DeadLine 14/05/19
Comunicazione Client-Server e Rete

AM 31

Scelte implementative

La gestione della fase della comunicazione tra client e server è implementata mediante lo sdoppiamento della view, intesa come modulo del pattern MVC adottato, in view remota (server side) e view effettiva (client side).

La view remota rappresenta il canale di comunicazione che il server ha con i client ed è rappresentata dalla classe *ClientConnection.* Allo stesso modo la view effettiva è il canale di comunicazione che il singolo client ha con il server ed è rappresentata dalla classe *ServerConnection.*

Le due classi di connessione vengono ereditate dalle sottoclassi che implementano i due protocolli comunicativi adottati nel progetto: socket e RMI. Le sottoclassi saranno quindi: *SocketClientConnection, RMIClientConnection*, *SocketServerConnection e RMIServerConnection.*

Socket

La classe *SocketClientConnection* è il thread che contiene gli stream di lettura e scrittura della client socket. Il thread è lanciato al connettersi del client con il server ed ha lo scopo di ascoltare ciclicamente il client e ricevere gli eventi diretti al controller o al server stesso. Allo stesso modo la classe *SocketServerConnection* ascolta ciclicamente il server e riceve gli eventi provenienti da tale server e passarli alla view. Entrambe le classi offrono i metodi per mandare gli eventi nelle due direzioni (Model to View, View to Controller).

RMI

A differenza della tecnologia Socket, le due classi *RMIClientConnection e RMIServerConnection* non sono thread ma sono oggetti remoti esportati rispettivamente dal server e dal client. Entrambe espongono, mediante la propria interfaccia che estende Remote, i metodi per inviare e ricevere eventi nelle due direzioni.

EVENTI

La soluzione scelta per gestire i messaggi scambiati consiste nel rendere più atomici possibili gli eventi scambiati dal client e dal server. Gli eventi si distinguono in due macrocategorie: *ServerEvent* e *EventFromModel*. Inoltre, gli eventi diretti verso il server (*ServerEvent*) possono essere destinati al server stesso *(ConfigEvent*) oppure destinati al controller (*EventFromView*).

EventFromModel, ConfigEvent e EventFromView sono le 3 classi astratte che vengono ereditate dai singoli eventi, i quali vengono spediti serializzati tramite le classi di connessione presentate sopra e, in seguito, deserializzati come *ServerEvent* (lato server nella i-esima *ClientConnection*) oppure come *EventFromModel* (lato client nella *ServerConnection*).

Rappresentazione remota del model

- Queste sono le classi che si è pensato di utilizzare per mantenere i clients aggiornati del model, vengono adesso esposte per una maggiore chiarezza nella prossima sezione
- Non vengono inviati veri e propri oggetti utilizzati nel model.
- Tutte le classi sono Serializzabili e inviabili tramite appositi messaggi.
- Le classi sono principalmente composte da stringhe e interi.

1) **GameRappresentation**
   myPlayerRepresentation
   List<PlayerRepresentation> eniemies' rappresentation.
   mapRepresentation;

2) **MapRappresentation**
   mapType
   Map<Position, AmmunitionCard> ammosInPosition
   List<WeaponInfo> redWeapons
   List<WeaponInfo> blueWeapons
   List<WeaponInfo> yellowWeapons

3) **MyPlayerRappresentation**
   name
   color
   isFirstPlayer
   isActive
   position
   ammoCubes
   <WeaponInfo> weapons
   List<EmpowerInfo> empowers
   List<Color> damages
   List<Color> marks
   points
   deaths

4) **EnemyPlayerRappresentation**
   name;
   color;
   isFirstPlayer;
   isActive;
   position;
   ammoCubes;
   List<WeaponInfo> unloadedWeapons;

numWeapons;
numEmpowers;
List<Color> damages;
List<Color> marks;
points;
deaths;
deathPoints;

5) **WeaponInfo**

id
name
color
buyCost
reloadCost
loaded
List<EffectInfo> basicEffects
List<EffectInfo> ultraEffect

