DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Triennale in
Ingegneria delle Tecnologie per l'Impresa Digitale

Relazione di Elementi di Progettazione Software

LUDO

Tommaso

Alessio

Paolo

Anno Accademico 2021/2022

Indice

Elenco delle figure

Capitolo 1

Obbiettivo

L'obbiettivo del presente progetto è quello di implementare il gioco da tavolo "LUDO" in versione digitale, tramite l'utilizzo di ambiente di programmazione Java.
Il progetto implica la creazione di tutte le classi necessarie per il corretto svolgimento della partita, quindi della gestione dei turni e dell'interfaccia grafica, ma anche il controllo dei bot.

Il gioco deve potersi svolgere in modalità Client/Server; pertanto, questa caratteristica deve essere ben nascosta agli occhi dell'utente finale, l'obbiettivo è mascherare il più possibile il funzionamento tramite un buono studio dell'interfaccia grafica.

Figura 1.1: confezione del gioco

Capitolo 2

Regole di gioco

Nel presente capitolo vengono descritte le regole di gioco di base di LUDO comprese le variazioni apportate al presente progetto, per semplificarne la gestione e favorirne la giocabilità.

2.1 La storia del gioco

Pachisi o venticinque è un gioco nato in Pakistan descritto come il "gioco nazionale del Pakistan". Si gioca su una tavola a forma di croce simmetrica. I pezzi del giocatore si muovono intorno alla tavola, basandosi sul lancio di sei o sette conchiglie, il numero di conchiglie che rimangono con una apertura indica il numero di caselle a cui corrisponde il movimento.

Successivamente nel 1896 viene semplificato dalla casa editrice londinese John Jaques & Son e nasce **Ludo**, esso venne introdotto al grande pubblico e cominciò ad avere un discreto successo.

Infine, nel 1908 il gioco venne ulteriormente rivisto a Monaco da Josef Friedrich che creo il gioco da tavolo **Non t'arrabbiare** prendendo spunto da Ludo e Pachisi. La peculiarità di questa versione, dalla quale nasce il nome, è quella di poter "mangiare" un segnalino avversario per costringere il giocatore a ricominciare, provocando in genere una reazione innervosita.

2.1.1 Le regole dei tre giochi

Le regole dei tre giochi sopra descritti si discostano di poco tra loro e si possono trovare in forma integrale ai seguenti link:

Regolamento Pachisi

Regolamento Ludo

Regolamento Non t'arrabbiare

2.2 Le nostre regole

Le regole che abbiamo creato per la nostra versione del gioco sono il risultato di un mix ben pensato tra il regolamento di Ludo e Non t'arrabbiare. Abbiamo cercato di mantenere le parti migliori e rielaborare le meccaniche più critiche che diversamente, avrebbero mandato in conflitto i due regolamenti.

