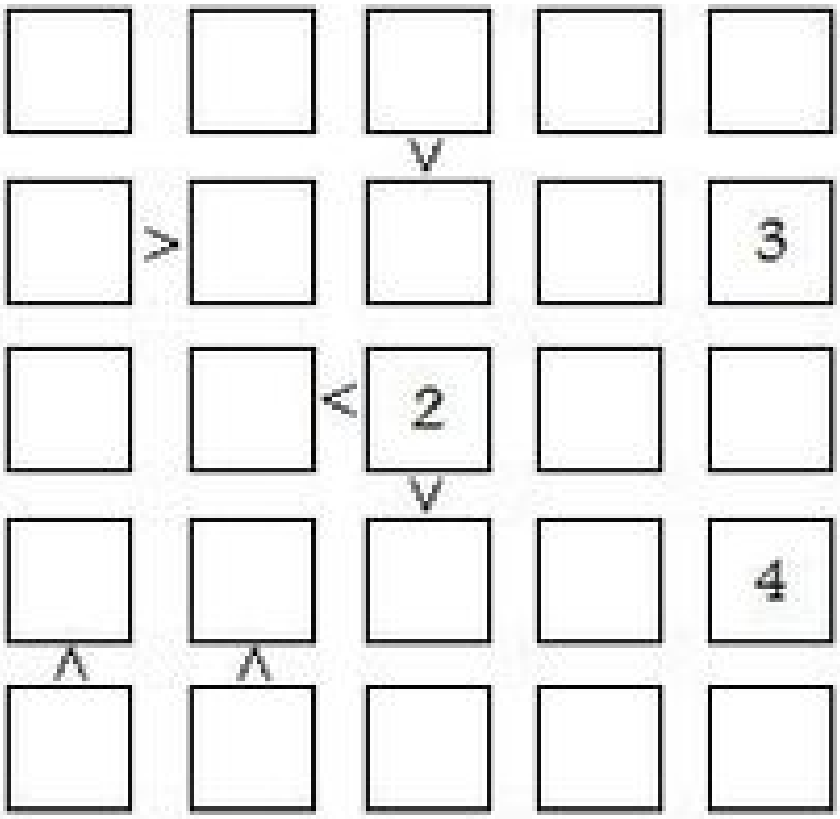
Progetto Ingegneria del Software

Anno Accademico 2016/2017

"Futoshiki"



Morelli Lorenzo
169153

Analisi dei requisiti

E’ stato richiesto lo sviluppo di un’applicazione che permettesse la risoluzione di un Futoshiki (puzzle game simile al famoso sudoku).

Descrizione del gioco

Il Futoshiki si pratica su una griglia quadrata composta solitamente da 25 caselle (5x5). Tuttavia la dimensione della griglia varia con la difficoltà , si parte da un minimo di 3x3 per arrivare ad un massimo di 9x9. L'obiettivo del gioco consiste nel completare la griglia con tutti i numeri da 1 alla dimensione scelta andando anche a rispettare le seguenti condizioni:

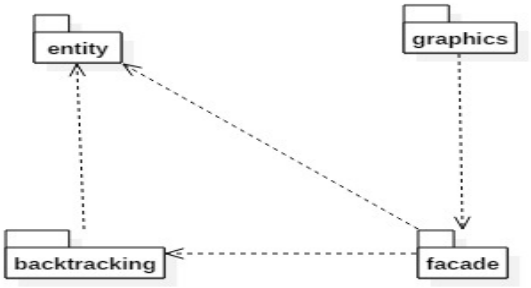
1) Lungo ogni riga e lungo ogni colonna non vi possono essere ripetizioni dello stesso numero
2) Qualora tra due caselle fosse indicato un simbolo di disuguaglianza anch'esso deve essere rispettato nell'inserimento della cifra esatta.

di seguito viene riportato il caso d'uso principale:

ID1	Risoluzione Futoshiki
Descrizione:	Permette all’utente di inserire e risolvere un Futoshiki
Attore:	Utente
Precondizioni	L’utente ha avviato il programma e conosce il Futoshiki da immettere
Svolgimento	1. L’utente inserisce il valore delle celle nello schema. 2. L’utente definisce il numero di soluzioni massimo che vuole cercare. 3. L’utente avvia il calcolo delle soluzioni. 4a. L’utente visualizza le soluzioni del Futoshiki. 4b. Il Futoshiki inserito non rispetta le regole.
Postcondizioni in caso di successo(a)	L’utente può scorrere le soluzioni del Futoshiki o inserirne un nuovo.
Postcondizioni in caso di fallimento(b)	L’utente è invitato a correggere gli errori o a fornire un nuovo schema.

