Liar's Dice
Sistemi Distribuiti AA 2015/2016

Davide Aguiari, Fabio Proietti

Università di Bologna, Dipartimento di Scienze dell'Informazione
davide.aguiari@studio.unibo.it
fabio.proietti@studio.unibo.it

Sommario Liar's Dice è un gioco da tavolo nato nel sud America e basato su un meccanismo di scommesse e bluff. Dopo una breve introduzione, andremo a spiegare quali sono state le scelte progettuali utilizzate per implementare questo passatempo, incluse l'architettura della rete, le librerie implementative e gli aspetti legati alla usabilità e correttezza del sistema.

1 Introduzione

La nostra idea di progetto è stata quella di sviluppare la versione multiplayer del famosissimo gioco da tavolo, con un approccio di rete distribuito e attribuendo particolare attenzione all'affidabilità del sistema al fine di tollerarne i guasti di tipo crash.
Liar's Dice [1] è originario del Sudamerica e importato in Europa dai *conquistadores* nel XV secolo. Il gioco è conosciuto anche con i nomi di *Perudo*, *Dudo* o, grazie anche al film Pirati dei Caraibi, come *Pirate's dice*.[1]
Le successive sezioni tratteranno la logica del gioco, gli aspetti progettuali, le scelte implementative adottate, la valutazione dopo la fase di test, e per finire alcune considerazioni conclusive sul sistema e possibili miglioramenti futuri.

2 Logica del gioco

Per rendere più semplice la comprensione progettuale del sistema, in questa sezione spiegheremo più in dettaglio le regole del gioco.
Ogni partita si gioca con un minimo di due giocatori ad un massimo di 8 (ma potrebbe essere superiore in base alla versione). Tutti i giocatori all'inizio di un nuovo round lanciano i dadi a loro disposizione (inizialmente sono 5) e li tengono coperti in modo che gli avversari non vedano i valori ottenuti. A questo punto colui che inizia il round, in base anche ai suoi dadi, deve effettuare una scommessa nel quale indicherà quanti dadi dello stesso numero sono potenzialmente in gioco, compresi quelli degli altri giocatori.
A questo punto il giocatore successivo può scegliere se credere alla precedente scommessa oppure dubitare. Nel primo caso esso è tenuto ad effettuare un rilancio sulla base dell'ultima scommessa, definendo quindi o un valore del dado maggiore rispetto al precedente oppure un numero di occorrenze superiore alla scommessa appena effettuata e passare di nuovo il turno. Nel caso in cui dubita (nella versione inglese il giocatore urla Liar!), tutti i partecipanti alla partita sveleranno i propri

[1] https://www.youtube.com/watch?v=piGg5ZrmoQA

dadi per decretare chi ha vinto. Se la scommessa contiene un numero di occorrenze minore o uguale rispetto al valore del dado, il giocatore che ha precedentemente scommesso vince, e il suo accusatore perde un dado; altrimenti il vincitore sarà chi ha dubitato e a perdere un dado è lo scommettitore. Il round successivo, quindi, il giocatore perdente lancerà un dado in meno rispetto agli altri. Se un giocatore rimane senza dadi avrà automaticamente perso la partita e a vincere sarà colui che riuscirà ad eliminare tutti i partecipanti alla sfida.
La nostra versione inoltre contiene anche il dado jolly: il dado di valore 1 può essere usato con un valore diverso a piacere, solo se nessuno, durante il turno, scommette utilizzando 1 come valore reale; a quel punto gli 1 smettono di valere valore jolly per tutta la durata del turno.

3 Aspetti progettuali

L'obiettivo primario del gioco progettato è la tolleranza ai guasti di tipo crash, ottenuta mediante uno studio accurato del sistema di comunicazione tra i processi giocanti e lo scambio di informazioni per mantenere la coerenza durante l'intera partita.

