Tetris

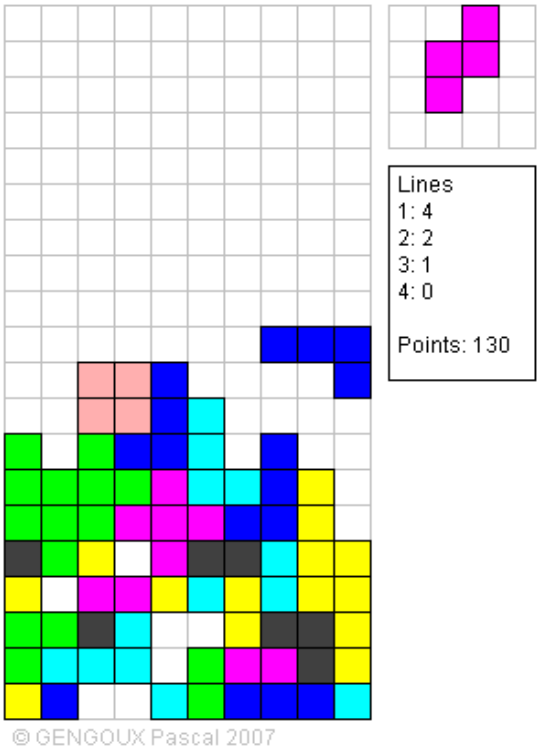
Jeu en java

Documentation

BAIG0607

Gengoux Pascal

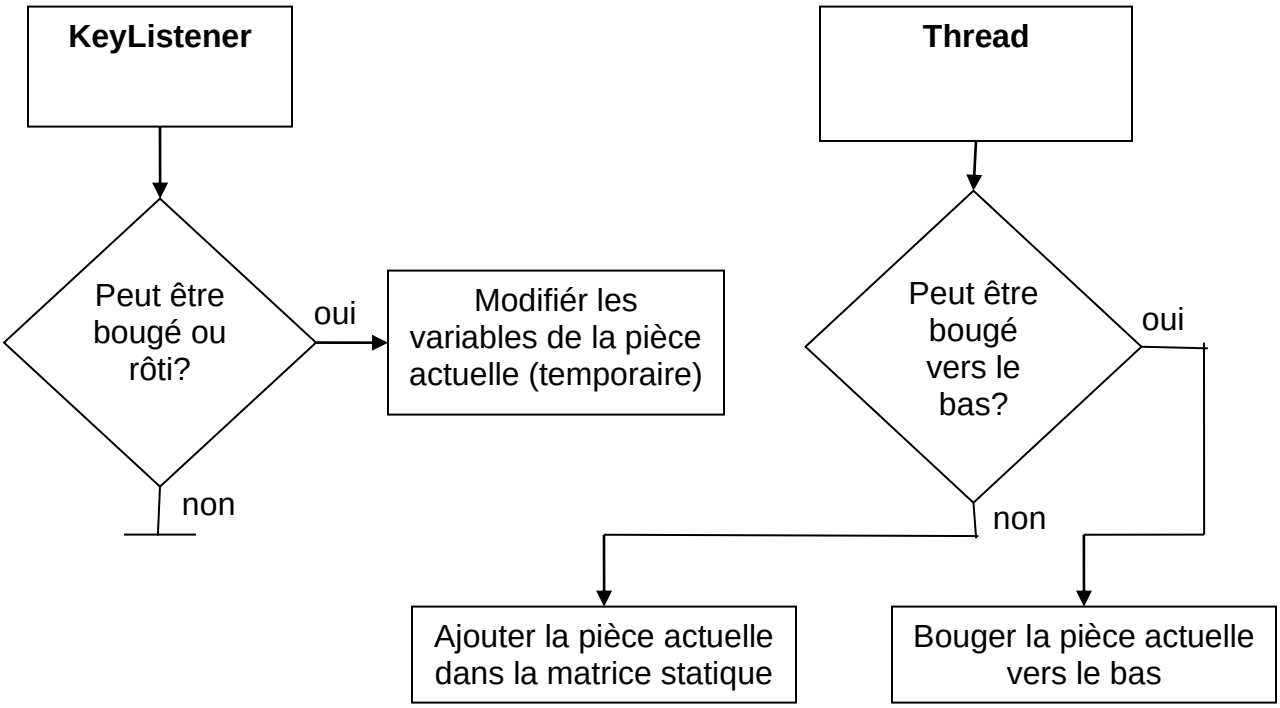
Introduction



Mon but est de réaliser le jeu « Tetris ». Ce jeu a comme but principal d'empiler des morceaux de sorte que l'un s'installe sur l'autre sans trous le mieux. En cas qu'une ligne sera remplie entièrement, les lignes en haut seront décalées en bas. Pendant le jeu la vitesse sera augmentée chaque Nième ligne complète. Le joueur gagne des points pour chaque ligne complétée. Détruire plusieurs lignes à la fois sera récompensé plus qu'une ligne après l'autre. La pièce suivante sera également affichée pour simplifier le jeu un peu. Chaque pièce a sa couleur spécifique et il peut être tourné vers gauche. C'est aussi possible de pauser le jeu. Avec les curseurs le jeu sera contrôlable. Un contrôle spécial permet de laisser tomber la pièce actuelle directe.
Le jeu est plus au moins comme l'image à droite. Un autre point intéressant du jeu est le highscore qui permet de comparer entre les joueurs leurs capacités ce qui motive à le rejouer.

