Cruci Puzzle

1 Introduzione 4

1.1 Informazioni sul progetto 4

1.2 Abstract 4

1.3 Scopo 4

2 Analisi 5

2.1 Analisi del dominio 5

2.2 Analisi e specifica dei requisiti 5

2.2.1 Spiegazione elementi tabella dei requisiti: 8

2.3 Use case 8

2.4 Pianificazione 9

2.5 Analisi dei mezzi 9

2.5.1 Software 9

2.5.2 Hardware 9

3 Progettazione 10

3.1 Design dell’architettura del sistema 10

3.2 Design delle interfacce 11

3.3 Design procedurale 12

4 Implementazione 13

4.1 Cartella sorgente 13

4.2 Classe Campo 14

4.2.1 Valore costanti 14

4.2.2 Metodo principale generazione campo 15

4.2.3 Metodo generazione parole 16

4.2.4 Metodo generazione coordinate e inserimento parola verticale 17

4.2.5 Metodo generazione coordinate e inserimento parola orizzontale 18

4.2.6 Metodo generazione coordinate e inserimento parola Obliqua 19

4.2.7 Metodo per controllare la plausibilità delle coordinate 20

4.2.8 Metodo che controlla le celle restanti 21

4.3 Classe Parole 23

4.3.1 Metodo costruttore classe 23

4.3.2 Metodo seleziona parole 24

4.3.3 Metodo di verifica della parola 24

4.3.4 Metodo seleziona parole con numero caratteri 25

4.4 Classe InputForm 25

4.4.1 Metodo selezione font 26

4.4.2 Metodo selezione 26

4.4.3 Metodo di generazione 27

4.5 Classe MainForm 27

4.5.1 Metodo Costruttore classe MainForm 28

4.5.2 Metodo di inserimento elementi campo 28

4.5.3 Metodo di inserimento parole 29

4.5.4 Metodo esportazione risultato in formato PNG 29

4.5.5 Metodo esportazione risultato in formato TXT 30

4.5.6 Metodo per stampare il risultato finale 31

4.5.7 Metodo che rende invisibili i bottoni nella pagina 31

4.6 Classe GameTableModel 32

4.6.1 Metodo per impostare le lettere 32

4.7 Classe WordsTableModel 32

4.7.1 Metodo per impostare le parole 32

4.8 Classe SoluzioneForm 33

4.8.1 Metodo generazione tabella 33

5 Test 34

5.1 Protocollo di test 34

5.2 Risultati test 41

5.3 Mancanze/limitazioni conosciute 45

6 Consuntivo 45

7 Conclusioni 45

7.1 Sviluppi futuri 45

7.2 Considerazioni personali 45

8 Link Utili 46

8.1 Sitografia 46

9 Glossario 46

10 Allegati 46

# Introduzione

## Informazioni sul progetto

Cruci Puzzle è un progetto che consiste nel fare un campo pieno di lettere e una lista di parole che sono all’interno del campo che l’utente deve trovare.

* Responsabile del progetto: professore Geo Petrini,
* Allievo: Jacopo Faul
* Scuola: Scuola D’arti e Mestieri Trevano – Sezione Informatica
* Inizio: 01.09.2023
* Fine: 01.12.2023

## Abstract

*Con l’avanzamento dell’età media e dei dispositivi elettronici, visto che creare questo tipo di gioco è molto complicato e richiede molto tempo, quando mi è stato consegnato ero molto contento di doverlo sviluppare, un’applicazione che generi CruciPuzzle in tempi molto ridotti è molto utile soprattutto quando ci si vuole svagare un pochino con dei giochi di logica, grazie a questo programma, l’utente ha la possibilità di generare infiniti campi di gioco, con la possibilità di stampare il risultato o di salvare sul proprio dispositivo in formato immagine o testo.*

## Scopo

Lo scopo del progetto è quello di riuscire ad aumentare le mie capacità nell’ambito della programmazione e le mie qualità nel “Problem Solving”.

# Analisi

## Analisi del dominio

Questo gioco verrà utilizzato principalmente da bambini e ragazzi che vogliono divertirsi un po’ o semplicemente dal signore o la signora in pensione che ha voglia di giocare a dei giochi di logica, ma sarà tranquillamente utilizzabile anche da qualsiasi altra persona grazie al file che restituisce l’applicazione.

Applicazione in grado di esportare il risultato finale in formato .txt e .png

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Multipiattaforma |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | L’applicazione deve funzionare su Windows, MAC e Linux |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Creazione interfaccia grafica |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Interfaccia iniziale dove l’utente può scegliere le varie modalità e difficoltà e un’altra schermata dove verrà mostrato il risultato finale con le varie funzioni di export |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Scelta modalità di gioco |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si dovrà poter scegliere la modalità, ovvero, modalità bambino (senza parola nascosta) e modalità normale (con parola nascosta) |
| **Sotto requisiti** | |
| **001** | Si dovrà poter scegliere la difficoltà del gioco intesa come quantità di parole all’interno del campo e rispettivamente la grandezza del campo stesso |
| **002** | Se la modalità bambino o modalità normale |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Input utente |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente deve poter inserire delle opzioni per generare il campo come preferisce personalizzandolo |
| **Sotto requisiti** | |
| **001** | Possibilità di poter passare un file di parole scelte dall’utente, verrà esclusivamente usato questo file per generare il campo con le parole. Se non dovesse passare alcun file ne verrà usato uno di default. |
| **002** | Possibilità di scelta del FONT con default “Monospace” |
| **003** | Controllo input |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Generazione campo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | La generazione e la rappresentazione del gioco ovvero il campo di gioco e la lista di parole rappresentate graficamente |
| **Sotto requisiti** | |
| **001** | Rappresentazione grafica del campo |
| **002** | Rettangolo attorno al campo di gioco |
| **003** | Lista di parole formato colonne non lista unica |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Orientamento parole all’interno del campo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Durante la generazione del campo le parole da trovare possono essere posizionate in 4 modi diversi |
| **Sotto requisiti** | |
| **001** | Parole orizzontali con lettura da sinistra a destra o viceversa |
| **002** | Parole verticali con lettura dall’alto al basso o viceversa |
| **003** | Parole oblique da sinistra a destra lette dall’alto verso il basso o viceversa |
| **004** | Parole oblique da destra a sinistra lette dall’alto verso il basso o viceversa |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Export |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Export deve essere fatto tramite GUI |
| **Sotto requisiti** | |
| **001** | Possibilità di Export in formato .txt e .png |
| **002** | Stampabile su di un foglio formato A4 |

|  |  |
| --- | --- |
| **ID: REQ-08** | |
| **Nome** | Soluzione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Soluzione CruciPuzzle visibili all’utente |

|  |  |
| --- | --- |
| **ID: REQ-09** | |
| **Nome** | Nessun margine di errore |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’applicazione non deve generare errori, niente sovrapposizione di parole né altro |

### Spiegazione elementi tabella dei requisiti:

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto.

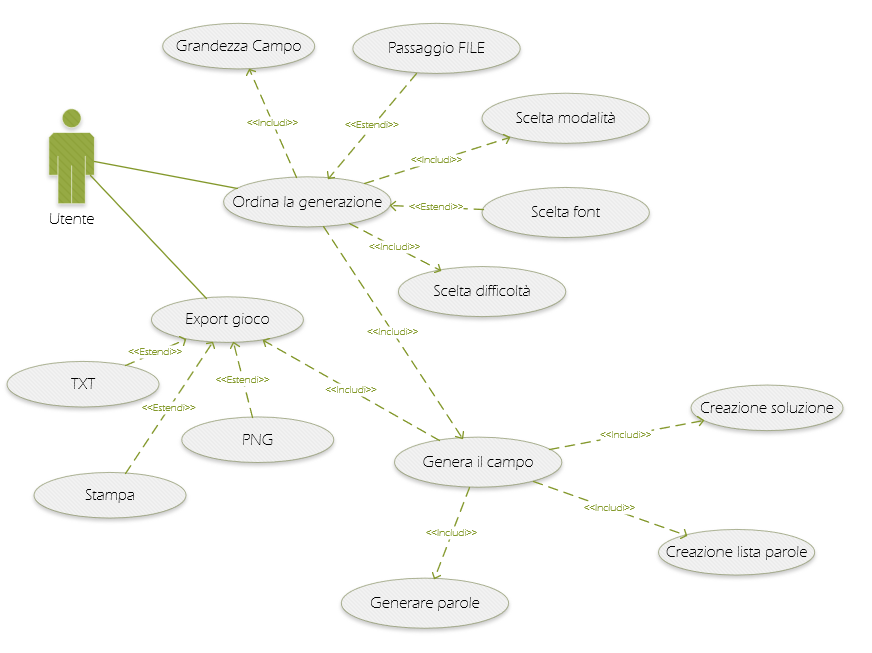


Figura 1 - Use Case

## Pianificazione

Figura 2 - Diagramma di Gantt

## Analisi dei mezzi

I mezzi utilizzati per lo sviluppo di questo progetto sono i seguenti e sono messi a disposizione dell’istituto scolastico Centro professionale tecnico Lugano-Trevano (CPTT).

* **Computer:** HP EliteDesk 800 G5 TWR (i7-9700, 32 GB Ram, RTX 2060, SSD 512 GB)
* **Software Sviluppo:** Apache NetBeans IDE 17
* **Versione JAVA:** JDK 19
* **Sistema Operativo:** Windows 10 22H2

### Software

Il software utilizzato per lo sviluppo è “**Apache NetBeans IDE 17**”, l’ultima versione disponibile in questo momento, la versione di Java che sto utilizzando al momento, come scritto sopra, è la **19**.