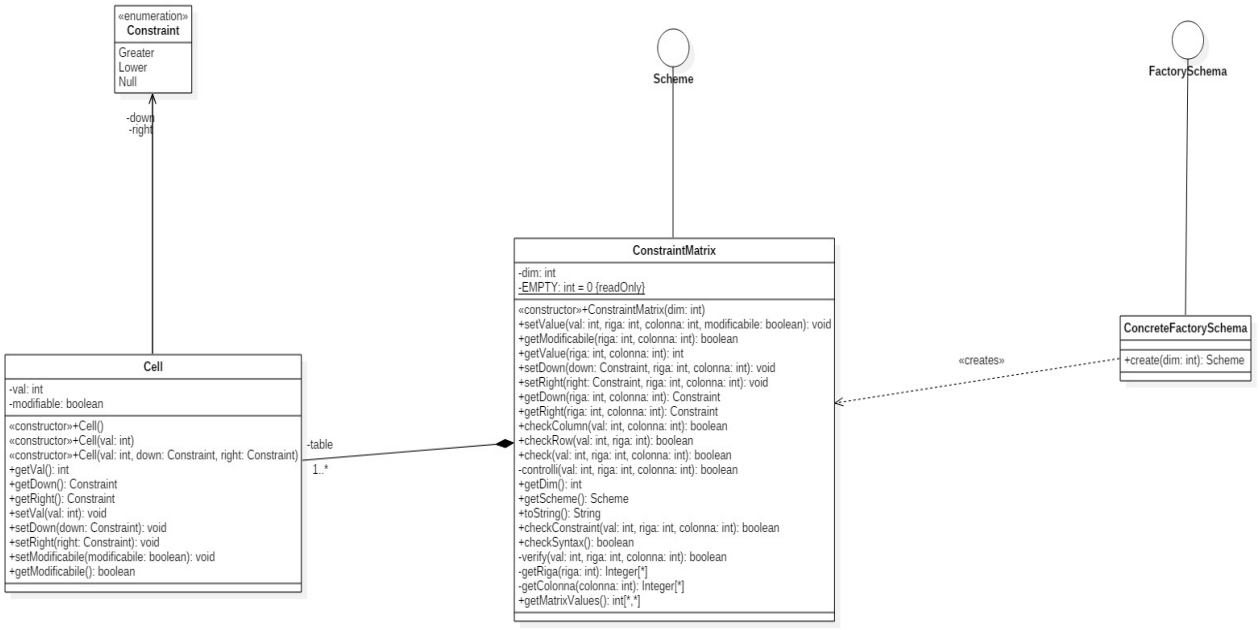
Struttura del progetto

il progetto è strutturato in maniera tale da raggruppare le classi in package per dominio applicativo seguendo i prinicipi dell'ingegneria del software quali il common closure e common Reuse così da garantire un'alta coesione interna e un basso accoppiamento tra i package.



Package entity

Si può modellare lo schema del Futoshiki mediante un'interfaccia, *Scheme,* in modo che esso possa essere rappresentato con strutture dati differenti compatibili con l'interfaccia. Per far sì che la creazione dello schema non sia legata alla specifica struttura dati scelta verrà utilizzato il pattern *Factory Method*.



Come è possibile osservare dal diagramma delle classi relativo a tale package la creazione di un oggetto conforme all'interfaccia *Scheme* è affidata ad un oggetto conforme all'interfaccia

FactoryScheme, la quale infatti fornisce il metodo *create()* per la creazione dello schema. Sarà dunque possibile cambiare il tipo specifico di *FactoryScheme* per cambiare il tipo di implementazione di *Scheme* da utilizzare all'interno del progetto andando a creare una gerarchia di classi parallele.

