Risoluzione automatica del cruciverba

Progetto di Esperienze di programmazione

Davide Coffaro Matricola 556603

2020

**DESCRIZIONE DEL PROBLEMA**  
Il problema di “Risoluzione automatica del cruciverba” è quello di riempire lo schema dato (con caselle bianche e caselle nere) con le parole disponibili, partendo da un’unica parola inserita nello schema.  
Bisogna ricordare che ogni inserimento influisce sugli inserimenti successivi dato che le parole sono collegate fra loro e infatti si potrà notare che gli algoritmi che fanno controlli riguardanti questa cosa riescono a completare lo schema del cruciverba sempre mentre gli altri solo in determinate situazioni in cui non si vengono a creare conflitti di più parole inseribili nello stesso spazio.

è possibile anche ampliare la lista di parole disponibili con parole che non dovranno essere inserite perché non permettono di completare lo schema ma questo fa sì che si possano verificare più frequentemente dei blocchi dovuti a conflitti fra più parole che possono essere inserite nello stesso spazio e per cui non è possibile completare lo schema. Viene risolto questo problema con un algoritmo che possa provare ad inserire una parola e nel caso di non completamento del cruciverba possa tornare indietro e inserire un’altra parola fino a trovarne la soluzione.

**ANALISI DEL PROBLEMA**È possibile trovare un limite inferiore alla complessità del problema di “Risoluzione automatica del cruciverba” scomponendo il problema in sottoproblemi, che saranno:

con n=numero parole da completare dello schema  
e m=lunghezza parola arrotondata alla parola più lunga per gestire il caso pessimo

* Inserire parole nello schema -> L\_inserimento(n-1)
* Cerca fra le parole disponibili da inserire -> L\_ricerca(n)
* Confronta le lettere delle caselle con quella della parola disponibile corrente -> L\_confronta(m)

E approssimando ulteriormente le m lettere della parola di lunghezza massima al valore di n -> m=O(n) e ricomponendo il problema ho che:

L\_risoluzioneCruciverba=Ω( L\_inserimento(n-1) \* L\_ricerca(n) \* L\_confronta(n) )= Ω( n^3 )

Mail professoressa pagli -> banalmente la dimensione del cruciverba omega(n^2)

**ALGORITMI PER RISOLVERE IL PROBLEMA**

* Algoritmo1 - parto dallo schema, cerco la parola da inserire fino a completare il cruciverba:

INPUT matrice rappresentante lo schema iniziale + lista parole da inserire + parola iniziale nello schema  
OUTPUT cruciverba completato o no (true o false)

PROCEDURA ALGORITMO1:

1. Inizializzo trovato(i)=false per i=2…n, questo indicherà per le parole con i caselle se sono state trovate tutte le parole.
2. Finche trovato(i)=true per ogni i=2…n, cicliEseguiti<=cicliMax;
3. Incremento i cicliEseguiti di 1. Prendo le parole dello schema di c numero di caselle e le inserisco in una lista (controllando solo le parole ancora da completare), con c=2…n, se c=n+1 torno al punto 2);
4. Ciclo fino a che la lista di parole trovate di c caselle non è vuota, se è vuota faccio un controllo se ho inserito nello schema tutte le parole di lunghezza c aggiorno trovato(c)=true e torno al punto 3) ricercando parole con numero di caselle c+1;
5. A parità di c caselle prendo la parola x con più lettere già inserite;
6. Controllo quali parole di c lettere del dizionario possono essere inserite nello spazio della parola x con c caselle, facendo il confronto dei caratteri lettere-caselle;
7. Se è stata trovata una parola la inserisco nello schema, aumento di 1 il numero di parole di lunghezza c inserite, aggiorno il dizionario (perché alcune parole potrebbero essersi completate automaticamente) e torno al punto 4), altrimenti torno al punto 4), prendendo la prossima parola di c caselle se presente.
8. setto variabile di esecuzione algoritmo=true e se trovato(i)=true per i=2…n setto anche risoluzione algoritmo=true, altrimenti risoluzione algoritmo=false

CONDIZIONI DI TERMINAZIONE:  
1) trovato(i)=true per i=2…n;  
2) cicliEseguiti>cicliMax.

COMPLESSITA’ algoritmo1:  
analizzando l’algoritmo si può notare che ci sono vari cicli annidati necessari per poter completare lo schema e possono dipendere da:   
- cicliMax indicati con n, i cicli massimi superati i quali l’algoritmo termina determinando il completamento o non dello schema;   
- lunghezzaMax indicati con L, la lunghezza massima possibile per le parole dello schema, per il momento impostata a priori dal programma;  
- parole schema e parole disponibili entrambi indicati con m, che indicano il numero delle parole dello schema e quelle della lista di parole da inserire.  
Per approssimare al caso peggiore imposto sia L=O(n) che m=O(n), quindi sia L che m sono dell’ordine di n, numeri molto alto rispetto agli altri input e trovo che la complessità dell’algoritmo1 è O(n^5)

* Algoritmo2 – procedura simile all’algoritmo1 ma:

1. Utilizzo di lista(i) che contengono le parole di lunghezza i per i=1…n
2. Quando inserisco la parola di lunghezza i nello schema, la elimino anche dalla sua lista(i)
3. Quando lista(i) è vuota per i=1…n termino il ciclo

Problemi relativi all’algoritmo2

* + Riempire lista(i) all’inizio durante la creazione delle caselle ecc…
  + Memoria occupata da listaParolaSchema, listaParoleSchemaDaCompletare, listaParoleDaInserire (dizionario input)

COMPLESSITA’ algoritmo2:  
Anche per questo algoritmo ho diversi cicli annidati e al caso peggiore ha complessità O(n^5) così come l’algoritmo1

* Algoritmo4 (con AI) - utilizzo di algoritmi di intelligenza artificiale per cercare la soluzione in modo più efficiente, correggendo gli errori per cui gli altri algoritmi si bloccavano ad un certo punto non trovando altre parole da inserire nello schema del cruciverba. Con questi algoritmi (che sfruttano la strategia CSP – Constraints Satisfaction Problem) analizzo le parole che possono essere inserite nello schema e nel caso in cui arrivassi ad un punto in cui non è possibile inserire altre parole ma lo schema non è ancora completo, si possa tornare indietro e provare le parole alternative fino al completamento dello stesso.

PROCEDURA ALGORITMO4:

al momento della creazione dello schema inserisco tutte le parole in orizzontale o in verticale (non ancora completate) all’interno di una lista di variabili; dopo analizzo i possibili valori che possono essere inseriti all’interno delle variabili inserendo anch’essi in una lista collegata alle singole variabili, creando così tanti domini delle parole da inserire nello schema, uno per ogni variabile.  
  
Passi di soluzione dell’algoritmo:  
1) ricerca della variabile a cui assegnare un valore (STRATEGIA MRV e grado maggiore a parità di MRV); MRV cioè analizzo i valori residui nel dominio di ogni variabile e prendo quelle che hanno il numero minore di valori; se sono presenti più variabili con lo stesso numero di valori nel dominio allora viene presa la variabile che è vincola più variabili ad essa collegata (cioè prendo la variabile con lunghezza maggiore perché coinvolgerà un maggior numero di variabili ad essa collegata);

2) inserimento di un valore del dominio della variabile all’interno della variabile (STRATEGIA di scelta valore per la variabile attuale in ordine di come sono stati inseriti i valori nel dominio di quella variabile);

3) meccanismo di inferenza in cui vengono ridotti i domini delle variabili collegate a quella corrente e delle variabili con lo stesso numero di lettere, in caso di dominio di una variabile collegata o di dominio di una variabile con lo stesso numero di lettere vuoto e schema del cruciverba non ancora completo, significa che uno degli assegnamenti di valori alle variabili precedentemente effettuato non era corretto e bisogna fare dei passi all’indietro provando un altro valore (sfrutta il backtracking cronologico ed inserisce il valore successivo del dominio della stessa variabile);

4)se tutti i passi precedenti sono andati a buon fine proseguo la ricerca sul sottoproblema in cui adesso ho assegnato un valore alla variabile su cui stavo lavorando.

