

Risoluzione automatica del cruciverba

PROGETTO DI ESPERIENZE DI PROGRAMMAZIONE

DAVIDE COFFARO MATRICOLA 556603

DESCRIZIONE DEL PROBLEMA

Il problema di “Risoluzione automatica del cruciverba” è quello di generare lo schema del cruciverba a partire da un input, il quale conterrà anche la prima parola da inserire nello schema, e da qui analizzare lo schema e completarlo con la lista di parole disponibili fornite al programma.

Dato che le parole sono collegate tra loro, nel momento in cui inserisco nello schema una parola tra quelle disponibili, questo inserimento influirà sulla ricerca successiva delle parole da inserire.

Potrebbe verificarsi che durante la ricerca della parola successiva da inserire, ci siano più parole da poter inserire in quelle caselle e questa situazione potrebbe bloccare la risoluzione del cruciverba.

Inoltre la lista di parole disponibili che servirà per completare il cruciverba potrebbe contenere parole in più che non sono necessarie al suo completamento, questo potrebbe bloccare il proseguimento della risoluzione del cruciverba oppure far sì che vengano inserite parole sbagliate all'interno dello schema del cruciverba, quindi è possibile prevedere una situazione in cui bisogna ripristinare una situazione precedente.

È possibile trovare un limite inferiore alla complessità del problema di “Risoluzione automatica del cruciverba” scomponendo il problema in sotto-problemi, che saranno:

con n =numero parole da completare dello schema

e m =lunghezza parola arrotondata alla parola più lunga per gestire il caso pessimo

- Cerca fra le parole disponibili da inserire -> $L_ricerca(n)$
- Confronta le lettere delle caselle con quella della parola disponibile corrente -> $L_confronta(m)$
- Inserire parole nello schema -> $L_inserimento(n-1)$

Ma per ricomporre i limiti ottenuti dovremmo stabilire la dipendenza funzionale, quindi scelgo banalmente la dimensione del cruciverba come limite inferiore e ottengo quindi che il problema di completare un cruciverba con le parole disponibili ha complessità $\Omega(n^2)$.

ALGORITMI PER RISOLVERE IL PROBLEMA

- Algoritmo1 – parto dallo schema, cerco la parola da inserire fino a completare il cruciverba
PROCEDURA ALGORITMO1:
 - 1) Memorizzo tutte le parole di una certa lunghezza;
 - 2) Prendo la parola con più lettere già inserite e cerco se sono presenti parole congrue (scarto quelle con lettere errate) nella struttura dati contenente le parole da inserire
 - 3) Se è presente solo una parola da poter inserire la inserisco aggiornando le parole ad essa collegate e la cancello dalla struttura dati contenente le parole disponibili;
 - 4) se sono presenti altre parole della stessa lunghezza memorizzate al punto 1) torno al punto 2);
 - 5) se il cruciverba non è ancora completo torno al punto 1) cercando parole di un'altra lunghezza;
 - 6) se il cruciverba è stato completato stampo “Cruciverba completato”

CONDIZIONI DI TERMINAZIONE:

- 1) il cruciverba è stato completato;
- 2) ho eseguito i passi da 1 a 5 un numero prefissato di volte -> il cruciverba non è stato completato

- Algoritmo2 – procedura simile all'algoritmo1 ma:
 - 1) Utilizzo di strutture dati per memorizzare parole della stessa lunghezza
 - 2) Quando inserisco la parola di una certa lunghezza nello schema, la elimino anche dalla struttura dati che la conteneva
 - 3) Quando tutte le strutture dati contenenti parole sono vuote, termino l'algoritmo

Problemi relativi all'algoritmo2

- Riempire la struttura dati delle parole all'inizio durante la creazione delle caselle
- Memoria occupata dalla struttura dati delle parole

- 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 questo algoritmo, che sfrutta 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 struttura dati contenente variabili; dopo analizzo i possibili valori che possono essere inseriti all'interno delle variabili inserendo anch'essi in una struttura dati 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 con strategia MRV (minimum remaining values) 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 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 con 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 vuoto (non sono presenti altre parole disponibili) di una variabile collegata o di una variabile con lo stesso numero di lettere e lo schema del cruciverba non è 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) tornando al passo 2 se sono presenti altri valori per la variabile corrente, altrimenti ritorno il controllo alla funzione ricorsiva chiamante. Se invece i domini di variabili collegate o di variabili con lo stesso numero di lettere non risultano vuoti vado al punto 4;
- 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 dello schema;
- 2) quando non esistono più variabili a cui assegnare un valore.

SCELTE IMPLEMENTATIVE

Nell'implementazione del cruciverba ho deciso di scomporre lo schema del cruciverba nelle parole e nelle caselle, 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 (vocale o consonante) ad esse collegate che vengono utilizzate nei confronti per trovare la parola successiva da inserire nello schema e anche per la stampa a video della casella. Per tutto questo utilizzo una struttura dati relativa allo schema per salvare i suoi dati attuali, cioè le parole e le caselle all'interno dello schema del cruciverba, oltre alle funzioni di aggiornamento schema, ricerca di una determinata casella o di un determinato tipo di parola o di una sua caratteristica particolare.

Lo schema del cruciverba è impostato all'interno del programma con una matrice che definisce le caselle bianche (carattere ".") e quelle nere (carattere "*"), la matrice è creata a partire da un file .txt (il cui nome è definito all'interno del programma) in cui la prima riga contiene il numero di righe e di colonne della matrice mentre le righe successive contengono i caratteri sopra descritti per definire le caselle. Un miglioramento dell'implementazione attuale prevede che lo schema venga individuato analizzando l'immagine dello schema stesso che verrà poi trasformata nella matrice.

La struttura dati Schema contiene strutture dati di tipo array per le parole e le caselle che hanno questa complessità temporale (n è il numero di parole da inserire nello schema):

- inserimento, viene inserito il nuovo elemento in fondo all'array, complessità $O(1)$;
- ricerca o modifica, conoscendo l'indice di un oggetto all'interno dell'array l'accesso ha complessità $O(1)$, se l'elemento deve essere ricercato per confronti abbiamo al caso peggior caso una complessità di $O(n)$;
- non vengono fatte operazioni di cancellazione sulle parole e sulle caselle del cruciverba.

In più utilizzo una struttura dati dinamico di tipo array per memorizzare le parole che possono essere inserite all'interno dello schema; questa struttura dati ha le stesse caratteristiche di quella utilizzata per le parole e le caselle solo che per le cancellazioni si sfrutta prima l'operazione di ricerca di un elemento e poi si effettua la cancellazione, quindi al caso peggior caso in cui l'elemento cercato è in fondo all'array abbiamo una complessità $O(m)$ con m numero di parole che possono essere inserite nello schema. La lista di parole disponibili per l'inserimento dentro lo schema è importata all'interno del programma nella struttura dati appena descritta da un file .txt letto dal programma e il cui nome è definito nel programma (funzione migliorabile con un'implementazione

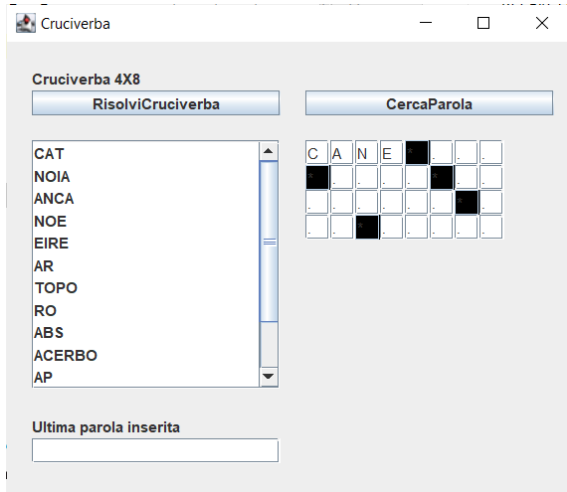
per selezionare un certo file). Le liste di parole che verranno utilizzate per il completamento dello schema sono una lista creata da me contenente solo le parole che effettivamente fanno parte della soluzione del cruciverba, oppure le altre liste sono state prese da un progetto su github come indicato in bibliografia e in cui ho aggiunto le parole che fanno parte della soluzione del cruciverba.

Dato che il numero di parole dello schema n è inferiore o uguale a m numero di parole che possono essere inserite, allora dico che $n=O(m)$ quindi pongo m come limite superiore ad n , cioè al più avrò complessità $O(m)$ nelle operazioni di ricerca relativo alle strutture dati array delle parole e delle caselle dello schema.

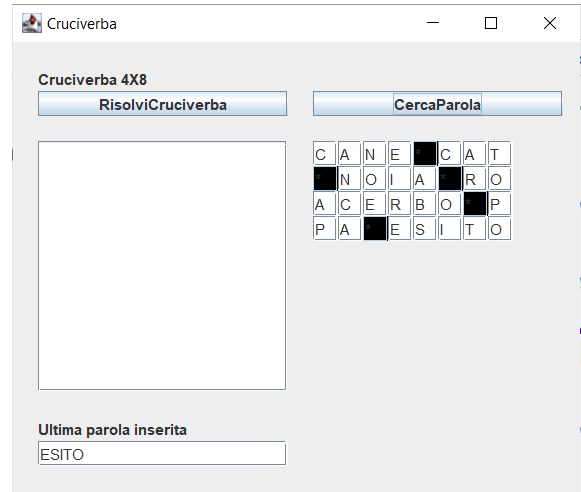
RISULTATI OTTENUTI

Gli esempi seguenti sono suddivisi per i 3 algoritmi implementati, utilizzando come lista di parole da utilizzare per riempire lo schema solo quelle che effettivamente fanno parte dello schema, con al più qualche parola in più (vedere esempio2)

