**“Ingegneria del Software”**

**2021-2022**

**Docente: Prof. Angelo Furfaro**

**<nome progetto>**

|  |  |
| --- | --- |
| **Data** | <gg/mm/aaaa> |
| **Documento** | Documento Finale – D3 |

|  |  |  |
| --- | --- | --- |
| **Team Members** | | |
| **Nome e Cognome** | **Matricola** | **E-mail address** |
| **De Marco Luca** | **234373** | **Dmrlcu01d24a053x@studenti.unical.it** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Sommario

**Non è stata trovata alcuna voce d'indice.**

## 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** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

A. Stato dell’Arte

Per cercare di comprendere meglio il dominio applicativo del gioco KenKen, è stata condotta un’analisi di alcune implementazioni di giochi KenKen disponibili online, sotto forma di applicazione web e mobile.

Questa analisi ha rilevato una varietà di approcci per quanto riguarda l’interfaccia utente (UI), le modalità di interazione, come l’inserimento dei numeri all’interno delle celle, la presentazione delle gabbie e delle regole, e le funzionalità offerte, come il controllo degli errori, i suggerimenti o la risoluzione automatica del gioco.

Dall’esame di queste soluzioni esistenti, sono emersi alcuni elementi particolarmente interessanti per la loro chiarezza e facilità d’uso, che si ritiene possano contribuire positivamente all’esperienza utente nell’uso del gioco. L’obiettivo non è replicare una specifica implementazione, ma trarre ispirazione dagli aspetti più riusciti osservati, combinandoli in modo coerente con i requisiti del progetto.

In particolare, si è notato come alcune interfacce web, come il sito [www.kenkenpuzzle.com](http://www.kenkenpuzzle.com) presentino la griglia e le gabbie in modo chiaro e leggibile. Questo approccio grafico sarà preso come riferimento per la progettazione dell’UI del progetto, in quanto permette all’utente di comprendere rapidamente lo schema di gioco. Anche per quanto riguarda l’inserimento dei valori numerici offre un metodo abbastanza intuitivo, si cercherà di implementare una soluzione simile, così da avere una maggiore facilità di gioco ed evitare l’inserimento di valori non concessi all’interno delle celle.

B. Raffinamento dei Requisiti

*A partire dai servizi minimali richiesti, raffinate la descrizione dei servizi offerti dal vostro applicativo. Descrivete anche I requisiti non funzionali*