3.1 Il gioco

Il gioco (fig.1), dal punto di vista progettuale, non differisce dalla versione realmente giocata e propone a chi gioca, una grafica accattivante e dinamica in grado di visualizzare ogni mossa fatta da ogni giocatore attivo.
Ognuno è costantemente aggiornato sulla situazione dei propri dadi e di quelli degli altri, pur non vedendoli scoperti se non alla fine del turno. La board indica chi è il giocatore attivo, ovvero colui che deve far la mossa, mediante un'apposita cornice, mentre il riquadro sinistro ne riporta le informazioni in formato testuale.
La logica, per ogni mossa, verifica se precedentemente vi sono già state fatte scommesse relative ai dadi sulla board: in caso negativo, significa che il giocatore attivo deve iniziare il turno e la board propone solamente la possibilità di fare una scommessa (*Make bet*); in caso positivo il giocatore può decidere se dubitare (*Doubt*) della scommessa del giocatore precedente, o rilanciare con una scommessa più alta, mediante gli appositi pulsanti.
Il sistema, se qualcuno dubita, verifica chi ha ragione tra chi ha dubitato e il giocatore precedente che ha scommesso che sulla board c'è almeno una certa quantità di dadi con un certo valore. A questo punto chi non aveva ragione, perde uno dei 5 dadi a disposizione, mentre chi vince ha diritto a iniziare il prossimo turno.

Figura 1. Screen della Board di gioco

3.2 Connessione dei nodi con il server

Prima di descrivere la comunicazione tra i nodi è necessario spiegare come questi reperiscono le informazioni utili per riconoscere con chi possono interagire.
Ogni nodo prima di comunicare con gli altri deve necessariamente stabilire un collegamento con una lobby che funge da server e si salva indirizzo IP e porta di ogni partecipante. Esso inizialmente attende una finestra di tempo in cui tutti i nodi possono notificare la loro volontà di connessione; al termine di questa fase, la lobby invia in broadcast un array contente gli indirizzi dei partecipanti alla partita in modo che d'ora in poi possa avvenire la comunicazione diretta solo tra nodi. Questa fase è l'unica in cui si presenta una interazione con una entità centralizzata.

3.3 La comunicazione

L'aspetto più complesso dell'intera progettazione è stato lo studio della comunicazione tra i processi in gioco.
La fasi in cui vengono scambiate le informazioni della board sono sostanzialmente tre:

- La distribuzione iniziale dei dadi
- La segnalazione di una mossa fatta
- La chiusura del turno se un giocatore dubita

La distribuzione iniziale dei dadi Inizialmente si era pensato di adottare un sistema di distribuzione di tipo *token ring* dove ognuno creava il proprio set di dadi e, partendo da 0, possessore del token, lo si copiava nelle strutture dati apposite del successivo; successivamente anche 1, ricevuto il token da 0, poteva informare 2 dei dadi di 0 e di 1, e così via, completando quindi 2 giri del ring per avere un'informazione completa della board.
Sfortunatamente si sono riscontrati alcuni problemi di latenza nello scambio dei dadi man mano che la partita proseguiva, preferendo la soluzione definitiva implementata: a inizio partita inizia sempre il giocatore con id 0 che si preoccupa di generare i dadi di tutti i giocatori connessi e di distribuirli ad ognuno, dimezzando così i messaggi richiesti per informare tutti e risolvendo i problemi riscontrati con il metodo precedente.

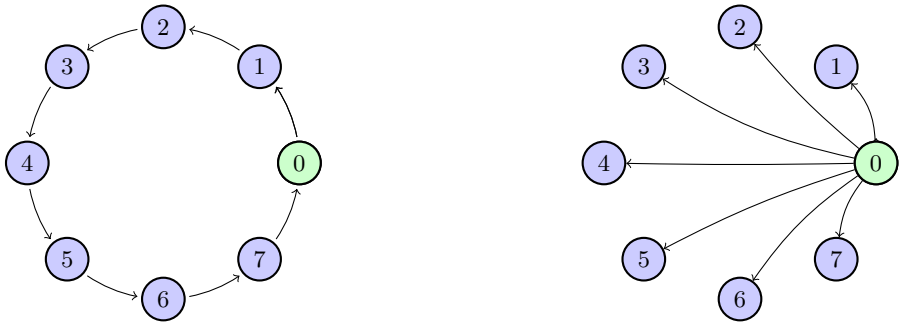

Figura 2. Primo giro, su due, dello scambio dadi con distribuzione token ring (sinistra) e soluzione broadcast adottata (destra).