### Hardware

La piattaforma è stata realizzata su Windows 10 ed è pensata per essere multipiattaforma, il che significa che quest’applicazione deve funzionare su più sistemi operativi (Windows, Linux, MacOS).

Come scritto nel punto soprastante, l’hardware su cui sto sviluppando è un HP EliteDesk.

# Progettazione

## Design dell’architettura del sistema

UML

## Design delle interfacce

Figura 3 – Interfaccia Input Utente

Come primissima interfaccia mostro l’interfaccia iniziale che l’utente vede appena esegue l’applicazione, essa è l’interfaccia principale del programma perché l’utente deve inserire diverse informazioni per generare il campo di gioco come meglio crede.

* **Modalità di gioco:** La modalità di gioco stabilisce se le parole da inserire nel campo di gioco devono essere contate o se devono essere inserite fino a quando ci stanno.
* **Difficoltà di gioco:** La difficoltà di gioco sarà disponibile unicamente se la modalità di gioco sarà impostata su “Bambini” e consiste in tre difficoltà basate sul numero di parole nascoste all’interno del campo e sono le seguenti:
  + **Facile:** 10 Parole nascoste
  + **Media:** 15 Parole nascoste
  + **Difficile:** 20 Parole nascosta
* **Seleziona Font:** L’utente ha la possibilità di scegliere un Font per il campo personalizzato, in modo da avere l’interfaccia grafica personalizzata come meglio crede, per questo motivo la scelta del Font è disponibile, se non venisse selezionato alcun Font di default verrà applicato il Font “Monospace”.
* **File personalizzato:** L’utente ha la possibilità di inserire un file di parole che verrà utilizzato per creare il campo, se non venisse aggiunto alcun file, il programma genererà il campo con un file di Default.

Una volta aggiunte tutte le informazioni obbligatorie si potrà avviare il programma schiacciando il bottone in basso “GENERA”, esso farà aprire un’altra finestra con all’interno il campo e le parole e la possibilità di stampare il tutto o di esportarlo in formato PNG o TXT.



Figura 4 – Finestra Risultato Finale

Questa finestra mostra infine il risultato finale dell’applicazione generata, come si può notare abbiamo i tre pulsanti per esportare il risultato finale in formato TXT, nel formato PNG oppure stamparla direttamente.

Possiamo anche notare come il campo abbia un bordo nero come richiesto e ci sia anche lo spazio per scrivere la parola nascosta, inoltre notiamo anche come abbiamo tutte le parole da cercare scritte sotto il riquadro di gioco.

## Design procedurale

Scelta Parola - Classe Parole

Controllo celle - Classe Campo

# Implementazione

## Cartella sorgente

Il mio codice sorgente si compone di 8 file, la struttura del mio progetto è la seguente:

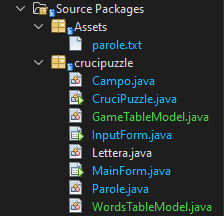


Figura 5 – Cartella SRC del progetto

Nella cartella “Source Packages” troviamo due package:

* **Assets**, contiene il file di default delle parole,
* **Crucipuzzle**, contiene le classi necessarie al corretto funzionamento del progetto

All’interno del package “Crucipuzzle” troviamo le seguenti classi:

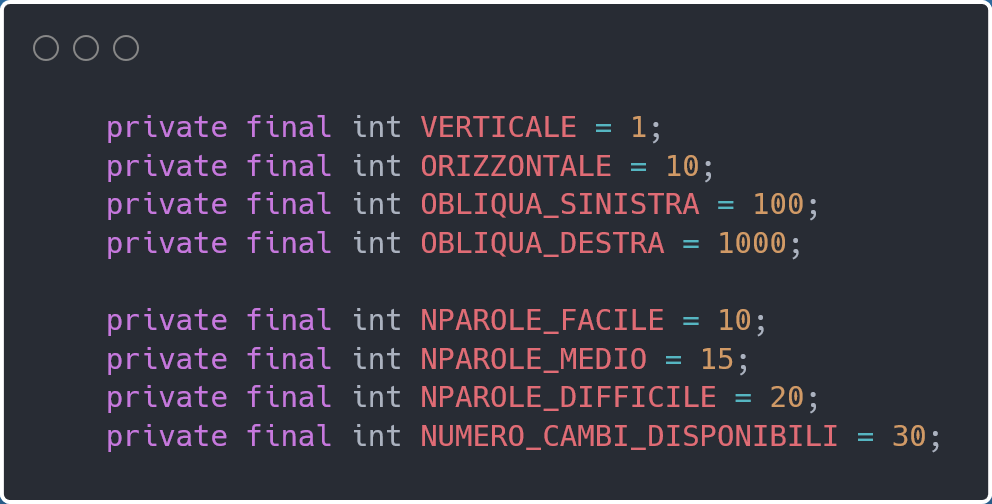
* **Campo**, contiene tutti i metodi necessari a popolare il campo, gli array delle parole e le soluzioni delle posizioni delle parole;
* **CruciPuzzle,** contiene il metodo Main del programma dove lo farà partire;
* **GameTableModel,** contiene il modello della JTable del campo, in questo modo posso impostare le varie impostazioni di visualizzazione e implementaggio della tabella dove verrà mostrato il campo sulla GUI;
* **InputForm,** contiene il codice necessario a ricevere le informazioni dall’utente riguardanti le impostazioni del campo, delle parole e del font, è anche la classe iniziale che mostra il programma appena esso viene aperto;
* **Lettera,** contiene la struttura all’oggetto che conterrà il campo;
* **MainForm,** contiene la struttura e il codice base all’avvio per la generazione del campo e la stampa di quest’ultimo, contiene infatti la struttura della schermata finale che avrà l’applicazione;
* **Parole,** contiene tutti i metodi necessari a scegliere le parole, scartarle e pulirle da eventuali caratteri speciali;
* **WordsTableModel,** contiene il modello della JTable delle parole, in questo modo posso impostare le varie impostazioni di visualizzazione e implementaggio della tabella dove verranno mostrate tutte le parole da trovare all’interno del campo.

## Classe Campo

La classe campo è la classe che contiene tutti i metodi più importanti alla realizzazione del campo di gioco, cominciamo quindi dal metodo per generare tutte le parole.

### Valore costanti

Figura 7 – Classe Campo, costanti



Le varie costanti definite all’interno della classe Campo, servono per le seguenti ragioni:

* **VERTICALE**, imposta il valore per la direzione verticale
* **ORIZZONTALE**, imposta il valore per la direzione orizzontale
* **OBLIQUA\_SINISTRA**, imposta il valore per la direzione obliqua **\**
* **OBLIQUA\_DESTRA**, imposta il valore per la direzione obliqua **/**

Le variabili che servono invece per la modalità bambini sono:

* **NPAROLE\_FACILE**, imposta il numero di parole per la difficoltà “facile”
* **NPAROLE\_MEDIO**, imposta il numero di parole per la difficoltà “media”
* **NPAROLE\_DIFFICILE**, imposta il numero di parole per la difficoltà “difficile”

Infine, la costante che mi permette di non entrare in un’eccezione “StackOverFlow”:

* **NUMERO\_CAMBI\_DISPONIBILE**, quando una parola non va bene per quel numero di volte ritorna al punto iniziale del programma cambiando l’ordine di accesso ai metodi del programma

### Metodo principale generazione campo



Figura 6 – Classe Campo metodo generaCampo()

Nella classe Campo, troviamo il metodo che genera tutte le parole in base alla modalità di gioco e alla difficoltà, per questo motivo ho, all’interno del metodo due parti separate, una, quella sopra, contente la parte di codice necessaria a generare il campo con la modalità “bambini” e la relativa difficoltà selezionata.  
La seconda parte del metodo, invece, è quella necessaria a generare il campo per la modalità di gioco “Normale”.

Nella prima parte, quella della modalità “bambini” vado ad impostare la variabile di tipo intero “numeroParole” con il suo valore definito nella costante (NPAROLE\_FACILE, NPAROLE\_MEDIO, NPAROLE\_DIFFICILE) scelta in base alla difficoltà scelta dall’utente, dopodiché comincio a selezionare le parole fino al numero scelto.

La generazione delle parole all’interno del ciclo for mi permette di controllare con un metodo più comprensibile il numero di parole, questo perché ogni volta che incrementerò la variabile “i” del ciclo saprò per certo che mi ha aggiunto una parola, in quanto tutto il lavoro lo fa il metodo “generaTutteLeParole()”.

Nella seconda parte del codice, quella della modalità “normale” troviamo un ciclo while, la cui condizione è un po’ particolare, in poche parole, io continuo il ciclo fino a quanto il numero di spazi vuoti all’interno del campo è maggiore del 10% del numero totale di caratteri nel campo, come esempio, se dovessi avere un campo “10x10” avrò 100 caratteri, il while continua fino a quando non avrò almeno 10 caratteri vuoti, dopodiché fermerà il ciclo.

All’interno del ciclo vengono fatti due controlli, il primo è se non ho più parole, esso mi fa fermare il ciclo se dovesse essere vero, il secondo controllo mi controlla che la parola non sia già stata inserita nel campo.

Il metodo che viene fatto in seguito “controllaLeCelleRestanti()” è il metodo che serve a riempire gli eventuali vuoti con più di 3 spazi di seguito.

### Metodo generazione parole

Grazie a questo metodo riesco a generare la direzione delle parole e inserirle nel campo.