CONDIZIONI DI TERMINAZIONE:

1. quando ho effettuato un numero di assegnamenti uguale al numero di variabili;
2. quando non esistono più variabili a cui assegnare un valore.

COMPLESSITA’ algoritmo4 con AI:

considerato che viene fatta una chiamata ricorsiva per ogni variabile all’interno dello schema del cruciverba avremo una complessità esponenzial. Questo è dovuto dal fatto che la struttura del grafo dei vincoli derivante dall’implementazione dell’algoritmo CSP è complessa e potrebbe essere ridotta andando ad utilizzare delle tecniche di riduzione del grafo iniziale dei vincoli verso una struttura ad albero risolvibile con complessità minore (tree structured, cutset conditioning e tree decomposition).

Per ogni chiamata a BACKTRACK ho:

1. n\*c operazioni per selectUnassignedVariable
2. ciclo di d iterazioni = d\*c operazioni per scorrere valori presi da orderDomainValues
3. 4\*n\*c operazioni per inference
4. Chiamata ricorsiva backtrack con n-1 variabili

Complessità algoritmo4\_AI -> T(n) = n\*c + d\*c\*(4\*n\*c+T(n-1)) per n>0

= 0 per n=0

E dovendo fare n chiamate ricorsive risolvo l’albero di ricorsione che avrà altezza n e in cui ogni nodo avrà costo j\*c+d\*c\*j con j=numero variabili ancora da assegnare

Complessità algoritmo4\_AI = O (c+c\*sommatoria per i=1..n di ( d^i)) = O(d^n)

**SCELTE IMPLEMENTATIVE**

Nell’implementazione del cruciverba ho deciso di scomporre lo schema del cruciverba nelle parole e nelle caselle nere, all’interno delle parole ho la stringa della parola associata che uso per cercare la parola da inserire nello schema ma scompongo ulteriormente la parola nelle singole caselle in cui ho il carattere ad esse collegate che vengono poi stampate a video. Ho deciso di utilizzare come struttura dati l’ArrayList per tutte le collezioni di oggetti utilizzate dal programma (perché struttura dati di dimensione variabile) e di utilizzare varie classi per implementare il cruciverba:

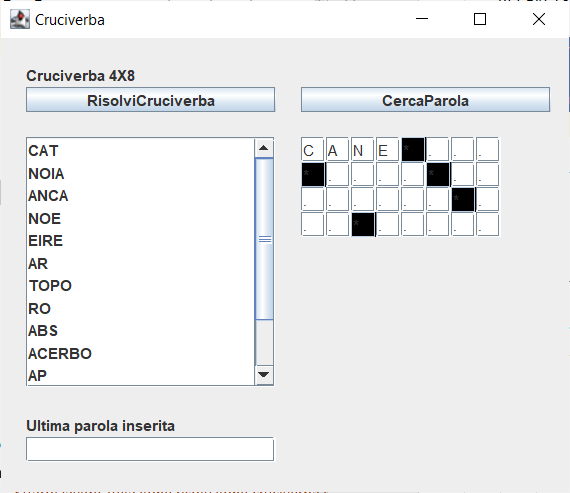
* InterfacciaCruciverba, è la classe che contiene il metodo Main e che viene utilizzata per inizializzare i componenti grafici della maschera del programma. Contiene anche i vari esempi creati suddivisi per gli algoritmi implementati. L’interfaccia utente del cruciverba è così composta: una lista di parole scorribili che possono essere inserite nello schema; lo schema del cruciverba con caselle bianche e nere e la prima parola inserita al suo interno; il pulsante “RisolviCruciverba” esegue l’algoritmo fino a completamento dello schema o fino a che non si blocca per cui non ha trovato una soluzione, comunque inserendo tutte le parole contemporaneamente e inviando un messaggio a video all’utente per segnalare il risultato del cruciverba; il pulsante “CercaParola” invece esegue un solo passo dell’algoritmo ed inserire una parola alla volta nello schema del cruciverba, inserendo anche l’ultima parola inserita nell’apposita casella sotto la lista di parole che possono essere inserite nello schema.
* ImplementazioneCruciverba, classe che contiene lo schema del cruciverba, la lista delle parole da inserire, lo stato di esecuzione dell’algoritmo di risoluzione e il risultato dell’esecuzione; implementa le procedure di aggiornamento dello schema del cruciverba, di aggiornamento della lista delle parole da inserire e di controllo completamento o meno del cruciverba; viene estesa dagli algoritmi di risoluzione del cruciverba (alg1, alg2, alg4\_AI).
* Schema, contiene le parole (completate o meno) inserite nello schema del cruciverba e le caselle dello schema stesso, si occupa della creazione dello schema a partire dalla matrice in input e dell’inserimento nello schema della parola iniziale, collega le caselle dello schema alla lista delle parole dello schema, facendo si che l’aggiornamento di uno sia collegato all’altro; contiene altre procedure utili per analizzare le proprietà dello schema o delle parole al suo interno.
* Parola, ha al suo interno i riferimenti della stringa, caselle dello schema associate, posizione all’interno dello schema, orientamento orizzontale o verticale, lunghezza parola e numero di lettere inserite; può aggiornare le caselle dello schema con la stringa attuale, oppure il contrario e aggiornare la stringa con le caselle dello schema attuali. Può verificare se due parole si riferiscono alle stesse caselle dello schema;
* Casella, si occupa della creazione del componente grafico della casella che viene visualizzata a video e della relativa posizione all’interno dello schema. Contiene il carattere associato alla casella e se è una casella nera (in cui non è possibile scrivere) o no; ha al suo interno procedure per controllo posizione della casella e aggiornamento carattere della casella.
* Posizione, contiene i riferimenti riga-colonna nello schema.
* Il dizionario delle parole disponibili/da inserire alla fine dell’elaborazione non è vuoto ma risulta contenere alcune parole le cui lettere sono state tutte inserite durante l’inserimento di altre parole, quindi la parola è completa ma non è stata inserita nello schema tramite la procedura degli algoritmi. Questo fa sì che ci sia una procedura di aggiornamento del dizionario delle parole disponibili (aggiornaDizionario) che cicla tutte le parole inserite nello schema che sono complete e le elimina dal dizionario se sono contenute al suo interno. AUMENTA LA COMPLESSITA’ dell’algoritmo/algoritmi di un ordine n.

**RISULTATI OTTENUTI**

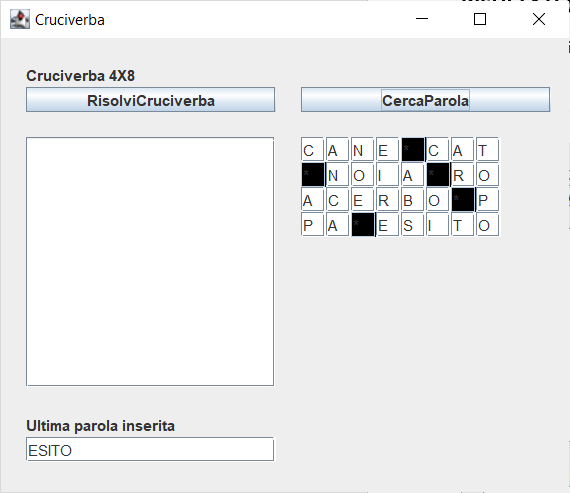
Questi sono gli esempi provati suddivisi per i 3 algoritmi implementati:

Esempio1:

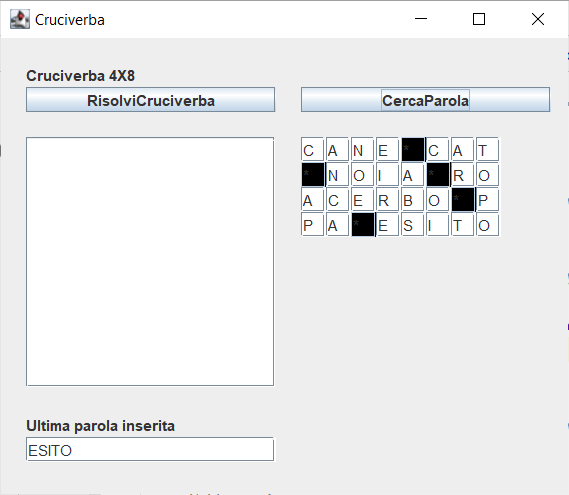
all’apertura del programma



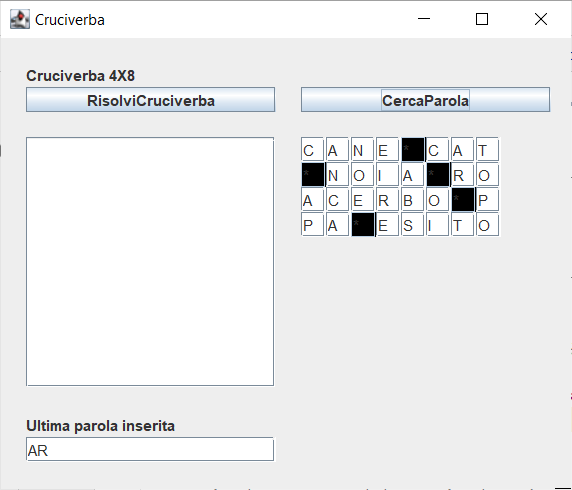
Dopo esecuzione algoritmo1



Dopo esecuzione algoritmo2

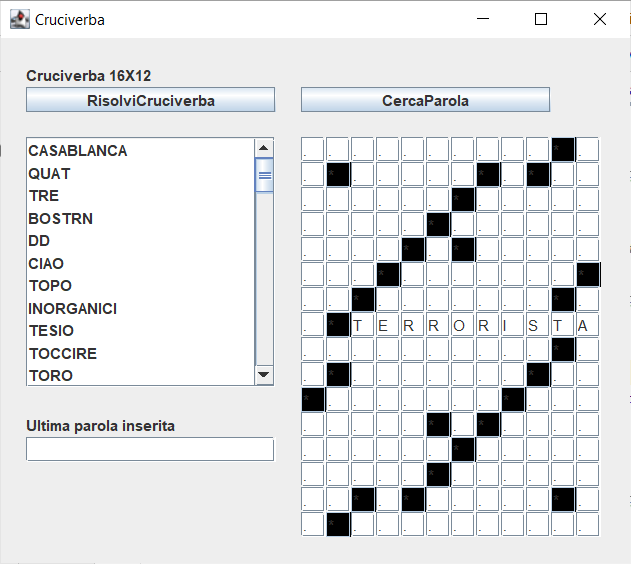


Dopo esecuzione algoritmo4\_AI

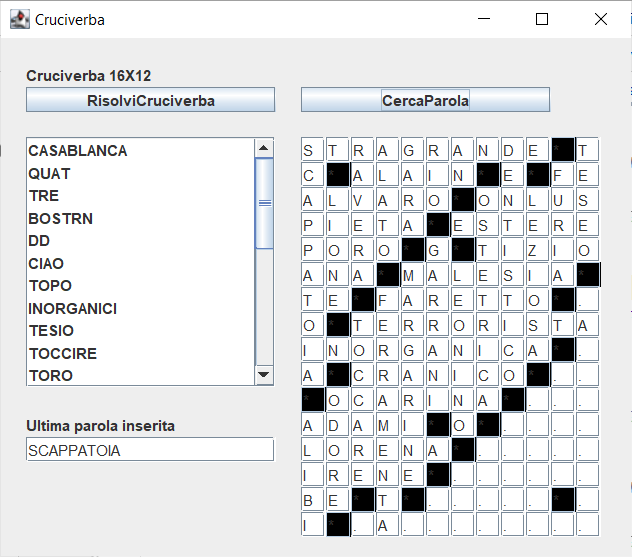


Esempio2:

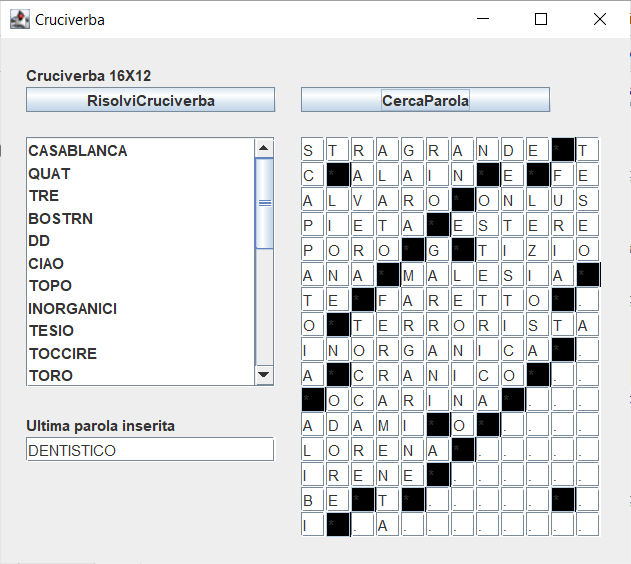
all’apertura del programma



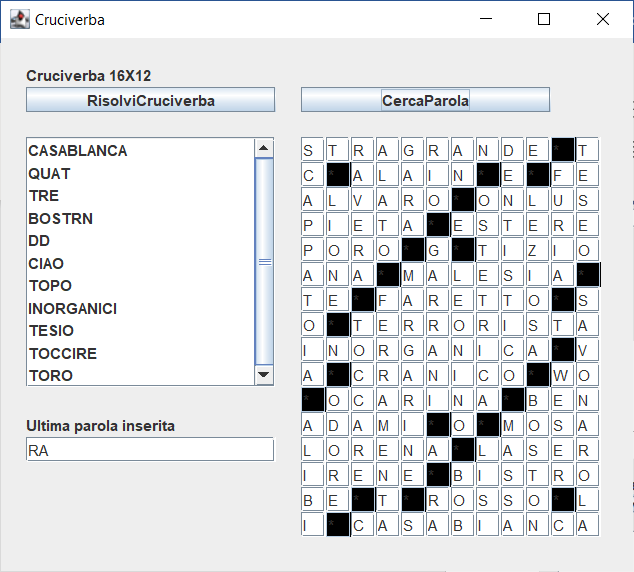
Dopo esecuzione algoritmo1



Dopo esecuzione algoritmo2



Dopo esecuzione algoritmo4\_AI

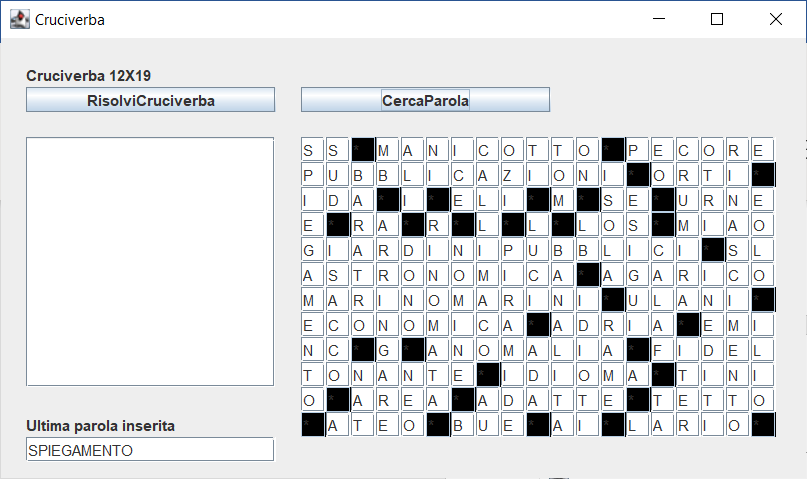


Esempio3:

all’apertura del programma



Dopo esecuzione algoritmo1



Dopo esecuzione algoritmo2



Dopo esecuzione algoritmo4\_AI



* Come possiamo notare dall’esempio 2 l’algoritmo1 e l’algoritmo2 non trovano sempre la soluzione, ma in alcuni casi in cui è possibile inserire più parole nelle stesse caselle non possono continuare la loro risoluzione. Questo problema non è invece presente nell’algoritmo4\_AI in cui se si verifica la precedente situazione l’algoritmo prova prima ad inserire una parola e continua la sua risoluzione, se in un certo momento non riesce ad inserire altre parole ma il cruciverba non è stato ancora completato allora effettua il “backtracking” per cui elimina i valori inseriti dopo aver avuto il blocco per più parole da poter inserire nello schema e prova ad inserire un altro valore nella casella precedentemente riempita ed effettua questa procedura fino al completamento dello schema.

