

```

1 /**
2 * Classe principale del prato fiorito, questa è la classe da cui
3 * parte il programma.
4 * Nella classe Main vengono chiesti tutti gli input che devono
5 * essere inseriti dall'utente, e viene fatto anche un controllo su eventuali
6 * valori non validi inseriti da parte dell'utente
7 *
8 * @author Davide Branchi
9 */
10
11 import java.time.LocalDateTime;
12 import java.time.format.DateTimeFormatter;
13 import java.util.Scanner;
14 public class Main {
15     public static void main(String[] args) {
16         SoundPlayer sound = new SoundPlayer("Assets/game.wav");
17         sound.play();
18         Scanner in = new Scanner(System.in);
19         String output = "";
20         String currentOutput;
21         System.out.println(currentOutput = Colors.cyan("PRATO FIORITO GAME"));
22         output += currentOutput + "\n";
23         System.out.print(currentOutput = Colors.yellow("Enter your name: "));
24         output += currentOutput + "\n";
25         String name;
26         do{
27             name = in.nextLine();
28             if(name.equals("")){
29                 System.out.println(currentOutput = Colors.redBg("Inserire un nome!"));
30                 output += currentOutput + "\n";
31                 System.out.print(currentOutput = Colors.yellow("Enter your name: "));
32                 output += currentOutput + "\n";
33             }
34         }while(name.equals(""));
35         output += name + "\n";
36         System.out.print(currentOutput = Colors.yellow("Enter dimension (2-50): "));
37         output += currentOutput + "\n";
38         int dimension;
39         try{
40             dimension = Integer.parseInt(in.nextLine());
41             output += dimension + "\n";
42         }catch(NumberFormatException e){
43             dimension = 10;
44             System.out.println(currentOutput = Colors.redBg("Valore non valido! Inserito dimensione 10 (default)"));
45             output += currentOutput + "\n";
46         }
47         if(dimension < 2 || dimension > 50){
48             System.out.println(currentOutput = Colors.redBg("Valore non valido! Inserito dimensione 10 (default)"));
49             output += currentOutput + "\n";
50         }
51         PratoFiorito pratoFiorito = new PratoFiorito(dimension);
52         Player player = new Player(name);
53         System.out.println(currentOutput = Colors.yellow("Game started"));
54         output += currentOutput + "\n";
55         pratoFiorito.inizializzaPrato();
56         pratoFiorito.printPrato();
57         output += pratoFiorito.getPrato();
58         boolean match = true;
59         int moves = 0;
60         while(match){
61             boolean validCell = false;
62             int row = 0;
63             int colum = 0;
64             while(!validCell){
65                 boolean validRow = false;
66                 while(!validRow){
67                     System.out.println(currentOutput = Colors.yellow("Enter row:"));
68                     output += currentOutput + "\n";
69                     try{
70                         row = Integer.parseInt(in.nextLine().trim());
71                         output += row + "\n";
72                         if(row > 0 && row <= pratoFiorito.getDimension()){
73                             validRow = true;
74                         }else{
75                             System.out.println(currentOutput = Colors.redBg("Valore inserito non valido, riprovare"));
76                             output += currentOutput + "\n";
77                         }
78                     }catch(NumberFormatException e){
79                         System.out.println(currentOutput = Colors.redBg("Valore inserito non valido, riprovare"));
80                         output += currentOutput + "\n";
81                     }
82                 }
83             }
84         }
85     }
86 }