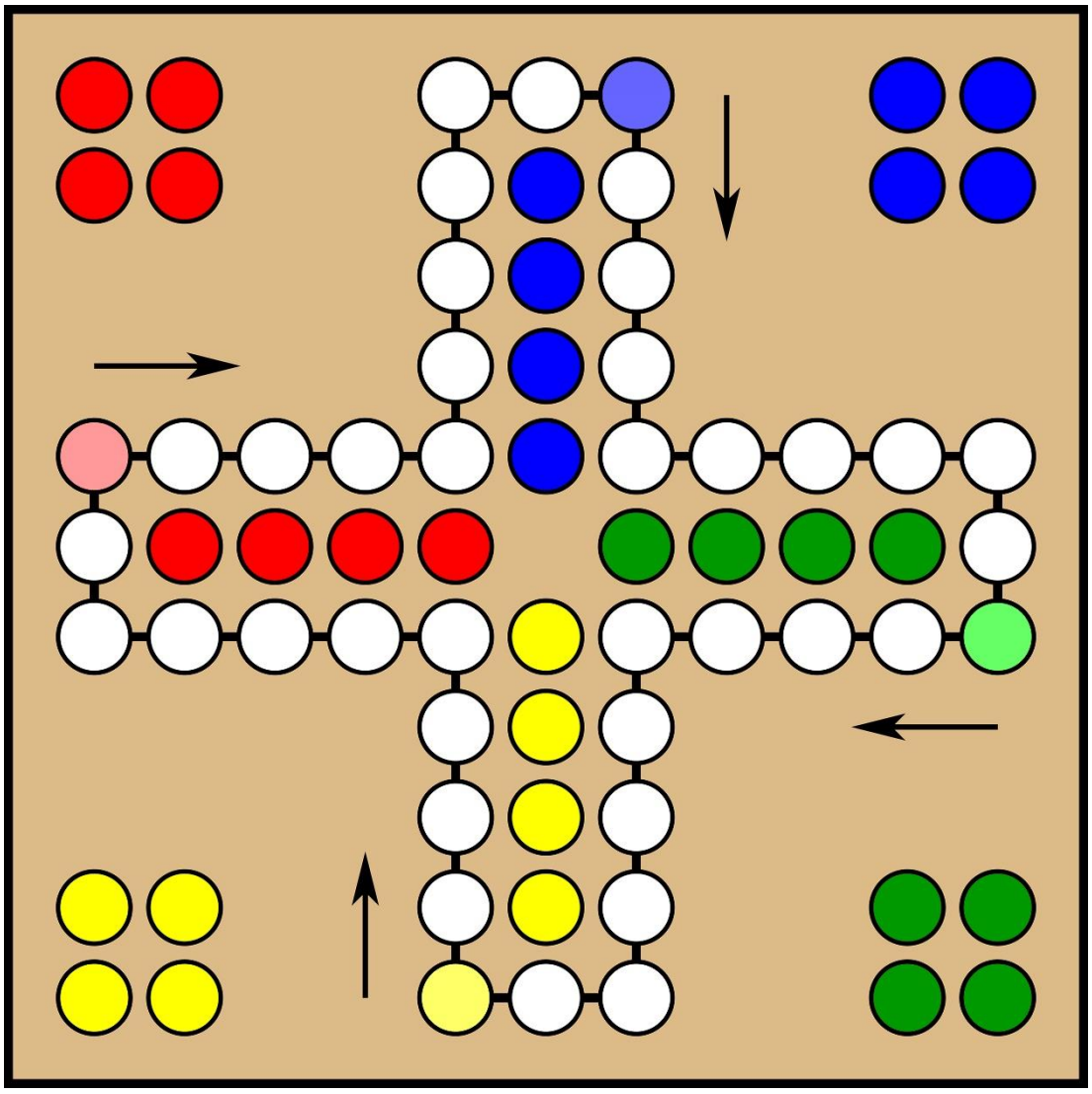

Figura 2.1: plancia di gioco

La plancia di gioco è formata da quattro caselle agli angoli, chiamate le caselle base e da un percorso centrale che segue i bordi di una croce e termina al centro in un'altra grande casella. In ciascuno dei bracci della croce è presente anche un percorso di 4 caselle che sono le basi finali. Inizialmente tutte e 4 le pedine di ogni colore si trovano nelle rispettive 4 basi.

Il gioco inizia con il rosso, che fa la prima mossa e gli altri giocano a turno, seguendolo in senso orario. Il turno di gioco consiste nel lancio del dado e nel conseguente spostamento di una delle pedine di tante caselle quant'è il numero ottenuto. Le pedine però possono lasciare la base di partenza solo se, con il dado, si ottiene un 6; ne consegue che, se nessuna pedina è fuori dalla base con un numero diverso da sei non si può fare alcuna mossa e il turno passa al giocatore seguente.

Quando un giocatore ottiene un 6 colloca la sua pedina sulla prima casella del suo percorso e lancia nuovamente il dado: la pedina entrata in gioco avanza di tante caselle quanti sono i punti ottenuti con il dado, nella direzione indicata dalla freccia.

Lo scopo del gioco è far compiere a tutte e 4 le proprie pedine il giro completo del tavoliere (sulle 40 caselle del circuito esterno) e farle quindi entrare nelle 4 caselle del proprio colore, poste all'interno del circuito.
Ogni volta che un giocatore ottiene un 6 ha facoltà di scelta fra il far uscire dalla base una sua pedina e il far avanzare una sua pedina che si trova già sul circuito.

