JKnightTour

Tesina di laboratorio di informatica grafica

Indice generale

1 Descrizione del Problema

Obbiettivi di questo lavoro sono la progettazione e la realizzazione di un'applicazione grafica dedicata alla risoluzione del problema del “Giro del Cavallo”, sia per via automatica che manuale.

Il Problema del Giro del Cavallo

Il problema del “Giro del Cavallo” (conosciuto in lingua inglese come Knight's Tour) consiste nel trovare un percorso lungo una scacchiera, composto da mosse del tipo eseguibile da un cavallo nel gioco degli scacchi, che passi per tutte le caselle della scacchiera una sola volta. In particolare va notato che:

1. La posizione iniziale del cavallo è arbitraria.
2. Le dimensioni della scacchiera sono arbitrarie, non è quindi detto che il problema vada affrontato su di una tavola da scacchi standard con 8x8 caselle.
3. Si può aggiungere anche il vincolo che il percorso da individuare sia chiuso, ovvero che il cavallo debba passare per tutte le caselle della scacchiera per poi tornare alla posizione di partenza. Per questo progetto tralasciamo questo vincolo aggiuntivo.

Un esempio di percorso ammissibile è riportato in figura 1:

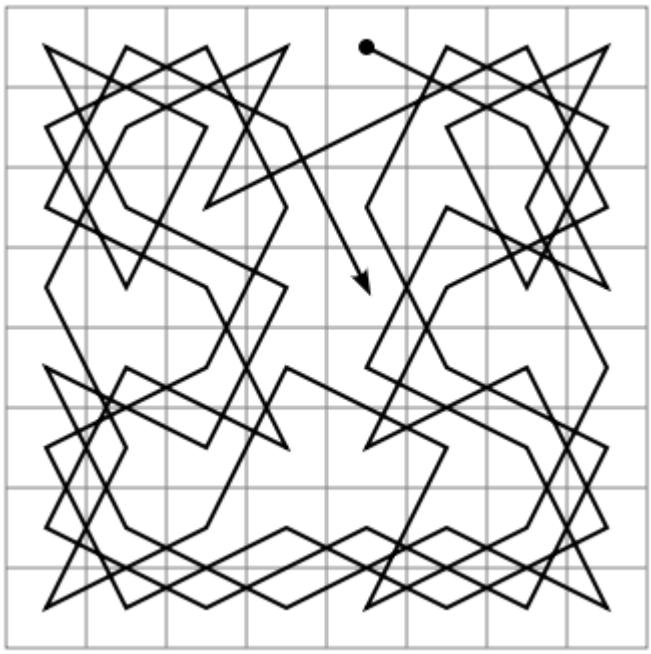

Figura 1: Giro del cavallo su scacchiera standard

Questo problema può essere formulato in maniera più formale utilizzando il linguaggio dei grafi. Ogni casella della scacchiera può essere vista come il vertice di un grafo i cui archi connettono due posizioni tra le quali il cavallo può “saltare”. Il problema si riduce quindi alla ricerca di un percorso che copra tutti i vertici senza passare due volte per lo stesso vertice. Questo problema è conosciuto in letteratura matematica come “Hamiltonian path problem”. Nel caso generale si dimostra che il problema è NP-completo mentre nel caso

particolare del giro del cavallo esistono algoritmi capaci di trovare la soluzione in tempo lineare.

L'applicazione JKnightTour

L'applicazione JKnightTour si propone di fornire un'interfaccia che, tramite una rappresentazione grafica della scacchiera offra all'utente due funzionalità:

1) Affrontare la risoluzione manuale del problema:
   I. Inizialmente l'utente sceglie una posizione di partenza e le dimensioni della scacchiera.
   II. Il programma evidenzia ad ogni passo le caselle raggiungibili dal cavallo, tra le quali l'utente può scegliere la mossa successiva.
   III. In ogni momento l'utente può tornare sui propri passi scegliendo una delle posizioni precedentemente attraversate.
2) Chiedere la risoluzione automatica del problema:
   I. L'utente sceglie le dimensioni della scacchiera ed il punto di partenza.
   II. L'applicazione calcola un percorso e una volta trovata una soluzione la visualizza con un'animazione.

2 Specifica dei requisiti

I requisiti per l'applicazione JKnightTour possono essere suddivisi per comodità in tre parti distinte: requisiti generali, requisiti per la modalità manuale e requisiti per la modalità automatica.

Requisiti generali

Vengono ora elencati i requisiti richiesti all'applicazione nel suo complesso:

- Il programma deve prevedere due modalità di funzionamento, una “manuale” in cui l'utente tenta di risolvere a mano il problema ed una “automatica” in cui il programma risolve autonomamente il problema. Le due modalità di funzionamento devono essere indipendenti: l'utente deve poter passare dall'una all'altra in qualsiasi momento e lo stato della scacchiera in una modalità non deve influenzare in alcun modo l'altra.
- L'interazione con l'utente si deve svolgere il più possibile per via grafica, privilegiando l'utilizzo del solo mouse per gestire tutte le funzionalità.
- L'interfaccia deve essere pulita e comprensibile, mettendo immediatamente a disposizione dell'utente tutti i controlli necessari evitando se possibile il ricorso a menù ed a finestre di configurazione.
- La scacchiera su cui si svolge il gioco deve essere l'elemento dominante dell'interfaccia grafica. La rappresentazione delle mosse e della posizione del cavallo deve essere chiara ed immediata.
- Tutti i movimenti del cavallo sulla scacchiera devono essere presentati all'utente tramite un'animazione. L'animazione deve essere fluida.
- Deve essere possibile cambiare le dimensioni della scacchiera.

- I colori della scacchiera devono essere modificabili tramite un opportuno file di configurazione.

Requisiti per la modalità manuale

Vengono ora elencati i requisiti richiesti per l'applicazione relativi alla modalità di funzionamento manuale.