```

```

83     boolean validColum = false;
84     while(!validColum){
85         try{
86             System.out.println(currentOutput = Colors.yellow("Enter Colum:"));
87             output += currentOutput + "\n";
88             colum = Integer.parseInt(in.nextLine().trim());
89             output += colum + "\n";
90             if(colum > 0 && colum <= pratoFiorito.getDimension()){
91                 validColum = true;
92             }else{
93                 System.out.println(currentOutput = Colors.redBg("Valore inserito non valido, riprovare"));
94                 output += currentOutput + "\n";
95             }
96         }catch(NumberFormatException e){
97             System.out.println(currentOutput = Colors.redBg("Valore inserito non valido, riprovare"));
98             output += currentOutput + "\n";
99         }
100    }
101    if(pratoFiorito.isAValidCell(row,colum)){
102        validCell = true;
103    }else{
104        System.out.println(currentOutput = Colors.redBg("Cella selezionata già scoperta"));
105        output += currentOutput + "\n";
106    }
107 }
108 if(pratoFiorito.isFlowerCell(row,colum)){
109     sound.stop();
110     sound.setSoundPath("Assets/gameover.wav");
111     sound.play();
112     match = false;
113     pratoFiorito.printPrato();
114     output += pratoFiorito.getPrato();
115     System.out.println(currentOutput = Colors.red("GAME OVER"));
116     output += currentOutput + "\n";
117     System.out.println(currentOutput = Colors.cyan("Player: " + player.getName()));
118     output += currentOutput + "\n";
119     System.out.println(currentOutput = Colors.cyan("Score: " + player.getScore()));
120     output += currentOutput + "\n";
121     System.out.print(currentOutput = Colors.yellow("Visualizzare la posizione di tutti i fiori? (S/N): "));
122     output += currentOutput + "\n";
123     String choice = in.nextLine().toLowerCase().trim();
124     output += choice + "\n";
125     if(choice.equals("s")){
126         pratoFiorito.printFiori();
127         output += pratoFiorito.getFiori();
128     }
129 }else{
130     pratoFiorito.mineCell(row, colum);
131     player.addScore();
132     pratoFiorito.printPrato();
133     output += pratoFiorito.getPrato();
134     moves++;
135 }
136 if(moves == (Math.pow(pratoFiorito.getDimension(), 2) - pratoFiorito.getDimension())){
137     sound.stop();
138     sound.setSoundPath("Assets/gamewin.wav");
139     sound.play();
140     match = false;
141     System.out.println(currentOutput = Colors.yellowBg("YOU WON!"));
142     output += currentOutput + "\n";
143     System.out.println(currentOutput = Colors.cyan("Player: " + player.getName()));
144     output += currentOutput + "\n";
145     System.out.println(currentOutput = Colors.cyan("Score: " + player.getScore()));
146     output += currentOutput + "\n";
147 }
148 }
149 System.out.print(currentOutput = Colors.yellow("Esportare la partita in un file pdf? (S/N): "));
150 output += currentOutput + "\n";
151 String choice = in.nextLine().toLowerCase().trim();
152 output += choice + "\n";
153 if(choice.equals("s")){
154     DateTimeFormatter dft = DateTimeFormatter.ofPattern("dd-MM-yyyy_HH.mm.ss");
155     LocalDateTime now = LocalDateTime.now();
156     String pdfName = "prato_fiorito_" + dft.format(now) + ".pdf";
157     ExportGame export = new ExportGame(pdfName, output);
158     try{
159         export.startExport();
160         System.out.println(Colors.green("Partita esportata nel file " + pdfName));
161     }catch (Exception e){
162         System.out.println(Colors.redBg("Errore nell'esportazione della partita in un file pdf"));
163         System.out.println(Colors.redBg(e.getMessage()));
164     }
}

```

165 }
166 }
167 }
168

```

1 /**
2 * Classe Colors utile per stampare dei testi colorati
3 * su console che interpretano gli ANSI escape codes.
4 *
5 * @author Davide Branchi
6 */
7 public class Colors {
8     //Costanti contenenti gli escape codes per ogni colore
9     public static final String RESET = "\u001B[0m";
10    public static final String BLACK = "\u001B[30m";
11    public static final String RED = "\u001B[31m";
12    public static final String GREEN = "\u001B[32m";
13    public static final String YELLOW = "\u001B[33m";
14    public static final String BLUE = "\u001B[34m";
15    public static final String PURPLE = "\u001B[35m";
16    public static final String CYAN = "\u001B[36m";
17    public static final String WHITE = "\u001B[37m";
18
19    public static final String BLACK_BG = "\u001B[40m";
20    public static final String RED_BG = "\u001B[41m";
21    public static final String GREEN_BG = "\u001B[42m";
22    public static final String YELLOW_BG = "\u001B[43m";
23    public static final String BLUE_BG = "\u001B[44m";
24    public static final String PURPLE_BG = "\u001B[45m";
25    public static final String CYAN_BG = "\u001B[46m";
26    public static final String WHITE_BG = "\u001B[47m";
27
28 /**
29 * Metodo che restituisce uns stringa con ANSI escape code
30 * per il colore nero
31 *
32 * @param text testo da colorare
33 * @return stringa con il testo nero
34 */
35 public static String black(String text) {
36     return BLACK + text + RESET;
37 }
38
39 /**
40 * Metodo che restituisce uns stringa con ANSI escape code
41 * per il colore rosso
42 *
43 * @param text testo da colorare
44 * @return stringa con il testo rosso
45 */
46 public static String red(String text) {
47     return RED + text + RESET;
48 }
49
50 /**
51 * Metodo che restituisce uns stringa con ANSI escape code
52 * per il colore verde
53 *
54 * @param text testo da colorare
55 * @return stringa con il testo verde
56 */
57 public static String green(String text) {
58     return GREEN + text + RESET;
59 }
60
61 /**
62 * Metodo che restituisce uns stringa con ANSI escape code
63 * per il colore giallo
64 *
65 * @param text testo da colorare
66 * @return stringa con il testo giallo
67 */
68 public static String yellow(String text) {
69     return YELLOW + text + RESET;
70 }
71
72 /**
73 * Metodo che restituisce uns stringa con ANSI escape code
74 * per il colore blu
75 *
76 * @param text testo da colorare
77 * @return stringa con il testo blu
78 */
79 public static String blue(String text) {
80     return BLUE + text + RESET;
81 }
82