Indipendentemente da questa scelta, dopo il lancio di un 6, si ha sempre diritto ad un altro lancio del dado e ad utilizzare i punti con esso ottenuti.

2.2.1 Sovrapposizione

Quando invece una pedina raggiunge una casella occupata da una pedina dello stesso colore, le due pedine si "sovrappongono" momentaneamente: finché una delle due non si sposta, esse non possono essere sorpassate da altre pedine (avversarie o amiche) e non possono neppure essere rimandate alla base. Attenzione però:

- Le pedine sovrapposte possono essere solo 2. Non esistono sovrapposizioni da 3 o 4.
- Non si possono effettuare sovrapposizioni nella casella di spawn né propria né altrui. Perciò, se la prima casella del percorso è occupata da una propria pedina, con un 6 non si può far entrare nel circuito un’altra pedina, ma bisogna utilizzare il punteggio per un'altra mossa e quindi ripetere il lancio del dado.
- Le pedine sovrapposte non possono essere sorpassate in nessun caso, né da pedine amiche né da quelle avversarie. Può quindi

succedere che, al proprio turno, un giocatore non possa fare nessuna mossa con il punteggio ottenuto: in questo caso egli non muove e il turno passa al giocatore successivo, senza altre conseguenze.

2.2.2 Mangiata

Quando muovendo una propria pedina si raggiunge esattamente una casella occupata da una pedina avversaria, quest'ultima viene "mangiata" e rimandata nella sua base di partenza, da dove dovrà poi ricominciare il percorso (sempre con un 6).

Una pedina che esce dalla base di partenza può colpire una pedina avversaria che eventualmente si trova nella sua casella d'ingresso al percorso.

2.2.3 Fine partita

Le pedine che hanno completato il giro del percorso devono occupare le 4 caselle di arrivo per far terminare il gioco. Queste caselle devono necessariamente essere raggiunte con un tiro preciso del dado.

Se, ad esempio, un giocatore ha una pedina sull'ultima casella del suo percorso ed ottiene un 5, non può posizionare la sua pedina nell'ultima delle sue caselle d'arrivo. Se il punto da utilizzare fosse stato un 1 o un 2, la pedina, ovviamente, avrebbe potuto raggiungere solo la prima o la seconda delle sue caselle d'arrivo e attendere lanci successivi per avanzare ulteriormente.

Naturalmente le pedine nelle caselle d'arrivo sono inattaccabili dai pezzi avversari.

Capitolo 3

Descrizione del codice

Nel presente capitolo vengono descritte le varie classi utilizzate nel progetto, comprendendo una breve descrizione degli attributi e dei metodi che li caratterizzano.

I vari commenti e la descrizione a tutte le funzioni del codice sono consultabili da questo sito dove abbiamo caricato il JavaDoc del nostro progetto. Inoltre, di seguito riportiamo un commento e una breve descrizione di come abbiamo programmato e progettato ogni parte fondamentale del programma, ovvero Model, View e Controller.

Link al JavaDoc: www.mensipedia.altervista.org/EPS_Ludo/

Repo GitHub: www.github.com/IlBuonTommy/Progetto_EPS_2022

3.1 View

Il nostro obbiettivo principale è stato fin da subito quello di nascondere le meccaniche di setup del server e del client all'utilizzatore finale, rendendo più semplice e scorrevole possibile il programma.

Di seguito viene descritto il flow dei vari Frame che inizialmente vengono mostrati al giocatore: la schermata iniziale consente di avviare il programma in versione client o server, questo accorgimento ci permette di evitare di avere due programmi diversi uno esclusivamente per la gestione del server e uno con la gestione del client, ma di averne invece uno unico.

Inizialmente descriviamo il flow del server; dopo aver premuto sul tasto "server" viene aperto un nuovo frame che consente di inserire il nome utente del giocatore e uno slider permette di impostare il numero massimo di giocatori che la partita ospiterà (da un minimo di 2 ad un massimo di 4). L'utente può successivamente avviare il server premendo sul tasto, automaticamente il programma si metterà in ascolto di nuove connessioni da parte dei client sulla porta 50358.

