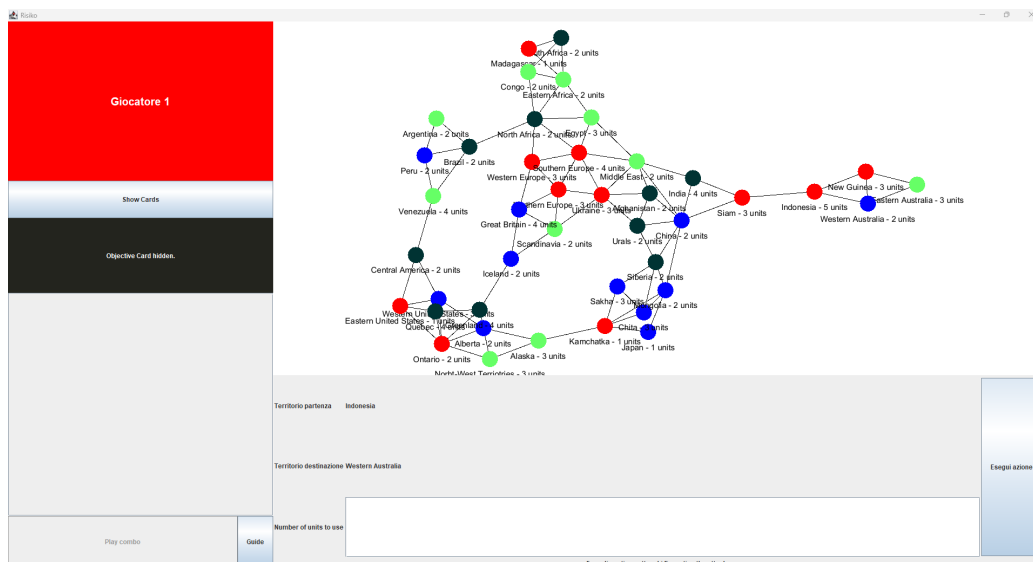


Figura A.9: Fase di attacco tra Indonesia e Australia Occidentale.

La fase di attacco permette a un giocatore di utilizzare le unità posizionate su un suo territorio per attaccare e conquistare un territorio adiacente di un altro giocatore.

Occorre pianificare l'attacco in vari passaggi, dopo i quali bisogna cliccare il pulsante **Esegui azione** per passare allo stato successivo:

1. Cliccare il territorio con cui attaccare, facendo attenzione che abbia un territorio nemico adiacente e che sia occupato da almeno due unità.
2. Cliccare il territorio da attaccare. Dev'essere adiacente al territorio con cui si sta attaccando e deve appartenere a un giocatore avversario.
3. Scegliere con quante unità bisogna svolgere l'attacco. Un massimo di 3 unità possono essere usate per attaccare, ma in caso di conquista del territorio tutte le unità definite qui saranno spostate nel territorio conquistato. Il numero di unità con cui attaccare deve essere compreso tra uno e il numero di unità del territorio con cui si sta attaccando meno uno.

Per eseguire l'attacco tirando i dadi bisogna cliccare un'ultima volta sul pulsante **Esegui azione**. A seconda dell'esito dell'attacco, le unità sconfitte saranno rimosse dai territori. Nel caso in cui le unità difensori sconfitte siano

state le ultime unità presenti su quel territorio, esso sarà conquistato dal giocatore attaccante e tutte le unità attaccanti rimaste saranno spostate nel territorio conquistato.

Se durante la fase di attacco il giocatore ha conquistato almeno un territorio, otterrà una Carta Territorio, che sarà utilizzabile per giocare combo nei turni successivi. Il giocatore corrente potrà decidere di effettuare altri attacchi o passare alla fase di spostamento strategico cliccando sul pulsante **Cambia stato**.