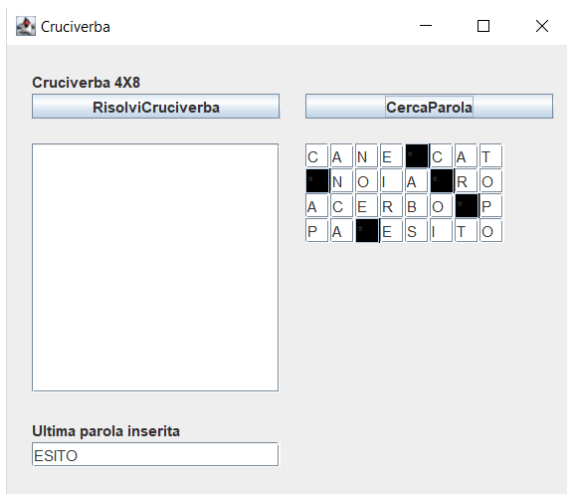
Esempio1: all'apertura del programma



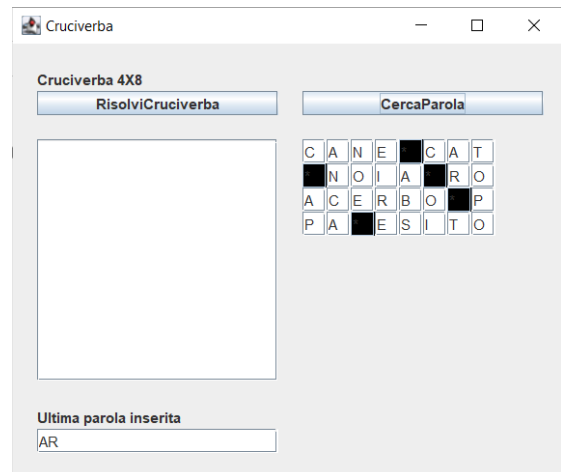
Dopo esecuzione algoritmo2



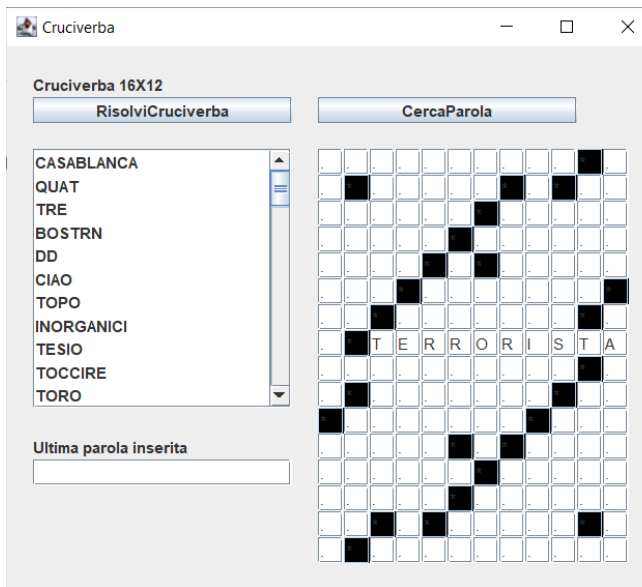
Dopo esecuzione algoritmo1



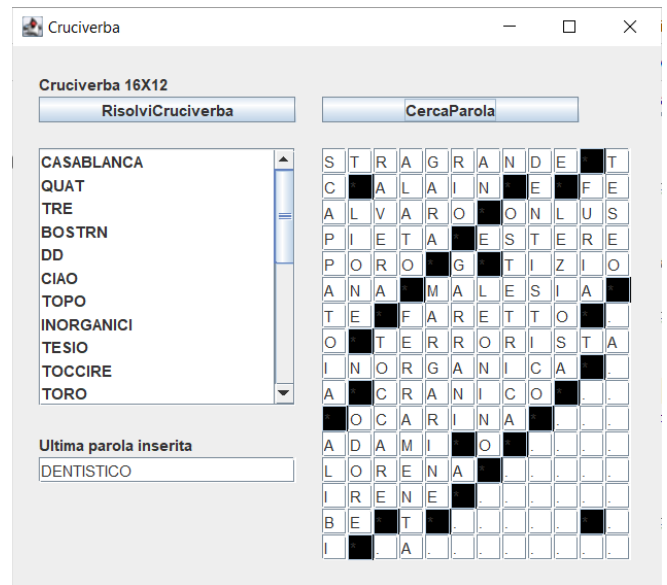
Dopo esecuzione algoritmo4_AI



Esempio2: all'apertura del programma



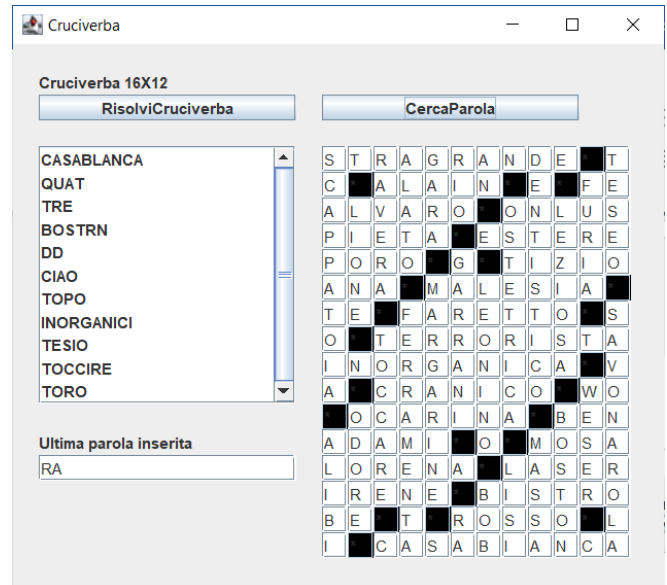
Dopo esecuzione algoritmo2



Dopo esecuzione algoritmo1



Dopo esecuzione algoritmo4_AI



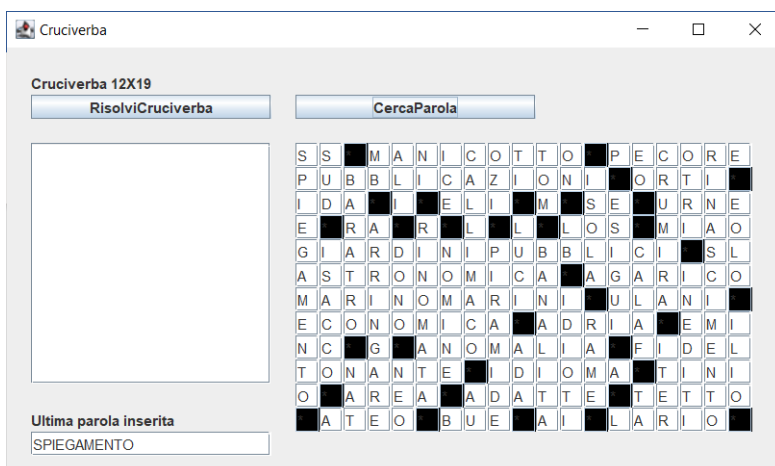
Esempio3: all'apertura del programma



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 risoluzione del cruciverba, se in un certo momento non riesce ad inserire altre parole e il cruciverba non è stato ancora completato allora effettua il "backtracking" per cui elimina i valori inseriti dopo aver avuto il blocco, prova ad inserire un altro valore nella casella precedentemente riempita e effettua questa procedura fino al completamento dello schema.

Altri test effettuati utilizzando liste di parole con 1000, 21000, 60000, 280000 e 660000 parole italiane mostrano che l'algoritmo1 e l'algoritmo2 non riescono a trovare mai la soluzione per nessuno degli esempi riportati, mentre l'algoritmo4_AI ha i seguenti risultati:

- l'esempio1 viene risolto con tutte le liste di parole provate nei test e si può notare che con poche parole oppure molte parole nella lista di parole italiane da inserire, l'algoritmo completa il cruciverba in modo rapido, mentre per valori intermedi (ad esempio con 21000 parole) impiega più tempo.
- l'esempio2 e l'esempio3 invece vengono risolti solo con una lista di 1000 parole italiane, con più parole l'algoritmo non riesce a completare lo schema per mancanze di risorse, infatti durante l'esecuzione viene sollevata un'eccezione di OutOfMemoryError dovuto dal fatto che vengono effettuate molte chiamate ricorsive per scorrere l'albero delle soluzioni che ha una profondità più alta rispetto all'esempio1 perché sono presenti più parole da completare: si passa infatti da 14 parole dell'esempio1 alle 64 e 72 dell'esempio2 e dell'esempio3; questo fa sì che le combinazioni possibili siano molte di più anche se viene utilizzato il meccanismo di inferenza per ridurre i rami dell'albero delle soluzioni da controllare.

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, calcolata analizzando lo schema del cruciverba
- 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 grandezza di n , numero molto alto rispetto agli altri input e trovo che la complessità dell'algoritmo1 è $O(n^5)$ al caso pessimo.

COMPLESSITA' algoritmo2:

Anche per questo algoritmo ho diversi cicli annidati e al caso peggiore l'algoritmo ha complessità $O(n^5)$ così come l'algoritmo1, quindi anche con l'utilizzo di liste con all'interno le parole ancora da completare invece di dover controllare una lista in cui ho le informazioni sul completamento delle parole di una certa lunghezza e poi eventualmente creare la lista delle parole di quella lunghezza e lavorarci, non migliora la complessità dell'algoritmo.

COMPLESSITA' algoritmo4 con AI:

considerato che viene fatta una chiamata ricorsiva per ogni variabile all'interno dello schema del cruciverba avremo una complessità esponenziale. 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 risolubile con complessità minore (tree structured, cutset conditioning e tree decomposition).

Per ogni chiamata a BACKTRACK ho:

- 1) $n \cdot c$ operazioni per selectUnassignedVariable

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

Complessità algoritmo4_AI $\rightarrow T(n) = n \cdot c + d \cdot c \cdot (4 \cdot n \cdot c + T(n-1))$ 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 $j \cdot c + d \cdot c \cdot j$ con $j = \text{numero variabili ancora da assegnare}$

Complessità algoritmo4_AI = $O(c + c \cdot \text{sommatoria per } i=1..n \text{ di } (d^i)) = O(d^n)$

Quindi l'algoritmo1 e l'algoritmo2 sarebbero migliori perché hanno una complessità polinomiale al caso pessimo mentre l'algoritmo4_AI ha complessità esponenziale sempre al caso pessimo, 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:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ActionListener;
9 import java.io.File;
10 import java.io.FileNotFoundException;
11 import java.util.ArrayList;
12 import java.util.InputMismatchException;
13 import java.util.Scanner;
14
15 public class InterfacciaCruciverba {
16     private JPanel panelMain;
17     private JPanel panelListaParole;
18     private JLabel labelCruciverba;
19     private JButton buttonRisolviCruciverba;
20     private JButton buttonCercaParola;
21     private JList listListaParole;
22     private JScrollPane scrollPanelListaParole;
23     private JLabel labelParolaInserita;
24     private JTextField textFieldParolaInserita;
25     private static ImplementazioneCruciverba cruciverba1;
26     private static int dimensioneFinestraMinima=500;
27     private long startTime=0;
28     private long stopTime=0;
29     private long totalTime=0;
30
31     //inizializzazione della matrici che faranno da base dello schema del cruciverba per i 3 esempi creati
32     //proposto un miglioramento a questa implementazione all'interno della relazione (nella sezione Scelte implementative)
33     //esempio1
34     private static char[][] matriceSchemaCruciverba;
35     private ArrayList<JTextField> text;
36     private static ArrayList<String> dizionarioInput;
37     private static String parolaIniziale;
38     private static int posizioneRigaIniziale;
39     private static int posizioneColonnaIniziale;
40     private static char orientamento;
41
42     public InterfacciaCruciverba() {
43         createUIComponents();
44         buttonRisolviCruciverba.addActionListener(new ActionListener() {
45             @Override
46             public void actionPerformed(ActionEvent e) {
47                 startTime=System.currentTimeMillis();
48
49                 //lancia procedura di risoluzione cruciverba e ritornando se è stata trovata la soluzione o meno, stampando un messaggio a
50                 video
51                 if (cruciverba1.risolviCruciverba()) {
52                     listListaParole.setListData(cruciverba1.dizionario.toArray());
53                     stopTime=System.currentTimeMillis();
54                     totalTime=stopTime-startTime;
55                     JOptionPane.showMessageDialog(null, "Cruciverba completato in " + totalTime, "Risultato cruciverba",
56                     JOptionPane.INFORMATION_MESSAGE);
57                 } else {
58                     listListaParole.setListData(cruciverba1.dizionario.toArray());
59                     JOptionPane.showMessageDialog(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.ERROR_MESSAGE);
60                 }
61             }
62         });
63     }
64     buttonCercaParola.addActionListener(new ActionListener() {
65         @Override
66         public void actionPerformed(ActionEvent e) {
67
68             //lancia procedura inserisci1Parola per cui esegue un passo dell'algoritmo e inserisce la parola trovata, se non trova nessuna
69             // parola controlla il risultato del cruciverba
70             String parolaInserita=cruciverba1.inserisci1Parola();
71             if (parolaInserita==null) {
72                 listListaParole.setListData(cruciverba1.dizionario.toArray());
73                 stopTime=System.currentTimeMillis();
74                 totalTime=stopTime-startTime;
75                 if (cruciverba1.isAlgResult()) {
76                     JOptionPane.showMessageDialog(null, "Cruciverba completato in " + totalTime, "Risultato cruciverba",
77                     JOptionPane.INFORMATION_MESSAGE);
78                 }else{
79                     JOptionPane.showMessageDialog(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.ERROR_MESSAGE);
80                 }
81             }
82         } else {
83             textFieldParolaInserita.setText(parolaInserita);
84             listListaParole.setListData(cruciverba1.dizionario.toArray());
85             //JOptionPane.showMessageDialog(null, "Cruciverba non completato", "Risultato cruciverba", JOptionPane.ERROR_MESSAGE);
86         }
87     }
88 }
89 }
90 }
```

```

91 public static void main(String[] args) {
92     JFrame frame = new JFrame("Cruciverba");
93     InterfacciaCruciverba window;
94
95     //importazione dello schema del cruciverba da un file .txt per i 3 esempi creati
96     //la prima riga del file .txt contiene le righe dello schema, la seconda riga contiene le colonne dello schema
97     //nelle successive righe si indica con . per le caselle bianche dello schema, * per le caselle nere
98
99     //come descritto nella sezione Scelte implementative della relazione, questa implementazione sarebbe
100     //migliorabile con una procedura di analisi dell'immagine
101     try{
102         //esempio1
103         File schema = new File("./schema1.txt");
104
105         //esempio2
106         //File schema = new File("./schema2.txt");
107
108         //esempio3
109         //File schema = new File("./schema3.txt");
110
111         if(schema.isFile()){
112             importaSchemaCruciverba(schema);
113         }
114     }
115
116 }catch(NullPointerException e){
117     System.out.println("Percorso file errato");
118     System.exit(-1);
119 }
120
121 window= new InterfacciaCruciverba();
122
123 frame.setContentPane(window.panelMain);
124 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
125 frame.pack();
126
127 //imposto la dimensione della finestra a seconda del numero di righe e di colonne del cruciverba, se sono troppo basse imposto una
128 //dimensione minima
129 int dimensioneFinestra=20*(matriceSchemaCruciverba.length+ matriceSchemaCruciverba[0].length)+100;
130 if (dimensioneFinestra<dimensioneFinestraMinima){
131     frame.setSize(dimensioneFinestraMinima, dimensioneFinestraMinima);
132 }else{
133     frame.setSize(dimensioneFinestra, dimensioneFinestra);
134 }
135 frame.setVisible(true);
136
137 try {
138     dizionarioInput = new ArrayList<String>();
139
140     //importo lista di parole che potranno essere inserite nello schema del cruciverba per i 3 esempi creati
141     //la prima riga del file .txt contiene la prima parola da inserire nello schema, con i riferimenti riga,
142     //colonna e orientamento.
143     //Come descritto nella sezione Scelte implementative della relazione, questa implementazione sarebbe
144     //migliorabile creando una procedura di selezione di un file .txt
145
146     //esempio1
147     //File listaParole = new File("./esempio1.txt");
148
149     //esempio2
150     //File listaParole = new File("./esempio2.txt");
151
152     //esempio3
153     //File listaParole = new File("./esempio3.txt");
154
155     //esempio4 - dizionario italiano completo
156     File listaParole = new File("./21000_parole_italiane.txt");
157
158     if (listaParole.isFile()) {
159         importaDizionario(listaParole);
160     }
161 }catch(NullPointerException e){
162     System.out.println("Percorso file errato");
163     System.exit(-1);
164 }
165
166 //window.createUIComponents();
167 window.open();
168 frame.setContentPane(window.panelMain);
169
170 }
171
172 public void open() {
173
174     //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo1
175     //cruciverba1 = new ImplAlg1Cruciverba(panelMain, matriceSchemaCruciverba, parolaIniziale, posizioneRigaIniziale,
176     posizioneColonnaIniziale, dizionarioInput, orientamento);
177
178     //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo2
179     //cruciverba1=new ImplAlg2Cruciverba(panelMain, matriceSchemaCruciverba, parolaIniziale, posizioneRigaIniziale,
180     posizioneColonnaIniziale, dizionarioInput, orientamento);
181
182     //creazione cruciverba per l'utilizzo di funzioni dell'algoritmo4
183     cruciverba1 = new ImplAlg4Cruciverba_AI(panelMain,matriceSchemaCruciverba, parolaIniziale, posizioneRigaIniziale,
184     posizioneColonnaIniziale, dizionarioInput, orientamento);
185

```

