**“Ingegneria del Software”**

**2020-202****1**

**Docente: Prof. Angelo Furfaro**

**KenKen**

|  |  |
| --- | --- |
| **Data** | <12/07/2021> |
| **Documento** | Documento Finale – D3 |

|  |  |  |
| --- | --- | --- |
| **Team Members** | | |
| **Nome e Cognome** | **Matricola** | **E-mail address** |
| Giulio Talarico | 200881 | giuliotal@gmail.com |

Sommario

[List of Challenging/Risky Requirements or Tasks 3](#_Toc76973110)

[A. Stato dell’Arte 4](#_Toc76973111)

[B. Raffinamento dei Requisiti 5](#_Toc76973112)

[***B.1 Servizi (con prioritizzazione)*** 5](#_Toc76973113)

[***B.2 Requisiti non funzionali*** 6](#_Toc76973114)

[***B.2 Scenari d’uso dettagliati*** 7](#_Toc76973115)

[***B.3 Assunzioni*** 9](#_Toc76973116)

[***B.4 Use Case Diagrams*** 9](#_Toc76973117)

[C. Architettura Software 10](#_Toc76973118)

[***C.1 The static view of the system: Component Diagram*** 10](#_Toc76973119)

[***C.2 The dynamic view of the software architecture: Sequence Diagram*** 10](#_Toc76973120)

[D. Scelte Progettuali (Design Decisions) 14](#_Toc76973121)

[E. Progettazione di Basso Livello 16](#_Toc76973122)

[G. Testing 19](#_Toc76973123)

[Appendix. Prototype  22](#_Toc76973124)

# List of Challenging/Risky Requirements or Tasks

|  |  |  |  |
| --- | --- | --- | --- |
| **Challenging Task** | **Date the task is identified** | **Date the challenge is resolved** | **Explanation on how the challenge has been managed** |
| Controllo del soddisfacimento del vincolo aritmetico all’interno dei blocchi dello schema di gioco | 26-06-2021 | 28-06-2021 | La griglia di gioco tiene traccia dei blocchi che ne costituiscono lo schema: per verificare che il vincolo è soddisfatto si effettuano tutte le possibili permutazioni dei numeri inseriti all’interno di ogni blocco. |
| Controllo dell’adiacenza delle celle selezionate dall’utente per creare un blocco | 01-07-2021 | 02-07-2021 | A partire dall’ultima cella della griglia che è stata selezionata dall’utente, essa viene marcata come visitata e si controlla se ci sono celle adiacenti lungo i due assi principali: in caso affermativo, tali celle effettuano ricorsivamente lo stesso controllo aggiornando una variabile contatore. Se alla fine del controllo il contatore è pari al numero di celle selezionate, allora tutte le celle sono adiacenti. |
| Visualizzazione grafica del mancato soddisfacimento dei vincoli all’interno di un blocco dello schema di gioco | 03-07-2021 | 03-07-2021 | Per ogni blocco che non soddisfa il vincolo aritmetico, si individuano all’interno della griglia di gioco le celle che lo costituiscono; soltanto la cella più in alto a sinistra contiene un riferimento testuale al vincolo da rispettare: essa viene individuata ed evidenziata in rosso |
| Rendere reversibili le operazioni di creazione dei blocchi e di cancellamento di tutti i numeri della griglia di gioco | 05-07-2021 | 05-07-2021 | Con l’ausilio del design pattern Memento, nel momento in cui viene invocato un comando reversibile, viene creato un oggetto GridMemento che cattura lo stato interno della griglia di gioco, ovvero i numeri inseriti e lo schema di gioco, in modo da poterli ripristinare con degli appositi comandi undo/redo |

A. Stato dell’Arte

L’inventore del KenKen, il matematico Tetsuya Miyamoto, crea i suoi puzzles a mano dal 2004. Con la diffusione del gioco al di là dei confini del Giappone, in particolare in America, dove i principali giornali iniziavano a pubblicare i puzzles di Miyamoto, la creazione artigianale comincia ad essere insufficiente per gestire la domanda sempre crescente di giochi di KenKen, pertanto si arriva alla produzione di un software proprietario per la generazione automatica: [*Kenerator*](http://www.kenkenpuzzle.com/about/kenerator)*.*

*Kenerator* viene sviluppato con la collaborazione di David Levy, campione di scacchi ed esperto di intelligenza artificiale, sotto la stretta supervisione di Miyamoto, ed è oggi il software utilizzato per la generazione automatica di giochi ufficiali di KenKen sia sul web, sul sito <http://www.kenkenpuzzle.com/>, che su dispositivi mobili, tramite l’app KenKen Classic.

Il processo di generazione avviene secondo diversi criteri: livello di difficoltà, numero di celle singole da includere all’interno della griglia di gioco, quale insieme di cifre e quali operazioni aritmetiche utilizzare e se mostrarne i simboli all’interno delle celle, per incrementare il livello di difficoltà.

Una volta verificato che tutte le celle sono disposte in maniera corretta, quindi che non vi siano cifre ripetute all’interno di una stessa riga o all’interno di una stessa colonna, e che il puzzle abbia una sola soluzione, un apposito modulo del software tenta di risolvere il puzzle fingendo di essere un utente in modo da determinarne la difficoltà: se essa corrisponde alla difficoltà richiesta il puzzle viene accettato, altrimenti viene scartato e il processo di generazione automatica ricomincia.

*Kenerator,* sebbene costituisca lo stato dell’arte nella categoria di software per la generazione e la risoluzione di giochi di KenKen, non è l’unico. Esempi degni di nota sono [*kenken-maker*](https://github.com/calebsander/kenken-maker)di Caleb Sander e [KenKen solver](https://tirl.org/software/kenken/)di Charlie DeTar.

Il primo, il cui codice sorgente è pubblicamente disponibile su github, segue un’interessante iter per la generazione: si parte dalla costruzione di una griglia valida, che rispetti dunque i vincoli di non ripetizione delle cifre all’interno della stessa riga o della stessa colonna, tramite scambi e trasposizioni di righe a partire da una configurazione di base, dopodiché si costruiscono in maniera randomica i vari blocchi con le operazioni aritmetiche da soddisfare e si tenta di risolvere il puzzle ottenuto per determinarne la difficoltà.

Il secondo, utilizzabile via web, permette non solo la generazione e la risoluzione automatica di giochi di KenKen, ma fornisce anche all’utente la possibilità di costruire una griglia di gioco manualmente.

Un ulteriore esempio sui generis è [KenKen Solver](https://kenkensolver.herokuapp.com/) di Kenneth Myers: esso si differenzia dai software precedentemente analizzati in quanto utilizza tecniche di *computer vision* per risolvere dei giochi di KenKen a partire da un’immagine della griglia di gioco, fornita dall’utente via web.

B. Raffinamento dei Requisiti

***B.1 Servizi (con prioritizzazione)***

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **SERVIZIO** | **IMPORTANZA** | **COMPLESSITA’** |
| #1 | Configurazione manuale della griglia di gioco | Alta | Media |
| Tramite GUI è possibile specificare le dimensioni della griglia di gioco, inizialmente vuota, e costruire in maniera dinamica, interattiva ed arbitraria lo schema di blocchi contigui su cui definire l’operazione aritmetica e il risultato da ottenere. | | | |

| #2 | Salvataggio/caricamento della griglia di gioco su/da file system | Bassa | Media |
| --- | --- | --- | --- |
| Si definisce un sistema di lettura/scrittura delle griglie di gioco per permettere di salvare su file system la griglia di gioco creata e ripristinare griglie di gioco precedentemente memorizzate. I file vengono memorizzati in un formato apposito *(*con estensione *.kenken)* in una cartella di salvataggio che può essere scelta dall’utente. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **SERVIZIO** | **IMPORTANZA** | **COMPLESSITA’** |
| #3 | Inserimento dinamico di numeri all’interno della griglia di gioco | Alta | Bassa |
| Una volta specificato e confermato lo schema di gioco le celle della griglia diventano campi di testo interattivi in cui è sempre possibile inserire e cancellare dei numeri. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| #4 | Controllo dinamico del soddisfacimento dei vincoli | Media | Alta |
| In ogni momento è possibile richiedere il controllo del soddisfacimento dei vincoli della griglia di gioco corrente: le celle contenenti dei numeri che risultano essere duplicati all’interno della stessa riga o della stessa colonna vengono evidenziate come errate. Inoltre si verifica che i blocchi completi rispettino il vincolo aritmetico: in caso negativo l’intero blocco viene evidenziato come errato. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| #5 | Calcolo delle soluzioni del gioco e visualizzazione su schermo | Alta | Alta |
| L’utente può richiedere la visualizzazione della soluzione in ogni momento del gioco, che può avere nessuna, una o più soluzioni. Si può specificare il numero massimo di soluzioni da visualizzare e navigare sullo spazio delle soluzioni trovate per visualizzarle individualmente. Le soluzioni vengono calcolate automaticamente utilizzando la tecnica del backtracking. | | | |

***B.2 Requisiti non funzionali***

1. Filtraggio dell’input dell’utente in base alle regole del gioco: è impossibile inserire numeri non validi o operazioni aritmetiche non ammesse

2. Cancellamento di tutti gli inserimenti avvenuti sulla griglia di gioco

3. Possibilità di annullare o ripetere le seguenti operazioni: creazione dei blocchi di gioco in fase di configurazione, cancellamento di tutti gli inserimenti avvenuti nella griglia di gioco, visualizzazione delle soluzioni del gioco corrente