Approche

Pour réaliser ce jeu j'ai choisi un la langue de programmation java avec une classe applet parce qu'on peut le jouer presque n'importe où sans le recompiler. J'utilise une matrice pour stocker toutes les pièces statiques et une autre matrice pour stocker toutes les pièces avec toutes les possibilités de rotation. Un thread est créé pour faire le jeu dynamique et qui a un certain délai pour chaque pas dans le jeu. Dans ce thread appart de variables (position en x, position en y, rotation actuelle, n° pièce actuelle), une pièce temporaire (pièce actuelle) est dessiné et peut être bougé ou rôti si c'est possible.



Autres détails techniques

- Pour éviter le clignotement du applet j'utilise un truc qui s'apple double pouffant. Avec cette technique l'image est premièrement dessiné sur un image invisible et ensuite sera affiché sur l'écran entièrement. J'ai utilisé le code de la méthode void update(Graphics g) du site : http://javacooperation.gmxhome.de/BildschirmflackernDeu.html
- La matrice des pièces est composée de 4 Dimension. Où 1 dimension est pour la sorte de la pièce, 1 dimension pour la rotation, et 2 dimensions pour la pièce elle-même. C'était aussi possible d'utiliser seulement une matrice à 3 dimensions mais c'est plus coûteux à calculer chaque rotation.

Code Source

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.io.*;
import java.net.HttpURLConnection;
import java.sql.*;
import java.net.*;
import java.util.Random;

public class Tetris extends java.applet.Applet implements KeyListener, Runnable,
MouseListener, ActionListener {
  /* Buttons & TextFields */
  Button btnStart;
  Button btnSubmit;
  TextField edtName;
  /* Constants */
  final int xx = 10;  // Number of vertical cases
  final int yy = 20;  // Number of horizonal cases
  final int nn = 20;  // Width & Height of one case (square)
  int rx,ry,rxmax,rymax;
  /* Matrixes */
  boolean[][] tetris = new boolean[xx][yy];  // Tetris Matrix
  int[][]col = new int[xx][yy];              // Support Tetris Matrix with Colors
  // Pieces Matrix
  boolean[][][][] pieces =
{{{{false,false,true,false},{false,false,true,false},{false,false,true,false},{fals
e,false,true,false}},{{false,false,false,false},{true,true,true,true},
{false,false,false,false},{false,false,false,false}},{{false,false,true,false},{fal
se,false,true,false},{false,false,true,false},{false,false,true,false}},{{false,fal
se,false,false},{true,true,true,true},{false,false,false,false},{false,false,false,
false}}},{{{false,false,true,false},{false,false,true,true},{false,false,true,false
},{false,false,false,false}},{{false,false,true,false},{false,true,true,true},{fals
e,false,false,false},{false,false,false,false}},{{false,false,true,false},{false,tr
ue,true,false},{false,false,true,false},{false,false,false,false}},{{false,false,fa
lse,false},{false,true,true,true},{false,false,true,false},{false,false,false,false
}}},{{{false,false,true,false},{false,false,true,false},{false,false,true,true},{fa
lse,false,false,false}},{{false,false,false,false},{false,false,false,true},{false,
true,true,true},{false,false,false,false}},{{false,false,true,true},{false,false,fa
lse,true},{false,false,false,true},{false,false,false,false}},{{false,false,false,f
alse},{false,true,true,true},{false,true,false,false},{false,false,false,false}}},{
{{false,false,true,true},{false,false,true,false},{false,false,true,false},{false,f
alse,false,false}},{{false,false,false,false},{false,true,true,true},{false,false,f
alse,true},{false,false,false,false}},{{false,false,false,true},{false,false,false,
true},{false,false,true,true},{false,false,false,false}},{{false,false,false,false}
,{false,true,false,false},{false,true,true,true},{false,false,false,false}}},{{{fal
se,false,true,false},{false,true,true,false},{false,true,false,false},{false,false,
false,false}},{{false,false,false,false},{true,true,false,false},{false,true,true,f
alse},{false,false,false,false}},{{false,false,true,false},{false,true,true,false},
{false,true,false,false},{false,false,false,false}},{{false,false,false,false},{tru
e,true,false,false},{false,true,true,false},{false,false,false,false}}},{{{false,fa
lse,false,false},{false,true,true,false},{true,true,false,false},{false,false,false
,false}},{{false,true,false,false},{false,true,true,false},{false,false,true,false}
,{false,false,false,false}},{{false,false,false,false},{false,true,true,false},{tru
e,true,false,false},{false,false,false,false}},{{false,true,false,false},{false,tru
e,true,false},{false,false,true,false},{false,false,false,false}}},{{{false,false,f
alse,false},{false,false,true,true},{false,false,true,true},{false,false,false,fals
e}},{{false,false,false,false},{false,false,true,true},{false,false,true,true},{fal
se,false,false,false}},{{false,false,false,false},{false,false,true,true},{false,fa
lse,true,true},{false,false,false,false}},{{false,false,false,false},{false,false,t
```