Quindi l’algoritmo1 e l’algoritmo2 hanno una complessità polinomiale mentre l’algoritmo4\_AI ha complessità esponenziale, l’unico problema è che l’algoritmo1 e 2 non sono completi, cioè non riescono sempre a trovare una soluzione anche se questa è presente. l’algoritmo4\_AI è completo, quindi se è presente una soluzione riesce a trovarla in tempo esponenziale, ma non è ottimo perché la soluzione trovata potrebbe non essere quella meno costosa da trovare nell’albero di ricerca delle soluzioni al problema.

APPENDICE: codice

Classe InterfacciaCruciverba:  
**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** org.junit.rules.Stopwatch;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.image.ImageObserver;  
**import** java.text.AttributedCharacterIterator;  
**import** java.util.ArrayList;  
**import** java.lang.Object;  
  
**public class** InterfacciaCruciverba {  
 **private** JPanel **panelMain**;  
 **private** JPanel **panelListaParole**;  
 **private** JLabel **labelCruciverba**;  
 **private** JButton **buttonRisolviCruciverba**;  
 **private** JButton **buttonCercaParola**;  
 **private** JList **listListaParole**;  
 **private** JScrollPane **scrollPaneListaParole**;  
 **private** JLabel **labelParolaInserita**;  
 **private** JTextField **textFieldParolaInserita**;  
 **private static** ImplementazioneCruciverba *cruciverba1*;  
 **private static int** *dimensioneFinestraMinima*=500;  
 **private long startTime**=0;  
 **private long stopTime**=0;  
 **private long totalTime**=0;  
  
 *//inizializzazione della matrici che faranno da base dello schema del cruciverba per i 3 esempi creati  
 /\*//esempio1  
 private static char[][] matrice = {{'.', '.', '.', '.', '\*', '.', '.', '.'},  
 {'\*', '.', '.', '.', '.', '\*', '.', '.'},  
 {'.', '.', '.', '.', '.', '.', '\*', '.'},  
 {'.', '.', '\*', '.', '.', '.', '.', '.'}};  
\*/  
  
 /\*//esempio2  
 private static char[][] matrice = {{'.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '\*', '.'},  
 {'.', '\*', '.', '.', '.', '.', '.', '\*', '.', '\*', '.', '.'},  
 {'.', '.', '.', '.', '.', '.', '\*', '.', '.', '.', '.', '.'},  
 {'.', '.', '.', '.', '.', '\*', '.', '.', '.', '.', '.', '.'},  
 {'.', '.', '.', '.', '\*', '.', '\*', '.', '.', '.', '.', '.'},  
 {'.', '.', '.', '\*', '.', '.', '.', '.', '.', '.', '.', '\*'},  
 {'.', '.', '\*', '.', '.', '.', '.', '.', '.', '.', '\*', '.'},  
 {'.', '\*', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},  
 {'.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '\*', '.'},  
 {'.', '\*', '.', '.', '.', '.', '.', '.', '.', '\*', '.', '.'},  
 {'\*', '.', '.', '.', '.', '.', '.', '.', '\*', '.', '.', '.'},  
 {'.', '.', '.', '.', '.', '\*', '.', '\*', '.', '.', '.', '.'},  
 {'.', '.', '.', '.', '.', '.', '\*', '.', '.', '.', '.', '.'},  
 {'.', '.', '.', '.', '.', '\*', '.', '.', '.', '.', '.', '.'},  
 {'.', '.', '\*', '.', '\*', '.', '.', '.', '.', '.', '\*', '.'},  
 {'.', '\*', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.'}};  
\*/  
  
 //esempio3* **private static char**[][] *matrice* = {{**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**},  
 {**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'\*'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'\*'**,**'.'**,**'\*'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**},  
 {**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**},  
 {**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'.'**,**'\*'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**},  
 {**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'\*'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'\*'**}};  
  
  
 **private** ArrayList<JTextField> **text**;  
 **private static** ArrayList<String> *dizionarioInput*;  
  
 **public** InterfacciaCruciverba() {  
 createUIComponents();  
 **buttonRisolviCruciverba**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **startTime**=System.*currentTimeMillis*();  
  
 *//lancia procedura di risoluzione cruciverba e ritornando se è stata trovata la soluzione o meno, stampando un messaggio a video* **if** (*cruciverba1*.risolviCruciverba()) {  
 **listListaParole**.setListData(*cruciverba1*.**dizionario**.toArray());  
 **stopTime**=System.*currentTimeMillis*();  
 **totalTime**=**stopTime**-**startTime**;  
 JOptionPane.*showMessageDialog*(**null**, **"Cruciverba completato in "** + **totalTime**, **"Risultato cruciverba"**, JOptionPane.***INFORMATION\_MESSAGE***);  
 } **else** {  
 **listListaParole**.setListData(*cruciverba1*.**dizionario**.toArray());  
 JOptionPane.*showMessageDialog*(**null**, **"Cruciverba non completato"**, **"Risultato cruciverba"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 }  
 });  
 **buttonCercaParola**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 *//lancia procedura inserisci1Parola per cui esegue un passo dell'algoritmo e inserisce la parola trovata, se non trova nessuna  
 // parola controlla il risultato del cruciverba* String parolaInserita=*cruciverba1*.inserisci1Parola();  
 **if** (parolaInserita==**null**) {  
 **listListaParole**.setListData(*cruciverba1*.**dizionario**.toArray());  
 **stopTime**=System.*currentTimeMillis*();  
 **totalTime**=**stopTime**-**startTime**;  
 **if** (*cruciverba1*.isAlgResult()) {  
 JOptionPane.*showMessageDialog*(**null**, **"Cruciverba completato in "** + **totalTime**, **"Risultato cruciverba"**, JOptionPane.***INFORMATION\_MESSAGE***);  
 }**else**{  
 JOptionPane.*showMessageDialog*(**null**, **"Cruciverba non completato"**, **"Risultato cruciverba"**, JOptionPane.***ERROR\_MESSAGE***);  
  
 }  
 } **else** {  
 **textFieldParolaInserita**.setText(parolaInserita);  
 **listListaParole**.setListData(*cruciverba1*.**dizionario**.toArray());  
 *//JOptionPane.showMessageDialog(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.ERROR\_MESSAGE);* }  
  
 }  
 });  
 }  
  
 **public static void** main(String[] args) {  
 JFrame frame = **new** JFrame(**"Cruciverba"**);  
 InterfacciaCruciverba window = **new** InterfacciaCruciverba();  
  
 frame.setContentPane(window.**panelMain**);  
 frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 frame.pack();  
  
 *//imposto la dimensione della finestra a seconda del numero di righe e di colonne del cruciverba, se sono troppo basse imposto una  
 //dimensione minima* **int** dimensioneFinestra=20\*(*matrice*.**length**+*matrice*[0].**length**)+100;  
 **if** (dimensioneFinestra<*dimensioneFinestraMinima*){  
 frame.setSize(*dimensioneFinestraMinima*, *dimensioneFinestraMinima*);  
 }**else**{  
 frame.setSize(dimensioneFinestra, dimensioneFinestra);  
 }  
 frame.setVisible(**true**);  
  
 *//creo lista di parole che potranno essere inserite nello schema del cruciverba per i 3 esempi creati  
 /\*//esempio1  
 dizionarioInput = new ArrayList<String>();  
 dizionarioInput.add("CAT");  
 dizionarioInput.add("NOIA");  
 dizionarioInput.add("ANCA");  
 dizionarioInput.add("NOE");  
 dizionarioInput.add("EIRE");  
 dizionarioInput.add("AR");  
 dizionarioInput.add("TOPO");  
 dizionarioInput.add("RO");  
 dizionarioInput.add("ABS");  
 dizionarioInput.add("ACERBO");  
 dizionarioInput.add("AP");  
 dizionarioInput.add("OI");  
 dizionarioInput.add("PA");  
 dizionarioInput.add("ESITO");  
\*/  
  