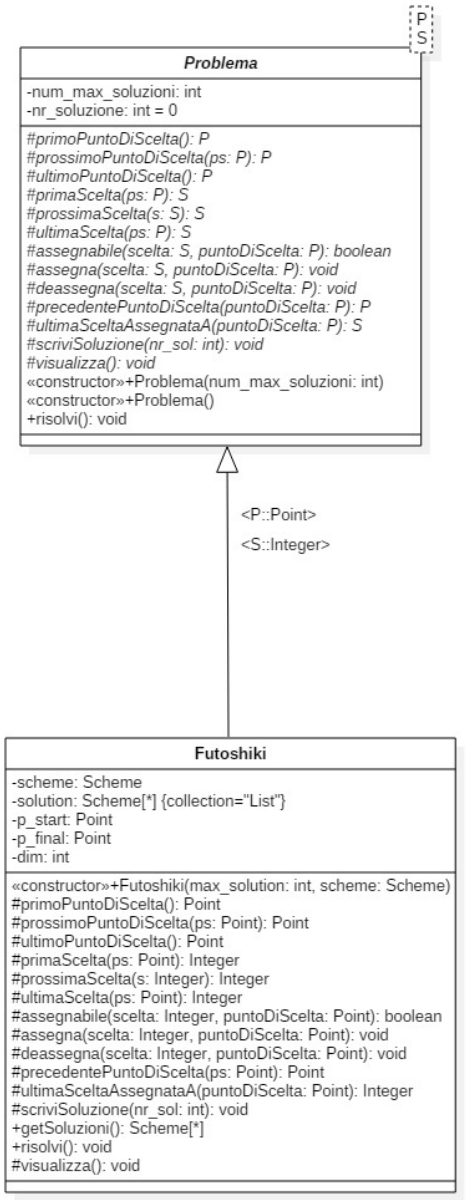
Per lo sviluppo corrente è stata realizzata una classe *ConstraintMatrix* che si basa su una matrice di *Cell,* celle che, oltre a conservare un valore intero che rappresenta il valore in tale cella, possiedono due oggetti *Constraint* per i vincoli di segno ed una variabile booleana *modifiable* che specifica la modificabilità della cella (i valori inseriti dall'utente tramite la GUI sono statici mentre i valori inseriti dal programma sono sovrascrivibili). In particolare ogni oggetto *Cell* mantiene all'interno di sé i vincoli di segno alla sua destra e in basso. Questi ultimi possono assumere valori : *Maggiore, Minore, Nulla*.

