Eriantys
(Protocollo di comunicazione)

Samuele Scherini
Matteo Spreafico
Ludovica Tassini

1 Luglio 2022

Indice

1 Generalità

Il seguente protocollo di comunicazione descrive i percorsi che tutti i messaggi che attraversano la rete devono intraprendere, affinché le azioni che gli utenti vorrebbero eseguire sul gioco vengano effettivamente svolte dal server e mostrate ai client. Di tutti gli intermediari presenti negli scambi di messaggi tra client e server, quali *virtual view* e *socket handlers*, si evidenzierà nel seguito quello maggiormente degno di nota, ovvero il *ClientController*. Tale classe si occupa infatti di:

- confezionare l'input fornito dagli utenti in un messaggio che attraversi la rete.
- ricevere le risposte del server e chiamare i metodi corretti della *view*.

1.1 Serializzazione

Per lo scambio di messaggi tra il client e il server - che rimane *passivo*, cioè ascolta e al più risponde, il gruppo ha optato per la serializzazione Java. Tale serializzazione viene resa possibile dal fatto che le classi che dovranno essere incapsulate in messaggi da scambiare in rete implementano l'interfaccia *Serializable*.

1.2 Messaggi

In rete, vengono scambiati diversi tipi di messaggi per il corretto funzionamento del software. Le classi corrispondenti a tutti i tipi di messaggi scambiati ereditano dalla classe *Message*, il che implica che ognuno di loro possiede nativamente un attributo di tipo *String* che contiene il nickname del client che manda il messaggio (qualora il messaggio non neccesitasse di un client associato, come l'*ErrorMessage*, tale valore viene posto a *null*).

Quando un server risponde, potrebbe utilizzare un messaggio di una particolare classe, denominata *AskMessage*. Tale messaggio viene usato se il server ha bisogno di un altro input, e si differenzia dagli altri perché possiede un "tipo", proveniente da un'apposta enumerazione (*AskType*), che viene assegnato in fase di creazione del messaggio in base al contesto di gioco in cui ci si trova. Ad esempio, se si ha appena spostato Madre Natura, il server risponderà con un messaggio in cui chiede quale nuvola si desidera scegliere, e dunque l'AskType sarà *CLOUD_CHOICE*. È proprio in base a quel tipo che il ClientController decide quale metodo della *view* eseguire.

Si tenga presente che il server accetta sempre messaggi in arrivo dai client, anche se giungono in momenti inattesi (si fa riferimento all'unico contesto in cui sono ammissibili messaggi "inattesi" - ovvero durante una partita di gioco, a causa della natura *turn based* di Eriantys). L'implementazione, sia della CLI che della GUI, dovrebbe impedire che messaggi corretti vengano effettivamente inviati nei momenti sbagliati, ma anche qualora questa premessa non dovesse risultare vera - per la presenza di qualche bug, il *controller* è stato programmato affinché risponda con:

- *WrongTurnException* se un messaggio correttamente incapsulato viene mandato da un giocatore che non è quello corrente.
- *WrongMessageSentException* se un messaggio correttamente incapsulato viene mandato in un momento inatteso (ad esempio, un *CloudChoiceMessage* durante la fase pianificazione).

2 Scenari

Ora verranno approfonditi, con degli appositi *sequence diagram* accompagnati da un commento testuale, i possibili scenari in cui un client incapsula messaggi contenenti input forniti dall'utente per poi inviarli al server. Si noti che, nelle successive rappresentazioni:

- con *client* si intende una delle possibili interfacce che possono interagire con l'utente (nel caso di Eriantys, CLI o GUI).
- con *server* si intende una macchina che esegue operazioni sul modello di gioco e che le notifica tramite la rete. Dunque, non si vuole sottointendere che l'oggetto che si interfaccia direttamente con un'istanza della classe *ClientController* sia un'istanza della classe *Server*, ma si vogliono soltanto mettere in evidenzia i metodi che verranno chiamati per eseguire gli effettivi aggiornamenti.