***B.2 Scenari d’uso dettagliati***

|  |  |
| --- | --- |
| **Creazione di un nuovo gioco** | |
| Tipo | Primario |
| Svolgimento normale | **1.** L’utente seleziona la voce del menu per la creazione di un nuovo gioco  **2.** L’utente specifica la dimensione della griglia di gioco desiderata  **3.** Il sistema presenta una griglia vuota costituita da celle individualmente selezionabili dall’utente  **4.** L’utente seleziona le celle per costruire, uno alla volta, i blocchi di uno schema di gioco  **5.** L’utente conferma la selezione con un apposito pulsante  **6.** Il sistema richiede all’utente un risultato da ottenere combinando i valori contenuti nel blocco selezionato  **7.** Il sistema richiede all’utente un’operazione aritmetica con cui combinare i valori contenuti nel blocco selezionato  **8.** Una volta che tutte le celle appartengono ad un blocco l’utente preme il pulsante per l’avvio del gioco: il sistema rende le celle della griglia di gioco editabili |
| Svolgimento alternativo | **1a.**Una griglia di gioco è stata precedentemente definita: il sistema chiede all’utente se vuole effettuare il salvataggio della griglia di gioco corrente prima di procedere alla creazione di un nuovo gioco  **4a.** L’utente seleziona delle celle non contigue tra loro: il sistema suggerisce all’utente tramite finestra di dialogo di cambiare la selezione  **6a.** L’utente inserisce un valore non intero: il sistema segnala l’errore ed esegue nuovamente la richiesta  **6b.** L’utente annulla l’inserimento di un valore: il sistema riporta l’utente davanti alla griglia con l’ultima selezione di celle eseguita dall’utente ancora valida  **7a.** L’utente inserisce un simbolo non valido: il sistema segnala l’errore ed esegue nuovamente la richiesta  **7b.** L’utente annulla l’inserimento di un simbolo: il sistema riporta l’utente davanti alla griglia con l’ultima selezione di celle eseguita dall’utente ancora valida |
| Descrizione | Questa operazione porta alla creazione di una griglia di gioco con uno schema ben definito e pronta per essere riempita con dei numeri dall’utente |
| **Salvataggio della griglia di gioco** | |
| Tipo | Primario |
| Svolgimento normale | **1.** Il sistema richiede all’utente la selezione di una directory di salvataggio  **2.** L’utente seleziona una directory e sceglie il nome del file in cui memorizzare la griglia di gioco corrente  **3.** Il sistema conferma l’avvenuto salvataggio |
| Svolgimento alternativo | **2a.**L’utente annulla l’operazione di salvataggio  **.1** Il sistema riporta l’utente davanti alla griglia precedentemente definita  **2b.** L’utente sceglie il nome di un file che esiste già  **.1** Il sistema propone la sovrascrittura del file |
| Descrizione | Questa operazione permette all’utente di memorizzare su file system una griglia di gioco precedentemente definita, compresi gli eventuali numeri con cui è stata completata |

|  |  |
| --- | --- |
| **Inserimento di numeri nella griglia di gioco** | |
| Tipo | Primario |
| Svolgimento normale | **1.** L’utente inserisce un numero in una cella della griglia di gioco |
| Svolgimento alternativo | **1a.**L’utente inserisce un valore non intero oppure un numero minore di 1 o maggiore della dimensione della griglia di gioco  **.1** Il sistema impedisce tale inserimento e la cella rimane vuota |
| Descrizione | Questa operazione permette all’utente di inserire all’interno della griglia di gioco soltanto numeri ammissibili |

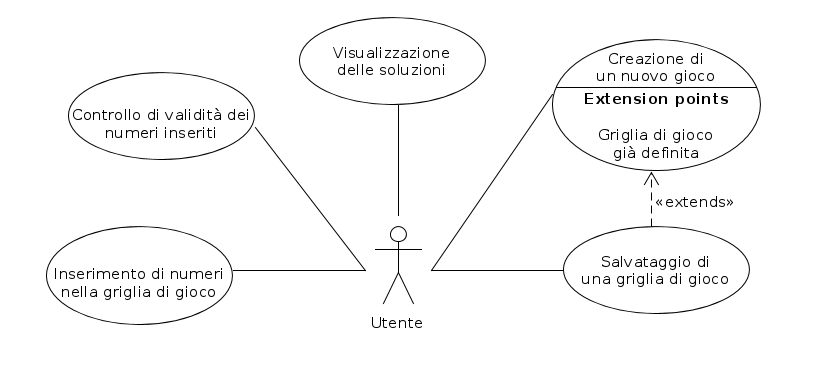
|  |  |
| --- | --- |
| **Controllo di validità dei numeri inseriti** | |
| Tipo | Primario |
| Svolgimento normale | **1.** L’utente richiede tramite GUI il controllo di validità dei numeri inseriti all’interno della griglia di gioco  **2.** Il sistema evidenzia in rosso i numeri che risultano essere duplicati all’interno della stessa riga o della stessa colonna e per ogni blocco completo che non rispetta il vincolo aritmetico, evidenzia in rosso il risultato da ottenere |
| Descrizione | Questa operazione permette all’utente di verificare che gli inserimenti effettuati rispettano le regole del gioco |

|  |  |
| --- | --- |
| **Visualizzazione delle soluzioni** | |
| Tipo | Primario |
| Svolgimento normale | **1.** L’utente richiede tramite GUI la visualizzazione delle soluzioni del gioco corrente  **2.** Il sistema chiede all’utente il numero massimo di soluzioni da visualizzare  **3.** Il sistema calcola le soluzioni, visualizza a schermo la prima soluzione ottenuta e mette a disposizione dell’utente un sistema di navigazione per visualizzarle tutte individualmente  **4.** L’utente scorre tramite GUI tra le soluzioni ottenute |
| Svolgimento alternativo | **2a.**L’utente annulla l’operazione di richiesta delle soluzioni  **.1** Il sistema riporta l’utente davanti alla griglia precedentemente definita  **3a.** Non esistono soluzioni per lo schema di gioco corrente: il sistema informa l’utente tramite una finestra di dialogo |
| Descrizione | Questa operazione permette all’utente di visualizzare dinamicamente le soluzioni del gioco |

***B.3 Assunzioni***

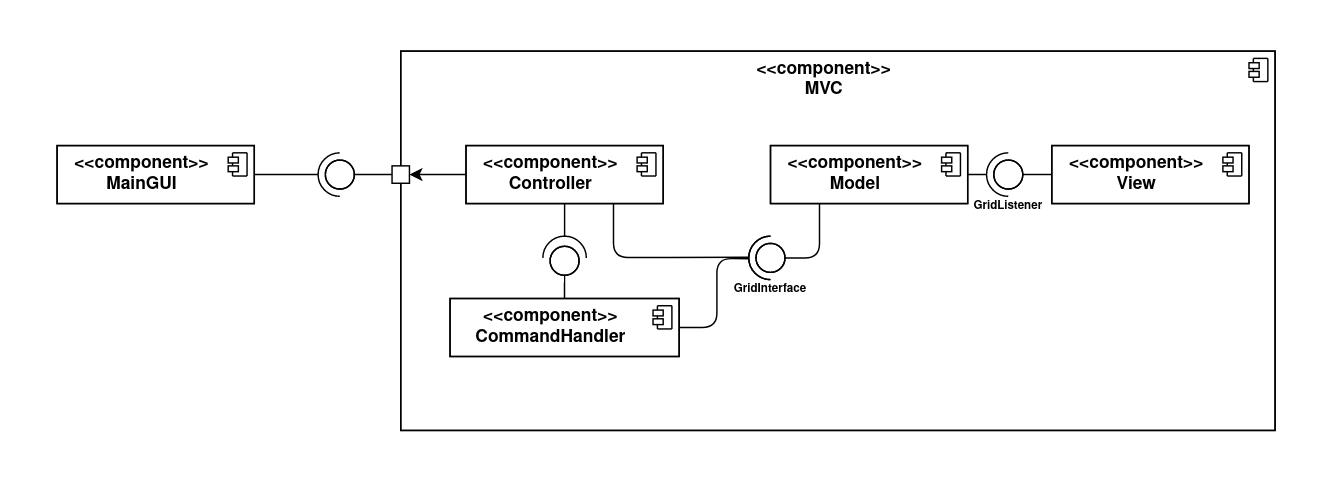
* L’utente inserisce ed elimina i numeri all’interno della griglia di gioco utilizzando la tastiera
* Una volta avviata la partita, lo schema di gioco non può più essere modificato

