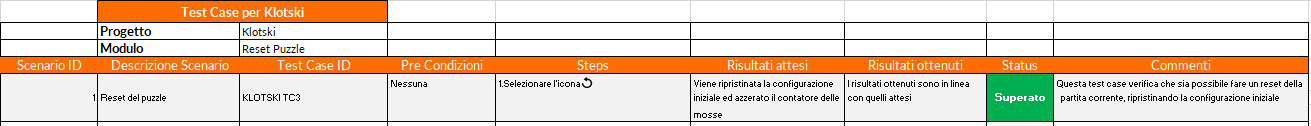
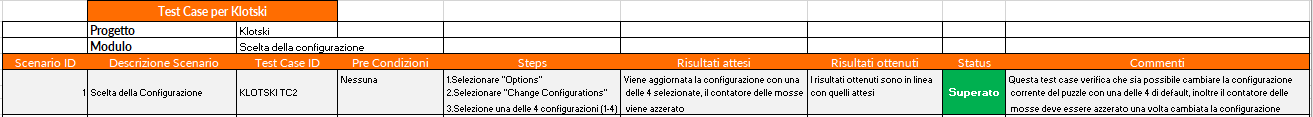
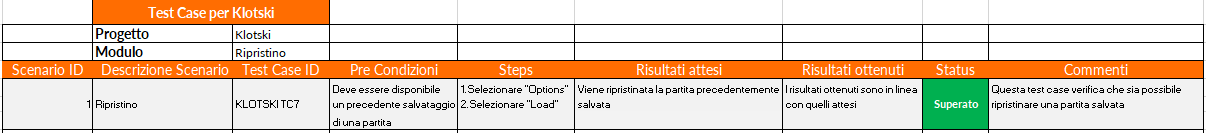
**Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, linea, numero, Carattere

Descrizione generata automaticamente**

**DEFINIZIONE DEI TEST CASES**

**Immagine che contiene testo, Carattere, linea, numero

Descrizione generata automaticamenteImmagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente**

**SYSTEM TEST REPORT**

**Sommario**

Come osservabile nella definizione dei test case questi sono suddivisi in 8 categorie principali:

1. **I test case che verificano il meccanismo di movimento manuale dei blocchi del puzzle (KLOTSKI TC1)**

Questi consistono nello spostamento dei blocchi da parte dell’utente, nello specifico esistono 3 possibilità: l’utente ha eseguito lo spostamento che determina la risoluzione del puzzle, l’utente ha eseguito un generico spostamento valido, l’utente ha provato ad eseguire uno spostamento non valido

1. **Il test case che verifica il corretto funzionamento della scelta della configurazione (KLOTSKI TC2)**
2. **Il test case che determina il corretto funzionamento dell’opzione di reset del puzzle (KLOTSKI TC3)**
3. **I test case che verificano l’undo (KLOTSKI TC4)**

Quindi nel caso sia stato eseguito almeno uno spostamento, si verifichi che sia possibile annullare l’ultima mossa e ripristinare la configurazione precedente

1. **I test case ideati per la next best move (KLOTSKI TC5)**

Servono per stabilire se il solver riesce a trovare per le varie configurazioni la sequenza risolutiva con il minor numero di spostamenti

1. **Il test case che verifica il salvataggio di una partita in corso (KLOTSKI TC6)**

Serve per testare che si possa salvare correttamente una partita in corso

1. **Il test case per il ripristino di un salvataggio (KLOTSKI TC7)**

Serve per verificare che si possa ripristinare correttamente il salvataggio di una partita

1. **Il test case per il Quit (KLOTSKI TC8)**

Si stabilisce se l’opzione di Quit permette effettivamente di terminare l’esecuzione del programma

Questi test case sono stati verificati tramite 3 classi di test: **BoardTest, SaverLoaderTest e SolverTest** , sviluppate con l’ausilio di JUnit.