```

186         listaParole.setListData(dizionarioInput.toArray());
187     }
188 }
189
190
191 private void createUIComponents() {
192     // creo tutti gli elementi grafici dell'interfaccia utente per il cruciverba escluse tutte le caselle bianche e nere che creo in futuro
193     panelMain = new JPanel();
194     GroupLayout layoutGUI = new GroupLayout(panelMain);
195     panelMain.setLayout(layoutGUI);
196     layoutGUI.setAutoCreateGaps(true);
197     layoutGUI.setAutoCreateContainerGaps(true);
198
199     labelCruciverba = new JLabel("Cruciverba " + matriceSchemaCruciverba.length + "X" + matriceSchemaCruciverba[0].length);
200     labelCruciverba.setBounds(20, 20, 400, 20);
201
202     buttonRisolviCruciverba = new JButton("RisolviCruciverba");
203     buttonRisolviCruciverba.setBounds(20, 40, 200, 20);
204     buttonCercaParola = new JButton("CercaParola");
205     buttonCercaParola.setBounds(240, 40, 200, 20);
206
207     listaParole=new JList();
208     listaParole.setBounds(20, 80, 200,200);
209
210     scrollPanelListaParole = new JScrollPane(listaParole);
211     scrollPanelListaParole.setPreferredSize(new Dimension(300,200));
212     scrollPanelListaParole.setBounds(20,80,800,800);
213
214     panelListaParole = new JPanel();
215     BorderLayout groupLayoutListaParole = new BorderLayout();
216     panelListaParole.setLayout(groupLayoutListaParole);
217     panelListaParole.add(scrollPanelListaParole);
218     panelListaParole.setBounds(20,80,200,200);
219
220     labelParolaInserita = new JLabel("Ultima parola inserita");
221     labelParolaInserita.setBounds(20, 300, 400, 20);
222
223     textFieldParolaInserita=new JTextField();
224     textFieldParolaInserita.setSize(200, 20);
225     textFieldParolaInserita.setLocation(20, 320);
226
227
228     panelMain.add(labelCruciverba);
229     panelMain.add(buttonRisolviCruciverba);
230     panelMain.add(buttonCercaParola);
231     panelMain.add(panelListaParole);
232     panelMain.add(labelParolaInserita);
233     panelMain.add(textFieldParolaInserita);
234     panelMain.revalidate();
235 }
236
237
238 //funzione per importazione schema del cruciverba e inizializzazione variabile matriceSchemaCruciverba
239 private static void importaSchemaCruciverba(File f){
240     try {
241         Scanner s = new Scanner(f);
242         if (s == null) {
243             System.out.println("Scanner non creato");
244         } else {
245
246             if (s.hasNextInt()) {
247                 int nRighe = Integer.valueOf(s.next());
248
249                 if (s.hasNextInt()) {
250                     int nColonne = Integer.valueOf(s.next());
251
252                     //creo matrice di caratteri che utilizzerò per creare lo schema del cruciverba
253                     char[][] matriceCruciverba = new char[nRighe][nColonne];
254                     for(int i=0; i<nRighe; i++){
255                         if(s.hasNextLine()){
256                             s.nextLine();
257                             for(int j=0; j<nColonne; j++){
258                                 if(s.hasNext()){
259                                     matriceCruciverba[i][j]=s.next().charAt(0);
260                                 }
261                             }
262                         }
263                     }
264                     matriceSchemaCruciverba = matriceCruciverba;
265
266                 } else {
267                     System.out.println("Numero di colonne dello schema non inserito");
268                     throw new InputMismatchException();
269                 }
270             } else {
271                 System.out.println("Numero di righe dello schema non inserito");
272                 throw new InputMismatchException();
273             }
274         }
275         s.close();
276     }
277 }
278 }catch(FileNotFoundException e){
279     System.out.println("File not found exception");
280     System.exit(-1);

```

```

281     }catch(InputMismatchException e){
282         System.exit(-1);
283     }
284 }
285
286 //funzione per importazione delle parole da inserire all'interno dello schema all'interno del dizionario
287 private static void importaDizionario(File f){
288     try {
289         Scanner s = new Scanner(f);
290         if (s == null) {
291             System.out.println("Scanner non creato");
292         } else {
293             //imposto il delimitatore per poter analizzare la prima riga contenente la parola da inserire per prima nello schema
294             // con la sua posizione e il suo orientamento
295             //s.useDelimiter(",");
296             if (s.hasNext()) {
297                 parolaIniziale = s.next();
298                 if (s.hasNextInt()) {
299                     posizioneRigaIniziale = Integer.valueOf(s.next());
300                     if (s.hasNextInt()) {
301                         posizioneColonnaIniziale = Integer.valueOf(s.next());
302                         if (s.hasNext()) {
303                             orientamento = s.next().charAt(0);
304                             s.nextLine();
305                             //inserimento delle parole nella lista di parole disponibili
306                             while (s.hasNextLine()) {
307                                 String riga = s.nextLine();
308                                 dizionarioInput.add(riga);
309                             }
310                         } else {
311                             System.out.println("Nessuna orientamento inserito");
312                             throw new InputMismatchException();
313                         }
314                     } else {
315                         System.out.println("Nessuna posizione colonna iniziale inserita");
316                         throw new InputMismatchException();
317                     }
318                 } else {
319                     System.out.println("Nessuna posizione riga iniziale inserita");
320                     throw new InputMismatchException();
321                 }
322             } else {
323                 System.out.println("Nessuna parola iniziale inserita");
324                 throw new InputMismatchException();
325             }
326         }
327         s.close();
328     }
329 }catch(FileNotFoundException e){
330     System.out.println("File not found exception");
331     System.exit(-1);
332 }catch(InputMismatchException e){
333     System.exit(-1);
334 }
335 }
336 }

```

Classe Schema:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.util.ArrayList;
7
8 public class Schema {
9     private ArrayList<Parola> paroleSchema;
10    private ArrayList<Casella> caselleSchema;
11
12
13    //costruttore che prende la matrice in ingresso e crea lo schema aggiungendo la prima parola
14    public Schema(JPanel panel, char matrice[][], String parolaIniziale, Posizione posizioneParolaIniziale, char orientamentoInput) {
15        //inizializzazione variabili classe
16        paroleSchema = new ArrayList<Parola>();
17        caselleSchema = new ArrayList<Casella>();
18        ArrayList<Casella> caselleParola = new ArrayList<Casella>();
19        Casella casellaAttuale;
20        StringBuilder creazioneParola = new StringBuilder();
21        Posizione posizioneIniziale = new Posizione();
22        Posizione posizioneAttuale = new Posizione();
23        Parola parolaCreato;
24        int lunghezzaParola = 0;
25        char carattere = '.';
26        char casellaNera = '*';
27        boolean primaLettera = true;
28
29        //ricerca parole orizzontali
30        for (int i = 0; i < matrice.length; i++) {
31            for (int j = 0; j < matrice[0].length; j++) {
32                //inizializzazione posizione casella
33                posizioneAttuale.setRiga(i);
34                posizioneAttuale.setColonna(j);
35
36                //ricerca della prima lettera della parola che verrà inserita nello schema
37                if (matrice[i][j] == carattere && primaLettera) {
38                    creazioneParola.append(' ');
39                    primaLettera = false;
40                    posizioneIniziale.setRiga(i);
41                    posizioneIniziale.setColonna(j);
42                    lunghezzaParola++;
43
44                    //inizializzazione casella
45                    casellaAttuale = new Casella(panel, posizioneAttuale, matrice[i][j], false);
46
47                    caselleSchema.add(casellaAttuale);
48                    caselleParola.add(casellaAttuale);
49                } else if (matrice[i][j] == carattere && (!primaLettera)) { //ricerca lettere successive della parola che verrà inserita nello
50schema
51                    creazioneParola.append(' ');
52                    lunghezzaParola++;
53
54                    //inizializzazione casella
55                    casellaAttuale = new Casella(panel, posizioneAttuale, matrice[i][j], false);
56
57                    caselleSchema.add(casellaAttuale);
58                    caselleParola.add(casellaAttuale);
59                } else if (matrice[i][j] == casellaNera) { //ricerca delle caselle nere dello schema
60                    if (lunghezzaParola >= 2) { //Lunghezza minima di una parola raggiunta, la inserisco nello schema così come la casella
61nera e ripristino i valori iniziali
62                        caselleSchema.add(new Casella(panel, posizioneAttuale, matrice[i][j], true));
63                        parolaCreato = new Parola(creazioneParola.toString(), posizioneIniziale, 'O', lunghezzaParola, caselleParola);
64                        paroleSchema.add(parolaCreato);
65                        creazioneParola = new StringBuilder();
66                        posizioneIniziale.ripristinaPosizione();
67                        lunghezzaParola = 0;
68                        primaLettera = true;
69                        caselleParola = new ArrayList<Casella>();
70                    } else { // Lunghezza minima di una parola non raggiunta, inserisco solo la casella nera e ripristino i valori iniziali
71                        caselleSchema.add(new Casella(panel, posizioneAttuale, matrice[i][j], true));
72                        creazioneParola = new StringBuilder();
73                        posizioneIniziale.ripristinaPosizione();
74                        lunghezzaParola = 0;
75                        primaLettera = true;
76                        caselleParola = new ArrayList<Casella>();
77                    }
78                }
79            }
80        }
81        if (lunghezzaParola >= 2) { //raggiunta la fine della riga, inserisco la parola nello schema se ha la lunghezza minima altrimenti
82no
83            parolaCreato = new Parola(creazioneParola.toString(), posizioneIniziale, 'O', lunghezzaParola, caselleParola);
84            paroleSchema.add(parolaCreato);
85            creazioneParola = new StringBuilder();
```

```

86         posizioneIniziale.ripristinaPosizione();
87         lunghezzaParola = 0;
88         primaLettera = true;
89         caselleParola = new ArrayList<Casella>();
90     } else {
91         creazioneParola = new StringBuilder();
92         posizioneIniziale.ripristinaPosizione();
93         lunghezzaParola = 0;
94         primaLettera = true;
95         caselleParola = new ArrayList<Casella>();
96     }
97 }
98
99 //ricerca parole verticali, stessa procedura di quelle orizzontali con l'aggiunta del controllo per le caselle già presenti
100 for (int j = 0; j < matrice[0].length; j++) {
101     for (int i = 0; i < matrice.length; i++) {
102         //ricerca della prima lettera della parola che verrà inserita nello schema
103         if (matrice[i][j] == carattere && primaLettera) {
104             creazioneParola.append(' ');
105             primaLettera = false;
106             posizioneIniziale.setRiga(i);
107             posizioneIniziale.setColonna(j);
108             lunghezzaParola++;
109
110             //cerco la casella già esistente relativa alla riga i, colonna j per poi inserirla nella parola ed averla
111             // collegata alle caselle dello schema, così come alle parole in orizzontale
112             cercaCasella(caselleParola, i, j);
113
114         } else if (matrice[i][j] == carattere && (!primaLettera)) {
115             creazioneParola.append(' ');
116             lunghezzaParola++;
117             cercaCasella(caselleParola, i, j);
118         } else if (matrice[i][j] == casellaNera) {
119             if (lunghezzaParola >= 2) {
120                 parolaCreata = new Parola(creazioneParola.toString(), posizioneIniziale, 'V', lunghezzaParola, caselleParola);
121                 paroleSchema.add(parolaCreata);
122                 creazioneParola = new StringBuilder();
123                 posizioneIniziale.ripristinaPosizione();
124                 lunghezzaParola = 0;
125                 primaLettera = true;
126                 caselleParola = new ArrayList<Casella>();
127             } else {
128                 creazioneParola = new StringBuilder();
129                 posizioneIniziale.ripristinaPosizione();
130                 lunghezzaParola = 0;
131                 primaLettera = true;
132                 caselleParola = new ArrayList<Casella>();
133             }
134         }
135     }
136     if (lunghezzaParola >= 2) {
137         parolaCreata = new Parola(creazioneParola.toString(), posizioneIniziale, 'V', lunghezzaParola, caselleParola);
138         paroleSchema.add(parolaCreata);
139         creazioneParola = new StringBuilder();
140         posizioneIniziale.ripristinaPosizione();
141         lunghezzaParola = 0;
142         primaLettera = true;
143         caselleParola = new ArrayList<Casella>();
144     } else {
145         creazioneParola = new StringBuilder();
146         posizioneIniziale.ripristinaPosizione();
147         lunghezzaParola = 0;
148         primaLettera = true;
149         caselleParola = new ArrayList<Casella>();
150     }
151 }
152 //aggiorna schema con la prima parola iniziale data in input
153 aggiornaSchema(parolaIniziale, posizioneParolaIniziale, orientamentoInput);
154 }
155
156 //metodo costruttore schema a partire da uno schema già esistente
157 public Schema(Schema s){
158     this.paroleSchema=s.getParoleSchema();
159     this.caselleSchema=s.getCaselleSchema();
160 }
161
162 public ArrayList<Parola> getParoleSchema() { return new ArrayList<Parola>(paroleSchema); }
163
164 public ArrayList<Casella> getCaselleSchema(){ return new ArrayList<Casella>(caselleSchema); }
165
166 // inserisce la parola in input nella parola dello schema con la stessa posizione e orientamento
167 public void aggiornaSchema(String parola, Posizione posizioneParola, char orientamentoInput) {
168     Parola p = new Parola(parola, posizioneParola, orientamentoInput, parola.length());
169     for (Parola parolaSchema : paroleSchema) {
170         if (parolaSchema.confrontaCaselle(p)) {
171             parolaSchema.setParola(parola);
172             parolaSchema.setLunghezza(parola.length());
173             parolaSchema.aggiornaCaselleParola();

```

```

174         //paroleSchema.set(paroleSchema.indexOf(parolaSchema), p);
175         break;
176     }
177 }
178
179 //dopo aver aggiornato una parola dello schema devo fare in modo di aggiornare le parole dello schema che hanno lettere collegate
180 // alla parola appena aggiornata
181 for (Parola parolaSchema : paroleSchema){
182     parolaSchema.aggiornaParola();
183 }
184
185 }
186
187 //cerca all'interno delle caselle dello schema e se già presente una casella con posizione riga, colonna allora la
188 // assegna alle caselleParola senza creare una nuova casella apposita
189 public void cercaCasella(ArrayList<Casella> caselleParola, int riga, int colonna) {
190     for (Casella c : caselleSchema) {
191         if (c.confrontaPosizione(riga, colonna)) {
192             caselleParola.add(c);
193         }
194     }
195 }
196
197 //cerca le parole dello schema di lunghezza n non ancora completate
198 public ArrayList<Parola> ricercaLunghezzaParole(int n){
199     ArrayList<Parola> paroleLunghezzaN = new ArrayList<Parola>();
200     for (Parola p : paroleSchema){
201         int lunghezzaParola=p.getLunghezza();
202         int lettereInserite=p.getLettereInserite();
203         if (lunghezzaParola==n && lettereInserite<lunghezzaParola){
204             paroleLunghezzaN.add(p);
205         }
206     }
207     return paroleLunghezzaN;
208 }
209
210 //cerca la lunghezza massima tra le parole dello schema
211 public int cercaLunghezzaParolaMax(){
212     int lunghezzaMax=0;
213     for (Parola p : paroleSchema){
214         int lunghezzaCorrente=p.getLunghezza();
215         if (lunghezzaCorrente > lunghezzaMax){
216             lunghezzaMax=lunghezzaCorrente;
217         }
218     }
219     return lunghezzaMax;
220 }
221
222 }

