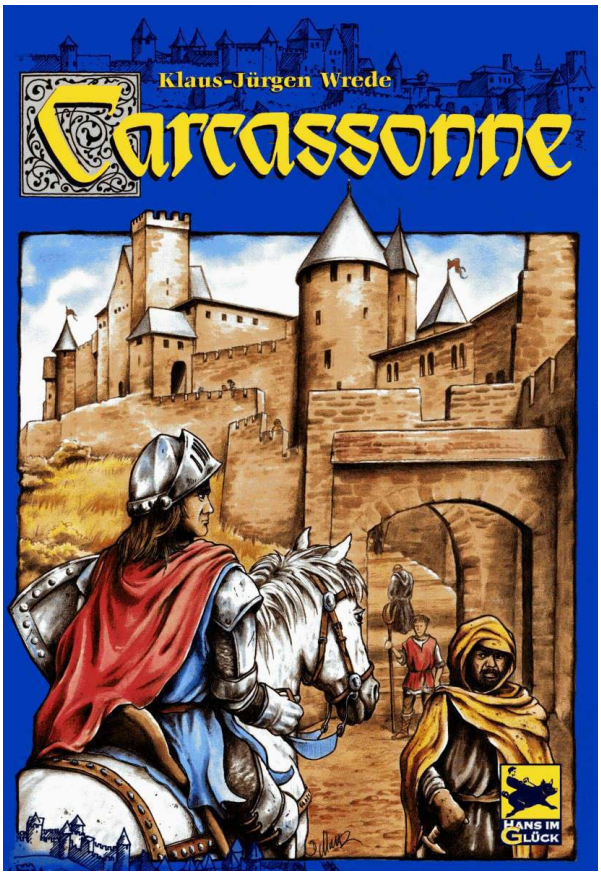
Carcassonne

Sviluppo in ambiente Java del gioco da tavolo Carcassonne



Giacomo Bellezza & Luca Comminiello

a.a. 2014/2015

INDICE GENERALE

CAPITOLO 1 - DESCRIZIONE DEL PROBLEMA

Carcassonne è un gioco da tavolo in stile medievale,progettato da Klaus-Jurgen Wrede e prende il nome dall'omonima città francese,caratterizzata da mura e fortificazioni.Si tratta di un gioco a mappa componibile in cui la plancia viene creata dagli stessi giocatori durante la partita e l'obiettivo di ciascuno è totalizzare quanti più punti possibile dall'accostamento di varie tessere,che rappresentano città, strade, campi e monasteri,prima che queste si esauriscano.Il turno di ogni giocatore si compone di due fasi principali : inizialmente viene pescata una tessera dal mazzo e posizionata sul tabellone;in seconda fase è possibile ma non obbligatorio piazzare segnalini (*seguaci)* sulla tessera appena collocata.

Per posizionare le tessere devono essere rispettate le seguenti regole:

1. la nuova tessera deve avere almeno un bordo in contatto con una delle tessere già in gioco.
2. la nuova tessera va posizionata in modo che città, strade, campi siano contigui a quelli della/e tessera/e con cui è in contatto (es. una strada non può finire in un campo o un città non può attaccarsi ad un campo senza chiudere le mura).

soluzione corretta

soluzione errata

Piazzamento del Seguace

Il seguace reclama la proprietà di un elemento di terreno (una strada, un campo, una città o un monastero) e non può essere piazzato su un elemento già reclamato da un altro seguace. Ciononostante, è possibile che un elemento (strada, città o campo ma non il monastero) sia reclamato da due o più seguaci, se tratti inizialmente separati (e sui quali siano stati piazzati dei seguaci) vengono uniti successivamente.

Punteggio e Vincitore

Il calcolo del punteggio viene effettuato progressivamente durante la partita,al termine del turno di ogni giocatore,sulla base delle città completate e sulle strade chiuse,sulle quali sia presente un seguace.Alla fine del gioco vengono conteggiate

anche le strade e città incomplete,occupate da un seguace.

Vince il giocatore che ha totalizzato il maggior numero di punti.

Si rimanda al seguente link per il regolamento completo :

Regolamento ufficiale con espansioni

Regolamento gioco base

La versione da noi sviluppata non prevede la variante del "contadino".

CAPITOLO 2 - REQUISITI DI SISTEMA

- Il gioco deve prevedere la possibilità di effettuare partite con un numero di partecipanti variabile (da 2 a 4)
- Deve essere possibile ai giocatori salvare / caricare le partite.
- E' previsto l'inserimento di effetti audio appena il giocatore avrà posizionato la tessera sulla plancia (a seguito di alcune partite si è preferito riprodurre gli effetti audio all'inizio e alla fine della partita e ogni volta che vengono totalizzati nuovi punti)
- L'interazione con l'utente deve prevedere comandi rapidi con i quali velocizzare le mosse (rotazione tessere,avanzamento del proprio turno)
- ~~Al termine di ogni partita verranno visualizzate le statistiche di ogni giocatore e sarà possibile visualizzare record e punteggi delle precedenti~~

  Realizzare un breve Tutorial per spiegare in modo rapido come effettuare una partita,sfruttando i comandi rapidi attraverso il mouse (N.B. a seguito della consultazione è stata stabilità la modifica del requisito)
- Realizzare un sistema che avverte il giocatore della non completa visualizzazione della mappa a video

REQUISITI DI GIOCO

Per rendere l'applicazione giocabile,necessariamente :

