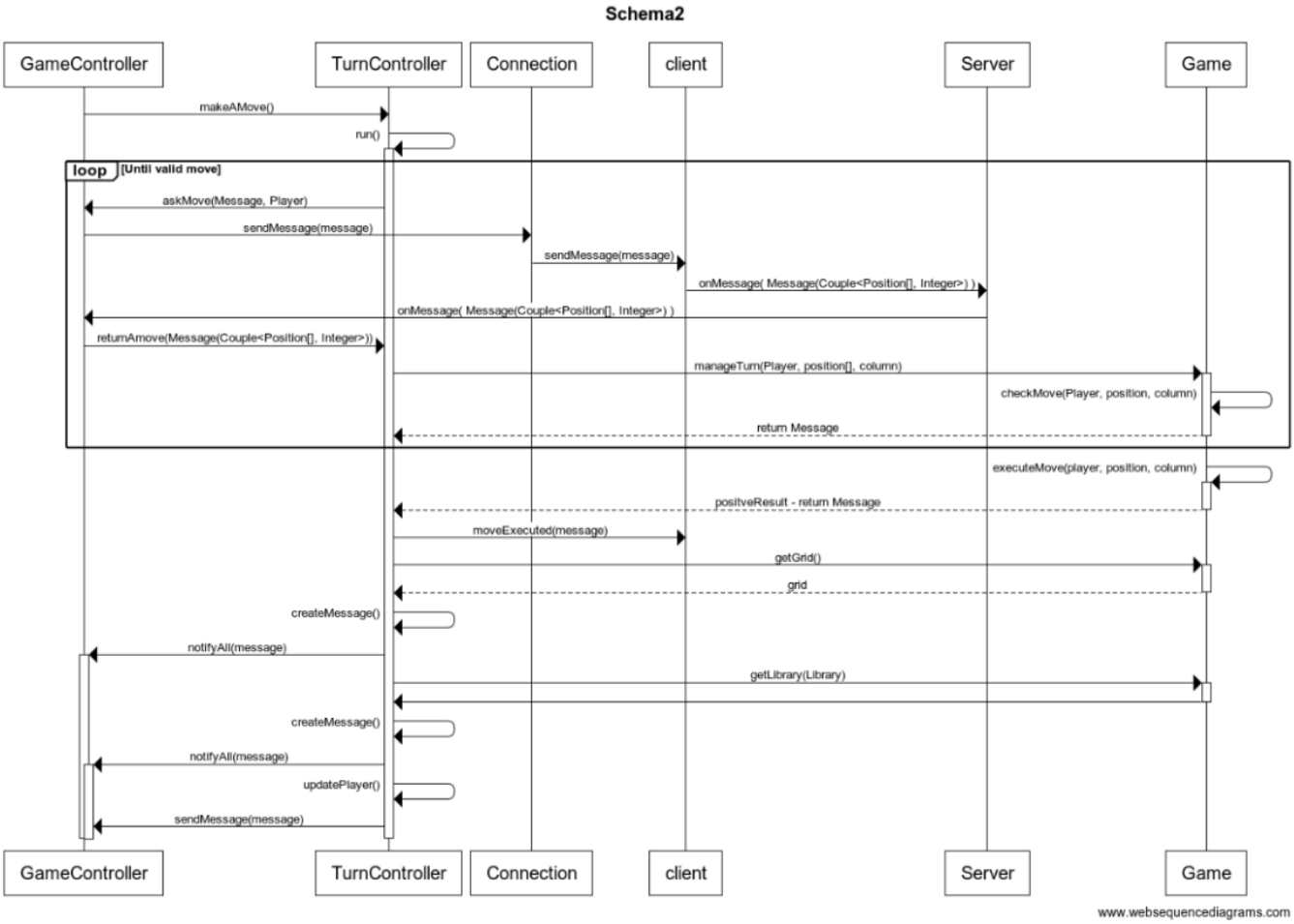
Schema: L'azione ''Selezionare le carte oggetto ed inserirle nella propria libreria"

Nei schemi successivi suppongo che tutte le classi coinvolte siano già istanziate e "pronte all'uso" nel normale svolgimento della partita. Inoltre, suppongo che non sorgano eccezioni che bloccano la partita come l'assenza della risposta del client, lo scollegamento del client o del server e l'inserimento continuo di mosse sbagliate.

LATO SERVER

Nello schema e nella spiegazione non sono presenti tutte le classi lato server coinvolte nell'architettura di rete perché non è il focus di questa relazione. Quello che è necessario sapere è che GameController possiede una lista di giocatori generici di tipo Connection in cui è contenuto un solo metodo "sendMessage". Ho definito Connection generico perché è una interfaccia e a seconda del tipo di client (RMI o socket) il metodo sendMessage è implementato in modo diverso



Considerando il fatto che il model (quindi la classe Game) come unico input ha la mossa da svolgere e che quindi reagisce solo a messaggi di tipo MY_MOVE_ANSWER (mma) abbiamo deciso che un thread si attiverà proprio in seguito alla ricezione di uno di questi messaggi.

Il GameController, ogni volta che capta un messaggio di tipo MY_MOVE_ANSWER, richiama il metodo makeAmove in TurnController, che a sua volta istanzierà un nuovo thread. Gestire lo svolgimento di una mossa tramite un thread permette al controller di eseguire contemporaneamente i operazioni inerenti alla chat (o altre non direttamente collegate alla partita).