```

```

83  /**
84   * Metodo che restituisce una stringa con ANSI escape code
85   * per il colore viola
86   *
87   * @param text testo da colorare
88   * @return stringa con il testo viola
89   */
90  public static String purple(String text) {
91      return PURPLE + text + RESET;
92  }
93
94 /**
95  * Metodo che restituisce una stringa con ANSI escape code
96  * per il colore ciano
97  *
98  * @param text testo da colorare
99  * @return stringa con il testo ciano
100 */
101 public static String cyan(String text) {
102     return CYAN + text + RESET;
103 }
104
105 /**
106 * Metodo che restituisce una stringa con ANSI escape code
107 * per il colore bianco
108 *
109 * @param text testo da colorare
110 * @return stringa con il testo bianco
111 */
112 public static String white(String text) {
113     return WHITE + text + RESET;
114 }
115
116 /**
117 * Metodo che restituisce una stringa con ANSI escape code
118 * per lo sfondo nero di un testo
119 *
120 * @param text testo da colorare
121 * @return stringa con lo sfondo nero
122 */
123 public static String blackBg(String text) {
124     return BLACK_BG + text + RESET;
125 }
126
127 /**
128 * Metodo che restituisce una stringa con ANSI escape code
129 * per lo sfondo rosso di un testo
130 *
131 * @param text testo da colorare
132 * @return stringa con lo sfondo rosso
133 */
134 public static String redBg(String text) {
135     return RED_BG + text + RESET;
136 }
137
138 /**
139 * Metodo che restituisce una stringa con ANSI escape code
140 * per lo sfondo verde di un testo
141 *
142 * @param text testo da colorare
143 * @return stringa con lo sfondo verde
144 */
145 public static String greenBg(String text) {
146     return GREEN_BG + text + RESET;
147 }
148
149 /**
150 * Metodo che restituisce una stringa con ANSI escape code
151 * per lo sfondo giallo di un testo
152 *
153 * @param text testo da colorare
154 * @return stringa con lo sfondo giallo
155 */
156 public static String yellowBg(String text) {
157     return YELLOW_BG + text + RESET;
158 }
159
160 /**
161 * Metodo che restituisce una stringa con ANSI escape code
162 * per lo sfondo blu di un testo
163 *
164 * @param text testo da colorare

```

```

165     * @return stringa con lo sfondo blu
166     */
167     public static String blueBg(String text) {
168         return BLUE_BG + text + RESET;
169     }
170
171     /**
172      * Metodo che restituisce una stringa con ANSI escape code
173      * per lo sfondo viola di un testo
174      *
175      * @param text testo da colorare
176      * @return stringa con lo sfondo viola
177      */
178     public static String purpleBg(String text) {
179         return PURPLE_BG + text + RESET;
180     }
181
182     /**
183      * Metodo che restituisce una stringa con ANSI escape code
184      * per lo sfondo ciano di un testo
185      *
186      * @param text testo da colorare
187      * @return stringa con lo sfondo ciano
188      */
189     public static String cyanBg(String text) {
190         return CYAN_BG + text + RESET;
191     }
192
193     /**
194      * Metodo che restituisce una stringa con ANSI escape code
195      * per lo sfondo bianco di un testo
196      *
197      * @param text testo da colorare
198      * @return stringa con lo sfondo bianco
199      */
200     public static String whiteBg(String text) {
201         return WHITE_BG + text + RESET;
202     }
203 }
```

```

1 /**
2 * Classe Player che viene utilizzata per memorizzare il nome
3 * del giocatore e gestire il suo punteggio
4 *
5 * @author Davide Branchi
6 */
7 public class Player {
8     private String name;
9     private int score;
10
11    /**
12     * Costruttore Player
13     * @param name nome del giocatore
14     */
15    public Player(String name){
16        this.name = name;
17        this.score = 0;
18    }
19
20    /**
21     * Metodo getter che ritorna il nome del giocatore
22     * @return nome del giocatore
23     */
24    public String getName() {
25        return name;
26    }
27
28    /**
29     * Metodo setter che imposta il nome del giocatore
30     * @param name nome del giocatore
31     */
32    private void setName(String name) {
33        this.name = name;
34    }
35
36    /**
37     * Metodo getter che ritorna lo score del giocatore
38     * @return score del giocatore
39     */
40    public int getScore() {
41        return score;
42    }
43
44    /**
45     * Metodo setter che imposta lo score del giocatore
46     * @param score score del giocatore
47     */
48    private void setScore(int score) {
49        this.score = score;
50    }
51
52    /**
53     * Metodo che incrementa lo score del giocatore di 1
54     */
55    public void addScore(){
56        this.score++;
57    }
58 }
59