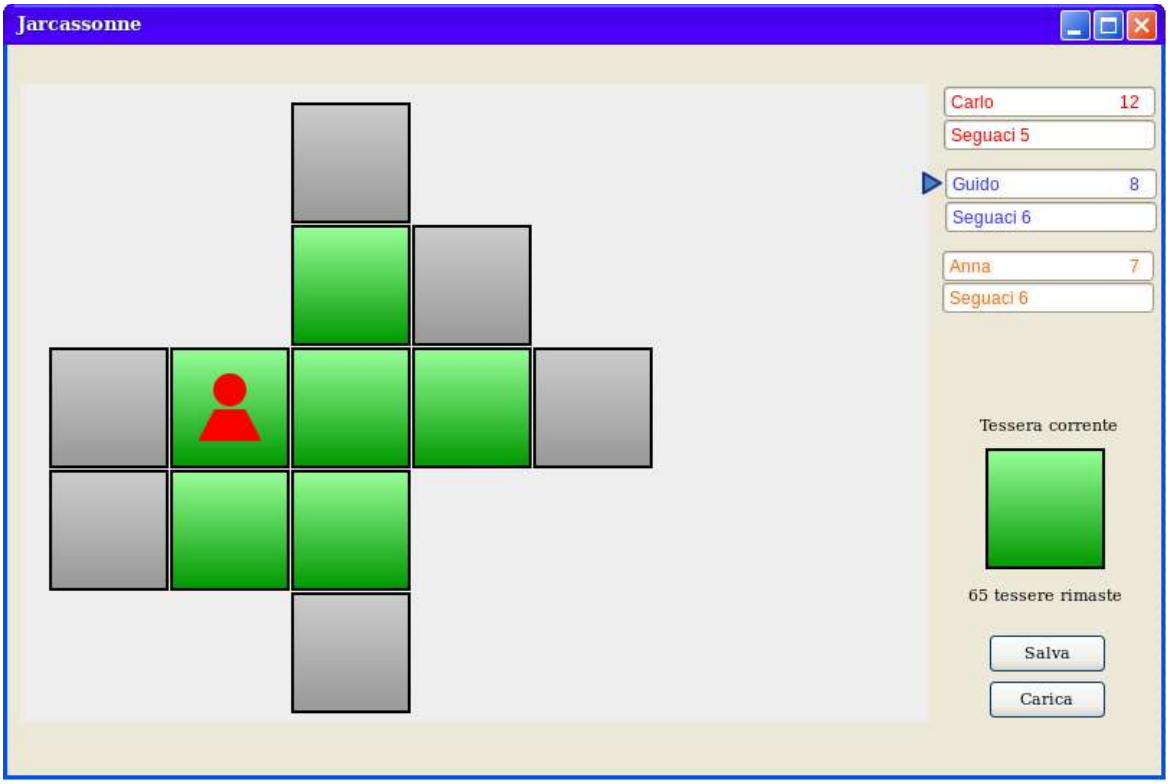
- Dovranno essere evidenziate le caselle sulle quali il giocatore potrà posizionare la tessera appena estratta dal mazzo e tramite un'anteprima sarà possibile visualizzare la tessera nella cella selezionata dal puntatore
- Le tessere dovranno poter essere ruotate

- La plancia si potrà ridimensionare
- Per garantire partite sempre differenti il set da 72 tessere verrà mescolato randomicamente all'inizio di ogni partita
- Il seguace si portà posizionare in diverse parti della tessera,compatibilmente al suo contenuto (su una strada o una città) con la possibilità di riprenderlo non appena sarà stata chiusa la città o la strada sulla quale era stato posizionato; tuttavia verrà data la possibilità mediante un bottone di non piazzare il seguace durante un turno se non ritenuto necessario
- Durante la partita è probabile che una strada o una città vengano contese da più giocatori; perciò il gioco dovrà prevedere la gestione di queste eccezioni
- Il programma verificherà che una città/strada non ancora completata ma già occupata da un giocatore,mediante segnalino,non possa essere reclamata da un altro giocatore

REQUISITI FACOLTATIVI

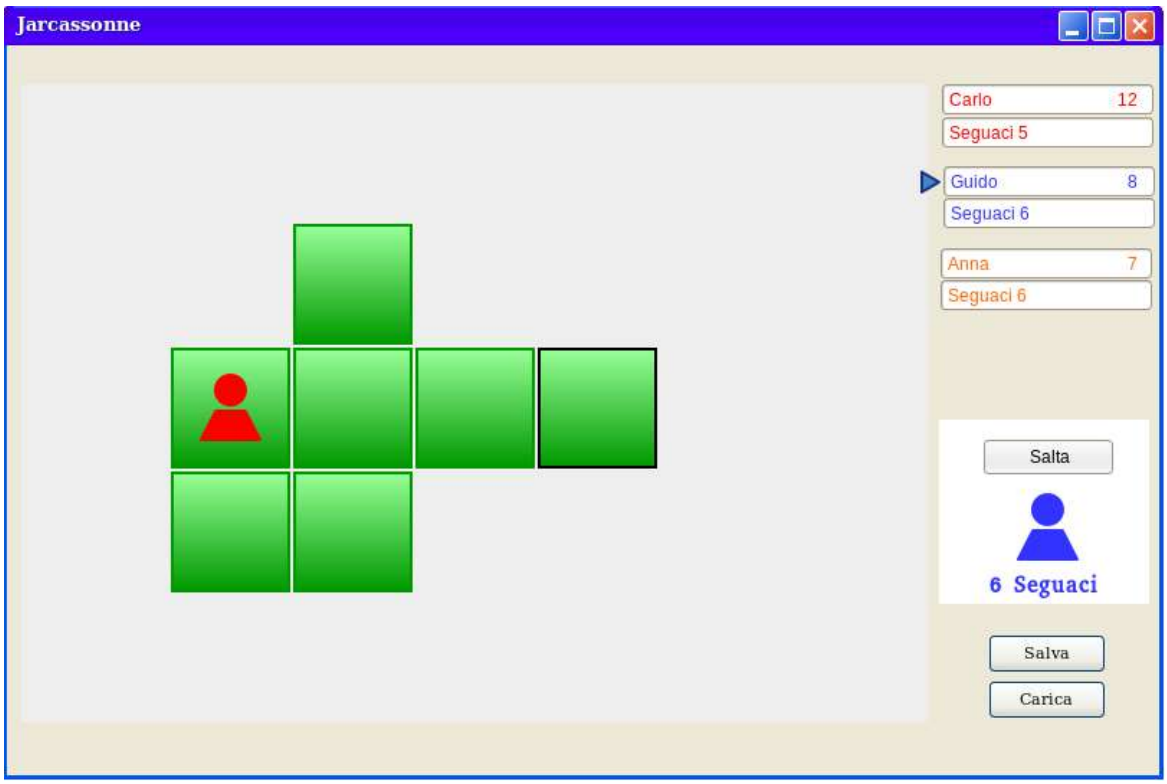
- Visualizzazione tramite icona del regolamento del gioco durante una partita
- Possibilità di far girare l'applicazione in ambiente Windows e Linux

Ecco come potrebbe apparire la GUI al giocatore durante le due fasi del proprio turno : (Fig 1) a destra della plancia,sulla quale è stato già piazzato un seguace, è visualizzato il punteggio di ogni giocatore con relativo numero di seguaci ancora a disposizione.I quadrati in verde indicano le tessere già posizionate mentre quelli in grigio si riferiscono alle possibili celle sulle quali è consentito piazzare la tessera corrente*(fig 1)*.



