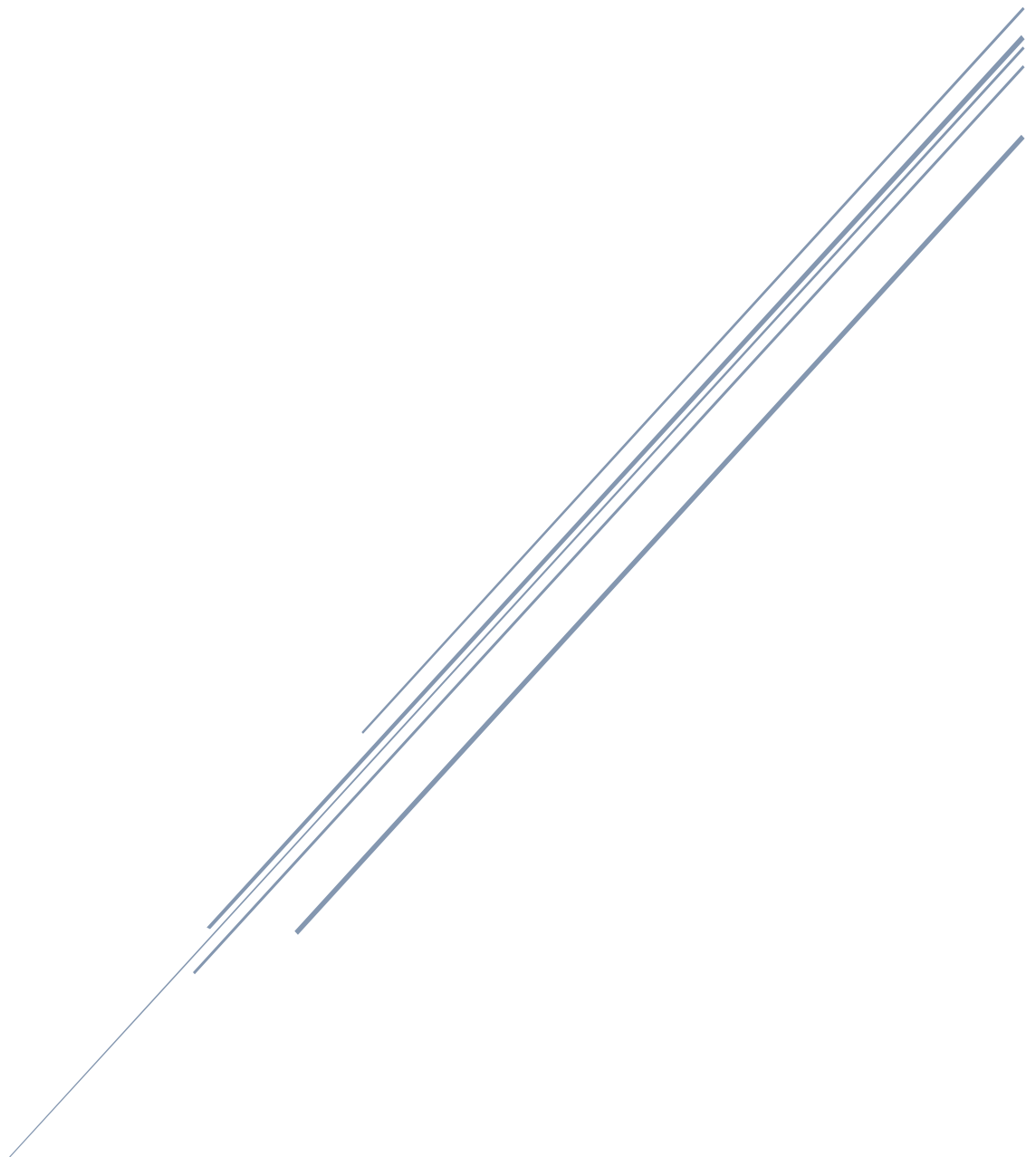


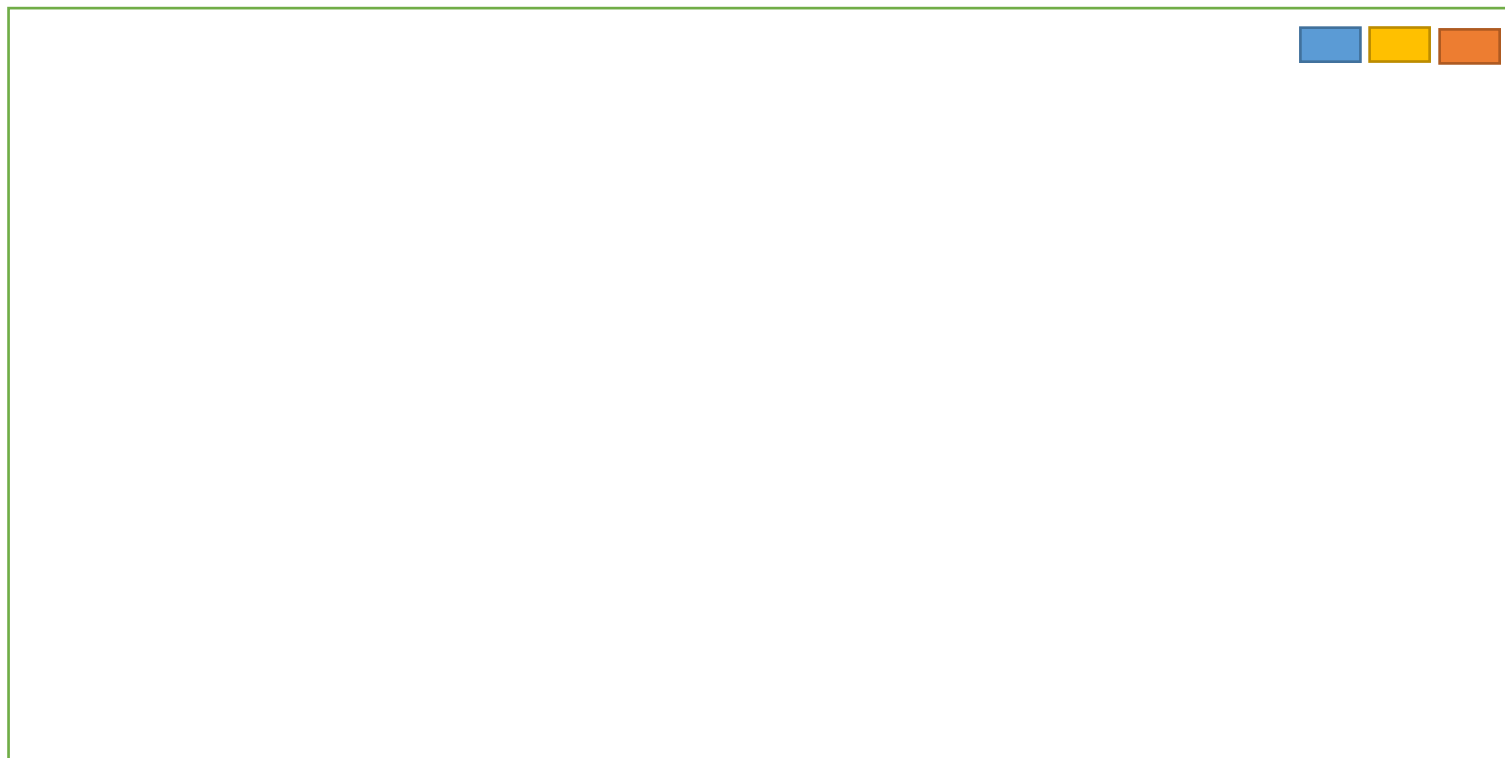
PUZZLE ART !