Nella classe *ConstraintMatrix* vengono concretizzati i metodi di *Scheme*. Tali metodi permettono di accedere allo schema del Futoshiki per modificare, inserire e verificare valori e vincoli al fine di risolvere il gioco.

Package backtracking

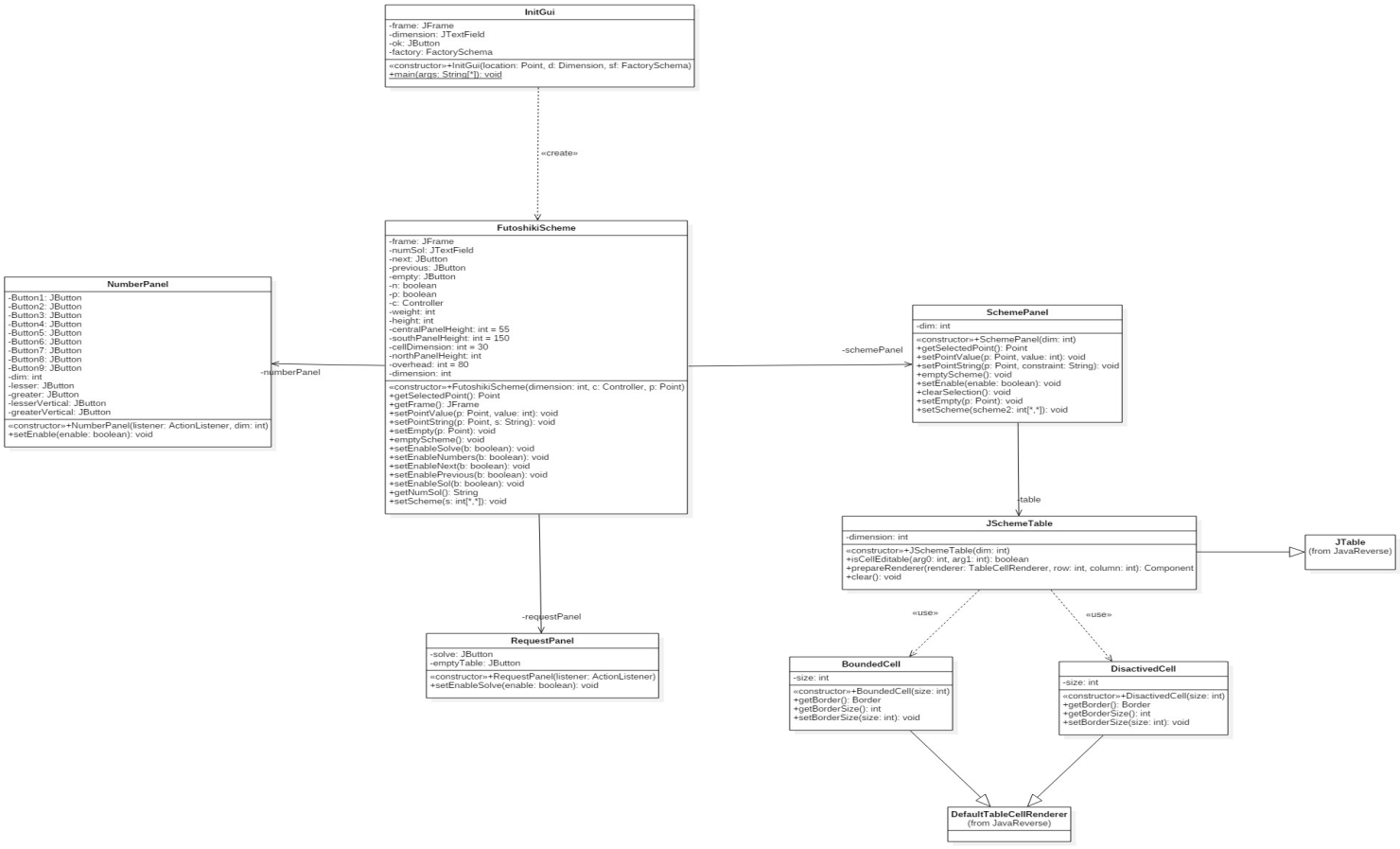
Per risolvere il Futoshiki è possibile utilizzare la tecnica del backtracking, che realizza la ricerca esaustiva di tutte le soluzioni del problema e scarta quelle che non soddisfano i vincoli.
Si può quindi utilizzare il pattern *TemplateMethod* attraverso la classe *Futoshiki* che concretizza la classe *Problema<P,S>*.La Classe Problema incapsula l'algoritmo di gestione di una categoria di problemi simili al Futoshiki. Attraverso l'utilizzo di tale pattern si è così solamente dovuto riscrivere i metodi strettamente inerenti