(fig 1)

La cella con bordi in nero indica la tessera appena posizionata sulla quale il giocatore potrà scegliere se disporre o meno il seguace.Qualora il giocatore decidesse di non collocare il seguace potrà cliccare sul tasto salta.Di fatto questa mossa pone fine al proprio turno (Fig 2)

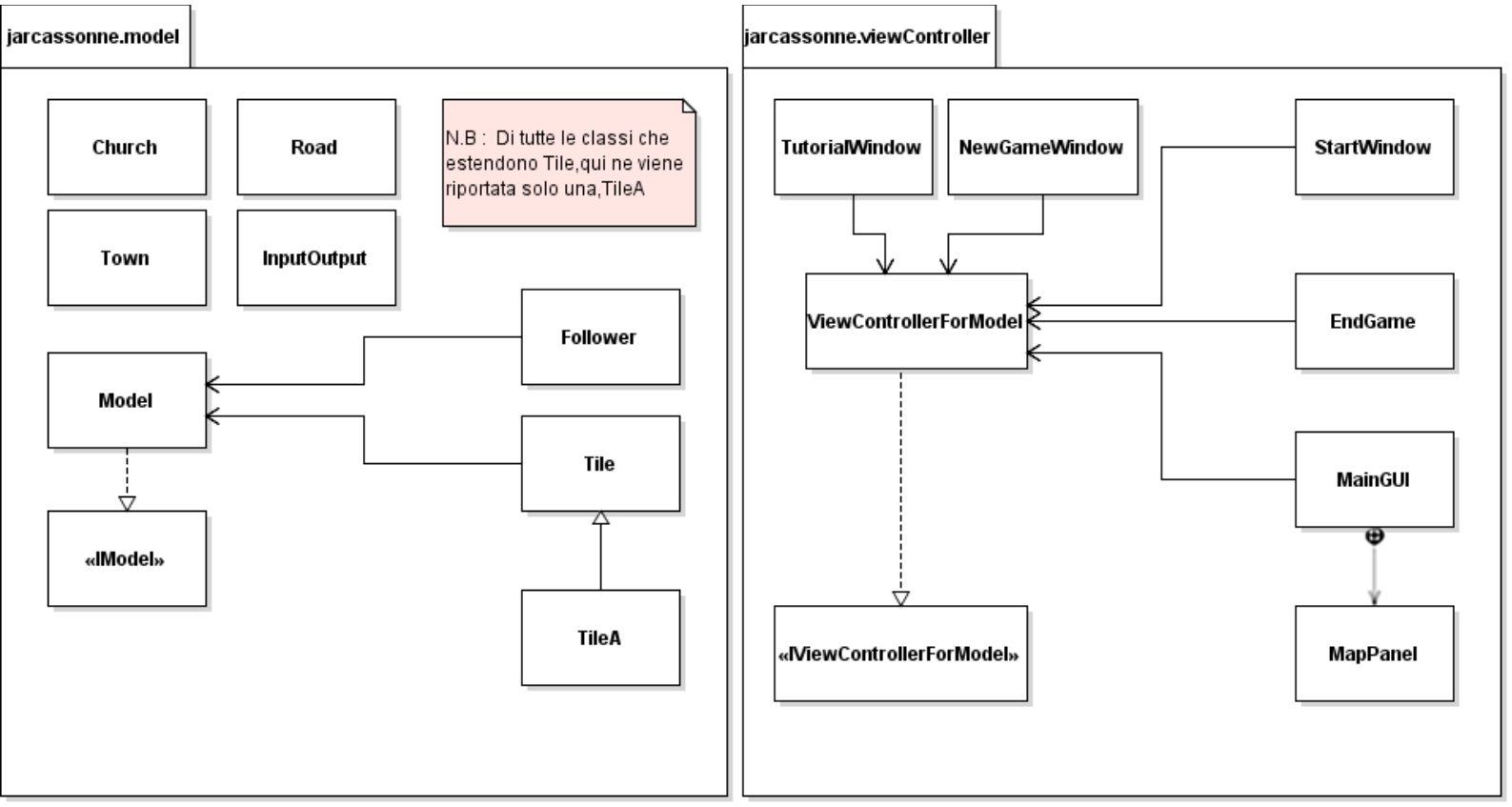


(fig 2)