La segnalazione di una mossa fatta Quando un utente ha deciso la mossa da fare (rialzare la scommessa o dubitare), il sistema notifica prima il giocatore successivo e successivamente anche tutti gli altri, così chi non è direttamente interessato alla singola puntata, conosce l'evoluzione del turno in corso.

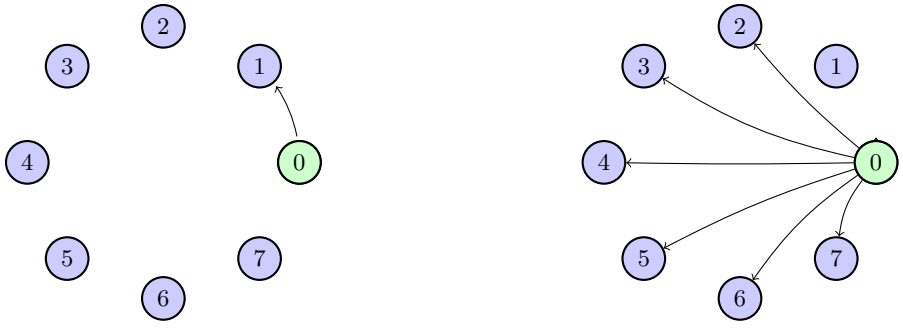

Figura 3. Il giocatore 0 fa una scommessa e la segnala a 1 indicandogli che sarà il prossimo a dover giocare (sinistra); il giocatore 0 notifica a tutti la propria scommessa (destra).

La chiusura del turno se un giocatore dubita Nel momento in cui un giocatore dubita, il turno finisce. Il sistema verifica se il giocatore in questione ha ragione o torto ad aver dubitato. In ogni caso sarà questo giocatore ad avvisare tutti dell'esito del dubbio, rigenerando tutti i dadi e notificando in broadcast come si è concluso il turno.

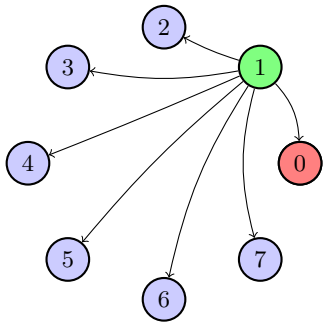

Figura 4. Il giocatore 1 dubita della scommessa del giocatore 0, vince il turno; rigenera i dadi e li distribuisce a tutti notificando l'esito del turno precedente

3.4 La tolleranza ai guasti

La tolleranza ai guasti di tipo crash è stata progettata come segue:
se un giocatore in attesa del proprio turno va in crash, il primo processo che interagisce con lui[1] si accorge e notifica a tutti l'evento crash; a questo punto il gioco continua tenendo conto del set di dadi del player in crash fino alla fine del turno.
Se, invece, il giocatore con il controllo del turno va in crash, tutti vengono automaticamente notificati dell'evento e sarà il giocatore con id successivo a continuare il gioco.

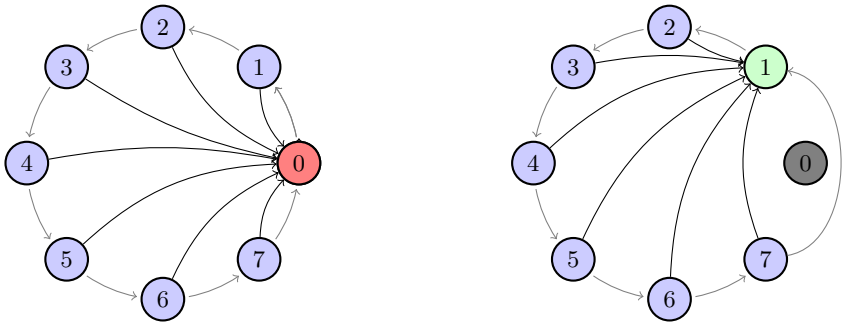

Figura 5. Riconfigurazione dei giocatori dopo un crash del giocatore con il controllo del turno

E' chiaro quindi come il crash di un client non inficia lo sviluppo di un turno, dato che tutti conoscono i dadi degli altri e il sistema permette di concludere la scommessa senza bisogno di un

[1] Interagisce per notificargli una mossa effettuata o la fine di un turno o l'utilizzo del jolly ecc.