- Il programma deve mostrare chiaramente in ogni istante il percorso seguito fino a quel momento dal cavallo. Deve essere facile per l'utente individuare quali sono le caselle già attraversate.
- L'utente muove il cavallo facendo click con il mouse sulla casella che vuole raggiungere.
- L'applicazione deve evidenziare in ogni momento all'utente quali sono le caselle della scacchiera raggiungibili con una mossa.
- L'applicazione deve mantenere in vista un elenco delle mosse eseguite, con la possibilità di ritornare ad una qualsiasi delle mosse precedenti.
- L'applicazione deve avvertire con una notifica il raggiungimento di un “vicolo cieco” o della fine del percorso, che corrisponde alla vittoria.
- Deve essere possibile salvare lo stato attuale della partita e caricare uno stato di gioco precedentemente salvato.

Requisiti per la modalità automatica

Vengono ora elencati i requisiti richiesti per l'applicazione relativi alla modalità di funzionamento automatico.

- L'utente indica la posizione di partenza per il percorso facendo click sulla casella corrispondente della scacchiera.
- L'applicazione deve dare la possibilità di interrompere la ricerca di una soluzione in qualsiasi momento, dato che il processo potrebbe richiedere anche tempi piuttosto lunghi.
- Una volta trovata la soluzione questa deve essere visualizzata tramite un'animazione. L'animazione deve poter essere messa in pausa o portata direttamente alla fine.
- La parte dell'applicazione che si occupa di risolvere il problema deve essere estensibile: tramite un file di configurazione è possibile specificare un diverso algoritmo risolutivo.

3 Progetto

Viene ora descritta la struttura dell'applicazione realizzata, illustrandone prima l'architettura software per poi scendere nel dettaglio dei blocchi funzionali che la compongono.

Architettura software

Per la realizzazione di JKnightTour si è scelto di basarsi sul pattern di programmazione MVC (Model View Controller). Il Model si occupa di

rappresentare i dati gestiti dall'applicazione: in questo caso lo stato della scacchiera, la posizione corrente del cavallo ed il percorso da esso seguito. Il View si occupa di rappresentare i dati all'utente raccogliendone l'input: in questo caso è costituito dall'interfaccia grafica. Il Controller si occupa di gestire l'input dell'utente e la logica di funzionamento dell'applicazione.

Con il diagramma in figura 2 viene rappresentata l'architettura software implementata:

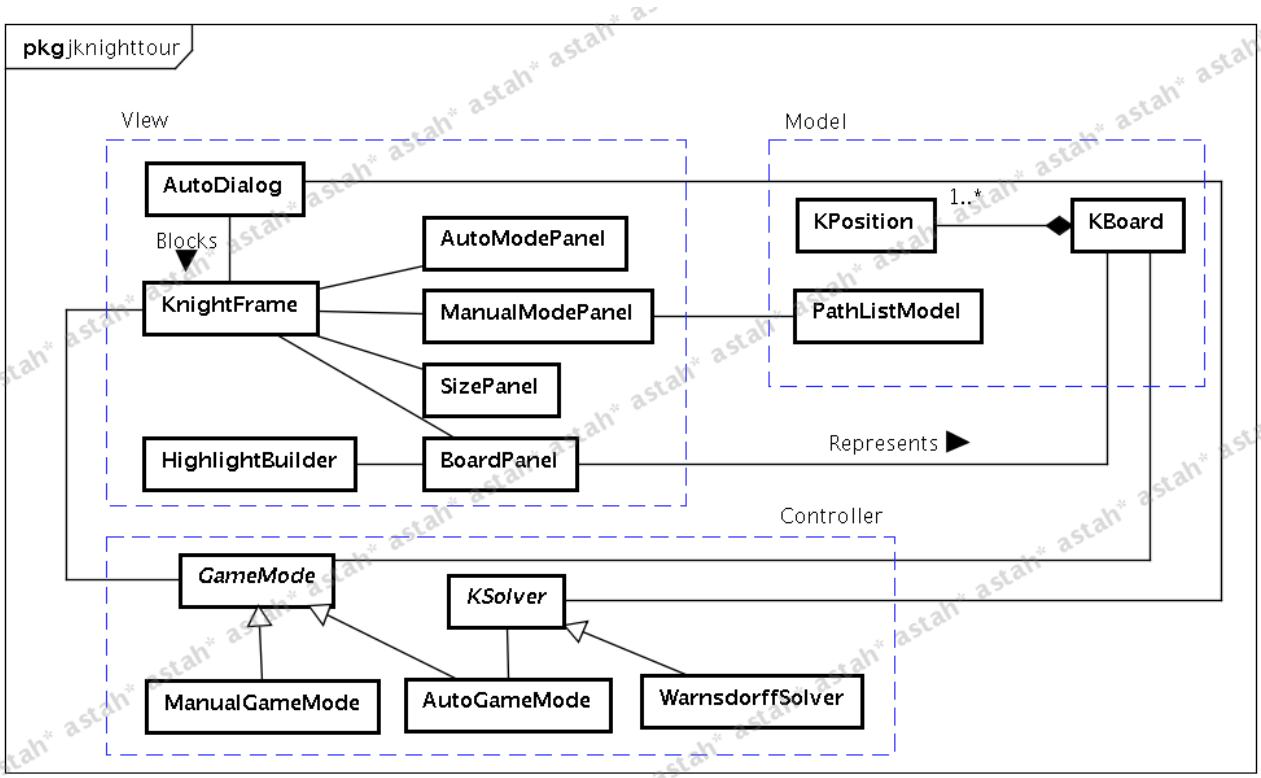

Figura 2: Architettura dell'applicazione

Oltre alle classi rappresentate nel precedente schema l'applicazione contiene le due classi JKnightTour e Utilities. La prima si occupa di lanciare l'interfaccia grafica su di un thread separato all'avvio del programma, la seconda fornisce alcuni metodi statici di utilità (salvataggio e caricamento, caricamento delle immagini e caricamento delle impostazioni dal file jknighttour.ini).