**BoardTest**

* **testReset()**

**SOMMARIO:** Questo metodo permette di verificare il test Case del Reset

**TEST CASE DESIGN:** Questo metodo crea una Board con una determinata configurazione, fa un’operazione di movimento blocco sulla Board e successivamente un’operazione di Reset, verifica quindi con un assert che la configurazione della Board sia uguale a quella iniziale

**PRE-CONDIZIONI:** L’operazione di movimento blocco deve essere valida, altrimenti il test non è sufficiente per stabilire la correttezza del Reset

**POST-CONDIZIONI:** La board è configurata alla configurazione iniziale

**RISULTATI ATTESI**: L’asserzione conferma l’uguaglianza tra la board nella configurazione iniziale e la board dopo il reset

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

* **testSetConfiguration1(), testSetConfiguration2(), testSetConfiguration3(), testSetConfiguration4()**

**SOMMARIO:** Ciascuno di questi metodi di test permette di verificare che sia possibile cambiare la configurazione corrente nella rispettiva configurazione di default

**TEST CASE DESIGN:** La prima istruzione serve per l’impostazione della configurazione della Board, successivamente si verifica con delle asserzioni che nell’oggetto Board l’attributo identificativo della configurazione sia corretto e che il posizionamento dei blocchi sia coerente con quello atteso.

**PRE-CONDIZIONI:** Per il confronto tra oggetti di tipo Board, viene invocato il metodo toString() su ciascun oggetto, il quale restituisce la configurazione della board sottoforma di stringa, è necessario che il corretto funzionamento di questo metodo sia stato precedentemente verificato nella Unit Testing.

**POST-CONDIZIONI:** Ogni metodo configura la Board alla rispettiva configurazione (testSetConfiguration1()-> Configurazione 1, ecc…)

**RISULTATI ATTESI:** L’asserzione conferma il corretto posizionamento dei blocchi.

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

* **testGetBestSolutionSequence()**

**SOMMARIO:** Questo metodo permette di verificare che ad ogni best move, la Board venga configurata correttamente.

**TEST CASE DESIGN:** Nella prima istruzione viene invocato sull’oggetto board il metodo GetBestSolution(), in modo da ottenere la sequenza delle mosse migliori,

successivamente viene invocato il metodo evaluate() per compiere spostamenti di blocchi su Board e per ogni spostamento di blocchi, si verifica attraverso le asserzioni che la Board venga aggiornata correttamente.

**PRE-CONDIZIONI:** Nello Unit Testing deve essere verificato che il metodo GetBestSolution() restituisca la sequenza più breve di mosse con cui è possibile risolvere il puzzle

**POST-CONDIZIONI:** Alla fine dell’esecuzione del test, la Board rappresenta una configurazione vincente

**RISULTATI ATTESI:** La Board viene aggiornata fino alla configurazione vincente, senza che siano presenti delle anomalie

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

* **testEvaluate()**

**SOMMARIO:** L’esecuzione di questo metodo di test è in relazione con il test case dello spostamento di un blocco, in quanto verifica che il metodo evaluate selezioni il primo spostamento valido nell’insieme delle mosse valide e aggiorni correttamente la configurazione

**TEST CASE DESIGN:** La prima istruzione richiede l’insieme dei possibili spostamenti di un blocco, con evaluate() viene eseguito il primo spostamento valido e con un asserzione si controlla che la board venga aggiornata correttamente dopo lo spostamento.

**PRE-CONDIZIONI:** Affinché il test convalidi la correttezza di evaluate(), è necessario che il metodo che restituisce le possibili mosse valide, quindi GetPossibleMoves()venga a sua volta testato.

**POST-CONDIZIONI:** La board è correttamente configurata all’ultima operazione di spostamento

**RISULTATI ATTESI:** L’asserzione dimostra che il metodo evaluate() aggiorna correttamente la configurazione dopo lo spostamento del blocco.