***B.4 Use Case Diagrams***



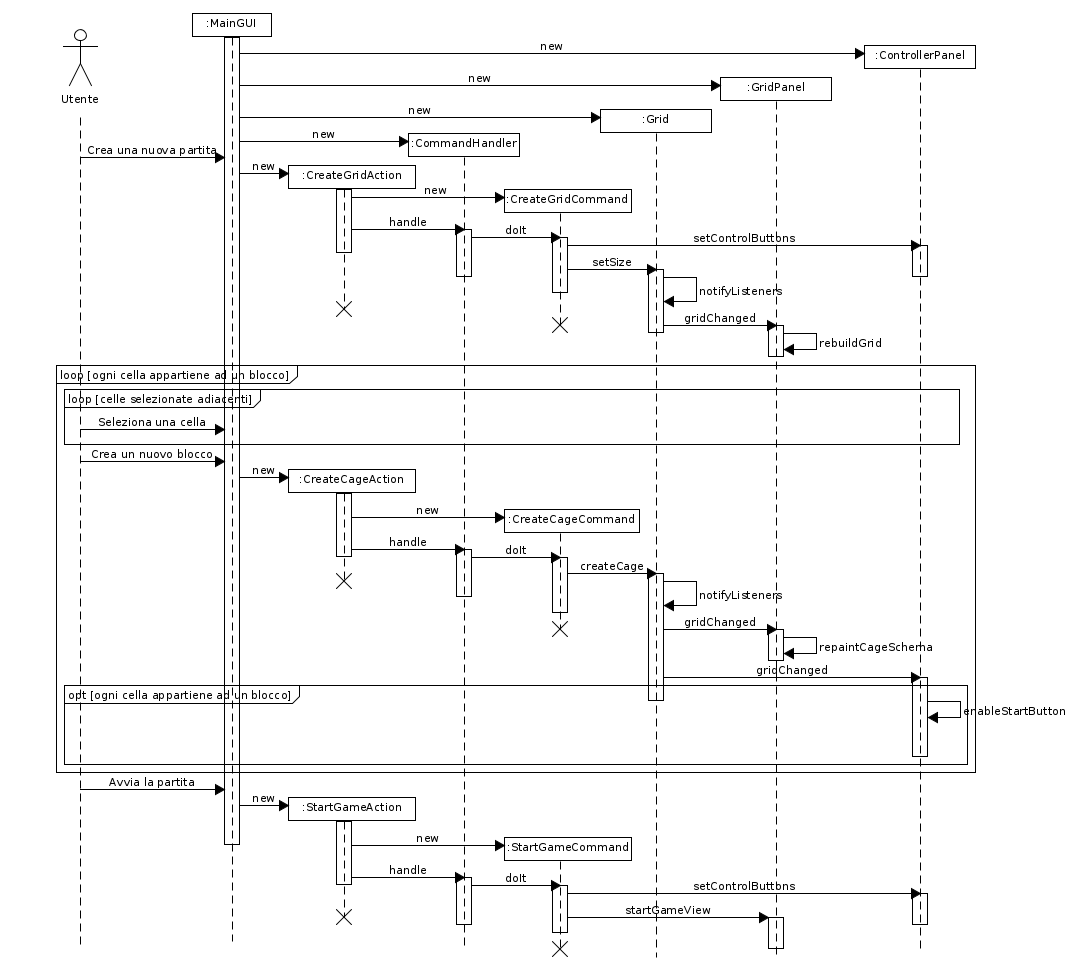
C. Architettura Software

***C.1 The static view of the system: Component Diagram***

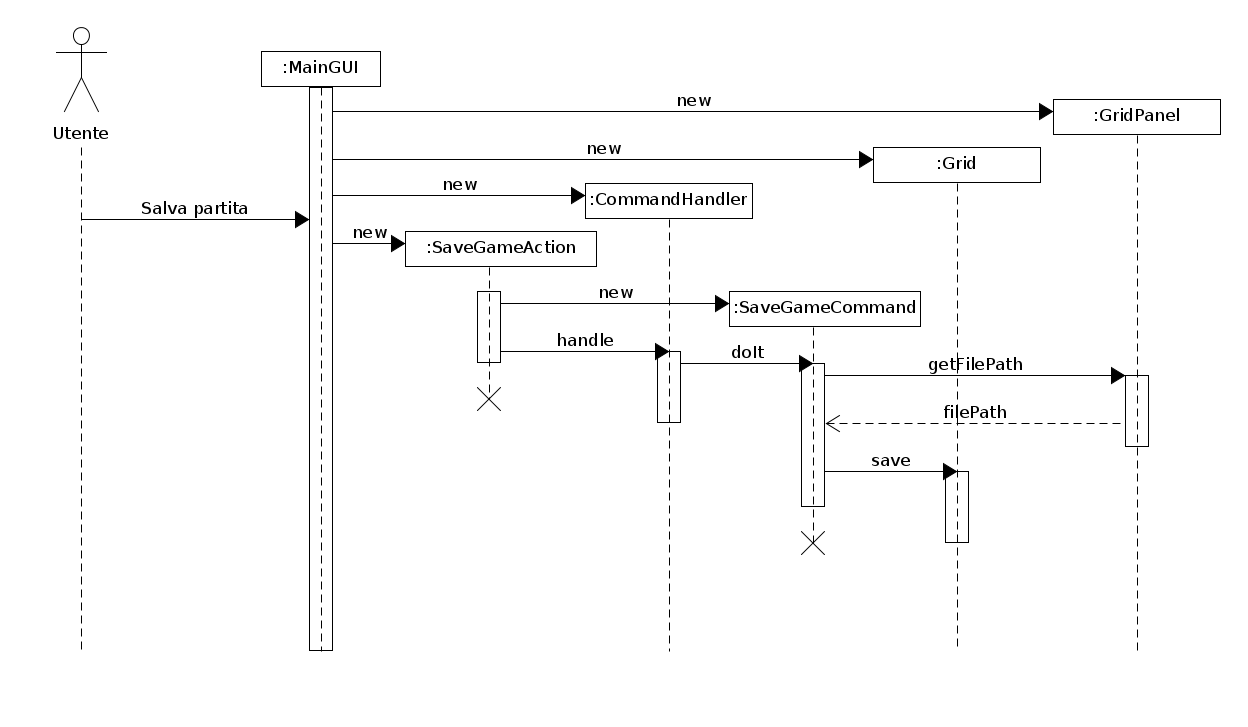


***C.2 The dynamic view of the software architecture: Sequence Diagram***

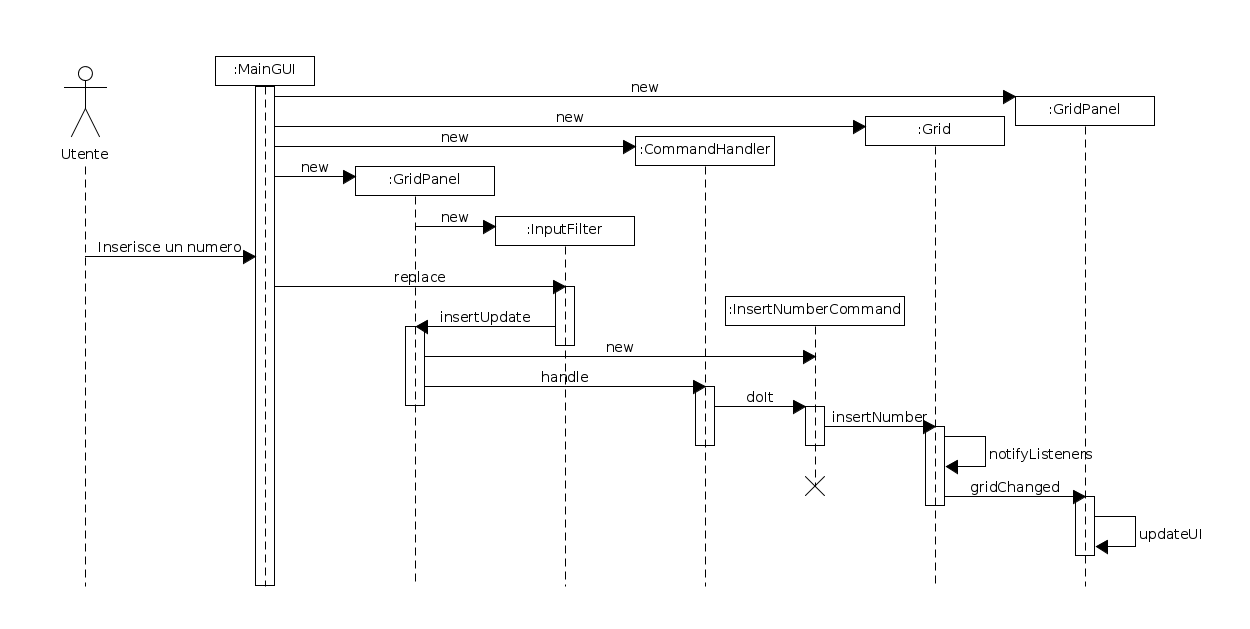
|  |
| --- |
| **Creazione di un nuovo gioco** |