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

* **ID:** RF01
  + **Nome Servizio:** Definizione Parametri Griglia KenKen
  + **Descrizione Dettagliata:** L’applicazione deve consentire all’utente di iniziare la creazione di un nuovo puzzle KenKen specificando la dimensione della griglia che deve essere necessariamente N (dove N indica la dimensione delle righe e delle colonne, in quanto il puzzle deve essere necessariamente quadrato), da un minimo di 3 ad un massimo di 6, tramite un elemento dell’UI. Questa selezione determinerà la creazione di una griglia di dimensione N su cui verranno poi create le gabbie. Per quanto riguarda la scelta della modalità di gioco l’utente potrà selezionare tra le diverse tipologie, che consistono nei vari tipi di operazioni matematiche (ossia +, -, x, ÷), ci saranno 3 tipi di modalità. Nella modalità easy saranno presenti solo operazioni di addizione e sottrazione, nella modalità media saranno presenti operazioni di moltiplicazione e divisione, mentre nella modalità hard saranno presenti tutti i tipi di operazioni. La scelta di dimensione e di vincolo delle operazioni determineranno la difficoltà del puzzle.
  + **Soluzione Concettuale:** È necessaria una funzione che presenti all’utente l’opzione per determinare la dimensione della griglia e della scelta dei vincoli. La scelta dell’utente viene memorizzata e utilizzata per inizializzare la struttura dati principale che rappresenta la griglia di gioco.
  + **Priorità:** 
    - Importanza: Alta (È il primo passo indispensabile per la creazione del puzzle)
    - Complessità: Bassa (Input facile da gestire)
* **ID:** RF02
  + **Nome Servizio:** Generazione Automatica Puzzle KenKen
  + **Descrizione Dettagliata:** L’applicazione deve avere la capacità di generare proceduralmente un nuovo Puzzle KenKen valido. L’utente dovrà poter richiedere la generazione specificando la dimensione N della griglia e la modalità di gioco. L’algoritmo dovrà creare una disposizione valida di gabbie, con i relativi target numerici e utilizzando operatori aritmetici conformi alla modalità scelta (vedi **RF01**), assicurando che il puzzle abbia almeno una soluzione disponibile (idealmente una). La configurazione viene quindi rappresentata all’utente così che possa iniziare a giocare
  + **Soluzione Concettuale:** È necessario sviluppare un algoritmo di generazione del Puzzle KenKen. Questo algoritmo parte da una griglia latina completa (una griglia già risolta, quindi una potenziale soluzione), la suddivide in modo casuale, calcola i vincoli per ciascuna gabbia, e verifica le proprietà del puzzle risultante.
  + **Priorità:**
    - Importanza: Alta
    - Complessità: Alta
* **ID:** RF03 (Pattern Memento)
  + **Nome Servizio:** Salvataggio Stato Partita Corrente
  + **Descrizione Dettagliata:** Il sistema deve fornire all’utente la funzionalità di salvare lo stato corrente della partita, così che possa interromperla e riprenderla in un momento successivo dallo stesso punto. L’operazione di salvataggio verrà avviata dall’utente tramite un comando esplicito nell’interfaccia grafica. Il salvataggio deve catturare e memorizzare in maniera persistente su un repository le informazioni necessarie a ricostruire la sessione di gioco. Obbligatoriamente deve includere la dimensione N del puzzle, la modalità selezionata, la struttura dettagliata di ogni gabbia (le celle che la compongono, l’operatore e il numero target), la posizione e il valore di eventuali numeri fissi iniziali che definiscono il puzzle, i vari numeri immessi dall’utente nelle varie celle se presenti, specificandone la riga e la colonna.
  + **Soluzione Concettuale:** Il servizio richiede una funzione capace di fare uno snapshot all’intero stato della partita al momento della richiesta di salvataggio. Questa istantanea deve includere sia la configurazione statica del puzzle ma anche lo stato dinamico ossia gli input dell’utente. Questo snapshot deve essere serializzato in un formato dati standardizzato e persistente. I dati serializzati vengono scritti sul supporto di memorizzazione prescelto.
  + **Priorità:**
    - Importanza: Alta
    - Complessità: Alta
* **ID:** RF04 ( Pattern Observer)
  + **Nome Servizio:** Inserimento Numeri Utente
  + **Descrizione Dettagliata:** Una volta visualizzato il puzzle KenKen (generato tramite RF02 o caricato da un salvataggio), il sistema deve permettere all’utente di interagire con esso per tentare di risolverlo. Una volta selezionata una cella che non contenga un numero fisso iniziale, l’utente deve poter inserire in essa un valore numerico. Il sistema deve effettuare una validazione di base dell’input: sono accettati solo valori interi compresi nell’intervallo da 1 ad N, dove N è la dimensione della griglia. Qualsiasi input non valido deve essere segnalato come errore. Se il numero è valido deve essere visualizzato all’interno della cella. L’utente deve inoltre poter cancellare un numero precedentemente inserito in una cella riportandolo allo stato di vuoto.
  + **Soluzione Concettuale:** È necessario implementare un meccanismo nell’interfaccia utente che permetta all’utente di indicare univocamente la cella bersaglio della sua azione. Serve poi un gestore di eventi per catturare l’input numerico proveniente dalla fonte di scelta. Una logica di validazione deve verificare che l’input sia intero consentito. Lo stato interno del gioco che rappresenta i valori correnti in ogni cella, deve essere aggiornato in base all’input validato ricevuto. Infine, la rappresentazione visiva della cella nell'interfaccia utente (Vista) deve essere sincronizzata per riflettere il valore aggiornato
  + **Priorità:**
    - Importanza: Alta
    - Complessità: Media
* **ID:** RF05
  + **Nome Servizio:** Controllo dei vincoli in tempo reale
  + **Descrizione Dettagliata:** Il sistema deve fornire all’utente la possibilità di abilitare/disabilitare un controllo automatico dei vincoli KenKen durante la fase di inserimento dei numeri (vedi RF04). Questa opzione deve essere controllabile tramite un elemento dell’UI. Quando è attiva la funzione deve verificare immediatamente, se il valore che l’utente inserisce o modifica, viola qualche vincolo (l’unicità dello stesso numero in riga e in colonna, se inserendo il numero la gabbia viene completata e il risultato dei numeri inseriti corrisponda al target assegnato), in caso di violazione di almeno uno di questi vincoli, il sistema deve fornire un feedback visivo immediato all’utente, senza impedire di continuare ma provare. Mentre se questa opzione non è attiva nessuno di questi controlli viene effettuati, ma deve essere cura dell’utente la responsabilità del soddisfacimento dei vincoli.
  + **Soluzione Concettuale:** Serve una variabile booleana che memorizzi se la modalità di controllo è attiva o meno. Deve essere implementata una logica di validazione che, prendendo in input lo stato attuale della griglia e la posizione della cella modificata, esegua i controlli di unicità su riga e colonna, e il controllo sulla gabbia. Questa logica di validazione deve essere invocata dopo ogni inserimento valido, ma solo se la modalità di controllo è attiva. Serve inoltre un meccanismo nell'interfaccia utente per applicare il feedback visivo richiesto in caso di violazione
  + **Priorità:** 
    - Importanza: Media
    - Complessità: Media
* **ID:** RF06
  + **Nome Servizio:** Calcolo e Visualizzazione Soluzione Puzzle
  + **Descrizione Dettagliata:** Il sistema deve mettere a disposizione dell’utente una funzione per calcolare automaticamente almeno una soluzione valida per il puzzle KenKen corrente. Questa funzione viene attivata su richiesta esplicita dell’utente. Il sistema dovrà implementare un algoritmo di risoluzione in grado di trovare le configurazioni complete della griglia che soddisfino simultaneamente tutti i vincoli del gioco, ossia unicità dei numeri per ogni riga e colonna, e rispetto del risultato e operatore specificato per ogni gabbia. L’algoritmo deve essere capace di trovare soluzioni multiple, se il puzzle ne ammette più di una. Per gestire questo e per limitare i tempi di calcolo si trovano le prime X possibili soluzioni esistenti scelte dall’utente tramite UI. L’algoritmo cercherà al massimo le prime X soluzioni valide esistenti e quelle trovate dovranno essere memorizzate. Se l’utente non inserirà nessun valore nel calcolo delle soluzioni il sistema di default calcolerà solo 3 soluzioni, ma allo stesso tempo se l’utente inserirà un numero maggiore di 100 il sistema non permetterà l’inserimento. Al termine del calcolo delle soluzioni l’utente visualizzerà un messaggio che informa il numero delle soluzioni trovate, e la possibilità di visualizzarle e navigare tra di esse.
  + **Soluzione Concettuale:** è richiesto un algoritmo risolutore per problemi e soddisfacimento dei vincoli, con la tecnica del backtracking ricorsivo.L’algoritmo dovrà esplorare lo spazio delle possibili assegnazioni numeriche alle celle vuote, verificando ad ogni passo la coerenza dei vincoli e tornando indietro in caso di violazione. Per trovare le soluzioni l’algoritmo si fermerà solo quando non trova più soluzioni possibili, o altrimenti quando avrà raggiunto il limite di soluzioni scelte dall’utente. Per la visualizzazione servirà una logica che gestisca l’indice della soluzione correntemente mostrata e aggiorni la griglia nell’interfaccia utente in base alla soluzione selezionata tramite dei comandi di navigazione. La logica deve gestire l’abilitazione dei pulsanti successivo/precedente in base alla soluzione mostrata.
  + **Priorità:**
    - Importanza: Alta
    - Complessità: Alta
* **ID:** RF07
  + **Nome Servizio:** Caricamento Partita Salvata
  + **Descrizione Dettagliata:** Il sistema deve consentire all’utente di caricare uno stato di partita precedentemente salvato (vedi RF03), permettendo all’utente di riprendere il gioco esattamente dal punto in cui era stato interrotto. È possibile attivare questa operazione tramite l’UI. Il sistema deve tentare di leggere i dati di salvataggio dal repository designato. Devono essere gestiti anche i casi in cui il salvataggio non esista o sia corrotto, informando l’utente con un messaggio appropriato. Se i dati vengono reperiti correttamente il sistema dovrà deserializzarli e ripristinare la partita, questo include la ricostruzione della definizione del puzzle così com’era al momento del salvataggio, ossia la dimensione, le gabbie, i target, operatori, i numeri fissi iniziali e il ripristino dei numeri inseriti dall’utente nelle celle non fisse. Infine, l'interfaccia utente deve essere aggiornata per visualizzare fedelmente lo stato della partita caricata
  + **Soluzione Concettuale:** quando l’utente avvia la richiesta di caricamento, la prima fase consisterà nel leggere i dati grezzi dal supporto di memorizzazione persistente. È necessaria una fase di validazione e deserializzazione, in cui i dati vengono convertiti dal formato di salvataggio alla rappresentazione interna dello stato di gioco, questo processo deve gestire i potenziali errori. La logica applicativa usa quindi questi dati per ricostruire lo stato interno della partita. Infine, è necessario notificare o comandare all'interfaccia utente di aggiornarsi per riflettere completamente lo stato caricato.
  + **Priorità:**
    - Importanza: Alta
    - Complessità: Media
* **ID:** RF08
  + **Nome Servizio:** Avvio Nuova Partita
  + **Descrizione Dettagliata:** Il sistema deve fornire all’utente un modo per iniziare una nuova partita, questa funzionalità deve essere attivata tramite un comando esplicito. Se l’utente ha una partita in corso dovrà essere avvisato se desidera effettuare un salvataggio oppure no. Una volta effettuata questa operazione si dovrà procedere ad avviare una nuova partita, seguendo in sequenza i passi dettagliati nei requisiti RF01 ed RF02. Al termine della generazione il nuovo puzzle viene visualizzato nell’UI, pronto per essere risolto dal giocatore, interagendoci come spiegato in RF04 e RF05.
  + **Soluzione Concettuale:** è necessario un punto di attivazione dall’UI, questo comando dovrebbe prima verificare se ci sono modifiche non salvate nella partita corrente e, in caso, chiedere all'utente come procedere, poi deve invocare la logica di RF01 per permettere all’utente di inserire i parametri desiderati, una volta ottenuti i parametri si passa alla logica di RF02, dopo di che si passa ad abilitare le funzioni di gioco come da RF04
  + **Priorità:** 
    - Importanza: Alta
    - Complessità: Media
* **ID:** RF09
  + **Nome Servizio:** Gestione Fine Partita
  + **Descrizione Dettagliata:** Una volta riempita la griglia il sistema deve verificare automaticamente la correttezza del puzzle. L’attivazione di questa verifica è l’inserimento di un numero nell’ultima cella vuota, una volta attivo verificherà che la configurazione inserita dall’utente soddisfi i vincoli contemporaneamente, quindi l’unicità della riga e colonna e il vincolo della gabbia. In base all’esito della verifica il sistema notificherà all’utente un messaggio positivo se il puzzle è corretto, altrimenti un messaggio negativo se il puzzle risulta non coerente con i vincoli.In caso di successo, l'interazione con la griglia risolta potrebbe essere disabilitata e/o potrebbe essere proposta una nuova partita (RF08). In caso di errore, l'utente potrà continuare a modificare la griglia (RF04).
  + **Soluzione Concettuale:** è necessario un meccanismo che rilevi in automatico l’inserimento di un numero nell’ultima cella vuota e che verifichi la correttezza del puzzle. Tale funzione deve implementare la logica per iterare su tutte le righe, colonne e gabbie della griglia, verificando il rispetto di tutti i vincoli KenKen. La funzione restituirà un risultato indicante la correttezza o meno della soluzione e, opzionalmente, informazioni sugli errori specifici riscontrati. In base al risultato della funzione di validazione, una logica di presentazione mostrerà il feedback appropriato all'utente attraverso l'interfaccia (messaggio di successo o errore). Se è prevista l'evidenziazione degli errori, questa logica dovrà mappare gli errori identificati dalla funzione di validazione agli specifici elementi della UI da modificare visivamente. Dovrà anche gestire lo stato dell'applicazione post-successo, come la proposta di una nuova partita e bloccare l’input sul puzzle appena risolto, o permettere la continuazione del gioco in caso di errore.
  + **Priorità:**
    - Importanza: Alta
    - Complessità: Media
* **ID:** RF10
  + **Nome Servizio:** Reset Griglia
  + **Descrizione Dettagliata:** Il sistema deve fornire all'utente un comando per resettare lo stato del puzzle corrente, cancellando tutti i numeri inseriti dal giocatore fino a quel momento. Questo permette all'utente di ricominciare il tentativo di soluzione dello stesso puzzle senza doverne generare o caricare uno nuovo. Questa funzione viene attivata tramite un comando esplicito dell'utente. Alla ricezione di questo comando si deve identificare ogni cella in cui l’utente abbia inserito un numero (escludendo ovviamente le celle i numeri fissi inziali), rimuovere il valore inserito riportando la cella allo stato di vuoto, aggiornando l’UI. I numeri fissi iniziali, essendo parte della definizione del puzzle, devono rimanere inalterati
  + **Soluzione Concettuale:** è necessario un comando attivabile dall'utente nell'interfaccia. La logica associata a questo comando deve accedere allo stato interno della griglia. Per ogni cella, deve verificare se essa contiene un numero inserito dall'utente, e resettarlo al suo stato iniziale. Infine, deve essere garantito l'aggiornamento dell'interfaccia utente per riflettere lo stato resettato
  + **Priorità:**
    - Importanza: Media
    - Complessità: Bassa