CAPITOLO 3 - PROGETTO

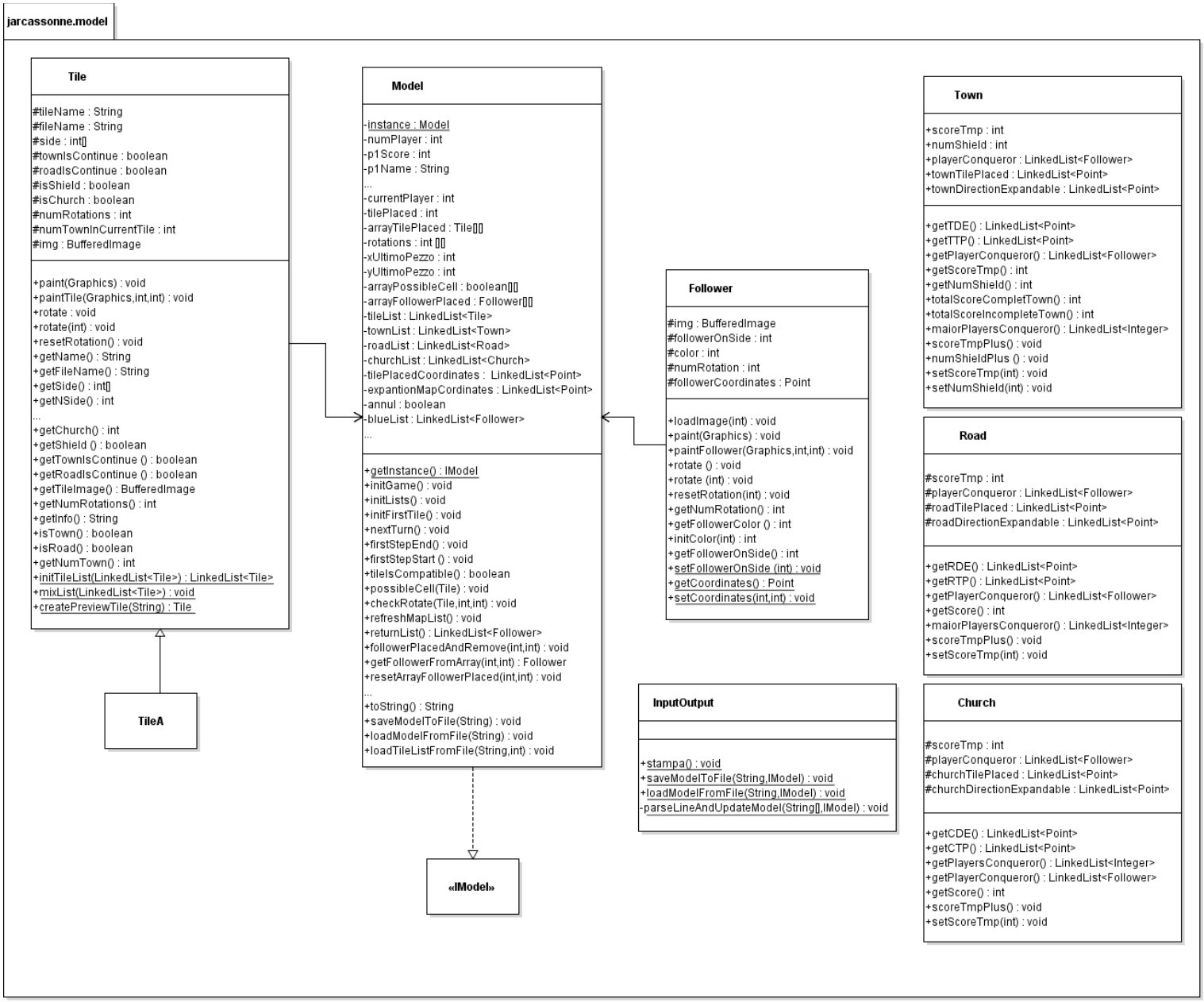
3.1) Architettura software

Nell'implementazione di Jarcassonne si è scelto di sviluppare l'architettura Software con approccio differente rispetto al consueto MVC,fondendo View e Controller in un unico package.Nell'applicazione infatti non sono presenti componenti in movimento con aggiornamento continuo dello stato del View; un aggiornamento della plancia di gioco si verifica solo a seguito di una mossa del giocatore.Si potrebbe dire che Jarcassonne non è di tipo Time Driven ma Event Driven.
Il Model si occupa di rappresentare i dati gestiti dall'applicazione : in modo particolare tiene traccia delle tessere posizionate sul tabellone e di quelle non ancora in gioco,dei seguaci e della città/strada/monastero a cui sono assegnati e delle liste di città/strada/monastero non ancora chiuse,con le loro possibili espansioni.Ruolo fondamentale è svolto in tal senso dalle classi Town,Road,Church presenti nello stesso package così come Tile e Follower che contengono le informazioni essenziali riferite a ciascun oggetto utilizzato durante il gioco.Ultima,ma non per importanza,la classe InputOutput che implementa le meccaniche di salvataggio e caricamento della partita.
Il ViewController provvede a gestire l'interazione con il mouse,definisce i metodi per la riproduzione del suono,modifica lo stato della plancia verificando la fattibilità di alcune operazioni e aggiorna le informazioni presenti nel Model.Si occupa inoltre di gestire le eccezioni a livello di regolamento.Fanno parte di questo package anche le GUI che guidano i giocatori nel primi istanti di gioco a seguito dell'avvio dell'applicazione o al termine della partita,oltre alla MainGUI che dispone dei metodi di visualizzazione dell'intero tabellone.



3.2) Descrizione dei moduli

jarcassonne.Model



sono stati inserite solo le variabili e i metodi principali delle classi descritte

Model

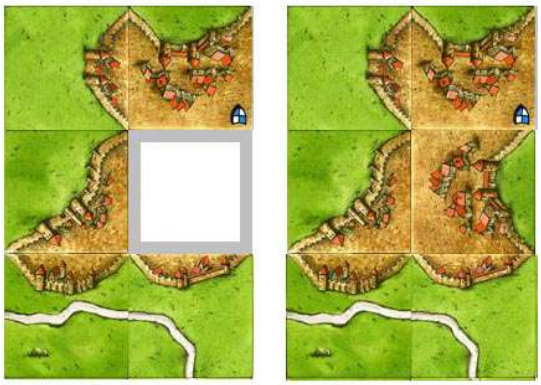
Il tabellone di gioco è definito da matrici di dimensioni 100x100, che non garantiscono il funzionamento di una partita interamente sviluppata in una dimensione (caso limite e non concorde alla logica del gioco) perchè si prevede ad ogni sfida un'espansione a "macchia d'olio" a partire dal primo pezzo sempre presente all'inizio di ogni partita .

Le matrici in questione sono :

- `arrayTilePlaced` : tiene traccia dei pezzi già posizionati dai giocatori sul tabellone
- `arrayPossibleCell` : salva le possibili compatibilità tra il pezzo da piazzare e la mappa già costruita.
- `arrayFollowerPlaced` : associa ad ogni seguace la tessera su cui è piazzato.
- `rotations` : salva per ogni pezzo il numero di rotazioni. È un'informazione necessaria per la stampa della mappa e per il caricamento di una partita.

Viene poi istanziata una lista che tiene traccia di tutte le città aperte : dall'elemento i-esimo di ciascuna è possibile risalire all'insieme dei pezzi che fanno parte dell'i-esima città.La logica di gioco prevede la fusione di due o più liste nel momento in cui due o più città vengono unite dal piazzamento di un pezzo.

Inserimento del pezzo centrale ha derminato l'unione di 4 città contemporanemanete



L'ultima mossa ha unito 3 città

Allo stesso modo esiste una lista di Road e una di Church.