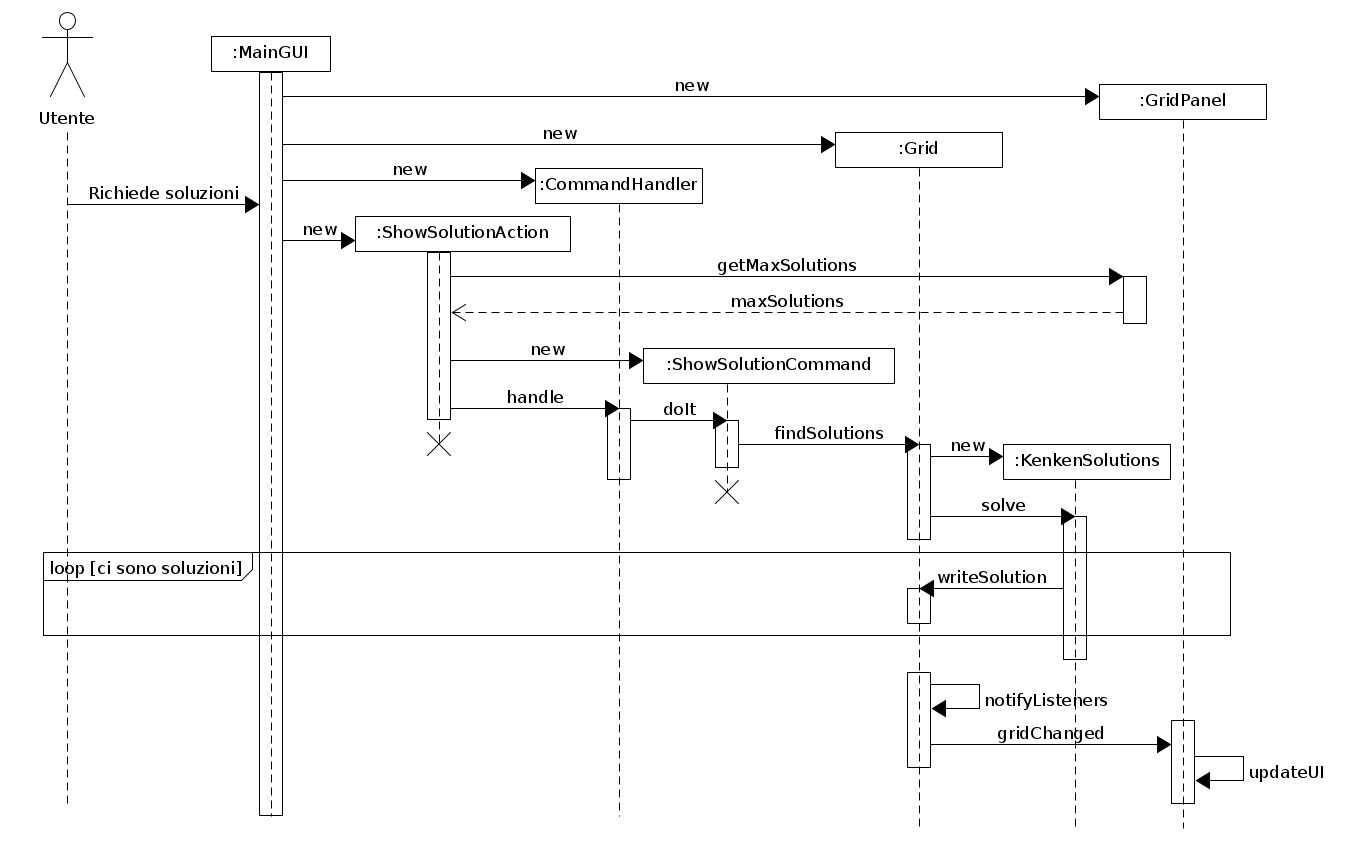
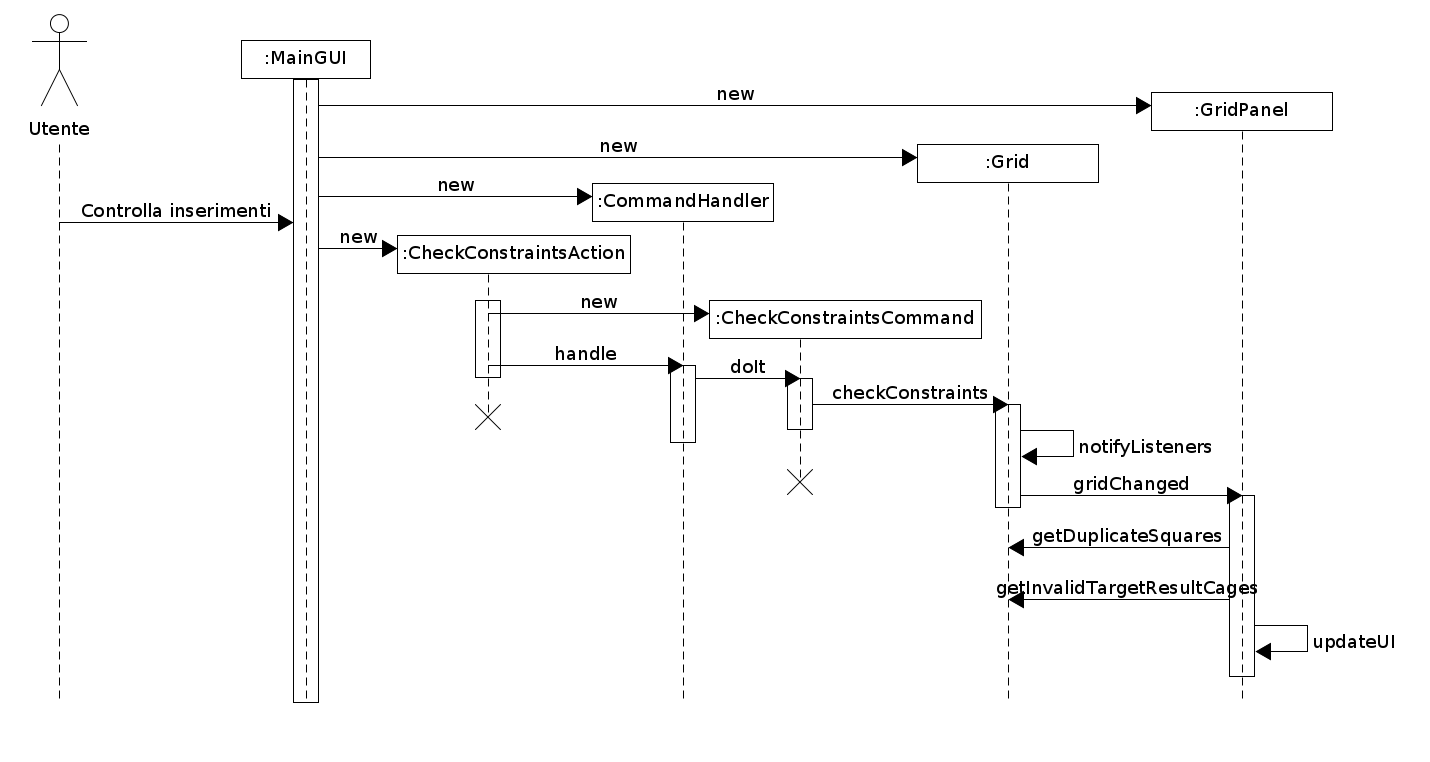
|  |
| --- |
| **Salvataggio della griglia di gioco** |

******

|  |
| --- |
| **Inserimento di numeri nella griglia di gioco** |

******

|  |
| --- |
| **Controllo di validità dei numeri inseriti** |



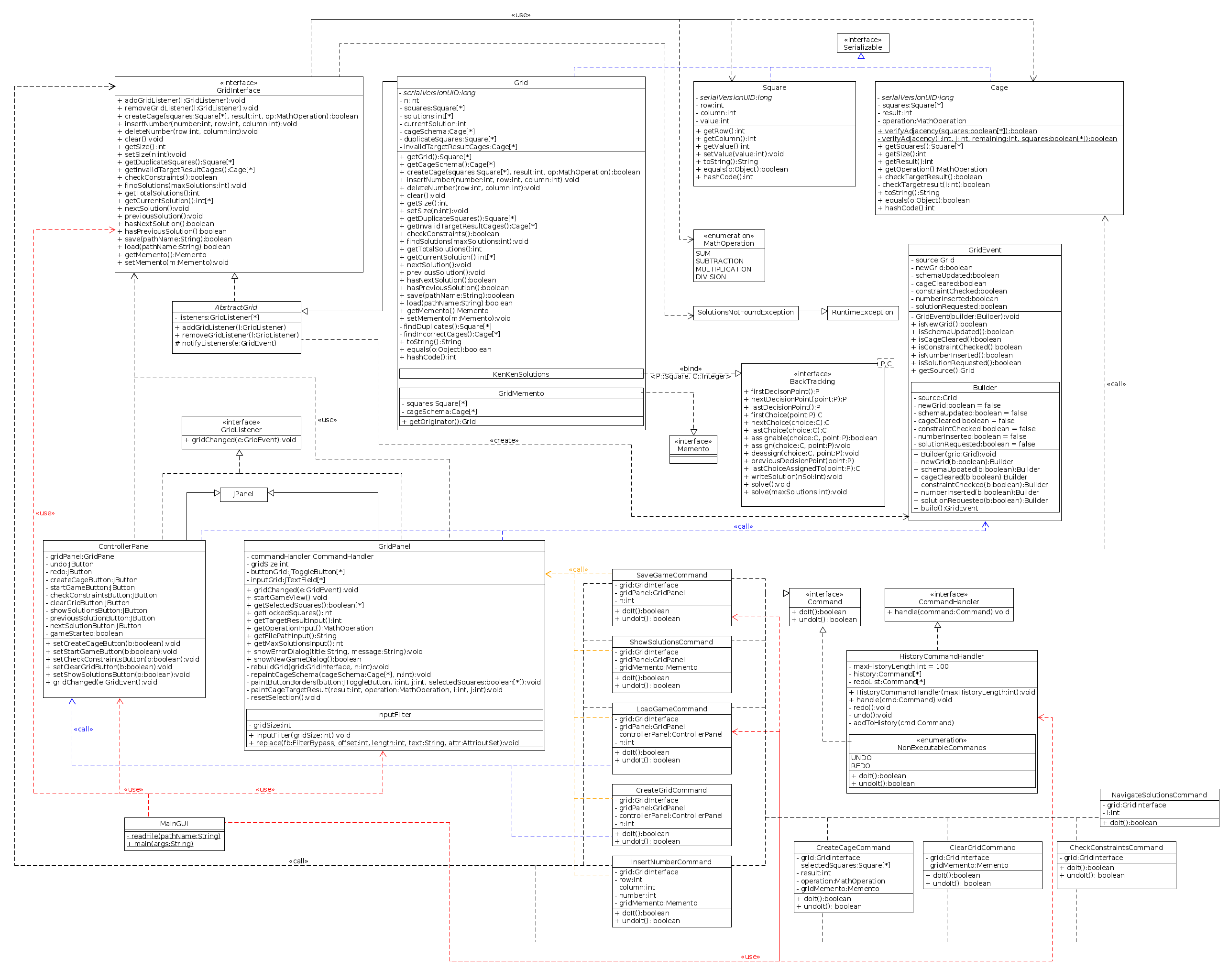
|  |
| --- |
| **Visualizzazione delle soluzioni** |