al problema specifico, ad esempio la verifica dell'assegnabilità di un valore o i punti di scelta da usare per la risoluzione. Nello specifico problema le scelte possibili sono oggetti Integer che saranno assegnati alle celle della matrice la cui posizione è identificata da un oggetto Point.



Package view

Sono presenti tutti i moduli che formano la grafica del progetto in modo da far interagire, in maniera controllata , l'utente con le strutture sottostanti .



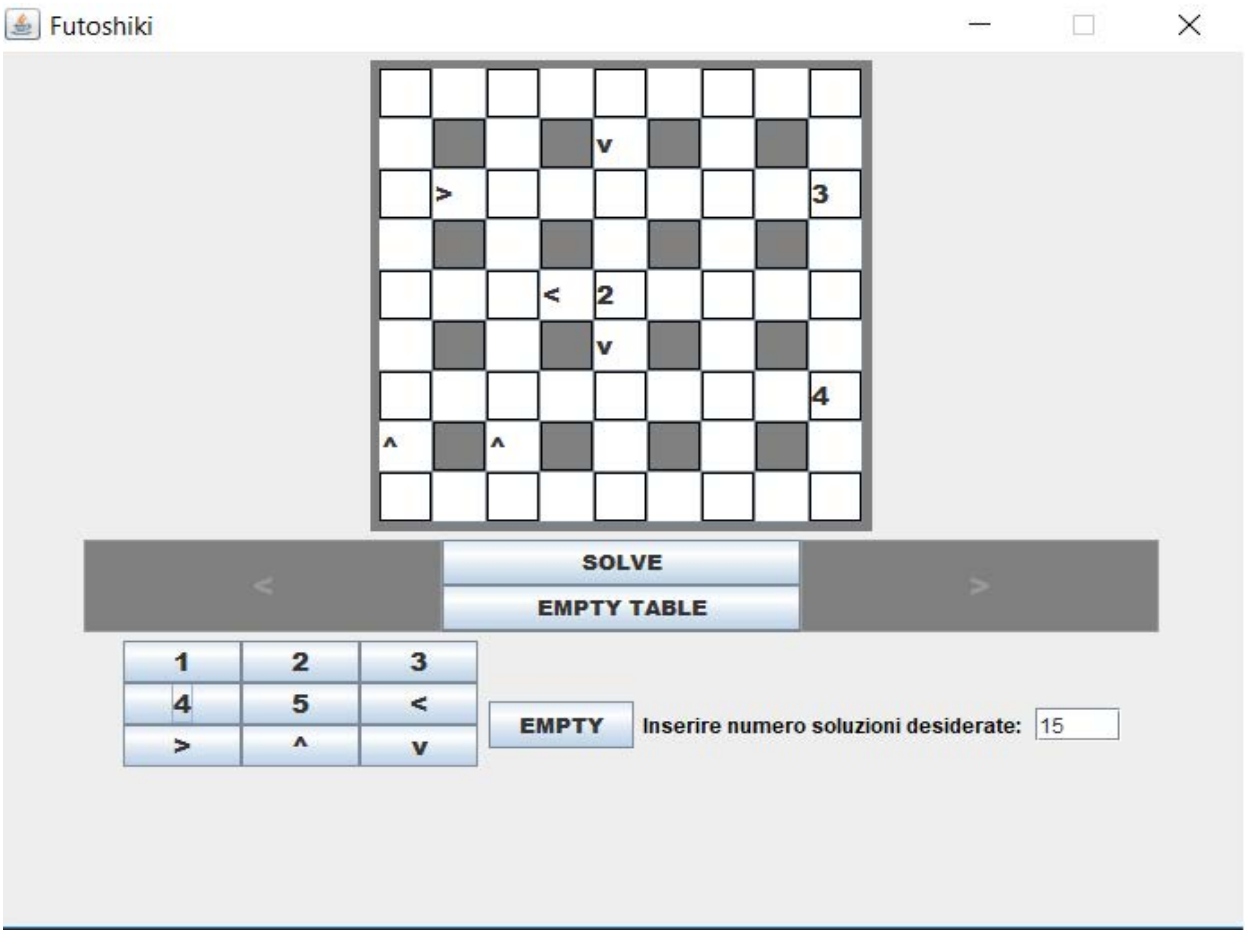
La classe *InitGui* fornisce una finestra per permettere al cliente di scegliere la dimensione dello schema di gioco e verifica che essa sia compresa tra 3 e 9.

In seguito per rappresentare il Futoshiki in forma tabellare, è stato scelto di usare la classe integrata nella libreria Java JTable.Per rappresentare al meglio il layout delle celle del futoshiki si è deciso di estendere la classe DefaultTableCellRenderer per creare degli oggetti conformi all'interfaccia TableCellRenderer che la tabella utilizza per rappresentare le celle, ed inserire appositamente dei Renderer in grado di rappresentare celle con i bordi.
Sia *BoundedCell* che *DisactivedCell* riscrivono il metodo *getBorder* per restituire una cella con un contorno su tutti i lati. Inoltre *DisactivedCell* colora di grigio la cella per segnalare visivamente che non è possibile inserire valori o vincoli su di essa.

E' stata quindi sviluppata la classe *JSchemeTable* che estende la classe *JTable* per specializzarne il comportamento. In particolare modo fornisce la possibilità di rappresentare una tabella di tipo *Scheme* e di rappresentare il contenuto, inserito dall'utente. Inoltre permette all'utente soltanto la selezione delle celle impedendogli di poterci scrivere all'interno ridefinendo il metodo *isCellEditable()* in modo da ritornare a priori false.