Di seguito vengono descritti nel dettaglio Model, View e Controller e le relative classi.

Model (jknighttour.model)

Le classi appartenenti al blocco Model di JKnightTour si trovano nel package jknighttour.model. La loro struttura è rappresentata nel diagramma UML in figura 3.

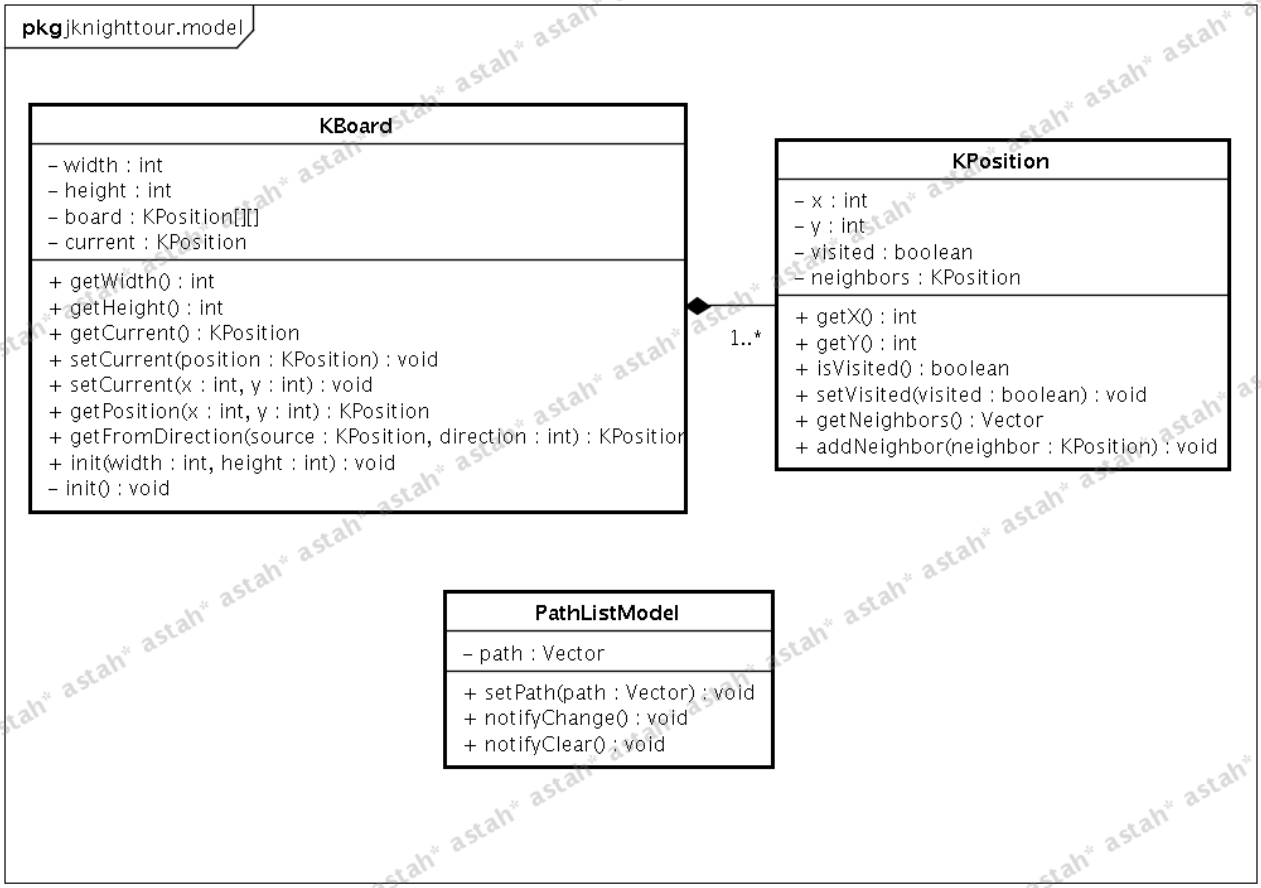

Figura 3: Diagramma UML delle classi di jknighttour.model

In particolare:

- **KPosition**: rappresenta una posizione (casella) nella scacchiera. Oltre a memorizzare le proprie coordinate, contiene un Vector (neighbors) con tutte le posizioni raggiungibili dal cavallo con una mossa a partire da quella casella e un boolean che viene impostato a true se la casella è stata visitata.
- **KBoard**: rappresenta la scacchiera. Memorizza le dimensioni della scacchiera ed un array di KPosition con tutte le caselle, inoltre tiene traccia della posizione corrente del cavallo. KBoard mette anche a disposizione un metodo per determinare la casella raggiunta in un movimento dal cavallo data una posizione di partenza ed una direzione (la direzione è un intero da 0 a 7).
- **PathListModel**: collega un Vector di KPosition alla lista delle mosse eseguite dal cavallo, contenuta in ManualModePanel.

Va notato che la struttura dati ottenuta con le KPosition non è altro che un grafo rappresentato con liste di adiacenza. Questa particolare struttura dati è stata scelta in vista dell'utilizzo dell'algoritmo di Warnsdorff per la risoluzione automatica del problema: il requisito essenziale dell'algoritmo infatti è quello di poter accedere rapidamente alle posizioni adiacenti ad una posizione data.

In tutta l'applicazione un percorso del cavallo sulla scacchiera viene rappresentato semplicemente attraverso un Vector di KPosition che contiene nell'ordine le caselle attraversate dal cavallo.

View (jknighttour.view)

Le classi appartenenti al blocco View di JKnightTour si trovano nel package jknighttour.view. La loro struttura è rappresentata nel diagramma UML in figura 4.