Created by Federico March and Isabel ToroSima

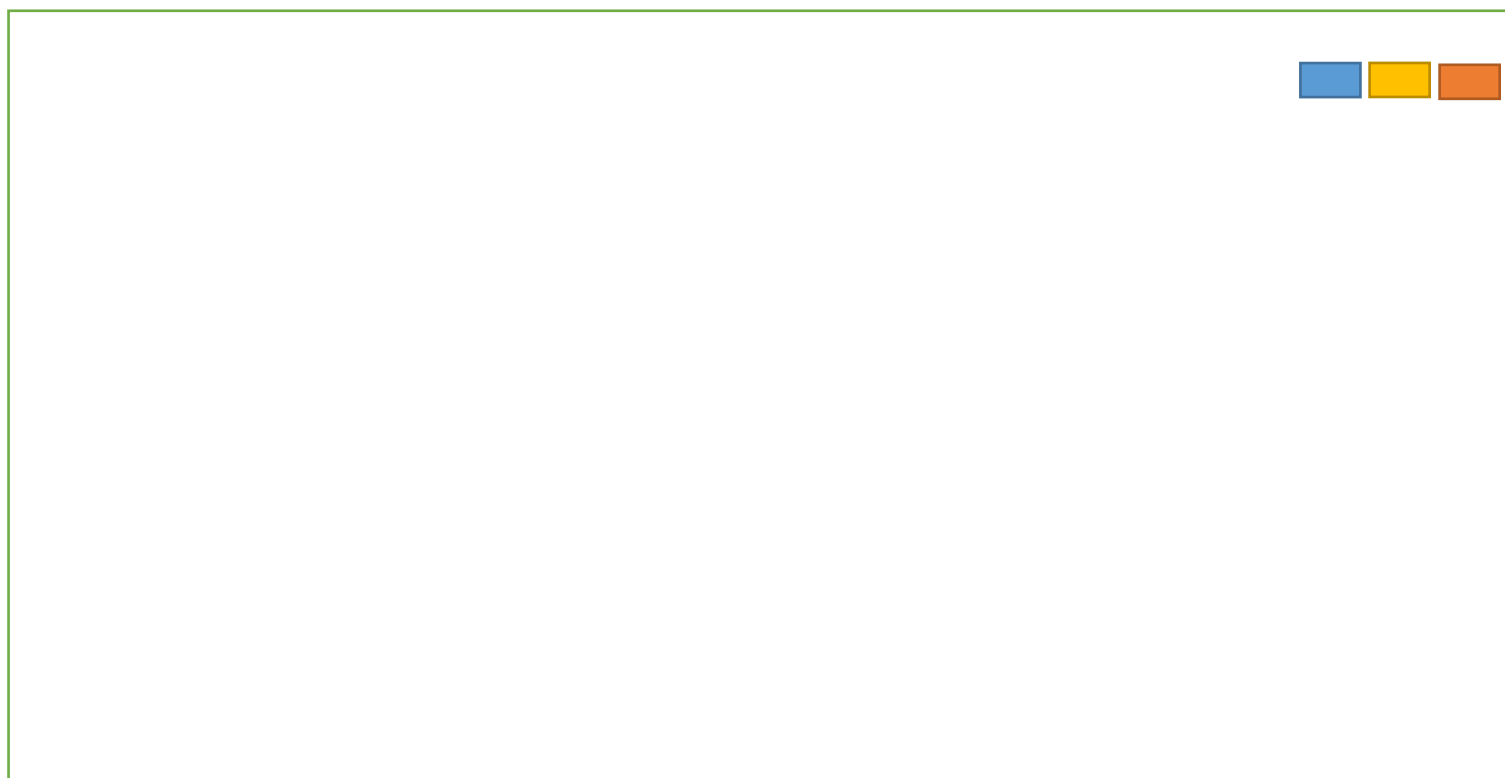


STM-Tor Vergata
Programmazione in Java e Gestione della Grafica

Schermata Play



Schermata di Livello



IMPORTA_AUDIO extends File

```
{
    private String s;
    private Clip audio_clip;

    Costruttore (String)
    {
        Super(String); essendo file la superclasse.

        Prova a creare un file audio, importando oggetti da javax.sound.sample .

        In questa classe vengono importati file di tipo WAV, usando oggetti di
        tipo File e Clip.

        Inizializziamo un oggetto audio_clip di tipo Clip, all'interno di un try e
        catch e prendiamo la clip con il metodo getClip() dall'AudioSystem.

        Carichiamo ora la clip audio che vogliamo sentire tramite il comando:
        audio_clip.open(AudioSystem.getAudioInputStream(File Sound)).

        Il metodo getAudioInputStream() prende come argomento un oggetto di tipo
        File. Dunque , poiche' la classe Importa_Audio extends File, possiamo
        riscrivere il comando in questo modo:

        audio_clip.open(AudioSystem.getAudioInputStream(this)).

    }
```

Metodi:

```
public void PlaySound() permette di avviare la riproduzione del file audio
importato.

public void StopSound() permette di stoppare la riproduzione del file audio
importato.

public void RepeatInALoopSound() permette di riprodurre in loop il file
audio importato, attraverso il metodo audio:clip.loop(int) →
audio_clip.loop(Clip.LOOP_CONTINUOUSLY). (Poiche' audio_clip e' un oggetto
della classe Clip).

}
```

IMPORTA IMMAGINE

Nota: Permette di importare un file di tipo immagine, avendo come input una string in cui e' contenuto il titolo.formato del file che si desidera importare all'interno del programma.

```
{
    private String s;
```

Costruttore(String)

```
{
    Vengono importati oggetti della classe java.awt.Image
    Prende in input la stringa contenente il titolo.formato del file immagine
    da importare e la inizializza con la stringa presa dall'argomento.
    Invoca il metodo getImmagine(String).
}
```

Metodi:

```
public Image getImmagine(String) restituisce l'immagine richiesta.
Sfrutta oggetti importati da javax.swing.ImageIcon, nel seguente modo:
Image img = new ImageIcon(getClass().getResource(String)).getImage().
Return img;

getClass().getResource(String) rappresenta il percorso da effettuare per
raggiungere la cartella contenente il file di tipo immagine desiderato.
}
```

GRIGLIAFOTOCOMPONENT

Nota: Permette di creare un griglia di rettangoli. Tale classe si limita a creare solamente i rettangoli come oggetti, senza disegnarla. Sarà ,poi, compito della classe AssegnazioneFotoComponent disegnare la griglia di rettangoli sul Frame.

```
{
    private double x,y,l,h;//coordinate x e y della posizione, l e h altezza e
    lunghezza del rettangolo
    private int gridLines;//righe della griglia
    private int gridColumns;//colonne della griglia
    private Rectangle2D.Double[][] fotoGriglia;//griglia
```

Costruttore (x,y,l,h,gl,cg)
{

 Inizializza le variabili e crea una griglia di rettangoli. La griglia di rettangoli viene interpretata come una matrice (o array bidimensionale) di oggetti di tipo rectangle2D.Double

La griglia sarà, dunque, una matrice costituita da gl linee (o righe)(mat.lenght) e gc colonne (mat[0].length.

Posizionamento dei rettangoli all'interno della griglia:

doppio ciclo for:

si crea un'oggetto di tipo Rectangle2D.Double rect di default , importandolo dalla classe java.awt.geom

Il primo rettangolo (alla posizione i=0 e j=0) e' il rettangolo principale, dal quale poi possiamo impostare gli altri.

Impostiamo le coordinate e le dimensioni tramite il metodo void rect.setRect(x,y.l,h).

Assegnamo ora l'oggetto rect ottenuto all'elemento grid[i][j] della griglia (matrice) .