***A.2 Requisiti non Funzionali***

* **ID:** RNF-US-01
  + **Nome Requisito:** Intuitività Interfaccia Utente
  + **Descrizione Dettagliata:** L’interfaccia utente deve essere progettata in modo da risultare intuitiva e facile da apprendere per le funzioni principali, come l’avvio della partita (vedi RF08), l’inserimento dei numeri (vedi RF04) o anche la richiesta della soluzione (vedi RF06), anche per gli utenti che non conoscono il gioco KenKen.
* **ID:** RNF-US-02
  + **Nome Requisito:** Chiarezza Feedback Utente
  + **Descrizione Dettagliata:** Il feedback fornito dal sistema all’utente, come i messaggi di errore o di successo (vedi RF07 e RF09) o l’evidenziazione dei vincoli violati (vedi RF05), deve essere tempestivo, non ambiguo e facilmente visibile all’interno dell’interfaccia.
* **ID:** RNF-PE-01
  + **Nome Requisito:** Prestazioni Calcolo Soluzione
  + **Descrizione Dettagliata:** Il tempo richiesto per il calcolo delle prime 5 soluzioni (vedi RF06) per una griglia di dimensione massima (6x6) non deve superare i 15 secondi.
* **ID:** RNF-PE-02
  + **Nome Requisito:** Reattività Interfaccia Utente
  + **Descrizione Dettagliata:** L’interfaccia utente deve rispondere alle azioni comuni dell’utente, come la selezione di una cella o l’inserimento di un numero (vedi RF04) senza ritardi percepibili, entro 0.5 secondi.
* **ID:** RNF-AF-01
  + **Nome Requisito:** Correttezza Algoritmi
  + **Descrizione Dettagliata:** Gli algoritmi implementati per la generazione dei puzzle (vedi RF02), il calcolo delle soluzioni (vedi RF06) e la verifica della soluzione finale (vedi RF09) devono essere corretti rispetto alle regole matematiche e logiche del KenKen.
* **ID:** RNF-AF-02
  + **Nome Requisito:** Affidabilità Persistenza Dati
  + **Descrizione Dettagliata:** Il salvataggio (vedi RF03) e il caricamento dello stato della partita (vedi RF07) non devono introdurre corruzione o perdita dei dati dell’utente
* **ID:** NFR-MA-01
  + **Nome Requisito:** Adozione Design Pattern
  + **Descrizione Dettagliata:** Lo sviluppo del software deve impiegare Design Pattern riconosciuti e appropriati al contesto, per migliorare la struttura, la manutenibilità, la leggibilità e la modificabilità del codice, come esplicitamente richiesto dalla specifica del progetto. Le scelte dei pattern principali dovranno essere motivate nella Sezione E.
* **ID:** NFR-MA-02
  + **Nome Requisito:** Documentazione con UML
  + **Descrizione Dettagliata:** Le fasi rilevanti del processo di sviluppo (analisi, architettura, design) devono essere documentate tramite diagrammi UML standard ove necessario, al fine di facilitare la comprensione del progetto e la sua manutenzione futura.
* **ID:** NFR-TE-01
  + **Nome Requisito:** Test Unitari con JUnit
  + **Descrizione Dettagliata:** Moduli software significativi o critici, in particolare quelli contenenti la logica di business (es. generazione puzzle RF02, validazione vincoli RF05/RF09, calcolo soluzione RF06), devono essere accompagnati da test unitari automatici sviluppati utilizzando il framework JUnit, per verificarne la correttezza e supportare modifiche future.
* **ID:** NFR-VI-01
  + **Nome Requisito:** Tecnologia Interfaccia Utente
  + **Descrizione Dettagliata:** L'interfaccia utente grafica (GUI) dell'applicazione sarà implementata utilizzando la tecnologia Java Swing.
* **ID:** NFR-VI-02
  + **Nome Requisito:** Tecnologia di Persistenza
  + **Descrizione Dettagliata:** La persistenza dei dati relativi allo stato della partita (RF03/RF07) sarà realizzata utilizzando un database MySQL.

***A.3 Scenari d’uso dettagliati***

*Descrivere gli scenari più comuni, più interessanti, o più complicati d’uso dei vostri servizi.*

**Utente 🡪 U, Sistema 🡪 S.**

* Scenario 1: Avvio Nuova Partita

1. **U:** Avvia l’applicazione Puzzle KenKen Solver
2. **S:** Visualizza home applicazione
3. **U:** Seleziona il comando “New Game” (RF08)
4. **S:** Presenta le opzioni per i parametri (RF01)
5. **U:** Selezione dimensione “4x4” e modalità “Easy”.Conferma selezione
6. **S:** Esegue la generazione del puzzle (RF02) con i parametri scelti. Visualizza la nuova griglia 4x4 con le gabbie e i vincoli (+ e -), pronta per l’inserimento dei numeri.
7. **U:** Selezione una cella modificabile (RF04), e inserisce il numero “2”
8. **S:** Visualizza “2” nella ella selezionata. Se il controllo dei vincoli è attivo (RF05), verifica che i vinicoli siano rispettati e fornisce eventuale feedback

* Scenario 2: Risoluzione Partita e Navigazione Soluzione

1. **U:** Ha caricato un puzzle di dimensione 5x5 in modalità “Hard”, con alcuni numeri già inseriti. È rimasto bloccato, non riesce a proseguire con il gioco. Quindi seleziona il comando “Show Solution” (RF06), specificando 7 soluzioni.
2. **S­:** Avvia l’algoritmo di calcolo (RF06). Trova solo 3 soluzioni ammissibili, le memorizza. Visualizza un messaggio “Find 3 solutions”. Mostra la prima soluzione completa sulla griglia. Attiva il pulsante di navigazione “Next”
3. **U:** Clicca sul pulsante “Next”
4. **S:** Aggiorna la griglia visualizzando la soluzione trovata. Aggiorna lo stato dei pulsanti di navigazione attivando sia “Next” che “Previous”
5. **U:** Clicca nuovamente su “Next”
6. **S:** Mostra la terza e ultima soluzione, disabilitando il pulsante “Next”

***A.4 Excluded Requirements***

*Descrivere i servizi eventualmente i esclusi, e spiegare il perchè*

1. **Generazione Puzzle Avanzata:** 
   * Descrizione: l’algoritmo di generazione (RF02) non garantirà proprietà avanzate come l’esistenza di una soluzione unica per ogni puzzle generato, ne fornirà livelli di difficoltà calibrati in modo fine.
   * Motivazione: implementare algoritmi che garantiscano soluzione unica è difficoltà precisa è significativamente più complesso
2. **Funzionalità di Gioco Avanzate:** 
   * Descrizione: Non saranno implementate funzionalità come le “Pencil Marks”, ossia la funzionalità di annotare numeri nelle celle, o funzioni come “Undo/Redo” per le mosse
   * Motivazione: Aggiungono complessità all’interfaccia utente e alla gestione dello stato
3. **Caricamento di Puzzle da Fonti Esterne:** 
   * Descrizione: L’applicazione permetterà solo di caricare lo stato di partite salvate internamente (RF07). Non sarà possibile caricare definizioni di puzzle KenKen da file esterni, ad esempio scaricati da internet o creati da altri programmi.
   * Motivazione: la gestione di formati esterni e relativa validazione complicherebbero lo sviluppo, l’applicazione si basa sui puzzle generati internamente.
4. **Funzionalità UI aggiuntive:**
   * Descrizione: Non sono previste funzionalità di personalizzazione dell’aspetto grafico, come i temi o i colori.
   * Motivazione: Considerate come abbellimenti non sono necessari al funzionamento dell’applicazione
5. **Editor Manuale di Puzzle:**
   * Descrizione: Non sarà implementata una modalità “Editor” che permetta all’utente di creare manualmente i propri puzzle KenKen disegnando gabbie o specificando numeri/operatori.
   * Motivazione: Il requisito “Specificare la configurazione” è stato interpretato e soddisfatto tramite la selezione dei parametri (RF01) e la generazione automatica (RF02), per focalizzare il progetto sul gioco e risoluzione, piuttosto che sulla creazione manuale, ritenuta troppo complessa

***A.5 Assunzioni***

*<Briefly document, in this section, the most relevant requirement assumptions/decisions you had to made during your project>*

Le assunzioni e decisioni più rilevanti prese durante il raffinamento dei requisiti sono:

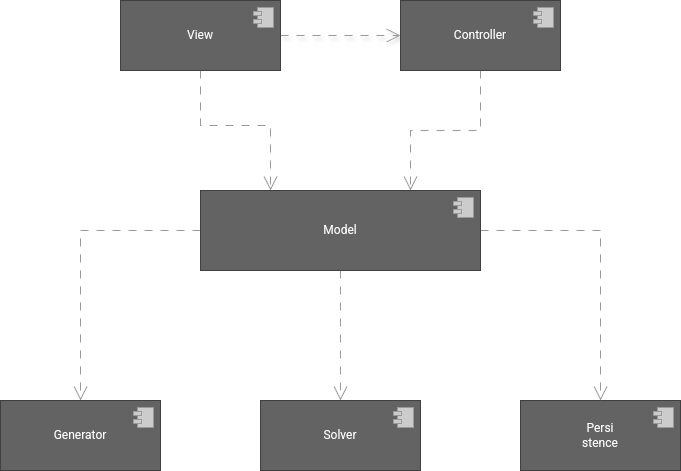
* Si assume che l'utente conosca le regole base del gioco KenKen.
* Il requisito di "specificare la configurazione" è stato interpretato come selezione di parametri (RF01) per la generazione automatica (RF02), escludendo un editor manuale.
* Il requisito di "salvare/caricare” è stato interpretato come salvataggio/caricamento dello stato completo della partita (RF03/RF07).
* L'algoritmo di generazione (RF02) non garantisce l'unicità della soluzione.
* L'utente può richiedere al massimo 100 soluzioni (RF06), con un default di 3, per motivi di performance.
* La tecnologia GUI scelta è Java Swing
* La persistenza (RF03/RF07) avverrà su Data Base MySQL.
* La dimensione della griglia (RF01) è limitata a N compreso tra 3 e 6

***.6 Use Case Diagrams***

Immagine che contiene testo, schermata, diagramma, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.C. Architettura Software   
*<IF RELEVANT, Report here both the static and the dynamic view of your system design, in terms of a Component Diagram, and their related Sequence Diagrams >*

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

**

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

Salvataggio Partita:

Immagine che contiene schermata, design

Il contenuto generato dall'IA potrebbe non essere corretto.

Avvio Nuova Partita:

Immagine che contiene schermata, diagramma, linea, design

Il contenuto generato dall'IA potrebbe non essere corretto.

Calcolo Soluzione e Navigazione:

Immagine che contiene schermata, design

Il contenuto generato dall'IA potrebbe non essere corretto.

D. Dati e loro modellazione (se il sistema si interfaccia con un DBMS)

*Definite le sorgenti di dati a voi necessarie per realizzare I servizi di cui sopra. Modellate tali dati tramite un ER o similari. Specificate se e quali di tali dati sono gia’ forniti da applicativi esistenti.*

*Immagine che contiene testo, Carattere, numero, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.*

Questa sezione descrive il modello dati progettato per supportare la persistenza dello stato dell'applicazione KenKen Solver, in particolare per soddisfare i requisiti funzionali di salvataggio (RF03) e caricamento (RF07) delle partite.

**Sorgente dei Dati:** Come specificato nel requisito non funzionale NFR-VI-02, la persistenza dei dati relativi allo stato della partita viene realizzata utilizzando un database relazionale **MySQL**. Tutti i dati necessari per il salvataggio e il caricamento sono gestiti internamente dall'applicazione; non vengono utilizzati dati provenienti da applicativi esterni preesistenti.

**Modello Entità-Relazione (E-R):** Per strutturare le informazioni da memorizzare in modo efficiente e coerente nel database MySQL, è stato definito il modello Entità-Relazione (E-R) illustrato nella all’inizio del paragrafo. Il modello si basa su tre entità principali: *savegames*, *cages* e *cell*.

Di seguito viene fornita una descrizione dettagliata di ciascuna entità e dei relativi attributi:

1. **savegames**:
   * **Scopo:** Rappresenta una singola sessione di gioco salvata dall'utente.
   * **Attributi Principali:**
     + idSaveGame (INT, Primary Key): Identificatore univoco per ogni partita salvata.
     + saveName (VARCHAR): Nome assegnato dall'utente al salvataggio per facilitarne il riconoscimento.
     + saveTimestamp (DATETIME): Data e ora in cui è stato effettuato il salvataggio.
     + puzzleSize (INT): Dimensione N della griglia (es. 3 per 3x3, 6 per 6x6).
     + difficulty (VARCHAR): Modalità di gioco selezionata (es. 'Easy', 'Medium', 'Hard'), che determina gli operatori utilizzati.
2. **cages**:
   * **Scopo:** Definisce le caratteristiche di ogni "gabbia" (cage) che compone il puzzle di una specifica partita salvata.
   * **Attributi Principali:**
     + idCages (INT, Primary Key): Identificatore univoco per ogni gabbia all'interno di una partita
     + idSaveGame (INT, Foreign Key, Parte della Primary Key): Riferimento alla partita salvata (savegames) a cui questa gabbia appartiene.
     + targetValue (INT): Il valore numerico target che deve essere ottenuto dalle cifre nella gabbia.
     + operator (ENUM): L'operatore aritmetico (+, -, x, ÷) associato alla gabbia.
   * **Relazione:** Esiste una relazione uno-a-molti da savegames a cages (una partita salvata contiene molte gabbie).
3. **cell**:
   * **Scopo:** Memorizza lo stato di ogni singola cella della griglia per una specifica partita salvata.
   * **Attributi Principali:**
     + idSaveGame (INT, Foreign Key, Parte della Primary Key): Riferimento alla partita salvata (savegames).
     + row (INT, Parte della Primary Key): Indice di riga della cella (da 0 a N-1).
     + col (INT, Parte della Primary Key): Indice di colonna della cella (da 0 a N-1).
     + idCages (INT, Foreign Key): Riferimento alla gabbia (cages) a cui questa cella appartiene.
     + value (INT): Il valore numerico presente nella cella (può essere NULL o 0 se la cella è vuota).
     + isEditable (ENUM('0', '1')): Flag booleano che indica se la cella è modificabile dall'utente ('1' = modificabile/vuota o con valore utente) o se contiene un valore fisso iniziale del puzzle ('0' = non modificabile). Questo è cruciale per distinguere i numeri preimpostati da quelli inseriti dall'utente, come richiesto per le funzionalità di caricamento e reset (RF10).
   * **Chiave Primaria:** La chiave primaria di questa tabella è composta dalla terna (idSaveGame, row, col), garantendo l'unicità di ogni cella all'interno di una specifica partita salvata.
   * **Relazioni:** Esiste una relazione molti-a-uno da cell a cages (molte celle appartengono a una gabbia) e una relazione molti-a-uno da cell a savegames (molte celle appartengono a una partita salvata).

Questo modello dati permette di salvare in modo completo e recuperare fedelmente lo stato di qualsiasi partita KenKen generata o giocata all'interno dell'applicazione.

E. Scelte Progettuali (Design Decisions)   
<Document here the **5** most important design decisions you had to take. You can use both a textual or a diagrammatic specification.>

**E.1 Adozione del Pattern Architetturale Model-View-Controller (MVC)**

Per conferire una struttura organizzata e manutenibile all'applicazione, si è scelto di adottare un'architettura basata sul pattern **Model-View-Controller (MVC)**. Questa decisione nasce dall'esigenza fondamentale di separare chiaramente le responsabilità: la gestione dei dati e della logica di business del gioco KenKen (il **Model**), la presentazione dell'interfaccia utente grafica realizzata in Swing (NFR-VI-01) (la **View**), e il coordinamento dell'interazione utente (il **Controller**). Sebbene un'alternativa potesse essere una struttura più monolitica, questa è stata scartata per le inevitabili difficoltà di manutenibilità e testabilità che comporterebbe. L'adozione di MVC, invece, favorisce un elevato disaccoppiamento tra i componenti. Questo permette, ad esempio, di testare la logica del Model indipendentemente dalla UI (supportando NFR-TE-01) e di modificare l'interfaccia senza impattare direttamente le regole del gioco, migliorando così la manutenibilità generale (NFR-MA-01) e la flessibilità del sistema. Come conseguenza diretta, questa scelta architetturale richiede l'adozione di meccanismi di comunicazione ben definiti e disaccoppiati tra i componenti, come il pattern Observer (vedi E.3), per sincronizzare lo stato tra Model e View.

**E.2 Utilizzo del Pattern Memento per Salvataggio/Caricamento**

La gestione delle funzionalità di salvataggio (RF03) e caricamento (RF07) dello stato di una partita è stata affidata al **pattern Memento**. Il problema principale era poter "fotografare" lo stato interno completo del gioco (configurazione del puzzle, valori inseriti dall'utente, ecc.) per memorizzarlo su MySQL (NFR-VI-02) e poterlo ripristinare fedelmente in seguito, il tutto senza violare l'incapsulamento dell'oggetto Model che detiene tale stato. Un'alternativa come la serializzazione diretta dell'intero oggetto Model è stata scartata perché avrebbe reso la struttura interna del Model dipendente dal formato di salvataggio e avrebbe esposto dettagli implementativi al componente di persistenza. Il pattern Memento risolve elegantemente questo problema: il Model crea un oggetto *Memento* contenente i dati necessari al ripristino, che viene poi gestito dal componente di persistenza come una "scatola nera", senza necessità di comprenderne la struttura interna. Questo approccio preserva l'incapsulamento del Model, semplifica la logica di salvataggio/ripristino e contribuisce all'affidabilità della persistenza dei dati (RNF-AF-02). L'implementazione di questo pattern richiede la definizione precisa della struttura dati del Memento e la logica per mapparla sul database relazionale.