restart del turno. Quest'ultimo avviene solo nel caso in cui sia il crash avvenga per il giocatore che sta generando i dadi di tutti: questo particolare caso può essere presente in due distinte fasi dell'intera partita:

- A inizio partita, dopo che ogni giocatore conosce gli altri, può capitare che il giocatore con id 0 sia in crash, non permettendo agli altri di iniziare il turno. E' stato impostato un timeout di tempo arbitrario, per notificare a tutti l'anomalia; a questo punto il giocatore con id successivo si farà carico di ridistribuire i dadi e di verificare la reale situazione del giocatore 0.
- A inizio turno successivo al primo della partita, il giocatore che ha vinto può essere in crash prima o dopo aver notificato l'esito; se il crash avviene prima, l'esito viene annullato e il giocatore successivo continua come nessuno avesse dubitato, mentre se avviene dopo allora il giocatore successivo prende il controllo del turno rigenerando i dadi per tutti.

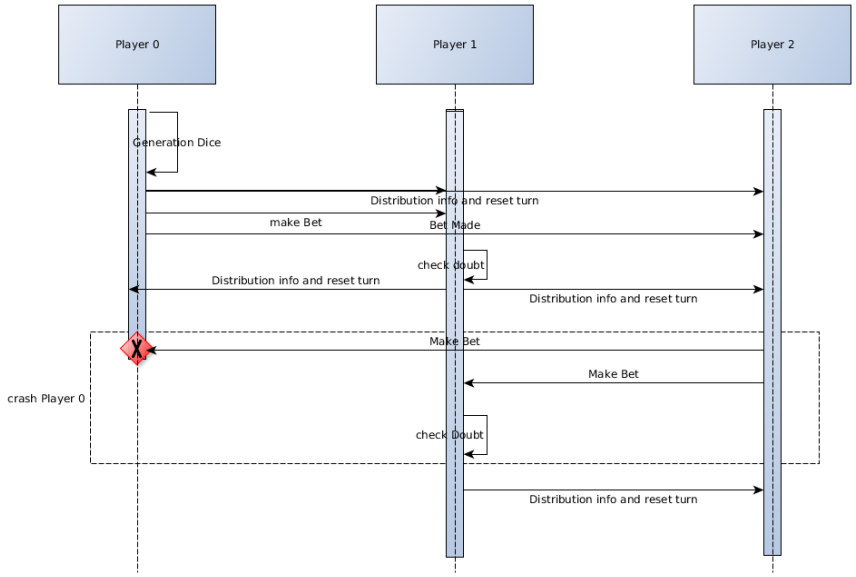

Figura 6. Sequence Diagram di 3 giocatori e con presenza di crash.

4 Aspetti Implementativi

In questa sezione spieghiamo quali sono i dettagli implementati adottati, i design pattern utilizzati e il framework grafico per rendere l'esperienza di gioco più coinvolgente.
Si è scelto di utilizzare il linguaggio a oggetti JAVA perchè ben si presta all'implementazione di un gioco di questo tipo grazie anche a RMI, descritto in seguito, per invocare metodi di processi remoti.

4.1 Design Pattern

La nostra implementazione si è subito orientata sull'utilizzo del design pattern Model View Controller[2]. MVC si sposa perfettamente con le nostre esigenze: la parte che costituisce il Model infatti gestisce direttamente i dati, la logica e le regole del gioco; la View ha lo scopo di mostrare le informazioni elaborate (e quindi l'output grafico) all'utente, che tramite il Controller è in grado di manipolare.
I diagrammi UML nelle seguenti pagine mostrano in dettaglio dipendenze e interazioni tra le classi implementate.

4.2 Implementazione della logica del gioco

La figura 7 mostra il digramma UML delle classi utilizzate per implementare la logica della nostra applicazione.
Lo scopo del Model è quindi quello di generare inizialmente la baord di gioco insieme ai giocatori e ai dadi; la board si preoccupa di indicare al giocatore del turno la possibilità di fare una scommessa, verificandone la correttezza secondo le regole inizialmente descritte; infine si riorganizza lo stato del nuovo turno generando nuovi dadi e aggiornando gli attributi dei giocatori.
La classe DiceLiar contiene il metodo *main()* del progetto e, oltre a far connettere tra di loro i processi mediante la lobby, ha lo scopo di inizializzare la *board*.
I metodi remoti utilizzati da RMI sono tutti organizzati in *RMIController*.