Creiamo la prima riga, costituita da tutti quei rettangoli che soddisfano la seguente condizione : prima riga (i=0) e (j>0)

Il rettangolo inserito nella seconda colonna della prima riga avrà le stesse dimensioni, la stessa ordinata y del primo rettangolo creato,ma la ascissa x sarà traslata verso destra di una quantità pari alla lunghezza del rettangolo principale, e moltiplicando il valore della lunghezza per j (in questo modo stiamo indicando in quale colonna dobbiamo inserire il rettangolo; ad esempio j=2 x=x+2*l inserisce un rettangolo nella seconda colonna, mentre j=5 x=x+5*l inserisce un rettangolo nella quinta colonna), ovvero $x = x + j * l$. Tale ragionamento logico si adopera per tutti gli altri rettangoli che comporgono la prima riga.

Dunque, per la prima riga si ottiene che

`Rect.setrect(x+j*l,y,l,h)`. Assegnamo ora l'oggetto `rect` ottenuto all'elemento `grid[i][j]` della griglia (matrice) .

Creiamo la prima colonna, costituita da tutti quei rettangoli che soddisfano la seguente condizione : prima colonna ($j=0$) e ($i>0$)

Il rettangolo inserito nella seconda riga della prima colonna avr  le stesse dimensioni, la stessa ascissa x del primo rettangolo creato, ma l'ordinata y sar  traslata verso il basso di una quantita' pari 'altezza del rettangolo principale, e moltiplicando il valore dell'altezza per i (in questo modo stiamo indicando in quale riga dobbiamo inserire il rettangolo; ad esempio $i=2$ $y=y+2*h$ inserisce un rettangolo nella seconda riga, mentre $i=5$ $y=y+5*h$ inserisce un rettangolo nella quinta riga), ovvero $y = y+i*h$. Tale ragionamento logico si adopera per tutti gli altri rettangoli che compongo la prima colonna.

Dunque, per la prima colonna si ottiene che

`Rect.setrect(x,y+i*h,l,h)`. Assegnamo ora l'oggetto `rect` ottenuto all'elemento `grid[i][j]` della griglia (matrice) .

Per le altre righe e colonne si combinano i ragionamenti visti per la prima riga e la prima colonna. Ad esempio il rettangolo inserito alla seconda riga e alla terza colonna avr  coordinate $x=x+3*l$ e $y=y+2*h$ e dimensioni del primo rettangolo creato.

$X=x + j*l$, $Y=y+i*h$

Dunque, per le altre righe e colonne ($i>0$ e $j>0$) si ottiene che

`Rect.setrect(x+j*l,y+i*h,l,h)`. Assegnamo ora l'oggetto `rect` ottenuto all'elemento `grid[i][j]` della griglia (matrice) .

}

Griglia:

Metodi:

public double getAscisse() restituisce il valore dell'ascissa x del rettangolo

public double getOrdinate() restituisce il valore dell'ordinata y del rettangolo

public double getLung() restituisce il valore della lunghezza del rettangolo

public double getAltezza() restituisce il valore dell'altezza del rettangolo

public Rectangle2D.Double[][] getGriglia() restituisce la griglia (matrice mxn (glxgc)) creata

public int getRighe() restituisce il numero di righe della griglia (matrice) creata.

public int getColonne() restituisce il numero di colonne della griglia (matrice) creata.

}

CRONOMETRO implements ActionListener

Nota: Crea un contatore che permette di comunicare all'utente il tempo rimasto a disposizione per completare il livello, decrementando i secondi (e quindi anche i minuti) ad ogni tick di un timer. Tale classe fara' da ascoltatore per l'oggetto timer che si occupera' della gestione del tempo rimasto a disposizione per completare il livello corrente. I secondi rimanenti saranno stampati su una label, la quale sara' aggiunta ad un pannello, per essere poi montati , in seguito, tramite la classe AssegnazioneFotoComponent , sulla schermata di gioco.

```
{
    private int minute;
    private int second;
    private int secondCent;
    private JLabel timer_screen;

    private String msg=" ";
    private JPanel screen;
    private JPanel main_panel;

    Costruttore(minute, second, JPanel)
    {
        Vengono inizializzate le variabili d'istanza.

        Viene creata la label di testo che mostra il tempo rimasto.
    }
}
```

Metodi:

public void actionPerformed(ActionEvent) imposta l'aggiornamento del tempo, decrementando i minuti e i secondi ogni 25 centosecondi.

Quando i centosecondi raggiungeranno il valore -100, i centosecondi verranno impostati a 0, mentre i secondi saranno decrementati di un'unita'.

Quando i secondi raggiungono valore -1 (perche' si deve mostrare il valore 0 per i secondi, che corrisponde a 60 secondi, ovvero il minuto precedente il minuto corrente; dunque 1 min e -1 s corrisponde a 0 min 59 s), i secondi vengono impostati a 59, mentre i minuti verranno decrementati di un'unita'.

Il testo della Label viene aggiornato, mostrando i minuti e secondi rimanenti.

Quando si raggiunge lo scadere del tempo (min=0 e s=0) viene importato e riprodotto un file audio (tramite la classe Importa_Audio) "Gong.WAV" e dopo aver mostrato "GAME OVER!" , il programma viene chiuso.

Infine, la label viene aggiunta ad un pannello, e successivamente, il pannello viene aggiunto al pannello principale (che sara' il pannello relativo alla schermata di gioco).

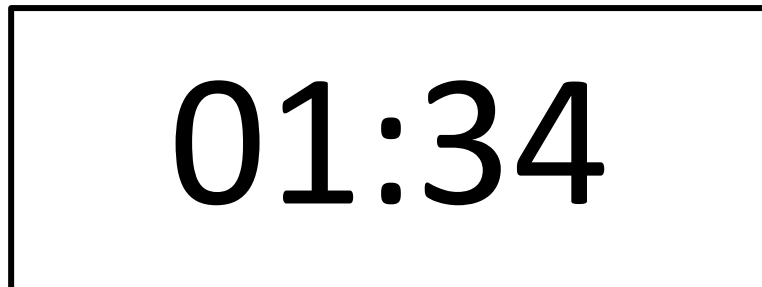
```
public int getMinute() restituisce il valore dei minuti
```

```
public int getSecond() restituisce il valore dei secondi
```

```
public JPanel getScreen() restituisce il pannello su cui e' stata montata  
label. Tale pannello sara' montato sul pannello principale relativo alla  
schermata di gioco.
```

Gli oggetti di tipo Component sono importati da javax.swing, ActionListener e ActionEvent sono importati da java.awt.(event)

```
}
```



01:34

CardViewer extends JFrame

Nota: Permette creare una finestra in cui verra' mostrata l'immagine da visualizzare , che si aprira' al click sul tasto "View Image".

```
{  
  
    private JLabel photo_label;  
    private Image icon;
```

Costruttore (String, JLabel, Image)

```
{  
  
    Super(String); essendo JFrame la superclasse.  
  
    L'immagine passata come argomento viene usata per creare un'icona.  
    L'icona viene poi impostata come icona della label, tramite il metodo void  
    label.setIconImage(icon).  
  
    Viene inizializzata la variabile relativa alla label, e una volta  
    inizializzata, aggiunta al frame.  
  
}
```

Metodi:

```
non presenti.  
  
}
```

Gli oggetti di tipo Component sono stati importati da javax.swing



VIEWERACTION implements ActionListener

Nota: Permette di creare le azioni che saranno svolte quando verra' premuto il tasto "View Image". In particolare apre una finestra in cui viene mostrata l'immagine da ricostruire.

```
{  
    private CardViewer viewer;
```

Costruttore (CardViewer)

```
{  
    Inizializza le variabili d'istanza.  
  
}
```

Metodi:

```
public void actionPerformed(ActionEvent)  imposta il CardViewer, e rende il  
Frame del CardViewer visibile, ponendo la condizione HIDE_ON_CLOSE al frame.  
  
Public void setViewer(CardViewer)  imposta il CardViewer.  
  
}
```

ActionListener e ActionEvent sono stati importati da java.awt.(event)