 /\*//esempio2  
 dizionarioInput = new ArrayList<String>();  
 //per prova parole in più nella lista da inserire  
 dizionarioInput.add("CASABLANCA");  
 dizionarioInput.add("QUAT");  
 dizionarioInput.add("TRE");  
 dizionarioInput.add("BOSTRN");  
 dizionarioInput.add("DD");  
 dizionarioInput.add("CIAO");  
 dizionarioInput.add("TOPO");  
 dizionarioInput.add("INORGANICI");  
 dizionarioInput.add("TESIO");  
 dizionarioInput.add("TOCCIRE");  
 dizionarioInput.add("TORO");  
  
  
 dizionarioInput.add("STRAGRANDE");  
 dizionarioInput.add("TESEO");  
 dizionarioInput.add("CASABIANCA");  
 dizionarioInput.add("ALAIN");  
 dizionarioInput.add("FE");  
 dizionarioInput.add("FURIA");  
 dizionarioInput.add("ROSSO");  
 dizionarioInput.add("ALVARO");  
 dizionarioInput.add("ONLUS");  
 dizionarioInput.add("LEZIOSA");  
 dizionarioInput.add("BISTRO");  
 dizionarioInput.add("PIETA");  
 dizionarioInput.add("ESTERE");  
 dizionarioInput.add("DENTISTICO");  
 dizionarioInput.add("LASER");  
 dizionarioInput.add("PORO");  
 dizionarioInput.add("TIZIO");  
 dizionarioInput.add("OSTETRICA");  
 dizionarioInput.add("MOSA");  
 dizionarioInput.add("ANA");  
 dizionarioInput.add("MALESIA");  
 dizionarioInput.add("AN");  
 dizionarioInput.add("LEONINO");  
 dizionarioInput.add("BEN");  
 dizionarioInput.add("TE");  
 dizionarioInput.add("FARETTO");  
 dizionarioInput.add("RIO");  
 dizionarioInput.add("GARRANI");  
 dizionarioInput.add("WO");  
 dizionarioInput.add("LISI");  
 dizionarioInput.add("GARA");  
 dizionarioInput.add("MARGARINE");  
 dizionarioInput.add("INORGANICA");  
 dizionarioInput.add("ALATO");  
 dizionarioInput.add("FERRAMENTA");  
 dizionarioInput.add("CRANICO");  
 dizionarioInput.add("RA");  
 dizionarioInput.add("RAVERA");  
 dizionarioInput.add("TOCCARE");  
 dizionarioInput.add("OCARINA");  
 dizionarioInput.add("BOB");  
 dizionarioInput.add("LIONE");  
 dizionarioInput.add("ODORE");  
 dizionarioInput.add("ADAMI");  
 dizionarioInput.add("SCAPPATOIA");  
 dizionarioInput.add("ALIBI");  
 dizionarioInput.add("LORENA");  
 dizionarioInput.add("MASSA");  
 dizionarioInput.add("SAVONAROLA");  
 dizionarioInput.add("IRENE");  
 dizionarioInput.add("WESER");  
 dizionarioInput.add("BOSTON");  
 dizionarioInput.add("BE");  
\*/  
  
  
 //esempio3  
 dizionarioInput* = **new** ArrayList<String>();  
 *dizionarioInput*.add(**"BANALITA"**);  
 *dizionarioInput*.add(**"MIAO"**);  
 *dizionarioInput*.add(**"TETTO"**);  
 *dizionarioInput*.add(**"ISOLA"**);  
 *dizionarioInput*.add(**"LUCI"**);  
 *dizionarioInput*.add(**"IDA"**);  
 *dizionarioInput*.add(**"TONANTE"**);  
 *dizionarioInput*.add(**"EO"**);  
 *dizionarioInput*.add(**"NOMINE"**);  
 *dizionarioInput*.add(**"NAT"**);  
 *dizionarioInput*.add(**"SPIEGAMENTO"**);  
 *dizionarioInput*.add(**"TA"**);  
 *dizionarioInput*.add(**"OTRI"**);  
 *dizionarioInput*.add(**"ADATTE"**);  
 *dizionarioInput*.add(**"EOLO"**);  
 *dizionarioInput*.add(**"AGARICO"**);  
 *dizionarioInput*.add(**"IDIOMA"**);  
 *dizionarioInput*.add(**"PECORE"**);  
 *dizionarioInput*.add(**"GIARDINIPUBBLICI"**);  
 *dizionarioInput*.add(**"ESIGUI"**);  
 *dizionarioInput*.add(**"AI"**);  
 *dizionarioInput*.add(**"ELI"**);  
 *dizionarioInput*.add(**"MANICOTTO"**);  
 *dizionarioInput*.add(**"ITER"**);  
 *dizionarioInput*.add(**"ARRINGARE"**);  
 *dizionarioInput*.add(**"TI"**);  
 *dizionarioInput*.add(**"MARINOMARINI"**);  
 *dizionarioInput*.add(**"FIDEL"**);  
 *dizionarioInput*.add(**"INEDITI"**);  
 *dizionarioInput*.add(**"TOM"**);  
 *dizionarioInput*.add(**"NI"**);  
 *dizionarioInput*.add(**"BARATRO"**);  
 *dizionarioInput*.add(**"LARIO"**);  
 *dizionarioInput*.add(**"ULANI"**);  
 *dizionarioInput*.add(**"ATEO"**);  
 *dizionarioInput*.add(**"ON"**);  
 *dizionarioInput*.add(**"CALLIMACO"**);  
 *dizionarioInput*.add(**"NEO"**);  
 *dizionarioInput*.add(**"ISACCO"**);  
 *dizionarioInput*.add(**"TINI"**);  
 *dizionarioInput*.add(**"RINASCIMENTO"**);  
 *dizionarioInput*.add(**"LB"**);  
 *dizionarioInput*.add(**"ADRIA"**);  
 *dizionarioInput*.add(**"SL"**);  
 *dizionarioInput*.add(**"NC"**);  
 *dizionarioInput*.add(**"PIRAMIDE"**);  
 *dizionarioInput*.add(**"ALI"**);  
 *dizionarioInput*.add(**"ADA"**);  
 *dizionarioInput*.add(**"ILIO"**);  
 *dizionarioInput*.add(**"AU"**);  
 *dizionarioInput*.add(**"ECONOMICA"**);  
 *dizionarioInput*.add(**"ICE"**);  
 *dizionarioInput*.add(**"SUD"**);  
 *dizionarioInput*.add(**"AREA"**);  
 *dizionarioInput*.add(**"RAME"**);  
 *dizionarioInput*.add(**"MB"**);  
 *dizionarioInput*.add(**"CALAF"**);  
 *dizionarioInput*.add(**"BUE"**);  
 *dizionarioInput*.add(**"EMI"**);  
 *dizionarioInput*.add(**"ASTRONOMICA"**);  
 *dizionarioInput*.add(**"CRUMIRA"**);  
 *dizionarioInput*.add(**"OZI"**);  
 *dizionarioInput*.add(**"RINOMATA"**);  
 *dizionarioInput*.add(**"ANOMALIA"**);  
 *dizionarioInput*.add(**"IDIOTI"**);  
 *dizionarioInput*.add(**"LOS"**);  
 *dizionarioInput*.add(**"SS"**);  
 *dizionarioInput*.add(**"URNE"**);  
 *dizionarioInput*.add(**"ORTI"**);  
 *dizionarioInput*.add(**"RA"**);  
 *dizionarioInput*.add(**"PUBBLICAZIONI"**);  
 *dizionarioInput*.add(**"SE"**);  
  
  
  
  
 *//window.createUIComponents();* window.open();  
 frame.setContentPane(window.**panelMain**);  
  
 }  
  
 **public void** open() {  
  
 *//creazione cruciverba per l'utilizzo di funzioni dell'algoritmo1 per i 3 esempi creati  
 /\*//esempio1  
 cruciverba1 = new ImplAlg1Cruciverba(panelMain, matrice, "CANE", 0, 0, dizionarioInput,'O');  
\*/  
  
 /\*//esempio2  
 cruciverba1=new ImplAlg1Cruciverba(panelMain,matrice,"TERRORISTA",7,2,dizionarioInput,'O');  
\*/  
  