6) **EmpoweInfo**

id
name
type
description
color

7) **EffectInfo**

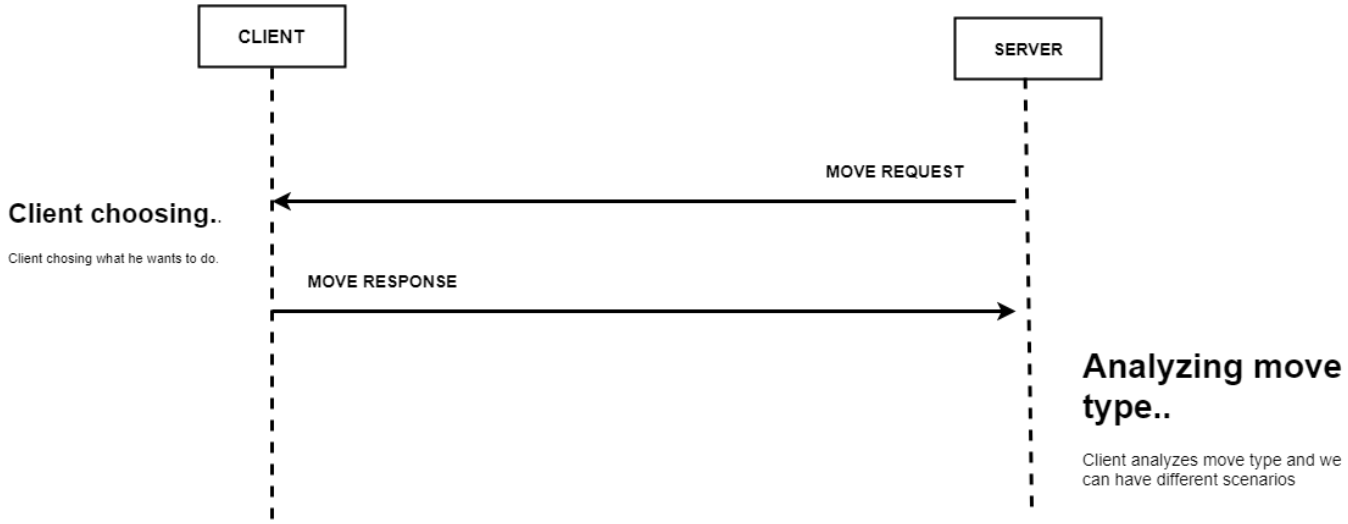
name
cost
description

Interazioni Client-Server

PREMESSA: Ogni iterazione presente nel documento è stata già implementata e funzionante, tuttavia per non appesantire troppo i grafici sono stati omessi i messaggi di aggiornamento della situazione di gioco. Questo messaggio, chiamato GameRepUpdate viene inviato dal Server al client ogni qualvolta il model venga aggiornato.

Semplificando molto, un'interazione base tra Client e Server è la seguente scelta della mossa.
Nella nostra implementazione si è pensato di far scegliere al client una mossa singola, tra quelle che il Controller ha reso disponibili. Per mossa è inteso: RACCOGLIERE DA TERRA/SPARARE/MUOVERSI DI UNA CELLA.

Il server conosce già quali mosse sono abilitate dal Client e permette di scegliere tra quelle.
Il client è così impossibilitato a scegliere qualcosa che non sia stato già inviato dal Server.