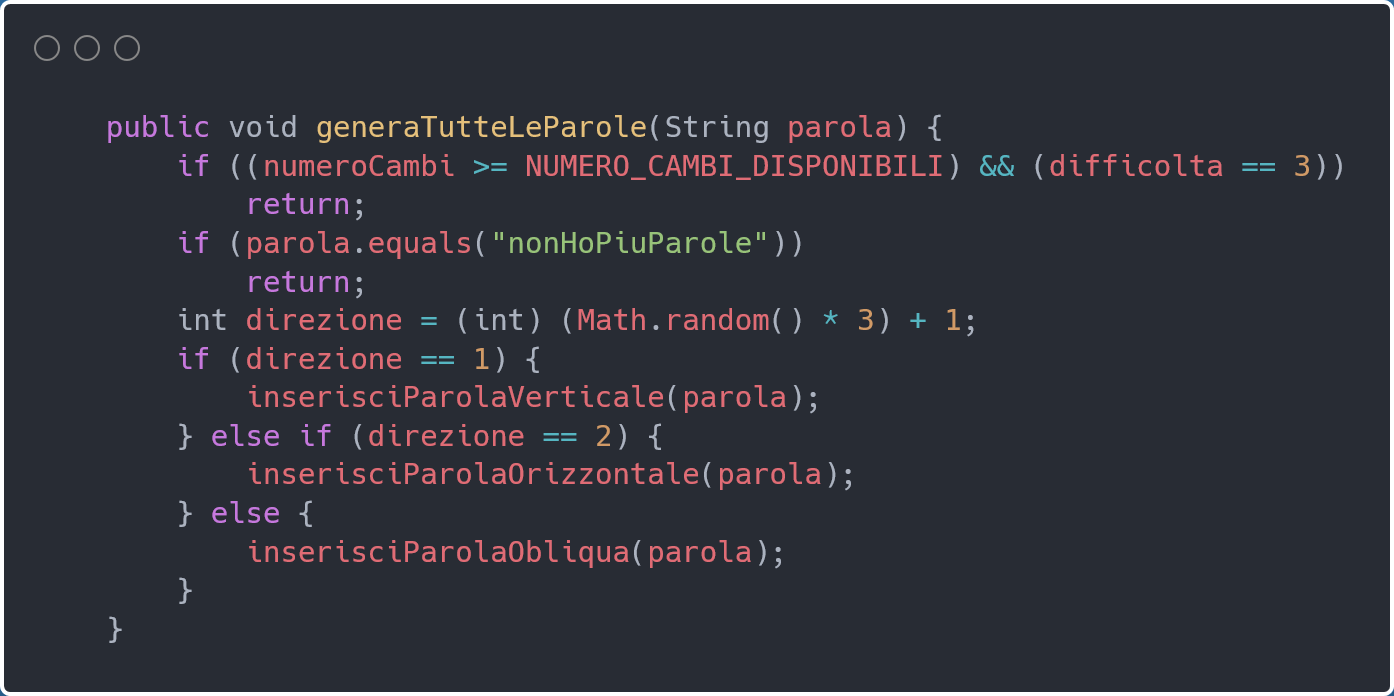


Figura 7 – Classe Campo, metodo generaTutteLeParole()

Il metodo “generaTutteLeParole()” richiede un parametro, esso sarà la parola che dovrà essere inserita all’interno del campo, o meglio, la parola che bisognerà verificare se possibile inserirla o meno.

Questo metodo è anche il metodo di ritorno per tutte le parole che non vengono inserite correttamente, infatti, se una parola non viene inserita correttamente, essa verrà rigenerata e passata a questo metodo per continuare con la generazione.

All’interno di questo metodo possiamo notare come ci sono diversi controlli da fare, il primo è il numero di cambi, ossia, proprio ricollegandoci alla funzionalità del metodo scritta prima, esso è il punto di ritorno di tutte le parole, per questo motivo, dobbiamo controllare (per evitare di entrare in uno StackOverflow) il numero di volte che una parola non è stata inserita correttamente e cambiare così lo stack di chiamate.

Un altro controllo che va fatto è quello sulla quantità di parole, infatti, se non ho più parole mentre ne sostituisco non devo provare ad inserirla nel campo, per questo motivo ho implementato anche il secondo “if” presente nel metodo.

Infine, ma non per importanza, troviamo il metodo che genera, in modo casuale, la direzione della parola:

* **Risultato = 1**, inserisco una parola orientata verticalmente;
* **Risultato = 2**, inserisco una parola orientata orizzontalmente;
* **Risultato = 3**, inserisco una parola orientata obliqua

### Metodo generazione coordinate e inserimento parola verticale



Figura 7 – Classe Campo, metodo inserisciParolaVerticale()

Questo metodo mi permette di generare le coordinate e di verificare se una parola è inseribile a quelle coordinate, per questo motivo, all’inizio, con una probabilità del 25% giro la parola da inserire per poi generare le coordinate.

All’interno del “Do–While” vediamo come ho implementato un sistema per cambiare la parola dopo tre tentativi che quest’ultima non funziona all’interno del codice.

Inoltre, genero una riga e una colonna permettendomi di non dover controllare in seguito la validità delle coordinate generate, per questo motivo, per generare la riga e la colonna ho impostato che:

* Riga = un numero compreso tra 0 e la grandezza del campo – la lunghezza della parola.
* Colonna = un numero compreso tra 0 e la grandezza del campo.

La condizione del “Do-While” consiste nel verificare tutto con un altro metodo che, se ritorna false significa che quelle coordinate non vanno bene per la parola selezionata.

Se non dovesse andare a buon fine, entro tre tentativi rigenero le coordinate, dopodiché cambio la parola.

Infine, una volta trovate le coordinate inserisco la parola all’interno del campo tramite il metodo predisposto.

### Metodo generazione coordinate e inserimento parola orizzontale



Figura 8 – Classe Campo, metodo inserisciParolaOrizzontale()

Questo metodo è uguale a quello precedente, l’unica cosa che cambia è sostanzialmente la generazione della riga e della colonna, che in questo caso si comporta in questo modo:

* Riga = un numero compreso tra 0 e la grandezza del campo;
* Colonna = un numero compreso tra 0 e la grandezza del campo – la lunghezza della parola.

### Metodo generazione coordinate e inserimento parola Obliqua

Per il metodo che genera le coordinate e l’inserimento della parola in obliquo cambiano un po’ di cose, come per esempio il fatto che, all’inizio del metodo, genero in base ad un numero casuale se la direzione è verso sinistra “**\**” o verso destra “**/**”.

Dopodiché, come in tutti i metodi precedenti, anche in questo genero con il 25% di possibilità la parola al contrario, successivamente procedo con la generazione delle coordinate.

Per la generazione delle coordinate ho le seguenti opzioni:

* **Obliqua Sinistra**
  + Riga = un numero compreso tra 0 e grandezza del campo – la lunghezza della parola;
  + Colonna = un numero compreso tra 0 e grandezza del campo – la lunghezza della parola;
* **Obliqua Destra**
  + Riga = un numero compreso tra la lunghezza della parola e la grandezza del campo;
  + Colonna = un numero compreso tra 0 e grandezza del campo – la lunghezza della parola;

L’immagine successiva mostra il codice per la generazione delle coordinate della parola.



Figura 9 – Classe Campo, metodo inserisciParolaObliqua()

### Metodo per controllare la plausibilità delle coordinate

  
  
Questo metodo mi serve per verificare che le coordinate generate in precedenza vadano bene per la parola selezionata precedentemente, inoltre, verifico che la cella dove dovrebbe essere inserita la parola sia libera, altrimenti, che la suddetta cella abbia la stessa lettera della lettera della parola all’indice corretto, questo significa che questo metodo comprende:

Figura 10 – Classe Campo, metodo controllaCelle()

* Verifica che la parola alle coordinate selezionate sia inseribile rispetto alle parole già presenti;
* Verifica che la parola alle coordinate selezionate non incroci una parola nella stessa direzione;

### Metodo che controlla le celle restanti



Figura 11 – Classe Campo, metodo controllaLeCelleRestanti()

Questo metodo serve a verificare che tutti gli sposti con più di tre spazi vuoti consecutivi nelle quattro direzioni siano finiti, in caso contrario li riempie, per questo motivo faccio passare tutto il campo e verifico, se la cella che verifico in quel momento è vuota controllo in tutte le direzioni in modo ricorsivo, se infine la somma di almeno una direzione è maggiore di 2 (almeno 3 spazi).

#### Metodo di controllo verticale



Figura 12 – Classe Campo, metodo controllaCellaSuccessivaVerticale()

Questo è il metodo relativo al controllo delle celle in direzione verticale, esso richiede due parametri, che sono rispettivamente “riga” e “colonna”, successivamente verifica la plausibilità e controlla che la cella richiesta sia effettivamente vuota, in caso affermativo richiama sé stesso fino a quando o esce dal campo con la riga o il campo alla riga e colonna selezionata non è vuoto.