Gestione dei seguaci: esistono 4 liste di Follower, una per colore, che memorizzano 7 oggetti Follower ciascuna.Quando un seguace viene posizionato sulla plancia l'oggetto Follower viene spostato sulle liste appositamente create degli elementi conquistati (`playerConqueror`). Quando si ha una chiusura,i seguaci memorizzati nelle liste conquistatori vengono restituiti,cioè inseriti nuovamente nelle liste del Model.

`tilePlacedCoordinates` e `expantionMapCoordinates` sono due liste di oggetti,adibite al salvataggio delle coordinate delle tessere posizionate che vanno a comporre la mappa e le coordinate del bordo,cioè l'espansione futura;

- Particolare interesse merita il metodo `initFristTile()` nel quale vengono inizializzati i precedenti attributi descritti.Inoltre provvede al piazzamento del primo pezzo che,in base al regolamento del gioco,risulta essere sempre lo stesso,poichè compatibile con tutti gli altri.

- `tileIsCompatible(...)` : verifica se una determinata tessera T possa essere inserita in modo coerente alla posizione (X,Y) passata come parametro.Questo metodo verifica la compatibilità dei lati per tutti gli orientamenti della tessera corrente.

- `refreshMapList()` : una volta posizionata una tessera devono essere aggiornate le liste che salvano mappa ed espansione,.Questo metodo si occupa di spostare le coordinate dell'ultimo pezzo dalla lista `expantionMapCoordinates` alla lista `tilePlacedCoordinates` e aggiungere qualora ci fosse bisogno le nuove coordinate di espansione.

- `possibleCell(...)`: invoca per tutte le coordinate dell' `expantionMapCoordinates` il metodo `tileIsCompatible(...)` e aggiorna l'array `arrayPossibleCell.` Inoltre questo metodo sposta il pezzo corrente,se non ci sono compatibilità,in fondo alla lista di pezzi non ancora usciti.

Il Model dispone anche di metodi quali get-set che non riportiamo, metodi legati alla gestione dei punteggi, degli attributi dei giocatori, gestione del turno(effettuata considerando il numero di tessere giocate mod numero dei giocatori), gestione dell'operazione di annullamento, e dei salvataggi-caricamenti.
Per il salvataggio si è scelto di creare un file con i dati strettamente necessari per immortalare lo stato del gioco. Ogni informazione viene salvata su una differente riga secondo un ordine preciso e nel caricamento il file è scandito riga per riga.

Town

Per descrivere l'oggetto città si è scelto di non utilizzare un array, valido per risalire velocemente da un qualunque pezzo agli adiacenti, per due motivi: il primo perchè l'array ha dimensioni fisse al contrario delle città che si creano durante una partita, il secondo perchè un array che salva la disposizione dei pezzi sulla mappa già esiste e si trova nel Model.Per salvare una città allora è sufficiente salvare le coordinate della sua estensione nella mappa e questo viene fatto in una lista, `townTilePlaced`. Si è introdotta anche un'altra lista che registra l'espansione futura della città e permette di stabilire in modo estremamente rapido quando una città è completa, ovvero quando la `townDirectionExpandable` risulta vuota.
Ad ogni oggetto città sono associati dei parametri che completano la sua descrizione:

- `scoreTmp` memorizza il punteggio temporaneo,
- `numShield` memorizza il numero di scudi appartenenti alla città,
- `playerConqueror` una lista che salva i giocatori che hanno conquistato la città (in alcuni casi sono più di uno e potrebbe accadere che un giocatore ha impegnato più di un seguace per conquistarla). Tra i metodi rilevanti di Town segnaliamo `maiorPlayersConqueror()` che si occupa di stabilire quale giocatore "domina" la città.

Road

L'oggetto Road ricalca strutturalmente Town, infatti è composto da due liste di coordinate con funzionalità identiche a quelle già viste, il punteggio temporaneo e la lista di seguaci conquistatori.

Church

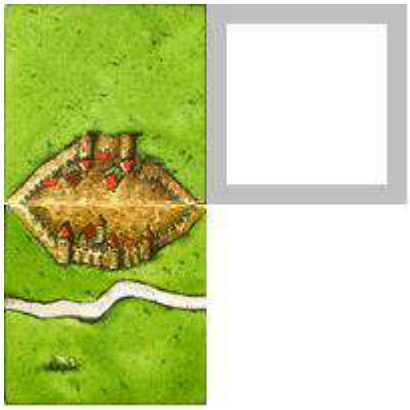
Similmente a Town e Road è stato modellato il monastero,nonostante sarebbe stato possibile implementarlo tramite array. La scelta di usare comunque le liste è stata fatta per avere strutture simili, più ordinato da un punto di vista logico, e operare più facilmente su di esse.

Tile

La modellazione dei pezzi è basata sulla classe Tile : in essa troviamo attributi che descrivono le caratteristiche peculiari delle tessere,variamente inizializzati dalla classe *TileSpecifico* (TileA,TileB...) che la estendono.

Ogni singolo pezzo è modellato dalle informazioni relative ai suoi lati,conservate all'interno dell'array `side`,tramite le quali è possibile calcolare le compatibilità tra pezzi sulla plancia per determinarne i possibili incastri.

Tra gli attributi riferiti alle città i più importanti sono `numTownInCurrentTile` e `townIsContinue`.Il primo serve principalmente nella definizione di nuove città perchè un singolo pezzo può portare all'apertura contemporanea di due città e questa informazione va gestita accuratamente.



il posizionamento dell'ultimo pezzo ha portato all'apertura contemporanea di due città

Il secondo è di grande importanza nella chiusura poichè serve a verificare che una città non è ulteriormente espandibile in una certa direzione.Se essa non lo è in nessuna direzione la città è da considerarsi chiusa.

la città è continua ed espandibile nella direzione est

La classe Tile contiene anche il metodo mixList tramite cui,all'inizio di ogni partita,è possibile ottenere un ordine casuale delle tessere che garantisce ogni volta differenti sfide.