ASSEGNAZIONE FOTOCOMPONENT extends JPanel

Nota: crea un pannello in cui viene disegnata la griglia e le immagini con coordinate spaziali randomizzate. Tale classe contiene la classe Muovi_Immagine che permette di spostare le immagini nel frame e nella griglia nella giusta posizione. La classe Muovi_immagine contiene l'istruzione per passare ai livelli successivi.

```
{
    private Rectangle2D.Double[] rect_array; // array di rettangoli.
    private Image[] img_array; // array di foto.
    private ArrayList<Rectangle2D.Double> griglia_estesa; // griglia di
    rettangoli che si sviappa orizzontalmente.
    private JPanel grid_panel;
    private Importa_Audio audio;
    private Rectangle2D.Double[][] gfc; //griglia di rettangoli.
    private Rectangle2D.Double[][] griglia_ordinata;
    private Rectangle2D.Double[] grid_array; //contiene i rettangoli che
    costituiscono la griglia.

    private JFrame level;
    private Timer time;
    private CardViewer il_bacio,monaLisa,notteStellata;

    private JLabel points;
    private Rectangle2D.Double border_mask1,border_mask2,border_mask3; //per
    evitare il messaggio posizione errata se l'immagine non viene spostata nella
    griglia.
    //Rispettivamente:
    //border_mask1 e' il rettangolo maschera che va dal punto 0,0 e si estende
    fino a 3 pixel prima della griglia.
    //border_mask2 e' il rettangolo maschera che si estende 3 pixel al di
    sotto della fine della griglia.
    //border_mask3 e' il rettangolo maskera che si estende 3 pixel a sinistra
    della griglia.

    Costruttore(Importa_Audio audio,JFrame level,Rectangle2D.Double[] array1_rect,
    Image[] array2_img, Rectangle2D.Double[] grid_array,Rectangle2D.Double[][] gfc)
    {
        Inizializza le variabili.

        Si crea la Label relativa al punteggio che mostra i punti. A seconda della
        lunghezza dell'array di rettangoli che viene passato nel costruttore, si
        distinguono i seguenti casi:
        lunghezza 4 --> primo livello, 0 punti
        lunghezza 9 --> secondo livello, 40 punti
        lunghezza 16 --> terzo livello, 130 punti
    }
}
```

un pannello che ospiterà il timer screen. Si imposta il timer che, prendendo come ascoltatore un oggetto di tipo Cronometro, determina il tempo a disposizione entro cui completare il livello.

A seconda della lunghezza dell'array di rettangoli che viene passato nel costruttore, si distinguono i seguenti casi:

```
lunghezza 4 --> primo livello, timer 2min 00 s
lunghezza 9 --> secondo livello, timer 1 min 45 se
lunghezza 16 --> terzo livello, timer 1 min 20 s
```

si avvia il timer e si aggiunge il tutto al pannello che ospita il timer. Il tutto poi si aggiunge al pannello principale (ovvero il pannello relativo alla schermata di gioco, che viene creata qui. Poiché la classe AssegnazioneFotoComponent extends JFrame, si può utilizzare this, per indicare il pannello principale relativo alla schermata di gioco.

Si crea un ArrayList<Rectangle2D.Double> che ospiterà i rettangoli a cui sarà associata la parte di immagine da ricostruire corrispondente, ovvero al rect1 sarà associata part1, rect2 part2, e così via.

Dopo aver ordinato i vari rettangoli dell'array[] nell'ArrayList, si passa a descrivere il comando che permette di randomizzare la posizione degli elementi all'interno dell'ArrayList.

Tale comando è:

```
Collections.shuffle(ArrayList<Rectangle2D.Double>) .
```

Il metodo void shuffle(ArrayList<?> list) della Class Collections è un metodo statico (quindi è possibile invocarlo o in un main o in un costruttore) che mescola, "disordina", gli elementi della lista utilizzando una sorgente di casualità predefinita.

Si passa a richiamare il metodo che ordina gli elementi dell'ArrayList<Rectangle2D.Double> in una griglia nxn o mxn, dove gl sono il numero di righe e gc sono il numero delle colonne della griglia (interpretata come una matrice rectangle2D.Double[][]).

Si aggiunge alla schermata di gioco il JButton "ViewImage" che permette di visualizzare l'immagine da ricostruire.

Si passa alla costruzione del CardViewer che contiene l'immagine da visualizzare.

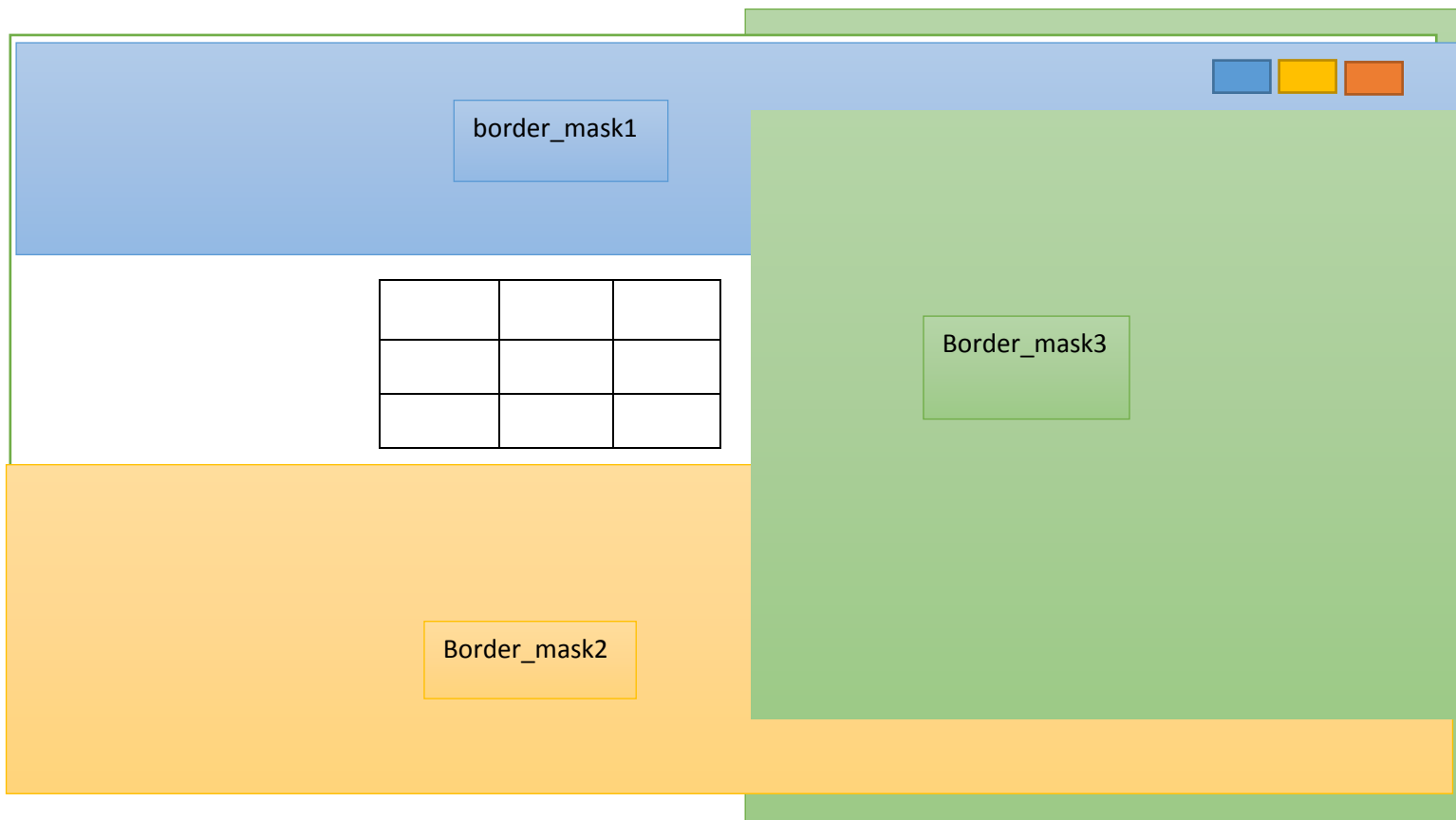
A seconda della lunghezza dell'array di rettangoli che viene passato nel costruttore, si distinguono i seguenti casi:

```
lunghezza 4 --> primo livello, CardViewer il_bacio,
lunghezza 9 --> secondo livello, CardViewer MonaLisa,
lunghezza 16 --> terzo livello, CardViewer NotteStellata
```

Una volta creato il CardViewer, si passa a creare l'azione ad esso associata, ovvero la ViewerAction, che permette di aprire una finestra contenente l'immagine richiesta al click sul JButton.

