Laurea Magistrale in informatica - Università di Salerno
Corso di Ingegneria *Gestione ed Evoluzione del Software*

Strategic Robots
Impact Analysis

Versione	1.0
Data	18/05/2021
Destinatario	Prof. Andrea De Lucia Dott. Fabiano Pecorelli Dott. Emanuele Iannone Dott. Manuel De Stefano
Presentato da	Antonio Martella

Sommario

1. Introduzione

Questo documento nasce in seguito alla presentazione di una richiesta di modifica nel "Strategic Robots".

Nella prima sezione viene analizzato il sistema iniziale con i rispettivi requisiti funzionali.

Successivamente, dopo aver descritto la proposta di modifica e gli eventuali nuovi requisiti funzionali, si è analizzato l'impatto della modifica sul sistema corrente.

Infine, sono state riportate le modifiche effettuate e quali potrebbero essere gli sviluppi futuri.

2. Sistema corrente

2.1 Descrizione del sistema

Il sistema esistente preso come riferimento per l'implementazione di una serie di modifiche è il sistema "Strategic Robots", un gioco strategico in 2D sviluppato in Java durante il corso di programmazione 2 della laurea triennale in Informatica.

Sostanzialmente il gioco è caratterizzato da uno scenario in cui sono presenti degli ostacoli, un banco rifornimenti e 2 tipologie di robot che interagiscono tra di loro.
Dopo aver scelto le varie impostazioni della partita, l'utente può iniziare a giocare contro un altro utente sullo stesso computer alternando l'invio delle mosse da compiere.
Ogni utente può controllare una squadra di robot composta da robot combattenti (riconoscibili graficamente da una spada) e robot lavoratori (riconoscibili graficamente da una chiave inglese).
I robot combattenti possono spostarsi all'interno dello scenario e possono attaccare ostacoli (spostandosi nella loro casella) o altri robot facendo diminuire la loro energia.
I robot lavoratori possono spostarsi all'interno dello scenario, possono spostare ostacoli (spostandosi nella loro casella) e possono curare altri robot facendo aumentare la loro energia.
Ogni azione richiede un quantitativo di energia fissato. Se l'energia residua non è sufficiente per compiere un'azione, allora questa non potrà essere eseguita.
Entrambi i robot possono ricaricare la propria energia tramite il banco rifornimenti presente nello scenario.
Il banco rifornimento è indistruttibile, non può essere spostato e ha una riserva di energia per le ricariche. Per interagire col banco rifornimenti bisogna recarsi vicino con un robot e premere la barra spaziatrice.

Il gioco è organizzato in round. In ogni round viene scelto casualmente un robot che può essere controllato dall'utente. Il robot controllato è evidenziato.

Ogni utente può effettuare una sola mossa durante il round, dopodiché passa il controllo all'avversario. La mossa termina quando l'utente preme il tasto invio.

Quando il giocatore esegue una delle seguenti azioni non può continuare ad eseguire altre operazioni, di conseguenza è costretto a premere invio per passare il controllo all'altro giocatore:

- L'utente attacca un robot attraverso il robot combattente controllato.
- L'utente attacca un ostacolo attraverso il robot combattente controllato.
- L'utente cura un robot attraverso un robot lavoratore controllato.
- L'utente ricarica l'energia di un robot controllato tramite il banco dei rifornimenti

I robot e gli ostacoli che non hanno più energia dopo aver subito un attacco vengono distrutti. La partita termina quando uno dei due giocatori non ha più robot combattenti a disposizione.

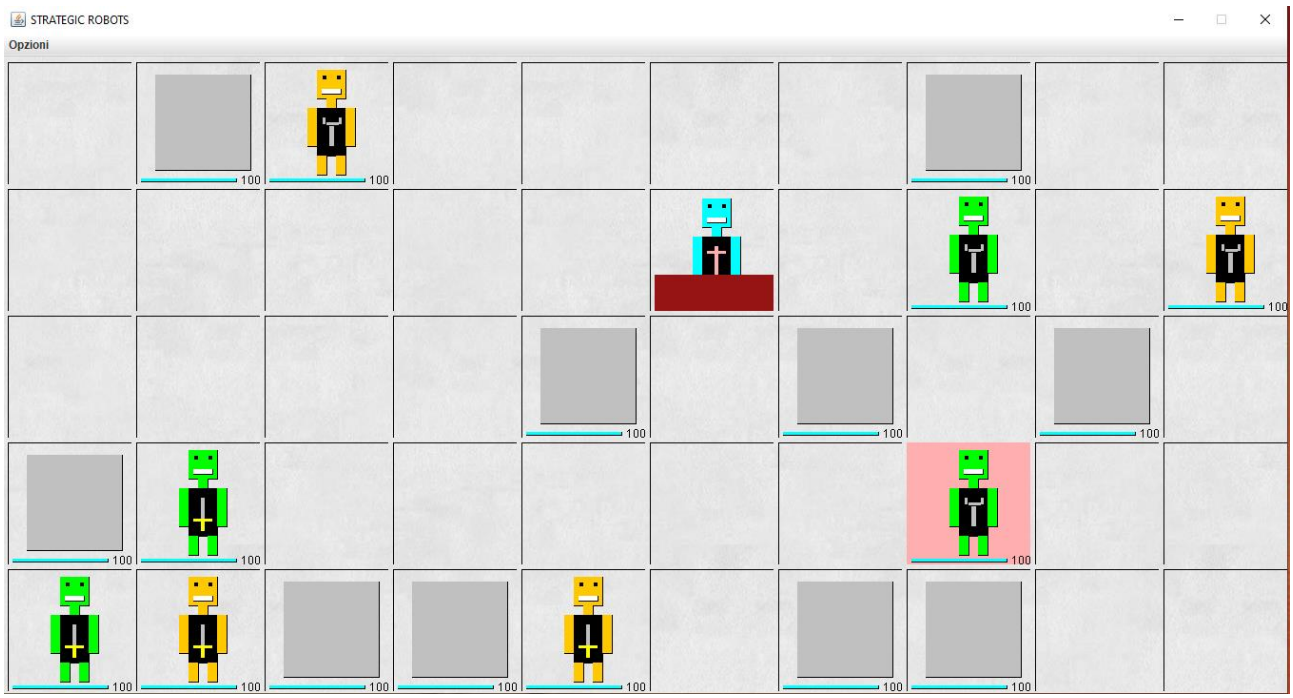

Figura 1 – Esempio scenario

2.1 Requisiti funzionali principali

Il sistema attuale presenta la modalità di gioco "giocatore 1 vs computer" ma non verrà presa in considerazione in quanto verrà sostituita con la modalità di gioco "online", di conseguenza in seguito sono presentati i requisiti funzionali del sistema che tiene conto solo della modalità "giocatore 1 vs giocatore 2", utili ai fini della comprensione del sistema e della successiva modifica proposta.

RF_1 – Visualizza Home

Il sistema permette all'utente di visualizzare la home page.

RF_2 – Impostazioni partita:

Il sistema permette all'utente di impostare i parametri di una nuova partita.

RF_3 – Inizio nuova partita:

Il sistema permette all'utente di iniziare una nuova partita in locale.

RF_4 – Visualizza regole:

Il sistema permette all'utente di visualizzare le regole del gioco.

RF_5 – Controllo Robot:

Il sistema permette all'utente di controllare un robot combattente/lavoratore in maniera random.

RF_6 – Spostamento robot:

Il sistema permette all'utente di spostare un robot combattente/lavoratore.

RF_7 – Spostamento ostacolo:

Il sistema permette all'utente di spostare un ostacolo tramite un robot lavoratore.

RF_8 – Attacco robot:

Il sistema permette all'utente di attaccare un robot combattente/lavoratore tramite un robot combattente.

RF_9 – Attacco ostacolo:

Il sistema permette all'utente di attaccare un ostacolo tramite un robot combattente.

RF_10 – Cura robot:

Il sistema permette all'utente di curare un robot combattente/lavoratore tramite un robot lavoratore.

RF_11 – Ricarica energia:

Il sistema permette all'utente di ricaricare l'energia di un robot tramite il banco rifornimenti.

RF_12 – Fine mossa:

Il sistema permette all'utente di terminare la mossa e passare il controllo alla squadra avversaria sulla stessa macchina.