#### Metodo di controllo orizzontale

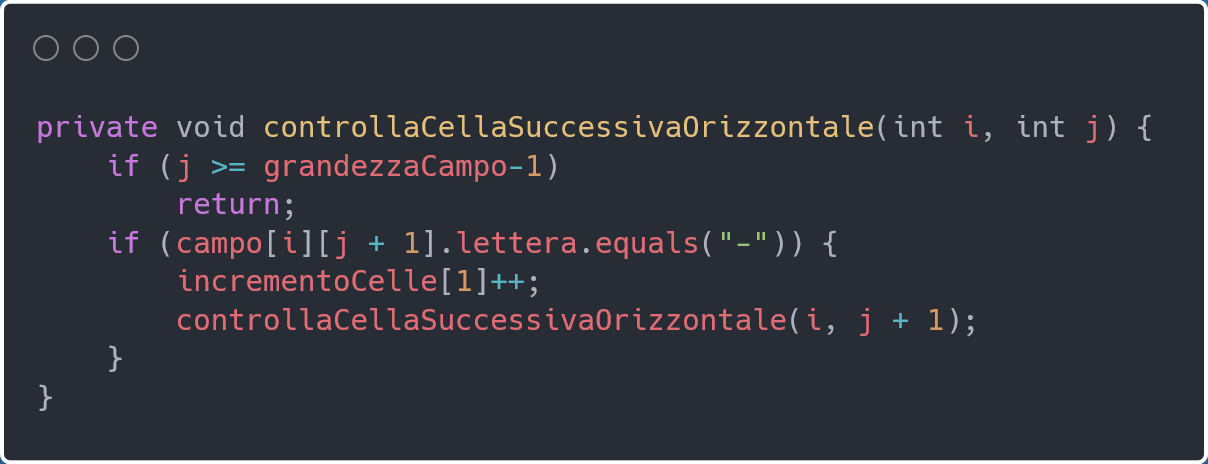


Figura 13 – Classe Campo, metodo controllaCellaSuccessivaOrizzontale()

Come il metodo precedente, con l’unica differenza che al posto di incrementare la riga, incremento la colonna.

#### Metodo di controllo obliquo verso sinistra

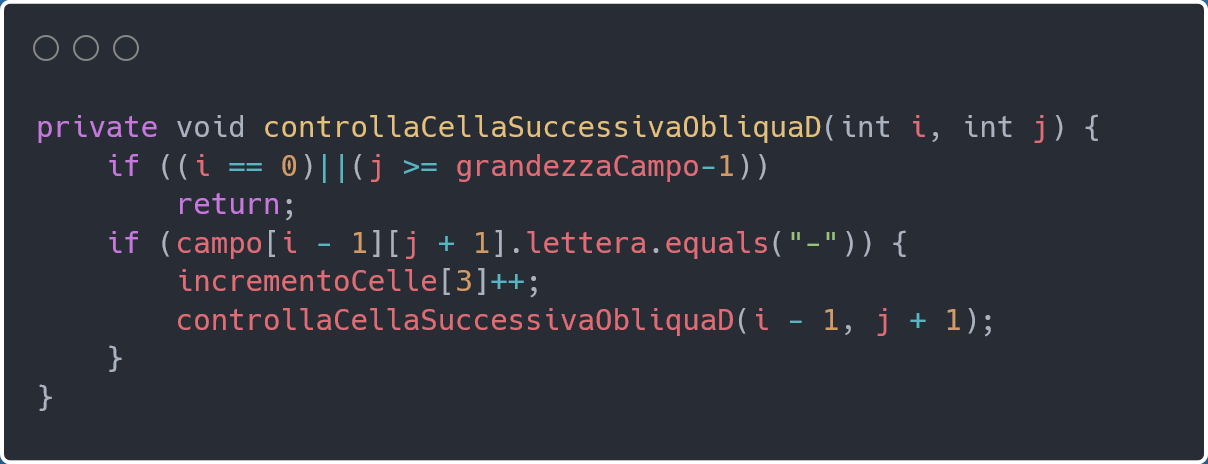


Figura 14 – Classe Campo, metodo controllaCellaSuccessivaObliquaS()

Come il metodo in verticale, con l’unica differenza che al posto di incrementare solamente la riga incremento anche la colonna.

Figura 15 – Classe Campo, metodo controllaCellaSuccessivaObliquaD()

#### Metodo di controllo obliquo verso destra



Come il metodo per obliquo verso sinistra, decremento solamente la riga e incremento la colonna.

## Classe Parole

Questa classe gestisce la selezione delle parole, in modo da avere una maggiore pulizia del codice.

### Metodo costruttore classe

Nel metodo costruttore della classe “Parole” ho semplicemente inizializzato le varie variabili che avevo dichiarato all’inizio della classe e successivamente ho letto il file e salvato le sue informazioni all’interno di un ArrayList, per questo motivo, il codice è il seguente:



Figura 16 – Classe Parole, costruttore

Possiamo infatti notare come uso un “try and catch” per aprire il file, ed infine un ciclo “while” che mi salva tutte le parole all’interno dell’ArrayList “aList”.

### Metodo seleziona parole

Questo metodo è il metodo principale della classe, esso si occupa di selezionare le parole e verificare che esse non siano più piccole di 2 caratteri e più grandi della grandezza del campo.



Figura 17 – Classe Parole, metodo selezionaParole()

Come possiamo notare, infatti, il metodo “selezionaParole” mi seleziona una parola casualmente all’interno di un ciclo “Do-While”, successivamente me la pulisce da caratteri speciali e me la controlla:

* Se è più grande della grandezza del campo
* Più piccola di 3 caratteri
* Oppure se è già all’interno del campo

Rigenerami una parola, altrimenti prova ad inserirla nel campo.

### Metodo di verifica della parola

Questo metodo mi permette di verificare che ogni parola sia valida, in termini di dimensioni e di parola non utilizzata all’interno della griglia creata.

Il metodo è molto semplice nel suo insieme e si presenta nella seguente maniera:

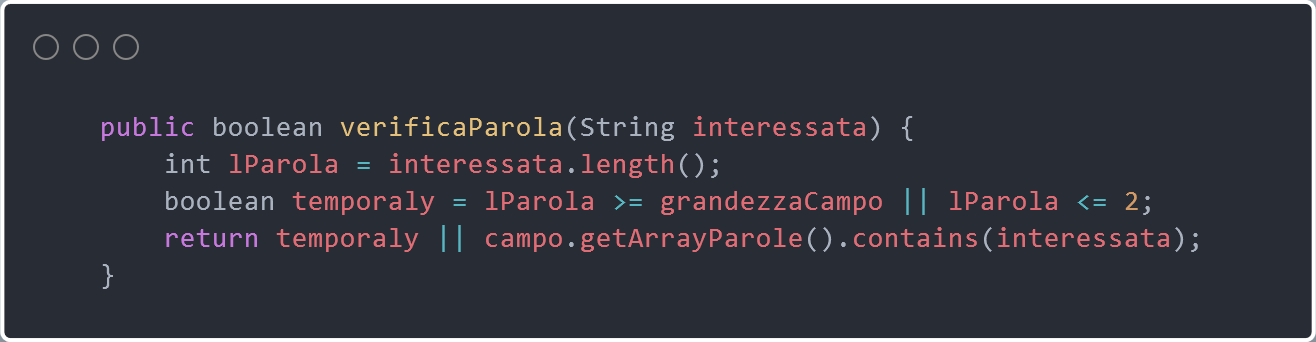


Figura 18 – Classe Parole, metodo verificaParola()

Come possiamo vedere, infatti, controllo la grandezza della parola che deve essere tra la grandezza del campo e la lunghezza “2”, successivamente ritorno il risultato precedente oppure se la parola è già stata utilizzata.

In questo caso, se una di queste condizioni è soddisfatta il metodo ritornerà “true”, questo significa che il metodo “selezionaParola” genererà un’altra parola.

### Metodo seleziona parole con numero caratteri

Esiste un ovverride del metodo precedente con un parametro “nChar” che corrisponde alla lunghezza che deve avere la parola, è usato soprattutto quando devo selezionare la parola all’interno del “controllaLeCelleRestanti”, in quanto devo trovare una parola della lunghezza corretta rispetto al numero di caratteri a disposizione.



Figura 19 – Classe Parole, metodo selezionaParole() con caratteri

Come possiamo notare, infatti, all’interno del metodo non abbiamo più un numero casuale, ma faccio passare tutte le parole e controllo che esse siano della stessa lunghezza di caratteri dei caratteri che gli ho chiesto di fornirmi.

In seguito mi pulirà la parola e me la passerà nella variabile dove avrò invocato il metodo.

## Classe InputForm

Questa classe rappresenta la classe principale di generazione del gioco, questo significa che in questa interfaccia l’utente può scegliere le varie impostazioni di generazione del campo.  
Si presenta in questo modo:

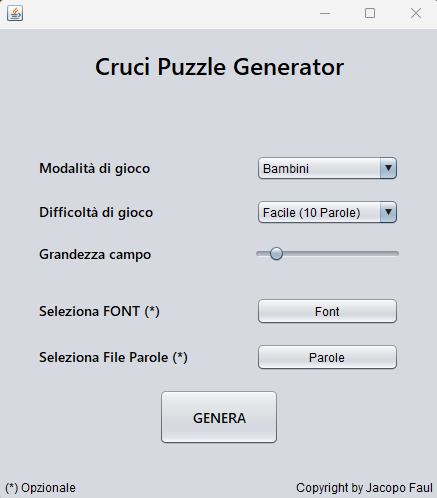


Figura 20 – Classe InputForm, GUI

In questa interfaccia l’utente può scegliere diverse opzioni, la prima è la modalità di gioco (Bambini o Normale), la seconda è la difficoltà di gioco che si differenzia per il numero di parole:

* Facile (10 Parole)
* Media (15 Parole)
* Difficile (20 Parole)

La grandezza del campo che in base alla modalità di gioco varia la grandezza minima del campo, infatti, se la modalità di gioco scelta è quella “Bambini” la dimensione minima sarà 14, mentre se la modalità è “Normale” la dimensione minima sarà 8.

Infine, le due opzioni facoltative che hanno valori di default se non inserite sono la scelta del font (default: MONOSPACE) ed il file di parole (default: parole.txt).