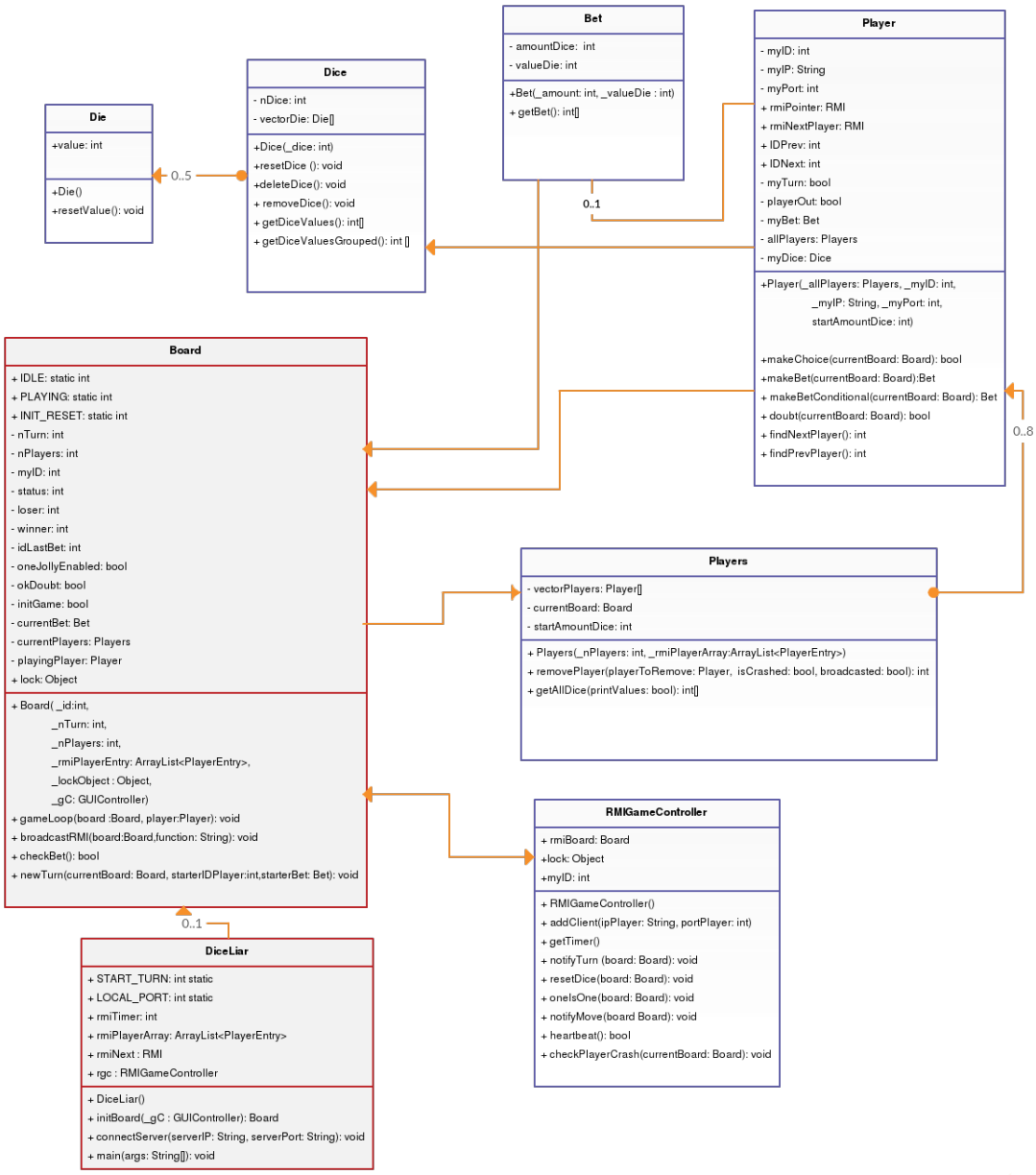

Figura 7. Class Diagram della componente Model.