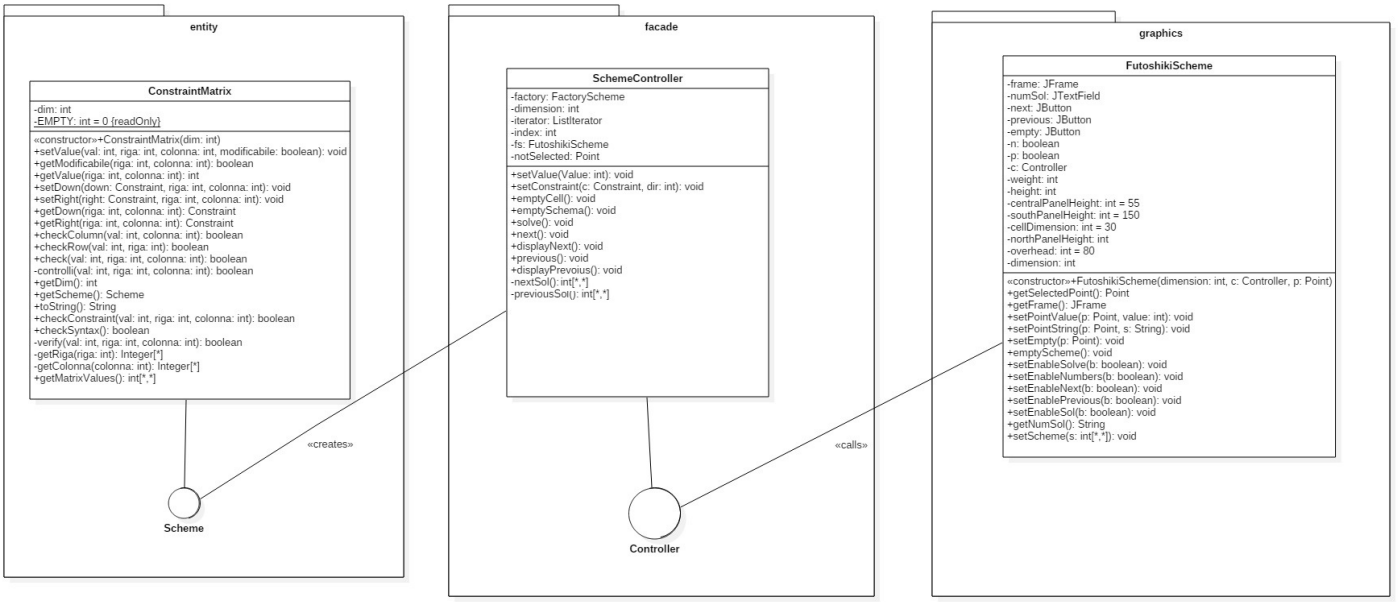
Questa tabella sarà poi aggiunta ad una classe che estende *JPanel*, ossia *SchemePanel,* la quale fornisce metodi per modificare i valori delle celle che saranno visualizzate.

Questo pannello sarà aggiunto al *JFrame* principale contenuto nella classe *FutoshikiScheme*. Tale classe modella la finestra principale di gioco che non solo contiene lo schema *SchemePanel*, ma anche altri pannelli per interagire con la GUI. In particolare modo ha un pannello *NumberPanel* che contiene i bottoni dei numeri e dei segni che andranno a popolare lo schema(che saranno assegnati in base alla dimensione), un pannello *RequestPanel* che contiene i bottoni solve e empty che consentono rispettivamente di risolvere il Futoshiki e di svuotare la griglia. Inoltre vi è una JTextField per inserire il numero di soluzioni desiderate, un tasto empty per svuotare il contenuto di una cella, e i tasti next e previous per navigare le soluzioni trovate una volta risolto il Futoshiki.
Di seguito è mostrato come appare all'utente la GUI:

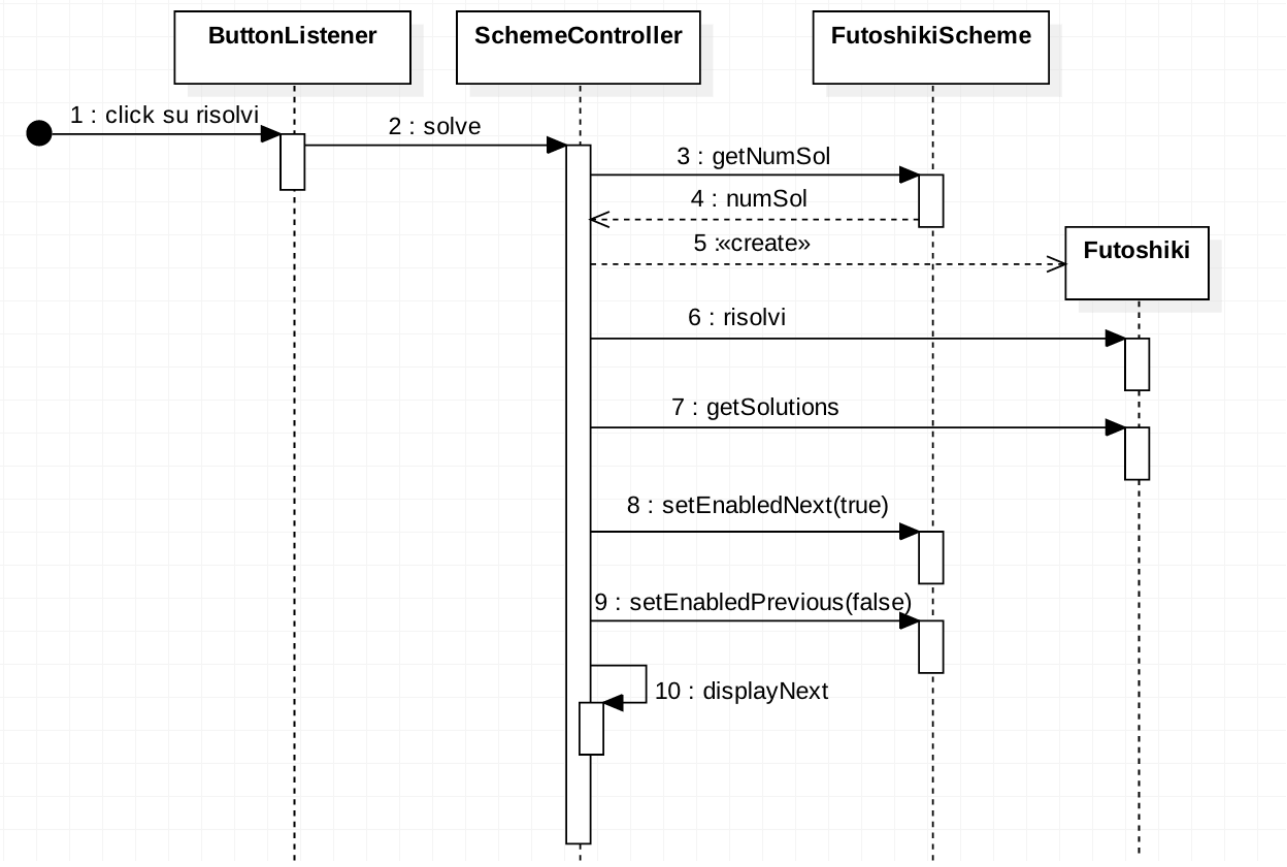


Package façade

Le classe contenute in tale package servono per disaccoppiare la parte grafica dalla relativa parte logica fornendo un'interfaccia che va a nascondere al client la complessità del sistema. In particolare modo l'oggetto controller riceverà i comandi dall'*ActionListener,* contenuto all'interno della finestra principale, e sarà lui ad effettuare le eventuali modifiche sullo *Scheme* contenuto al suo interno. Inoltre il controller avviserà di tali cambiamenti la parte grafica grazie ad un riferimento che ha ad essa.