### Metodo selezione font

Questo metodo serve a far scegliere all’utente un font personalizzato.



Figura 21 – Classe InputForm, metodo selezione font

Come si può vedere, istanzio l’oggetto JFontChooser che è una libreria importata e successivamente mostro la finestra di dialogo del componente utile per selezionare il font.

### Metodo selezione



Figura 22 – Classe InputForm, metodo selezione file

Questo metodo, come quello precedentemente mostrato, permette all’utente di selezionare solamente i file di formato “.txt” e seleziona poi un eventuale file nella variabile predisposta che verrà poi utilizzata per istanziare l’oggetto della classe Parole e della classe Campo.

### Metodo di generazione

Questo metodo viene invocato solamente quando l’utente clicca sul bottone “Genera”.  
Una volta richiamato, il metodo salva in delle variabili

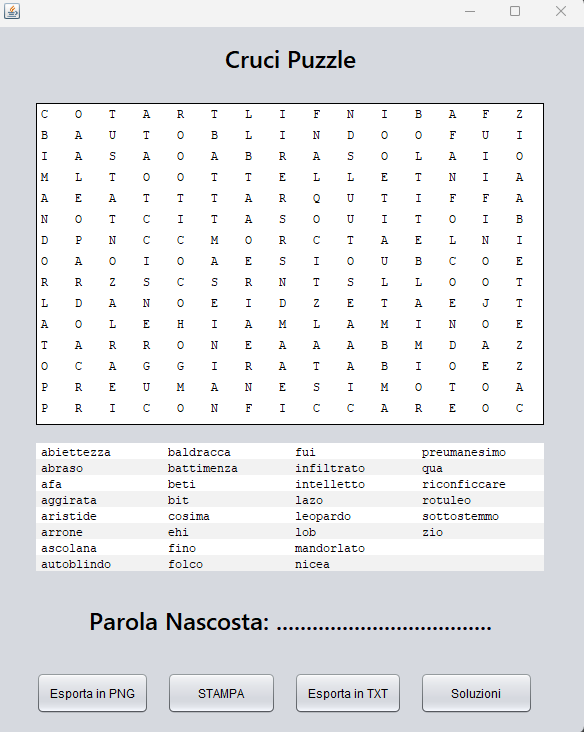


Figura 23 – Classe InputForm, metodo generazione campo

Per questo motivo imposto la difficoltà in base alla modalità scelta e alla difficoltà relativa, successivamente la grandezza del campo viene salvata in un’altra variabile globale e successivamente la path di un eventuale file di parole alternativo.

Per ultima cosa, creo l’oggetto della classe Parole con i vari parametri presi precedentemente, l’oggetto Campo e il MainForm con le due classi precedenti.

## Classe MainForm

Questa classe rappresenta l’interfaccia grafica che l’utente visualizza una volta che il campo è completato, essa si presenta in questo modo:

Come possiamo ben vedere, l’applicazione si presenta con una tabella per mostrare il campo sopra e una tabella di 4 colonne per vedere le parole da trovare all’interno del campo, inoltre, se la modalità è quella “Normale” vedremo anche il posto per inserire la parola nascosta.

Figura 24 – Classe MainForm, GUI

Sul fondo della pagina possiamo notare come siano presenti 4 pulsanti, da sinistra:

* Pulsante che esporta l’applicazione in formato PNG
* Pulsante che stampa il progetto su carta
* Pulsante che esporta l’applicazione in formato TXT
* Pulsante che mostra una tabella con le coordinate delle parole (soluzioni).

### Metodo Costruttore classe MainForm

Il metodo costruttore di questa classe inizializza le varie variabili presenti e dà l’ordine di generare il campo, successivamente, se la modalità di gioco è impostata su “Normale” verrà richiesta la parola nascosta all’utente.

Un’altra caratteristica di questa classe è che le due tabelle presenti sono state create utilizzando dei modelli di tabelle predisposti e creati apposta per le tabelle presenti, infatti, nel metodo costruttore possiamo anche notare che vengono inizializzati due modelli per le tabelle:

* GameTableModel 🡪 Il modello della tabella dove vi è il campo;
* WordsTableModel 🡪 Il modello della tabella dove vi sono le parole da trovare.



Figura 25 – Classe MainForm, Metodo costruttore

Come possiamo notare e come detto in precedenza, ho creato i due modelli e li ho impostati alle due tabelle, successivamente imposto il font selezionato e richiamo i due metodi per riempire le tabelle.

### Metodo di inserimento elementi campo

Questo metodo inserisce nella tabella tutte le lettere che sono contenute nel campo per il gioco.



Figura 26 – Classe MainForm, Metodo inserisciInTabella

Infatti, possiamo notare che in questo metodo ci sono due for, uno dentro l’altro, che mi cicla tutto il campo e imposto il valore alla cella interessata in modo da avere la mia tabella che rappresenta il campo di gioco pieno di lettere.

Successivamente, imposto l’altezza delle righe in modo dinamico, per far si che avendo un campo piccolo, la mia tabella sia riempita completamente.

### Metodo di inserimento parole

Questo metodo riempie la tabella delle parole sotto il campo in modo da avere tutte le parole da cercare all’interno del campo il più ordinate possibili e divise in colonne come da requisiti.



Figura 27 – Classe MainForm, Metodo inserisciParole

Come nel metodo sopra utilizzo due for, uno che rappresenta le colonne (il primo) e il secondo che rappresenta la riga della parola che varia in base alla variabile “ripetizionePerFor”, successivamente controllo che l’indice della parola sia minore del numero delle parole totali e se minore imposto alla riga “j” colonna “i” la parola all’indice corrente.

### Metodo esportazione risultato in formato PNG

Un requisito del progetto è quello di poter esportare il risultato finale in diversi formati, in questo caso, sviluppando il tutto in java, devo dare la possibilità all’utente di esportare il risultato finale del gioco in formato PNG.



Figura 28 – Classe MainForm, Metodo savePNG

Come possiamo vedere, all’interno del “try – catch” viene scritto il nuovo file in formato “png” salvato nella cartella di avvio del programma.

### Metodo esportazione risultato in formato TXT

Questo metodo permette, come quello precedente, di esportare il risultato dell’applicazione questa volta in formato .txt, il metodo è il seguente:

Figura 29 – Classe MainForm, Metodo saveTXT



Come si può notare, il metodo si divide in due parti, la prima parte corrisponde alla stampa del campo, la seconda parte, invece, rappresenta la stampa delle parole in colonne da quattro.

Per la stampa delle parole in colonne da quattro ho pensato ad un sistema che crea una stringa, composta da 4 parole, per fare un esempio:

Dividendo 10 parole in 4 colonne, avrò per ogni colonna un massimo di 3 parole, per questo motivo, nella creazione della 4 parole per riga avrò una condizione da superare che corrisponde al numero corrente rispetto al numero di parole da inserire.

Ogni volta che costruisco la frase prendo anche la parola che sta al primo posto della colonna successiva, per fare ciò calcolo il numero di parole per righe e lo sommo all’indice attuale facendo in modo di passare direttamente ad una colonna (virtuale) successiva.

### Metodo per stampare il risultato finale

Questo metodo mi serve a completare i requisiti di esportazione del mio progetto, il metodo si presenta nella seguente maniera:

Figura 30 – Classe MainForm, Metodo printToPrinter

Questo metodo, per prima cosa, imposta tutti i bottoni in modo che essi non siano visibili, successivamente avvia un dialogo per impostare la stampante, successivamente salva in una variabile l’immagine da stampare (la pagina) e successivamente me la stampa come possiamo vedere grazie al comando “job.print()”, infine, reimposto i bottoni in modo che essi siano di nuovo visibili.

### Metodo che rende invisibili i bottoni nella pagina

Questo metodo, come abbiamo visto in precedenza, serve a impostare la pagina per esportare il documento in diversi formati, come per esempio in PNG oppure in stampa direttamente, per questo motivo, devo rendere invisibili i bottoni presenti.

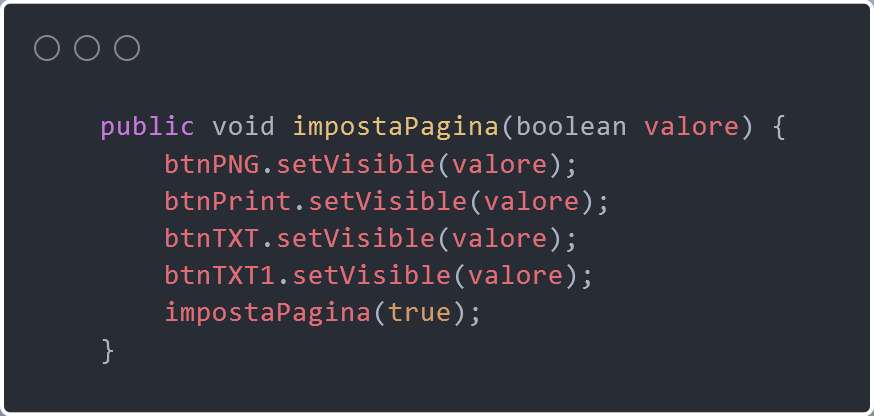


Figura 31 – Classe MainForm, Metodo impostaPagina

## Classe GameTableModel

Questa classe definisce la struttura per la tabella dove viene mostrato il gioco.  
In essa troviamo diversi metodi utili per lavorare con le tabelle, per far funzionare questa classe ci basterà estendere la classe alla AbstractTableModel, una classe astratta che contiene l’elenco di diversi metodi di manipolazioni delle tabelle.