4.3 L'interfaccia grafica

Per l'implementazione dell'interfaccia grafica ci siamo affidati alla libreria Open Source Slick2D [3]. E' formata da un insieme di tools e funzionalità basate sulle librerie grafiche OpenGL e LWJGL, offrendo quindi un supporto per includere animazioni, immagini, suoni, ecc. alla nostra grafica 2D. L'utilizzo di Slick2D passa attraverso l'implementazione di una classe principale che ha lo scopo di gestire tutte le caratteristiche principali della view come il frame rate, la grandezza della finestra, la modalità di rendering, il passaggio di stato ecc. Tutte le classi che invece definiscono gli stati del sistema, si basano su tre metodi principali:

- void init(): è il metodo richiamato al lancio dell'interfaccia e serve principalmente per l'inizializzazione e il caricamento degli elementi;
- void update(): è un metodo ciclico che aggiorna costantemente lo stato degli elementi inizializzati dal precedente metodo attraverso le informazioni generate dalla componente Controller;
- void render(): in base al refresh rate definito nella classe principale mostra gli elementi disegnati costantemente aggiornati.

L'intero processo della view è eseguito su un thread dedicato. La classe Main si occupa di gestire i vari stati in base alle interazioni che l'utente ha con essa. Per quanto riguarda gli aggiornamenti grafici (dadi correnti, ultima scommessa, ecc.) la view (fig.9) utilizza i valori definiti nella classe *GUIController*, che appunto funge da controller (fig.8).

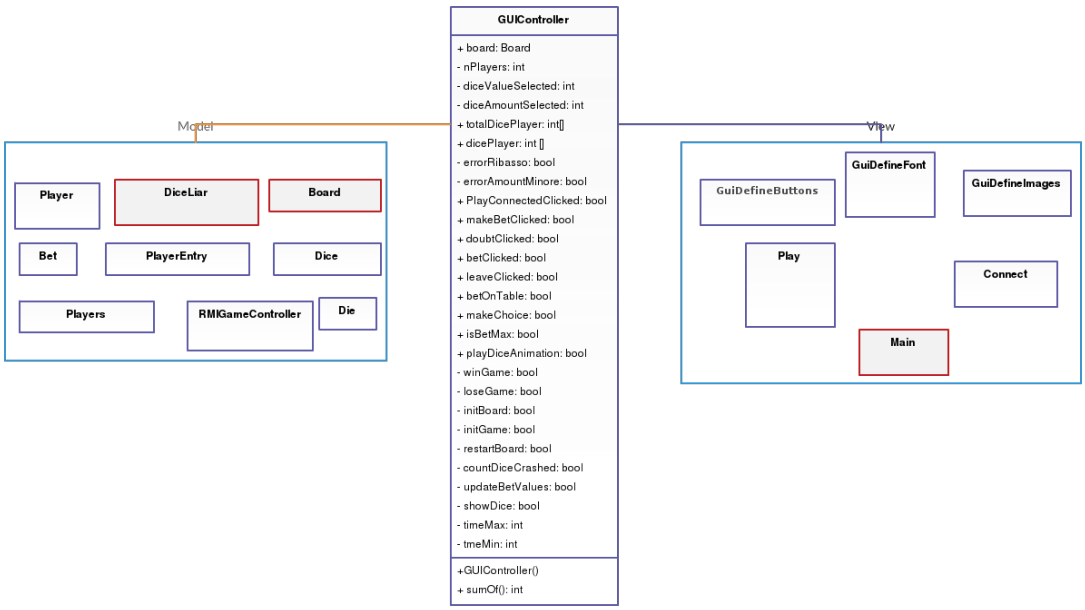

Figura 8. Class Diagram della componente Controller.