RF_13 – Notifica robot quasi distrutto:

Il sistema avvisa l'utente, tramite un messaggio, quando l'energia di un robot scende sotto il 25%.

RF_14 – Notifica fine partita:

Il sistema avvisa l'utente, tramite un messaggio, quando c'è una squadra vincitrice e quindi la fine della partita

2.2 Activity diagram

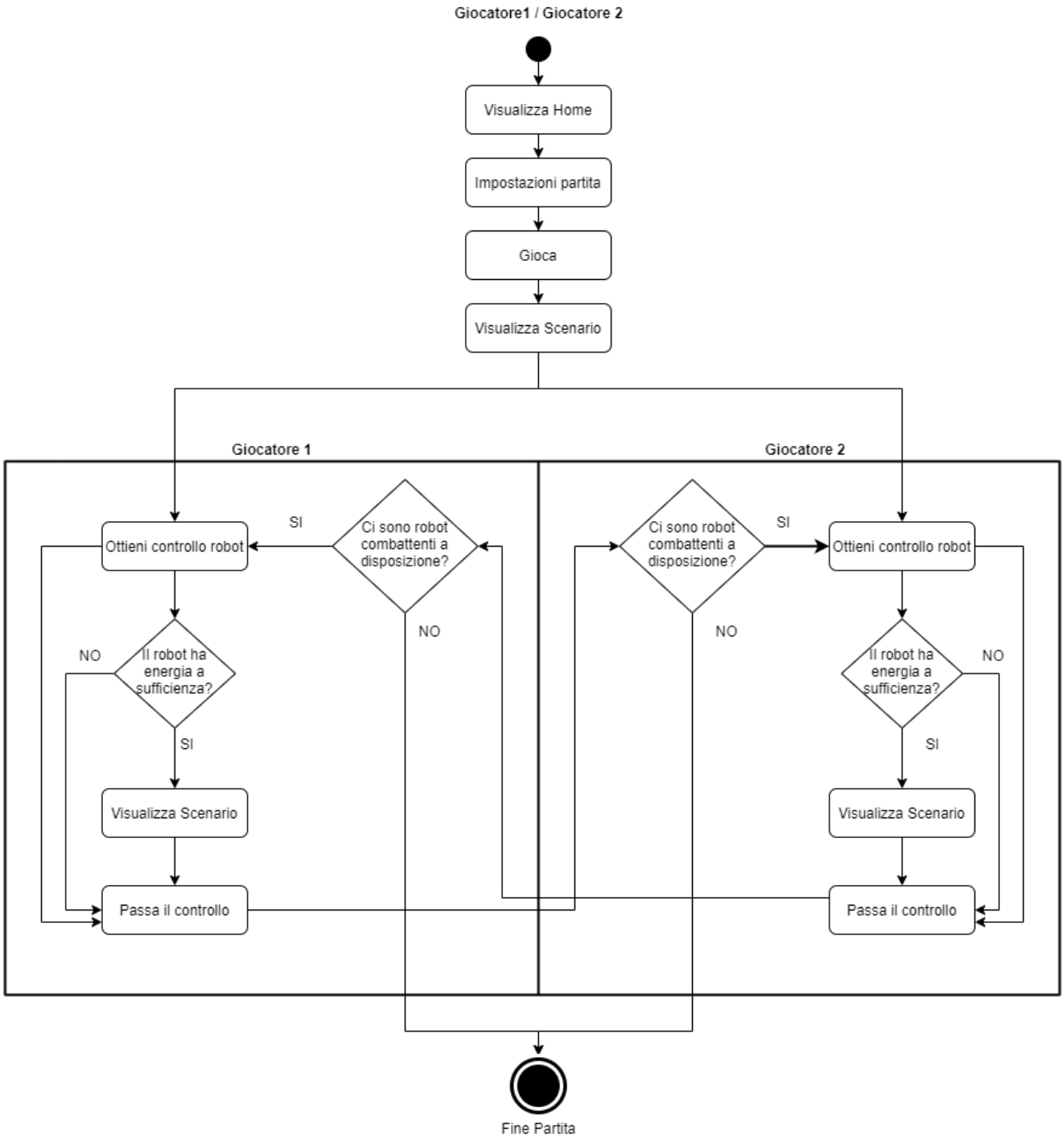

Figura 2 - Activity diagram partita

3. Funzionalità introdotte

3.1 Proposta di modifica

La proposta di modifica è quella di rendere il gioco concorrente aggiungendo un'architettura client-server attraverso l'utilizzo di java RMI in modo da dare la possibilità agli utenti di giocare su diverse macchine.
Di conseguenza verrà aggiunta una nuova modalità dove l'utente si connette ad un server che farà partire una partita contro un altro utente.
In aggiunta, si vuole dare la possibilità di salvare una partita e di riprenderla a giocare in un secondo momento. Verrà aggiunta una nuova modalità che permette all'utente di caricare l'ultima partita salvata e di conseguenza verrà data la possibilità di salvare una partita (sovrascrivendo un eventuale salvataggio) mentre la si sta giocando. Considerando che questo gioco viene visto come una specie di partita a scacchi, si è deciso di implementare il salvataggio solo per la modalità in locale, in quanto non avrebbe senso riprendere una partita online contro un avversario sconosciuto che non si sa in che momento si riconnette al server.

3.2 Nuovi requisiti funzionali

L'accettazione della modifica implica l'aggiunta di nuovi requisiti funzionali di seguito riportati:

RF_15 – Inizio nuova partita online:

Il sistema permette all'utente di connettersi ad un server e iniziare una nuova partita online.

RF_16 – Fine mossa online:

Il sistema permette all'utente di terminare la mossa e passare il controllo alla squadra avversaria che sta giocando su un'altra macchina.

RF_17 – Abbandona partita online

Il sistema permette all'utente di abbandonare la partita

RF_18 – Salva Partita

Il sistema permette agli utenti che giocano in locale di salvare una partita.

RF_19 – Carica Partita

Il sistema permette agli utenti di riprendere a giocare l'ultima partita salvata in locale.

4. Impact Analysis

4.1 Diagramma UML delle classi

Nella seguente figura è riportato il diagramma UML delle classi. Successivamente andremo ad analizzare nel dettaglio le classi principali che terremo in considerazione per apportare le modifiche al sistema.