Vengono costruiti dei rettangoli maschera di tipo Rectangle2D.Double che evitano che appaia il popup "Posizione Errata!" nel caso in cui il rettangolo contenente l'immagine sia spostato e rilasciato tramite mouse in un area del frame situata

al di fuori della griglia. I rettangoli maschera sono posizionati in base alle dimensioni della griglia:



Il `border_mask1` e' un `Rectangle2D.Double` che si estende dalla X e Y del frame fino a 30 precedenti la Y della griglia. Il `border_mask2` e' un `Rectangle2D.Double` che si estende verticalmente a partire dai 30 pixel seguenti la fine della griglia. Il `border_mask3` e' un `rectangle2D.Double` che si estende orizzontalmente a partire dai 30 pixel seguenti la fine della griglia.

Infine, il costruttore aggiunge il movimento alle immagini, creando un oggetto di tipo `Muovi_Immagine(Timer,JPanel,Rectangle2D.Double grid_array, Rectangle2D.Double rect_array)`.

L'oggetto `Muovi_Immagine` che implementa `MouseListener` e `MouseMotionListener` viene aggiunto al pannello relativo alla schermata di gioco.

Si effettua un `Repaint` del pannello, tramite il metodo `repaint()` della classe `swing`. Il metodo `repaint()` non pu' essere sovrascritto.

Tale metodo controlla il ciclo `update()` → `paint()`, costringendo ogni componente del frame e del pannello a ridisegnare se stessa, sia nel caso in cui si abbia cambiato una sua proprieta' sia in caso in cui non si sia cambiato nulla relativamente alla componenente.

}

Metodi:

public Image getViewerImageIcon() Restituisce l'icona / immagine desiderata.

A seconda della lunghezza dell'array di rettangoli che viene passato nel costruttore, si distinguono i seguenti casi:
 lunghezza 4 --> primo livello, viewer_livell1
 lunghezza 9 --> secondo livello, viewer_level2
 lunghezza 16 --> terzo livello, viewer_level3
public Rectangle2D.Double[][] ordinaInGriglia(ArrayList<rectangle2D.Double> ordina gli elementi di un ArrayList (esteso orizzontalmente) all'interno di una griglia. In questa griglia verranno rodinati i rettangoli a cui e' stata associata un'immagine e ai quali e' stato fatto un mescolamento casuale (shuffle).

Griglia nxn o Griglia mxn

--	--	--	--

Si crea una griglia nxn, rappresentata da una matrice (array bidimensionale) , di tipo rectangle2D.Double .

Si inizializzano nuove variabili x2,y2,l2,h2 relative ai nuovi rettangoli che comporranno la griglia , inizializzati con l'elemento alla posizione 0 dell'array di rettangoli importato nell'argomento del costruttore di AssegnazioneFotoComponent.

Si riempie la griglia in questo modo: innanzitutto e' necessario ripescare l'elemento corretto dell'ArrayList e posizionarlo nella giusta posizione, ovvero l'elemento 0 dell'ArrayList andra' in posizione [0][0] nella griglia, l'elemento in posizione 1 in [0][1] e cosi' via.

Tale algoritmo e' il seguente. In un doppio ciclo for l'espressione

List.get(grid_columns*i + j) restituisce il rettangolo giusto dell'ArrayList nella posizione giusta nella griglia.

Passiamo alla disposizione in griglia.

si crea un'oggetto di tipo Rectangle2D.Double rect di default , importandolo dalla classe java.awt.geom

si pone rect = list.get(grid_column*i + j).

Il primo rettangolo (alla posizione i=0 e j=0) e' il rettangolo principale, dal quale poi possiamo impostare gli altri.

Impostiamo le coordinate e le dimensioni tramite il metodo void rect.setRect(x2,y2.l2,h2).

Assegnamo ora l'oggetto rect ottenuto all'elemento grid[i][j] della griglia (matrice) .

Creiamo la prima riga, costituita da tutti quei rettangoli che soddisfano la seguente condizione : prima riga ($i=0$) e ($j>0$)

Il rettangolo inserito nella seconda colonna della prima riga avra' le stesse dimensioni, la stessa ordinata y del primo rettangolo creato, ma la ascissa x sara' traslata verso destra di una quantita' pari alla lunghezza del rettangolo principale, e moltiplicando il valore della lunghezza per j (in questo modo stiamo indicando in quale colonna dobbiamo inserire il rettangolo; ad esempio $j=2$ $x=x+2*1$ inserisce un rettangolo nella seconda colonna, mentre $j=5$ $x=x+5*1$ inserisce un rettangolo nella quinta colonna), ovvero $x = x+j*1$. Tale ragionamento logico si adopera per tutti gli altri rettangoli che compongo la prima riga.

Dunque, per la prima riga si ottiene che

`Rect.setrect(x2+j*12,y2,12,h2)`. Assegnamo ora l'oggetto rect ottenuto all'elemento grid[i][j] della griglia (matrice) .

Creiamo la prima colonna, costituita da tutti quei rettangoli che soddisfano la seguente condizione : prima colonna ($j=0$) e ($i>0$)

Il rettangolo inserito nella seconda riga della prima colonna avra' le stesse dimensioni, la stessa ascissa x del primo rettangolo creato, ma l' ordinata y sara' traslata verso il basso di una quantita' pari 'altezza del rettangolo principale, e moltiplicando il valore dell'altezza per i (in questo modo stiamo indicando in quale riga dobbiamo inserire il rettangolo; ad esempio $i=2$ $y=y+2*h$ inserisce un rettangolo nella seconda riga, mentre $i=5$ $y=y+5*h$ inserisce un rettangolo nella quinta riga), ovvero $y = y+i*h$. Tale ragionamento logico si adopera per tutti gli altri rettangoli che compongo la prima colonna.

Dunque, per la prima colonna si ottiene che

`Rect.setrect(x2,y2+i*h2,12,h2)`. Assegnamo ora l'oggetto rect ottenuto all'elemento grid[i][j] della griglia (matrice) .

Per le altre righe e colonne si combinano i ragionamenti visti per la prima riga e la prima colonna. Ad esempio il rettangolo inserito alla seconda riga e alla terza colonna avra' coordinate $x=x+3*1$ e $y=y+2*h$ e dimensioni del primo rettangolo creato.

$X=x +j*1$, $Y=y+i*h$

Dunque, per le altre righe e colonne ($i>0$ e $j>0$) si ottiene che

`Rect.setrect(x2+j*12,y2+i*h2,12,h2)`. Assegnamo ora l'oggetto rect ottenuto all'elemento grid[i][j] della griglia (matrice) .

--	--	--	--	--	--	--	--

Public void paintComponent(Graphics) che richiama il metodo paintComponent(Graphics) della superclasse JPanel; attraverso un oggetto di tipo Graphics importato da java.awt e poi convertito in un oggetto di tipo Graphics2D vengono disegnati sullo stesso pannello sia la griglia scheletro di soli rettangoli (attraverso il metodo g_2D.draw(shape)) sia le immagini attraverso il metodo

(g_2D.drawImage(Image,int X,int Y, int L, int H, ImageObserver null)).