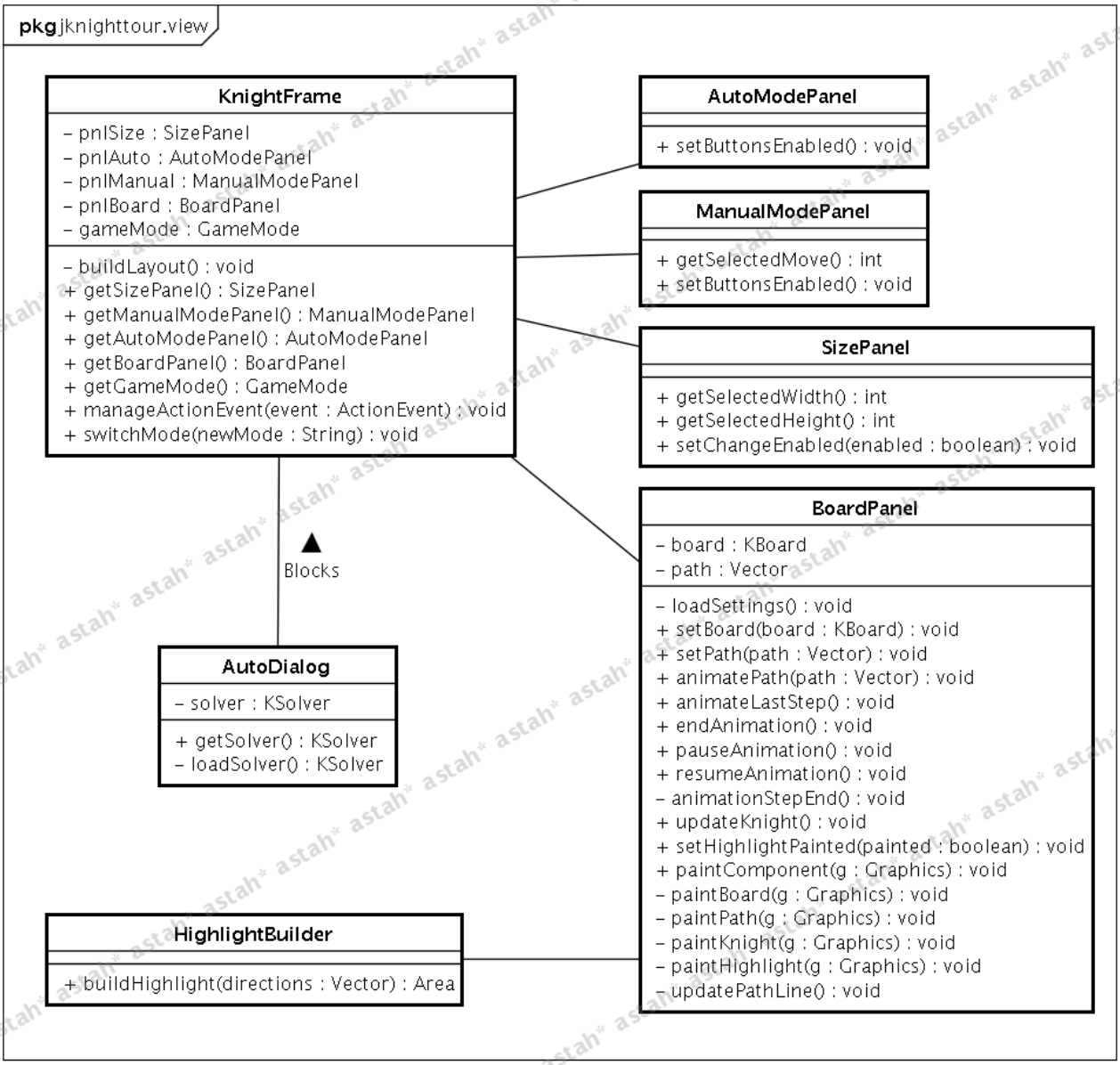

Figura 4: Diagramma UML delle classi di jknighttour.view

KnightFrame estende JFrame e costituisce la finestra principale dell'applicazione. Al suo interno si trovano 4 pannelli, ciascuno istanza delle classi AutoModePanel, ManualModePanel, SizePanel e BoardPanel. Queste classi definiscono ciascuna una parte specifica dell'interfaccia:

- **AutoModePanel**: Contiene un'insieme di bottoni per il controllo dell'applicazione nella modalità automatica di funzionamento: “Risolvi” per avviare la ricerca di una soluzione; “Reset” per resettare la scacchiera; “Play” per avviare la riproduzione dell'animazione del percorso calcolato; “Pausa” per mettere in pausa l'animazione; “Vai alla fine” per visualizzare direttamente tutto il percorso calcolato.

- **ManualModePanel**: Contiene controlli necessari nella modalità manuale di funzionamento del programma. Visualizza una JList con le mosse eseguite, un tasto “Reset” per riavviare il gioco, un tasto “Salva” ed un tasto “Carica” per salvare o caricare le partite.
- **SizePanel**: Contiene due JSpinner dedicati alla selezione di larghezza ed altezza della scacchiera ed un bottone per applicare la modifica delle dimensioni.

BoardPanel, svolgendo il ruolo fondamentale di rappresentare la scacchiera merita una descrizione più approfondita. Questo componente conserva un riferimento all'oggetto KBoard che rappresenta lo stato corrente della scacchiera e ad un Vector<KPosition> che rappresenta il percorso seguito dal cavallo. Ad ogni repaint di BoardPanel il metodo paintComponent() richiama nell'ordine i metodi privati paintBoard(), paintPath(), paintKnight() e paintHighlight(), ciascuno dei quali si occupa di disegnare una parte specifica della scacchiera:

- paintBoard(): disegna lo sfondo quadrettato della scacchiera su di una buffered image, paintComponent() poi disegna direttamente questa buffered image. Questo accorgimento permette di rendere più fluide le animazioni riducendo i tempi di calcolo: il metodo paintBoard() infatti viene richiamato solamente quando le dimensioni della scacchiera hanno subito un cambiamento.
- paintPath(): disegna il percorso seguito dal cavallo. Il percorso è rappresentato da una linea lungo le caselle della scacchiera, disegnata tramite un oggetto Path2D.Double che viene aggiornato periodicamente quando il percorso da visualizzare cambia richiamando il metodo updatePathLine().
- paintKnight(): disegna l'immagine che rappresenta la posizione del cavallo sulla scacchiera, nella posizione specificata dalle variabili private d'istanza knightX e knightY.
- paintHighlight(): disegna l'highlight che mostra all'utente le caselle raggiungibili dal cavallo a partire dalla sua posizione corrente. Per completare questo compito il metodo sfrutta la classe HighlightBuilder che con il suo metodo buildHighlight() crea un oggetto Area a partire dalle direzioni ammissibili per il cavallo. Questa area viene poi traslata da paintHighlight() sopra alla casella attualmente occupata dal cavallo e riempita con un colore semitrasparente. Il metodo setHighlightPainted() permette di scegliere se l'highlight va disegnato o no (in modalità automatica ad esempio l'highlight non è necessario).

Tutti i colori utilizzati nei metodi di disegno di BoardPanel sono parametrizzati e possono essere cambiati intervenendo sul file di configurazione jknighttour.ini.

Le animazioni vengono controllate dai metodi animatePath(), animateLastStep(), pauseAnimation(), resumeAnimation() ed endAnimation().

- animatePath(): riceve in ingresso un Vector<KPosition> che rappresenta un percorso per il cavallo e avvia l'animazione del movimento del cavallo lungo tutto questo percorso. Viene utilizzato per mostrare le soluzioni calcolate in modalità automatica o quando si carica una partita salvata.
- animateLastStep(): esegue l'animazione dell'ultima mossa del cavallo.

- pauseAnimation(): mette in pausa l'animazione.
- resumaAnimation(): fa ripartire un'animazione in pausa.
- endAnimation(): porta l'animazione alla fine.

Tutti questi metodi sfruttano un oggetto Timer, memorizzato in una variabile d'istanza, al quale viene passata un'istanza di StepAnimator: una sottoclasse privata di BoardPanel che implementa l'interfaccia ActionListener. StepAnimator prende nel costruttore una posizione iniziale, una posizione finale ed un numero di passi e, con il suo metodo actionPerformed() (richiamato dal timer), si occupa di spostare il cavallo, modificando le variabili knightX e knightY, dalla posizione iniziale a quella finale nel numero di passi specificato. Ad ogni passo knightX e knightY vengono incrementate di una quantità ottenuta con una interpolazione sinusoidale (metodo interpolate() di Utilities). Una volta completato il suo compito StepAnimator richiama il metodo animationStepEnd() per informare BoardPanel. A seconda che l'animazione corrente sia quella di un percorso intero o dell'ultimo passo animationStepEnd() avvia l'animazione della mossa successiva nel percorso o termina l'animazione.

Tornando alla descrizione di KnightFrame si osservi che ManualModePanel e AutoModePanel vengono visualizzati alternativamente a seconda della modalità di funzionamento corrente sfruttando un JPanel impostato con un CardLayout. Il passaggio da una modalità di funzionamento all'altra viene gestito tramite la pressione di due bottoni situati lungo il lato inferiore dell'interfaccia.

La gestione degli eventi generati nei vari componenti dell'interfaccia avviene per passi successivi: ciascuno dei pannelli di cui si è parlato sopra implementa l'interfaccia Listener appropriata per ascoltare gli eventi generati dai componenti che esso contiene. Questi eventi vengono poi reindirizzati al metodo manageActionEvent() di KnightFrame il quale li gestisce richiamando il metodo appropriato del GameMode corrente (GameMode verrà approfondito nella sezione riguardante la View). Unica accezione a questo percorso per gli eventi è l'evento di click sulla scacchiera che viene gestito direttamente da BoardPanel richiamando il metodo manageClickOnBoard() del GameMode corrente.

Nelle due figure successive è rappresentata l'interfaccia di JKnightTour rispettivamente in modalità manuale ed in modalità automatica.