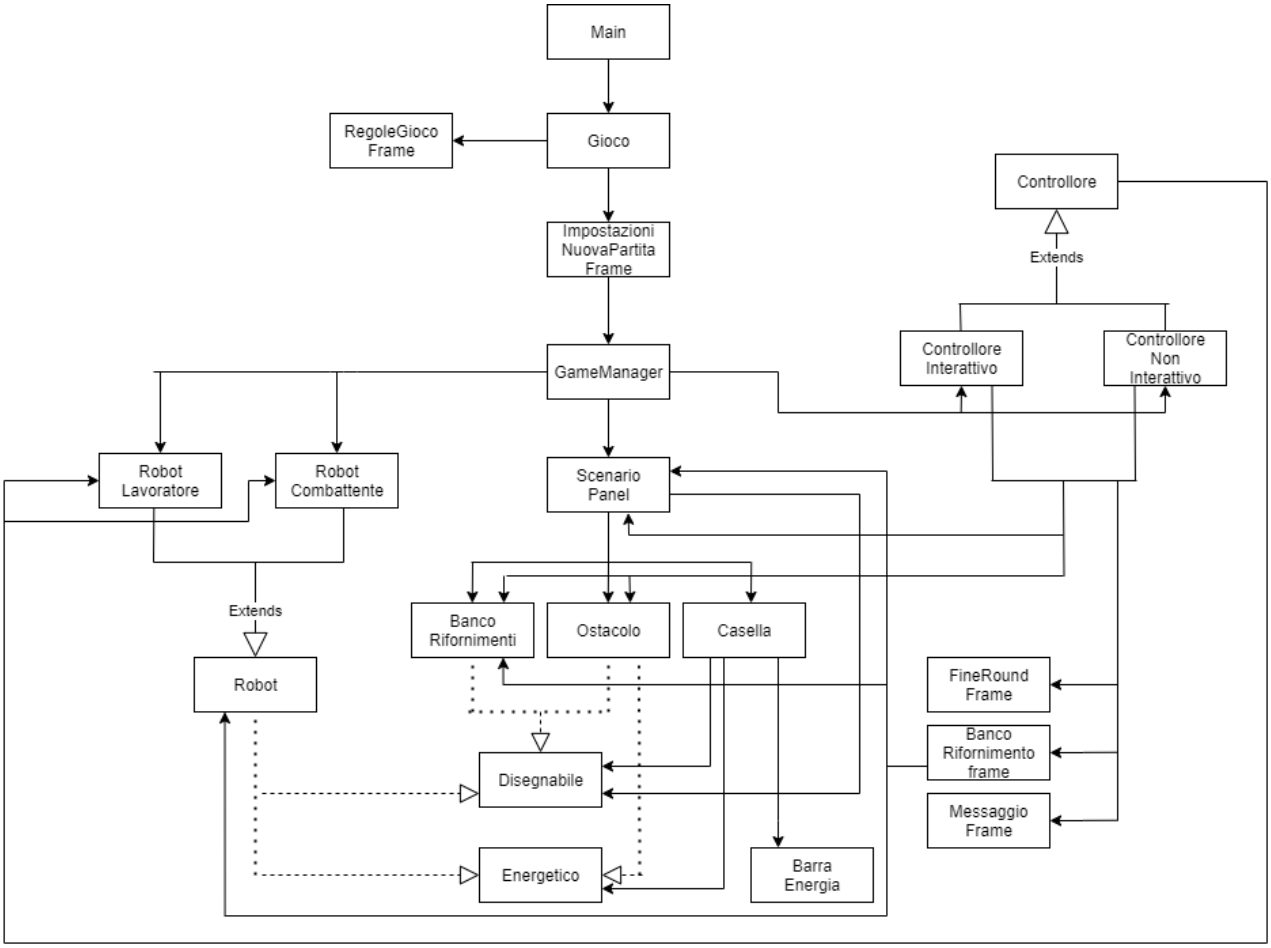

Figura 3 -UML diagram

4.2 Analisi iniziale

Per effettuare le modifiche precedentemente descritte, l'analisi iniziale si sofferma principalmente sulle classi GameManager.Java e ControlloreInterattivo.Java mentre la classe ControlloreNonInterattivo.java non verrà presa in considerazione in quanto riguarda una modalità che è stata tolta ai fini del progetto.

La classe ControlloreInterattivo.Java gestisce tutti i movimenti dei robot di una squadra all'interno dello scenario. Al suo interno è implementato un KeyListner che permette all'utente di effettuare delle azioni tramite i tasti della tastiera. Se il controllore è attivato, per ogni input dell'utente viene controllato se il robot può effettuare la mossa richiesta e eventualmente viene modificato lo scenario.

La classe Gamemanager.java si occupa invece di iniziare e gestire le mosse di una partita. All'interno del metodo "startNuovaPartita()" viene creato uno scenario con all'interno gli ostacoli e il banco rifornimenti posti in posizioni casuali. Successivamente vengono istanziati i due controllori delle rispettive squadre, aggiungendo lo scenario e la squadra di robot creata casualmente. All'interno di questa classe viene implementato un KeyListner che attiva e disattiva i controllori quando un giocatore preme il tasto invio.

Di seguito è riportato il class diagram delle classi precedentemente descritte

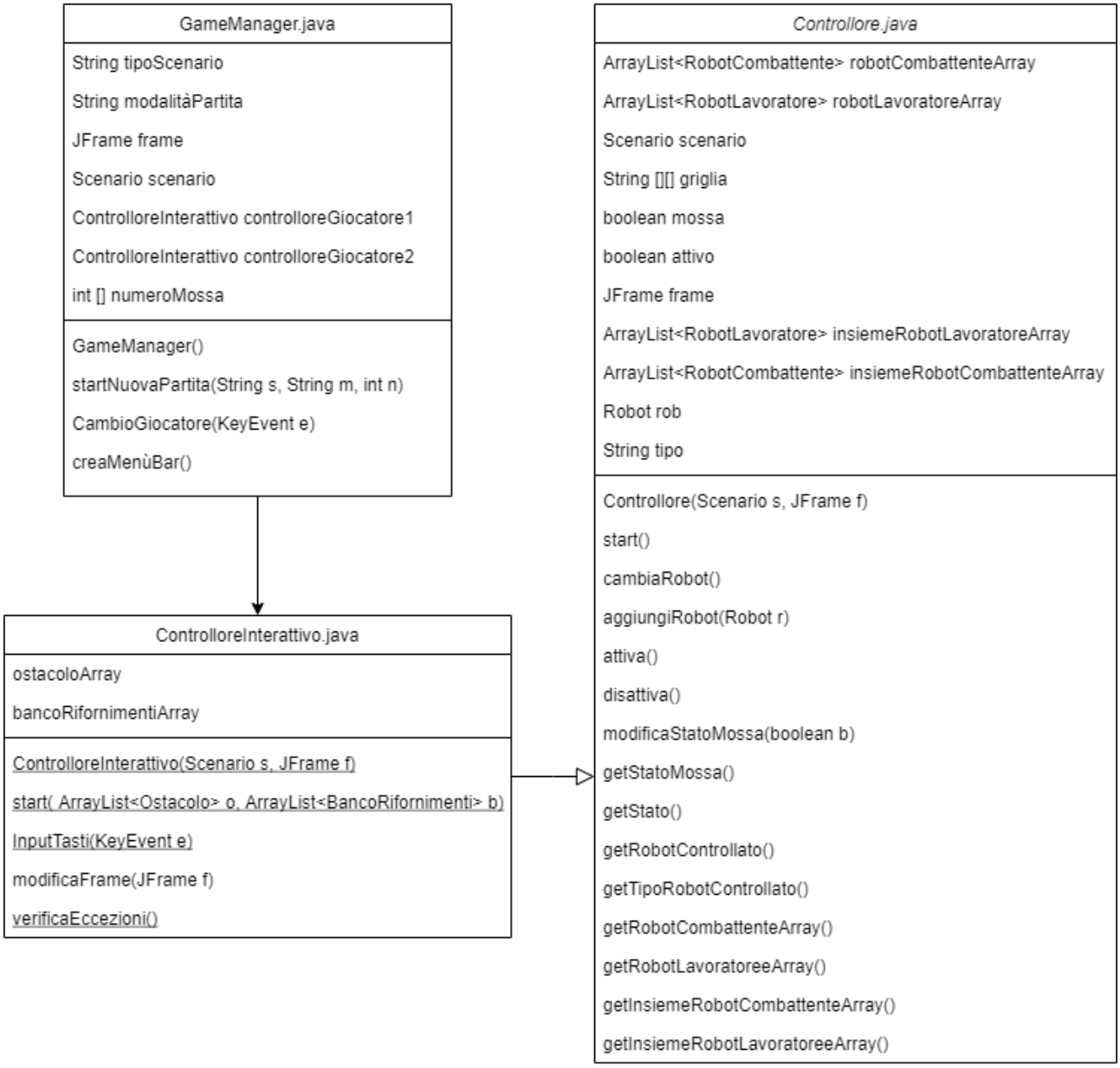

Figura 4 - Class diagram 1

L'intenzione è quella di rendere il gioco concorrente, aggiungendo un'architettura client-server attraverso java RMI. Tutti i dati della partita che servono al GameManager per gestire una partita dovrebbero essere inviati in qualche modo al server che dovrà poi aggiornare i client (che rappresentano i giocatori) ogni volta che c'è una modifica della partita. Di conseguenza verrà creata la classe DatiPartita.java contenente tutti i dati necessari.