La tabella che verrà istanziata con questo modello utilizza una matrice che corrisponde alla tabella stessa, manipolando la matrice si manipolerà anche la tabella.

In poche parole, questa matrice corrisponde alla nostra tabella in formato di array e non visuale.

### Metodo per impostare le lettere

Questo metodo permette di impostare le lettere ad una posizione ben precisa, in realtà, quando si richiama questo metodo viene modificato solamente la matrice della tabella e viene avvisato il componente Jtable della modifica effettuata.

Il codice per questo metodo è il seguente:

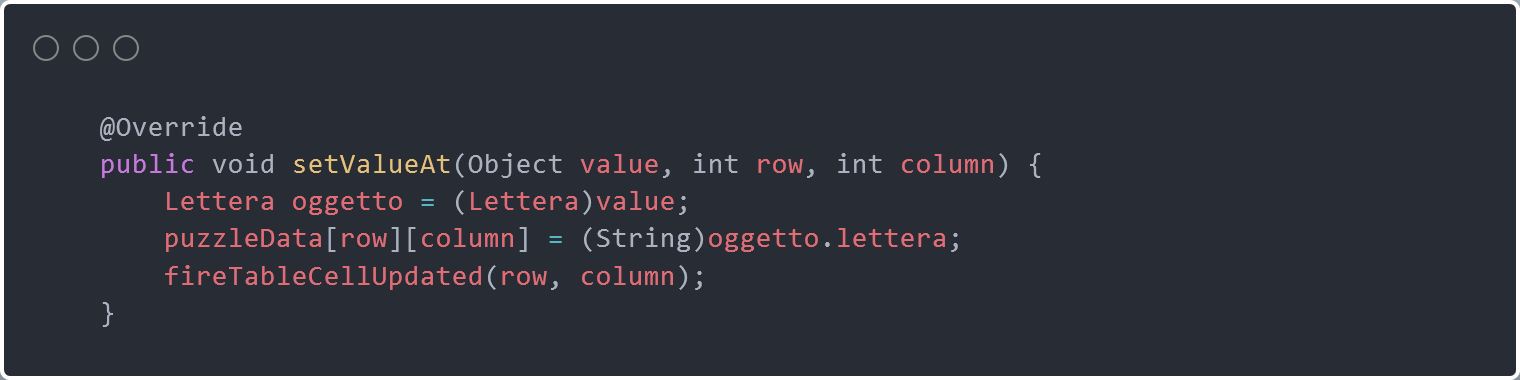


Figura 30 – Classe GameTableModel, Metodo setValueAt

## Classe WordsTableModel

Questa classe, come quella di prima, ha solamente una differenza, la grandezza della matrice, questo perché se nella classe precedente “GameTableModel” avevo bisogno della matrice come grandezza del campo, qui devo invece separare tutte le parole in 4 colonne, infatti, avrò due parametri nel metodo costruttore che corrispondono al numero di righe e al numero delle colonne della matrice.

### Metodo per impostare le parole

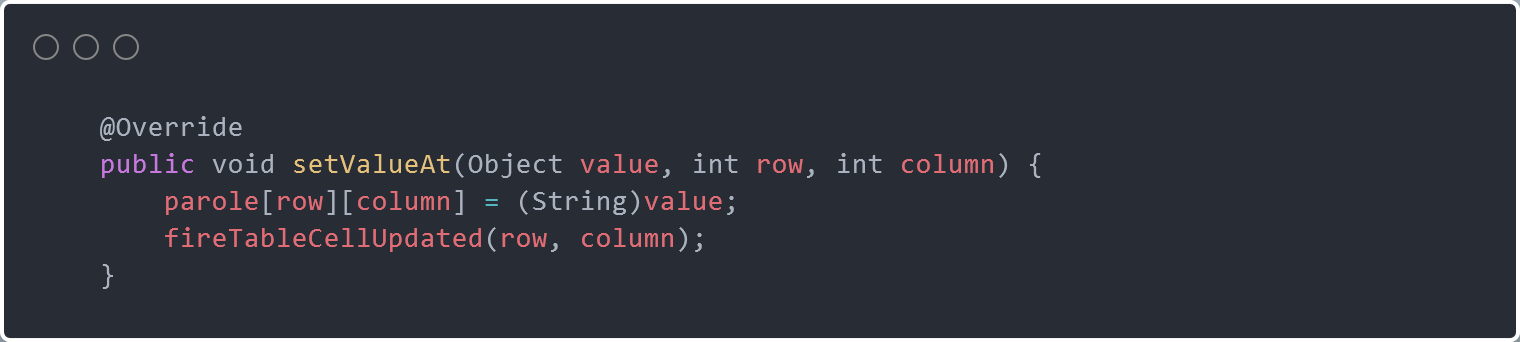
Questo metodo imposta le parole alla riga e colonna selezionata e passata tramite parametro del metodo.

Figura 31 – Classe WordsTableModel, Metodo setValueAt

## Classe SoluzioneForm

Questa classe permette all’utente di visualizzare le soluzioni del gioco, mostrerà semplicemente all’utente una tabella composta di due colonne che conterrà tutte le parole con le seguenti informazioni:

* Riga di partenza
* Colonna di partenza
* Direzione della parola (Verticale, Orizzontale, ObliquaD (/), ObliquaS (\))

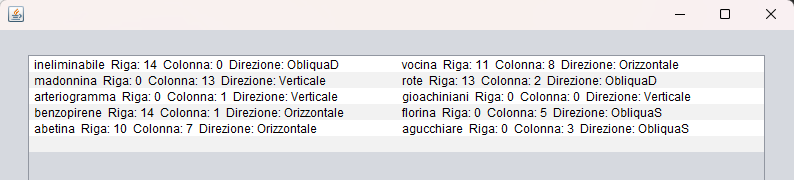


Figura 31 – Classe SoluzioneForm, GUI

### Metodo generazione tabella

Questo metodo divide le parole in due colonne e ne imposta il valore alla corrispondente riga e colonna.

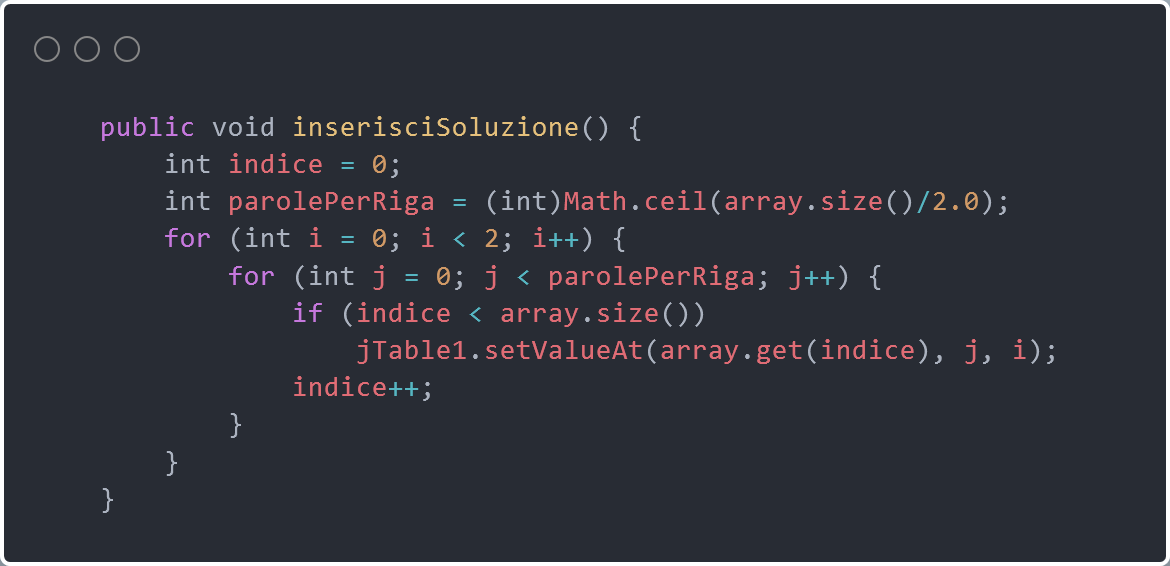


Figura 32 – Classe SoluzioneForm, metodo inserisciSoluzione

Come possiamo notare, il metodo è molto semplice nel complesso, l’unica operazione difficile da capire è il valore della variabile “parolePerRiga” che è semplicemente il valore della divisione tra il numero di parole ed il numero di colonne approssimato all’eccesso, quindi, con il risultato **2.5**, la variabile imposterà il valore **3**.  
  
Successivamente, aggiungo le parole all’interno della tabella impostata con lo stesso modello della tabella delle parole “WordsTableModel” inizializzato con il valore 2 come numero di colonne.