 /\*//esempio3  
 cruciverba1=new ImplAlg1Cruciverba(panelMain,matrice,"DONO",4,4,dizionarioInput,'V');  
 \*/  
  
 //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo2 per i 3 esempi creati  
 /\*//esempio1  
 cruciverba1 = new ImplAlg2Cruciverba(panelMain, matrice, "CANE", 0, 0, dizionarioInput,'O');  
 \*/  
  
 /\*//esempio2  
 cruciverba1=new ImplAlg2Cruciverba(panelMain,matrice,"TERRORISTA",7,2,dizionarioInput,'O');  
\*/  
  
 /\*//esempio3  
 cruciverba1=new ImplAlg2Cruciverba(panelMain,matrice,"DONO",4,4,dizionarioInput,'V');  
 \*/  
  
 //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo4 per i 3 esempi creati  
 /\*//esempio1  
 cruciverba1=new ImplAlg4Cruciverba\_AI(panelMain,matrice,"CANE",0,0, dizionarioInput,'O');  
 \*/  
  
 /\*//esempio2  
 cruciverba1 = new ImplAlg4Cruciverba\_AI(panelMain,matrice, "TERRORISTA", 7, 2, dizionarioInput,'O');  
\*/  
  
 //esempio3  
 cruciverba1*= **new** ImplAlg4Cruciverba\_AI(**panelMain**, *matrice*, **"DONO"**,4,4,*dizionarioInput*, **'V'**);  
  
 **listListaParole**.setListData(*dizionarioInput*.toArray());  
  
 }  
  
  
 **private void** createUIComponents() {  
 *// creo tutti gli elementi grafici dell'interfaccia utente per il cruciverba escluse tutte le caselle bianche e nere che creo in futuro* **panelMain** = **new** JPanel();  
 GroupLayout layoutGUI = **new** GroupLayout(**panelMain**);  
 **panelMain**.setLayout(layoutGUI);  
 layoutGUI.setAutoCreateGaps(**true**);  
 layoutGUI.setAutoCreateContainerGaps(**true**);  
  
 **labelCruciverba** = **new** JLabel(**"Cruciverba "** + *matrice*.**length** + **"X"** + *matrice*[0].**length**);  
 **labelCruciverba**.setBounds(20, 20, 400, 20);  
  
 **buttonRisolviCruciverba** = **new** JButton(**"RisolviCruciverba"**);  
 **buttonRisolviCruciverba**.setBounds(20, 40, 200, 20);  
 **buttonCercaParola** = **new** JButton(**"CercaParola"**);  
 **buttonCercaParola**.setBounds(240, 40, 200, 20);  
  
 **listListaParole**=**new** JList();  
 **listListaParole**.setBounds(20, 80, 200,200);  
  
 **scrollPaneListaParole** = **new** JScrollPane(**listListaParole**);  
 **scrollPaneListaParole**.setPreferredSize(**new** Dimension(300,200));  
 **scrollPaneListaParole**.setBounds(20,80,800,800);  
  
 **panelListaParole** = **new** JPanel();  
 BorderLayout groupLayoutListaParole = **new** BorderLayout();  
 **panelListaParole**.setLayout(groupLayoutListaParole);  
 **panelListaParole**.add(**scrollPaneListaParole**);  
 **panelListaParole**.setBounds(20,80,200,200);  
  
 **labelParolaInserita** = **new** JLabel(**"Ultima parola inserita"**);  
 **labelParolaInserita**.setBounds(20, 300, 400, 20);  
  
 **textFieldParolaInserita**=**new** JTextField();  
 **textFieldParolaInserita**.setSize(200, 20);  
 **textFieldParolaInserita**.setLocation(20, 320);  
  
  
 **panelMain**.add(**labelCruciverba**);  
 **panelMain**.add(**buttonRisolviCruciverba**);  
 **panelMain**.add(**buttonCercaParola**);  
 **panelMain**.add(**panelListaParole**);  
 **panelMain**.add(**labelParolaInserita**);  
 **panelMain**.add(**textFieldParolaInserita**);  
 **panelMain**.revalidate();  
  
 }  
}

Classe Schema:  
**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.util.ArrayList;  
  
**public class** Schema {  
 **private** ArrayList<Parola> **paroleSchema**;  
 **private** ArrayList<Casella> **caselleSchema**;  
  
  
 *//costruttore che prende la matrice in ingresso e crea lo schema aggiungendo la prima parola* **public** Schema(JPanel panel, **char** matrice[][], String parolaIniziale, Posizione posizioneParolaIniziale, **char** orientamentoInput) {  
 *//inizializzazione variabili classe* **paroleSchema** = **new** ArrayList<Parola>();  
 **caselleSchema** = **new** ArrayList<Casella>();  
 ArrayList<Casella> caselleParola = **new** ArrayList<Casella>();  
 Casella casellaAttuale;  
 StringBuilder creazioneParola = **new** StringBuilder();  
 Posizione posizioneIniziale = **new** Posizione();  
 Posizione posizioneAttuale = **new** Posizione();  
 Parola parolaCreata;  
 **int** lunghezzaParola = 0;  
 **char** carattere = **'.'**;  
 **char** casellaNera = **'\*'**;  
 **boolean** primaLettera = **true**;  
  
 *//ricerca parole orizzontali* **for** (**int** i = 0; i < matrice.**length**; i++) {  
 **for** (**int** j = 0; j < matrice[0].**length**; j++) {  
 *//inizializzazione posizione casella* posizioneAttuale.setRiga(i);  
 posizioneAttuale.setColonna(j);  
  
 *//ricerca della prima lettera della parola che verrà inserita nello schema* **if** (matrice[i][j] == carattere && primaLettera) {  
 creazioneParola.append(**' '**);  
 primaLettera = **false**;  
 posizioneIniziale.setRiga(i);  
 posizioneIniziale.setColonna(j);  
 lunghezzaParola++;  
  
 *//inizializzazione casella* casellaAttuale = **new** Casella(panel, posizioneAttuale, matrice[i][j], **false**);  
  
 **caselleSchema**.add(casellaAttuale);  
 caselleParola.add(casellaAttuale);  
 } **else if** (matrice[i][j] == carattere && (!primaLettera)) { *//ricerca lettere successive della parola che verrà inserita nello schema* creazioneParola.append(**' '**);  
 lunghezzaParola++;  
  
 *//inizializzazione casella* casellaAttuale = **new** Casella(panel, posizioneAttuale, matrice[i][j], **false**);  
  
 **caselleSchema**.add(casellaAttuale);  
 caselleParola.add(casellaAttuale);  
 } **else if** (matrice[i][j] == casellaNera) { *//ricerca delle caselle nere dello schema* **if** (lunghezzaParola >= 2) { *//lunghezza minima di una parola raggiunta, la inserisco nello schema così come la casella nera e ripristino i valori iniziali* **caselleSchema**.add(**new** Casella(panel, posizioneAttuale, matrice[i][j], **true**));  
 parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'O'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** { *// lunghezza minima di una parola non raggiunta, inserisco solo la casella nera e ripristino i valori iniziali* **caselleSchema**.add(**new** Casella(panel, posizioneAttuale, matrice[i][j], **true**));  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
  
 }  
 }  
 **if** (lunghezzaParola >= 2) { *//raggiunta la fine della riga, inserisco la parola nello schema se ha la lunghezza minima altrimenti no* parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'O'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** {  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
 }  
  
 *//ricerca parole verticali, stessa procedura di quelle orizzontali con l'aggiunta del controllo per le caselle già presenti* **for** (**int** j = 0; j < matrice[0].**length**; j++) {  
 **for** (**int** i = 0; i < matrice.**length**; i++) {  
 *//ricerca della prima lettera della parola che verrà inserita nello schema* **if** (matrice[i][j] == carattere && primaLettera) {  
 creazioneParola.append(**' '**);  
 primaLettera = **false**;  
 posizioneIniziale.setRiga(i);  
 posizioneIniziale.setColonna(j);  
 lunghezzaParola++;  
  
 *//cerco la casella già esistente relativa alla riga i, colonna j per poi inserirla nella parola ed averla  
 // collegata alle caselle dello schema, così come alle parole in orizzontale* cercaCasella(caselleParola, i, j);  
  