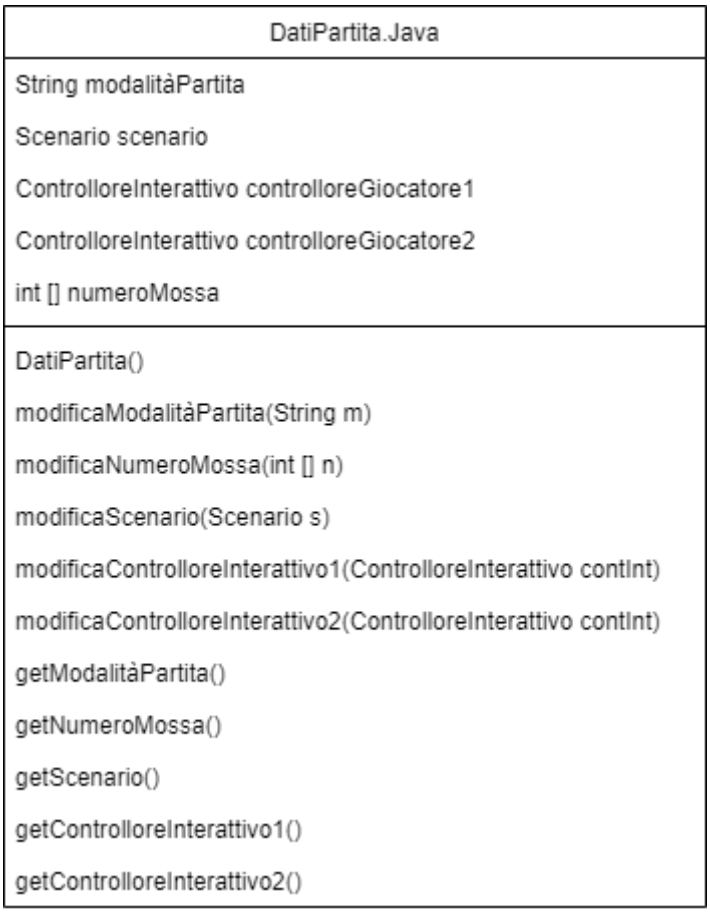

Figura 5 – DatiPartita.Java

Questa classe verrà utilizzata anche per il salvataggio e il caricamento di una partita salvata. Di conseguenza nella classe GameManager.java verrà implementato un metodo "*salvaPartita()*" che salverà in un file l'oggetto "DatiPartita" e un altro metodo "startPartitaSalvata()" che invece lo caricherà per recuperare i dati salvati.

4.3 Architettura Client-Server

Nell'architettura client-server, le funzionalità sono rigidamente divise tra due componenti dell'architettura: il server fornisce servizi e il client li richiede. Questo significa che lo scambio di informazioni viene sempre iniziato dal client e il server può solamente rispondere ad un'invocazione dei servizi da parte del server. Questa asimmetria, però, rappresenta una limitazione nelle iterazioni tra client e server in quanto il server non è in grado di iniziare una comunicazione con il client quando si vuole aggiornare lo stato della partita.

La soluzione che verrà adottata è quella della callback che segue, tipicamente, il pattern dell'Observer. Bisogna stabilire un meccanismo tramite il quale è possibile registrare delle dipendenze tra oggetti, in modo che un oggetto Observable riceve delle richieste da parte di un oggetto Observer di essere informato quando accade un evento. Ovviamente, sono necessarie, due interfacce: una per poter permettere all'oggetto Observer di registrarsi sull'Observable, ed una da parte dell'Observer per poter ricevere lo stato della partita quando l'avversario ha concluso una mossa.
Con questo meccanismo, il client effettua una richiesta di essere "richiamato" per ottenere le informazioni che riguardano la partita. In questa maniera le invocazioni di callback vengono effettuate da server verso il client. Questo non significa che i ruoli nell'architettura vengono scambiati, infatti il server continua a essere la componente che offre i servizi, che, però, prevedono che il server faccia richieste anche ai client connessi. Quando il client vuole invocare un servizio asincrono del server, fa in modo che il server possa effettuare una callback, effettuando una registrazione che permetta al server di memorizzare le informazioni sufficienti per poter effettuare la callback.

Il server avrà il compito di iniziare una partita e gestire il controllo delle mosse dei client. Quando il controllore di un client verrà attivato, darà la possibilità all'utente di compiere una mossa, mentre un altro client rimane in attesa fino a quando il giocatore preme il tasto invio per terminare la mossa. Dopo aver premuto il tasto invio, il server deve attivare il controllore dell'altro client passando tutti i dati aggiornati che fanno riferimento alla partita in corso.
Per realizzare tutto ciò verrà implementato un server che esporrà dei metodi remoti che servono per assicurare la possibilità di poter effettuare una callback, permettendo la registrazione e la de-registrazione del client. Il server implementerà l'interfaccia remota "ServerCallbackRemote" che avrà

i metodi: "registerForCallBack" e "unregisterCallBack". Inoltre, implementerà anche l'interfaccia remota "ServerGestioneStatoRemote" che avrà il metodo "aggiornaStatoServer" che dovrà essere chiamato dal client quando il giocatore avrà termitato la mossa in modo da poter passare tutti i dati della partita. Inoltre, verrà implementato il metodo "nuovaPartita" per iniziare una nuova partita inserendo nello scenario oggetti casuali (precedentemente veniva fatto dalla classe GameManager.java).

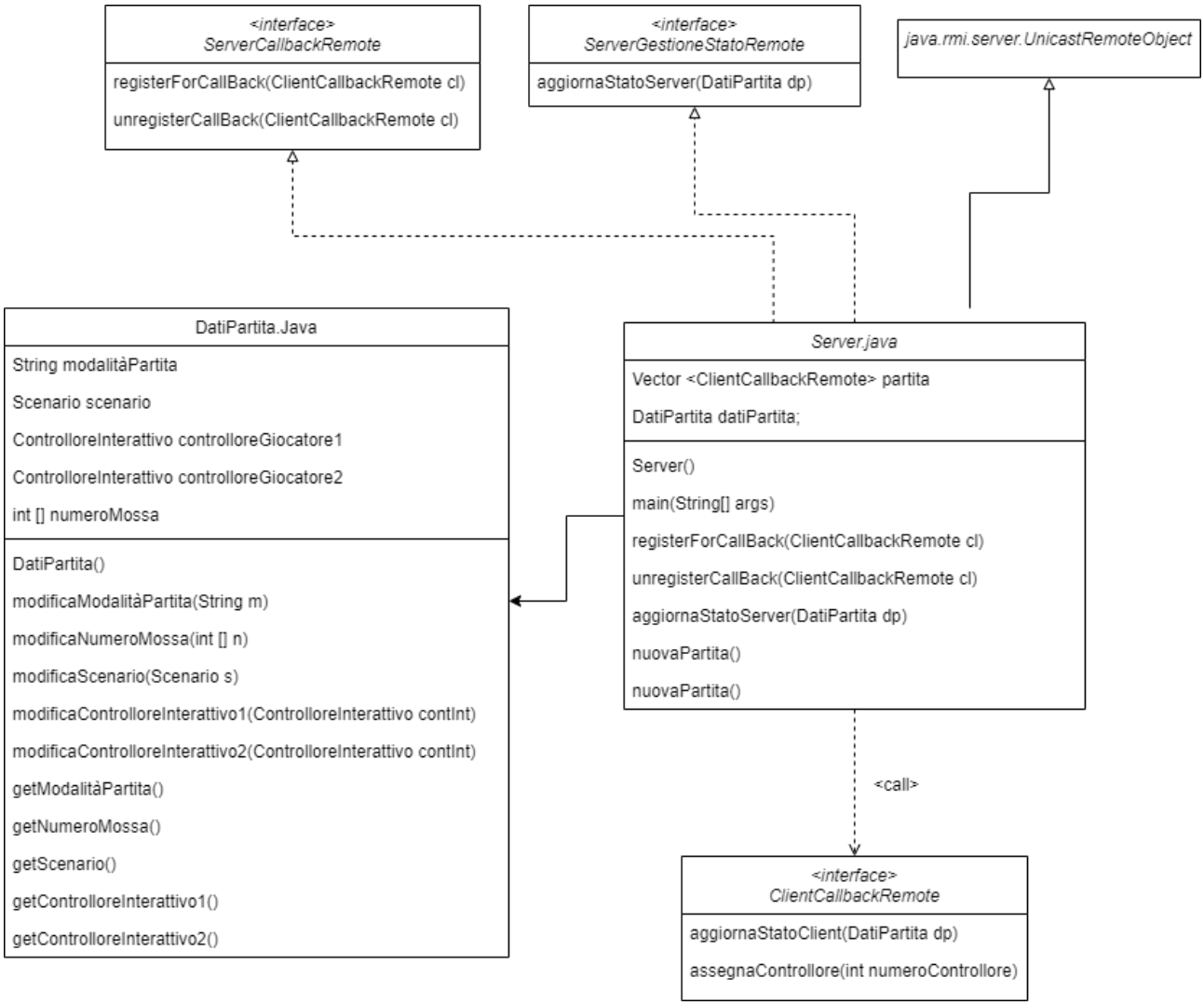

Figura 6 - Class diagram 2

Dal lato client verrà creatala classe ClientGameManager.java che avrà il compito di stabilire una connessione iniziale con il server, caricare i dati della partita dal server quando riceve il controllo e inviare i dati della partita quando il giocatore ha terminato la mossa oppure dare la possibilità di abbandonare la partita. Inoltre, dovrà gestire un keylistener per le mosse del controllore assegnato e notificare la fine di una partita.