```

```

1 /**
2 * Classe che permette di esportare un testo in un pdf
3 * la classe utilizza una libreria esterna (itextpdf)
4 * che è contenuta in un file jar dentro al package com.itextpdf
5 *
6 * @author Davide Branchi
7 */
8
9 import com.itextpdf.text.BaseColor;
10 import com.itextpdf.text.Chunk;
11 import com.itextpdf.text.Document;
12 import com.itextpdf.text.DocumentException;
13 import com.itextpdf.text.Font;
14 import com.itextpdf.text.Font.FontFamily;
15 import com.itextpdf.text.Paragraph;
16 import com.itextpdf.text.pdf.PdfWriter;
17
18 import java.io.FileNotFoundException;
19 import java.io.FileOutputStream;
20 import java.util.Scanner;
21
22 public class ExportGame {
23     private String path;
24     private String pdfContent;
25     Font red = new Font(FontFamily.COURIER, 11, Font.BOLD, BaseColor.RED);
26     Font black = new Font(FontFamily.COURIER, 11, Font.BOLD, BaseColor.BLACK);
27     Font cyan = new Font(FontFamily.COURIER, 11, Font.BOLD, BaseColor.BLUE);
28     Font yellow = new Font(FontFamily.COURIER, 11, Font.BOLD, BaseColor.ORANGE);
29     Font green = new Font(FontFamily.COURIER, 11, Font.BOLD, BaseColor.GREEN);
30     Font white = new Font(FontFamily.COURIER, 11, Font.BOLD, BaseColor.WHITE);
31
32 /**
33 * Metodo costruttore della classe che permette di istanziare un oggetto
34 * ExportGame a cui deve essere passato come parametro il percorso del pdf
35 * da creare e il contenuto di esso
36 * @param path percorso del pdf
37 * @param pdfContent contenuto del pdf
38 */
39 public ExportGame(String path, String pdfContent) {
40     this.path = path;
41     this.pdfContent = pdfContent;
42 }
43
44 /**
45 * Metodo che esporta il contenuto in un pdf interpretando anche gli
46 * ANSI escape codes per scrivere il testo colorato
47 * @throws FileNotFoundException
48 * @throws DocumentException
49 */
50 public void startExport() throws FileNotFoundException, DocumentException {
51     Scanner content = new Scanner(pdfContent);
52     Document document = new Document();
53     PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(path));
54     document.open();
55     while (content.hasNextLine()) {
56         String line = content.nextLine();
57         Paragraph p = new Paragraph();
58         String ansiColor = "";
59         for (int i = 0; i < line.length(); i++) {
60             if (line.charAt(i) == '\u001B') { //Trovato escape code
61                 ansiColor = "";
62                 i++;
63                 while (line.charAt(i) != 'm') {
64                     ansiColor += line.charAt(i);
65                     i++;
66                 }
67                 i++;
68             }
69             if (i < line.length()) {
70                 if (line.charAt(i) != '\u001B') {
71                     switch (ansiColor) { //Switch per colorare il testo in base all'escape code trovato
72                         case "[31":
73                             Chunk redText = new Chunk(line.charAt(i), red);
74                             p.add(redText);
75                             break;
76                         case "[32":
77                             Chunk greenText = new Chunk(line.charAt(i), green);
78                             p.add(greenText);
79                             break;
80                         case "[33":
81                             Chunk yellowText = new Chunk(line.charAt(i), yellow);
82                             p.add(yellowText);