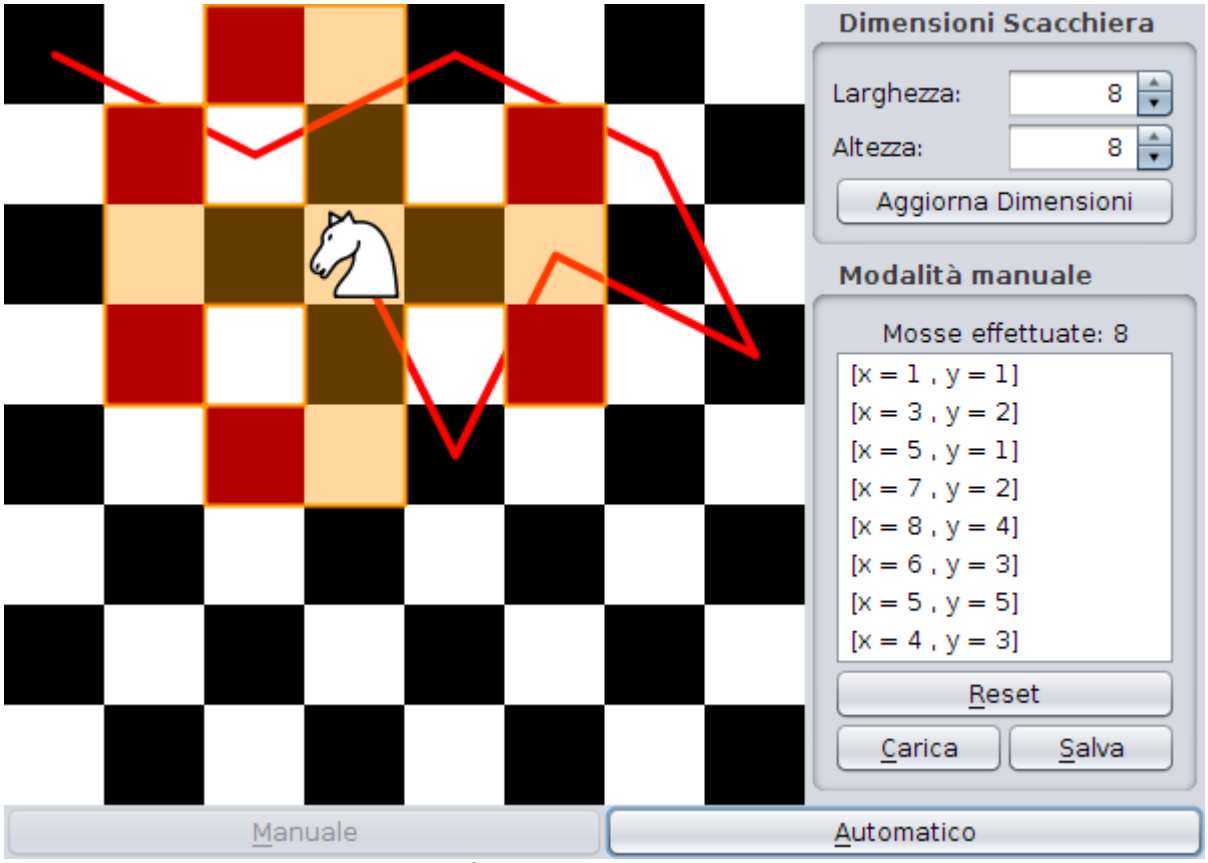

Figura 5: JKnightTour in modalità manuale

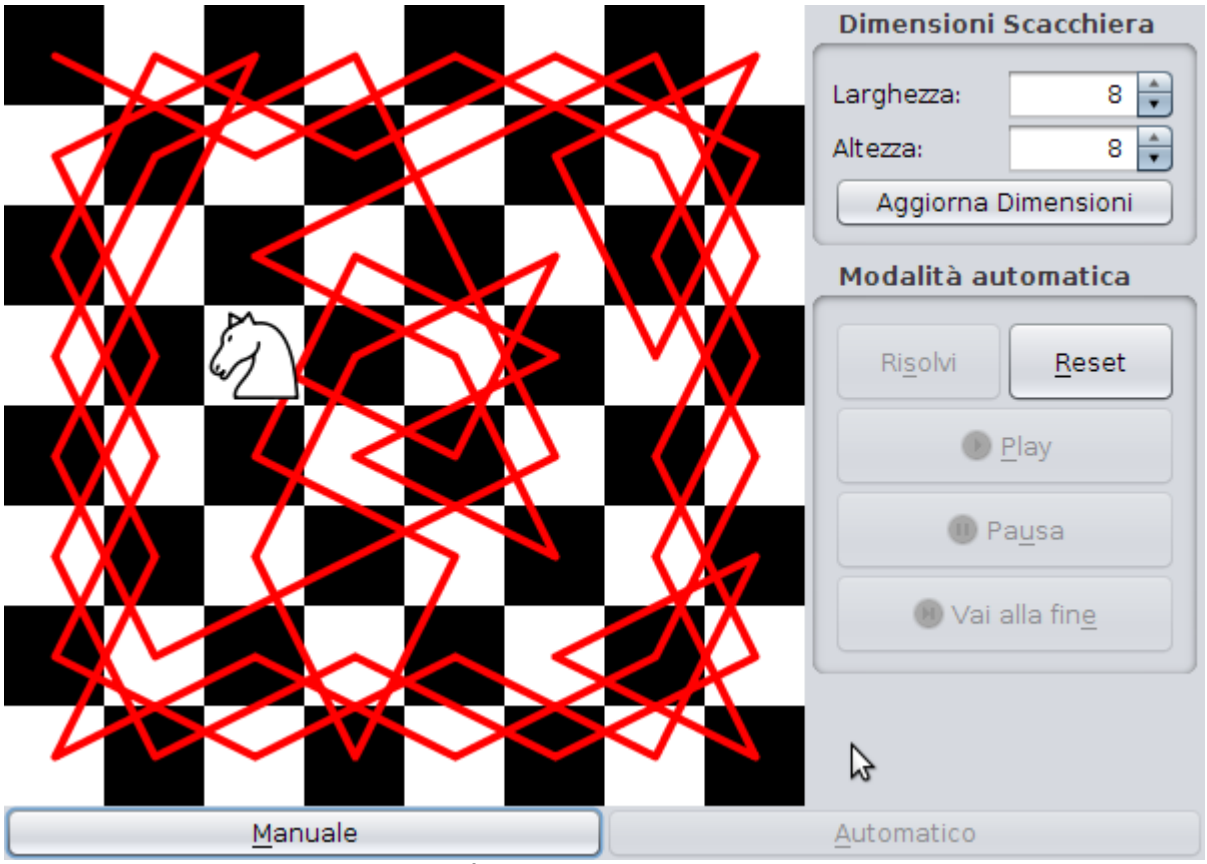

Figura 6: JKnightTour in modalità automatica

Controller (jknighttour.controller)

Le classi appartenenti al blocco Controller di JKnightTour si trovano nel package jknighttour.controller. La loro struttura è rappresentata nel diagramma UML in figura 7.