Di conseguenza, la classe ClientGameManager implementerà l'interfaccia remota "ClientCallbackRemote" che avrà i metodi "aggiornaStatoClient" e "assegnaControllore". Inoltre, verrà implementato il metodo "start" per stabilire la connessione con il server, "aggiornaStatoPartitaServer" dove al suo interno ci sarà la chiamata al server che passerà i dati della partita, un KeyListener per gestire la fine della mossa, "finePartita" per notificare la fine della partita e il metodo "nuovoFrame" per creare il frame dove caricare lo scenario.

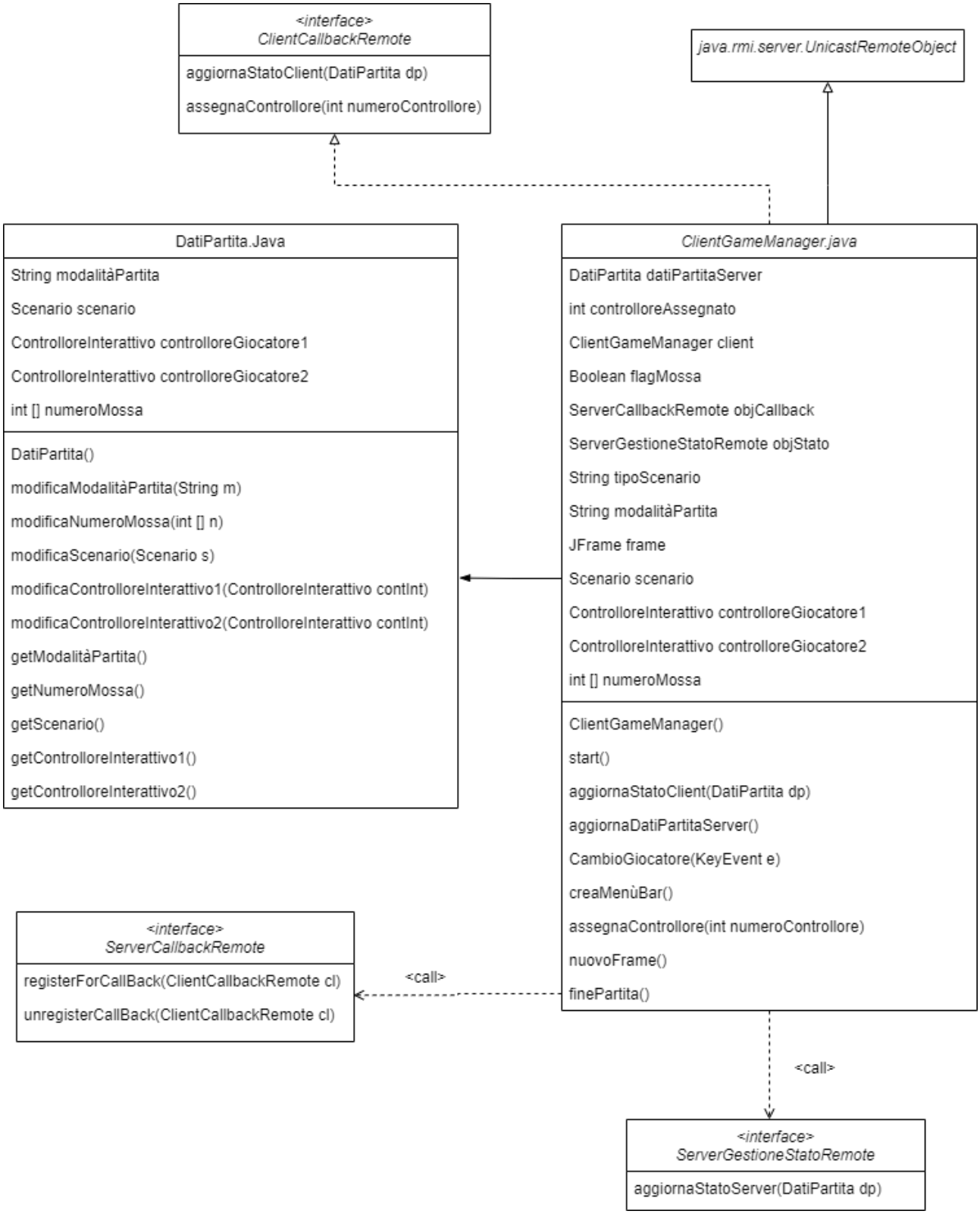

Figura 7 - Class diagram

4.4 Use case

Questo use case fa riferimento al requisito funzionale RF_03 già implementato

Identificativo UC_RF _03	Iniziare una nuova partita	Data	09/10/21
		Vers.	1.0
		Autore	Antonio Martella
Descrizione	Lo UC descrive la funzionalità di iniziare una nuova partita giocando in locale		
Attori Principali	**Giocatore1 / Giocatore2** I 2 giocatori vogliono iniziare a giocare.		
Attori secondari	NA		
Entry Condition	Giocatore 1 / 2 visualizza la home RF1. Giocatore 1 / 2 clicca su "nuova partita" e visualizza la pagina delle impostazioni RF2.		
Exit condition On success	Il sistema mostra ai giocatori lo scenario iniziale della partita.		
Exit condition On failure	NA		
Rilevanza/User Priority	NA		
Frequenza stimata	NA		
Extension point	NA		
Generalization of	NA		

FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO		
1	Giocatore 1/2	Uno dei giocatori imposta i parametri della partita e clicca su "gioca"
2	Sistema:	Il sistema mostra lo scenario della partita sulla stessa macchina con all'interno gli ostacoli, il banco di rifornimento, numero di robot (scelti dal giocatore), sfondo dello scenario (scelto dal giocatore) e dà la possibilità di compiere una mossa a uno dei due.
Special Requirements	NA	

Questo use case fa riferimento al requisito funzionale RF_15 che verrà implementato

Identificativo UC_RF _15	Iniziare una nuova partita online	Data	09/10/21
		Vers.	1.0
		Autore	Antonio Martella
Descrizione	Lo UC descrive la funzionalità di iniziare una nuova partita giocando da remoto		
Attori Principali	**Giocatore1 / Giocatore2** I 2 giocatori vogliono iniziare a giocare.		
Attori secondari	NA		
Entry Condition	Il server è stato avviato correttamente		
Exit condition On success	Il sistema mostra ai giocatori lo scenario iniziale della partita		
Exit condition On failure	Il sistema mostra un messaggio di errore del server oppure che il server è occupato		
Rilevanza/User Priority	NA		
Frequenza stimata	NA		
Extension point	NA		
Generalization of	NA		

FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO		
1	Giocatore 1	Il giocatore 1 è nella home page e clicca su "Gioca online".
2	Sistema:	Il sistema mostra il messaggio "Attendi giocatore".
3	Giocatore 2	Il giocatore 2 è nella home page e clicca su "Gioca online".
4	Sistema	Il sistema mostra lo scenario della partita ad entrambi i giocatori e dà la possibilità di compiere una mossa a uno dei due.
I Scenario/Flusso di eventi Alternativo: il server sta già gestendo una partita		
4a.1	**Sistema:**	Il sistema mostra il messaggio "Problemi con il server".
II Scenario/Flusso di eventi di ERRORE: Eventuali problemi con il server		
4b.1	**Sistema:**	Il sistema mostra il messaggio "Server occupato".
Special Requirements	NA	