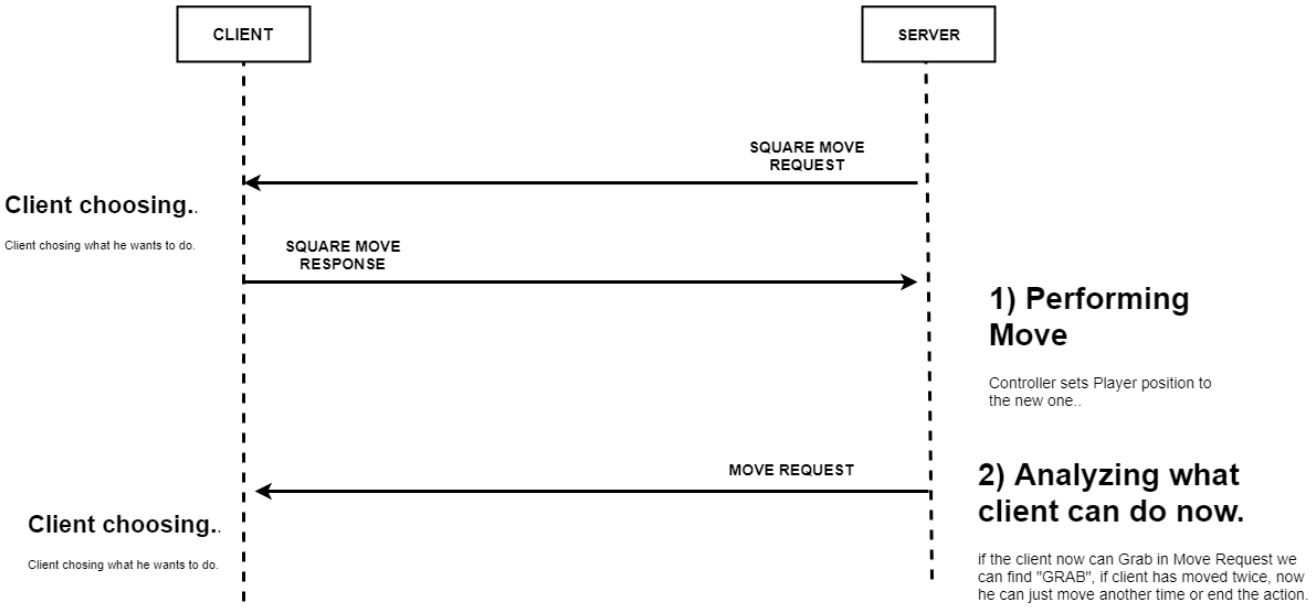
MoveResponse
MoveType
playerName

MoveRequest
List<MoveTypes>

Da qui si possono aprire diversi scenari. In base alla scelta dell'Utente

1) **Primo scenario. Utente ha selezionato di muoversi**

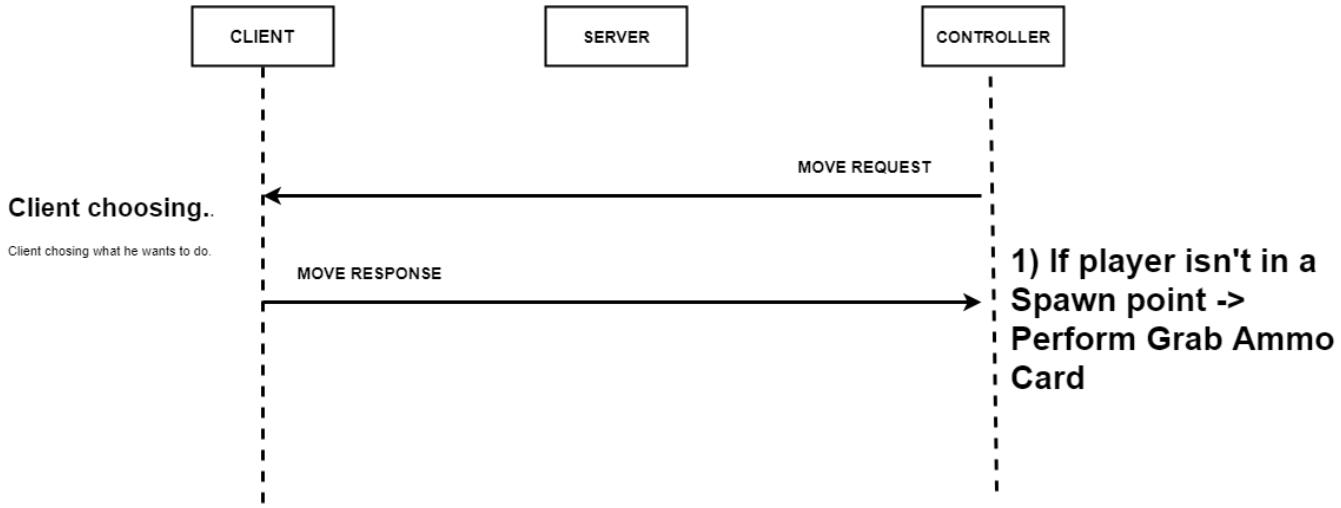
Se l'utente ha scelto di muoversi, potrà farlo tra una delle scelte proposte dal Server. Come già esposto, il server ha già deciso e inviato al client in quali posizioni può effettuare il movimento, che può essere di una cella soltanto.