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-001 | **Nome:** | L’applicazione deve essere multipiattaforma |
| **Descrizione:** | Avviare una VM con Linux e verificare il funzionamento del programma. | | |
| **Prerequisiti:** | VM con OS Linux ed installato Java | | |
| **Procedura:** | 1. Aprire la VM 2. Importare il Progetto all’interno della VM 3. Avviare il Progetto e verificare il funzionamento | | |
| **Risultati attesi:** | Il programma funziona senza problemi | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-002 | **Nome:** | Interfaccia grafica completa |
| **Descrizione:** | GUI con scelte varie e possibilità, una volta generato il campo, ci sia la possilbità di export nei 3 modi definiti. | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Verificare interfaccia grafica input completa e funzionante 3. Generare il campo 4. Verificare interfaccia grafica completa e funzionante | | |
| **Risultati attesi:** | Interfaccia Grafica funzionante senza problemi | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-003 | **Nome:** | Verifica funzionamento modalità bambini difficoltà facile |
| **Descrizione:** | Verificare che la modalità di gioco “Bambini” alla difficoltà “Facile” funzioni correttamente | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Selezionare la modalità di gioco “Bambini” 3. Selezionare la difficoltà facile 4. Cambiare la grandezza del campo alla minima disponibile 5. Generare il campo e controllare il numero di parole (10) | | |
| **Risultati attesi:** | La modalità e la difficoltà scelta funzioni correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-003 | **Nome:** | Verifica funzionamento modalità bambini difficoltà media |
| **Descrizione:** | Verificare che la modalità di gioco “Bambini” alla difficoltà “Media” funzioni correttamente | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Selezionare la modalità di gioco “Bambini” 3. Selezionare la difficoltà media 4. Cambiare la grandezza del campo alla minima disponibile 5. Generare il campo e controllare il numero di parole (15) | | |
| **Risultati attesi:** | La modalità e la difficoltà scelta funzioni correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-003 | **Nome:** | Verifica funzionamento modalità bambini difficoltà difficile |
| **Descrizione:** | Verificare che la modalità di gioco “Bambini” alla difficoltà “Difficile” funzioni correttamente | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Selezionare la modalità di gioco “Bambini” 3. Selezionare la difficoltà difficile 4. Cambiare la grandezza del campo alla minima disponibile 5. Generare il campo e controllare il numero di parole (20) | | |
| **Risultati attesi:** | La modalità e la difficoltà scelta funzioni correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-003 | **Nome:** | Verifica funzionamento modalità normale |
| **Descrizione:** | Verificare che la modalità di gioco “Normale” funzioni correttamente | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Selezionare la modalità di gioco “Normale” 3. Cambiare la grandezza del campo alla minima disponibile 4. Schiacciare il pulsante per generare il campo 5. Inserire la parola nascosta 6. Verificare la corretta generazione del campo con i vari requisiti inseriti dall’utente | | |
| **Risultati attesi:** | La modalità e la difficoltà scelta funzioni correttamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-004 | **Nome:** | Test input utente FONT |
| **Descrizione:** | Verificare che l’utente possa cambiare il FONT dell’applicazione | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sulla selezione del FONT e verificare che si possa scegliere un font diverso. 3. Verificare che il FONT sia cambiato | | |
| **Risultati attesi:** | Funzionamento corretto della procedura | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-004 | **Nome:** | Test input utente FILE corretto |
| **Descrizione:** | Verificare che l’utente possa cambiare il FILE delle parole dell’applicazione | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sulla selezione del FILE e verificare che si possa scegliere un file diverso in formato TXT 3. Verificare che il gioco sia stato generato con il file passato | | |
| **Risultati attesi:** | Funzionamento corretto della procedura | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-009  REQ-004 | **Nome:** | Test input utente FILE non corretto |
| **Descrizione:** | Verificare che l’utente possa cambiare il FILE delle parole dell’applicazione passando un file non corretto | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sulla selezione del FILE e verificare che si possa scegliere un file (non conforme) 3. Verificare che il gioco sia stato generato con il file di default oppure che generi senza generare errori | | |
| **Risultati attesi:** | Funzionamento corretto della procedura | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-005 | **Nome:** | Test bordi Campo |
| **Descrizione:** | Verificare che il rettangolo di gioco venga rappresentato con un bordo. | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Verificare che il rettangolo di gioco abbia il bordo | | |
| **Risultati attesi:** | Il bordo è presente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-011  REQ-005 | **Nome:** | Test lista parole in colonne e ordine alfabetico |
| **Descrizione:** | Verificare che le parole siano rappresentate in Colonna e non in una sola lista in ordine alfabetico | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Verificare che le parole siano divise in colonna | | |
| **Risultati attesi:** | Parole in Colonna in ordine alfabetico | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-012  REQ-006 | **Nome:** | Test orientamento parola orizzontale |
| **Descrizione:** | Verificare che le parole vengano generate in orizzontale e in modo completo | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante che mostra la soluzione 4. Cercare una parola con orientamento orizzontale 5. Verificare all’interno del campo che la parola sia presente, in orizzontale e completa | | |
| **Risultati attesi:** | Parola presente, complete e nella direzione corretta | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-013  REQ-006 | **Nome:** | Test orientamento parola verticale |
| **Descrizione:** | Verificare che le parole vengano generate in verticale e in modo completo | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante che mostra la soluzione 4. Cercare una parola con orientamento verticale 5. Verificare all’interno del campo che la parola sia presente, in verticale e completa | | |
| **Risultati attesi:** | Parola presente, complete e nella direzione corretta | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-014  REQ-006 | **Nome:** | Test orientamento parola obliqua Sinistra “\” |
| **Descrizione:** | Verificare che le parole vengano generate in obliquo e in modo completo | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante che mostra la soluzione 4. Cercare una parola con orientamento obliquo 5. Verificare all’interno del campo che la parola sia presente, in obliquo e completa | | |
| **Risultati attesi:** | Parola presente, complete e nella direzione corretta | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-015  REQ-006 | **Nome:** | Test orientamento parola obliqua Destra “/” |
| **Descrizione:** | Verificare che le parole vengano generate in obliquo e in modo completo | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante che mostra la soluzione 4. Cercare una parola con orientamento obliquo 5. Verificare all’interno del campo che la parola sia presente, in obliquo e completa | | |
| **Risultati attesi:** | Parola presente, complete e nella direzione corretta | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-016  REQ-007 | **Nome:** | Test Export PNG |
| **Descrizione:** | Verificare che venga esportato correttamente il Progetto in formato PNG | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante di esportazione in PNG 4. Inserire il nome del file 5. Verificare che nella cartella dove si trova l’eseguibile ci sia anche il file .png con il campo corretto | | |
| **Risultati attesi:** | File generato correttamente e presente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-017  REQ-007 | **Nome:** | Test Export TXT |
| **Descrizione:** | Verificare che venga esportato correttamente il Progetto in formato TXT | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante di esportazione in TXT 4. Inserire il nome del file 5. Verificare che nella cartella dove si trova l’eseguibile ci sia anche il file .txt con il campo corretto | | |
| **Risultati attesi:** | File generato correttamente e presente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-018  REQ-007 | **Nome:** | Test Stampa applicazione |
| **Descrizione:** | Verificare che venga stampato il risultato finale dell’applicazione | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante stampa 4. Verificare che la stampante abbia stampato, oppure stampare “Microsoft Print To PDF” e verificare che il file si sia salvato in formato PDF | | |
| **Risultati attesi:** | File stampato correttamente o presente come .PDF | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-019  REQ-008 | **Nome:** | Test soluzione |
| **Descrizione:** | Verificare che la soluzione del gioco sia presente e visualizzabile dall’utente | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Avviare l’app 2. Cliccare sul pulsante di generazione 3. Cliccare sul pulsante per mostrare le soluzioni 4. Verificare che le soluzioni siano presenti e siano corrette | | |
| **Risultati attesi:** | Apertura della finestra contenente le soluzioni senza errori e soluzioni complete | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-020  REQ-009 | **Nome:** | Test nessun margine d’errore |
| **Descrizione:** | Verificare che non ci siano errori nella generazione del campo, delle parole, dell’applicazione in sé | | |
| **Prerequisiti:** | Applicazione funzionante | | |
| **Procedura:** | 1. Verificare che tutti i test siano andati a buon fine 2. Far testare l’applicazione ad un’altra persona. | | |
| **Risultati attesi:** | Apertura della finestra contenente le soluzioni senza errori e soluzioni complete | | |

## Risultati test

|  |  |  |
| --- | --- | --- |
| ID TEST | COMMENTI | RISULTATO |
| TC-001 | Figura 33– TEST, KALI Linux. | PASSATO |
| TC-002 | La GUI si presenta in modo completo (con il rettangolo di gioco, i pulsanti di export,…). | PASSATO |
| TC-003 | La modalità bambini, difficoltà facile funziona correttamente, essa genera un campo con all’interno 10 parole da trovare. | PASSATO |
| TC-004 | La modalità bambini, difficoltà media funziona correttamente, essa genera un campo con all’interno 15 parole da trovare. | PASSATO |
| TC-005 | La modalità bambini, difficoltà difficile funziona correttamente, essa genera un campo con all’interno 20 parole da trovare. | PASSATO |
| TC-006 | La modalità normale correttamente, essa richiede una parola segreta di x lettere che verrà poi inserita negli spazi vuoti del campo, la quantità di parole varia da generazione a generazione. | PASSATO |
| TC-007 | Cambiando il font dell’applicazione, il campo e le parole acquisiscono il font selezionato in modo corretto. | PASSATO |
| TC-008 | Cambiando il file delle parole ed inserendo un file corretto, il programma genera perfettamente un campo in base alle parole passate all’interno del file. | PASSATO |
| TC-009 | Passando al programma un file non corretto, esso genera il campo solamente con le parole che riesce a trovare all’interno del file, altrimenti, prende quello di default. | PASSATO |
| TC-010 | L’applicazione, una volta generata, genera anche i bordi del campo come possiamo ben vedere anche dall’immagine relativa al TC-001 | PASSATO |
| TC-011 | L’applicazione, una volta generata, genera correttamente anche le parole, dividendole per colonne da 4 e ordinandole in ordine alfabetico. | PASSATO |
| TC-012 | Le parole vengono generate in direzione orizzontale in modo corretto e completo.  Figura 34 - Test | PASSATO |
| TC-013 | Le parole vengono generate in direzione verticale in modo corretto e completo.  Figura 35 – Test | PASSATO |
| TC-014 | Le parole vengono generate in direzione obliquo sinistra in modo corretto e completo.  Figura 36 - Test | PASSATO |
| TC-015 | Le parole vengono generate in direzione obliquo destra in modo corretto e completo.  Figura 37 - Test | PASSATO |
| TC-016 | Una volta generato il programma, l’esportazione in formato .png funziona correttamente, generando il risultato.  Figura 38 - Test | PASSATO |
| TC-017 | Una volta generato il programma, l’esportazione in formato .txt funziona correttamente, generando il risultato.  Figura 39 -Test | PASSATO |
| TC-018 | Una volta generato il programma, la stampa di esso avviene in modo completamente funzionante.  Figura 40 - Test | PASSATO |
| TC-019 | La soluzione del test viene mostrata all’utente in modo completo e dettagliato.  Figura 41 - Test | PASSATO |
| TC-020 | Non ci sono errori in quanto testato, per questo motivo anche l’ultimo requisito è stato soddisfatto con successo. | PASSATO |