### Tiro di dadi

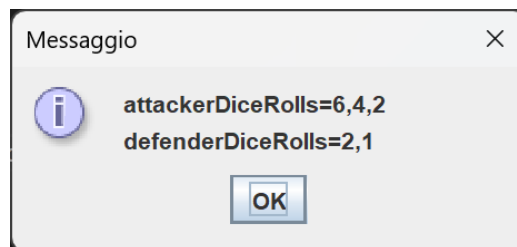


Figura A.10: Tiro di dadi.

L'attacco avviene tramite il tiro di dadi. Vengono tirati tanti dadi quante le unità attaccanti (massimo 3) e tanti dadi quante le unità difensori (massimo 3). I risultati dei dadi vengono ordinati in ordine decrescente e confrontati tra loro. Per ogni coppia di risultati, il dado attaccante deve essere strettamente maggiore di quello difensore per sconfiggere un'unità difensore, in caso contrario sarà un'unità attaccante a perdere.

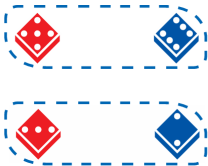

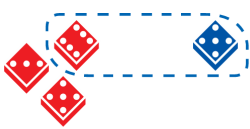
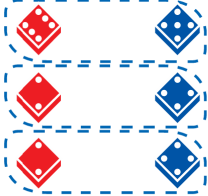
<p>ESEMPIO 1</p> <p>dadi dell'attaccante      dadi del difensore</p>  <p>RISULTATO</p> <p>L'attaccante perde un'armata e anche il difensore perde un'armata.</p>	<p>ESEMPIO 3</p> <p>dadi dell'attaccante      dadi del difensore</p>  <p>RISULTATO</p> <p>L'attaccante perde un'armata.</p>
<p>ESEMPIO 2</p> <p>dadi dell'attaccante      dadi del difensore</p>  <p>RISULTATO</p> <p>Il difensore perde un'armata.</p>	<p>ESEMPIO 4</p> <p>dadi dell'attaccante      dadi del difensore</p>  <p>RISULTATO</p> <p>Il difensore perde un'armata e l'attaccante ne perde due.</p>

Figura A.11: Esempi di tiri di dadi dal manuale di Risiko.

## A.6 Fase di spostamento strategico

La fase di spostamento strategico permette di spostare delle unità da un territorio ad un territorio adiacente, entrambi di proprietà del giocatore corrente, in un solo senso.

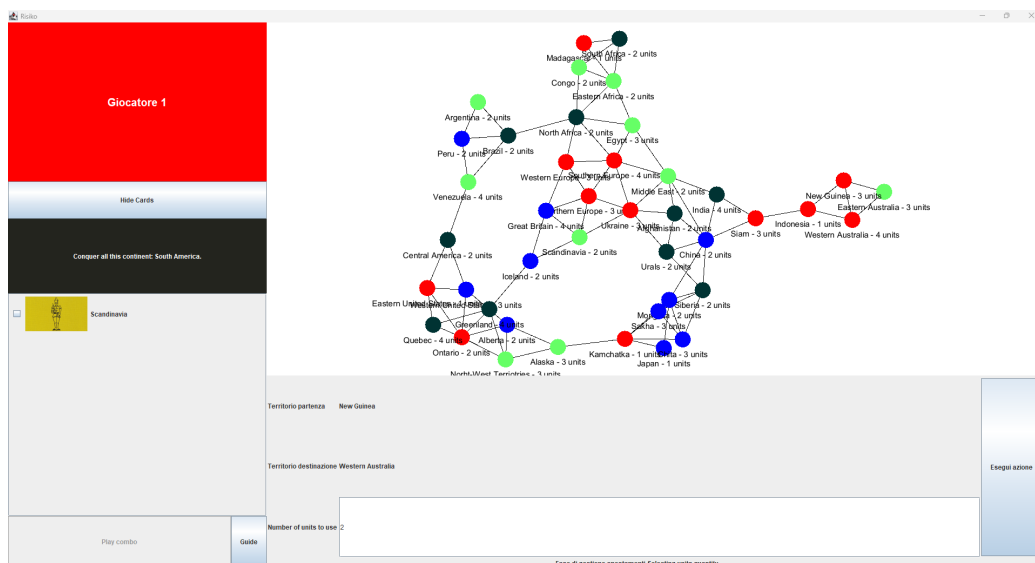


Figura A.12: Spostamento di un'unità dalla Nuova Guinea all'Australia Occidentale.