```


Classe Parola:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.util.ArrayList;
7
8 public class Parola {
9     private String parola;
10    private ArrayList<Casella> caselleParola;
11    private Posizione posizioneParola;
12    private int lunghezza;
13    private int lettereInserite;
14    private char orientamento;
15
16    //metodo costruttore crea una parola con le relative informazioni, senza caselleParola collegate
17    public Parola(String parola, Posizione posizioneIniziale, char orientamento, int lunghezza) {
18        try {
19            //controllo parola
20            if (parola.length() > 0) {
21                this.parola = parola;
22            } else {
23                throw new Exception("Parola non corretta.");
24            }
25
26            //controllo posizione
27            if (posizioneIniziale.getRiga() >= 0 && posizioneIniziale.getColonna() >= 0) {
28                posizioneParola = new Posizione(posizioneIniziale);
29            } else {
30                throw new Exception("Posizione indicata non corretta.");
31            }
32
33            //controllo orientamento
34            if (orientamento == 'V' || orientamento == 'O') {
35                this.orientamento = orientamento;
36            } else {
37                throw new Exception("Orientamento (verticale o orizzontale) non corretto.");
38            }
39
40            //controllo lunghezza
41            if (lunghezza > 1) {
42                this.lunghezza = lunghezza;
43            } else {
44                throw new Exception("Lunghezza non corretta. Vengono inserite le parole con almeno 2 lettere");
45            }
46
47        } catch (Exception ex) {
48            JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
49        }
50    }
51
52
53    //metodo costruttore crea una parola con le relative informazioni con le caselleParola collegate
54    public Parola(String parola, Posizione posizioneIniziale, char orientamento, int lunghezza, ArrayList<Casella> caselleParolaInput) {
55        try {
56
57            //inizializzo le caselle della parola con le caselle dello schema create in precedenza
58            if (caselleParolaInput != null) {
59                caselleParola = caselleParolaInput;
60            }
61
62            //controllo parola
63            if (parola.length() > 0) {
64                this.parola = parola;
65            } else {
66                throw new Exception("Parola non corretta.");
67            }
68
69            //controllo posizione
70            if (posizioneIniziale.getRiga() >= 0 && posizioneIniziale.getColonna() >= 0) {
71                posizioneParola = new Posizione(posizioneIniziale);
72            } else {
73                throw new Exception("Posizione indicata non corretta.");
74            }
75
76            //controllo orientamento
77            if (orientamento == 'V' || orientamento == 'O') {
78                this.orientamento = orientamento;
79            } else {
80                throw new Exception("Orientamento (verticale o orizzontale) non corretto.");
81            }
82
83            //controllo lunghezza
84            if (lunghezza > 1) {
85                this.lunghezza = lunghezza;
86            } else {
87                throw new Exception("Lunghezza non corretta. Vengono inserite le parole con almeno 2 lettere");
88            }
89
90        } catch (Exception ex) {
91            JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
92        }
93    }
94 }
```

```

94 //metodo costruttore crea una parola a partire da una passata in input
95 public Parola(Parola p){
96     this.setParola(p.getParola());
97     this.setCaselleParola(p.getCaselleParola());
98     this.setPosizioneParola(p.getPosizioneParola());
99     this.setLunghezza(p.getLunghezza());
100     this.setLettereInserite(p.getLettereInserite());
101     this.setOrientamento(p.getOrientamento());
102 }
103
104 public ArrayList<Casella> getCaselleParola() {
105     return new ArrayList<Casella>(caselleParola);
106 }
107
108 public void setCaselleParola(ArrayList<Casella> caselleParola) {
109     this.caselleParola = caselleParola;
110 }
111
112 public String getParola() {
113     return parola;
114 }
115
116 public void setParola(String parola) {
117     this.parola = parola;
118 }
119
120 public Posizione getPosizioneParola() {
121     return posizioneParola;
122 }
123
124 public void setPosizioneParola(Posizione posizioneParola) {
125     this.posizioneParola = posizioneParola;
126 }
127
128 public int getLunghezza() {
129     return lunghezza;
130 }
131
132 public void setLunghezza(int lunghezza) {
133     this.lunghezza = lunghezza;
134 }
135
136 public char getOrientamento() {
137     return orientamento;
138 }
139
140 public void setOrientamento(char orientamento) {
141     this.orientamento = orientamento;
142 }
143
144 public int getLettereInserite() {
145     return lettereInserite;
146 }
147
148 public void setLettereInserite(int lettereInserite) {
149     this lettereInserite = lettereInserite;
150 }
151
152 //controlla se le due parole corrispondono alla stessa casella nello schema, sia come orientamento che come inizio e lunghezza parola
153 public boolean confrontaCaselle(Parola p) {
154     if (this.orientamento == p.orientamento && posizioneParola.equals(p.getPosizioneParola()) && this.lunghezza == p.lunghezza) {
155         return true;
156     } else {
157         return false;
158     }
159 }
160
161 //aggiorna il testo delle caselle con la stringa dentro parola
162 public void aggiornaCaselleParola(){
163     lettereInserite=0;
164     try{
165         if (lunghezza==caselleParola.size()){
166             for (int i=0; i<lunghezza; i++){
167                 char currentChar = parola.charAt(i);
168                 if (currentChar!='.'){
169                     lettereInserite++;
170                     caselleParola.get(i).aggiornaCarattere(currentChar);
171                 }
172             }
173         }else{
174             throw new Exception ("Lunghezza parola diversa dalla lunghezza della parola nel cruciverba");
175         }
176     } catch (Exception ex){
177         JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
178     }
179 }
180
181 //aggiorna la stringa dentro parola con il testo delle caselle, contando anche le lettere inserite
182 public boolean aggiornaParola(){
183     StringBuilder strParolaNuova=new StringBuilder();
184     boolean result=false;
185     lettereInserite=0;

```

```

189     try{
190         if (lunghezza==caselleParola.size()){
191             for (int i=0; i<lunghezza; i++){
192                 char carattereCasella=caselleParola.get(i).getCarattereCasella();
193                 if (carattereCasella!='.'){
194                     lettereInserite++;
195                     result=true;
196                 }
197                 strParolaNuova.append(carattereCasella);
198             }
199             parola=strParolaNuova.toString();
200         }else{
201             throw new Exception ("Lunghezza parola diversa dalla lunghezza della parola nel cruciverba");
202         }
203         return result;
204     }
205     catch (Exception ex){
206         JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
207         return false;
208     }
209 }
210
211 //controlla se una parola è stata completata
212 public boolean isComplete(){
213     if (this.lunghezza==this.lettereInserite){
214         return true;
215     }else{
216         return false;
217     }
218 }
219
220 //controlla se esistono caselle uguali tra due parole
221 public boolean isLinked(Parola p){
222     for (Casella casellaParolaCorrente : caselleParola){
223         for (Casella casellaParolaDaConfrontare : p.getCaselleParola()){
224             if (casellaParolaCorrente.confrontaCaselle(casellaParolaDaConfrontare)){
225                 return true;
226             }
227         }
228     }
229     return false;
230 }
231
232 //Ripristino i valori iniziali della parola, quindi tutti i caratteri a "." e le lettere inserite=0
233 public void resetParola(){
234     StringBuilder strParolaNuova=new StringBuilder();
235     boolean result=false;
236     lettereInserite=0;
237     try{
238         if (lunghezza==caselleParola.size()){
239             for (int i=0; i<lunghezza; i++){
240                 strParolaNuova.append('.');
241             }
242             lettereInserite=0;
243             parola=strParolaNuova.toString();
244         }else{
245             throw new Exception ("Lunghezza parola diversa dalla lunghezza della parola nel cruciverba");
246         }
247     }
248     catch (Exception ex){
249         JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
250     }
251 }
252
253 }

```

Classe Casella:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.awt.*;
7
8 public class Casella {
9     private Posizione posizioneCasella;
10    private JTextField textFieldCasella;
11    private char carattereCasella;
12    private boolean casellaNera;
13
14    //costruttore casella con input il JPanel dove verrà inserita la casella
15    public Casella(JPanel panel, Posizione posizioneInput, char carattereInput, boolean casellaNeraInput) {
16        int xIniziale = 240;
17        int yIniziale = 80;
18        int i, j = 0;
19        try {
20            if (posizioneInput != null) {
21                posizioneCasella = new Posizione(posizioneInput);
22                i = posizioneCasella.getRiga();
23                j = posizioneCasella.getColonna();
24            } else {
25                throw new Exception("Posizione non esistente");
26            }
27            carattereCasella = carattereInput;
28            casellaNera = casellaNeraInput;
29            textFieldCasella = new JTextField();
30            textFieldCasella.setSize(20, 20);
31            //calcolo la posizione della casella a partire dalla posizione passata in input
32            textFieldCasella.setLocation(xIniziale + j * 20, yIniziale + i * 20);
33            textFieldCasella.setText(String.valueOf(carattereCasella));
34
35
36            //imposto lo sfondo della casella nera e la disabilito
37            if (casellaNera) {
38                textFieldCasella.setBackground(new Color(0, 0, 0));
39                textFieldCasella.setEditable(false);
40                textFieldCasella.setFocusable(false);
41            }
42
43            //aggancio il textField al pannello passato in input
44            panel.add(textFieldCasella);
45        } catch (Exception ex) {
46            JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
47        }
48    }
49
50    public char getCarattereCasella() {
51        return carattereCasella;
52    }
53
54    public void setCarattereCasella(char carattereCasella) {
55        this.carattereCasella = carattereCasella;
56    }
57
58    //controlla ritornando true o false se la posizione della casella dell'oggetto corrente è uguale a quella passata in input
59    public boolean confrontaPosizione(int riga, int colonna) {
60        Posizione posizioneDaConfrontare = new Posizione(riga, colonna);
61        return posizioneCasella.equals(posizioneDaConfrontare);
62    }
63
64    //procedura di confronto caselle, ritorna true se la casella in input è la stessa della casella dell'oggetto corrente
65    public boolean confrontaCaselle(Casella c){
66        return posizioneCasella.equals(c.posizioneCasella);
67    }
68
69    //aggiorna il carattere dell'oggetto corrente e il testo del componente visualizzato a video
70    public void aggiornaCarattere(char carattereInput){
71        carattereCasella=carattereInput;
72        textFieldCasella.setText(String.valueOf(carattereInput));
73    }
74 }
```

Classe Posizione:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 public class Posizione {
6     int riga;
7     int colonna;
8
9     //costruttore posizione senza valori in input, inserisco valori standard
10    public Posizione() {
11        this.riga = -1;
12        this.colonna = -1;
13    }
14
15    //costruttore posizione a partire da valori in input
16    public Posizione(int rigaInput, int colonnaInput) {
17        this.riga = rigaInput;
18        this.colonna = colonnaInput;
19    }
20
21    //costruttore posizione da un'altra posizione già esistente
22    public Posizione(Posizione posizioneInput) {
23        this.riga = posizioneInput.getRiga();
24        this.colonna = posizioneInput.getColonna();
25    }
26
27    public int getRiga() {
28        return riga;
29    }
30
31    public void setRiga(int riga) {
32        this.riga = riga;
33    }
34
35    public int getColonna() {
36        return colonna;
37    }
38
39    public void setColonna(int colonna) {
40        this.colonna = colonna;
41    }
42
43    //ripristina valori di default
44    public void ripristinaPosizione() {
45        this.riga = -1;
46        this.colonna = -1;
47    }
48
49    //controlla se la posizione passata in input corrisponde alla stessa riga e colonna dell'oggetto corrente
50    public boolean equals(Posizione posizioneDaConfrontare) {
51        if (this.riga == posizioneDaConfrontare.getRiga() && this.colonna == posizioneDaConfrontare.getColonna()) {
52            return true;
53        } else {
54            return false;
55        }
56    }
57 }
```

Classe ImplementazioneCruciverba:

```
1 package com.cruciverbapackage;
2 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
3 import javax.imageio.plugins.tiff.ExifTIFFTagSet;
4 import javax.swing.*;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import java.lang.reflect.Array;
8 import java.util.ArrayList;
9
10 public class ImplementazioneCruciverba {
11     protected Schema schema_originale;
12     protected ArrayList<String> dizionario;
13     protected boolean algResult;
14     protected boolean algExecuted;
15     //costruttore cruciverba con una struttura passata in input
16     public ImplementazioneCruciverba(JPanel panel, char matrice[][], String parolaIniziale, int posizioneRigaIniziale, int
17     posizioneColonnaIniziale, ArrayList<String> dizionarioInput, char orientamento) {
18         schema_originale = new Schema(panel, matrice, parolaIniziale, new Posizione(posizioneRigaIniziale, posizioneColonnaIniziale),
19     orientamento);
20         if (dizionarioInput != null && dizionarioInput.size() != 0) {
21             dizionario = dizionarioInput;
22         }
23     }
24
25     public ArrayList<String> getDizionario() {
26         return dizionario;
27     }
28
29     public boolean isAlgResult() {
30         return algResult;
31     }
32
33     public boolean isAlgExecuted() {
34         return algExecuted;
35     }
36
37     // inserisce la parola all'interno del cruciverba nella riga e nella colonna specificata e con quell'orientamento
38     public void aggiornaParola(String parola, int riga, int colonna, char orientamento) {
39         schema_originale.aggiornaSchema(parola, new Posizione(riga, colonna), orientamento);
40     }
41
42     // ricerca la prossima parola da inserire nel cruciverba
43     public String cercaParolaDaInserire(Parola casellaDaCompletare, ArrayList<String> dizionario){
44         return "";
45     }
46
47     //corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)
48     //inserisce una parola nello schema del cruciverba
49     public String inserisci1Parola(){ return null; }
50
51     //chiama cercaParolaDaInserire finche lo schema non è completato, cioè isComplete=true
52     public boolean risolviCruciverba(){
53         return true;
54     }
55
56     //cerco la parola all'interno della lista che ha più lettere già inserite, a parità di lettere già inserite prendo la prima che ho trovato
57     public Parola cercaParolaConPiuLettere(ArrayList<Parola> listaParole){
58         int maxLettereInserite=-1, contatoreLettereInserite=0;
59         Parola maxParolaLettereInserite=null;
60         for (Parola p : listaParole){
61             contatoreLettereInserite=p.getLettereInserite();
62             if (contatoreLettereInserite>maxLettereInserite){
63                 maxLettereInserite=contatoreLettereInserite;
64                 maxParolaLettereInserite=p;
65             }
66         }
67         return maxParolaLettereInserite;
68     }
69
70     // controllo se cruciverba è finito o no, quindi non risultano altre parole del dizionario da inserire
71     public boolean isComplete() {
72         if (dizionario.size()==0){
73             return true;
74         }else{
75             return false;
76         }
77     }
78
79     //aggiornamento dizionario con le parole dello schema COMPLETATE, in questo modo quelle che parole che si sono completate automaticamente
80     inserendo
81     //altre parole nello schema vengono eliminate dal dizionario
82     public void aggiornaDizionario(){
83         ArrayList<Parola> paroleSchema = schema_originale.getParoleSchema();
84         String parolaCorrente;
85         for (Parola p : paroleSchema){
86             parolaCorrente=p.getParola();
87             if (p.getLunghezza()==p.getLettereInserite() && dizionario.contains(parolaCorrente)){
88                 dizionario.remove(parolaCorrente);
89             }
90         }
91     }
92 }
93 }
```

Classe ImplAlg1Cruciverba:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.util.ArrayList;
7
8 public class ImplAlg1Cruciverba extends ImplementazioneCruciverba{
9
10     //costruttore cruciverba con una struttura passata in input che richiama semplicemente il costruttore della classe padre
11     public ImplAlg1Cruciverba(JPanel panel, char matrice[][], String parolaIniziale, int posizioneRigaIniziale, int posizioneColonnaIniziale,
12 ArrayList<String> dizionarioInput, char orientamento) {
13         super(panel,matrice,parolaIniziale,posizioneRigaIniziale,posizioneColonnaIniziale,dizionarioInput, orientamento);
14     }
15
16     // ricerca la prossima parola da inserire nel cruciverba con algoritmo1
17     public String cercaParolaDaInserire(Parola casellaDaCompletare, ArrayList<String> dizionario){
18         ArrayList<String> paroleDizionarioTrovate=new ArrayList<String>();
19         int i=0;
20         boolean parolaUguale;
21         String parolaDaCompletare=casellaDaCompletare.getParola();
22         int lunghezzaParolaDaCompletare = casellaDaCompletare.getLunghezza();
23         /*ciclo tutte le parole del dizionario in ricerca di una o più parole che possono entrare nelle caselle a disposizione,
24          * a seconda dei caratteri già inseriti,
25          * se nessuna parola viene trovata si ritorna una stringa vuota
26          * se ci sono più parole da poter inserire si ritorna anche qui una stringa vuota
27          * se ce n'è solo una invece ritorno la stringa da inserire all'interno di queste caselle
28          */
29
30         for (String s : dizionario){
31             parolaUguale=true;
32             //se la lunghezza delle caselle da completare è diversa da quelle della parola s del dizionario salto il ciclo che confronta i
33             caratteri
34             // delle due parole per vedere se sono compatibili
35             i=0;
36             if (s.length()==lunghezzaParolaDaCompletare){
37                 //confronto carattere per carattere quelle dell'oggetto Parola corrente con quelle del dizionario
38                 while (i<lunghezzaParolaDaCompletare && parolaUguale){
39                     char carattere = parolaDaCompletare.charAt(i);
40                     //confronto i caratteri solo se è un carattere valido (diverso da '.' impostato inizialmente)
41                     if (carattere!='.'){
42                         if (carattere!=s.charAt(i)){
43                             parolaUguale=false;
44                         }
45                     }
46                     i++;
47                 }
48                 if (parolaUguale){
49                     paroleDizionarioTrovate.add(s);
50                 }
51             }
52         }
53         if (paroleDizionarioTrovate.size()==1){
54             return paroleDizionarioTrovate.get(0);
55         }else{
56             return "";
57         }
58     }
59 }
60
61 //corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)
62 //inserisce una parola nello schema del cruciverba
63 public String inserisci1Parola(){
64     boolean trovataParola=false;
65     String parolaDaInserire=null;
66     int cicliEseguiti=0,cicliMax=100,c;
67     //analizzo le parole dello schema e prendo la lunghezza massima
68     int lunghezzaMax=schema_originale.cercaLunghezzaParolaMax();
69     int numeroParoleLunghezzaC=0,numeroParoleLunghezzaCInserite=0;
70     ArrayList<Boolean> trovato = new ArrayList<Boolean>();
71
72     ArrayList<Parola> ricercaParole;
73
74     //se già eseguito l'algoritmo ritorno null
75     if (isAlgExecuted()){
76         return null;
77     }
78     //imposto tutto l'arraylist trovato a false
79     for (int i=0; i<lunghezzaMax;i++){
80         trovato.add(false);
81     }
82
83     //ciclo finchè tutto l'arraylist trovato è uguale a true oppure fino ad arrivare a un'iterazione>cicliMax oppure se non ho trovato
84     //nessuna parola al ciclo precedente
85     while(daTrovare(trovato) && cicliEseguiti<cicliMax && !(trovataParola)){
86         cicliEseguiti++;
87         c=0;
88         //ciclo da 0 fino alla lunghezza massima delle parole nello schema oppure esco se non ho trovato nessuna parola al ciclo precedente
89         while(c<lunghezzaMax && !(trovataParola)){
90             //se ho già trovato tutte le parole con lunghezza c mi fermo e passo al numero c successivo
91             if (!(trovato.get(c))){
92                 ricercaParole=schema_originale.ricercaLunghezzaParole(c);
93             }
```

```

94         numeroParoleLunghezzaC=ricercaParole.size();
95         numeroParoleLunghezzaCInserite=0;
96         //ciclo Le parole di lunghezza c finchè la lista non è vuota oppure esco quando non ho trovato una parola al ciclo
97 precedente
98         while(ricercaParole.size()>0 && !(trovataParola)){
99             Parola casellaDaCompletare=cercaParolaConPiuLettere(ricercaParole);
100             ricercaParole.remove(casellaDaCompletare);
101             //chiamo la procedura di ricerca parola da inserire dell'algoritmo1
102             parolaDaInserire=cercaParolaDaInserire(casellaDaCompletare,dizionario);
103             if (!( parolaDaInserire.equals(""))){
104                 trovataParola=true;
105                 casellaDaCompletare.setParola(parolaDaInserire);
106                 //procedura per aggiornare lo schema con la parola trovata
107                 aggiornaParola(casellaDaCompletare.getParola(),casellaDaCompletare.getPosizioneParola().getRiga()
108                     , casellaDaCompletare.getPosizioneParola().getColonna(),casellaDaCompletare.getOrientamento());
109                 numeroParoleLunghezzaCInserite++;
110
111                 //aggiorno il dizionario togliendo la parola che è stata inserita nello schema
112                 // + eventuali parole che si sono autocompletate inserendo una parola nello schema
113                 aggiornaDizionario();
114             }
115
116         }
117         if (numeroParoleLunghezzaC==numeroParoleLunghezzaCInserite){
118             trovato.set(c,true);
119         }
120     }
121
122     //incremento il numero di caselle di cui voglio cercare le parole da inserire
123     c++;
124 }
125
126 }
127
128 //ritorno la parola se è stata trovata, altrimenti ritorno null ma lanciando prima la procedura di RisolviCruciverba
129 //per vedere se è stato completato correttamente o no
130 if (trovataParola){
131     return parolaDaInserire;
132 }else{
133     risolviCruciverba();
134     return null;
135 }
136
137 }
138
139 }
140
141 //risoluzione cruciverba attraverso l'utilizzo dell'algoritmo 1 e ritorno se è stato completato o no
142 public boolean risolviCruciverba(){
143
144     //se è già stato eseguito una volta non ripeto l'esecuzione ma ritorno il risultato ottenuto in precedenza
145     if(algExecuted){
146         return algResult;
147     }else {
148
149         int cicliEseguiti = 0, cicliMax = 100, c;
150         //analizzo le parole dello schema e prendo la lunghezza massima
151         int lunghezzaMax = schema_originale.cercaLunghezzaParolaMax();
152         int numeroParoleLunghezzaC = 0, numeroParoleLunghezzaCInserite = 0;
153         ArrayList<Boolean> trovato = new ArrayList<Boolean>();
154
155         ArrayList<Parola> ricercaParole;
156
157         //imposto tutto l'arrayList trovato a false
158         for (int i = 0; i <= lunghezzaMax; i++) {
159             trovato.add(false);
160         }
161
162         //ciclo finchè tutto l'arrayList trovato è uguale a true oppure fino ad arrivare a un'iterazione>cicliMax
163         while (daTrovare(trovato) && cicliEseguiti < cicliMax) {
164             cicliEseguiti++;
165             c = 0;
166             //ciclo da 0 fino alla lunghezza massima delle parole nello schema
167             while (c <= lunghezzaMax) {
168                 //se ho già trovato tutte le parole con lunghezza c mi fermo e passo al numero c successivo
169                 if (!(trovato.get(c))) {
170                     ricercaParole = schema_originale.cercaLunghezzaParole(c);
171                     numeroParoleLunghezzaC = ricercaParole.size();
172                     numeroParoleLunghezzaCInserite = 0;
173                     //ciclo Le parole di lunghezza c finchè la lista non è vuota
174                     while (ricercaParole.size() > 0) {
175                         Parola casellaDaCompletare = cercaParolaConPiuLettere(ricercaParole);
176                         ricercaParole.remove(casellaDaCompletare);
177                         //chiamo la procedura di ricerca parola da inserire dell'algoritmo1
178                         String parolaDaInserire = cercaParolaDaInserire(casellaDaCompletare, dizionario);
179                         if (!(parolaDaInserire.equals(""))){
180                             casellaDaCompletare.setParola(parolaDaInserire);
181
182                             //procedura per aggiornare lo schema con la parola trovata
183                             aggiornaParola(casellaDaCompletare.getParola(), casellaDaCompletare.getPosizioneParola().getRiga()
184                                 , casellaDaCompletare.getPosizioneParola().getColonna(), casellaDaCompletare.getOrientamento());
185                             numeroParoleLunghezzaCInserite++;
186
187                             //aggiorno il dizionario togliendo la parola che è stata inserita nello schema
188                             // + eventuali parole che si sono autocompletate inserendo una parola nello schema
189                             aggiornaDizionario();

```



```

189         }
190     }
191 }
192     if (numeroParoleLunghezzaC == numeroParoleLunghezzaCInserite) {
193         trovato.set(c, true);
194     }
195 }
196
197     //incremento il numero di caselle di cui voglio cercare le parole da inserire
198     c++;
199 }
200
201 }
202
203     //aggiorno la variabile per sapere che ho eseguito una volta l'algoritmo di RisolviCruciverba
204     algExecuted=true;
205     //faccio un controllo se ho completato tutte le parole dello schema, aggiornando la variabile che contiene il risultato del cruciverba
206     // e lo ritorno
207     if (!(daTrovare(trovato))){
208         algResult=true;
209         return algResult;
210     }else{
211         algResult=false;
212         return algResult;
213     }
214 }
215
216 }
217
218     //controllo se sono presenti valori a false dentro l'arraylist trovato, in quel caso significa che ancora non ho trovato tutte
219     //le parole di lunghezza i
220     public boolean daTrovare(ArrayList<Boolean> trovato){
221         int i=0;
222         try {
223             if (trovato!=null){
224                 while(i<trovato.size()){
225                     if (trovato.get(i)){
226                         i++;
227                     }else{
228                         return true;
229                     }
230                 }
231             }else{
232                 throw new NullPointerException("Trovato è null");
233             }
234         } catch (NullPointerException ex) {
235             JOptionPane.showMessageDialog(null, ex.toString(), "Errore", JOptionPane.ERROR_MESSAGE);
236             System.exit(101);
237         }
238     }
239     return false;
240 }
241
242
243     //cerco la parola all'interno della lista che ha più lettere già inserite, a parità di lettere già inserite prendo la prima che ho trovato
244     public Parola cercaParolaConPiuLettere(ArrayList<Parola> listaParole){
245         int maxLettereInserite=-1, contatoreLettereInserite=0;
246         Parola maxParolaLettereInserite=null;
247         for (Parola p : listaParole){
248             contatoreLettereInserite=p.getLettereInserite();
249             if (contatoreLettereInserite>maxLettereInserite){
250                 maxLettereInserite=contatoreLettereInserite;
251                 maxParolaLettereInserite=p;
252             }
253         }
254         return maxParolaLettereInserite;
255     }
256 }
257 }

```

Classe ImplAlg2Cruciverba:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.util.ArrayList;
7
8 public class ImplAlg2Cruciverba extends ImplementazioneCruciverba {
9
10     //costruttore cruciverba con una struttura passata in input che richiama il costruttore del parole
11     public ImplAlg2Cruciverba(JPanel panel, char matrice[][], String parolaIniziale, int posizioneRigaIniziale, int posizioneColonnaIniziale,
12 ArrayList<String> dizionarioInput, char orientamento) {
13         super(panel,matrice,parolaIniziale,posizioneRigaIniziale,posizioneColonnaIniziale,dizionarioInput, orientamento);
14     }
15
16     //ricerca la prossima parola da inserire nel cruciverba
17     public String cercaParolaDaInserire(Parola casellaDaCompletare, ArrayList<String> dizionario){
18         ArrayList<String> paroleDizionarioTrovate=new ArrayList<String>();
19         int i=0;
20         boolean parolaUguale;
21         String parolaDaCompletare=casellaDaCompletare.getParola();
22         int lunghezzaParolaDaCompletare = casellaDaCompletare.getLunghezza();
23         /*ciclo tutte le parole del dizionario in ricerca di una o più parole che possono entrare nelle caselle a disposizione,
24          * a seconda dei caratteri già inseriti,
25          * se nessuna parola viene trovata si ritorna una stringa vuota
26          * se ci sono più parole da poter inserire si ritorna anche qui una stringa vuota
27          * se ce n'è solo una invece ritorno la stringa da inserire all'interno di queste caselle
28          */
29
30         for (String s : dizionario){
31             parolaUguale=true;
32             //se la lunghezza delle caselle da completare è diversa da quelle della parola s del dizionario salto il ciclo che confronta i
33             caratteri
34             // delle due parole per vedere se sono compatibili
35             i=0;
36             if (s.length()==lunghezzaParolaDaCompletare){
37                 //confronto i caratteri della parola da completare con quelle delle parole nel dizionario, se non è presente
38                 //nessun vincolo violato allora inserisco la parola del dizionario nella lista di parola da inserire
39                 while (i<lunghezzaParolaDaCompletare && parolaUguale){
40                     char carattere = parolaDaCompletare.charAt(i);
41                     if (carattere!='.'){
42                         if (carattere!=s.charAt(i)){
43                             parolaUguale=false;
44                         }
45                     }
46                     i++;
47                 }
48                 if (parolaUguale){
49                     paroleDizionarioTrovate.add(s);
50                 }
51             }
52         }
53     }
54
55     //se ho trovato solo una parola da inserire la ritorno alla funzione chiamante
56     if (paroleDizionarioTrovate.size()==1){
57         return paroleDizionarioTrovate.get(0);
58     }else{
59         return "";
60     }
61 }
62
63 //corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)
64 //inserisce una parola nello schema del cruciverba
65 public String inserisci1Parola(){
66     boolean trovataParola=false;
67     String parolaDaInserire=null;
68
69     int cicliEseguiti = 0, cicliMax = 100, c, lunghezzaMax = schema_originale.cercaLunghezzaParolaMax();
70     int numeroParoleLunghezzaC = 0, numeroParoleLunghezzaCInserite = 0;
71
72     ArrayList<ArrayList<Parola>> listaParoleLunghezzaC = new ArrayList<ArrayList<Parola>>();
73
74     //se già eseguito l'algoritmo ritorno null
75     if (isAlgExecuted()){
76         return null;
77     }
78
79     //inserisco all'interno della lista delle parole di lunghezza i le parole di lunghezza i
80     for (int i = 0; i < lunghezzaMax; i++) {
81         ArrayList<Parola> paroleLunghezzaC;
82         paroleLunghezzaC = schema_originale.ricercaLunghezzaParole(i + 1);
83         if (paroleLunghezzaC.size() > 0) {
84             listaParoleLunghezzaC.add(i, paroleLunghezzaC);
85         } else {
86             listaParoleLunghezzaC.add(new ArrayList<Parola>());
87         }
88     }
89     //ciclo finché il cruciverba non è completo, fino ad aver fatto n>cicliMax iterazioni e se non trovo una parola da inserire
90     while (!isComplete() && cicliEseguiti < cicliMax && !(trovataParola)) {
91         cicliEseguiti++;
92         c = 0;
93         while (c < lunghezzaMax && !(trovataParola)) {
94             //prendo le parole di lunghezza=c su cui lavorerò
```