2.1 Connessione alla lobby

Il client richiede di instaurare una connessione col server una volta forniti il suo indirizzo IP nella rete e la porta su cui è in ascolto. Se i valori forniti corrispondono a quelli di un server presente sulla rete, la connessione viene isntaurata; altrimenti, il client chiede all'utente di riprovare. Una volta che la connessione è stata stabilita, il server chiederà all'utente di inserire un *nickname* con il quale verrà identificato d'ora in avanti. Se tale nickname è unico tra quelli presenti nel server, il client entra nella lobby con quel nickname; altrimenti, il server chiede all'utente di riporvare inserendo un nuovo nickname.

Per collegarsi al server, il client costruisce una sua socket. Per comunicare invece il nickname scelto, viene creato un messaggio denominato *LoginRequest*. Se il server deve richiedere il nickname a causa della sua non unicità, invia un *AskMessage* di tipo *NICKNAME_NOT_UNIQUE*.

2.2 Connessione a una partita

Una volta entrato nella lobby, l'utente può decidere se creare una nuova partita oppure unirsi a una già esistente. Se decide di creare una nuova partita, l'utente dovrà:

- fornire un identificativo numerico intero del gioco da creare, tale che sia unico tra gli identificativi presenti sul server.
- fornire un numero valido di giocatori che potranno unirsi alla partita (nel caso in questione, 2 o 3).
- indicare se la partita da creare dovrà seguire le regole normali o quelle per esperti.

Se invece l'utente decide di unirsi a una partita già esistente, dovrà unicamente fornire l'identificativo del gioco a cui vuole unirsi. Se l'identificativo corrisponde a una partita esistente e tale partita ha ancora dei posti disponibili, l'utente potrà unirsi; altrimenti, l'utente dovrà scegliere nuovamente cos'ha intenzione di fare.
Qualunque strada si sia scelta, una volta entrati in una partita, l'utente dovrà selezionare un mago (*wizard*) da impersonare, tale che non sia stato scelto da nessun altro giocatore. In caso venga scelto un mago precedentemente selezionato da qualcun altro, all'utente verrà chiesto di sceglierne un altro.

Una volta che il client ha fornito al server un nickname valido, il server risponderà con un *ShowExistingGamesMessage*, per mostrare all'utente gli identificativi delle partite presenti sul server (indicando anche se accettano nuovi giocatori oppure no). Per comunicare i parametri di una partita da creare, viene incapsulato un *CreateGameMessage*; per comunicare l'identificativo di una partita esistente, viene incapsulato un *JoinGameMessage*. Qualsiasi sia il messaggio inviato, il server risponderà con un *AskMessage* di tipo *GAME_ID* se i parametri forniti non sono corretti; altrimenti, rispinderà con un *AskMessage* di tipo *WIZARD_ID*
Per comunicare l'identificativo del mago scelto, viene incapsulato un *WizardIDMessage*. Se l'identificativo comunicato non è corretto, il server risponderà nuovamente con un *AskMessage* di tipo *WIZARD_ID*

Ciò che accade a seguito della comunicazione di un WizardID unico dipende da alcuni fattori. Si delineano le tre alternative seguenti:

- il server risponde con un *GenericMessage* per comunicare all'utente di aspettare che la partita completi la propria coda di giocatori.
- se la coda è completa, risponde con un *ShowGameStatusMessage* che mostra a tutti i giocatori lo stato della *board* di gioco.
- se l'utente è il giocatore corrente, il server risponde anche con un *AskMessage* di tipo *ASSISTANT_CARD* e con un *ShowDeckMessage*, affinché l'utente possa visualizzare le carte assistente che ha a disposizione e sceglierne una.