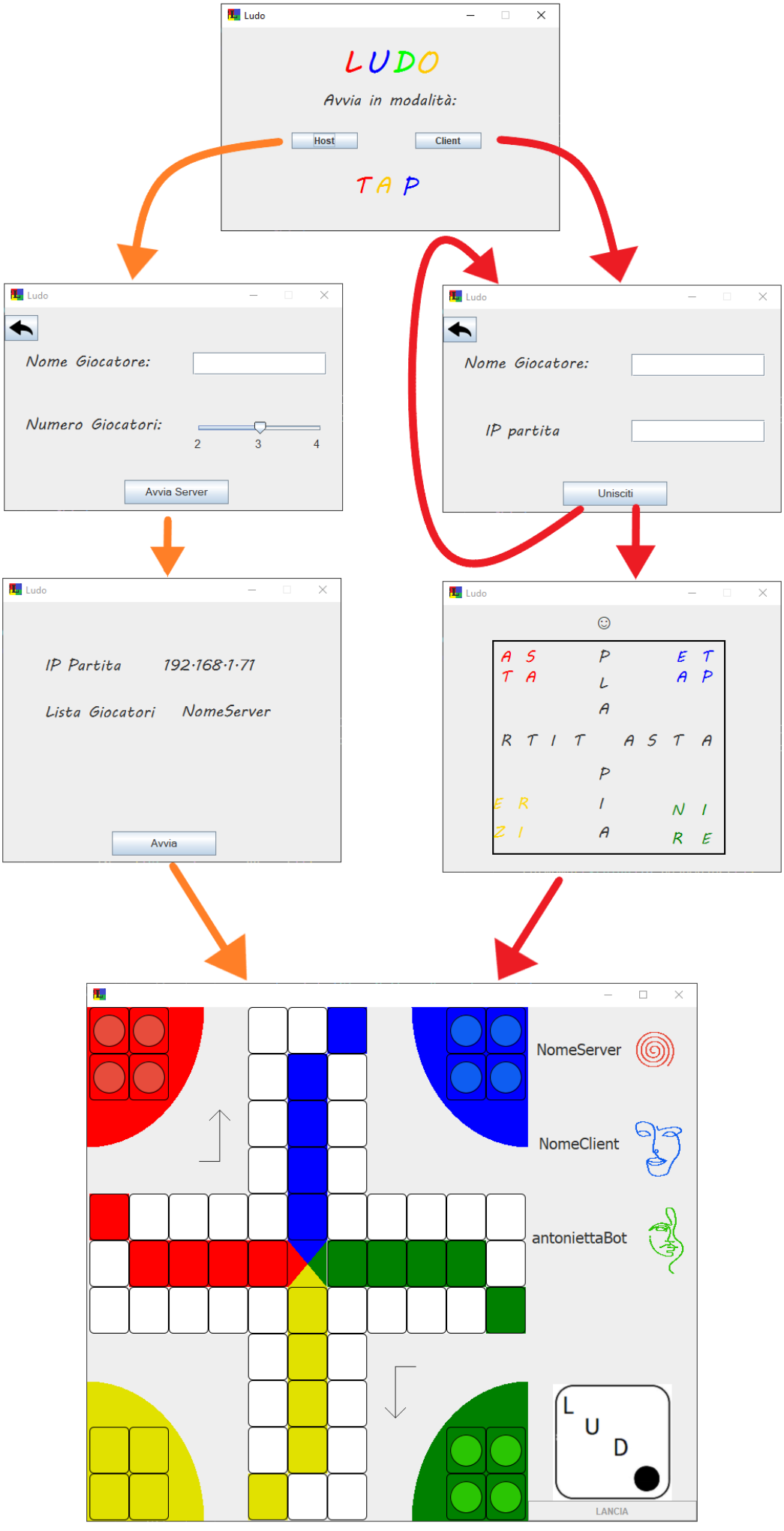

Figura 3.1: flow dei Frame

Sul nuovo frame viene mostrato l'IP della macchina server in modo che possa facilmente comunicarlo agli altri giocatori che intendono giocare con lui, inoltre viene visualizzata una lista aggiornata in tempo reale dei giocatori connessi al server. L'utente può iniziare la partita in qualsiasi momento premendo il tasto "avvia", in caso non si siano connessi abbastanza giocatori la lobby viene riempita automaticamente con dei bot. Ad esempio: se inizialmente l'utente aveva creato una partita da 3 giocatori, ma si connette solo un client e viene premuto il tasto "avvia" l'altro giocatore viene impostato come un bot.