Questo use case fa riferimento al requisito funzionale RF_12 già implementato

Identificativo UC_RF _12	Fine mossa	Data	09/10/21
		Vers.	1.0
		Autore	Antonio Martella
Descrizione	Lo UC descrive la funzionalità di terminare una mossa dando il controllo all'altro giocatore		
Attori Principali	**Giocatore1** Il giocatore1 vuole passare il controllo al giocatore2		
Attori secondari	**Giocatore2** Il giocatore2 attende la fine della mossa del giocatore 1		
Entry Condition	UC_RF_03		
Exit condition On success	Il sistema passa il controllo al giocatore 2 evidenziando il robot che dovrà controllare		
Exit condition On failure	NA		
Rilevanza/User Priority	NA		
Frequenza stimata	NA		
Extension point	NA		
Generalization of	NA		

FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO		
1	Giocatore 1	Il giocatore 1 preme il tasto invio per terminare la mossa
2	Sistema:	Il sistema evidenzia all'interno dello stesso scenario il robot che può controllare il giocatore 2
3	Giocatore 2	Il giocatore 2 può effettuare una mossa sulla stessa macchina

I Scenario/Flusso di eventi Alternativo: il giocatore 2 non ha a disposizione robot combattenti		
2a.1	**Sistema:**	Il sistema mostra un messaggio che comunica la fine della partita
Special Requirements	NA	

Questo use case fa riferimento al requisito funzionale RF_16 che dovrà essere implementato

Identificativo UC_RF _16	Fine mossa online	Data	09/10/21
		Vers.	1.0
		Autore	Antonio Martella
Descrizione	Lo UC descrive la funzionalità di terminare una mossa dando il controllo all'altro utente che sta giocando da remoto		
Attori Principali	**Giocatore1:** Il giocatore1 vuole passare il controllo al giocatore2		
Attori secondari	**Giocatore2:** Il giocatore2 attende la fine della mossa del giocatore 1		
Entry Condition	UC_RF_15		
Exit condition On success	Il sistema passa il controllo al giocatore 2 evidenziando il robot che dovrà controllare		
Exit condition On failure	NA		
Rilevanza/User Priority	NA		
Frequenza stimata	NA		
Extension point	NA		
Generalization of	NA		

FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO		
1	Giocatore1	Il giocatore1 preme il tasto invio per terminare la mossa
2	Sistema:	Il sistema evidenzia sulla macchina del giocatore1 il robot che dovrà controllare il giocatore2
3	Sistema	Il sistema aggiorna lo scenario sulla macchina del giocatore 2 e il robot che può controllare
4	Giocatore2	Il giocatore 2 può effettuare una mossa sulla sua macchina

I Scenario/Flusso di eventi Alternativo: il giocatore 2 non ha a disposizione robot combattenti		
2a.1	**Sistema:**	Il sistema mostra sulle macchine di entrambi i giocatori un messaggio che comunica la fine della partita
Special Requirements	NA	

Questo use case fa riferimento al requisito funzionale RF_17 che dovrà essere implementato

Identificativo UC_RF _17	Abbandona partita online	Data	09/10/21
		Vers.	1.0
		Autore	Antonio Martella
Descrizione	Lo UC descrive la funzionalità di terminare di abbandonare una partita contro un utente che sta giocando da remoto		
Attori Principali	**Giocatore1:** Il giocatore1 vuole abbandonare la partita		
Attori secondari	**Giocatore2:** Il giocatore2 attende la fine della mossa del giocatore 1		
Entry Condition	UC_RF_15		
Exit condition On success	Il sistema comunica al giocatore2 che ha vinto		
Exit condition On failure	NA		
Rilevanza/User Priority	NA		
Frequenza stimata	NA		
Extension point	NA		
Generalization of	NA		

FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO		
1	Giocatore1	Il giocatore1 clicca su "opzioni" nella barra del menù e successivamente su "abbandona partita"
2	Sistema:	Il sistema mostra sulla macchina del giocatore2 un messaggio che comunica la sua vittoria

Special Requirements	NA

Questo use case fa riferimento al requisito funzionale RF_18 che dovrà essere implementato

Identificativo UC_RF _18		Salva partita	Data	09/10/21
			Vers.	1.0
			Autore	Antonio Martella
Descrizione		Lo UC descrive la funzionalità di salvare una partita giocata in locale		
Attori Principali		**Giocatore1/ Giocatore2:** Un giocatore vuole salvare la partita		
Attori secondari				
Entry Condition		UC_RF_3		
Exit condition On success		Il sistema comunica che la partita è stata salvata		
Exit condition On failure		NA		
Rilevanza/User Priority		NA		
Frequenza stimata		NA		
Extension point		NA		
Generalization of		NA		
FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO				
1	Giocatore1/ Giocatore2	Il giocatore clicca su "opzioni" nella barra del menù e successivamente su "salva partita"		
2	Sistema:	Il sistema sovrascrive l'ultimo salvataggio e mostra un messaggio che comunica l'avvenuto salvataggio		
Special Requirements		NA		

Questo use case fa riferimento al requisito funzionale RF_19 che dovrà essere implementato

Identificativo UC_RF _19	Fine mossa online	Data	09/10/21
		Vers.	1.0
		Autore	Antonio Martella
Descrizione	Lo UC descrive la funzionalità di caricare l'ultima partita salvata in locale		
Attori Principali	**Giocatore1/ Giocatore2:** I giocatori vogliono riprendere a giocare l'ultima partita salvata		
Attori secondari			
Entry Condition	Esiste il salvataggio di una partita		
Exit condition On success	Il sistema mostra lo scenario della partita precedentemente salvata		
Exit condition On failure	NA		
Rilevanza/User Priority	NA		
Frequenza stimata	NA		
Extension point	NA		
Generalization of	NA		

FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO		
1	Giocatore1/ Giocarore2	Il giocatore visualizza la home page e clicca su "Carica partita"
2	Sistema:	Il sistema mostra lo scenario della partita precedentemente salvata
3	Giocatore1/ Giocarore2	Il giocatore che aveva il controllo nella partita salvata può effettuare la mossa

I Scenario/Flusso di eventi Alternativo: non esiste nessun salvataggio		
2a.1	**Sistema:**	Il sistema mostra sulle macchine di entrambi i giocatori un messaggio che comunica la fine della partita

Special Requirements	NA

4.5 Sequence diagram

Questo sequence diagram fa riferimento allo UC_RF _15 che riguarda l'inizio di una partita online.