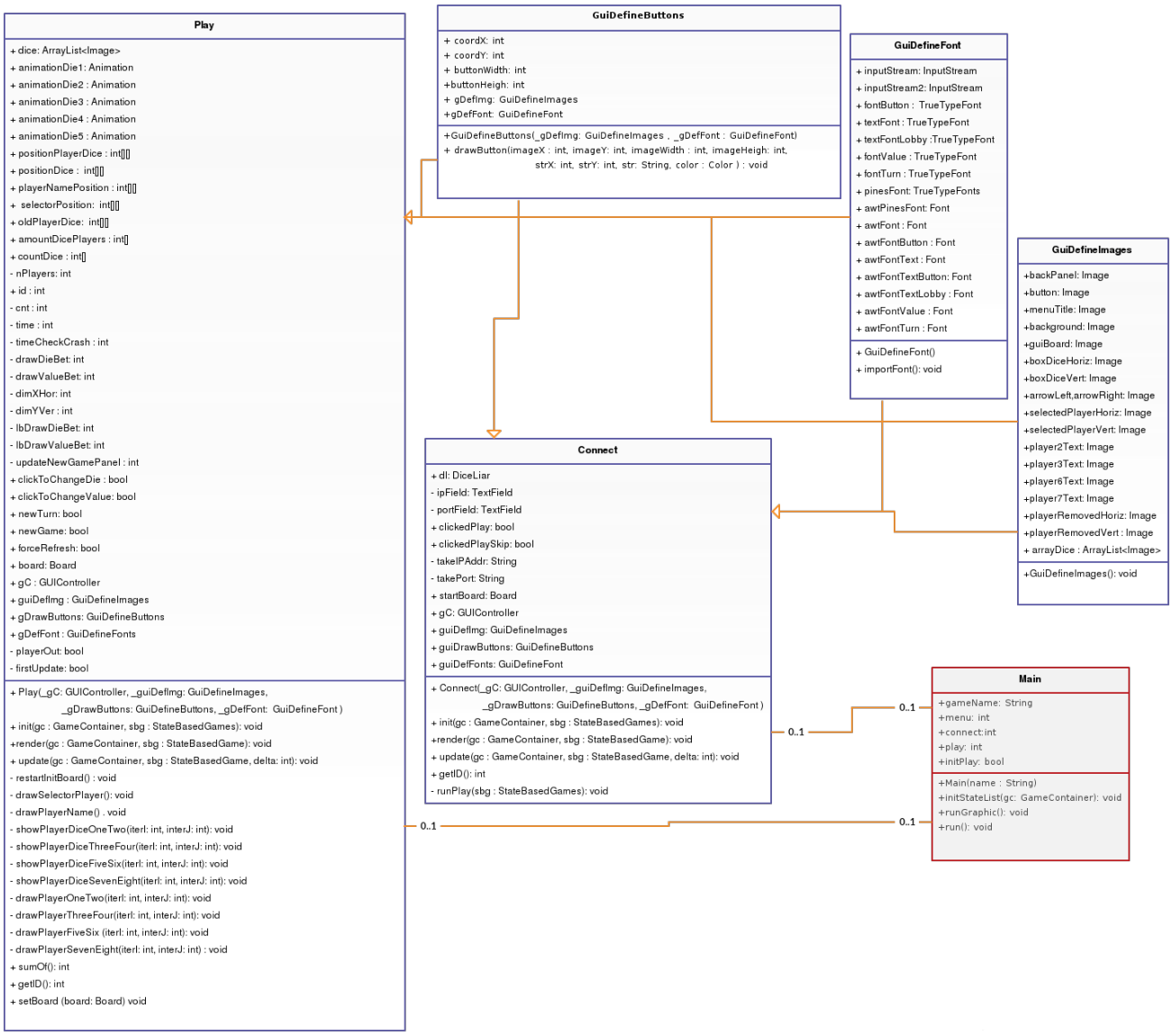

Figura 9. Class Diagram della componente View.

4.4 Comunicazione tra nodi

Lo scambio di informazioni tra i nodi distribuiti viene effettuato attraverso le API di java RMI. La Remote Method Invocation infatti permette a tutti i processi di invocare dei metodi remoti, fornendo così omogeneità e simmetria nel progetto; infatti la procedura utilizzata è disegnata sul paradadigma Object Oriented e i vari processi si comportano come degli oggetti che effettuano delle chiamate a metodi. Più precisamente ogni nodo presente in una differente Virtual Machine, ha

l'obbligo di implementare una interfaccia comune a tutti gli altri mostrando i metodi esportabili. In generale RMI è composto da 3 entità distinte:

- Uno o più server RMI;
- Uno o più client RMI;
- Uno o più Java RMI Registry (uno per ogni server).