Lato client, il primo frame che viene mostrato permette di inserire l'username del giocatore e l'IP del server al quale ci si vuole connettere, in caso l'IP inserito sia errato viene riproposto lo stesso frame finché non si crea una connessione valida con un server. Successivamente il client entra in una waiting room e viene visualizzato un apposito frame, il giocatore client resterà bloccato qui finché dal server non verrà avviata la partita.

Quando la partita viene avviata a tutti sarà visualizzata la plancia di gioco, nella quale sul lato destro sono visualizzati i nomi dei giocatori e dei segnalini formati da punti creati nel metodo paint di un JPanel e quindi completamente ridimensionabili, lo spirale indica di chi è il turno e cambierà la sua posizione di turno in turno. Da qui la partita ha inizio!

3.2 Controller

Abbiamo visto come il controller ha come compito specifico quello di collegare la view al model. In questo caso il model è uno solo ed è salvato all'interno del server. Una volta avviato esso viene successivamente ripreso dai client (che ciclicamente ne creano una copia) tramite l'utilizzo delle connessioni client-server.

Nel caso del turno dell'host il frame, tramite l'utilizzo di alcuni action listener, è in grado di comunicare con il serverModel tramite l'uso di getter and setter passando per la classe Server. Nel caso del turno del client è stato necessario collegare gli action listener delle view alla classe client; essa poi comunica tramite canali di input e output con il server e con il Client Handler, che a sua volta aggiorna il serverModel interagendo con la classe Server.

3.3 Model

Nel model abbiamo dovuto digitalizzare tutti i dati del gioco, abbiamo optato di creare una classe “casella” che contenesse le seguenti informazioni: un intero per il colore della pedina al suo interno (-1 se vuota) e una variabile booleana che indichi se è presente una doppia.

```
5   public class Casella implements Serializable{
6       private int colore;
7       private boolean doppio;
8
9       public Casella() {
10          this.colore = -1;
11          this.doppio = false;
12      }
```

Successivamente abbiamo dovuto raggruppare queste tipologie di dati per poter realizzare una classe che riuscisse a rappresentare tutti i dati necessari a visualizzare in maniera completa la plancia di gioco, essi sono contenuti nella classe GameModel.java

```
8   public class GameModel implements Serializable{
9
10      private Casella[] plancia;
11      private Casella[][] base;
12      private Casella[][] finale;
13      private Giocatore[] player;
14      private int currentPlayerIndex;
15      public int numGiocatori;
```

Essa contiene un array monodimensionale di caselle che rappresentano le 40 caselle della plancia di gioco, due matrice bi-dimensionale per rappresentare le 4 caselle di ogni colore della base e della sezione finale di gioco, un array di player, un intero che indica il turno del giocatore corrente e un intero che indica il numero di giocatori della partita.

Nel model sono presenti tutti i metodi che servono per interagire con la plancia di gioco e le regole, ad esempio: capire se il giocatore attuale è un bot, passare il turno, capire se è rimasta qualche pedina di un determinato colore in base, controllare che qualcuno abbia vinto, mangiare una pedina, gestione dello spostamento di una pedina, la gestione del movimento di un BOT, capire quali tasti può abilitare la view…