D. Scelte Progettuali (Design Decisions)

|  |  |
| --- | --- |
| **N°** | **SCELTA PROGETTUALE** |
| 1 | Architettura MVC con *design pattern Observer*  Per gestire al meglio l’interazione dell’utente con la griglia di gioco, che svolge il ruolo di Model, la GUI incorpora i ruoli di View e di Controller in quanto visualizza su schermo lo stato della griglia di gioco e mette a disposizione tutti i pulsanti per effettuare su di essa le operazioni. Per ogni operazione effettuata sul Model (*Grid*) tramite il Controller (*ControllerPanel)* la View del sistema (*GridPanel*) viene notificata e automaticamente aggiornata. Il processo di notifica avviene con logica PUSH: il soggetto osservato incapsula il proprio stato interno in un oggetto *GridEvent*, al quale fornisce informazioni aggiuntive che permettono agli observers di determinare cosa è effettivamente cambiato nel soggetto osservato e cosa aggiornare di conseguenza; ciò aumenta l’accoppiamento tra il soggetto e i suoi osservatori, rendendo questi ultimi meno riusabili, tuttavia rende più efficiente il processo di notifica in quanto non c’è bisogno di interrogare il soggetto per individuarne i cambiamenti. |
| 2 | *Design pattern Builder* per la costruzione dell’oggetto *GridEvent*  Nel notificare i suoi ascoltatori, l’oggetto *Grid*, oltre a passare un riferimento a sé stesso, aggiunge all’oggetto *GridEvent* informazioni dettagliate sulla modifica appena subita (sotto forma di variabili booleane); per migliorare la manutenibilità e l’evolvibilità del sistema, tale oggetto viene costruito per mezzo di un *Builder*: in questo modo è possibile specificare di volta in volta e in funzione delle variazioni subite dall’oggetto *Grid,* a quali campi di *GridEvent* assegnare un valore,evitando di creare una moltitudine di costruttori per ogni combinazione di campi. |
| 3 | *Design pattern Command* per l’interazione del controller con il model  Per aumentare la modularità del sistema, le operazioni che possono essere eseguite dal controller vengono incapsulate in degli appositi oggetti che implementano un’interfaccia comune *Command*: in questo modo si favorisce la manutenibilità in quanto eventuali modifiche sulle operazioni sono circoscritte alle classi che le incapsulano. Inoltre si predispone il sistema per supportare l’annullamento delle operazioni. |

|  |  |
| --- | --- |
| **N°** | **SCELTA PROGETTUALE** |
| 4 | *Design pattern Memento* per catturare lo stato interno della griglia di gioco in modo tale da poterlo ripristinare con operazioni di UNDO e REDO  L’oggetto *Grid*, che svolge il ruolo di Model nell’architettura del sistema, è caratterizzato da due variabili d’istanza principali, una matrice di *Square*, oggetti che identificano le celle della griglia di gioco incapsulandone la posizione all’interno della griglia e il valore numerico contenuto, e una lista di oggetti *Cage*, che rappresentano i blocchi dello schema di gioco. Essendo tale stato interno piuttosto complesso ma soprattutto privato, il design pattern Memento permette di catturarlo senza violare l’incapsulamento. La creazione di oggetti Memento concreti, istanze della classe *GridMemento*, viene delegata alla classe *Grid* nel momento in cui viene richiesta l’esecuzione di un nuovo comando annullabile: così facendo sarà possibile riportare la griglia di gioco allo stato in cui si trovava prima di essere modificata. |
| 5 | *Design pattern Template method* per il calcolo delle soluzioni del gioco con backtracking  La struttura tabellare del gioco del KenKen si presta ad una risoluzione esaustiva con tecniche di forza bruta. La presenza dei vincoli aritmetici che caratterizzano il gioco tuttavia permette di ottimizzare tali tecniche di risoluzione ad esempio utilizzando il backtracking: sfruttando il design pattern *Template Method* la struttura risolutiva viene definita a priori in un’apposita interfaccia *Backtracking,* per poi essere ridefinita appositamente per il gioco del Kenken in una classe concreta *KenkenSolutions*. Tutte le soluzioni trovate vengono memorizzate in una struttura dati e possono essere richieste individualmente con dei metodi messi a disposizione dalla classe *Grid*. |

E. Progettazione di Basso Livello



F. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs) L’architettura del sistema software è caratterizzata da tre classi principali modellate secondo il design pattern Model-View-Controller: *Grid, GridPanel* e *ControllerPanel.*

La classe *Grid* svolge il ruolo di Model ed è dunque responsabile della gestione dei dati inseriti dall’utente e della verifica della loro correttezza: tutti i servizi offerti dal sistema dipendono strettamente dalla sua implementazione. *Grid* modella la griglia di gioco con una matrice di oggetti *Square,* *squares*, che rappresentano le singole celle, e una lista di oggetti *Cage*, *cageSchema*, che rappresentano i blocchi dello schema di gioco.  
Per la configurazione dello schema di gioco (**#1**), la classe *Grid* mette a disposizione il metodo *createCage(),* tramite il quale è possibile specificare una lista di celle, un risultato da ottenere e l’operazione aritmetica con cui combinare i valori all’interno del blocco; per il salvataggio e il caricamento della griglia di gioco (**#2**) si predispongono i metodi *save()* e  *load()*; per l’inserimento e la rimozione di numeri nella griglia di gioco (**#3**) si predispongono i metodi *insertNumber()* e *deleteNumber();* per la verifica dei vincoli imposti dalle regole del *KenKen (****#4****)* si predispone un metodo *checkConstraints()*, che restituisce un booleano per attestare la correttezza degli inserimenti avvenuti fino al momento della sua invocazione ed aggiorna le liste di celle e di blocchi che non soddisfano i vincoli, rispettivamente *duplicateSquares* e *invalidTargetResultSquares;* per la risoluzione del gioco (**#5**) si introduce un’apposita interfaccia *Backtracking* in cui si definisce in maniera astratta la struttura algoritmica del backtracking: tale interfaccia viene implementata da una inner class di Grid, *KenkenSolutions*, che ha la responsabilità di ridefinire le operazioni primitive dell’algoritmo per ottenere le soluzioni del gioco.

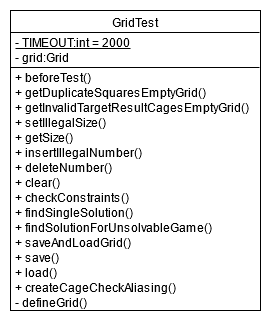
Affinché il sistema possa offrire i servizi definiti in fase di specifica dei requisiti all’utente finale, l’invocazione di tali operazioni sulla griglia di gioco viene incapsulata in degli appositi oggetti che implementano l’interfaccia *Command,* la cui esecuzione viene delegata ad un Controller grafico con cui l’utente può interagire*: ControllerPanel.*

La classe *ControllerPanel* ha la responsabilità di mettere a disposizione dell’utente una serie di pulsanti adibiti all’interazione con la griglia di gioco, quindi di rilevare l’input dell’utente ed invocare degli appositi comandi (servizi **#1**, **#4**, **#5**). Per migliorare l’usabilità del sistema, *ControllerPanel,* seguendo le modalità definite dal design pattern Observer, implementa l’interfaccia *GridListener* per poterrimanere anche in ascolto delle modifiche effettuate sulla griglia di gioco in modo tale da abilitare o disabilitare di conseguenza i pulsanti messi a disposizione dell’utente.

L’utente è in grado di visualizzare le modifiche effettuate sulla griglia di gioco tramite il Controller grazie alla componente View del sistema, la classe *GridPanel*. Essa è caratterizzata da una matrice di celle *buttonGrid* che, in fase di configurazione, sono individualmente selezionabili dall’utente per costruire lo schema di gioco (**#1**). Una volta avviata la partita l’utente può inserire o rimuovere numeri utilizzando la tastiera (**#3**)semplicemente interagendo con una matrice di caselle di testo editabili, *inputGrid*.

La creazione di una nuova partita, il salvataggio della partita corrente o il caricamento di una partita precedentemente salvata (**#2**), sono dei servizi che l’utente può raggiungere dal menu principale della GUI, definita nella classe *MainGUI*.

L’utilizzo del design pattern Command insieme al design pattern Memento permette che anche i requisiti non funzionali siano soddisfatti, migliorando ulteriormente robustezza ed usabilità del sistema: *ControllerPanel* mette a disposizione un apposito pulsante per eliminare tutti gli inserimenti avvenuti nella griglia, e dei pulsanti di UNDO/REDO per annullare le operazioni di creazione di un blocco, eliminazione di tutti gli inserimenti, visualizzazione delle soluzioni, ritornando allo stato in cui si trovava la griglia di gioco prima di eseguire il comando.