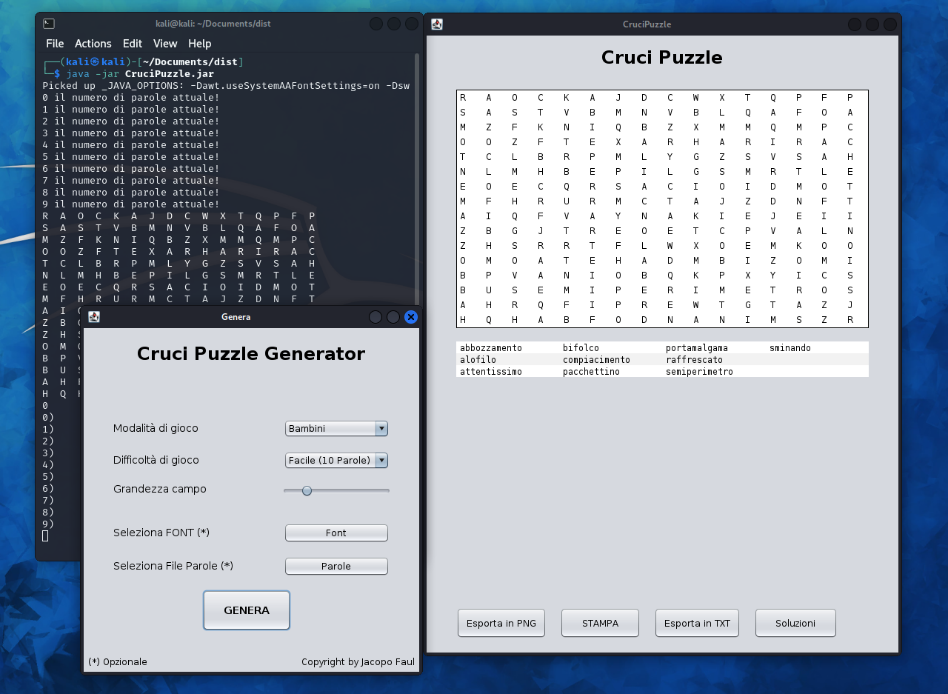
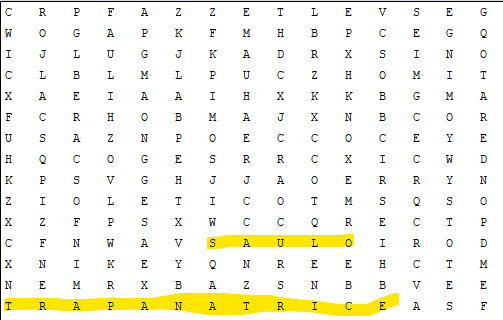


Figura 33 – Risultati TEST, TC 001



Figura 34 – Risultati TEST, TC 012

*\*****n.b.*** *L’evidenziazione sulle parole non fanno parte dell’app*

Figura 35 – Risultati TEST, TC 013



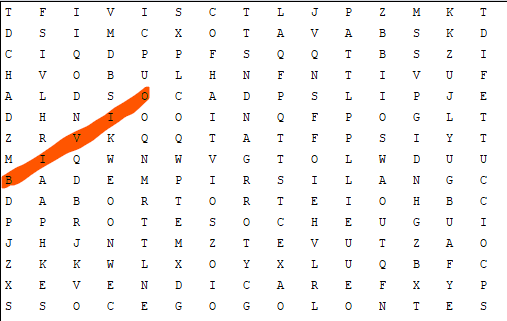


Figura 36 – Risultati TEST, TC 014

Figura 37 – Risultati TEST, TC 015

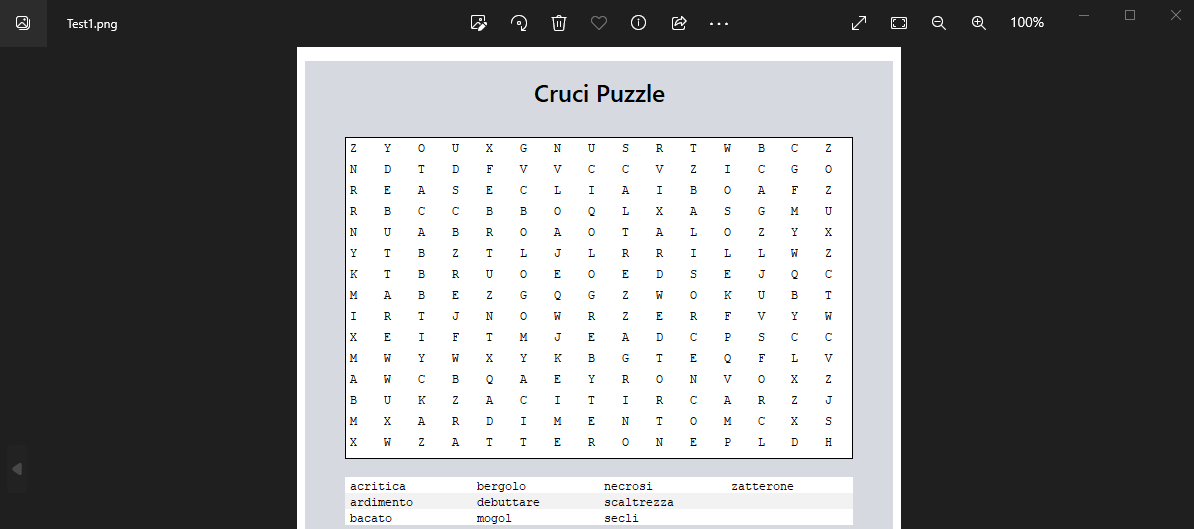


Figura 38 – Risultati TEST, TC 016

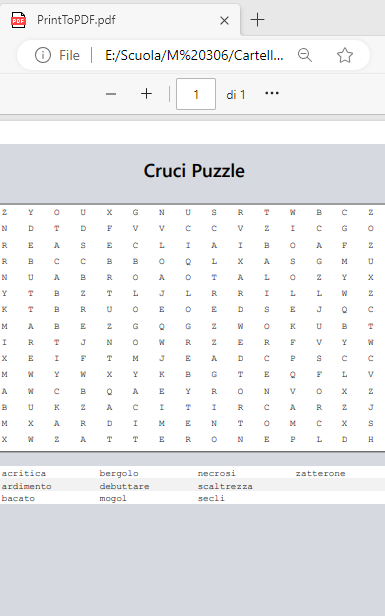


Figura 40 – Risultati TEST, TC 018

Figura 39 – Risultati TEST, TC 017

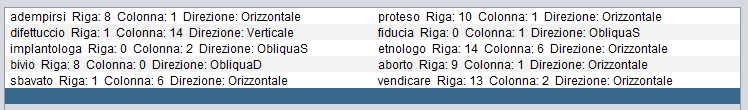


Figura 41 – Risultati TEST, TC 019

## Mancanze/limitazioni conosciute

In realtà devo dire che avrei voluto implementare la soluzione grafica, non solamente scritta in una finestra a parte, ma avrei voluto mostrare il campo con dei colori sulle varie parole e cerchiando la parola nascosta, purtroppo non sono riuscito ad implementare questa parte causa mancanza di tempo.

# Consuntivo

Figura 42 - Consuntivo

Come si può notare, i miei progressi non sono andati proprio come pensato all’inizio.

# Conclusioni

La mia soluzione per questa applicazione di sicuro non è perfetta, ma mi ha aiutato a sviluppare una maggior tecnica mentre programma, risolvendo i problemi in modo procedurale e con più logica e furbizia.

Mettendomi alla prova, questo progetto è riuscito a spronarmi a voler raggiungere tutti gli obbiettivi (requisiti) predisposti per il software e arrivare al punto di essere soddisfatto del proprio lavoro.

Questo progetto mi ha reso a conoscenza delle mie capacità e le ha affinate.

## Sviluppi futuri

Non penso implementerò questo progetto, lo terrò come ricordo come primo progetto completo mai realizzato in tutta la mia vita.

## Considerazioni personali

Da questo progetto, come detto prima, ho imparato molte altre tecniche, a risolvere i problemi in modo più semplice e non complicandomi la vita, mi ha inoltre insegnato che ogni minuto è importante e che ogni dettaglio conta.

# Link Utili

## Sitografia

* <https://docs.oracle.com/javase/8/docs/technotes/guides/>
* <https://stackoverflow.com/>
* <https://www.w3schools.com/>

# Glossario

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| Problem Solving | È il processo per raggiungere un obiettivo superando gli ostacoli |

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o QdC
* Prodotto
* …