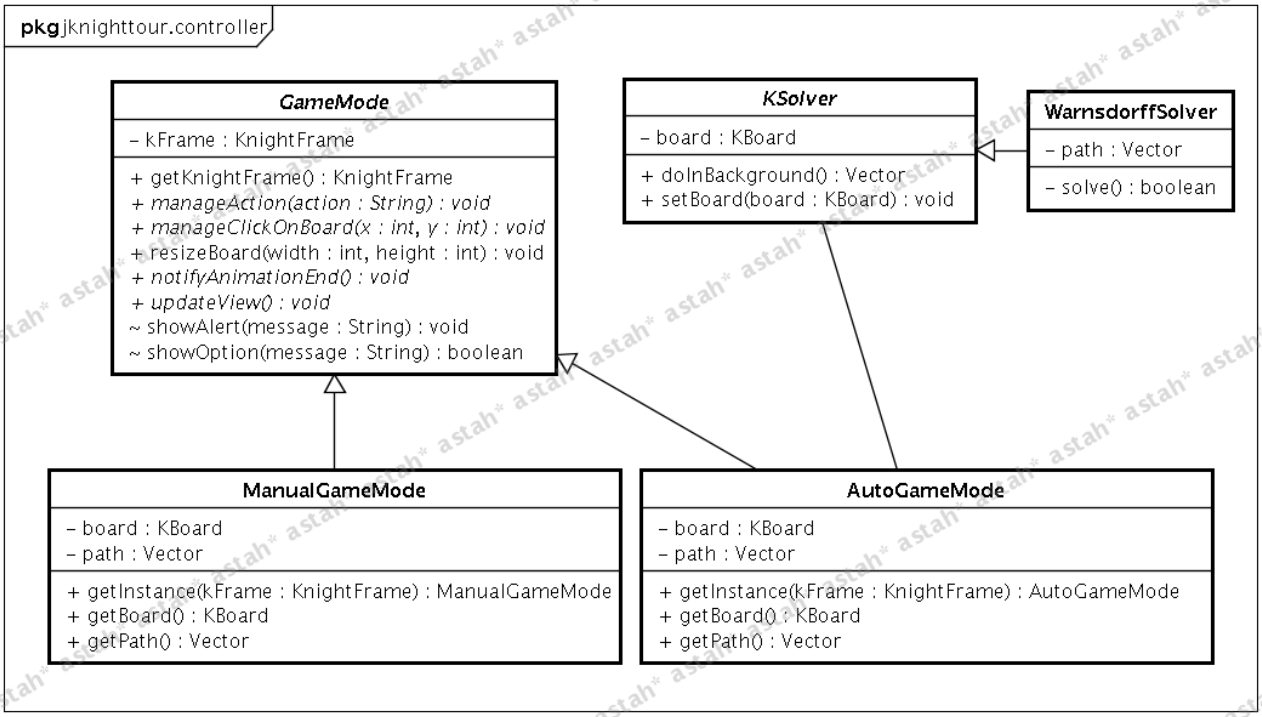

Figura 7: Diagramma UML delle classi di jknighttour.controller

GameMode è una classe astratta che definisce una serie di metodi utilizzati per gestire gli eventi generati dall'interfaccia di JKnightTour. Questa classe viene estesa da ManualGameMode e AutoGameMode. Entrambe queste classi utilizzano il pattern Singleton e ciascuna si occupa di gestire gli eventi rispettivamente in modalità manuale ed automatica per implementare la logica dell'applicazione. KnightFrame conserva un riferimento al GameMode corrente che può appunto essere l'istanza singleton di ManualGameMode o di AutoGameMode.

Viene ora esposta la funzione dei vari metodi astratti di GameMode:

- manageAction(): viene richiamato da KnightFrame passando come argomento il nome dell'azione eseguita dall'utente (il click su di un bottone o su di una mossa nella lista delle mosse di ManualModePanel). A seconda dell'azione specifica manageAction si occuperà di eseguire tutte le operazioni necessarie (far partire un'animazione, caricare un salvataggio ecc.)
- manageClickOnBoard(): viene richiamato da BoardPanel passando come argomento le coordinate della casella della scacchiera cliccata dall'utente e gestisce di conseguenza l'evento. In modalità automatica seleziona la casella di partenza, in modalità manuale seleziona la mossa successiva.
- resizeBoard(): viene chiamato da BoardPanel per eseguire l'aggiornamento delle dimensioni della scacchiera. L'aggiornamento delle

dimensioni della scacchiera comporta sempre la perdita di tutte le mosse eseguite fino a quel momento, per questo resizeBoard() mostra un pop-up di conferma.

- notifyAnimationEnd(): viene richiamato da BoardPanel quando le animazioni terminano per informare il GameMode corrente che può riattivare i componenti disabilitati durante l'animazione.
- updateView(): è un metodo che viene richiamato da KnightFrame a seguito di un cambio di modalità di esecuzione per richiedere un aggiornamento dell'interfaccia.

Di seguito viene riportata una rapida descrizione di alcuni aspetti importanti delle due modalità di funzionamento dell'applicazione.

In **Modalità Manuale** l'utente ha la possibilità di salvare a caricare lo stato corrente della scacchiera. Salvataggio e caricamento avvengono grazie a due metodi di classe di Utilities che si occupano di codificare e decodificare lo stato corrente della scacchiera in maniera appropriata. In particolare si è adottato questo formato:

- Il primo byte del file scritto o letto rappresenta le dimensioni della scacchiera: nei quattro bit alti viene scritta la larghezza della scacchiera, nei quattro bit bassi la sua altezza.
- Gli altri byte del file rappresentano in ordine le caselle del percorso seguito dal cavallo: per ogni byte i quattro bit alti memorizzano la coordinata x della casella corrispondente e i quattro bit bassi la coordinata y.
- I dati possono essere memorizzati in questo modo senza possibilità di overflow dato che si sono fissate dimensioni massime per la scacchiera di 12x12 caselle.

Sempre in modalità manuale, cliccando su di una mossa tra quelle mostrate nella lista delle mosse nel ManualModePanel, l'utente vedrà l'immagine del cavallo spostarsi sulla casella corrispondente della scacchiera e un highlight comparire mostrando le mosse alternative che si sarebbero potute compiere in quel passo del percorso. Questo però non corrisponde ad annullare delle mosse: fintanto che l'utente non selezionerà una mossa alternativa a quella precedentemente eseguita lo stato della partita rimarrà inalterato.