Ogni immagine avra' la x e la y e le dimensioni del rettangolo a cui sono associate per permettere lo spostamento tramite mouse attraverso il frame.

X=rect.getX(),

Y=rect.getY(),

L

In questo modo le coordinate dell'immagine variano a seconda di come variano le coordinate del rettangolo. Se avessimo inserito dei numeri al posto di X=rect.getX() e Y=rect.getY(), come ad esempio X=300, Y=600, l'immagine sarebbe rimasta fissa alla posizione di coordinate (300,600) in quanto i valori passati sono valori costanti e non valori variabili.

public Rectangle2D.Double[] getRectangleArray() restituisce l'array relativo ai rettangoli.

public Image[] getImageArray() restituisce l'array relativo alle immagini

public ArrayList<Rectangle2D.Double> getRectangleArrayList() restituisce l'ArrayList di tipo <rectangle2D.Double>

}

BREVE DESCRIZIONE DELL'ALGORITMO CHE GENERA IL PANNELLO CHE CONTIENE GRIGLIA E IMMAGINI

La classe AssegnazioneFotoComponent crea un pannello contenente la griglia "scheletro" su cui posizionare le immagini per ricostruire la foto completa e le varie parti sparpagliate dell'immagine. Viene creata una griglia nxn (matrice nxn di rettangoli di tipo Rectangle2D.Double), la quale viene disegnata sul pannello con il paint component. Da tale griglia si estrapolano poi i vari elementi (rettangoli) che la costituiscono e si crea un array griglia (contenente i vari rettangoli/sezioni che costituiscono la griglia scheletro su cui ricostruire l'immagine). Si ricrea un'altra griglia nxn di rettangoli di tipo Rectangle2D.Double e da essa si estrapola un array di rettangoli che, in seguito, conterranno le immagini relative alle parti della foto da ricostruire. Gli elementi (rettangoli) di questo array vengono, poi, inseriti in un array list. (sia nell'array di rettangoli, che nell'arraylist i rettangoli sono ordinati in modo sequenziali, ovvero alla posizione 0 troviamo il rettangolo 1, alla posizione 1, il rettangolo 2 e così via..) Si passa ad associare al rettangolo 1 la parte 1 dell'immagine, al rettangolo 2 la parte 2 e così via tramite il metodo drawImage del paint component. L'immagine avrà le stesse coordinate e le stesse dimensioni del rettangolo a cui fa riferimento. Ecco perché i rettangoli sono stati ordinati in modo sequenziali nell'array e nell'arraylist, in modo tale che l'assegnazione delle foto potesse risultare facile. Una volta associate le immagini, in uno step successivo, si potrà randomizzare la posizione delle varie parti dell'immagine da ricostruire. Inserire tali rettangoli in un array list serve per poter lanciare lo shuffle, che randomizzerà le posizioni dei vari rettangoli all'interno dell'arraylist. Dopo lo shuffle, ciò che cambia sono solo le POSIZIONI DEI RETTANGOLI ALL'INTERNO DELL'ARRAYLIST, mentre LE COORDINATE SPAZIALI DEI RETTANGOLI RIMANGONO LE MEDESIME. Attraverso il metodo ordinaGriglia(ArrayList list), gli elementi dell'arraylist vengono ordinati in una griglia nxn (ovvero il primo elemento dell'arraylist [alla posizione 0] sarà inserito nella posizione 0,0 della griglia, il secondo elemento dell'array list [alla posizione 1] sarà inserito nella posizione 0,1 nella griglia e così via, fino a riempire tutte le righe e colonne della griglia. [ovviamente alla posizione 0 dell'array list non ci sarà più il rettangolo 1 ma un altro rettangolo poiché è stato lanciato lo shuffle.]. Questa volta le coordinate nello spazio dei vari rettangoli vengono modificate. Ed ecco che si possono ottenere le parti dell'immagine in modo sparpagliato e casuale.

MUOVI IMMAGINE extends MouseAdapter

Nota: Permette spostare i rettangoli tramite mouse. Tale classe si divide in tre parti, riconducibili alle parti che costituiscono un evento Mouse: premere il mouse (mousePressed), il trascinamento (mouseDragged) e il rilascio del mouse (mouseReleased). In questa classe sono gestite le modalita' per passare al livello superiore.

```
{
    Private Rectangle2D.Double[] grid;    // griglia
    private Rectangle2D.Double[] rett;    // rettangoli ai quali sono
    asociate le immagini
    private boolean var=false;
    private Point2D pos;
    private int punteggio;
    private Rectangle2D.Double rettangolo_in_movimento; //rettangolo di
    referimento che verra' spostato.
    private Rectangle2D.Double griglia_fissa; // griglia fissa di
    referimento che dovra' contenere il rettangolo in movimento per avviare il
    gioco.
    private JFrame level;
    private Timer t;
    private String pp;
    private int index;

    Costruttore(Timer, JFrame, RectGrid[], Rect[])
    {
        Vengono iinizializzate le variabili. In particolare viene inizializzato il
        punteggio a seconda del tipo di livello.

        A seconda della lunghezza dell'array di rettangoli che viene passato nel
        costruttore, si distinguono i seguenti casi di inizializzazione della variabile
        di istanza this.punteggio:
            lunghezza 4 --> primo livello, this.punteggio =0
            lunghezza 9 --> secondo livello, this.punteggio=40
            lunghezza 16 --> terzo livello, this.punteggio=120
    }
}
```

Metodi:

```
public void mousePressed(MouseEvent e)    //metodo che gestisce il click del
mouse sull'immagine.
Si fa il click con il mouse. Se la posizione del mouse al click del mouse e'
dentro l'immagine allora si pone la variabile click a valore true.
```

Per fare cio' ci si riconduce al caso in cui si ha un solo rettangolo da spostare in una sola sezione di griglia.

Si associa alla variabile rettangolo_in_movimento un elemento dell'array Rett per ogni i del ciclo for, e si controlla che la posizione sia contenuta in rettangolo_in_movimento.

Si controlla ora, se la mossa e' valida e se l'immagine sia gia' stata posizionata al posto giusto nella griglia.

Se l'immagine e' gia stata posizionata al posto corretto nella griglia , si vuole che essa si blocchi.

Quindi, se il rettangolo che si trova alla giusta posizione contiene la posizione al click del mouse viene escluso dal trascinamento (ponendo var=false e facendo continuare il ciclo in caso in cui rettangolo_in_movimento sia uguale al rettangolo in questione),e si passa a controllare che la posizione sia contenuta in un altro rettangolo, ancora da posizionare.

// Le seguenti istruzioni sono relative al caso in cui un immagine I sia gia' stata posizionata al posto giusto nella griglia. Immaginiamo di spostare una seconda immagine M sopra l'immagine I coprendo una certa sezione A(viene coperta una certa sezione A dell'immagine I quando l'immagine I viene sormontata da un'immagine M, dove questa porzione A puo' essere piccola o grande quanto I). Se non vi fossero le seguenti istruzioni se si cliccasse nella sezione A per spostare M , M non si sposterebbe, in quanto la posizione al click del mouse risulterebbe essere dentro l'immagine I che e' bloccata perche' al posto giusto. Per spostare l'immagine M si dovrebbe fare click su m in una sezione che sia al di fuori della sezione A che copre I. Ma le seguenti istruzioni permettono di spostare M , cliccando ovunque su M, anche se M copre per intero, o per una piccola sezione,l'immagine I.

```

if(click ==true )
{
    if(p_giusta==true)
    {
        if(rett[index].contains(pos)) // Le seguenti
        istruzioni sono relative al caso in cui un immagine I sia gia' stata posizionata
        al posto giusto nella griglia. Immaginiamo di spostare una seconda immagine M
        sopra l'immagine I coprendo una certa sezione A(viene coperta una certa sezione
        A dell'immagine I quando l'immagine I viene sormontata da un'immagine M, dove
        questa porzione A puo' essere piccola o grande quanto I). Se non vi fossero le
        seguenti istruzioni se si cliccasse nella sezione A per spostare M , M non si
        spoterebbe, in quanto la posizione al click del mouse risulterebbe essere dentro
        l'immagine I che e' bloccata perche' al posto giusto. Per spostare l'immagine M
        si dovrebbe fare click su m in una sezione che sia al di fuori della sezione A
        che copre I. Ma le seguenti istruzioni permettono di spostare M , cliccando
        ovunque su M, anche se M copre per intero, o per una piccola sezione,l'immagine
        I.

        {
            for(int i=0; i<rett.length; i++)
            {
                this.rettangolo_in_movimento=rett[i];

                if(this.rettangolo_in_movimento==rett[index])
                {
                    var=false; //blocca l'immagine
                    che gia' e' al posto giusto al fine di evitare che il giocatore possa barare.
                    continue;
                }
                else
                {