```

```
83         break;
84     case "[36":
85         Chunk cyanText = new Chunk(line.charAt(i), cyan);
86         p.add(cyanText);
87         break;
88     case "[41":
89         Chunk redBgText = new Chunk(line.charAt(i), white);
90         redBgText.setBackground(BaseColor.RED);
91         p.add(redBgText);
92         break;
93     case "[43":
94         Chunk yellowBgText = new Chunk(line.charAt(i), white);
95         yellowBgText.setBackground(BaseColor.ORANGE);
96         p.add(yellowBgText);
97         break;
98     default:
99         Chunk blackText = new Chunk(line.charAt(i), black);
100        p.add(blackText);
101        break;
102    }
103    } else {
104        i--;
105    }
106}
107p.setAlignment(1);
108document.add(p);
109}
110}
111document.close();
112}
113}
114}
```

```

1 /**
2 * Classe SoundPlayer che permette di riprodurre dei suoni
3 *
4 * @author Davide Branchi
5 */
6
7 import javax.sound.sampled.AudioInputStream;
8 import javax.sound.sampled.AudioSystem;
9 import javax.sound.sampled.Clip;
10 import java.io.File;
11
12 public class SoundPlayer {
13     private static String soundPath;
14     private Clip clip;
15
16     /**
17      * Metodo costruttore che permette di istanziare un oggetto
18      * SoundPlayer per far riprodurre dei suoni
19      * @param soundPath percorso dove si trova il file audio (solo file .wav)
20      */
21     public SoundPlayer(String soundPath) {
22         this.soundPath = soundPath;
23         try{
24             clip = AudioSystem.getClip();
25         }catch(Exception e){
26             System.out.println(Colors.redBg("Errore nel caricamento dell'audio"));
27             System.out.println(Colors.redBg(e.getMessage()));
28         }
29     }
30
31     /**
32      * Metodo che fa partire la riproduzione del file audio
33      */
34     public synchronized void play() {
35         try{
36             AudioInputStream input = AudioSystem.getAudioInputStream(new File(soundPath));
37             clip.open(input);
38             clip.start();
39             clip.loop(Clip.LOOP_CONTINUOUSLY);
40         }catch(Exception e){
41             System.out.println(Colors.redBg("Errore nel caricamento dell'audio"));
42             System.out.println(Colors.redBg(e.getMessage()));
43         }
44     }
45
46     /**
47      * Metodo che interrompe la riproduzione del file audio
48      */
49     public void stop(){
50         clip.stop();
51         clip.close();
52     }
53
54     /**
55      * Metodo getter che ritorna il percorso del file audio
56      * @return percorso del file audio
57      */
58     public String getSoundPath() {
59         return soundPath;
60     }
61
62     /**
63      * Metodo setter che imposta il percorso del file audio
64      * @param soundPath percorso del file audio (solo file .wav)
65      */
66     public void setSoundPath(String soundPath) {
67         SoundPlayer.soundPath = soundPath;
68     }
69 }
70

```

```

1 /**
2 * Classe PratoFiorito che permette di gestire il prato
3 * e la posizione de fiori
4 *
5 * @author Davide Branchi
6 */
7 public class PratoFiorito {
8     private int dimension;
9     private String[][] prato;
10    private String[][] fiori;
11
12    /**
13     * Metodo costruttore della classe che definisce la dimensione del prato
14     * @param dimension dimensione del prato
15     */
16    public PratoFiorito(int dimension){
17        setDimension(dimension);
18    }
19
20    /**
21     * Metodo che ritorna il prato come stringa
22     * @return stringa equivalente al prato
23     */
24    public String getPrato() {
25        String output = "";
26        for(int i = 0; i < prato.length; i++){
27            for(int j = 0; j < prato[i].length; j++){
28                if(prato[i][j].equals("*")){
29                    output += Colors.red(prato[i][j]) + " ";
30                }else{
31                    output += Colors.green(prato[i][j]) + " ";
32                }
33            }
34            output += "\n";
35        }
36        return output;
37    }
38
39    /**
40     * Metodo setter che imposta una matrice stringa per il prato
41     * @param prato matrice da utilizzare come prato
42     */
43    private void setPrato(String[][] prato) {
44        this.prato = prato;
45    }
46
47    /**
48     * Metodo che ritorna la matrice dei fiori come stringa
49     * @return stringa equivalente alla matrice dei fiori
50     */
51    public String getFiori() {
52        String output = "";
53        for(int i = 0; i < fiori.length; i++){
54            for(int j = 0; j < fiori[i].length; j++){
55                if(fiori[i][j].equals("*")){
56                    output += Colors.red(fiori[i][j]) + " ";
57                }else{
58                    output += Colors.green(fiori[i][j]) + " ";
59                }
60            }
61            output += "\n";
62        }
63        return output;
64    }
65
66    /**
67     * Metodo setter che imposta una matrice stringa come fiori
68     * @param fiori matrice da utilizzare come fiori
69     */
70    public void setFiori(String[][] fiori) {
71        this.fiori = fiori;
72    }
73
74    /**
75     * Metodo getter che ritorna la dimensione del prato
76     * @return dimensione del prato
77     */
78    public int getDimension() {
79        return dimension;
80    }
81
82    /**