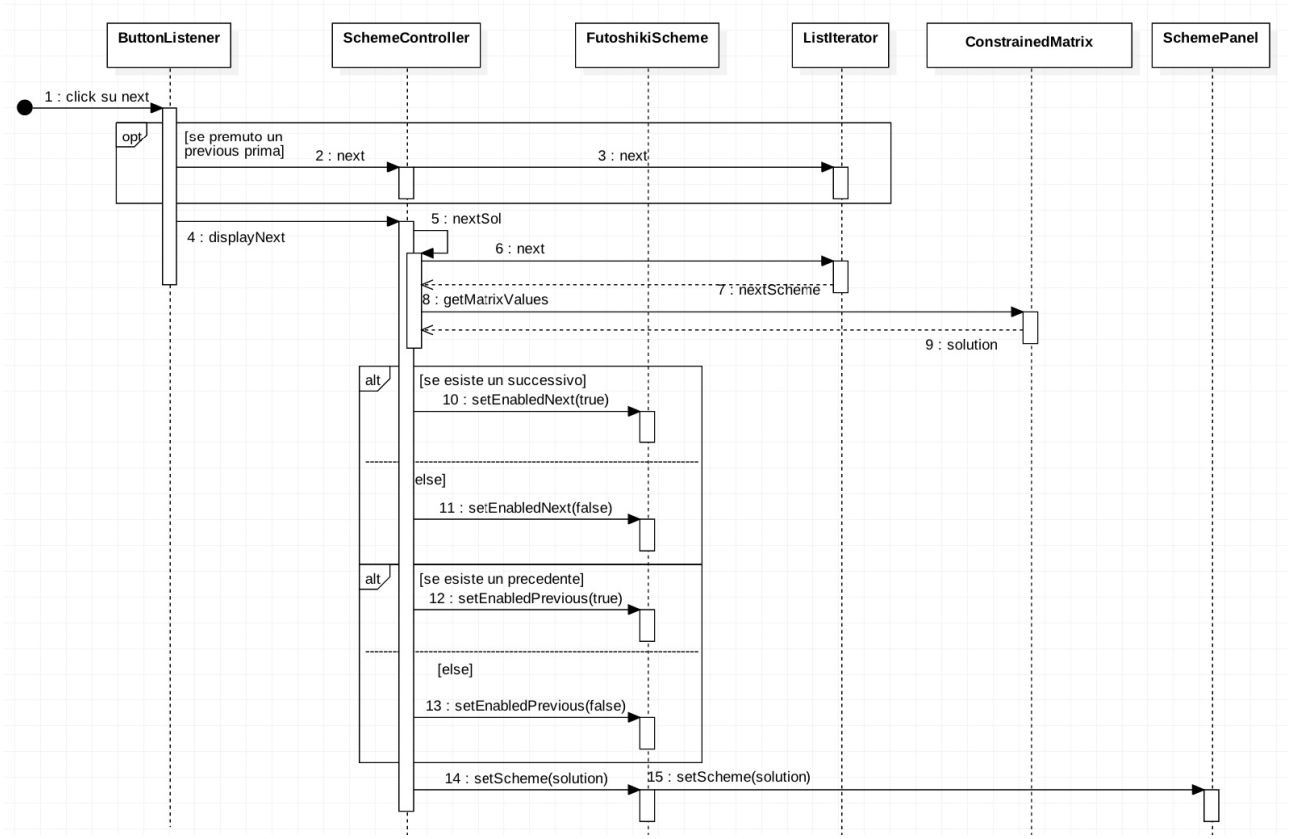


Lo *SchemeController* è responsabile della risoluzione creando un oggetto Futoshiki come mostra il seguente *Sequence Diagram*



Una volta risolto il Futoshiki l'utente può visualizzare le soluzioni mediante i bottoni previous e next. Per permettere la loro visualizzazione occorre poter iterare su di esse e per tale motivo le sono inserite , sotto forma di matrici di interi , in una *LinkedList* da cui è possibile ricavare un *ListIterator.*

Il seguente Sequence Diagram illustra come è possibile, dopo aver premuto il tasto next, ottenere la prossima soluzione con l'iteratore e come visualizzarla sulla finestra tramite il controller



Test

Sono stati effettuati alcuni test di tipo black box sui moduli *ConstraintMatrix* e *Futoshiki.*

La prima sequenza di test è definita nella classe *FutoshikiTest* che definisce il metodo *@BeforeClass buildScheme()* che costruisce il Futoshiki preso come esempio dalla traccia del progetto andando a rilassare due vincoli così da ottenere più soluzioni. L'insieme dei possibili input, quindi delle tabelle di Futoshiki possibili, è stato partizionato in due sottoinsiemi: il primo contiene i Futoshiki risolvibili, il secondo i Futoshiki irrisolvibili.
Sono eseguiti sequenzialmente tre metodi di test. Il primo, *@Test correctScheme(),* risolve il Futoshiki precedentemente costruito e verifica tramite il metodo *checkSyntax()* che siano rispettati da tutte le soluzioni i vincoli del Futoshiki.

Il secondo, *@Test exampleScheme*, inserisce i vincoli mancanti dell'esempio riportato sul progetto e risolve il Futoshiki andando a valutare l'unicità della soluzione . Mentre il metodo *@Test uncorrectScheme()* come prima operazione inserisce un valore già contenuto nella seconda riga e nel risolvere il Futoshiki verifica che l'insieme delle soluzioni sia vuoto.Si e basato il tutto sull'utilizzo della classe Assert e del metodo assertTrue(boolean). La classe *ConstraintMatrixTest* definisce invece il modulo di test per la classe *ConstraintMatrix*. L'insieme dei possibili input dei metodi di costruzione della tabella è stato partizionato nei seguenti sottonsiemi:

- **valore valido in una cella valida**
- **vincolo valido in una cella non valida**
- **valore non valido in una cella valida**

- @Test(expected = IllegalArgumentException.class) NegativeIndex() che inserisce un vincolo in una cella con indice negativo
- @Test(expected = IllegalArgumentException.class) UncorrectIndex() che inserisce un valore in una cella con indice superiore alla dimensione dello schema
- @Test(expected = IllegalArgumentException.class) NegativeValue() che inserisce una valore negativo in una cella corretta
- @Test(expected = IllegalArgumentException.class) UncorrectValue() che inserisce un valore superiore alla dimensione dello schema
- @Test correctInsert() che verifica il corretto inserimento di un valore e della proprietà modificable.