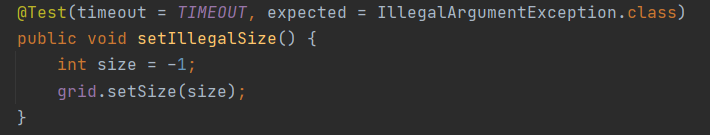
G. Testing

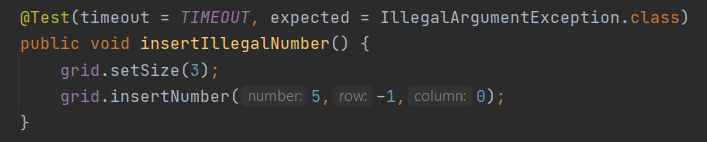
Si definisce un’apposita classe per testare con l’utilizzo di *JUnit4* la classe *Grid*, dalla quale dipende la correttezza del sistema software.

Il metodo *beforeTest()* viene annotato con @Before in modo da essere eseguito prima di ogni metodo della classe: il suo compito è quello di inizializzare la variabile di istanza *grid* ad una griglia vuota.

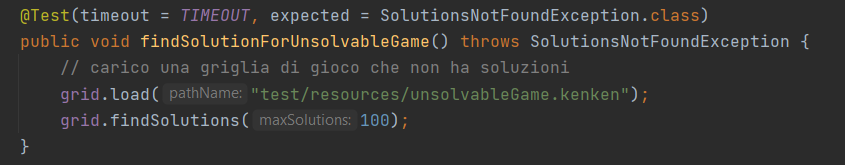
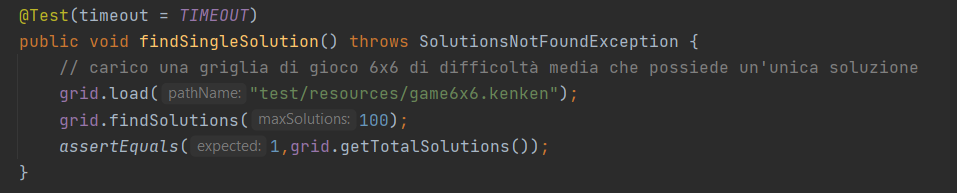
Per tutti i test si verifica che il tempo di esecuzione sia inferiore ai 2000ms in modo da riscontrare eventuali cicli infiniti o deadlock.

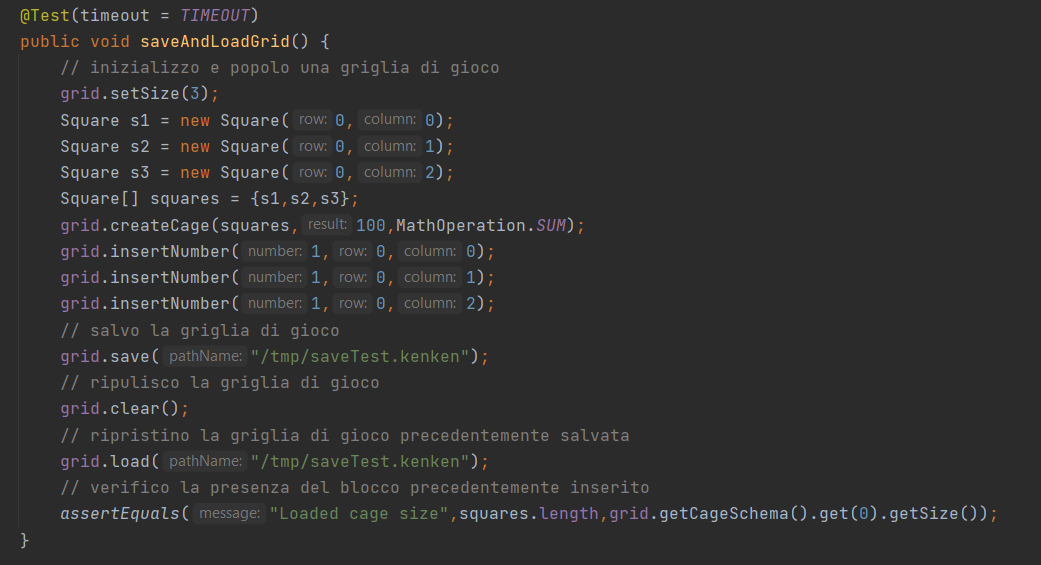
Metodi come *setIllegalSize()* e *insertIllegalNumber()* verificano che il sistema sollevi correttamente l’eccezione IllegalArgumentException nei casi in cui i parametri passati non rispettano i vincoli del gioco:



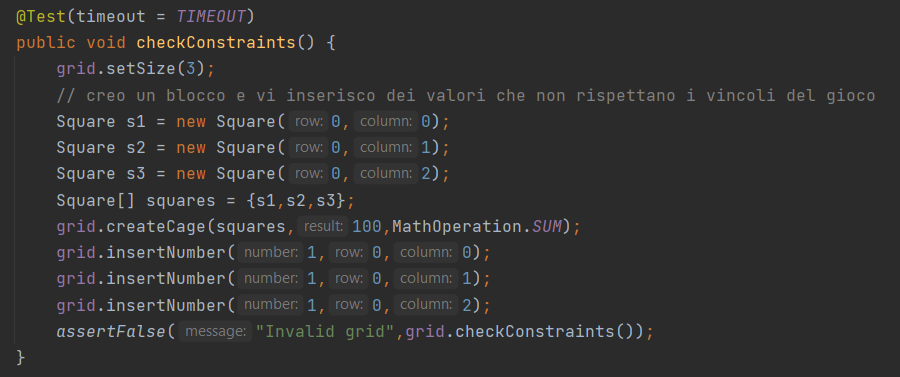


I metodi *findSingleSolution()* e *findSolutionForUnsolvableGame()* sono votati alla verifica della correttezza del metodo risolutivo con backtracking. Il primo carica una griglia che ammette una sola soluzione e verifica che la soluzione ottenuta sia effettivamente una, il secondo carica una griglia che non ammette soluzioni e verifica che un’apposita eccezione venga lanciata per segnalare l’irresolvibilità del gioco:

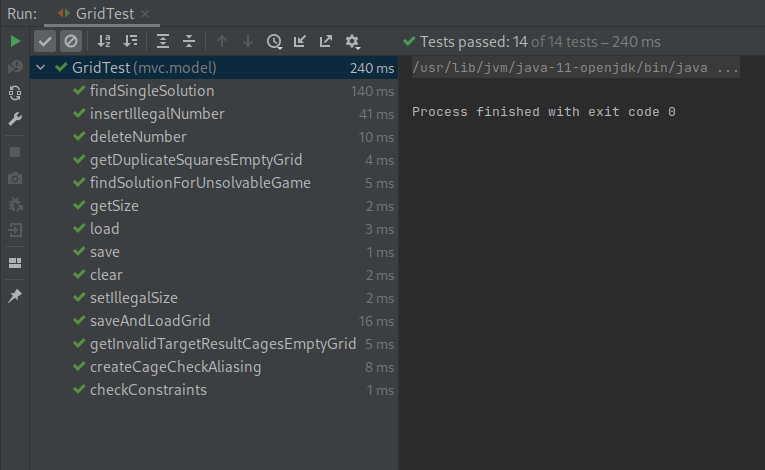


Il metodo *saveAndLoadGrid()* verifica la correttezza del processo di salvataggio e caricamento di una griglia di gioco: affinché il processo vada a buon fine infatti la griglia di gioco caricata deve trovarsi nello stesso stato in cui si trovava la griglia nel momento del salvataggio; in questo test si verifica che il blocco inserito prima del salvataggio sia della stessa dimensione del blocco ottenuto dopo il caricamento:

Infine, per completare una panoramica dei test effettuati sulle operazioni fondamentali offerte dal sistema software, il metodo *checkConstraints()* verifica la correttezza del processo di validazione degli inserimenti avvenuti:



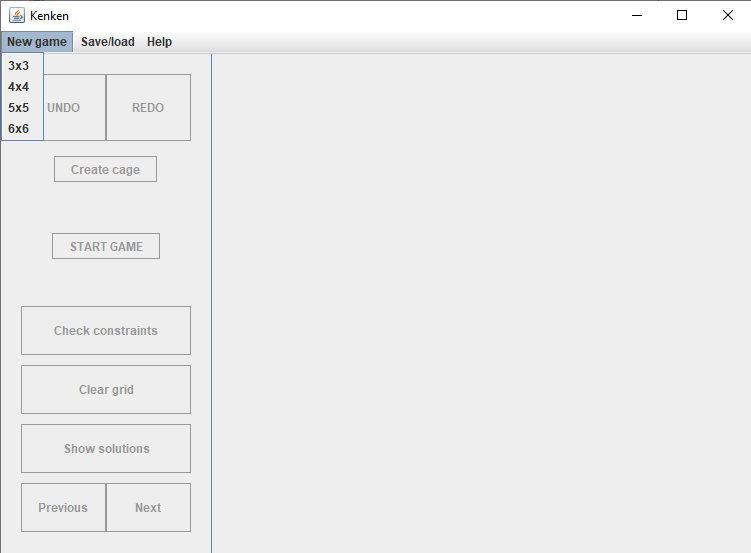
Risultati finali:



Appendix. Prototype 

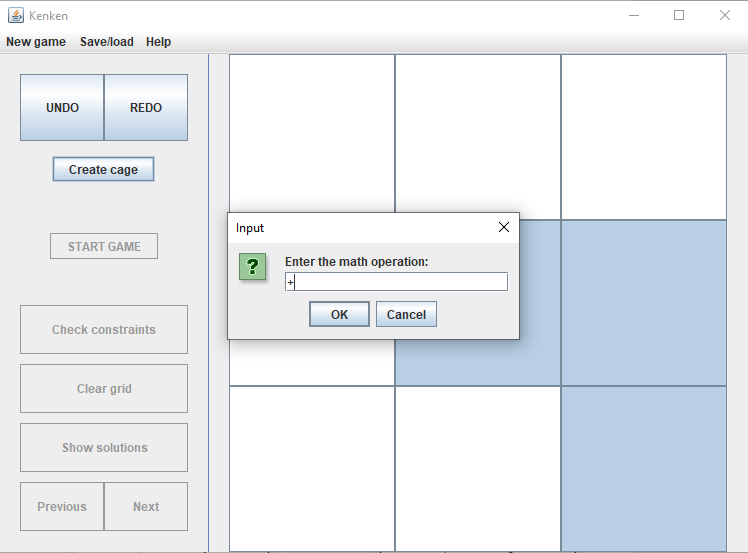
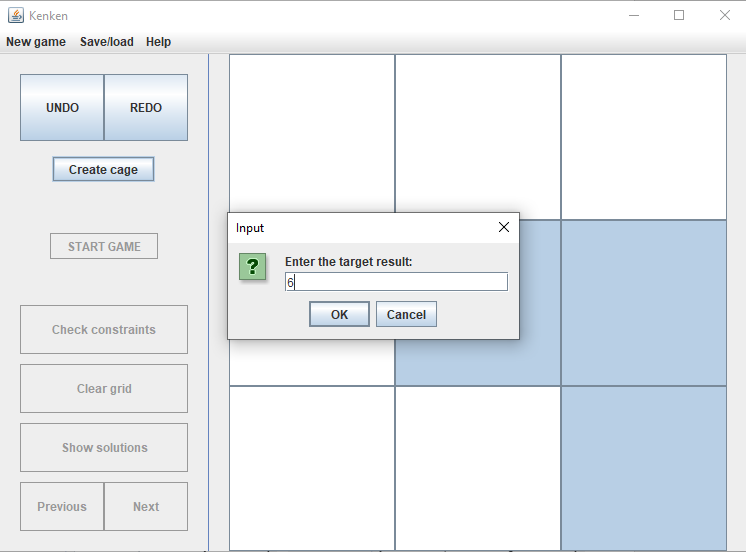
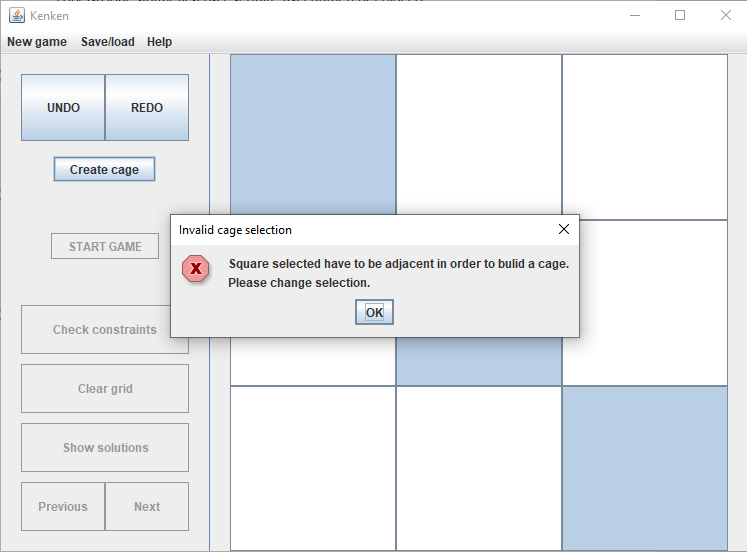
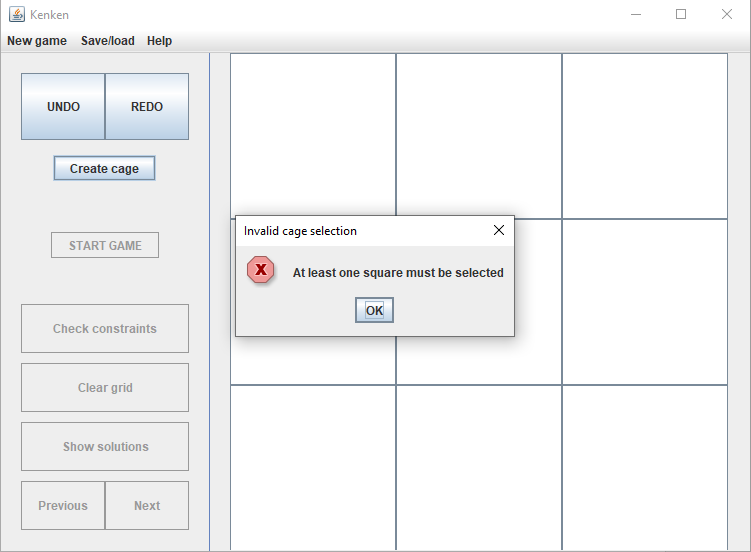
Il sistema software si presenta all’utente finale in una veste grafica implementata dalla classe *MainGUI* che costruisce una *JFrame*, la quale comprende:

* un *JMenu* con delle voci *JMenuItem* per le operazioni di creazione di una nuova partita, salvataggio/caricamento di una partita, richiesta di informazioni sul funzionamento del software e sulle regole del gioco (che vengono lette da appositi file di testo);
* un *JPanel* di controllo laterale implementato dalla classe *ControllerPanel*;
* un *JPanel* per contenere la griglia di gioco, implementato dalla classe *GridPanel*

**

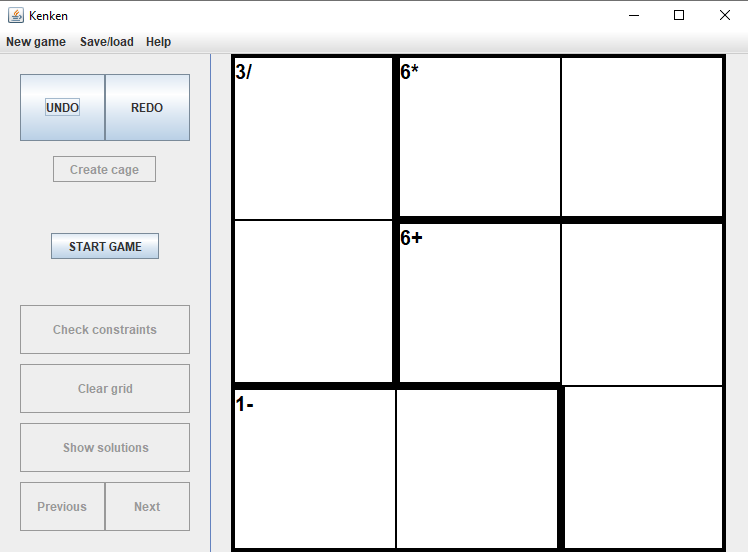
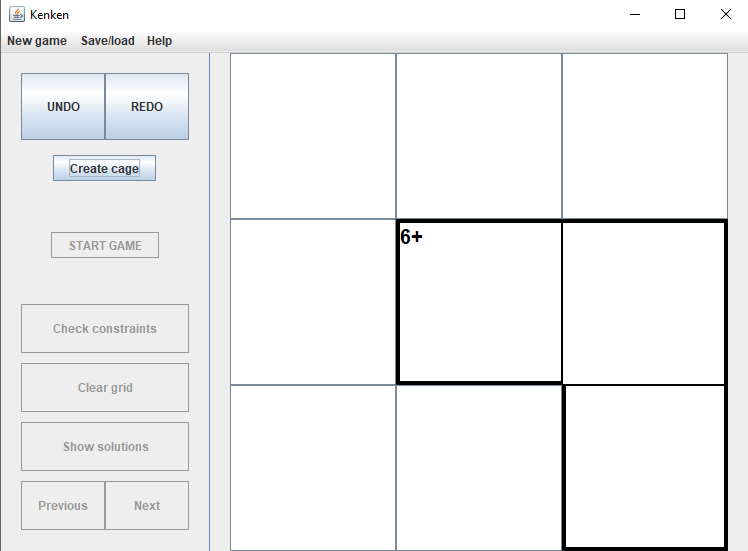
Alla creazione di un nuovo gioco, si invoca sull’oggetto *Grid* il metodo *setSize()* per inizializzare le variabili d’istanza in base alla dimensione specificata e, tramite il metodo *notifyListeners()*, si informano gli oggetti registrati come *GridListener* dell’avvenuta creazione: *GridPanel* visualizza su schermo una matrice di *JToggleButton*, *ControllerPanel* abilita i pulsanti di per permettere la configurazione dello schema di gioco.

Alla pressione del pulsante CREATE CAGE, *ControllerPanel* richiede la selezione corrente al *GridPanel,* verifica che le celle selezionate siano adiacenti, invocando il metodo statico *verifyAdjacency()* di *Cage* e, in caso affermativo, richiede all’utente tramite *inputDialog* un risultato da ottenere e l’operazione aritmetica da eseguire tra i numeri del blocco.