Nello specifico cerchiamo di capire cosa deve fare il model quando deve spostare una pedina dalla plancia di gioco: innanzitutto deve sapere la posizione della pedina, il valore del dado e se deve eseguire la mossa o deve solo capire se quella mossa è fattibile. Inizialmente calcola la posizione che dovrà avere la pedina con il valore del dado che è stato passato e successivamente controlla che non ci siano conflitti con le regole del gioco. L'ordine con il quale vengono effettuate queste verifiche è molto importante, si parte dal controllo delle pedine doppie sul percorso che porta la pedina da spostare dalla sua vecchia posizione a quella nuova (in caso ce ne siano si ha un *return false* perché le pedine doppie non possono essere superate). Il secondo controllo riconosce se la pedina deve spostarsi nella parte finale del suo percorso (ovvero le ultime 4 caselle del suo colore). Successivamente controlla se deve effettuare una mangiata di un'altra pedina:

```
if(plancia[nuovaPosizione].getColore()!=-1 &&
        plancia[nuovaPosizione].getColore()!=plancia[posizione].getColore()){

    if(plancia[nuovaPosizione].getDoppio())
        return false;
    if(daEseguire){
        mangiata(nuovaPosizione);
        plancia[nuovaPosizione].setColore(plancia[posizione].getColore());
        if(plancia[posizione].getDoppio()){
            plancia[posizione].setDoppio(doppio: false);
        }else{
            plancia[posizione].setColore(-1);
        }
        return true;
    }
}
```

Il quarto controllo consiste nel verificare che la mossa richiesta vada a creare una doppia e in caso di verifica positiva va a settare il booleano doppio a *true* effettuando lo spostamento. Infine, se la funzione non è stata fermata precedentemente esegue la mossa.

Anche il movimento del BOT viene gestito dal model e per essere eseguito deve ricevere il colore che lo richiede e il valore del dado, successivamente l'algoritmo dà la precedenza alle seguenti mosse: controllare se una sua pedina può andare in base, uscire con una nuova pedina creare la doppia pedina più distante possibile, mangiare la pedina nemica più distante, liberare l'uscita, spostare la più distante…

Capitolo 4

Conclusioni

4.1 Difficoltà incontrate

Durante lo sviluppo del codice del progetto, la difficoltà principale è stata implementare una buona connessione client-server che riesca a mandare e ricevere array di oggetti e riesca a gestire tutti e 4 i giocatori. Non è stato per nulla semplice, inoltre, visualizzare i dati scambiati sulla view di ogni giocatore. Infatti, una errata comunicazione tra Client e Server e tra questi e la View avrebbe potuto portare ad una non coerenza tra le partite visualizzate dai giocatori, rendendo di fatto ingiocabile il match.

Neppure rielaborare i vari regolamenti delle versioni del gioco precedenti è stato facile, il nostro obbiettivo è stato fin da subito creare qualcosa di nuovo, ma che avesse delle meccaniche semplici e divertenti, come i giochi originali dai quali abbiamo preso spunto.

Un altro aspetto critico per il codice è stata l'implementazione dell'interfaccia grafica: essa deve avere un aspetto chiaro e deve contenere tutte le informazioni che un giocatore deve tenere in considerazione per decidere la propria mossa successiva, comprese le mosse che hanno fatto i suoi avversari. Uno dei nostri obbiettivi è stato cercare di nascondere la complessità delle connessioni client-server all'utente finale, cercando di semplificare il settaggio del server e l'ingresso nella lobby di gioco da parte del client.

4.2 Possibili sviluppi futuri

Avendo creato una meccanica di gioco nuova è possibile che alcuni giocatori vogliano utilizzare le regole originali di Ludo o di Non t'arrabbiare. Sarebbe possibile consentire al server di scegliere il regolamento da utilizzare durante tutta la partita, in questo caso bisognerebbe aggiungere dei model in base alla tipologia e quantità di regolamenti da implementare, tenendo invariati la view e il controller.

Si potrebbe anche consentire al server, una volta finita una partita, di effettuarne una nuova, tenendo conto dei punteggi delle partite passate e costruendo una classifica dei giocatori che hanno ottenuto più vittorie.

Infine, si potrebbe migliorare maggiormente la personalizzazione e l'interazione dei giocatori fra loro, inserendo l'opzione di scegliere il proprio colore o implementare una chat di gioco dove tutti i giocatori abbiano la possibilità di interagire tra di loro.