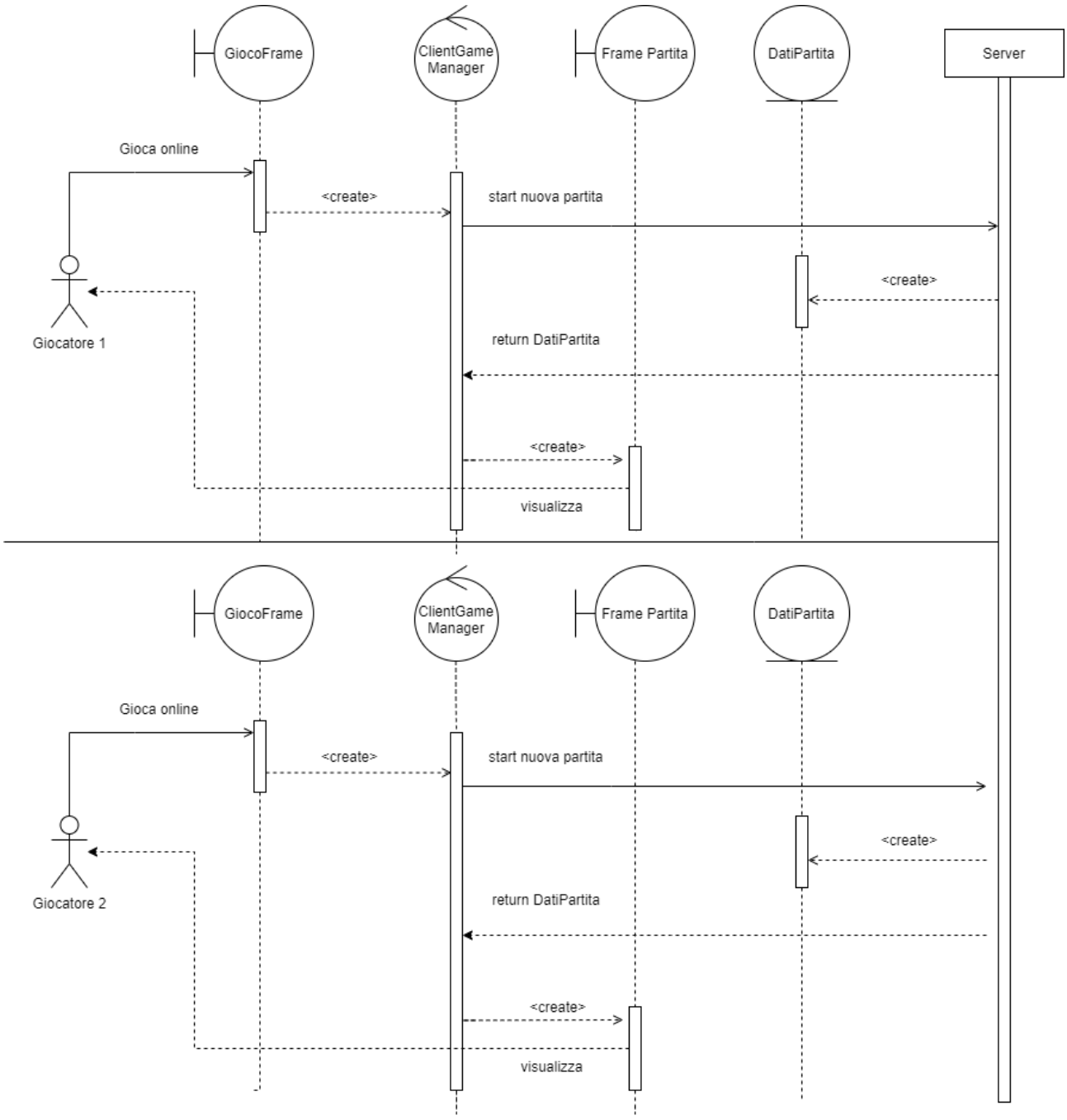

Figura 8 - sequenze diagram 1

Questo sequence diagram fa riferimento allo UC_RF _16 che riguarda la fine di una mossa di una partita online.

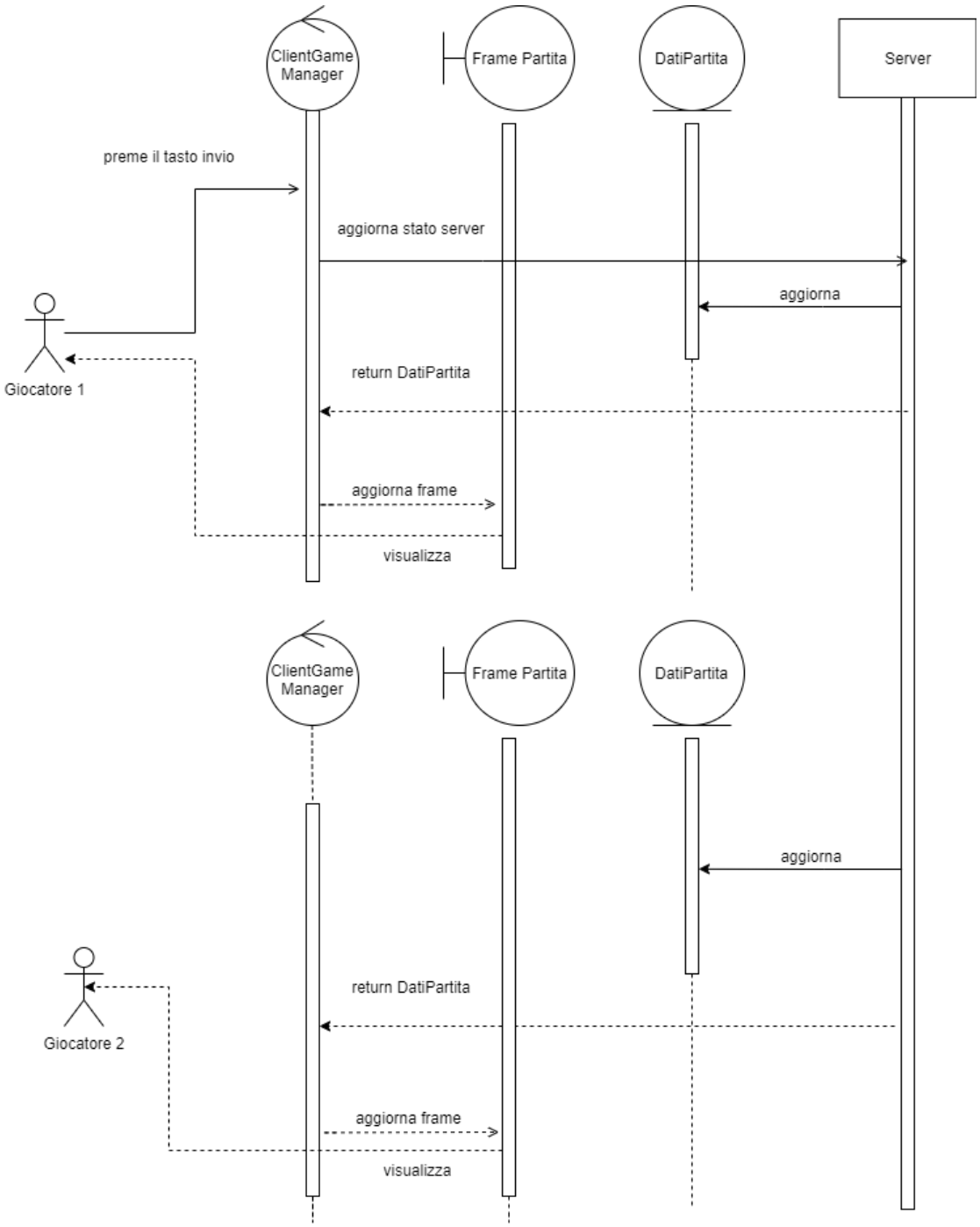

Figura 9 - sequenze diagram 2

4.6 Impatto sulle classi esistenti

La creazione delle classi ClientGameManager.java, Server.java, DatiPartita.java e delle interfacce remote che serviranno lato server e lato client, dovrebbero avere un impatto con alcune classi esistenti che dovranno essere modificate.

Nella classe GiocoFrame.java bisognerà implementare i pulsanti e i relativi ActionListener con il quale l'utente potrà iniziare una partita online oppure caricare una partita salvata. Per quanto riguarda la partita online, la classe GiocoFrame.java dovrà istanziare anche la nuova classe ClientGameManager.java che dovrebbe implementare i metodi dichiarati nei paragrafi precedenti.

Per quanto riguarda il caricamento di una partita salvata, bisognerà modificare l'attuale classe GameManager.java che dovrà anche poter caricare da un file un'istanza della nuova classe DatiPartita.java, inoltre, per permettere il salvataggio di una partita e quini creare un metodo che salvi un oggetto DatiPartita, dovrà anche creare nel frame dello scenario un'opzione che permetta all'utente di cliccare e salvare la partita. Inoltre, bisogna disabilitare la modalità Giocatore1 vs computer in quanto non verrà presa in considerazione.

Per quanto riguarda il Server, considerando che verrà implementato attraverso l'utilizzo di Java RMI, verrà creato un progetto a parte che avrà al suo interno oltre la classe Server.java, DatiPartita.java e le interfacce remote, anche una copia delle classi che vengono salvate all'interno dell'oggetto DatiPartita, di conseguenza, considerando che questo oggetto viene "passato" tra client e server da remoto, tutte le classi al suo interno dovranno essere rese serializable (anche nel progetto dove è presente il client).