Come nella fase di attacco, bisogna pianificare lo spostamento. Per passare allo stato successivo bisogna cliccare il pulsante **Esegui azione**:

1. Cliccare il territorio da cui spostare le unità, facendo attenzione che abbia almeno due unità e che sia adiacente a un territorio dello stesso giocatore.
2. Cliccare il territorio dove le unità verranno spostate. Dev'essere adiacente al territorio scelto nel passaggio precedente e di proprietà del giocatore.
3. Scegliere quante unità spostare. Dev'essere un numero compreso tra uno e il numero massimo di unità del territorio di partenza meno uno.

Per confermare lo spostamento, cliccare un'ultima volta il pulsante **Esegui azione**. Il turno del giocatore attuale è concluso, tocca quindi al giocatore successivo.

## A.7 Vittoria

Un giocatore vince se riesce a completare il suo obiettivo. Il controllo avviene alla conclusione di ogni fase e di ogni attacco. Verrà mostrato il vincitore e sarà possibile giocare una nuova partita cliccando il pulsante **Play Again** o chiudere l'applicazione cliccando il pulsante **Close**.

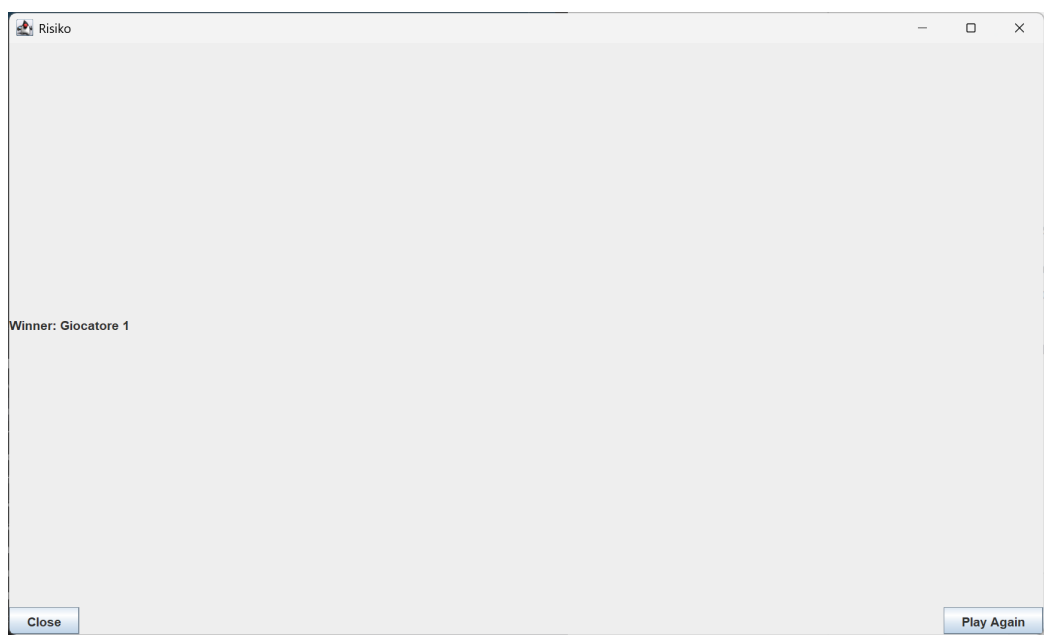


Figura A.13: Messaggio di vittoria.



# Appendice B

## Esercitazioni di laboratorio

### B.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>

### B.2 `franceso.sacripante@studio.unibo.it`

- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=177162#p246099>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=178723#p247200>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=179154#p248347>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=180101#p249155>
- Laboratorio 11: <https://virtuale.unibo.it/mod/forum/discuss.php?d=181206#p250657>