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

* **testCheckwin()**

**SOMMARIO:** Questo metodo di test verifica che la configurazione vincente determini effettivamente la condizione di vittoria

**TEST CASE DESIGN:** Viene inizialmente creato un oggetto Board configurato alla configurazione vincente, a questo punto con un’asserzione si verifica che il sistema segnali la condizione di vittoria

**PRE-CONDIZIONI:** L’oggetto Board su cui si esegue il test deve effettivamente rappresentare una configurazione vincente

**POST-CONDIZIONI:** Il sistema ha appurato la risoluzione del puzzle

**RISULTATI ATTESI:** Viene verificato che una configurazione vincente identifichi la risoluzione del puzzle

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

* **testGetPossibleMoves()**

**SOMMARIO:** Anche questo metodo di test è in relazione con il test case dello spostamento di un blocco ed insieme al metodo testEvaluate() permette di verificare ulteriormente che il sistema reagisca in maniera corretta allo spostamento di un blocco

**TEST CASE DESIGN:** Attraverso delle asserzioni di uguaglianza tra evaluate() e le varie mosse restituite dall’invocazione di GetPossibleMoves(), si verifica che il metodo GetPossibleMoves() restituisca unicamente spostamenti di blocchi validi

**PRE-CONDIZIONI:** Deve essere testato il corretto funzionamento del metodo evaluate()

**POST-CONDIZIONI:** Viene esaminata la validità per ogni singola mossa restituita da GetPossibleMoves()

**RISULTATI ATTESI:** Viene verificato che GetPossibleMoves() restituisca unicamente mosse valide

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

**SaverLoaderTest**

* **testSave()**

**SOMMARIO:** Questo metodo di test verifica che sia possibile salvare correttamente una partita

**TEST CASE DESIGN:** Sull’oggetto SaverLoader, la cui funzione è quella di contenere le informazioni relative alla configurazione salvata, viene eseguita un’operazione di salvataggio, si verifica che questa venga completata senza anomalie

**PRE-CONDIZIONI:** É sufficiente aver inizializzato un oggetto Board prima di eseguire l’operazione di salvataggio

**POST-CONDIZIONI:** Il metodo save restituisce la stringa “ok” per segnalare che il salvataggio è avvenuto correttamente, inoltre, l’oggetto SaverLoader contiene ancora le informazioni relative alla configurazione salvata

**RISULTATI ATTESI:** L’asserzione usata per verificare il corretto salvataggio, convalida che il metodo save abbia restituito “ok”

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

* **testLoad()**

**SOMMARIO:** Questo metodo di test verifica che il ripristino di un salvataggio funzioni propriamente

**TEST CASE DESIGN:** Sull’oggetto SaverLoader (su cui era precedentemente stato salvato un oggetto Board) viene eseguito il metodo load() che permette di restituire l’ultimo oggetto Board salvato, con un’asserzione si verifica che la Board inizialmente salvata nel SaverLoader coincida con la Board restituita da load()

**PRE-CONDIZIONI:** Prima di eseguire testLoad() è necessario aver eseguito il metodo testSave() ed aver appurato il corretto funzionamento del metodo save(), dato che all’istante d’esecuzione di testLoad() il SaverLoader deve contenere l’istanza della Board

**POST-CONDIZIONI:** L’oggetto SaverLoader non viene modificato

**RISULTATI ATTESI:** L’asserzione d’uguaglianza conferma che la Board salvata viene ripristinata correttamente

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi

**SolverTest**

* **testGetNextMove()**

**SOMMARIO:** Questo metodo di test estende la Test Coverage per quando riguarda l’algoritmo di Solving e dimostra che una sequenza vincente è individuabile per ciascuna delle quattro configurazioni di default

**TEST CASE DESIGN:** Viene eseguito GetNextMove() sul solver, fintantoché questo restituisce una Board per cui l’invocazione di checkwin() restituisce TRUE, questo test viene eseguito per ogni Board di default

**PRE-CONDIZIONI:** Per ritenere il test efficace, il testCheckwin() deve essere stato precedentemente eseguito e deve aver dato esiti positivi

**POST-CONDIZIONI:** Al termine del test, l’oggetto Board si trova alla configurazione vincente ed il sistema ha segnalato la risoluzione del puzzle

**RISULTATI ATTESI:** Per ciascuna Board di default, il solver attraverso un numero limitato di invocazioni di GetNextMove(), deve aggiornare la Board nella configurazione vincente

**RISULTATI OTTENUTI:** I risultati ottenuti sono in linea con quelli attesi