```

```

        if(this.rettangolo_in_movimento.contains(pos)||
this.rettangolo_in_movimento.intersects(rett[index]))
        {
            var=true;
            break;
        }
    }
}
}
else
{
    var=true;
}
}
else
{
    var=false;    // se var = false stampa le coordinate
del punto preso al click del mouse
    System.out.println("Coordinata del punto al click
"+e.getX()+" "+e.getY());
}

```

public void mouseDragged(MouseEvent b)//metodo che gestisce il trascinamento delle immagini con il mouse

se var = true si impostano le condizioni che comportano il trascinamento dell'oggetto sul rettangolo, altrimenti non viene spostato nulla in caso in cui var=false.

Se var=true, si imposta l'offset (indicare la differenza rispetto ad un valore di riferimento, le coordinate della posizione al click del mouse), il nCenter , dove vengono impostate le coordinate del cursore durante il trascinamento (e.getX(), e.getY()).

public void mouseReleased(MouseEvent e) // metodo che descrive cio' che succede al rilascio del tasto del mouse.

Al rilascio del mouse, se il rettangolo contenente l'immagine e' stato posizionato nella sezione di griglia corretta si aumenta il punteggio di 10 punti. Raggiunto un certo numero di punti, e' possibile passare al livello successivo, tramite un oggetto di tipo Livello.

public boolean getGiustaPosizione() // metodo che controlla se un rettangolo contenente l'immagine viene posizionato nella sezione giusta della griglia. Ci riconduciamo al caso in cui vi e' un solo rettangolo da spostare (rettangolo_in_movimento) all'interno di una sola sezione di una griglia 1x1 (griglia_fissa).

boolean posizione_giusta=false;

```

    for(int i=0; i<grid.length; i++)//controlliamo che spostando
l'immagine il quadrato contenente l'immagine sia posizionato nella giusta
sezione di griglia
    {

```

```

        this.griglia_fissa=this.grid[i]; // per ogni i,
associamo a griglia_fissa un rettangolo (sezione di griglia) dell'array
grid(array che contiene i rettangoli che formano la griglia)

```

```

        if(this.griglia_fissa.contains(this.rettangolo_in_movimento)) //si
verifica se la griglia_fissa contiene il rettangolo in movimento.

```

```

        {
            if(this.rettangolo_in_movimento==this.rett[i]) //
si verifica che il rettangolo in movimento e' uguale al rettangolo
corrispondente alla giusta sezione della griglia
            {
                posizione_giusta=true;
                index=i;
                break;
            }
            else
            {
                posizione_giusta=false;
                break;
            }
        }
    }

    return posizione_giusta;

```

public boolean mossaNonValida() verifica se una mossa sia valida o meno, per evitare che il giocatore possa barare. Si controlla che la posizione al click del mouse sia o meno all'interno di una delle sezioni costituenti la griglia. In caso sia affermativo, che negativo, non vengono aumentati punti.

```

boolean mossa_non_valida=false;
        for(int i=0; i<grid.length; i++)//controlliamo che
spostando l'immagine il quadrato contenente l'immagine sia posizionato nella
giusta sezione di griglia
        {
            this.griglia_fissa=this.grid[i]; // per ogni i,
associamo a griglia_fissa un rettangolo (sezione di griglia) dell'array
grid(array che contiene i rettangoli che formano la griglia)
            if(this.griglia_fissa.contains(pos)) //si
verifica se la griglia_fissa contiene il rettangolo in movimento.

            {

                mossa_non_valida=true;

                break;
            }
            else
            {
                mossa_non_valida=false;
            }

        }
        return mossa_non_valida;

public int getPunti() restituisce il punteggio.

```

```

    }

```

PRESSPLAYACTION implements ActionListener

Nota: Permette di descrivere le azioni che saranno svolte quando verra' premuto il tasto play. In particolare crea la schermata relativa al primo livello.

```
{
    private JFrame sp; //schermata play.
    private Importa_Audio audio;

    Costruttore(JFrame, Importa_Audio)
    {
        Vengono inizializzati le variabili di istanza.
    }
```

Metodi:

```
public void actionPerformed(ActionEvent)
```

crea il primo livello. In particolare,

crea una griglia scheletro, determinando le coordinate e dimensioni del primo rettangolo della griglia in posizione 0,0 e il numero di righe e colonne della griglia (tramite la classe GrigliaFtoComponent).

Si estrapolano gli elementi di tale griglia, creando 4 variabili diverse, una per ogni elemento della griglia. I rettangoli ora creati vengono disposti all'interno di un array chiamato grid_array, per differenziarlo dal rect_array.

Successivamente, tramite la classe GrigliaFotoComponent , viene creata la griglia che andra' a contenere le immagini, dalla quale si estrapolano 4 rettangoli che verranno inseriti nell'array rect_array.

Si importano le immagini relative al primo livello e le si inseriscono in un array di immagini.

Si crea il frame.

E si carica tutto sul pannello relativo alla schermata di gioco, costruito dalla classe AssegnazioneFotoComponent, che disegnera' griglia e immagini, timer e JButton "ViewImage" e permettera' il movimento alle immagini, aggiornando i punteggi raggiunti dal giocatore e la possibilita' di passare al livello successivo.

Si rende invisibile la schermata relativa alla schermata di inizio, la schermata play.

```
public int getRandomColorValue() restituisce un valore random tra 0 e 255.
```

```
    public Color getRandomColorize() restituisce un colore random RGB,  
prendendo tre valori generati dal metodo getRandomColorValue().
```

```
}
```

Punteggio

Timer

VIEW IMAGE



SECONDO LIVELLO extends JFrame

Nota: Permette di contenere e creare i dati per caricare il secondo livello del gioco.

```
{
    private JFrame sl; // schermata di un livello (uno)
    private CardViewer cv; //CardViewer contenente l'immagine da
visualizzare.
    private Importa_Audio audio;
```

Costruttore (Jframe, CardViewer, Importa_Audio)

```
{

    Vengono inizializzati le variabili di istanza.

    Si crea il secondo livello. In particolare,

    crea una griglia scheletro, determinando le coordinate e dimensioni del
    primo rettangolo della griglia in posizione 0,0 e il numero di righe e
    colonne della griglia (tramite la classe GrigliaFtoComponent).

    Si estrapolano gli elementi di tale griglia, creando 9 variabili diverse, una
    per ogni elemento della griglia. I rettangoli ora creati vengono disposti
    all'interno di un array chiamato grid_array, per differenziarlo dal
    rect_array.

    Successivamente, tramite la classe GrigliaFotoComponent , viene creata la
    griglia che andra' a contenere le immagini, dalla quale si estrapolano 9
    rettangoli che verranno inseriti nell'array rect_array.

    Si importano le immagini relative al secondo livello e le si inseriscono in
    un array di immagini.

    Si crea il frame.

    E si carica tutto sul pannello relativo alla schermata di gioco, costruito
    dalla classe AssegnazioneFotoComponent, che disegnera' griglia e immagini,
    timer e JButton "ViewImage" e permettera' il movimento alle immagini,
    aggiornando i punteggi raggiunti dal giocatore e la possibilita' di passare
    al livello successivo.
```

```
}
```