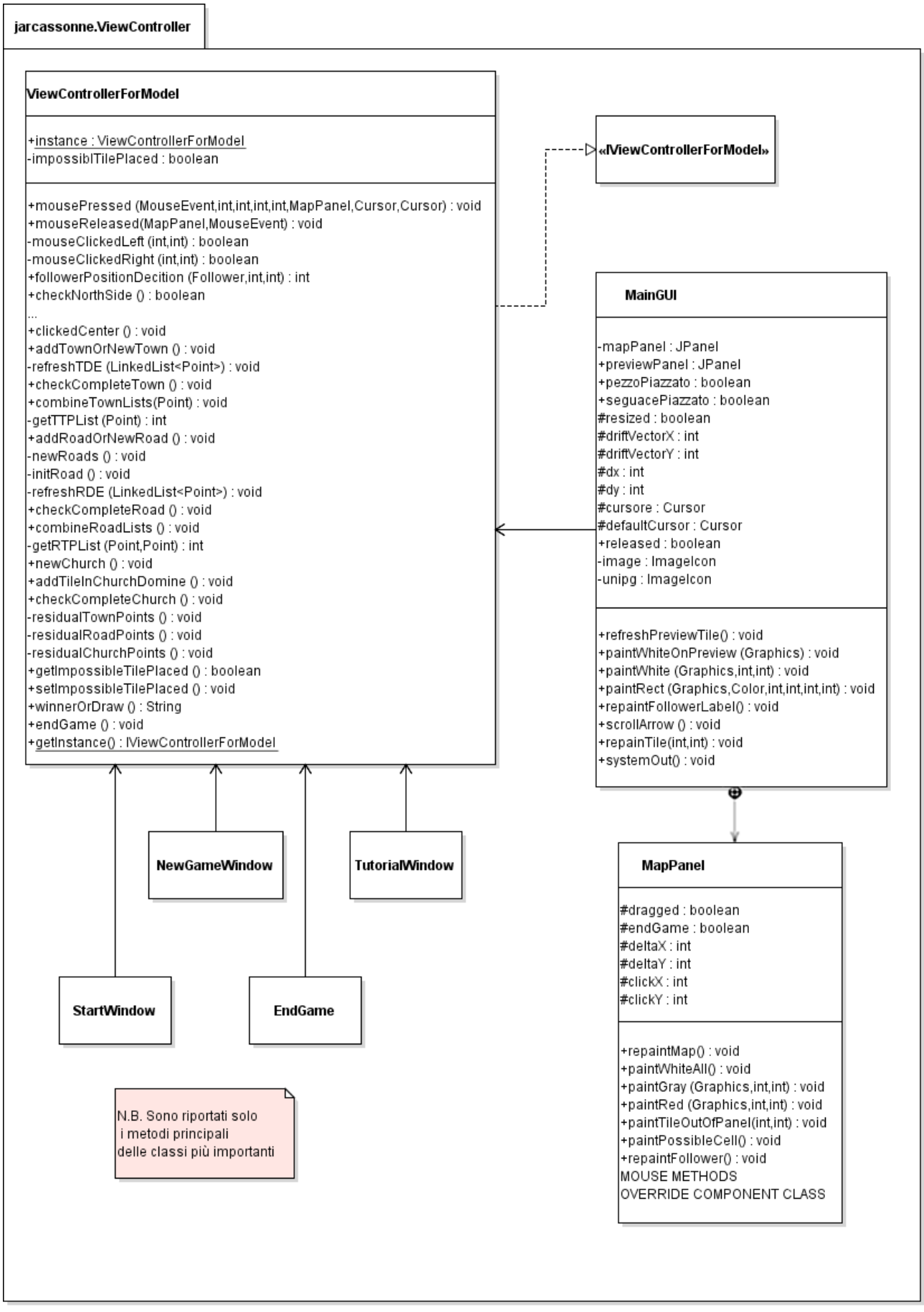
Follower

Un seguace posizionato sulla tessera per essere descritto e rappresentato deve avere tra gli attributi: l'immagine del giusto colore associata, il colore(rappresentato tramite costanti), il lato su cui è stato posizionato che serve ad individuare le coordinate per la stampa all'interno della tessera, il numero di rotazioni dell'immagine ed infine le coordinate della tessera a cui è associato.
I metodi sono metodi di utilità simili a quelli della classe Tile. Il blocco statico e i metodi statici sono utilizzati all'avvio del gioco per creare i componenti necessari alla partita.

InputOutput

È stato necessario creare una classe per gestire le operazioni di salvataggio e caricamento. La classe InputOutput ha metodi per scrivere il file tramite cui viene salvato il Model e metodi per leggerlo e riportarlo ad un certo stato.
Nel caricamento per leggere il file che contiene informazioni diverse per ogni stato differente della partita, quindi lunghezze differenti e non note a priori, è stato necessario introdurre variabili di stato che permettono di scorrere le righe del file salvataggio e leggerlo in modo dinamico.

Jarcassonne.ViewControllerForModel



Questo package presenta diverse tipologie di classi : alcune sono estensioni della classe JFrame che guidano il giocatore negli istanti successivi all'avvio dell'applicazione o al termine della partita,le altre contengono l'insieme dei metodi per la stampa delle componenti grafiche del gioco e per il controllo delle operazioni da mouse.

MainGUI

Il MainGUI contiene la plancia del gioco e in esso sono presenti i metodi che gestiscono tutte le operazioni di stampa,dai pezzi ai seguaci,dalla preview al possibleCells.
La classe MainGUI presenta la classe interna MapPanel al fine di gestire gli eventi del mouse e di stampa all'interno del pannello.Ogni click del mouse è in grado di innescare una serie di controlli e modifiche,eventualmente sul Model,che è necessario gestire: perciò `mousePressed(...)` e `mouseReleased(...)` richiamano i corrispondenti metodi nella classe ViewControllerForModel,a seguito di un'azione verificatasi nel mapPanel.
Tra i metodi relativi alla stampa ritroviamo :

- `repaintMap()` che viene invocato tutte le volte che l'utente shifta la plancia,in modo da ridisegnare le tessere e i seguaci,rileggendo le rotazioni così da ristampare le tessere secondo la giusta rotazione imposta in precedenza.

- Per far ciò vien sfruttato il `paintWhiteAll()`: qui piuttosto che provvedere a una ristampa selettiva di alcune tessere,si è preferito ristampare tutto di bianco ed eseguire il `repaintMap()`.Quest'ultimo presenta un'elevata complessità,poichè,per la stampa,devono essere recuperate le informazioni riguardo alle tessere in gioco,le loro rotazioni e il posizionamento corretto dei seguaci sul giusto lato della tessera.La ristampa di alcuni pezzi mostrava però inefficienza poichè rari i casi in cui,a seguito di un dragged,una parte delle tessere veniva mantenuta ovvero non ristampata,poichè già nella corretta posizione.