**E.3 Impiego del Pattern Observer per Aggiornamenti UI**

Per garantire che l'interfaccia utente sia sempre sincronizzata con lo stato corrente del gioco in modo efficiente e disaccoppiato, è stato adottato il **pattern Observer**. L'esigenza nasce dal fatto che numerose azioni (inserimento di un numero da parte dell’utente, validazione automatica dei vincoli, caricamento di una partita salvata, reset della griglia) modificano lo stato del Model e tali modifiche devono riflettersi immediatamente sulla View. Un approccio alternativo basato su un accoppiamento diretto, dove il Model chiama esplicitamente metodi della View, è stato evitato poiché creerebbe dipendenze rigide, in contrasto con i principi dell'architettura MVC scelta (E.1). Il pattern Observer permette invece al Model (*Subject*) di notificare genericamente tutti gli oggetti interessati (*Observer*, tra cui la View) ogni volta che il suo stato cambia significativamente. Gli Observer, ricevuta la notifica, possono quindi richiedere al Model i dati aggiornati e adeguare la propria rappresentazione. Questa scelta garantisce il disaccoppiamento, migliora la reattività dell'interfaccia (RNF-PE-02) e la chiarezza del feedback (RNF-US-02), ed è cruciale per l'aggiornamento dinamico richiesto da molti requisiti. L'implementazione richiede che Model e View implementino rispettivamente le interfacce Subject e Observer.

**E.4 Scelta dell'Algoritmo di Backtracking per il Risolutore**

La funzionalità di calcolo delle soluzioni del puzzle KenKen è implementata tramite l'algoritmo di **Backtracking ricorsivo**. Il compito consiste nel trovare assegnazioni complete di numeri alla griglia che soddisfino tutti i vincoli del gioco (unicità su righe/colonne e regole delle gabbie), configurandosi come un classico Problema di Soddisfacimento di Vincoli. L'alternativa più semplice, la forza bruta (testare tutte le combinazioni), è computazionalmente insostenibile anche per le dimensioni modeste della griglia (fino a 6x6). Il Backtracking rappresenta invece una tecnica standard ed efficace per questo tipo di problemi: esplora sistematicamente lo spazio delle soluzioni parziali, assegnando valori alle celle vuote e verificando la coerenza con i vincoli ad ogni passo; se un'assegnazione viola un vincolo, l'algoritmo "torna indietro" (backtrack) e prova un valore differente, potando così rami dello spazio di ricerca che non portano a soluzioni valide. Questo approccio garantisce la completezza (trova tutte le soluzioni se esistono), soddisfa il requisito di correttezza (RNF-AF-01) e permette di implementare facilmente la richiesta di limitare il numero di soluzioni restituite (RF06). Una conseguenza di questa scelta è che le prestazioni, pur accettabili per le dimensioni N<=6 (RNF-PE-01), dipendono dall'efficienza della verifica dei vincoli.

**E.5 Utilizzo di MySQL per la Persistenza dei Dati**

In linea con il requisito non funzionale NFR-VI-02, la persistenza dei dati relativi alle partite salvate (necessaria per RF03 e RF07) è realizzata tramite un database relazionale **MySQL**. Si è reso necessario scegliere un meccanismo affidabile e strutturato per memorizzare e recuperare i dati di gioco tra diverse sessioni. L'alternativa principale considerata è stata il salvataggio diretto su file system (es. file di testo, XML, o serializzazione di oggetti). Tuttavia, questa opzione è stata ritenuta meno robusta per quanto riguarda l'integrità dei dati e la gestione degli errori, oltre che meno adatta a rappresentare la natura intrinsecamente relazionale dei dati, come modellato nello schema E-R (Sezione D). La scelta di MySQL, un RDBMS (Relation DataBase Management System) maturo e ampiamente utilizzato, offre vantaggi significativi in termini di affidabilità (supporto a transazioni ACID, vincoli di integrità), strutturazione dei dati tramite SQL e separazione fisica dei dati dall'applicazione, contribuendo così a soddisfare il requisito di affidabilità RNF-AF-02. L'adozione di MySQL implica la necessità di gestire la connessione al database tramite JDBC e di implementare la logica di mappatura tra gli oggetti e le tabelle definite nello schema relazionale

F. Progettazione di Basso Livello

**F.1 Componente Model**

Questo componente è il cuore della logica applicativa e gestisce lo stato del gioco KenKen.

* **Classe GameModel**
  + **Responsabilità:** Gestire lo stato corrente della partita, le regole del gioco, la validazione e fornire i meccanismi per il Memento e l'Observer.
  + **Attributi Chiave:**
    - Grid grid: L'oggetto che rappresenta la griglia di gioco.
    - List<Cage> cages: La lista delle gabbie definite per il puzzle corrente.
    - int N: Dimensione della griglia (es. 4 per 4x4).
    - GameDifficulty difficulty: Difficoltà del gioco.
    - boolean validationEnabled: Flag per il controllo automatico dei vincoli.
    - (privato) List<GameObserver> observers: Lista degli observer registrati.
  + **Metodi Chiave:**
    - public void startGame(int N, GameDifficulty difficulty, PuzzleConfiguration config): Inizializza una nuova partita.
    - public void placeNumber(int row, int col, int value): Inserisce un numero nella cella, notifica gli observer.
    - public int getValue(int row, int col): Ottiene il valore di una cella.
    - public void clearCell(int row, int col): Cancella il valore di una cella utente, notifica gli observer.
    - public boolean validateCell(int row, int col, int value): Controlla se un valore è valido per una cella (unicità riga/colonna).
    - public boolean checkCageConstraint(Cage cage): Verifica se il vincolo di una specifica gabbia è soddisfatto.
    - public boolean isGameSolved(): Verifica se l'intera griglia è risolta correttamente.
    - public void resetGameToInitial(): Riporta la griglia allo stato iniziale del puzzle (RF10), notificando gli observer.
    - public GameStateMemento createMemento(): Crea un Memento dello stato corrente.
    - public void restoreFromMemento(GameStateMemento memento): Ripristina lo stato da un Memento, notifica gli observer.
    - public void attachObserver(GameObserver observer): Aggiunge un observer.
    - public void detachObserver(GameObserver observer): Rimuove un observer.
    - public void notifyObservers(): Notifica tutti gli observer di un cambiamento.
    - public Grid getGrid(): Restituisce l'oggetto griglia.
    - public List<Cage> getCages(): Restituisce la lista delle gabbie.
* **Classe Grid**
  + **Responsabilità:** Rappresentare la struttura della griglia N x N.
  + **Attributi Chiave:**
    - Cell[][] cells: Matrice di oggetti Cell.
    - int N: Dimensione.
  + **Metodi Chiave:**
    - public Cell getCell(int row, int col): Restituisce una cella specifica.
    - public void setCell(int row, int col, Cell cell): Imposta una cella (usato in fase di inizializzazione).
* **Classe Cell**
  + **Responsabilità:** Rappresentare una singola cella della griglia.
  + **Attributi Chiave:**
    - int row, int col: Coordinate.
    - int value: Valore numerico (0 o null se vuota).
    - boolean isEditable: true se modificabile dall'utente, false se è un numero fisso del puzzle.
    - Cage parentCage: Riferimento alla gabbia di appartenenza.
  + **Metodi Chiave:**
    - public int getValue(), public void setValue(int value)
    - public boolean isEditable(), public void setEditable(boolean flag)
    - public void clearValue(): Imposta value a 0/null se isEditable.
    - public Cage getParentCage()
* **Classe Cage**
  + **Responsabilità:** Rappresentare una gabbia con il suo vincolo.
  + **Attributi Chiave:**
    - List<Cell> cellsInCage: Lista delle celle che compongono la gabbia.
    - int targetValue: Il risultato numerico del vincolo.
    - OperationType operation: L'operazione aritmetica (+, -, x, ÷).
    - int idCage (opzionale, per riferimento al DB)
  + **Metodi Chiave:**
    - public void addCell(Cell cell)
    - public List<Cell> getCells()
    - public boolean checkConstraint(): Calcola se le celle correnti soddisfano targetValue e operation.
* **Classe GameStateMemento**
  + **Responsabilità:** Contenere lo stato di GameModel da salvare/ripristinare. Non dovrebbe avere logica di business.
  + **Attributi Chiave:**
    - (Privati, accessibili solo da GameModel e dal meccanismo di persistenza): Dati serializzabili della griglia (es. una matrice di valori e flag isEditable), definizione delle gabbie, difficoltà, ecc. Ad esempio: int[][] cellValues, boolean[][] cellEditabilityFlags, List<CageDefinition> cageDefinitions.
  + **Metodi Chiave:**
    - Metodi getter per permettere al componente Persistence di leggere i dati da salvare. Non dovrebbe avere setter pubblici per l'esterno.
* **Interfaccia GameObserver**
  + **Responsabilità:** Definire il contratto per gli oggetti che vogliono essere notificati dei cambiamenti in GameModel.
  + **Metodi Chiave:**
    - public void update(GameModel model) (o metodi più specifici tipo updateGrid(), updateValidationStatus(), ecc.)