Metodi:

```
public void actionPerformed(ActionEvent)
```

```
public int getRandomColorValue() restituisce un valore random tra 0 e 255.
```

```
public Color getRandomColorize() restituisce un colore random RGB,  
prendendo tre valori generati dal metodo getRandomColorValue().
```

```
public void closeW() chiude la schermata del livello precedente (livello1).
```

```
public void closeCardViewer() Chiude il CardViewer a livello terminato  
nel caso in cui esso rimane aperto.
```

```
}
```

Punteggio

Timer

VIEW IMAGE



TERZOLIVELLO extends JFrame

Nota: Permette di creare e contenere i dati per caricare il terzo livello del gioco.

```
{
    private JFrame sl; // schermata livello precedente (livello 2)
    private CardViewer cv; //CardViewer contenente l'immagine da
visualizzare.
    private Importa_Audio audio;

Costruttore (Jframe, CardViewer, Importa_Audio)
{
    Vengono inizializzati le variabili di istanza.

    Si crea il terzo livello. In particolare,

    crea una griglia scheletro, determinando le coordinate e dimensioni del
    primo rettangolo della griglia in posizione 0,0 e il numero di righe e
    colonne della griglia (tramite la classe GrigliaFtoComponent).

    Si estrapolano gli elementi di tale griglia, creando 16 variabili
    diverse, una per ogni elemento della griglia. I rettangoli ora creati vengono
    disposti all'interno di un array chiamato grid_array, per differenziarlo dal
    rect_array.

    Successivamente, tramite la classe GrigliaFotoComponent , viene creata la
    griglia che andra' a contenere le immagini, dalla quale si estrapolano 16
    rettangoli che verranno inseriti nell'array rect_array.

    Si importano le immagini relative al primo livello e le si inseriscono in un
    array di immagini.

    Si crea il frame.

    E si carica tutto sul pannello relativo alla schermata di gioco, costruito
    dalla classe AssegnazioneFotoComponent, che disegnera' griglia e immagini,
    timer e JButton "ViewImage" e permettera' il movimento alle immagini,
    aggiornando i punteggi raggiunti dal giocatore e la possibilita' di passare
    al livello successivo.

}
```

Metodi:

```
public void actionPerformed(ActionEvent)
```

```
public int getRandomColorValue() restituisce un valore random tra 0 e 255.
```

```
public Color getRandomColorize() restituisce un colore random RGB,  
prendendo tre valori generati dal metodo getRandomColorValue().
```

```
public void closeW() /chiude la schermata del livello precedente (livello 2).
```

```
public void closeCardViewer() Chiude il CardViewer a livello terminato  
nel caso in cui esso rimane aperto.
```

```
}
```

Punteggio

Timer

VIEW IMAGE



CONGRATULATION extends JFrame

Nota: Permette di creare la schermata per la chiusura del gioco, dove la cpu si congratula con il giocatore per aver completato i livelli.

```
{
    private JFrame sl;    // schermata di un livello precedente (livello 3).
    private CardViewer cv; //CardViewer contenente l'immagine da
visualizzare.
    private Importa_Audio audio;

    Costruttore(Jframe, CardViewer,Importa_Audio)
{
    Vengono inizializzati le variabili di istanza.

    Crea schermata frame finale.

    crea label ospitante foto dei fuochi d'artificio.

    Si stoppa la nota audio che e' stata riprodotta in loop durante
tutta la partita e si avvia una nuova nota audio, relativa alla vittoria e
alla fine del gioco.

    Mostra messaggio e chiudi il gioco.
        JOptionPane.showMessageDialog(null, " CONGRATULATIONS!");
        System.exit(0);

}
```

Metodi:

```
public void closeW()    chiude la schermata del livello precedente (livello
3).

public void closeCardViewer()    Chiude il CardViewer a livello terminato
nel caso in cui esso rimane aperto.

}
```



TUTORIALACTION implements ActionListener

Nota: Permette di mostrare il tutorial al click sul bottone "Tutorial".

```
{
    private int i=0; // inutile al frame , serve sl per inizializzare e
    costruire un oggetto di tipo TutorialAction.
```

Costruttore()

```
{
    Vengono inizializzati le variabili di istanza.

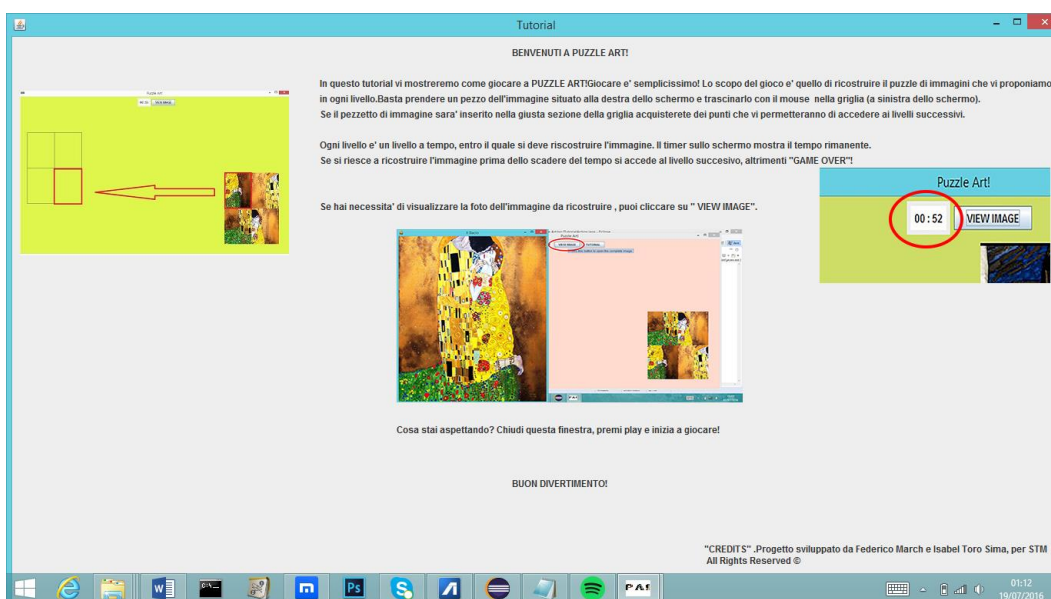
    Si crea frame.

    Il frame conterra' label di testo e label contenenti immagini per
    sviluppare il tutorial e le regole e modalita' di gioco.

}
```

Metodi:

Non sono presenti metodi.



PUZZLEART (contenente il Main)

Nota: Permette di creare un frame relativo alla schermata "home", "intro" e avvia il gioco. E' la classe in cui e' contenuto il main.

```
{  
  
    Public static void main(String[] args)  
  
    {  
  
        Si crea il frame e si imposta l'icona del programma.  
  
        Si imposta il pannello di background contente la label (alla quale sara'  
        associata l'immagine che fara' da sfondo per la schermata play), i JButton  
        "Play" e "Tutorial".  
  
        Il tutto viene aggiunto al frame.  
  
        Si aggiunge, tramite la classe Importa_Audio il file audio  
        "PuzzleArt_theme", che verra' riprodotto in loop durante tutta la partita.  
  
        Vengono create e associate le relative azioni, rispettivamente, per i  
        JButton "Play" e "Tutorial".  
  
    }  
  
}
```



PLAY

TUTORIAL

CREDITS .Progetto sviluppato da Federico March e Isabel Toro Sima, per STM

All Rights Reserved ©