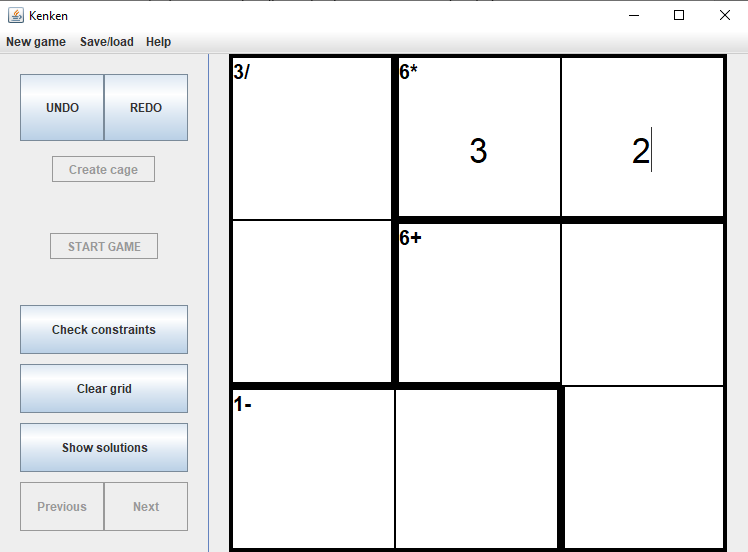
Una volta ottenute le informazioni necessarie per creare il blocco, si istanzia l’apposito comando *CreateCageCommand* che invoca sull’oggetto *Grid* il metodo *createCage()* e notifica i listeners.

Ogni volta che lo schema di gioco viene modificato, l’oggetto *GridEvent* passato come parametro del metodo *notifyListeners()* viene impostato con variabile booleana *schemaUpdated = true*. In questo modo *GridPanel* esegue il metodo *repaintCageSchema()* che scorre tra gli oggetti *Cage* della lista *cageSchema*, individua le celle adiacenti di uno stesso blocco e annerisce i bordi, mentre *ControllerPanel* richiede al *GridPanel* il numero di celle bloccate e, nel caso in cui ogni cella appartiene ad un blocco, abilita il pulsante START GAME.



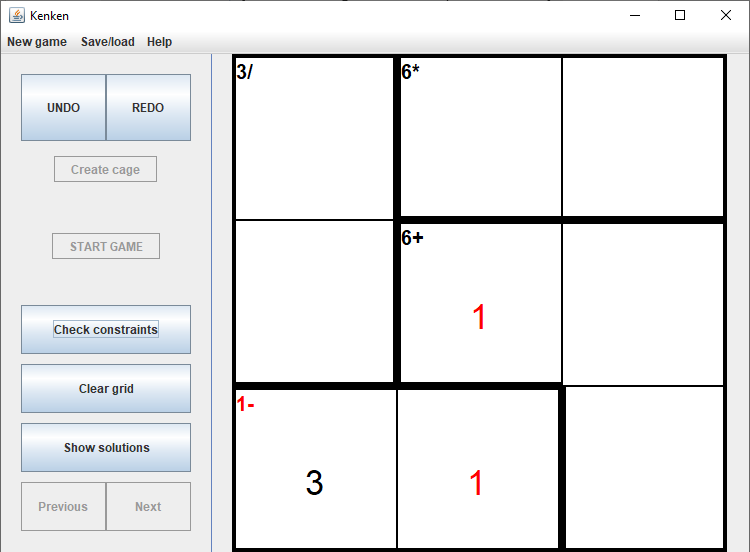
In qualsiasi momento nella fase di configurazione dello schema di gioco, è possibile annullare l’ultima operazione di creazione di un blocco: nel momento in cui l’oggetto *CreateCageCommand* viene istanziato infatti si inzializza una variabile d’istanza di tipo *Memento* che incapsula lo stato della griglia prima che il comando debba essere eseguito. Alla pressione del pulsante UNDO, l’oggetto *HistoryCommandHandler* provvede ad invocare sul comando il metodo *undoIt(),* il quale a sua volta invoca sull’oggetto *Grid* il metodo *setMemento()* passando come parametro il Memento precedentemente ottenuto, in modo da ripristinarne lo stato interno.

L’inserimento dei numeri nella griglia di gioco può avvenire quando tutte le celle appartengono almeno ad un blocco: *GridPanel* istanzia degli oggetti *JTextField* che vengono aggiunti all’interno degli oggetti *JToggleButton* che costituiscono la griglia di gioco e che vengono disabilitati nel momento in cui i blocchi vengono creati. Ad ogni *JTextField* è associato un *DocumentFilter* custom, istanza della inner class di *GridPanel*, *InputFilter*, il quale, prima di rendere visibile l’inserimento dell’utente, verifica che l’input sia valido con una regex (quindi che sia un numero intero di una sola cifra e che sia compreso tra 1 e la dimensione della griglia di gioco); gli inserimenti nella griglia di gioco vengono rilevati da un *DocumentListener* istanziato al volo la cui responsabilità è invocare l’apposito comando *InsertNumberCommand* per rendere persistente l’inserimento sull’oggetto *Grid.*



All’avvio della partita vengono abilitati i pulsanti CHECK CONSTRAINTS, CLEAR GRID, SHOW SOLUTIONS.

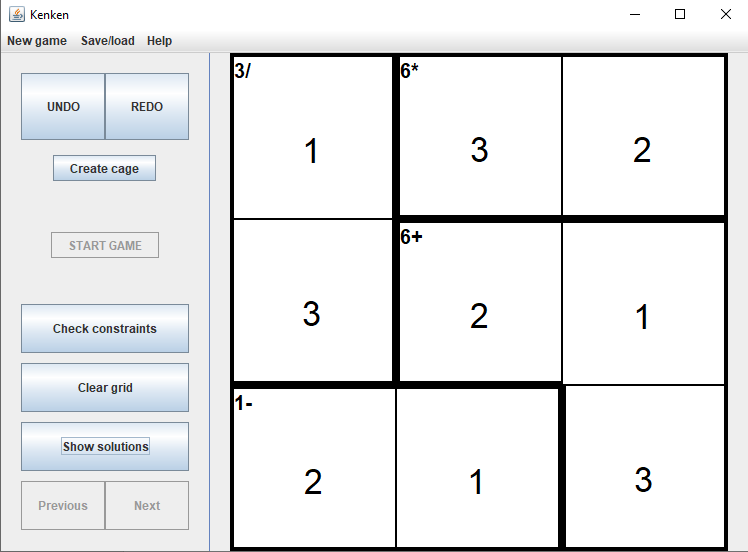
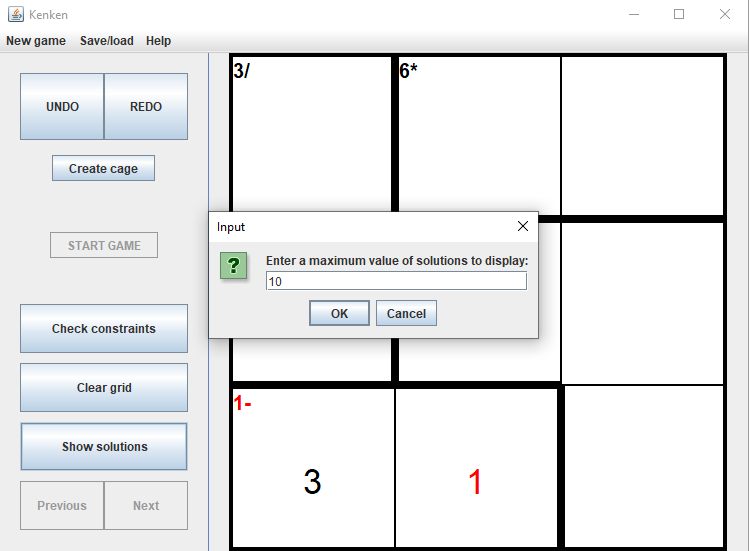
Il primo esegue l’omonimo comando che invoca sull’oggetto *Grid* il metodo *checkConstraints()*: il *GridPanel* viene notificato, estrapola le celle e i blocchi invalidi e li evidenzia di colore rosso:



Il secondo ripulisce la griglia di gioco dagli inserimenti effettuati dall’utente: come *CreateCageCommand,* anche *ClearGridCommand* è annullabile, grazie all’istanziazione di un oggetto Memento.

Il terzo richiede all’utente tramite *inputDialog* il numero massimo di soluzioni da individuare con la tecnica del backtracking e automaticamente visualizza la prima di esse, se esiste, su schermo. Anche *ShowSolutionsCommand* è annullabile: ciò è importante per l’usabilità del sistema in quanto, una volta analizzata la soluzione, si può tornare indietro riportando la griglia ad avere gli inserimenti effettuati prima della richiesta di risoluzione, ed eventualmente proseguire con il gioco.

Nel caso in cui vi siano più di una soluzione, è possibile visualizzarle individualmente con i pulsanti PREVIOUS e NEXT, i quali vengono abilitati da *ControllerPanel* solo nel caso in cui l’invocazione dei metodi *hasPreviousSolution()* e *hasNextSolution()* sull’oggetto *Grid* restituisce un valore booleano positivo



In qualsiasi momento è possibile decidere di salvare la partita corrente, compresi gli inserimenti effettuati, oppure di caricarne una dal file system.

I comandi *SaveGameCommand* e *LoadGameCommand* eseguono rispettivamente i metodi *save()* e *load()* dell’oggetto *Grid*: questi ultimi fanno uso rispettivamente di un *ObjectOutputStream* e di un *ObjectInputStream* per scrivere/leggere su/da file binario con estensione *.kenken*, motivo per cui le classi *Grid, Square* e *Cage* implementano l’interfaccia *Serializable*.