**F.2 Componente View (GUI Swing)**

Questo componente è responsabile della presentazione dell'interfaccia utente.

* **Classe MainFrame (estende JFrame)**
  + **Responsabilità:** Finestra principale dell'applicazione. Contiene gli altri elementi UI. Implementa GameObserver.
  + **Attributi Chiave:**
    - GridPanel gridPanel: Il pannello che disegna la griglia.
    - ControlPanel controlPanel: Pannello con i pulsanti di controllo.
    - GameController gameController: Riferimento al controller.
    - GameModel gameModel: Riferimento al modello (per leggere lo stato durante l'aggiornamento).
  + **Metodi Chiave:**
    - public MainFrame(GameController controller, GameModel model): Costruttore.
    - private void initUI(): Inizializza e dispone i componenti Swing.
    - public void update(GameModel model): Metodo dell'Observer; riceve notifiche dal GameModel e aggiorna la GridPanel e altri elementi della UI.
    - public void showErrorMessage(String message): Mostra un messaggio di errore.
    - public void showValidationResult(boolean isValid, String message): Mostra il risultato della validazione.
    - public String askForSaveGameName(): Chiede all'utente il nome per un salvataggio.
    - public String chooseGameToLoad(List<String> gameNames): Permette di scegliere una partita da caricare.
* **Classe GridPanel (estende JPanel)**
  + **Responsabilità:** Disegnare la griglia, le celle, i numeri, le linee delle gabbie e gestire l'input utente sulla griglia.
  + **Attributi Chiave:**
    - GameModel gameModel: Per accedere ai dati della griglia da disegnare.
    - GameController gameController: Per notificare l'input utente.
    - int cellSize: Dimensione in pixel di una cella.
  + **Metodi Chiave:**
    - protected void paintComponent(Graphics g): Logica di disegno della griglia, numeri, bordi delle gabbie.
    - private void handleMouseClick(MouseEvent e): Determina su quale cella l'utente ha cliccato e notifica il controller (es. per far apparire un dialog di inserimento numero).
    - public void refreshGrid(): Forza il ridisegno della griglia.
* **Classe ControlPanel (estende JPanel)**
  + **Responsabilità:** Contenere i pulsanti (Nuova Partita, Salva, Carica, Risolvi, Reset, ecc.) e altri controlli (selezione difficoltà/dimensione).
  + **Attributi Chiave:**
    - JButton newGameButton, saveButton, loadButton, solveButton, resetButton, etc.
    - GameController gameController: Per associare azioni ai pulsanti.
  + **Metodi Chiave:**
    - Costruttore che inizializza i pulsanti e aggiunge ActionListener che invocano metodi del gameController.

**F.3 Componente Controller**

Questo componente fa da mediatore tra la View e il Model.

* **Classe GameController**
  + **Responsabilità:** Ricevere input dalla View, invocare le azioni appropriate sul Model, sul Solver o sul Persistence, e aggiornare la View se necessario (indirettamente, tramite il Model che notifica la View).
  + **Attributi Chiave:**
    - GameModel gameModel
    - MainFrame mainView (o un'interfaccia più generica se si volesse cambiare UI)
    - PersistenceService persistenceService
    - SolverService solverService
  + **Metodi Chiave (corrispondono alle azioni utente):**
    - public void startNewGame(int N, GameDifficulty difficulty)
    - public void handleCellInput(int row, int col, int value): Chiama gameModel.placeNumber().
    - public void saveCurrentGame(): Chiede il nome alla View, prende il Memento dal Model, lo passa a persistenceService.
    - public void loadGame(): Chiede alla View quale partita caricare, la chiede a persistenceService, la passa a gameModel.restoreFromMemento().
    - public void solveCurrentGame(int maxSolutions): Chiama solverService.findSolutions(), poi aggiorna la View con la prima soluzione o abilita la navigazione.
    - public void navigateToNextSolution(), public void navigateToPreviousSolution()
    - public void resetCurrentGame(): Chiama gameModel.resetGameToInitial().
    - public void toggleValidation(boolean enable): Aggiorna gameModel.validationEnabled.

**F.4 Componente Persistence**

Gestisce il salvataggio e caricamento dei dati di gioco su MySQL.

* **Classe PersistenceService (o GameDAO)**
  + **Responsabilità:** Interagire con il database MySQL per salvare e caricare lo stato delle partite.
  + **Attributi Chiave:**
    - (privato) Dettagli della connessione al DB (URL, utente, password) o un oggetto Connection.
  + **Metodi Chiave:**
    - public boolean saveGame(GameStateMemento memento, String gameName): Converte i dati del Memento in record per le tabelle savegames, cages, cell ed esegue le query SQL INSERT/UPDATE.
    - public GameStateMemento loadGame(String gameId): Esegue query SQL SELECT per recuperare i dati dalle tabelle e ricostruisce un oggetto GameStateMemento.
    - public List<SaveGameInfo> getSavedGamesList(): Recupera l'elenco delle partite salvate (es. id, nome, data) dalla tabella savegames.
    - (privato) Metodi helper per stabilire/chiudere la connessione, eseguire query, mappare ResultSet a oggetti.
* **Classe SaveGameInfo (DTO - Data Transfer Object)**
  + **Responsabilità:** Contenere i metadati di una partita salvata per la visualizzazione all'utente.
  + **Attributi Chiave:**
    - String gameId (es. idSaveGame dal DB)
    - String gameName
    - Date saveDate

**F.5 Componente Solver**

Implementa la logica per risolvere il puzzle KenKen.

* **Classe SolverService (o KenKenSolver)**
  + **Responsabilità:** Implementare l'algoritmo di backtracking per trovare le soluzioni.
  + **Attributi Chiave:**
    - Nessuno stato persistente, è un servizio.
  + **Metodi Chiave:**
    - public List<SolutionGrid> findSolutions(GameModel gameToSolve, int maxSolutions): Metodo principale che inizializza e avvia il processo di backtracking. SolutionGrid potrebbe essere una semplice rappresentazione della griglia risolta (es. int[][]).
    - (privato ricorsivo) boolean backtrack(Grid currentGrid, int row, int col): Il cuore dell'algoritmo.
      * Caso base: se la griglia è piena e valida, soluzione trovata.
      * Passo ricorsivo:
        + Trova la prossima cella vuota.
        + Per ogni numero possibile (1-N):

Se il numero è valido per la cella (non viola unicità riga/colonna):

Piazza il numero.

Se la cella completa una gabbia, verifica il vincolo della gabbia. Se non valido, annulla e prova il prossimo numero.

Se backtrack(grid, nextRow, nextCol) ritorna true, allora ritorna true.

Annulla il piazzamento del numero (backtrack).

* + - * + Se nessun numero funziona, ritorna false.

Questo è un punto di partenza dettagliato. Per ogni classe, pensa a quali informazioni deve contenere (attributi) e cosa deve saper fare (metodi) per adempiere alle sue responsabilità e interagire con le altre. Non aver paura di aggiungere o modificare classi e metodi man mano che la tua comprensione del design si affina. L'importante è che questo schema ti guidi poi nella scrittura del codice vero e proprio.

G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs) *<Report in this section how the architectural and low level design you produced satisfies the FRs and the NFRs>*

**G.1 Soddisfacimento dei Requisiti Funzionali**

Di seguito, per ciascun requisito funzionale (RF), viene illustrato come il progetto software ne garantisce l'adempimento.