Il processo che funge da server implementa l'interfaccia relativa agli oggetti che intende esportare e li registra nel proprio registry, che non è altro che un processo demone che tiene traccia di questi oggetti. Il client quindi non fa altro che effettuare delle chiamate a questo registro al fine di utilizzarne gli oggetti presenti. La gestione della comunicazione è affidata allo stub [4].
Nel caso di Liar's Dice abbiamo deciso di implementare questa tecnica in modo che ogni nodo funga sia da client che da server traedone molteplici vantaggi, uno su tutti la possibilità definire la politica di tolleranza ai guasti descritta precedentemente.

4.5 Tolleranza ai guasti

In aggiunta ai dettagli più di tipo progettuale, descritti in precedenza, vengono qui descritti i dettagli implementativi per quanto riguarda la tolleranza ai guasti di tipo crash. Nel caso di un guasto di questo tipo sul player corrent, il sistema invoca un'eccezione di tipo *RemoteException* verso i nodi a lui connessi e in *wait()* nel metodo RMI *checkPlayerCrash()*; il vecchio player giocante viene quindi tolto dalla lista dei giocatori, conservando i dadi fino alla fine del turno, riconfigurando l'anello individuando chi sono i giocatori precedenti e successivi a quello in stato crash.
Nel caso di un guasto di un nodo intermedio, quindi non direttamente giocante, il risultato è il medesimo al momento di un'interrogazione da parte dei vari nodi dell'anello, in diverse operazioni durante il turno di gioco, come descritto precedentemente. Si è scelto questo approccio per permettere di completare un turno anche se vi sono stati dei giocatori in stato crash, al fine di mantenere coerente lo stato locale delle diverse istanze.
Il sistema così implementato assicura l'esecuzione della partita, tollerando fino a $N-2$ giocatori in crash, dove N è il numero di giocatori; nel momento in cui un giocatore si accorge di essere rimasto l'unico in gara, si autodichiara vincitore della partita.

4.6 Gestione della critical section

Il sistema di gioco e l'architettura token ring del turno è implementata mediante una critical section: solo il giocatore designato giocante (*PlayingPlayer*) è autorizzato a scommettere sulla *Bet* della board, verificando che il valore PlayingPlayer con il proprio id. I giocatori che non sono *PlayingPlayer* vengono bloccati mediante un oggetto di tipo *Lock* e sbloccati (*notifyAll()*) solo quando il turno è concluso, quindi se è stata alzata la scommessa o si è dubitato.
Il meccanismo di tolleranza ai guasti rimane comunque valido, dato che i processi bloccati vengono segnalati e sbloccati se l'eccezione *RemoteException* è sollevata.

5 Valutazione e Conclusioni

La fase più importante per trarre le nostre conclusioni è stata quella di testing; infatti nel momento in cui abbiamo avuto la nesessità di apportate delle modifiche al codice, siamo stati costretti a rivedere

alcune scelte implementative stabilite precedentemente. Dato che la nostra prima implementazione del design pattern MVC non era di facile gestione, abbiamo deciso di raffinare la sua struttura introducendo specifiche classi ponte contenenti tutti gli elementi necessari all'aggiornamento sia del model che della view.
Il progetto ha richiesto circa 1 mese/uomo e si è dimostrato ostico nella configurazione iniziale di RMI e nella risoluzione del bug nella distribuzione dei dadi iniziale che portava la CPU al 700% circa facendoci preferire una modalità diversa, di tipo broadcast. Il lavoro di bugfix è stato importante anche per l'introduzione di alcune migliorie grafiche che hanno migliorato l'esperienza di gioco dell'utente.
In conclusione, visto lo stato attuale del progetto, vi è la possibilità di estenderlo con features quali la modalità di gioco single player o un più semplice meccanismo di connessione al server. Infine la grande versatilità di java potrebbe favorire una semplice e fluida esperienza di gioco sia in ambienti mobile che desktop.

Riferimenti bibliografici

1. `https://en.wikipedia.org/wiki/Liar's_dice`
2. `https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`
3. `http://slick.ninjacave.com/`
4. `https://docs.oracle.com/javase/tutorial/rmi/`