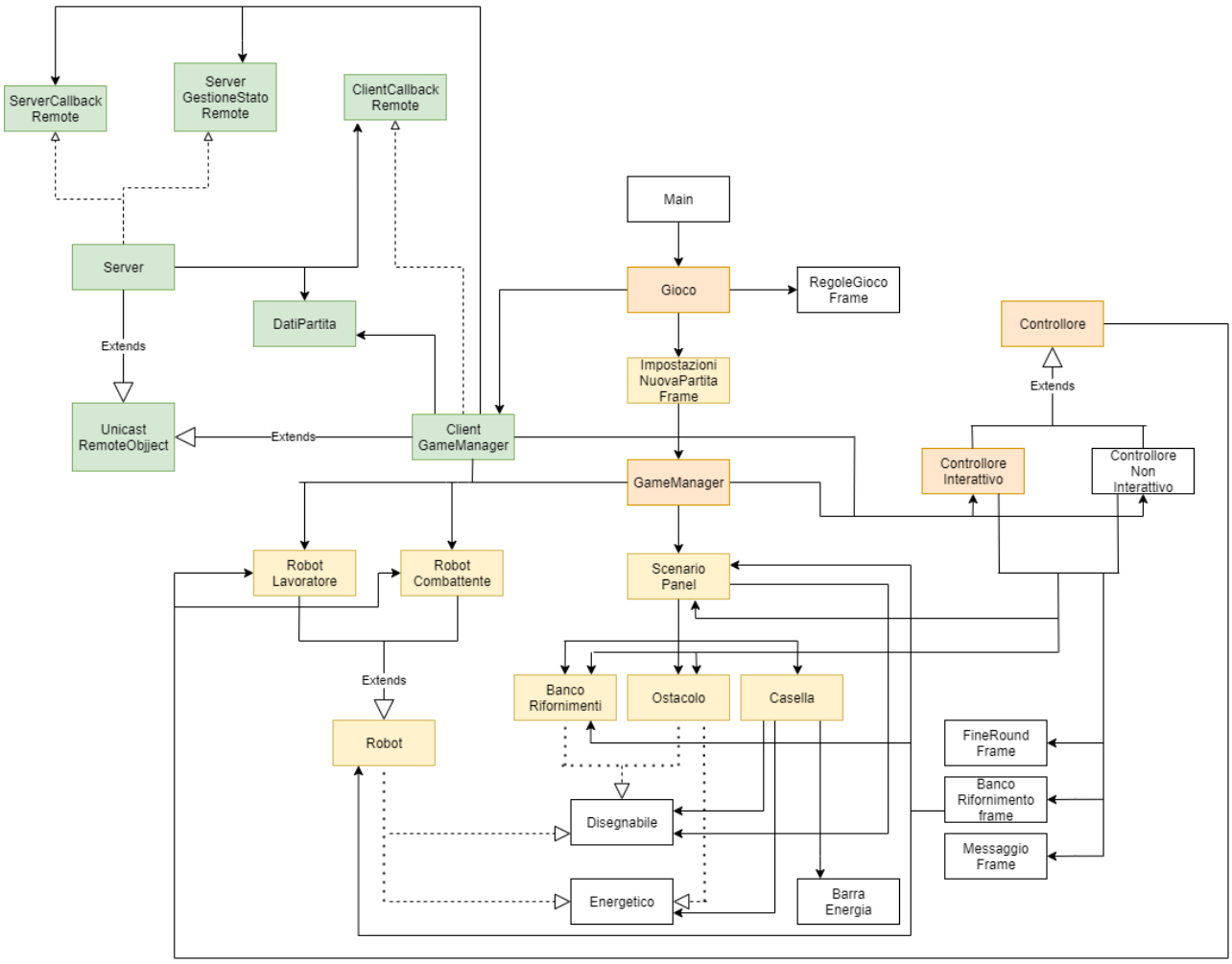

Figura 10 – UML aggiornato

5. Report modifiche

Sono stati implementati tutti i nuovi requisiti funzionali elencati nei capitoli precedenti.
Di seguito, sono riportate nello specifico le modifiche effettuate:

1) Nella classe Casella.java sono stati aggiunti i metodi getTipoScenario(), isDrawOggetto(), isDrawEnergia(), isRiempiCasella(), getColCas() che sono serviti per il testing iniziale.
2) Nella classe BarraEnergia.java sono stati aggiunti i metodi getEnergia(), getmaxenergia() che sono serviti per il testing iniziale.
3) Nella classe ImpostazioniNuovaPartitaFrame.java è stata disabilitata la scelta della modalità "giocatore vs computer" nel metodo creaSceltaModalità().
4) Aggiunta la classe DatiPartita.java con i metodi previsti nell'impact analysis.
5) Nella classe GameManager.java è stato aggiunto il metodo startPartitaSalvata() ed è stato aggiunto un ActionListener "clickSalvaPartita". All'interno del metodo creaMenùBar() è stato creato un item per salvare la partita al quale gli viene associato ActionListener "clickSalvaPartita".
6) Nella classe ControlloreInterattivo.java il KeyListener per permettere di compiere un'azione al giocatore tramite la tastiera veniva associato in uno dei metodi di inizializzazione. Di conseguenza, all'interno del controllore preso dall'oggetto DatiPartita caricato da un file non era presente nessun riferimento al KeyListener. Stesso discorso vale per il passaggio dello stesso oggetto da client-server e viceversa. Di conseguenza è stato aggiunto il metodo aggiungiKeyListner() nella classe ControlloreInterattivo.java. Questo metodo va richiamato ogni volta che si caricano i dati di una partita. Queste considerazioni non erano state fatte nella fase di impact analysis.
7) Rese serializzabili le classi DatiPartita.java, BancoRifornimenti.java, Ostacolo.java, Scenario.java, Casella.java., BarraEnergia.java, Robot.java
8) Aggiunto serialVersionUID nella classe ControlloreInterattivo.java

9) Creata interfaccia remota ClientCallbackRemote che ha, oltre ai metodi descritti nell'impact analysis, anche il metodo comunicaServerOccupato() che viene implementato nella classe ClientGameManager.java e viene chiamato dal Server quando un client prova a connettersi ma il Server gestisce già una partita.
10) Creata interfaccia remota ServerCallbackRemote che ha, oltre ai metodi descritti nell'impact analysis
11) Creata interfaccia remota ServerGestioneStatoRemote che ha, oltre ai metodi descritti nell'impact analysis
12) Creata la classe ClientGameManager.java che implementa l'interfaccia remota ClientCallbackRemote. Implementati i metodi previsti nell'impact analysis. All'interno del metodo creaMenùBar() è stato creato un item per l'opzione "abbandona partita" al quale gli viene associato ActionListener "clickAbbandona". Considerando che la partita termina quando un giocatore non ha più RobotCombattenti, quando l'utente clicca sull'item appena descritto, viene svuotato l'arrayList di RobotCombattenti e poi richiamato il metodo aggiornaStatoServer(datiPartitaServer) implementato dal Server.
13) Nella classe GiocoFrame.java sono stati aggiunti i metodi caricaPartitaPanel() e giocaOnlinePanel() che creano i pulsanti che permettono all'utente di caricare una partita e di giocare online. Aggiunti i relativi ActionListener come previsto nell'impact analysis. Modificate le dimensioni del frame.
14) Implementata la classe Server.java all'interno di un nuovo progetto che ha al suo interno la copia delle classi che servono agli oggetti remoti. La classe server.java implementa le interfacce remote ServerCallbackRemote, e ServerGestioneStatoRemote. Implementati tutti i metodi previsti nell'impact analysis.
15) Creata configurazione "rmic" che esegue rmic.exe all'interno della cartella dei progetti per generare lo stub e lo skeleton.
16) Creata configurazione "RMIregistry" che esegue rmiregistry.exe all'interno della cartella dei progetti per fornire il servizio di naming che permetterà al client di connettersi al server.

17) Risolto bug evidenziato nel testing di sistema. Quando il giocatore1 distruggeva un robot del giocatore2, questo veniva subito eliminato nell'ArrayList di robot dal controllore1 mentre nel controllore2 veniva eliminato solo nel momento in cui veniva selezionato per un'ipotetica mossa. Questo era un problema nel momento in cui un giocatore non aveva più robot combattenti disponibili in quanto in alcuni casi la fine della partita non veniva decretata all'istante. Di conseguenza, è stato risolto questo bug implementando il metodo aggiornaArrayRobotCombattenti() nella classe Controllore.java che viene richiamato all'inizio del metodo cambiaRobot() ogni volta che il controllore inizia a controllare un nuovo robot.

6. Sviluppi futuri

In questo progetto è stata aggiunta un'architettura base client-server per giocare una partita in modo concorrente e per salvare e riprendere a giocare una partita in un secondo momento.
Partendo da queste basi è possibile implementare la gestione di più partite contemporaneamente e di più salvataggi.
Inoltre, nelle modifiche apportate non è stata data molta importanza alla grafica che può essere sicuramente migliorata.