```

94 ArrayList<Parola> paroleLunghezzaC = listaParoleLunghezzaC.get(c);
95 int i=0;
96 while ( i < paroleLunghezzaC.size() && !(trovataParola)) {
97     Parola casellaDaCompletare = paroleLunghezzaC.get(i);
98     //prima di cercare la parolaDaInserire faccio un controllo se la parola nella lista paroleLunghezzaC è già completa,
99     // se sì la elimino dalla lista
100     if (casellaDaCompletare.getLunghezza() == casellaDaCompletare.getLettereInserite()) {
101         paroleLunghezzaC.remove(casellaDaCompletare);
102     } else {
103         parolaDaInserire = cercaParolaDaInserire(casellaDaCompletare, dizionario);
104         if (!(parolaDaInserire.equals(""))) {
105             //è stata trovata una parola da inserire nello schema, rimuovo quindi la casellaDaCompletare dalla lista di
106 parole di
107
108             // lunghezza c ancora da inserire
109             trovataParola=true;
110             paroleLunghezzaC.remove(casellaDaCompletare);
111
112             casellaDaCompletare.setParola(parolaDaInserire);
113             //aggiorno lo schema con la nuova parola trovata
114             aggiornaParola(casellaDaCompletare.getParola(), casellaDaCompletare.getPosizioneParola().getRiga()
115                 , casellaDaCompletare.getPosizioneParola().getColonna(), casellaDaCompletare.getOrientamento());
116
117             //aggiorno il dizionario togliendo la parola che è stata inserita nello schema
118             // + eventuali parole che si sono autocompletate inserendo una parola nello schema
119             aggiornaDizionario();
120         }else{
121             //incremento solo se la parola non era già completata o se non è stato inserito niente nello schema
122             i++;
123         }
124     }
125 }
126 c++;
127 }
128
129 //controllo se il cruciverba è stato completato o meno
130 if (trovataParola) {
131     return parolaDaInserire;
132 } else {
133     risolviCruciverba();
134     return null;
135 }
136
137 }
138
139 // risoluzione cruciverba attraverso l'utilizzo dell'algoritmo 2 ritorno true se il cruciverba è stato completato, altrimenti false
140 public boolean risolviCruciverba() {
141     int cicliEseguiti = 0, cicliMax = 100, c, lunghezzaMax = schema_originale.cercaLunghezzaParolaMax();
142     int numeroParoleLunghezzaC = 0, numeroParoleLunghezzaCInserite = 0;
143     ArrayList<ArrayList<Parola>> listaParoleLunghezzaC = new ArrayList<ArrayList<Parola>>();
144
145     //se ho già eseguito l'algoritmo di risoluzione ritorno il risultato salvato
146     if(algExecuted){
147         return algResult;
148     }else {
149         if (isComplete()) {
150             algExecuted=true;
151             algResult = true;
152             return true;
153         } else {
154             //inserisco all'interno della lista delle parole di lunghezza i le parole di lunghezza i
155             for (int i = 0; i < lunghezzaMax; i++) {
156                 ArrayList<Parola> paroleLunghezzaC;
157                 paroleLunghezzaC = schema_originale.ricercaLunghezzaParole(i + 1);
158                 if (paroleLunghezzaC.size() > 0) {
159                     listaParoleLunghezzaC.add(i, paroleLunghezzaC);
160                 } else {
161                     listaParoleLunghezzaC.add(new ArrayList<Parola>());
162                 }
163             }
164             //ciclo finché il cruciverba non è completo, fino ad aver fatto n>cicliMax iterazioni
165             while (!isComplete() && cicliEseguiti < cicliMax) {
166                 cicliEseguiti++;
167                 c = 0;
168                 while (c < lunghezzaMax) {
169                     //prendo le parole di lunghezza=c su cui lavorerò
170                     ArrayList<Parola> paroleLunghezzaC = listaParoleLunghezzaC.get(c);
171                     int i = 0;
172                     while (i < paroleLunghezzaC.size()) {
173                         Parola casellaDaCompletare = paroleLunghezzaC.get(i);
174                         //prima di cercare la parolaDaInserire faccio un controllo se la parola nella lista paroleLunghezzaC è già
175 completa,
176
177                         // se sì la elimino dalla lista
178                         if (casellaDaCompletare.getLunghezza() == casellaDaCompletare.getLettereInserite()) {
179                             paroleLunghezzaC.remove(casellaDaCompletare);
180                         } else {
181                             String parolaDaInserire = cercaParolaDaInserire(casellaDaCompletare, dizionario);
182                             if (!(parolaDaInserire.equals(""))) {
183 parole di
184
185                                 // lunghezza c ancora da inserire
186                                 paroleLunghezzaC.remove(casellaDaCompletare);
187
188                                 casellaDaCompletare.setParola(parolaDaInserire);
189                                 //aggiorno lo schema con la nuova parola trovata

```

```

189         aggiornaParola(casellaDaCompletare.getParola(), casellaDaCompletare.getPosizioneParola().getRiga()
190             , casellaDaCompletare.getPosizioneParola().getColonna(), casellaDaCompletare.getOrientamento());
191
192         //aggiorno il dizionario togliendo la parola che è stata inserita nello schema
193         // + eventuali parole che si sono autocompletate inserendo una parola nello schema
194         aggiornaDizionario();
195     } else {
196         //incremento solo se la parola non era già completata o se non è stato inserito niente nello schema
197         i++;
198     }
199 }
200 }
201     c++;
202 }
203 }
204
205 //setto la variabile di esecuzione algoritmo
206 algExecuted = true;
207 //controllo se il cruciverba è stato completato o meno e setto la variabile risultato
208 if (isComplete()) {
209     algResult = true;
210     return true;
211 } else {
212     algResult = false;
213     return false;
214 }
215 }
216 }
217 }
218 }
219 }

```

Classe ImplAlg4_AI:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import javax.swing.*;
6 import java.lang.reflect.Array;
7 import java.util.ArrayList;
8
9 public class ImplAlg4Cruciverba_AI extends ImplementazioneCruciverba{
10
11     //creo inner class SizeException che estende Exception, perché viene utilizzata solo all'interno di questa implementazione
12     public class SizeException extends Exception{
13         public SizeException(){
14             super();
15         }
16         public SizeException(String s){
17             super(s);
18         }
19     }
20
21     private CSP constraintsSolver;
22     private ArrayList<Parola> listSolution;
23
24     //costruttore cruciverba con una struttura passata in input che richiama il costruttore del padre e inizializza la variabile di tipo CPS
25     public ImplAlg4Cruciverba_AI(JPanel panel, char matrice[][], String parolaIniziale, int posizioneRigaIniziale, int
26     posizioneColonnaIniziale, ArrayList<String> dizionarioInput, char orientamento) {
27         super(panel,matrice,parolaIniziale,posizioneRigaIniziale,posizioneColonnaIniziale,dizionarioInput, orientamento);
28         constraintsSolver= new CSP(schema_originale,dizionario);
29     }
30
31     //corrisponde ad un ciclo di risolviCruciverba (in cui poi viene lanciata la funzione cercaParolaDaInserire)
32     //inserisce una parola nello schema del cruciverba
33     public String inserisci1Parola(){
34         int i=0;
35         Parola p=null;
36         //controllo se l'algoritmo di risoluzione era già stato eseguito
37         if (!(constraintsSolver.isCSPExecuted())){
38
39             //se non era stato eseguito lo eseguo e salvo il risultato nella variabile risultato di CPS, nella listSolution sarà contenuto
40             // l'ordine di inserimento delle parole nello schema
41             constraintsSolver.setCSPResult(backtrackSearch(constraintsSolver));
42
43             //se risultato dell'algoritmo è true prendo un elemento di listSolution e lo inserisco nello schema del cruciverba
44             if (constraintsSolver.isCSPResult()){
45                 if(listSolution.size()>0){
46                     p=listSolution.get(i);
47                     schema_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());
48                     listSolution.remove(i);
49                 }
50                 //aggiorno le parole disponibili nel dizionario dopo l'inserimento della nuova parola nel cruciverba
51                 aggiornaDizionario();
52             }
53         }else{
54             //in questo ramo non riesco l'algoritmo e se il risultato dell'algoritmo è true prendo un elemento di listSolution e lo inserisco
55             // nello schema del cruciverba
56             if(constraintsSolver.isCSPResult()) {
57                 if(listSolution.size()>0){
58                     p=listSolution.get(i);
59                     schema_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());
60                     listSolution.remove(i);
61                 }
62                 //aggiorno la lista delle parole disponibili nel dizionario dopo l'inserimento dell'ultima parola
63                 aggiornaDizionario();
64             }
65
66         }
67     }
68
69     algResult=constraintsSolver.isCSPResult();
70     try {
71         if (listSolution.size() == 0) {
72             if (p == null) {
73                 return null;
74             } else {
75                 return p.getParola();
76             }
77         } else {
78             return p.getParola();
79         }
80     }catch(NullPointerException e){
81         System.out.println("List solution è null");
82         return null;
83     }
84 }
85
86 //implementazione algoritmo 4 con AI utilizzo CSP
87 //CSP:
88 //variabili = paroleSchema
89 //domini = x domini ognuno relativo alle parole di x lettere
90 //vincoli = relativi alle variabili, ad esempio la variabile in posizione 2^ riga - 3^ colonna in orizzontale
91 // ha la lettera 'i' nella quarta casella
92 //backtracking cronologico = se non riesco a completare lo schema del cruciverba con l'attuale assegnamento faccio passi indietro
93 // e provo ad inserire un altro valore nella variabile a cui avevo assegnato un valore errato
```

```

94 public boolean risolviCruciverba(){
95     int i=0;
96     if (!(constraintsSolver.isCSPExecuted())){
97
98         constraintsSolver.setCSPResult(backtrackSearch(constraintsSolver));
99
100     if (constraintsSolver.isCSPResult()){
101         //inserisco tutte le parole nella listSolution all'interno dello schema del cruciverba, una alla volta
102         while(listSolution.size()>0){
103             Parola p=listSolution.get(i);
104             schema_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());
105             listSolution.remove(i);
106         }
107         //aggiorno la lista di parole disponibili del dizionario dopo l'inserimento delle parole di listSolution
108         aggiornaDizionario();
109     }
110     algResult=constraintsSolver.isCSPResult();
111     return algResult;
112 }else{
113     //se avevo già eseguito l'algoritmo inserisco solo le parole rimanenti nella listSolution all'interno dello schema del cruciverba
114     if(constraintsSolver.isCSPResult()) {
115         while (listSolution.size() > 0) {
116             Parola p=listSolution.get(i);
117             schema_originale.aggiornaSchema(p.getParola(),p.getPosizioneParola(),p.getOrientamento());
118             listSolution.remove(i);
119         }
120         //aggiorno la lista di parole disponibili dopo l'inserimento delle parole nel cruciverba
121         aggiornaDizionario();
122     }
123     algResult=constraintsSolver.isCSPResult();
124     return algResult;
125 }
126 }
127
128 }
129
130
131 //lancio procedura per soluzione cruciverba con AI
132 private boolean backtrackSearch( CSP csp){
133     //setto variabile di esecuzione algoritmo a true
134     csp.setCSPExecuted(true);
135     //salvo lo stato dello schema attuale per poterlo ripristinare dopo l'esecuzione dell'algoritmo
136     //che conterrà altrimenti tutti i riferimenti modificati durante l'esecuzione dell'algoritmo
137     Schema oldSchema=new Schema(schema_originale);
138     backtrack(new ArrayList<Parola>(), csp);
139     if (listSolution==null){
140         System.out.println("Soluzione non trovata");
141         return false;
142     }else {
143         if (listSolution.size() == constraintsSolver.getNumberVariables()) {
144             //ripristino lo schema originale e anche i valori a prima dell'esecuzione dell'algoritmo nelle caselle delle parole
145             schema_originale = oldSchema;
146             for (Parola p : schema_originale.getParoleSchema()) {
147                 p.aggiornaCaselleParola();
148             }
149             return true;
150         } else {
151             //ripristino lo schema originale e anche i valori a prima dell'esecuzione dell'algoritmo nelle caselle delle parole
152             schema_originale = oldSchema;
153             for (Parola p : schema_originale.getParoleSchema()) {
154                 p.aggiornaCaselleParola();
155             }
156             return false;
157         }
158     }
159 }
160
161 //ricerca soluzione cruciverba con AI con variable=MRV (minimum remaining values)+euristica del grado, value=prossimo valore del dominio
162 ancora
163 // non provato,
164 // inferenza con FC (forward checking), backtracking cronologico
165 // ritorno il valore quando sono arrivato in fondo alla procedura, altrimenti ritorno la variabile che volevo utilizzare per fare il
166 // backtracking intelligente sulle variabili collegate ad essa, adesso non è utile perché utilizzo il backtracking cronologico
167 private boolean backtrack(ArrayList<Parola> assignment, CSP csp){
168     int countAssignment=assignment.size();
169     boolean varResult=false;
170     if (assignment.size()==csp.getNumberVariables()){
171         listSolution=assignment;
172         return true;
173     }
174     Variable var=selectUnassignedVariable(constraintsSolver);
175     if (var==null){
176         //termino la procedura corrente
177         //nessuna variabile a cui assegnare un valore trovata
178         return false;
179     }else{
180         //mantengo una copia della vecchia variabile nel caso in cui l'assegnamento corrente non è corretto
181         Variable oldVar = new Variable(var);
182
183         //elimino dal dominio della variabile i possibili valori già assegnati a variabili precedenti (visto che
184         //quando ritorno da backtrack ripristino per la variabile il dominio iniziale
185         //var.updateDomain(assignment);
186
187         ArrayList<String> varDomain=orderDomainValues(var,assignment,csp);
188