 } **else if** (matrice[i][j] == carattere && (!primaLettera)) {  
 creazioneParola.append(**' '**);  
 lunghezzaParola++;  
 cercaCasella(caselleParola, i, j);  
 } **else if** (matrice[i][j] == casellaNera) {  
 **if** (lunghezzaParola >= 2) {  
 parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'V'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** {  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
 }  
 }  
 **if** (lunghezzaParola >= 2) {  
 parolaCreata = **new** Parola(creazioneParola.toString(), posizioneIniziale, **'V'**, lunghezzaParola, caselleParola);  
 **paroleSchema**.add(parolaCreata);  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 } **else** {  
 creazioneParola = **new** StringBuilder();  
 posizioneIniziale.ripristinaPosizione();  
 lunghezzaParola = 0;  
 primaLettera = **true**;  
 caselleParola = **new** ArrayList<Casella>();  
 }  
 }  
 *//aggiorna schema con la prima parola iniziale data in input* aggiornaSchema(parolaIniziale, posizioneParolaIniziale, orientamentoInput);  
 }  
  
 *//metodo costruttore schema a partire da uno schema già esistente* **public** Schema(Schema s){  
 **this**.**paroleSchema**=s.getParoleSchema();  
 **this**.**caselleSchema**=s.getCaselleSchema();  
 }  
  
 **public** ArrayList<Parola> getParoleSchema() { **return new** ArrayList<Parola>(**paroleSchema**); }  
  
 **public** ArrayList<Casella> getCaselleSchema(){ **return new** ArrayList<Casella>(**caselleSchema**); }  
  
 *// inserisce la parola in input nella parola dello schema con la stessa posizione e orientamento* **public void** aggiornaSchema(String parola, Posizione posizioneParola, **char** orientamentoInput) {  
 Parola p = **new** Parola(parola, posizioneParola, orientamentoInput, parola.length());  
 **for** (Parola parolaSchema : **paroleSchema**) {  
 **if** (parolaSchema.confrontaCaselle(p)) {  
 parolaSchema.setParola(parola);  
 parolaSchema.setLunghezza(parola.length());  
 parolaSchema.aggiornaCaselleParola();  
 *//paroleSchema.set(paroleSchema.indexOf(parolaSchema), p);* **break**;  
 }  
 }  
   
 *//dopo aver aggiornato una parola dello schema devo fare in modo di aggiornare le parole dello schema che hanno lettere collegate  
 // alla parola appena aggiornata* **for**(Parola parolaSchema : **paroleSchema**){  
 parolaSchema.aggiornaParola();  
 }  
  
 }  
  
 *//cerca all'interno delle caselle dello schema e se già presente una casella con posizione riga, colonna allora la  
 // assegna alle caselleParola senza creare una nuova casella apposita* **public void** cercaCasella(ArrayList<Casella> caselleParola, **int** riga, **int** colonna) {  
 **for** (Casella c : **caselleSchema**) {  
 **if** (c.confrontaPosizione(riga, colonna)) {  
 caselleParola.add(c);  
 }  
 }  
 }  
  
 *//cerca le parole dello schema di lunghezza n* **public** ArrayList<Parola> ricercaLunghezzaParole(**int** n){  
 ArrayList<Parola> paroleLunghezzaN = **new** ArrayList<Parola>();  
 **for** (Parola p : **paroleSchema**){  
 **int** lunghezzaParola=p.getLunghezza();  
 **int** lettereInserite=p.getLettereInserite();  
 **if** (lunghezzaParola==n && lettereInserite<lunghezzaParola){  
 paroleLunghezzaN.add(p);  
 }  
 }  
 **return** paroleLunghezzaN;  
 }  
  
 *//cerca la lunghezza massima tra le parole dello schema* **public int** cercaLunghezzaParolaMax(){  
 **int** lunghezzaMax=0;  
 **for** (Parola p : **paroleSchema**){  
 **int** lunghezzaCorrente=p.getLunghezza();  
 **if** (lunghezzaCorrente > lunghezzaMax){  
 lunghezzaMax=lunghezzaCorrente;  
 }  
 }  
 **return** lunghezzaMax;  
 }  
  
}

Classe Parola:  
**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.util.ArrayList;  
  
**public class** Parola {  
 **private** String **parola**;  
 **private** ArrayList<Casella> **caselleParola**;  
 **private** Posizione **posizioneParola**;  
 **private int lunghezza**;  
 **private int lettereInserite**;  
 **private char orientamento**;  
  
 *//metodo costruttore crea una parola con le relative informazioni, senza caselleParola collegate* **public** Parola(String parola, Posizione posizioneIniziale, **char** orientamento, **int** lunghezza) {  
 **try** {  
 *//controllo parola* **if** (parola.length() > 0) {  
 **this**.**parola** = parola;  
 } **else** {  
 **throw new** Exception(**"Parola non corretta."**);  
 }  
  
 *//controllo posizione* **if** (posizioneIniziale.getRiga() >= 0 && posizioneIniziale.getColonna() >= 0) {  
 **posizioneParola** = **new** Posizione(posizioneIniziale);  
 } **else** {  
 **throw new** Exception(**"Posizione indicata non corretta."**);  
 }  
  
 *//controllo orientamento* **if** (orientamento == **'V'** || orientamento == **'O'**) {  
 **this**.**orientamento** = orientamento;  
 } **else** {  
 **throw new** Exception(**"Orientamento (verticale o orizzontale) non corretto."**);  
 }  
  
 *//controllo lunghezza* **if** (lunghezza > 1) {  
 **this**.**lunghezza** = lunghezza;  
 } **else** {  
 **throw new** Exception(**"Lunghezza non corretta. Vengono inserite le parole con almeno 2 lettere"**);  
 }  
  
 } **catch** (Exception ex) {  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 }  
  
 *//metodo costruttore crea una parola con le relative informazioni con le caselleParola collegate* **public** Parola(String parola, Posizione posizioneIniziale, **char** orientamento, **int** lunghezza, ArrayList<Casella> caselleParolaInput) {  
 **try** {  
  
 *//inizializzo le caselle della parola con le caselle dello schema create in precedenza* **if** (caselleParolaInput != **null**) {  
 **caselleParola** = caselleParolaInput;  
 }  
 *//controllo parola* **if** (parola.length() > 0) {  
 **this**.**parola** = parola;  
 } **else** {  
 **throw new** Exception(**"Parola non corretta."**);  
 }  
  
 *//controllo posizione* **if** (posizioneIniziale.getRiga() >= 0 && posizioneIniziale.getColonna() >= 0) {  
 **posizioneParola** = **new** Posizione(posizioneIniziale);  
 } **else** {  
 **throw new** Exception(**"Posizione indicata non corretta."**);  
 }  
  
 *//controllo orientamento* **if** (orientamento == **'V'** || orientamento == **'O'**) {  
 **this**.**orientamento** = orientamento;  
 } **else** {  
 **throw new** Exception(**"Orientamento (verticale o orizzontale) non corretto."**);  
 }  
  
 *//controllo lunghezza* **if** (lunghezza > 1) {  
 **this**.**lunghezza** = lunghezza;  
 } **else** {  
 **throw new** Exception(**"Lunghezza non corretta. Vengono inserite le parole con almeno 2 lettere"**);  
 }  
  
 } **catch** (Exception ex) {  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 }  
  
 *//metodo costruttore crea una parola a partire da una passata in input* **public** Parola(Parola p){  
 **this**.setParola(p.getParola());  
 **this**.setCaselleParola(p.getCaselleParola());  
 **this**.setPosizioneParola(p.getPosizioneParola());  
 **this**.setLunghezza(p.getLunghezza());  
 **this**.setLettereInserite(p.getLettereInserite());  
 **this**.setOrientamento(p.getOrientamento());  
 }  
  