- Il metodo `paintTileOutOfPanel()`, servendosi di `paintRed(...)`,consente al giocatore di sapere se ci sono pezzi non visualizzati per intero,la cui espansione è certamente non visibile,se non attraverso uno shift della mappa.In tal modo si consente una visione non ristretta alla sola parte visibile della plancia,così da supportare il giocatore nel piazzamento della tessera e del seguace,in base alla strategia intrapresa.

- Le stesse considerazioni sono state effettuate nel caso del metodo `repaintTile(...)` : ogni volta che una strada,un monastero o una città viene chiusa,il seguace può rientrare nella lista dei Follower disponibili per quel giocatore;in questo caso,anzichè ristampare tutta la mappa si è

rivelato sufficiente ristampare la sola tessera sulla quale il segnalino era stato piazzato,al fine di alleggerire la fase del gioco in cui viene calcolato anche il punteggio ottenuto al termine della chiusura.Nella classe interna tuttavia,essendo modellata la plancia di gioco,ritroviamo anche i metodi di ristampa della mappa e definizione della griglia di gioco,presente anche se non visualizzata a video.

ViewControllerForModel

E' la classe che realizza unitamente il Controller e il View.Il primo è attuato dai metodi che gestiscono : l'input da mouse,gli effetti audio,il termine della partita e il controllo di alcune operazioni,in modo tale da rispettare le regole base del gioco.Dal punto di vista del View ritroviamo l'insieme dei metodi che curano l'apertura/chiusura delle finestre dell'applicazione.I metodi rilevanti sono :

- `mousePressed` (...) –> si occupa di gestire la sequenza di click che un giocatore effettua durante l'intero turno.E' realizzato tramite una serie di booleani che suddividono il turno in diverse fasi,in ognuna delle quali,a ogni input differente (tasto destro,sinistro,centrale),corrisponde il corretto susseguirsi di operazioni.Per rendere questo possibile è stato necessario passare in input un oggetto mapPanel

- `mouseReleased` (...) –> tramite questo metodo viene effettuata la traslazione della mappa a video.In prima istanza calcola il vettore traslazione come differenza tra le coordinate del punto in cui si è clickato con il mouse al punto in cui è stato rilasciato.In seconda battuta ristampa l'intera mappa,tramite il metodo repaintMap(),parametrico rispetto al vettore traslazione.

Alcuni metodi sono relativi al piazzamento dei seguaci :

- `followerPositionDecition(...)` determina,all'interno di una tessera,a partire dalle coordinate del click,qual è l'elemento su cui si intende disporre il Follower,ruotandolo opportunamente.

- `checkNorthSide()` viene invocato quando si intende piazzare il seguace nella parte superiore della tessera e il suo compito è controllare che il piazzamento sia valido.Analogamente esistono metodi per gli altri lati e il centro della tessera.

Ci sono i metodi che operano sull'oggetto città :

- `addTownOrNewTown()` : viene invocato ogniqualvolta si posiziona un Tile,avente almeno un lato città,controllando se la tessera espande una città già creata oppure ne apre una nuova

- `refreshTDE (...)` *aggiorna la lista delle espansioni di una determinata città,a seguito del piazzamento del pezzo*
- `checkCompleteTown ()` : *ad ogni fine turno verifica che qualunque TDE sia diversa dalla lista vuota.In quel caso la città è da considerarsi completa e viene aggiornato il punteggio a video del/dei giocatore/i,resituendo il seguace al proprietario e rimuovendo dalla lista delle città la città stessa*
- `combineTownLists(...)` *se posizionando un pezzo si uniscono due o più città vengono fusi tra loro gli oggetti città,salvati nel Model.*

Per strade e monasteri sono stati implementati metodi analoghi,modificati opportunamente.
E' presente un metodo `endGame()` *che,al termine della partita,calcola i punteggi residui e apre la GUI che mostra statistiche e vincitore.*
Tramite i metodi `runTrack()`*,che seleziona una delle possibili tracce presenti in Jarcassone,e* `playSound(...)` *vengono riprodotti effetti audio,come da specifiche.*

3.3) Problemi riscontrati

- *La modellazione delle tessere è stata essenzialmente divisa in due parti : in una prima fase è stata necessaria l'acquisizione delle immagini,a esse associate,tramite BufferedImage e successivamente la creazione di tutte le 72 tessere,divise nelle 24 tipologie diverse,tramite blocco statico.L'acquisizione tentata in un primo momento risultava essere incompatibile con l'eseguibile jar,poiché la finestra principale di gioco,dopo aver scelto il numero di giocatori partecipanti,non veniva visualizzata. Ciò perchè il blocco statico di istanziazione delle tessere contiene un blocco try - catch : se il caricamento dei pezzi non avviene correttamente e completamente la finestra di gioco MainGUI non è visualizzata.*

Prima soluzione incompatibile con il Jar

```
try{
    img=ImageIO.read(new File(getPathPhoto()+fileName));
}
catch(IOException e){
        System.out.println("error:file TILE path");
}
```

Soluzione definitiva adottata

```
try{
     img=ImageIO.read(NewGameWindow.class.getResourceAsStream(
          "/jarcassonne/img/Tessere_gioco/"+this.fileName));
}
catch(IOException e){
        System.out.println("error:file TILE path");
}
```

- *La modellazione delle tessere è inoltre arricchita dalle informazioni riguardanti i suoi quattro lati (TOWN – FIELD – ROAD) e la sfida è stata quella di posizionare il seguace al giusto lato ovvero assegnarlo alla giusta espansione (città/strada) che il giocatore corrente aveva intenzione di conquistare,se ovviamente non presidiata già da un altro avversario.Nel caso monastero senza strada (Tile B) è stato fatto in modo che il seguace potesse essere piazzato solo al centro della tessera mentre ,se presente ,anche sulla strada (Tile A).*
  Perciò ogni tessera è stata divisa in quattro parti così da rilevare univocamente le quattro diverse aree in base alle coordinate del click. Come dichiarato nelle specifiche di gioco,ricordiamo che non è possibile piazzare un seguace sul lato prato (in esso il click è pertanto muto).