```

```

189 //scorro i valori del dominio prendendoli uno ad uno dal dominio
190 for(String s : varDomain){
191     var.setNewParola(s);
192     var.aggiornaCaselleParola();
193     var.setValueAssigned(true);
194     assignment.add(var.getValue());
195     countAssignment++;
196
197     if (inference(csp,var,s,assignment)){
198         //chiamo di nuovo backtrack per trovare il prossimo assegnamento da fare
199         System.out.print(var.getValue().getPosizioneParola().getRiga());
200         System.out.print(var.getValue().getOrientamento());
201         System.out.print(var.getValue().getPosizioneParola().getColonna());
202         System.out.println(var.getValue().getParola());
203         varResult = backtrack(assignment,csp);
204         if (varResult){
205             return true;
206         }else{
207             //controllo se sono stati assegnati valori per il numero di variabili dentro lo schema del cruciverba, se si il
208             cruciverba è stato completato
209             //altrimenti ripristino i valori precedenti
210             if(assignment.size()==csp.getNumberVariables()){
211                 listSolution=assignment;
212                 return true;
213             }else {
214                 //operazioni di ripristino nel caso in cui l'inferenza non è andata a buon fine
215                 countAssignment--;
216                 assignment.remove(countAssignment);
217                 var.restore(oldVar, constraintsSolver.searchDomain(var.getValue().getLunghezza()));
218             }
219             restoreLinkedSameLengthVariables(csp, var, assignment);
220         }
221     }else{
222         //controllo se sono stati assegnati valori per il numero di variabili dentro lo schema del cruciverba, se si il cruciverba
223         è stato completato
224         //altrimenti ripristino i valori precedenti
225         if(assignment.size()==csp.getNumberVariables()){
226             listSolution=assignment;
227             return true;
228         }else {
229             //operazioni di ripristino nel caso in cui l'inferenza non è andata a buon fine
230             countAssignment--;
231             assignment.remove(countAssignment);
232             var.restore(oldVar, constraintsSolver.searchDomain(var.getValue().getLunghezza()));
233         }
234     }
235 }
236
237 }
238 }
239 return false;
240 }
241
242
243 //seleziono la variabile per la ricerca di un valore da inserire seguendo la strategia di CSP
244 private Variable selectUnassignedVariable(CSP constraintsSolver) {
245     //imposto il valore iniziale uguale al numero di parole da inserire nel cruciverba
246     int minValues = dizionario.size();
247     ArrayList<Variable> listCandidateVariables = null;
248
249     //procedura di selezione variabili per minor valore dei domini
250     for (Variable v : constraintsSolver.getVariables()) {
251         int variableValues = v.getValuesNumber();
252         if (!(v.isValueAssigned())) {
253             if (variableValues < minValues) {
254                 //se il numero di valori del sottodominio per questa variabile è inferiore a quella precedente creo una nuova lista
255                 // (la lista precedente contenente le variabili con numero valori dominio maggiore viene scartata)
256                 // in cui inserisco la variabile corrente e aggiorno il numero valore sottodominio minimo
257                 listCandidateVariables = new ArrayList<Variable>();
258                 listCandidateVariables.add(v);
259                 minValues = variableValues;
260             } else if (variableValues == minValues) {
261
262                 //controllo se non era ancora stata inizializzata la lista delle variabili candidate
263                 if (listCandidateVariables == null) {
264                     listCandidateVariables = new ArrayList<Variable>();
265                 }
266
267                 //inserisco la variabile corrente nella lista contenente le variabili con la stessa percentuale di completamento
268                 listCandidateVariables.add(v);
269             }
270         }
271     }
272
273     try{
274         if(listCandidateVariables.size()==0){
275             //se la lista contiene zero elementi sollevo un'eccezione. Inner class creata dentro questa classe perché
276             // la utilizzo solo al suo interno
277             throw new ImplAlg4Cruciverba_AI.SizeException("Non è stata trovata nessuna variabile candidata per l'inserimento di un nuovo
278             valore.");
279
280         }else if(listCandidateVariables.size()==1){
281             //se la lista delle variabili candidate contiene un solo elemento lo passo alla return della funzione
282             return listCandidateVariables.get(0);
283         }else{

```



```

284 //La lista contiene più elementi, cerco quella variabile che vincola maggiormente le altre variabili (quella con più lettere)
285 int maxLetters=0;
286 Variable maxLettersVariable=null;
287 for (Variable v : listCandidateVariables){
288     int currentLetters=v.getNumberLetters();
289     if(currentLetters>maxLetters){
290         maxLetters=currentLetters;
291         maxLettersVariable=v;
292     }
293 }
294 }
295 if(maxLettersVariable!=null){
296     return maxLettersVariable;
297 }else{
298     throw new NullPointerException("Non è stata trovata una variabile più vincolante dentro la lista delle variabili
299 candidate");
300 }
301 }
302 }catch (SizeException e){
303     System.out.println(e);
304     return null;
305 }catch (NullPointerException e){
306     System.out.println(e);
307     return null;
308 }
309 }
310 }
311 //creo la lista di assegnamenti di valori del dominio alla variabile, ordinata in ordine di inserimento nel dominio
312 private ArrayList<String> orderDomainValues(Variable var, ArrayList<Parola> assignment, CSP csp){
313     //TODO implementare soluzione per poter creare una lista ordinata di valori da quello meno vincolante a quello più vincolante
314     //per adesso ritorno semplicemente la lista dei valori del dominio possibili
315     return var.getListValuesDomain();
316 }
317 }
318 }
319 //procedura che mi permette di:
320 // 1) togliere dal problema csp la variabile var perché gli è stato assegnato un valore
321 // 2) cercare le variabili collegate a var di cui dovrò modificare il dominio dovuto alla nuova stringa s assegnata a var
322 // 3) fare un controllo che i domini risultanti delle variabili collegate non siano vuoti:
323 // se lo sono ritorno false altrimenti true
324 private boolean inference(CSP csp, Variable var, String s, ArrayList<Parola> assignment){
325
326     ArrayList<Variable> listLinkedVariables=null;
327     ArrayList<Variable> listSameLengthVariables=null;
328     int counterLinkedVariables=0;
329     int counterSameLengthVariables=0;
330     boolean result=true;
331
332     //trovare variabili collegate a var (cioè che condividono le stesse caselle) e ridurre il loro dominio
333     listLinkedVariables=csp.searchLinkedVariables(var);
334     while (listLinkedVariables!=null && counterLinkedVariables<listLinkedVariables.size() && result){
335         Variable currentVar=listLinkedVariables.get(counterLinkedVariables);
336         currentVar.setOldValue(currentVar.getValue());
337         if(currentVar.aggiornaParola()){
338             currentVar.restoreDomain(constraintsSolver.searchDomain(currentVar.getValue().getLunghezza()));
339             currentVar.updateDomain(assignment);
340         }
341         //procedura di inference sulla variabile corrente, se va a buon fine proseguo altrimenti ripristino i valori e imposto la
342         // variabile result a false
343         if (!(currentVar.inferenceAfterUpdateParola())){
344             currentVar.ripristinaParola();
345             result=false;
346         }
347         counterLinkedVariables++;
348     }
349
350     //guardo adesso tutte le variabili con la stessa lunghezza della variabile corrente a cui ho assegnato il valore e rimuovo il valore
351     // assegnato dal loro dominio (se presente)
352     listSameLengthVariables=csp.searchSameLengthVariables(var.getNumberLetters());
353     while(listSameLengthVariables!=null && counterSameLengthVariables<listSameLengthVariables.size() && result){
354         Variable currentVar=listSameLengthVariables.get(counterSameLengthVariables);
355         //currentVar.setOldValue(currentVar.getValue());
356         if (!(currentVar.inferenceAfterAssignedValue(s))){
357             result=false;
358         }
359         counterSameLengthVariables++;
360     }
361
362     //se il risultato dell'inferenza è false ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili
363     collegate
364     //partendo dalla penultima variabile perché l'ultima su cui si era verificato l'errore di inferenza li ha già ripristinati (torno
365     indietro di 2)
366     counterLinkedVariables=counterLinkedVariables-2;
367     while(!(result) && counterLinkedVariables>=0){
368         Variable currentVar=listLinkedVariables.get(counterLinkedVariables);
369         currentVar.ripristinaParola();
370         currentVar.restoreDomain(constraintsSolver.searchDomain(currentVar.getValue().getLunghezza()));
371         currentVar.updateDomain(assignment);
372         counterLinkedVariables--;
373     }
374     //se il risultato dell'inferenza è false ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili
375     con
376     // la stessa lunghezza
377     //partendo dalla penultima variabile perché l'ultima su cui si era verificato l'errore di inferenza li ha già ripristinati (torno
378

```



```

379 indietro di 2)
380 counterSameLengthVariables=counterSameLengthVariables-2;
381 while(! (result) && counterSameLengthVariables>=0){
382     Variable currentVar=listSameLengthVariables.get(counterSameLengthVariables);
383     //currentVar.ripristinaParola();
384     currentVar.restoreDomain(constraintsSolver.searchDomain(var.getValue().getLunghezza()));
385     currentVar.updateDomain(assignment);
386     counterSameLengthVariables--;
387 }
388 return result;
389 }
390
391 //ripristino i valori precedenti delle variabili collegate a var e delle variabili della stessa lunghezza di var
392 public void restoreLinkedSameLengthVariables(CSP csp, Variable var, ArrayList<Parola> assignment){
393     ArrayList<Variable> listLinkedVariables=null;
394     ArrayList<Variable> listSameLengthVariables=null;
395     int counterLinkedVariables=0;
396     int counterSameLengthVariables=0;
397
398     //trovare variabili collegate a var (cioè che condividono le stesse caselle)
399     listLinkedVariables=csp.searchLinkedVariables(var);
400
401     //guardo adesso tutte le variabili con la stessa lunghezza della variabile corrente a cui ho assegnato il valore
402     listSameLengthVariables=csp.searchSameLengthVariables(var.getNumberLetters());
403
404
405     //ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili collegate
406     while(counterLinkedVariables<listLinkedVariables.size()){
407         Variable currentVar=listLinkedVariables.get(counterLinkedVariables);
408         currentVar.aggiornaParola();
409         currentVar.restoreDomain(constraintsSolver.searchDomain(currentVar.getValue().getLunghezza()));
410         currentVar.updateDomain(assignment);
411         counterLinkedVariables++;
412     }
413
414     //ripristino tutti i valori dei domini che avevo modificato durante la procedura sulle variabili con
415     // la stessa lunghezza
416     while(counterSameLengthVariables<listSameLengthVariables.size()){
417         Variable currentVar=listSameLengthVariables.get(counterSameLengthVariables);
418         //currentVar.ripristinaParola();
419         currentVar.restoreDomain(constraintsSolver.searchDomain(var.getValue().getLunghezza()));
420         currentVar.updateDomain(assignment);
421         counterSameLengthVariables++;
422     }
423 }
424 }
425
426 }

```

Classe CSP:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import java.util.ArrayList;
6
7
8 //classe CSP = constraintsSatisfactionProblem, utilizzo di algoritmo di AI per la ricerca della soluzione del cruciverba
9 public class CSP {
10     private ArrayList<Parola> listSolution;
11     private boolean CSPExecuted;
12     private boolean CSPResult;
13     private ArrayList<Variable> variables;
14     private ArrayList<Domain> domains;
15
16     public ArrayList<Parola> getListSolution() {
17         return new ArrayList<Parola>(listSolution);
18     }
19
20     public ArrayList<Variable> getVariables() {
21         return new ArrayList<Variable>(variables);
22     }
23
24     public void setCSPExecuted(boolean CSPExecuted) {
25         this.CSPExecuted = CSPExecuted;
26     }
27
28     public boolean isCSPExecuted() {
29         return CSPExecuted;
30     }
31
32     public void setCSPResult(boolean CSPResult) {
33         this.CSPResult = CSPResult;
34     }
35
36     public boolean isCSPResult() {
37         return CSPResult;
38     }
39
40     public CSP(Schema s, ArrayList<String> d){
41         listSolution= new ArrayList<Parola>();
42         CSPExecuted=false;
43         CSPResult=false;
44         variables= new ArrayList<Variable>();
45         domains=new ArrayList<Domain>();
46         insertValuesInDomain(d);
47         insertValuesInVariables(s.getParoleSchema());
48     }
49
50     //inserisce tutte le stringhe all'interno del dominio con lettere corrispondenti
51     private void insertValuesInDomain(ArrayList<String> dizionario){
52         Domain foundD=null;
53         for (String s : dizionario){
54             //cerco dominio con elementi di lunghezza s.Length()
55             foundD=searchDomain(s.length());
56
57             if (foundD==null){
58                 //significa che non esisteva già un dominio con valori della lunghezza stringa e quindi creo un nuovo dominio,
59                 // ci inserisco la stringa e lo inserisco nella lista dei domini
60                 foundD=new Domain(s,s.length());
61                 domains.add(foundD);
62             }else{
63                 //significa che esisteva già un dominio con valori della lunghezza stringa e quindi inserisco in coda
64                 foundD.add(s);
65             }
66         }
67     }
68
69     public void insertValuesInVariables(ArrayList<Parola> list){
70         Domain foundD=null;
71         for (Parola p : list){
72             int i= 0;
73             if (!(p.isComplete())){ // se la parola non è già completa
74                 foundD=searchDomain(p.getLength());
75
76                 //se LettereInserite!=0 devo lanciare la procedura di inferenza sui domini delle variabili che creo
77                 if (foundD!=null){
78                     Variable v = new Variable(p,foundD);
79                     variables.add(v);
80                     //se LettereInserite per questa parola è diverso da zero lancio anche la procedura di inferenza per ridurre
81                     // i domini di questa variabile
82                     if (p.getLettereInserite()>0){
83                         v.inferenceAfterUpdateParola();
84                     }
85                 }else{
86                     throw new NullPointerException("Non è stato trovato un dominio per questa variabile");
87                 }
88             }
89         }
90     }
91 }
92
93 // cerco il dominio contenente elementi di lunghezza L
```