 **public** ArrayList<Casella> getCaselleParola() {  
 **return caselleParola**;  
 }  
  
 **public void** setCaselleParola(ArrayList<Casella> caselleParola) {  
 **this**.**caselleParola** = caselleParola;  
 }  
  
 **public** String getParola() {  
 **return parola**;  
 }  
  
 **public void** setParola(String parola) {  
 **this**.**parola** = parola;  
 }  
  
 **public** Posizione getPosizioneParola() {  
 **return posizioneParola**;  
 }  
  
 **public void** setPosizioneParola(Posizione posizioneParola) {  
 **this**.**posizioneParola** = posizioneParola;  
 }  
  
 **public int** getLunghezza() {  
 **return lunghezza**;  
 }  
  
 **public void** setLunghezza(**int** lunghezza) {  
 **this**.**lunghezza** = lunghezza;  
 }  
  
 **public char** getOrientamento() {  
 **return orientamento**;  
 }  
  
 **public void** setOrientamento(**char** orientamento) {  
 **this**.**orientamento** = orientamento;  
 }  
  
 **public int** getLettereInserite() {  
 **return lettereInserite**;  
 }  
  
 **public void** setLettereInserite(**int** lettereInserite) {  
 **this**.**lettereInserite** = lettereInserite;  
 }  
  
 *//controlla se le due parole corrispondono alla stessa casella nello schema, sia come orientamento che come inizio e lunghezza parola* **public boolean** confrontaCaselle(Parola p) {  
 **if** (**this**.**orientamento** == p.**orientamento** && **posizioneParola**.equals(p.getPosizioneParola()) && **this**.**lunghezza** == p.**lunghezza**) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
  
 }  
  
 *//aggiorna il testo delle caselle con la stringa dentro parola* **public void** aggiornaCaselleParola(){  
 **try**{  
 **if** (**lunghezza**==**caselleParola**.size()){  
 **for** (**int** i=0; i<**lunghezza**; i++){  
 **caselleParola**.get(i).aggiornaCarattere(**parola**.charAt(i));  
 }  
 }**else**{  
 **throw new** Exception (**"Lunghezza parola diversa dalla lunghezza della parola nel cruciverba"**);  
 }  
 }  
 **catch** (Exception ex){  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
  
 *//aggiorna la stringa dentro parola con il testo delle caselle, contando anche le lettere inserite* **public void** aggiornaParola(){  
 StringBuilder strParolaNuova=**new** StringBuilder();  
 **lettereInserite**=0;  
 **try**{  
 **if** (**lunghezza**==**caselleParola**.size()){  
 **for** (**int** i=0; i<**lunghezza**; i++){  
 **char** carattereCasella=**caselleParola**.get(i).getCarattereCasella();  
 **if** (carattereCasella!=**'.'**){  
 **lettereInserite**++;  
 }  
 strParolaNuova.append(carattereCasella);  
 }  
 **parola**=strParolaNuova.toString();  
 }**else**{  
 **throw new** Exception (**"Lunghezza parola diversa dalla lunghezza della parola nel cruciverba"**);  
 }  
 }  
 **catch** (Exception ex){  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
  
 *//controlla se una parola è stata completata* **public boolean** isComplete(){  
 **if** (**this**.**lunghezza**==**this**.**lettereInserite**){  
 **return true**;  
 }**else**{  
 **return false**;  
 }  
 }  
  
 *//controllo se esistono caselle uguali tra due parole* **public boolean** isLinked(Parola p){  
 **for** (Casella casellaParolaCorrente : **caselleParola**){  
 **for** (Casella casellaParolaDaConfrontare : p.getCaselleParola()){  
 **if** (casellaParolaCorrente.confrontaCaselle(casellaParolaDaConfrontare)){  
 **return true**;  
 }  
 }  
 }  
 **return false**;  
 }  
  
}

Classe Casella:  
**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** Casella {  
 **private** Posizione **posizioneCasella**;  
 **private** JTextField **textFieldCasella**;  
 **private char carattereCasella**;  
 **private boolean casellaNera**;  
  
 *//costruttore casella con input il Jpanel dove verrà inserita la casella* **public** Casella(JPanel panel, Posizione posizioneInput, **char** carattereInput, **boolean** casellaNeraInput) {  
 **int** xIniziale = 240;  
 **int** yIniziale = 80;  
 **int** i, j = 0;  
 **try** {  
 **if** (posizioneInput != **null**) {  
 **posizioneCasella** = **new** Posizione(posizioneInput);  
 i = **posizioneCasella**.getRiga();  
 j = **posizioneCasella**.getColonna();  
 } **else** {  
 **throw new** Exception(**"Posizione non esistente"**);  
 }  
 **carattereCasella** = carattereInput;  
 **casellaNera** = casellaNeraInput;  
 **textFieldCasella** = **new** JTextField();  
 **textFieldCasella**.setSize(20, 20);  
 *//calcolo la posizione della casella a partire dalla posizione passata in input* **textFieldCasella**.setLocation(xIniziale + j \* 20, yIniziale + i \* 20);  
 **textFieldCasella**.setText(String.*valueOf*(**carattereCasella**));  
  
  
 *//imposto lo sfondo della casella nera e la disabilito* **if** (**casellaNera**) {  
 **textFieldCasella**.setBackground(**new** Color(0, 0, 0));  
 **textFieldCasella**.setEditable(**false**);  
 **textFieldCasella**.setFocusable(**false**);  
 }  
  
 *//aggancio il textField al pannello passato in input* panel.add(**textFieldCasella**);  
 } **catch** (Exception ex) {  
 JOptionPane.*showMessageDialog*(**null**, ex.toString(), **"Errore"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
  
 **public char** getCarattereCasella() {  
 **return carattereCasella**;  
 }  
  
 **public void** setCarattereCasella(**char** carattereCasella) {  
 **this**.**carattereCasella** = carattereCasella;  
 }  
  
 *//controlla ritornando true o false se la posizione della casella dell'oggetto corrente è uguale a quella passata in input* **public boolean** confrontaPosizione(**int** riga, **int** colonna) {  
 Posizione posizioneDaConfrontare = **new** Posizione(riga, colonna);  
 **return posizioneCasella**.equals(posizioneDaConfrontare);  
 }  
  
 *//procedura di confronto caselle, ritorna true se la casella in input è la stessa della casella dell'oggetto corrente* **public boolean** confrontaCaselle(Casella c){  
 **return posizioneCasella**.equals(c.**posizioneCasella**);  
 }  
  
 *//aggiorna il carattere dell'oggetto corrente e il testo del componente visualizzato a video* **public void** aggiornaCarattere(**char** carattereInput){  
 **carattereCasella**=carattereInput;  
 **textFieldCasella**.setText(String.*valueOf*(carattereInput));  
 }  
}

Classe Posizione:  
**package** com.cruciverbapackage;  
  
*//Coffaro\_Davide\_mat556603\_Progetto ESP cruciverba***public class** Posizione {  
 **int riga**;  
 **int colonna**;  
  
 *//costruttore posizione senza valori in input, inserisco valori standard* **public** Posizione() {  
 **this**.**riga** = -1;  
 **this**.**colonna** = -1;  
 }  
  
 *//costruttore posizione a partire da valori in input* **public** Posizione(**int** rigaInput, **int** colonnaInput) {  
 **this**.**riga** = rigaInput;  
 **this**.**colonna** = colonnaInput;  
 }  
  
 *//costruttore posizione da un'altra posizione già esistente* **public** Posizione(Posizione posizioneInput) {  
 **this**.**riga** = posizioneInput.getRiga();  
 **this**.**colonna** = posizioneInput.getColonna();  
 }  
  
 **public int** getRiga() {  
 **return riga**;  
 }  
  
 **public void** setRiga(**int** riga) {  
 **this**.**riga** = riga;  
 }  
  
 **public int** getColonna() {  
 **return colonna**;  
 }  
  
 **public void** setColonna(**int** colonna) {  
 **this**.**colonna** = colonna;  
 }  
  
 *//ripristina valori di default* **public void** ripristinaPosizione() {  
 **this**.**riga** = -1;  
 **this**.**colonna** = -1;  
 }  
  
 *//controlla se la posizione passata in input corrisponde alla stessa riga e colonna dell'oggetto corrente* **public boolean** equals(Posizione posizioneDaConfrontare) {  
 **if** (**this**.**riga** == posizioneDaConfrontare.getRiga() && **this**.**colonna** == posizioneDaConfrontare.getColonna()) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
 }  
}