TurnController è la classe che ha accesso diretto al model Game. Quando viene lanciato il metodo run() significa che è arrivata una mossa e la prima cosa che svolge il metodo è controllare che il messaggio provenga dal player corretto; in caso contrario il messaggio viene scartato e il thread termina. (effettua la verifica controllando che il mittente sia uguale alla variabile che mantiene il giocatore corrente)

Nel caso in cui il messaggio proviene dal player corretto ne viene inoltrato il contenuto al model, tramite il metodo manageTurn(). Il model verifica con checkMove() se la mossa è fattibile e in caso positiva la effettua.

Senza entrare nelle specifiche, i vari controlli effettua sono:

- Sulla griglia: estrazione di massimo tre tessere consecutive con un bordo libero
- Sulla libreria: è possibile far scorrere tutte le tessere nella colonna scelta

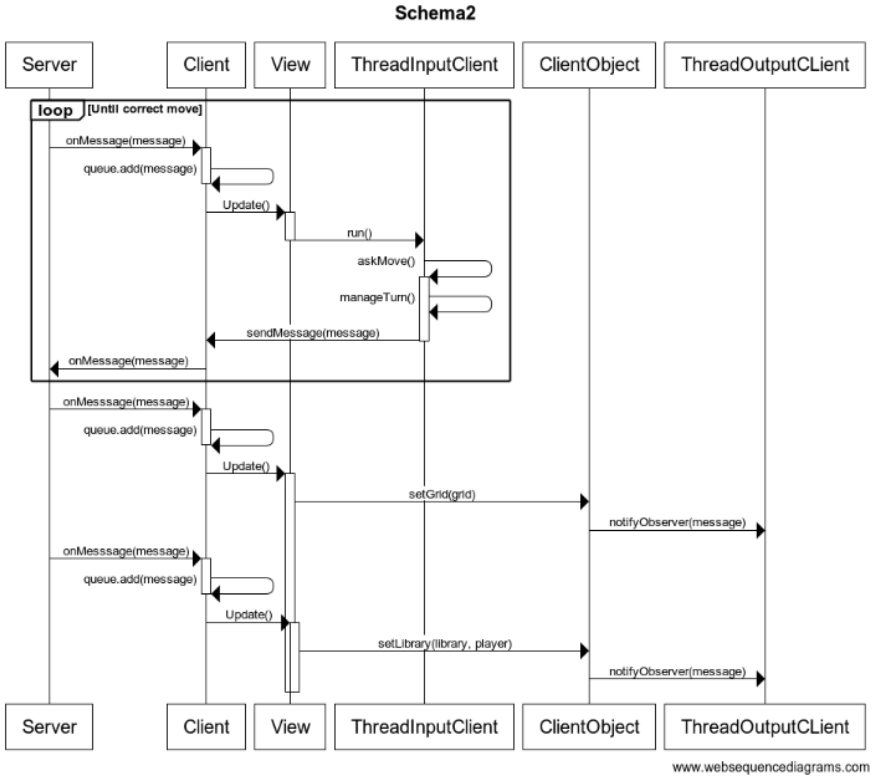
Il messaggio di ritorno dal Game contiene le informazioni riguardo l'esito della mossa: POSITIVE oppure NEGATIVE.

- Caso NEGATIVE: nel caso in cui viene notificato al TurnController che la mossa non è fattibile, quest'ultimo invoca "sendMessage()" sul controller game per richiedere ancora la mossa. Il processo, quindi, termina senza svolgere altre istruzioni.
- Caso POSITIVE: nel caso in cui viene notificato al TurnController che la mossa è andata a buon fine il controller estrapola le altre informazioni contenute nel messaggio. Queste sono opzionali e possono essere: raggiungimento obbiettivo comune o completamente libreria. Notifica quindi tutti i giocatori degli obbiettivi raggiunti dal player che ha svolto la mossa o se la partita sta per terminare.
  Il TurnController richiede anche griglia e libraria aggiornate che inoltra a tutti i player tramite il comando notifyAll(message).
  Prima che termini il processo manda un messaggio con la richiesta di inserimento di una mossa al giocatore successivo e aggiorna la variabile con il giocatore corrente

LATO CLIENT

Nello schema non ho inserito tutte le classi che fanno parte dell'architettura di rete del client, anche perché la maggior parte sono classi ausiliari al funzionamento o implementate/estese. Inoltre, le classi effettivamente istanziate dipendo dalla scelta tra comunicazione RMI e socket.

È necessario sapere che sia l'architettura RMI e socket al ricevimento di un messaggio lo salvano su una coda. Questa coda sarà letta da un secondo thread che si occuperà di gestire i messaggi arrivati.



In modo duale rispetto al server nel client ricevo il messaggio che mi chiede di effettuare una mossa e lo inserisco nella coda dei messaggi. Dopo che il client legge il messaggio dalla coda attiverà il metodo update sulla View. Questo metodo si preoccupa di capire di che tipo di messaggio si tratta e avviare le operazioni corrette.

In questo caso update() lancerà il thread ThreadInputClient. Quest'ultimo non fa altro che chiedere all'utente di inserire una mossa. I controlli si occuperanno solamente della giusta formattazione dell'input e che siano effettivamente presenti dei numeri. L'inserimento dura finché non è valido.

L'inserimento è svolto su un thread separato così che il client possa gestire altri messaggi, che siano di chat o inerenti alla partita.

Infine, arrivano i messaggi per l'aggiornamento della griglia e della mia libraria. Questi due messaggi seguiranno lo stesso percorso descritto precedentemente ma, al posto di utilizzare threadinput andranno a modificare la classe posseduta dal client con gli elementi di gioco (ClientObject). Questa classe è una observable e alla modifica attiverà l'observer ThreadOutputClient che stamperà i cambiamenti.