```

94 public Domain searchDomain(int l){
95     Domain foundD=null;
96     int i=0;
97     while (i<domains.size() && foundD==null){           //esco dal while se ho scorso tutto l'array dei domini o se ho trovato un elemento
98         Domain checkD=domains.get(i);
99         if(checkD.getLunghezzaParole()==1){
100             foundD=checkD;
101         }
102         i++;
103     }
104     return foundD;
105 }
106
107 //ritorna Le variabili collegata alla variabile passata in input var
108 public ArrayList<Variable> searchLinkedVariables(Variable var){
109     ArrayList<Variable> linkedVariables=null;
110
111     //scorro Le variabili dello schema alla ricerca di quelle collegata a quella in input
112     for(Variable searchVar : variables){
113         //faccio un controllo se è già assegnato un valore, in questo modo evito di verificare il collegamento con una variabile
114         // alla quale ho già assegnato un valore (sia variabile corrente che altre variabili all'interno dello schema)
115         if (!(searchVar.isValueAssigned())) {
116             if (searchVar.isLinked(var)) {
117                 if (linkedVariables==null){
118                     linkedVariables=new ArrayList<Variable>();
119                 }
120                 linkedVariables.add(searchVar);
121             }
122         }
123     }
124     return linkedVariables;
125 }
126
127 //ritorna Le variabili con la stessa lunghezza della variabile passata in input var
128 public ArrayList<Variable> searchSameLengthVariables(int lengthToCompare){
129     ArrayList<Variable> sameLengthVariables=null;
130
131     //scorro Le variabili dello schema alla ricerca di quelle con la stessa lunghezza di quella in input
132     for(Variable searchVar : variables){
133         //faccio un controllo se è già assegnato un valore, in questo modo evito di verificare il collegamento con una variabile
134         // alla quale ho già assegnato un valore (sia variabile corrente che altre variabili all'interno dello schema)
135         if (!(searchVar.isValueAssigned())) {
136             if (searchVar.getNumberLetters()==lengthToCompare) {
137                 if (sameLengthVariables==null){
138                     sameLengthVariables=new ArrayList<Variable>();
139                 }
140                 sameLengthVariables.add(searchVar);
141             }
142         }
143     }
144     return sameLengthVariables;
145 }
146
147 //ritorno il numero di variabili totali
148 public int getNumberVariables(){
149     return variables.size();
150 }
151
152 }

```

Classe Variable:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import java.util.ArrayList;
6
7 public class Variable {
8     private Parola value;
9     private Domain variableDomain;
10    private boolean valueAssigned;
11    private ArrayList<String> oldListValues;
12    private Parola oldValue;
13
14    //costruttore in cui inizializzo la parola e il dominio della variabile + altre variabili per lo stato dell'oggetto
15    public Variable(Parola var, Domain d){
16        value=new Parola(var);
17        variableDomain=new Domain(d);
18        valueAssigned=false;
19        oldListValues=null;
20        oldValue=null;
21    }
22
23    //costruisco una variabile da una già esistente
24    public Variable(Variable var){
25        this.value=var.getValue();
26        this.variableDomain=var.getVariableDomain();
27        this.valueAssigned=var.isValueAssigned();
28        this.oldListValues=var.getOldListValues();
29        this.oldValue=var.getOldValue();
30    }
31
32    public Domain getVariableDomain(){ return new Domain(this.variableDomain); }
33
34    public Parola getValue(){ return new Parola(this.value); }
35
36    public boolean isValueAssigned() {
37        return valueAssigned;
38    }
39
40    public void setValueAssigned(boolean valueAssigned) {
41        this.valueAssigned = valueAssigned;
42    }
43
44    public ArrayList<String> getOldListValues() {
45        if (oldListValues==null){
46            return null;
47        }else{
48            return new ArrayList<String>(oldListValues);
49        }
50    }
51
52    public Parola getOldValue() {
53        if (oldValue==null){
54            return null;
55        }else {
56            return new Parola(oldValue);
57        }
58    }
59
60    public void setOldValue(Parola oldValue) {
61        this.oldValue = oldValue;
62    }
63
64    //procedura di inferenza sul dominio di questa variabile per rimuovere i valori non permessi dopo assegnazione di un valore ad un'altra
65    //variabile
66    //ritorna false se il dominio risultante dall'inferenza risulta essere vuoto
67    public boolean inferenceAfterAssignedValue(String s){
68        ArrayList<String> listValuesDomain=variableDomain.getListValues();
69
70        //salvo una copia della lista valore del dominio per poterla ripristinare in caso di errore di inferenza (dominio vuoto)
71        oldListValues=new ArrayList<String>(listValuesDomain);
72        if (listValuesDomain.contains(s)){
73            listValuesDomain.remove(s);
74        }
75
76        //se il dominio dopo aver fatto inferenza non è vuoto lo aggiorniamo per la variabile corrente e ritorno true
77        //altrimenti ritorno false senza aggiornare il dominio della variabile corrente
78        if(listValuesDomain.size()>0){
79            variableDomain.setListValues(listValuesDomain);
80            return true;
81        }else{
82            return false;
83        }
84    }
85
86    //procedura di inferenza sul dominio di questa variabile per rimuovere i valori non permessi dopo aggiornamento parola
87    //ritorna false se il dominio risultante dall'inferenza risulta essere vuoto
88    public boolean inferenceAfterUpdateParola(){
89        ArrayList<String> listValuesDomain = variableDomain.getListValues();
90
91        //salvo una copia della lista valore del dominio per poterla ripristinare in caso di errore di inferenza (dominio vuoto)
92        if (listValuesDomain!=null) {
93            oldListValues = new ArrayList<String>(listValuesDomain);
```

```

94     }
95     String parolaSchema = value.getParola();
96     for (int i=0; i<parolaSchema.length(); i++){
97         int j=0;
98         char checkChar=parolaSchema.charAt(i);
99
100         //faccio la procedura di controllo carattere solo se è diverso dal carattere '.' che è quello preimpostato
101         // nelle caselle dello schema
102         if(checkChar!='.'){
103             while(j<listValuesDomain.size()){
104                 String s = listValuesDomain.get(j);
105
106                 //se il carattere della parola è diverso da quello del valore del dominio, tolgo quel valore dal dominio di questa
107 variabile
108                 //altrimenti incremento il valore di j e continuo sul valore successivo del dominio
109                 if(checkChar!=s.charAt(i)){
110                     listValuesDomain.remove(j);
111                 }else {
112                     j++;
113                 }
114             }
115         }
116     }
117 }
118
119 //se il dominio dopo aver fatto inferenza non è vuoto lo aggiorno per la variabile corrente e ritorno true
120 //altrimenti ritorno false senza aggiornare il dominio della variabile corrente
121 if(listValuesDomain.size()>0){
122     variableDomain.setListValues(listValuesDomain);
123     return true;
124 }else{
125     return false;
126 }
127 }
128 }
129
130 //ritorno il numero di lettere della variabile
131 public int getNumberLetters(){
132     return value.getLunghezza();
133 }
134
135 //ritorna il numero di valori dentro il dominio di questa variabile
136 public int getValuesNumber(){
137     return variableDomain.getValuesNumber();
138 }
139
140 public String getValueDomain(int index){
141     return variableDomain.getValueDomain(index);
142 }
143
144 public ArrayList<String> getListValuesDomain(){
145     return variableDomain.getListValues();
146 }
147
148 //setto value con una nuova parola
149 public void setNewParola(String s){
150     value.setParola(s);
151 }
152
153 public void aggiornaCaselleParola(){
154     value.aggiornaCaselleParola();
155 }
156
157 public boolean aggiornaParola(){
158     return value.aggiornaParola();
159 }
160
161 //confronto le caselle della Parola relativa a questo oggetto con quella della variabile passata in input, se una sola
162 // casella coincide allora le due variabili sono collegate e ritorno true
163 public boolean isLinked(Variable var){
164     return value.isLinked(var.getValue());
165 }
166
167 //ripristino i valori precedenti del dominio
168 public void restoreDomain(){
169     variableDomain.setListValues(oldListValues);
170 }
171
172 //ripristino i valori predefiniti del dominio (quelli iniziali)
173 public void restoreDomain(Domain initialDomain){
174     if (!(initialDomain==null)){
175         variableDomain.setListValues(initialDomain.getListValues());
176     }
177 }
178
179 //ripristino la parola precedente della variabile (prima dell'inferenza)
180 public void ripristinaParola(){
181     value=oldValue;
182 }
183
184 //ripristino i valori della variabile corrente con quelli della variabile passata in input
185 public void restore(Variable oldVar, Domain initialDomain){
186     if (oldVar!=null) {
187         this.value.setParola(oldVar.getValue().getParola());
188         this.value.aggiornaCaselleParola();

```

```

189         this.variableDomain.setListValues(initialDomain.getListValues());
190         this.valueAssigned = oldVar.isValueAssigned();
191         this.oldListValues = oldVar.getOldListValues();
192         Parola restoreOldValue = oldVar.getOldValue();
193         if (!(restoreOldValue == null)) {
194             this.oldValue.setParola(restoreOldValue.getParola());
195         }
196     }
197 }
198 }
199
200 public void updateDomain(ArrayList<Parola> assignment){
201     for(int i=0; i<assignment.size(); i++){
202         String parolaAssignment = assignment.get(i).getParola();
203         if(this.getNumberLetters()==parolaAssignment.length()) {
204             if (this.variableDomain.getListValues().contains(parolaAssignment)) {
205                 ArrayList<String> newListValuesDomain=this.getListValuesDomain();
206                 newListValuesDomain.remove(parolaAssignment);
207                 this.variableDomain.setListValues(newListValuesDomain);
208             }
209         }
210     }
211 }
212 }
213 }
214 }

```

Classe Domain:

```
1 package com.cruciverbapackage;
2
3 //Coffaro_Davide_mat556603_Progetto ESP cruciverba
4
5 import java.util.ArrayList;
6
7 //classe dei domini contenente le parole del dizionario suddivise per lunghezza parole
8 class Domain{
9     private ArrayList<String> listValues;
10    private int length;
11
12    //costruttore con inizializzazione delle variabili dell'oggetto Domain e con inserimento della stringa nella lista di valori del dominio
13    public Domain(String s, int dim){
14        listValues=new ArrayList<String>();
15        listValues.add(s);
16        length=dim;
17    }
18
19    //costruttore a partire da un altro oggetto di tipo dominio già esistente
20    public Domain(Domain d){
21        listValues=d.getListValues();
22        length=d.getLunghezzaParole();
23    }
24
25    //ritorna la lista di valori inseriti in questo dominio
26    public ArrayList<String> getListValues(){
27        return new ArrayList<String>(listValues);
28    }
29
30    //modifica la lista di valori inseriti in questo dominio
31    public void setListValues(ArrayList<String> listValues) {
32        this.listValues = listValues;
33    }
34
35    //ritorna la lunghezza delle parole inserite in questo dominio
36    public int getLunghezzaParole(){
37        return length;
38    }
39
40    //aggiunge una nuova parola nella lista di valori in questo dominio
41    public void add (String s){
42        listValues.add(s);
43    }
44
45    //conta il numero di valori all'interno del dominio
46    public int getValuesNumber(){
47        return listValues.size();
48    }
49
50    //restituisco la stringa alla posizione index della lista di valori (listValues)
51    public String getValueDomain(int index){
52        return listValues.get(index);
53    }
54
55 }
```


RIO
GARRANI
WO
LISI
GARA
MARGARINE
INORGANICA
ALATO
FERRAMENTA
CRANICO
RA
RAVERA
TOCCARE
OCARINA
BOB
LIONE
ODORE
ADAMI
SCAPPATOIA
ALIBI
LORENA
MASSA
SAVONAROLA
IRENE
WESER
BOSTON
BE

Esempio3

12 19
..*.....*.....
.....*.....*
...*.*.....*.*.....
.*.*.*.*.*.*.....
.....*.....
.....*.....
.....*.....*
.....*.....*
..*.*.....*.....
.....*.....*
.*.....*.....
.*.....*.....
..........*

DONO 4 4 V
BANALITA
MIAO
TETTO
ISOLA
LUCI
IDA
TONANTE
EO
NOMINE
NAT
SPIEGAMENTO
TA
OTRI
ADATTE
EOLO
AGARICO
IDIOMA
PECORE
GIARDINIPUBBLICI
ESIGUI
AI
ELI
MANICOTTO
ITER
ARRINGARE
TI
MARINOMARINI
FIDEL
INEDITI
TOM
NI
BARATRO
LARIO
ULANI
ATEO
ON
CALLIMACO
NEO
ISACCO

TINI
RINASCIMENTO
LB
ADRIA
SL
NC
PIRAMIDE
ALI
ADA
ILIO
AU
ECONOMICA
ICE
SUD
AREA
RAME
MB
CALAF
BUE
EMI
ASTRONOMICA
CRUMIRA
OZI
RINOMATA
ANOMALIA
IDIOTI
LOS
SS
URNE
ORTI
RA
PUBBLICAZIONI
SE

Bibliografia:

- Artificial Intelligence, A Moderne Approach Third Edition, Cap.6 Constraints Satisfaction Problems
- IntelliJ Idea Online Documentation about JPanel, JLabel, JButton, JList, JScrollPane, JTextField:
 - 1) <https://www.jetbrains.com/help/idea/designing-gui-major-steps.html>
 - 2) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/360003406579-Drawing-on-a-JPanel-of-a-form>
 - 3) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206233659-GUI-Designer-Question-How-to-dynamically-set-JLabel-Text>
 - 4) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206327529-Default-JButton-for-a-form>
 - 5) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/360000366919-Add-JScrollPane-to-GUI-Form>
 - 6) <https://intellij-support.jetbrains.com/hc/en-us/community/posts/206926855-JTextField-in-UI-Designer>
- StackOverFlow for JList <https://stackoverflow.com/questions/30009226/add-item-to-jlist-in-intellij-idea>
- Introduction to Algorithms 3rd edition Cormen Leiserson Rivest Stein Cap.4 Divide and Conquer
- https://github.com/coffa94/ESP_CruciverbaIntelliJ
- Lista di parole italiane <https://github.com/napolux/paroleitaliane/tree/master/paroleitaliane>