* **RF01: Specificare la configurazione del gioco (porre numeri nelle caselle, stabilire i vincoli, dimensione, difficoltà) e avviare una nuova partita.**
  + **Architettura (Sez. C):** Il diagramma dei componenti (C.1) mostra il Controller che media tra la View (da cui l'utente specifica i parametri) e il Model (che gestisce la logica di configurazione). Il diagramma di sequenza "Avvio Nuova Partita" (C.2) illustra questa interazione.
  + **Scelte Progettuali (Sez. E):** L'architettura MVC (E.1) facilita la separazione tra l'input utente per la configurazione (gestito da View e Controller) e la logica di creazione del gioco (Model).
  + **Progettazione di Basso Livello (Sez. F):**
    - La classe GameController (F.3) espone metodi come startNewGame(int N, GameDifficulty difficulty, PuzzleConfiguration config) che ricevono i parametri dalla View (es. ControlPanel, F.2).
    - La classe GameModel (F.1) contiene la logica per inizializzare la griglia (Grid), le celle (Cell) con i loro attributi isEditable e value (per i numeri preimpostati), e le Cage con i loro vincoli, basandosi sulla PuzzleConfiguration fornita.
    - La View (es. MainFrame, GridPanel, F.2) è responsabile di presentare all'utente le opzioni per selezionare dimensione e difficoltà e, potenzialmente, un'interfaccia per definire manualmente un puzzle.
* **RF02: Salvare lo stato della partita corrente (configurazione e progressi dell'utente).**
  + **Architettura (Sez. C):** Il componente Persistence (C.1) è dedicato alla gestione del salvataggio. Il diagramma di sequenza "Salvataggio Partita" (C.2) mostra l'interazione tra Controller, Model (per ottenere lo stato) e Persistence.
  + **Modellazione Dati (Sez. D):** Le tabelle savegames, cages, e cell (con il suo flag isEditable e value) sono progettate per memorizzare tutti gli aspetti di una partita, inclusi i numeri fissi e quelli inseriti dall'utente.
  + **Scelte Progettuali (Sez. E):** L'uso del pattern Memento (E.2) permette al GameModel di creare un GameStateMemento che incapsula il suo stato interno. La scelta di MySQL (E.5) fornisce un repository robusto.
  + **Progettazione di Basso Livello (Sez. F):**
    - Il GameModel (F.1) implementa createMemento().
    - Il GameController (F.3) orchestra l'azione, richiedendo il nome del salvataggio alla View e invocando persistenceService.saveGame().
    - La classe PersistenceService (F.4) riceve il GameStateMemento e si occupa della sua traduzione e scrittura nelle tabelle del database MySQL.
* **RF03: Caricare uno stato della partita precedentemente salvato.**
  + **Architettura (Sez. C):** Similmente a RF02, il componente Persistence (C.1) è centrale. Un diagramma di sequenza per il caricamento (non mostrato esplicitamente ma simmetrico al salvataggio) vedrebbe il Controller richiedere al Persistence i dati, che poi vengono usati per ripristinare il Model.
  + **Modellazione Dati (Sez. D):** Le stesse tabelle (savegames, cages, cell) usate per il salvataggio sono lette per ricostruire lo stato.
  + **Scelte Progettuali (Sez. E):** Il pattern Memento (E.2) è utilizzato anche qui: il PersistenceService recupera i dati per ricostruire un GameStateMemento, che viene poi passato al GameModel per il ripristino. MySQL (E.5) fornisce i dati.
  + **Progettazione di Basso Livello (Sez. F):**
    - La PersistenceService (F.4) implementa loadGame(String gameId) che legge dal DB e crea un GameStateMemento.
    - Il GameController (F.3) gestisce la selezione della partita da caricare (tramite la View) e passa il Memento recuperato al GameModel.
    - Il GameModel (F.1) implementa restoreFromMemento(GameStateMemento memento) e notifica la View (tramite Observer, E.3) dell'avvenuto caricamento.
* **RF04: Consentire a un utente di giocare inserendo i numeri nelle caselle.**
  + **Architettura (Sez. C):** L'interazione utente avviene tramite la View, le azioni sono gestite dal Controller che modifica il Model.
  + **Scelte Progettuali (Sez. E):** L'architettura MVC (E.1) e il pattern Observer (E.3) sono fondamentali. L'Observer assicura che la View si aggiorni quando il Model cambia a seguito dell'inserimento di un numero.
  + **Progettazione di Basso Livello (Sez. F):**
    - La GridPanel (F.2) nella View cattura l'input dell'utente (es. click su una cella e successivo inserimento tramite un dialog o tastiera).
    - Il GameController (F.3) riceve l'input (es. tramite handleCellInput(row, col, value)) e invoca gameModel.placeNumber(row, col, value).
    - La classe Cell (F.1) nel Model ha l'attributo value che viene modificato, e il flag isEditable assicura che solo le celle corrette possano essere modificate.
    - Il GameModel (F.1), dopo aver modificato lo stato, notifica la View tramite notifyObservers().
* **RF05: Abilitare il controllo del soddisfacimento dei vincoli man mano che si inseriscono i numeri (validazione automatica/manuale).**
  + **Architettura (Sez. C):** La logica di validazione risiede nel Model. Il Controller può triggerare la validazione e la View mostrare i risultati.
  + **Scelte Progettuali (Sez. E):** Il pattern Observer (E.3) permette di aggiornare la View con feedback sulla validazione.
  + **Progettazione di Basso Livello (Sez. F):**
    - Il GameModel (F.1) contiene metodi come validateCell(row, col, value) (per unicità riga/colonna) e checkCageConstraint(Cage cage) (per i vincoli delle gabbie). Un metodo validateGrid() può orchestrare la validazione completa.
    - L'attributo validationEnabled nel GameModel permette di attivare/disattivare la validazione automatica dopo ogni mossa.
    - Il GameController (F.3) può invocare questi metodi di validazione dopo un inserimento (se la validazione automatica è attiva) o su richiesta esplicita dell'utente.
    - La MainFrame (F.2) nella View riceve i risultati della validazione (tramite l'aggiornamento dell'Observer o chiamate dirette dal Controller) e li presenta all'utente (es. evidenziando celle errate, mostrando messaggi).
* **RF06: Calcolare la/le soluzione/i del gioco (l'utente può specificare il numero massimo desiderato di soluzioni).**
  + **Architettura (Sez. C):** Il componente Solver (C.1) è interamente dedicato a questa funzionalità. Il diagramma di sequenza "Calcolo Soluzione e Navigazione" (C.2) mostra come il Controller invoca il Solver e gestisce i risultati.
  + **Scelte Progettuali (Sez. E):** La scelta dell'algoritmo di Backtracking (E.4) è la decisione chiave per l'implementazione del risolutore.
  + **Progettazione di Basso Livello (Sez. F):**
    - La classe SolverService (F.5) implementa il metodo findSolutions(GameModel gameToSolve, int maxSolutions) che utilizza l'algoritmo di backtracking.
    - Il metodo ricorsivo privato backtrack(...) all'interno di SolverService esegue la ricerca sistematica delle soluzioni, rispettando i vincoli di unicità e delle gabbie.
    - Il GameController (F.3) riceve la richiesta dalla View, invoca il SolverService e gestisce la lista di soluzioni restituite.
* **RF07: Navigare tra le soluzioni trovate (con tasti tipo next/previous).**
  + **Architettura (Sez. C):** Il Controller gestisce la logica di navigazione, aggiornando la View per mostrare la soluzione corrente basandosi sui dati forniti dal Model (o da una struttura dati delle soluzioni). Il diagramma di sequenza "Calcolo Soluzione e Navigazione" (C.2) copre anche questa parte.
  + **Scelte Progettuali (Sez. E):** L'architettura MVC (E.1) e il pattern Observer (E.3) sono rilevanti per aggiornare la visualizzazione della soluzione corrente.
  + **Progettazione di Basso Livello (Sez. F):**
    - Il GameController (F.3) mantiene traccia della lista di soluzioni ottenute da RF06 e dell'indice della soluzione attualmente visualizzata. Espone metodi come MapsToNextSolution() e MapsToPreviousSolution().
    - Questi metodi aggiornano lo stato del GameModel (o caricano una rappresentazione della soluzione nel Model) affinché la View (tramite Observer) mostri la griglia della soluzione selezionata.
    - La View (MainFrame e ControlPanel, F.2) fornisce i pulsanti "Next" e "Previous" e aggiorna la GridPanel di conseguenza.
* **RF10: Resettare la partita allo stato iniziale del puzzle.**
  + **Architettura (Sez. C):** Il Controller riceve la richiesta dalla View e la inoltra al Model.
  + **Modellazione Dati (Sez. D):** L'attributo isEditable nella tabella cell (e nella classe Cell, F.1) è cruciale per distinguere i numeri preimpostati (da non cancellare) da quelli inseriti dall'utente (da cancellare durante il reset).
  + **Scelte Progettuali (Sez. E):** L'Observer (E.3) notifica alla View di aggiornarsi dopo il reset.
  + **Progettazione di Basso Livello (Sez. F):**
    - Il GameModel (F.1) implementa un metodo resetGameToInitial(). Questo metodo itera sulle celle della Grid e, per ogni Cell dove isEditable è true, imposta value a 0 (o null).
    - Il GameController (F.3) espone un metodo resetCurrentGame() che invoca il metodo del GameModel.
    - La View (ControlPanel, F.2) ha un pulsante "Reset" che attiva questa funzionalità.
    - Dopo il reset, il GameModel notifica gli observer affinché la GridPanel (F.2) venga ridisegnata allo stato iniziale.
* **RF-Display: Visualizzare la griglia di gioco, i numeri, le gabbie e i loro vincoli.**
  + **Architettura (Sez. C):** La View (C.1) è responsabile di tutta la presentazione grafica.
  + **Scelte Progettuali (Sez. E):** L'architettura MVC (E.1) garantisce che la View ottenga i dati dal Model. Il pattern Observer (E.3) assicura che la visualizzazione sia sempre aggiornata.
  + **Progettazione di Basso Livello (Sez. F):**
    - La classe GridPanel (F.2) contiene la logica di disegno (paintComponent) per rappresentare le celle, i loro valori, i bordi delle celle e i bordi più spessi delle gabbie.
    - Le informazioni sui vincoli delle gabbie (operatore e risultato) sono anch'esse disegnate dalla GridPanel, ottenendo i dati dalla lista di oggetti Cage nel GameModel (F.1).
    - La classe Cell (F.1) e Cage (F.1) forniscono i dati necessari per la visualizzazione.