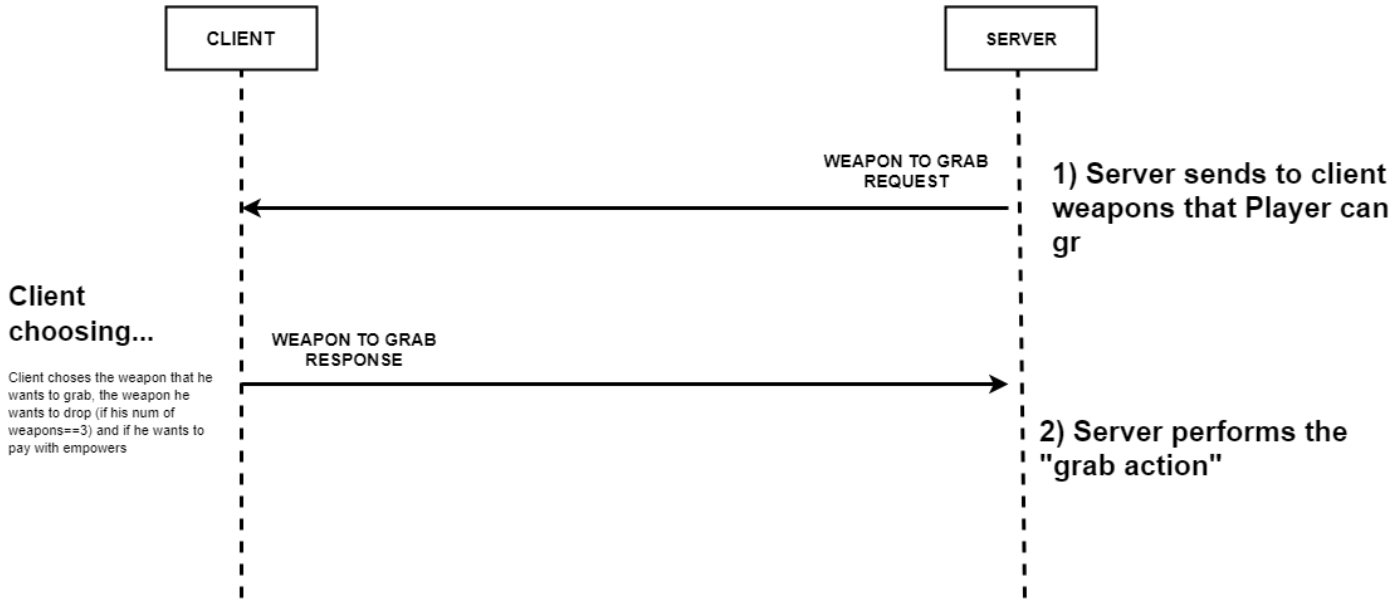
SquareToMoveResponse
position
isAborted

SquareMoveRequest
List<Positions>

1) **L'utente ha deciso di raccogliere una munizione**



2) L'utente ha deciso di raccogliere un arma



WeaponToGrabResponse
gainedWeapon
droppedWeapon
payingEmpowers

WeaponToGrabRequest
List<weaponToGrab>
List<weaponToDrop>
List<payingEmpowers>

3) L'utente ha scelto di sparare

La scelta dell'arma e dei target è articolata su più fasi.