2.3 Giocare una partita con le regole normali

All'inizio di un round, ad ogni ClientController viene inviato un *AskMessage* di tipo *ASSISTANT_CARD*, pertanto al client verrà chiesto di scegliere una carta assistente. Per comunicare al server la carta assistente scelta dal client, il ClientController incapsula un messaggio di tipo *AssistantCardMessage*.
Dopo che la fase pianificazione è terminata per ogni client, viene inviato al ClientController un *AskMessage* di tipo *MOVE_STUDENT*. Per comunicare al server lo studente che si vuole spostare e, soprattutto, la destinazione verso la quale lo si vuole spostare, viene inviato dal ClientController uno tra:

- *MoveToIslandMessage*, se si vuole spostare lo studente verso un'isola scelta.
- *MoveToTableMessage* se si vuole spostare lo studente verso un tavolo.

Dopo aver fatto scegliere al client tutti gli studenti che poteva spostare e dove spostarli, viene inviato al ClientController un *AskMessage* di tipo *MOVE_MOTHER_NATURE*. Per comunicare al server di quanti passi l'utente vuole far spostare Madre Natura, il ClientController incapsula un messaggio di tipo *MotherNatureStepsMessage*.
Dopodiché, prima di far iniziare la fase azione del giocatore successivo, viene inviato al ClientController un *AskMessage* di tipo *CLOUD_CHOICE*. Per comunicare al server la nuvola scelta dal client, viene inviato un messaggio di tipo *CloudChoiceMessage*.
Infine, si noti che:

- dopo ogni mossa, viene inviato al ClientController un messaggio di tipo *ShowGameStatusMessage*, che serve per aggiornare i giocatori sullo stato della board di gioco.
- qualora una mossa dovesse fallire, il processo di richiesta di input viene ripetuto esattamente come descritto.

2.4 Giocare una partita con le regole da esperti

Con le regole per esperti, i primi momenti della fase azione di un giocatore subiscono una variazione.
Non appena il ClientController ha comunicato al server la carta assistente scelta dal client, viene inviato al ClientController un *AskMessage* di tipo *ACTION_CHOICE*. Per comunicare la scelta fatta dal client (ovvero se giocare una carta personaggio oppure se spostare uno studente), viene inviato al server un *ActionChoiceMessage*. Se il client ha scelto di giocare una carta personaggio, viene inviato al ClientController un *AskMessage* di tipo *CHARACTER_CARD*. In base alla carta personaggio scelta e al tipo di input che questa richiede (se lo richiede), il ClientController può inviare diversi tipi di messaggio al server:

- se la carta personaggio non richiede input, viene inviato al server un *CharacterCardMessage*.
- se la carta personaggio richiede un parametro intero, viene inviato al server un *CharacterCardMessageInt*.
- se la carta personaggio richiede un parametro testuale, viene inviato al server un *CharacterCardMessageString*.
- se la carta personaggio richiede un parametro testuale e un intero, viene inviato al server un *CharacterCardMessageStringInt*.
- se la carta personaggio richiede una lista di parametri testuali, viene inviato al server un *CharacterCardMessageArrayListString*.

Invece, se l'utente sceglie di spostare uno studente, viene inviato al ClientController (come visto in precedenza) un *AskMessage* di tipo *MOVE_STUDENT* e in base alla scelta fatta dal client viene inviato al server un *MoveToIslandMessage* oppure un *MoveToTableMessage*.

Nel caso l'utente giocasse con successo una carta personaggio, il flusso di azioni successivo sarebbe equivalente a quello di un turno con le regole normali. Perciò, il server, sia dopo l'utilizzo della carta che dopo lo spostamento di ogni studente, risponderebbe con un *AskMessage* di tipo *MOVE_STUDENT* invece che con un *AskMessage* di tipo *ASK_ACTION*.

2.5 Altri scenari

Esistono altri scenari in cui il server può rispondere al client con particolari tipi di messaggi. I messaggi in questione sono:

- *UpdateMessage*. Questo messaggio viene inviato al ClientController dei giocatori che stanno aspettando il proprio turno, per comunicare le azioni fatte dal giocatore corrente.
- *GenericMessage*. Questo messaggio viene usato per comunicare ai client messaggi generici. Ad esempio, ogni qual volta viene fatta dal client una mossa non permessa, viene inviato al ClientController un *GenericMessage* con un avviso che specifica perché tale azione non è permessa. Dopo la ricezione del messaggio, viene data la possibilità al giocatore di riprovare a compiere tale mossa.