In **Modalità Automatica** il programma permette di risolvere automaticamente il problema. In particolare la ricerca di una soluzione viene affidata ad un KSolver. KSolver è una classe astratta che eredita da SwingWorker e si occupa di fornire un'interfaccia comune per diversi possibili algoritmi di risoluzione del problema. Nell'applicazione è fornita un'implementazione dell'algoritmo di Warnsdorff che verrà descritta in seguito. Quando l'utente fa click sul pulsante “Risolvi” di AutoModePanel l'evento generato raggiunge AutoGameMode, il quale crea un'istanza di AutoDialog. AutoDialog è un JDialog che blocca la finestra principale dell'applicazione e mostra un messaggio che chiede all'utente di attendere il calcolo della soluzione ed un bottone che permette di interrompere la computazione. AutoDialog a sua volta carica dal file di impostazioni jknighttour.ini il nome della classe che eredita da KSolver da utilizzare per risolvere il problema (oppure in caso di assenza dell'impostazione corrispondente o altro errore come default utilizza WarnsdorffSolver).

L'algoritmo risolutivo quindi viene eseguito su un thread separato da quello dell'interfaccia (grazie a SwingWorker). Una volta che i calcoli sono completi si avvierà un'animazione che mostra all'utente il percorso calcolato.

L'algoritmo di Warnsdorff implementato è piuttosto semplice: si tratta in sostanza di una ricerca tramite back-tracking della soluzione che sfrutta un'euristica. In passi:

1. Marca come visitata la posizione corrente
2. Se ci si trova in vicolo cieco segna come libera la posizione corrente e torna indietro, se si è raggiunta la fine del percorso termina.
3. Esplora in ordine tutte le direzioni possibili a partire dalla posizione corrente (torna al passo1).

L'euristica utilizzata riguarda l'ordine con il quale esplorare le varie direzioni possibili: in particolare l'algoritmo di Warnsdorff prevede di esplorare per prime le destinazioni con il minor numero di “uscite”, o per dirla con il linguaggio dei grafi, i nodi di grado minore. Questo accorgimento permette di avere complessità media lineare rispetto al numero di caselle che compongono la scacchiera e quindi di trovare soluzioni in tempi ragionevoli.

Problemi riscontrati

JKnightTour è dal punto di vista logico piuttosto semplice: sia la modellazione dei dati trattati che la gestione della logica applicativa non hanno comportato nessun problema di implementazione.

L'aspetto che ha richiesto più tempo e più sforzi sicuramente è stato quello della gestione dell'animazione. Il modello MVC scelto fin dall'inizio per la realizzazione dell'applicazione prevede una forte separazione tra il modello dei dati, la componente che si occupa della sua visualizzazione e la logica di controllo sottostante. Questa scelta ha complicato sicuramente il codice di BoardPanel, dato che questo componente si deve occupare di realizzare le animazioni in maniera indipendente dalla logica di controllo: si è resa necessaria l'implementazione di una logica di comunicazione tra i componenti simile a quella ad eventi. Le classi del Controller chiedono (tramite appositi metodi) al BoardPanel di iniziare le animazioni e il BoardPanel a sua volta notifica le classi che si occupano del controllo quando l'animazione è giunta a termine.

Altro aspetto cruciale nella gestione delle animazioni è stato quello della fluidità delle stesse: in un primo momento il ridisegno del componente BoardPanel era stato implementato in modo da ridisegnare e ricalcolare tutte le parti della grafica ad ogni frame. Ad ogni repaint quindi BoardPanel disegnava tanti quadrati, colorati in maniera alternata, quante sono le caselle della scacchiera e costruiva un nuovo oggetto Path2D.Double per disegnare il percorso del cavallo, iterando sulla lista delle mosse effettuate. Ci si è presto accorti che un approccio del genere era molto inefficiente. Per questo motivo si è deciso di implementare due “buffer” per accelerare le operazioni di ridisegno: un'immagine per lo sfondo della scacchiera, ridisegnata solamente quando le dimensioni della stessa variano; un oggetto Path2D.Double da ricalcolare solamente al variare del percorso del cavallo visualizzato (l'ultima mossa viene visualizzata sul percorso solamente quando l'animazione corrispondente è terminata).

4 Possibilità di estensione e personalizzazione

JKnightTour è stato progettato tenendo in mente la possibilità di personalizzarne in parte l'aspetto. Il file di configurazione jknighttour.ini contiene una serie di coppie chiave=valore che corrispondono alle impostazioni dei vari colori utilizzati dall'applicazione per il disegno della scacchiera. I colori sono espressi utilizzando la stesso formato esadecimale tipico ad esempio del html (ad eccezione di “highlightColor1” che dovendo supportare anche la trasparenza prevede l'utilizzo di 8 caratteri esadecimali invece che i soliti 6). Nel caso l'applicazione non dovesse riuscire a caricare il file jknighttour.ini o questo non contenesse i valori cercati vengono utilizzati i valori di default.

E possibile anche implementare un algoritmo alternativo per il calcolo automatico delle soluzioni: in questo caso basta creare una classe che estende KSolver e specificarne il nome nel file di configurazione in corrispondenza della chiave “solverClass”. Ad esempio potrebbe essere interessante implementare una versione alternativa dell'algoritmo di Warnsdorff che utilizzi strutture dati specifiche al posto di quelle standard interne all'applicazione per rappresentare le posizioni della scacchiera. In particolare una discreta ottimizzazione dell'algoritmo si otterrebbe utilizzando una lista ordinata per conservare l'elenco dei vicini di ogni cella, evitando così di doverla riordinare ad ogni iterazione dell'algoritmo come avviene nell'implementazione attuale.

5 Bibliografia

- Wikipedia, “Knight's Tour”, http://en.wikipedia.org/wiki/Knight_tour
- Wikipedia, “Hamiltonian Path Problem”, http://en.wikipedia.org/wiki/Hamiltonian_path_problem