**G.2 Soddisfacimento dei Requisiti Non Funzionali**

Di seguito, per ciascun requisito non funzionale (RNF/NFR), viene illustrato come il progetto software ne garantisce l'adempimento.

* **NFR-VI-01: Interfaccia utente grafica realizzata con tecnologia Java Swing.**
  + **Progettazione di Basso Livello (Sez. F):** L'intera progettazione del componente View (F.2) si basa sull'utilizzo di classi Java Swing come JFrame (MainFrame), JPanel (GridPanel, ControlPanel), JButton, ecc. Questa scelta tecnologica è rispettata a livello di implementazione di tutte le classi di interfaccia utente.
* **NFR-VI-02: Persistenza dei dati realizzata tramite database MySQL.**
  + **Modellazione Dati (Sez. D):** È stato definito uno schema ER specifico per un RDBMS, con tabelle savegames, cages, e cell, compatibile con MySQL.
  + **Scelte Progettuali (Sez. E):** La decisione E.5 motiva esplicitamente la scelta di MySQL.
  + **Progettazione di Basso Livello (Sez. F):** Il componente PersistenceService (F.4) è progettato per interagire con un database relazionale tramite JDBC, specificamente per eseguire query SQL compatibili con MySQL per le operazioni di salvataggio e caricamento.
* **NFR-MA-01: Alta manutenibilità del codice sorgente, anche tramite l'uso di Design Pattern.**
  + **Architettura (Sez. C):** La scomposizione in componenti (View, Controller, Model, Persistence, Solver) con responsabilità ben definite (C.1) riduce l'accoppiamento e migliora la manutenibilità.
  + **Scelte Progettuali (Sez. E):**
    - L'adozione del pattern MVC (E.1) è fondamentale per la separazione delle problematiche, facilitando modifiche isolate.
    - L'uso del pattern Memento (E.2) incapsula la gestione dello stato per il salvataggio/caricamento.
    - L'uso del pattern Observer (E.3) disaccoppia il Model dalla View.
  + **Progettazione di Basso Livello (Sez. F):** La chiara definizione delle interfacce (es. GameObserver) e delle responsabilità di ciascuna classe (F.1-F.5) contribuisce a un codice modulare e più facile da comprendere e modificare.
* **NFR-TE-01: Testabilità dei moduli significativi (con JUnit).**
  + **Architettura (Sez. C):** La separazione dei componenti, in particolare l'isolamento della logica di business nel Model (C.1), facilita il testing unitario.
  + **Scelte Progettuali (Sez. E):** L'architettura MVC (E.1) permette di testare il Model indipendentemente dalla View e dal Controller. Anche il componente Solver può essere testato in isolamento.
  + **Progettazione di Basso Livello (Sez. F):** Le classi come GameModel (F.1), Cell (F.1), Cage (F.1), e SolverService (F.5) sono progettate con interfacce pubbliche chiare che possono essere facilmente istanziate e testate con JUnit, verificando la correttezza della loro logica interna senza dipendenze dalla GUI o dal database (quest'ultimo può essere mockato/stubbed per il PersistenceService).
* **RNF-US-01 (Usabilità): Interfaccia utente intuitiva e facile da usare.**
  + **Architettura (Sez. C):** La View (C.1) è dedicata all'interazione con l'utente.
  + **Progettazione di Basso Livello (Sez. F):**
    - La MainFrame (F.2) organizza logicamente le aree funzionali (griglia, controlli).
    - Il ControlPanel (F.2) raggruppa azioni comuni (Nuova Partita, Salva, Risolvi, ecc.) in modo accessibile.
    - La GridPanel (F.2) fornisce una rappresentazione visiva chiara dello stato del gioco. L'interazione per l'inserimento dei numeri è progettata per essere diretta.
* **RNF-US-02 (Usabilità): Feedback chiaro all'utente sulle sue azioni e sullo stato del gioco.**
  + **Scelte Progettuali (Sez. E):** Il pattern Observer (E.3) è cruciale: ogni modifica rilevante al GameModel (inserimento, validazione, caricamento, soluzione) notifica la View.
  + **Progettazione di Basso Livello (Sez. F):**
    - La MainFrame (F.2) è progettata per mostrare messaggi di stato, errori o conferma.
    - La GridPanel (F.2) aggiorna dinamicamente la visualizzazione dei numeri inseriti e può usare segnali visivi (es. colori) per indicare lo stato di validazione delle celle o delle gabbie, in risposta agli aggiornamenti ricevuti dal GameModel.
* **RNF-AF-01 (Affidabilità): Correttezza degli algoritmi di gioco (validazione, soluzione).**
  + **Scelte Progettuali (Sez. E):** L'algoritmo di Backtracking (E.4) è uno standard per la risoluzione di CSP e, se implementato correttamente, garantisce di trovare tutte le soluzioni valide.
  + **Progettazione di Basso Livello (Sez. F):**
    - La logica di validazione in GameModel (F.1) (metodi validateCell, checkCageConstraint) è progettata per implementare fedelmente le regole del KenKen.
    - L'implementazione del metodo backtrack in SolverService (F.5) deve seguire attentamente i principi dell'algoritmo per assicurare la correttezza delle soluzioni generate.
* **RNF-AF-02 (Affidabilità): Affidabilità della persistenza dei dati (salvataggio/caricamento senza corruzione).**
  + **Modellazione Dati (Sez. D):** Lo schema ER è stato progettato per rappresentare in modo completo e non ambiguo lo stato della partita. L'uso di chiavi primarie e potenziali foreign key (se definite correttamente) aiuta a mantenere l'integrità.
  + **Scelte Progettuali (Sez. E):** L'uso di MySQL (E.5), un RDBMS maturo, fornisce meccanismi intrinseci per la gestione affidabile dei dati (transazioni ACID, se usate correttamente). Il pattern Memento (E.2) assicura che uno stato coerente venga passato per la persistenza.
  + **Progettazione di Basso Livello (Sez. F):** La classe PersistenceService (F.4) deve implementare correttamente la mappatura tra l'oggetto GameStateMemento e lo schema del database, gestendo potenziali errori durante le operazioni di I/O sul DB.
* **RNF-PE-01 (Performance): Tempi di risposta accettabili per il calcolo delle soluzioni (per N<=6).**
  + **Scelte Progettuali (Sez. E):** L'algoritmo di Backtracking (E.4), pur essendo potenzialmente esponenziale nel caso peggiore, è generalmente efficiente per le dimensioni di griglia tipiche del KenKen (fino a 6x6). La possibilità di limitare il numero massimo di soluzioni (RF06) aiuta a controllare il tempo di esecuzione.
  + **Progettazione di Basso Livello (Sez. F):** L'efficienza dell'implementazione del metodo backtrack in SolverService (F.5), in particolare la rapidità dei controlli di validità ad ogni passo, è importante. Non sono previste ottimizzazioni complesse, ma una corretta implementazione è sufficiente per N<=6.
* **RNF-PE-02 (Performance): Reattività dell'interfaccia utente durante il gioco.**
  + **Architettura (Sez. C):** La separazione tra View e Model (C.1) permette alla UI di rimanere reattiva anche se il Model esegue operazioni leggermente più lunghe (anche se per il KenKen le operazioni sul Model sono tipicamente veloci).
  + **Scelte Progettuali (Sez. E):** Il pattern Observer (E.3) garantisce che gli aggiornamenti della View avvengano solo quando necessario e in modo efficiente.
  + **Progettazione di Basso Livello (Sez. F):** Le operazioni di disegno nella GridPanel (F.2) devono essere ottimizzate per evitare lentezza. Le interazioni nel GameController (F.3) e le modifiche nel GameModel (F.1) per le normali azioni di gioco (inserimento numero) sono progettate per essere veloci. Le operazioni potenzialmente lunghe (come il calcolo delle soluzioni o il caricamento/salvataggio di file molto grandi, sebbene qui si usi un DB) dovrebbero idealmente essere eseguite in thread separati per non bloccare l'Event Dispatch Thread di Swing (questo è un dettaglio implementativo più avanzato non esplicitamente richiesto nel LLD finora, ma è una considerazione per la performance della UI).

Appendix. Prototype   
*<Provide a brief report on your prototype, and especially: information on what you have implemented, how the implementation covers the FR and NFR, how the prototypes demonstrates your project correctness with respect to the FR and NFR. You may add some screenshots to describe what required above. Be ready to show your prototype during the oral examination>* 