3.1 Dapprima viene scelta l'arma con cui sparare dal Client

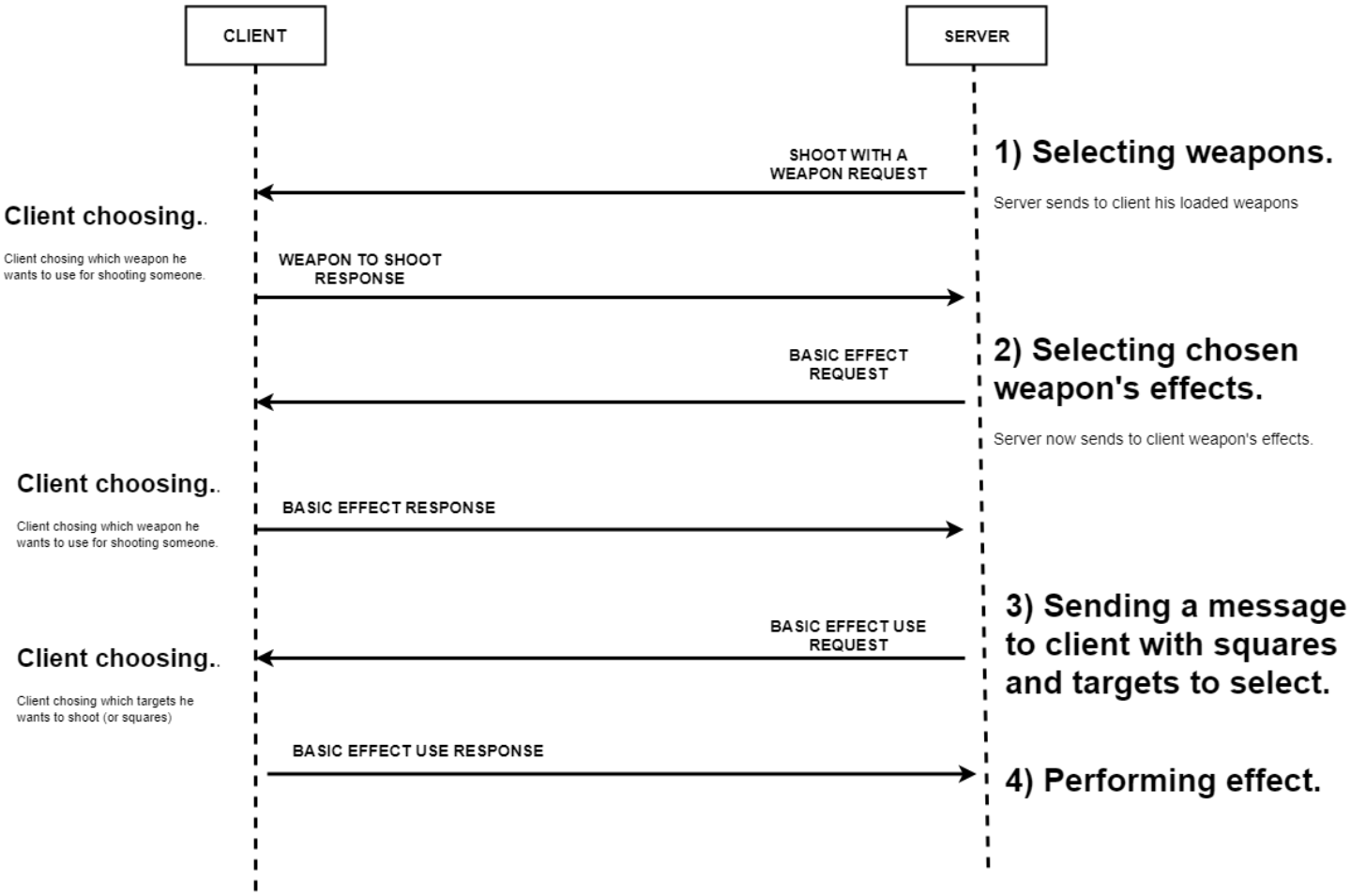
3.2 Il Server invia al Client la lista degli effetti che può performare con quell'arma. (BASIC EFFECT REQUEST)

3.3 Il Client sceglie l'effetto. (BASIC EFFECT RESPONSE)

3.4 Il server invia al client la lista di Squares e di Obbiettivi selezionabili (se un obbiettivo è presente nella lista allora è sparabile sicuramente)

3.5 Il client effettua le sue scelte e invia al server i Target selezionati e/o le Squares.

3.6 Il server tramite il controller fa effettuare al model l'azione.



BasicEffectUseResponse
playerNames
squarePositions
recoilPosition

BasicEffectUseResponse
playerNames
squarePositions
recoilPositions
maxPlayers
maxSquares