```

```

83     * Metodo setter che imposta la dimensione del prato
84     * @param dimension dimensione del prato
85     */
86    public void setDimension(int dimension) {
87        if(dimension < 2 || dimension > 50){
88            this.dimension = 10;
89        }else{
90            this.dimension = dimension;
91        }
92    }
93
94 /**
95  * Metodo che inizializza il prato e i fiori, la matrice del
96  * prato viene riempita con il carattere "#"
97  * la matrice dei fiori invece viene riempita casualmente con i fiori
98  * rappresentati dal carattere "*"
99 */
100 public void inizializzaPrato(){
101     prato = new String[dimension][dimension];
102     fiori = new String[dimension][dimension];
103     for(int i = 0; i < dimension; i++){
104         for(int j = 0; j < dimension; j++){
105             prato[i][j] = "#";
106             fiori[i][j] = "#";
107         }
108     }
109     for(int i = 0; i < dimension; i++){
110         int r = random(0, dimension-1);
111         int c = random(0, dimension-1);
112         if(fiori[r][c].equals("#")){
113             fiori[r][c] = "*";
114         }else{
115             i--;
116         }
117     }
118 }
119
120 /**
121  * Metodo che stampa il prato utilizzando gli ANSI escape codes
122 */
123 public void printPrato(){
124     for(int i = 0; i < prato.length; i++){
125         for(int j = 0; j < prato[i].length; j++){
126             if(prato[i][j].equals("*")){
127                 System.out.print(Colors.red(prato[i][j]) + " ");
128             }else{
129                 System.out.print(Colors.green(prato[i][j]) + " ");
130             }
131         }
132         System.out.println();
133     }
134 }
135
136 /**
137  * Metodo che stampa i fiori utilizzando gli ANSI escape codes
138 */
139 public void printFiori(){
140     for(int i = 0; i < fiori.length; i++){
141         for(int j = 0; j < fiori[i].length; j++){
142             if(fiori[i][j].equals("*")){
143                 System.out.print(Colors.red(fiori[i][j]) + " ");
144             }else{
145                 System.out.print(Colors.green(fiori[i][j]) + " ");
146             }
147         }
148         System.out.println();
149     }
150 }
151
152 /**
153  * Metodo che genera un numero casuale
154  * fra il minimo e il massimo (compresi)
155  * @param min minimo
156  * @param max massimo
157  * @return numero casuale fra min e max
158 */
159 private int random(int min, int max){
160     return (int) Math.floor(Math.random()*(max-min+1)+min);
161 }
162
163 /**
164  * Metodo che controlla se la cella passata tramite row e colum

```

```

165 * è valida (non è già stata scoperta)
166 * @param row riga della cella
167 * @param colum colonna della cella
168 * @return true se la cella è valida, false altrimenti
169 */
170 public boolean isAValidCell(int row, int colum){
171     if(prato[row-1][colum-1].equals("#")){
172         return true;
173     }else{
174         return false;
175     }
176 }
177 /**
178 * Metodo che controlla se la cella passata contiene un fiore
179 * @param row riga della cella
180 * @param colum colonna della cella
181 * @return true se la cella contiene un fiore, false altrimenti
182 */
183 public boolean isFlowerCell(int row, int colum){
184     if(fiori[row-1][colum-1].equals("*")){
185         prato[row-1][colum-1] = "*";
186         return true;
187     }else{
188         return false;
189     }
190 }
191 /**
192 * Metodo che scopre la cella passata
193 * @param row riga della cella
194 * @param colum colonna della cella
195 */
196 public void mineCell(int row, int colum){
197     prato[row-1][colum-1] = " ";
198 }
199 }
200 }
201 }
202

```