- *GameStatusFirstActionPhaseMessage*. Questo messaggio viene usato per comunicare lo stato corrente della board di gioco a tutti i client. È una versione speciale del *GameStatusMessage* creata specificamente per permettere alle CLI di mostrare le informazioni necessarie in modo più conveniente.
- *GamePhaseMessage*. Questo messaggio viene usato per comunicare al client un cambio di fase durante la partita. Viene adoperato dalla CLI.
- *ErrorMessage*. Questo messaggio viene usato per comunicare un generico errore al client.
- *EndGameMessage*. Questo messaggio viene usato per comunicare ai client che la partita è finita.
- *DisconnectionMessage*. Questo messaggio viene usato per comunicare ai client connessi che un giocatore si è disconnesso.
- *PlayerNumberMessage*. Questo messaggio viene usato per comunicare al server quanti giocatori dovrà avere la partita appena creata. Viene usato solo per ragioni di testing.

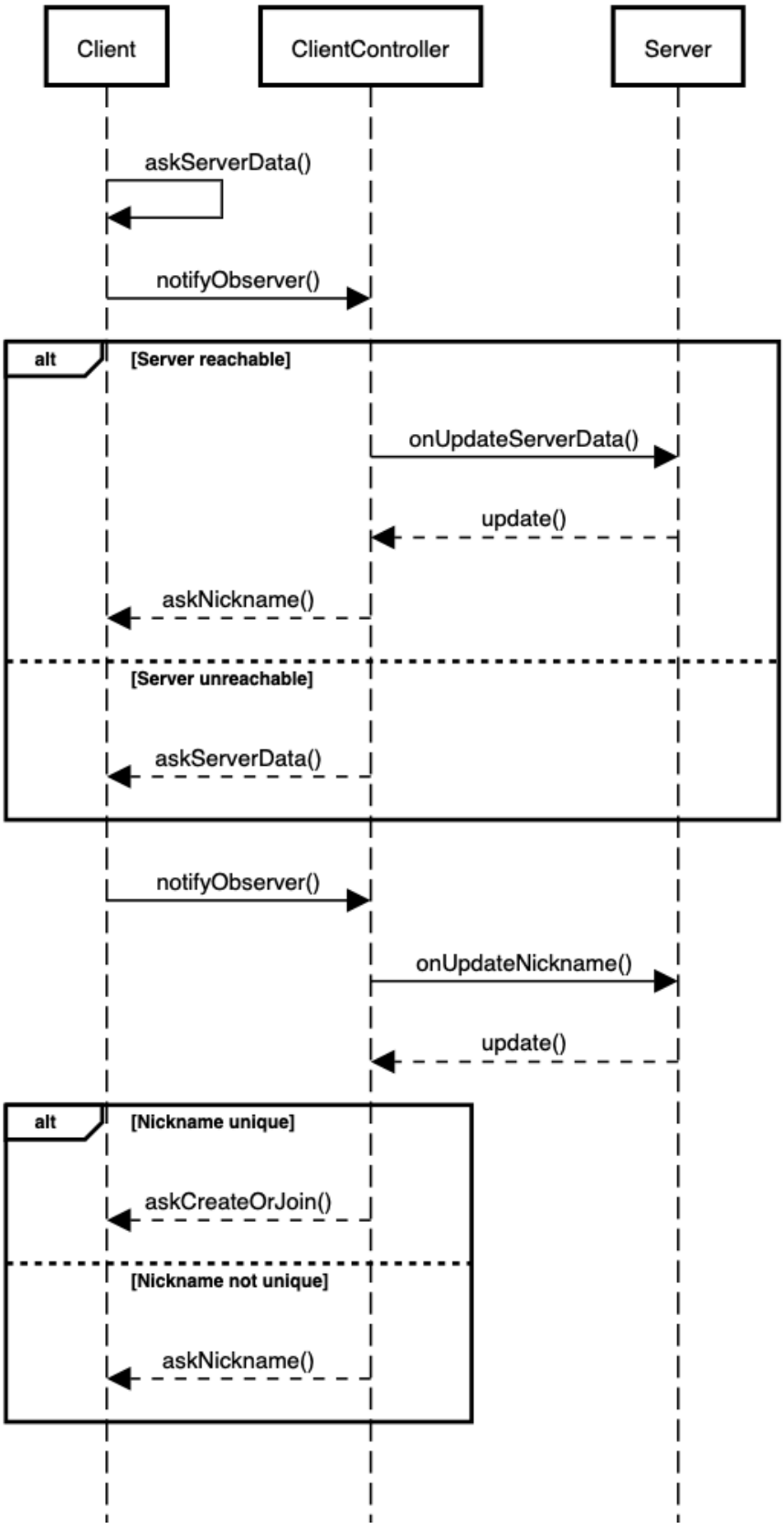

Figura 1: *sequence diagram* della connessione alla lobby

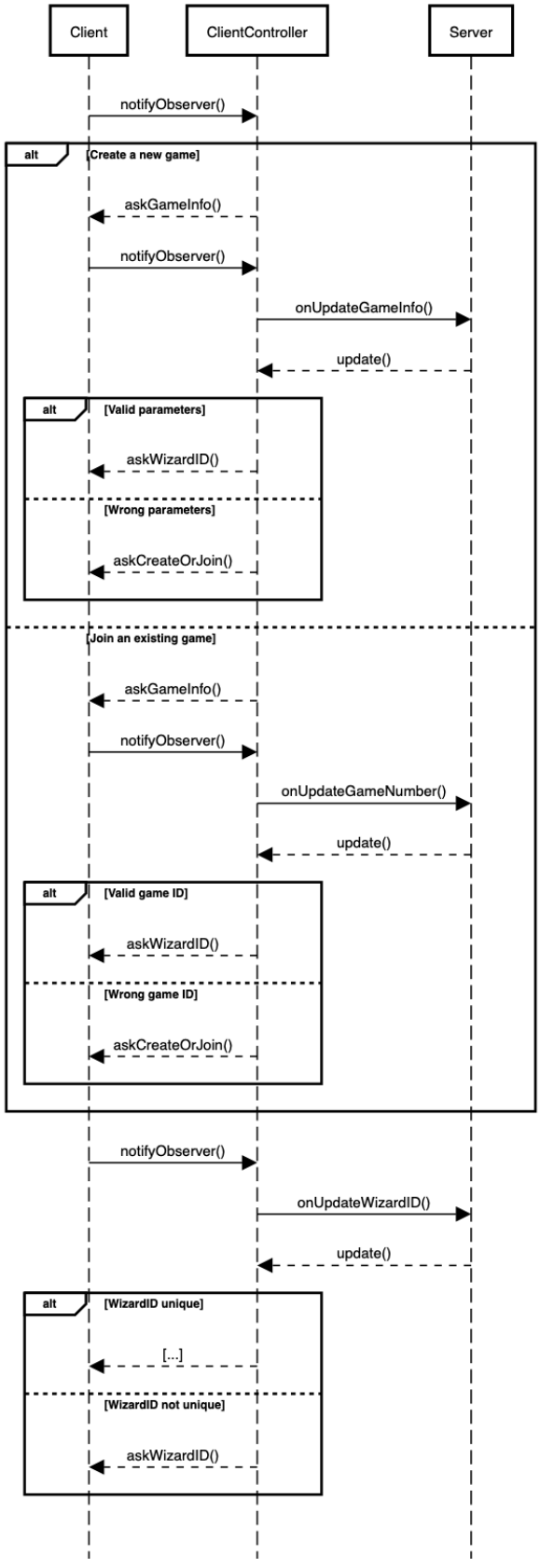

Figura 2: *sequence diagram* della connessione a una partita

Figura 3: *sequence diagram* di un turno con le regole normali

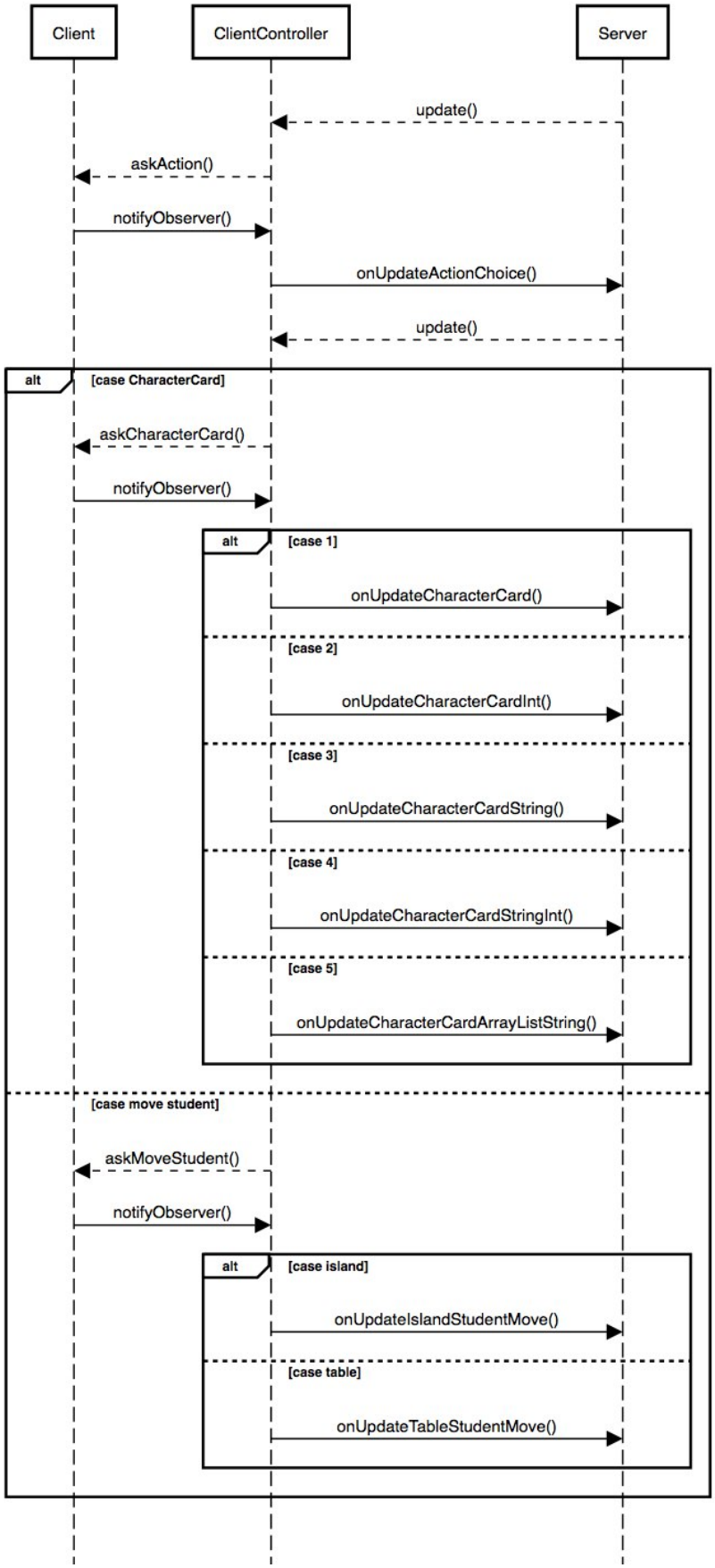

Figura 4: variazione del *sequence diagram* di un turno con le regole da esperti