- Prima della modifica,due pezzi uguali facevano riferimento allo stesso pezzo,istanziato nel blocco statico in memoria;se ruotati in modo differente l'ultimo pezzo determinava le informazioni sull'altro.Accadeva quindi che il metodi possibleCell(),che si occupa di stabilire le compatibilità,non mostrasse alcuni possibili accoppiamenti corretti o ne mostrasse altri errati.Anche il repaintMap() presentava malfunzionamenti in quanto copie diverse dello stesso pezzo venivano ristampate tutte con lo stesso orientamento.Per risolvere questi problemi è stato introdotto il previewTile che ha la funzione di salvare temporanemente le informazioni di un pezzo e modificarne gli attributi,evitando di operare direttamente su quelle dei pezzi salvati nel blocco statico.In questo modo è possibilile sistemare due pezzi identici vicini in modo coerente alle regole del gioco e ristampare correttamente la mappa.

- Sempre legato al problema dei pezzi simili è stato necessario introdurre il metodo `resetRotation()` che assicura che l'immagine di un pezzo piazzato non ancora in gioco possa essere svincolata dalla sua copia già posizionata sulla plancia e non,ad esempio,apparire già ruotata in preview qualora la prima fosse stata ruotata dal giocatore o dalla logica di gioco per poter essere incastrata.

- Per rendere migliore l'esperienza di gioco si è pensato di poter effettuare un'intera partita esclusivamente da mouse,non avvalendosi dei tasti SKIP – UNDO – CONFIRM sul lato destro della MainGUI. Per fare questo è stato necessario suddividere il turno di un giocatore in più fasi e ad ognuna associare il corretto comportamento del mouse. Sono state introdotte delle variabili booleane che, oltre ai metodi di gestione del mouse, scandiscono il susseguirsi di determinati eventi (il pezzo è piazzato sul tabellone, ruotare il pezzo appena messo, annullare la mossa, confermare la mossa, piazzare il seguace, non piazzare il seguace). La difficoltà è stata sincronizzare le variabili e le fasi del turno in modo da non generare situazioni ambigue e malfunzionamenti.

- Ridimensionamento: quando la finestra di gioco cambia dimensioni è necessario ristampare l'intera plancia. Poichè il metodo

`repaintMap()`,dovendo recuperare molte informazioni, impiega tempo, eseguire delle chiamate in successione provocava dei flash ravvicinati.La soluzione definitiva adottata prevede,pertutti,uno stato di pausa a seguito del ridimensionamento.Con un click su un qualunque punto della mappa si potrà riprendere la partita.

- `possibleCell()`: è il metodo che si occupa di stabilire dove un pezzo può essere posizionato. Per fare questo in un primo momento si valutava il pezzo corrente(in tutti i sui possibili orientamenti) per tutte le celle dell'array di pezzi(la plancia). Questa soluzione era altamente inefficiente tanto da riscontrare nel gioco la pesantezza del metodo.E' stata pertanto introdotta una lista che salva le coordinate dei pezzi piazzati e una per l'espansione(il bordo della mappa creata). Anche se sono informazioni gia salvate e quindi si introduce una ridondanza, il possibleCell migliora perchè deve valutare il pezzo corrente solo per la lista espansione. La lista dei pezzi piazzati per lo stesso motivo torna comoda per il `repaintMap()`.

- Durante la partita può accadere che il pezzo corrente non possa essere sistemato sulla plancia perchè incompatibile.Per evitare il "deadlock" è stato necessario spostare il pezzo in coda alla lista dei pezzi non ancora in gioco.Questa soluzione è valida perchè si sposta il problema a fine partita,dove la `expantionMapCoordinates` è massima e dunque sono pressocchè nulle le probabilità che si verifichi nuovamente un'incompatibilità.

- Quando il numero dei pezzi piazzati è considerevole può accadere che non sia interamente visualizzata la mappa a video.Perciò si rende necessario un meccanismo che consenta la visualizzazione dei pezzi "nascosti".Si è deciso di non utilizzare nè ridimensionamenti nè scrollbar ma si è implementato un meccanismo di spostamento per trascinamento.In aggiunta i pezzi non completamente visualizzati sulla plancia vengono evidenziati da una cornice rossa,in virtù della contiguità della mappa,proprietà caratterizzante del gioco.

- Sul alcuni calcolatori,dopo aver selezionato il numero dei giocatori e cliccato su start,veniva visualizzata la MainGUI ma non il primo pezzo già piazzato sul tabellone e neanchè il pezzo in preview,come se ci fosse stato un ridimensionamento della finestra di gioco,nonstante dimensioni e risoluzione dello schermo di tali calcolatori fossero le stesse dei computer su cui il problema non era presente.La soluzione definitiva adottata prevede,per tutti,un primo click su un pannello bianco,con solo una Jlabel che informa il giocatore di effettuare questa prima operazione per poi iniziare la partita.

CAPITOLO 4 - EVENTUALI SVILUPPI FUTURI

Un possibile sviluppo di Jarcassonne potrebbe prevedere un'applicazione web per effettuare una partita da diversi calcolatori contemporaneamente,sfruttando le classi descritte nell'API di Java,quali URLConnection,Socket e ServerSocket.
Un'altra idea potrebbe prevedere l'intelligenza virtuale,sviluppando algoritmi ottimali di piazzamento di tessere e seguaci Un singolo giocatore potrà così effettuare una partita contro il computer.

CAPITOLO 5 – BIOGRAFIA

Risorse utilizzate in fase di elaborazione del progetto:

- *H.Schildt* "Java,La guida completa" , McGraw-Hill (7° ed)
- *Mazzanti-Milanese* "Programmazione di applicazioni grafiche in JAVA" , Apogeo
- http://www.giocopocomagioco.com/istruzioni-giochi/340-manuale-istruzioni-regole-gioco-carcassone.html
- https://docs.oracle.com/javase/7/docs/api/
- https://docs.oracle.com/javase/tutorial/2d/images/drawimage.html
- https://docs.oracle.com/javase/tutorial/2d/advanced/transforming.html
- Materiale didattico fornito dal docente
- Nomenclatura.jpg : file di riferimento per la corrispondenza nome - immagine del pezzo