```
rue,true},{false,false,true,true},{false,false,false,false}}};
  int[] points = new int[4];                  // Points Matrix

  Thread t;
  /* Valeurs relevants au jeu */
  // Piece derniere
  int prev_p    = 0;
  // Piece après
  int next_p    = 0;  //= (int)(Math.random()*1000000)%7;
  // Piece actuelle
  int active_p  = 0;  //= (int)(Math.random()*1000000)%7;
  // Roation actuelle
  int rotate    = 0;
  // Position en x actuelle
  int ax        = xx/2-max_x(rotate)/2;
  // Position en y actuelle
  int ay        = 0;//0-min_y(rotate+1);
  // Delai de départ en ms
  int ival      = 500;
  int vval      = 0;

  // L'indice maximale en x de la piece actuelle
  int maxx = max_x(rotate);
  // L'indice maximale en y de la piece actuelle
  int maxy = max_y(rotate);
  // L'indice minimale en x de la piece actuelle
  int minx = min_x(rotate);
  // L'indice minimale en y de la piece actuelle
  int miny = min_y(rotate);

  // Le jeu est en cours??
  boolean start    = false;
  // Le jeu s'est terminé??
  boolean gameover = false;
  // Le jeu est lancé pour la premiere fois??
  boolean firstrun = true;
  // Le processeur est-il occupé à calculer qqch.??
  boolean busy     = false;
  // Le jeu est en mode "pause"
  boolean pause    = false;

  // Un image et un graphique suplémentaire pour éviter un "blink"
  // en rédessinant l'image
  private Image dbImage;
  private Graphics dbg;

  private Random MyRandom;

  public void mouseEntered(MouseEvent me) {}
  public void mousePressed(MouseEvent me) {}
  public void mouseReleased(MouseEvent me) {}
  public void mouseExited(MouseEvent me) {}
  public void keyReleased(KeyEvent e) {}

  public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() == btnStart) {
      btnStart.setVisible(false);
      btnSubmit.setVisible(false);
      edtName.setVisible(false);
      requestFocus();
      // Seulement en cas que le jeu se lance la premiere fois on peut
      // lancer le thread t
```

```
      init_game();
      if(firstrun) {
        // lancer le thread
        t.start();
        firstrun = false;
      }
      start=true;
      repaint();
    }else if(evt.getSource() == btnSubmit) {
       int p=(points[0]*10+points[1]*12*2+points[2]*14*3+points[3]*16*4);
       String pp;
       pp = String.valueOf(p);
       try {
        URL home = getCodeBase();
        String                                    url                                    =
home.toString()+"index.php?xx=100&flag=2&yx=2424&xc="+edtName.getText()+"&xy="+pp;
        getAppletContext().showDocument(new URL(url));
        } catch(Exception x) {}
    }
  }
  /**
   * Dessiner les lignes du jeu ("grid")
   */
  public void draw_grid(Graphics g) {
    int i;
    set_color(0,g);
    for(i=0;i<=xx;i++)
      g.drawLine(i*nn,0,i*nn,yy*nn);
    for(i=0;i<=yy;i++)
      g.drawLine(0,i*nn,xx*nn,i*nn);
  }

  /**
   * Dessiner une piece à la position x,y
   */
  public void draw_piece(int x, int y, Graphics g) {
    int k,m;
    set_color(active_p+1,g);
    for(k=0;k<4;k++)
      for(m=0;m<4;m++)
        if(pieces[active_p][rotate][k][m])
          draw_rect((x+k),(y+m),g);
  }

 /**
  * Dessiner tout le tetris(tous les pieces collantes)
  */
  public void draw_tetris(Graphics g) {
    int i,j;
    for(i=0;i<xx;i++)
      for(j=0;j<yy;j++)
        if(tetris[i][j]) {
          set_color(col[i][j],g);
          draw_rect(i,j,g);
        }
  }

 /**
  * Dessiner des lignes(grid) et la piece qui suit
  */
  public void draw_next(Graphics g) {
    int i, offsetx=nn*xx+10,offsety=0;
```

```
    // Dessiner les lignes(grid)
    set_color(0,g);
    for(i=0;i<=4;i++)
      g.drawLine(offsetx+i*nn,offsety+0,offsetx+i*nn,offsety+4*nn);
    for(i=0;i<=4;i++)
      g.drawLine(offsetx,offsety+i*nn,offsetx+4*nn,offsety+nn*i);
    int k,m;
    // Dessiner la piece suivante
    set_color(next_p+1,g);
    for(k=0;k<4;k++)
      for(m=0;m<4;m++)
        if(pieces[next_p][0][k][m]) {
          set_color(next_p+1,g);
          g.fillRect(offsetx+k*nn,offsety+m*nn,nn,nn);
          g.setColor(Color.black);
          g.drawRect(offsetx+k*nn,offsety+m*nn,nn,nn);
        }
  }

 /**
  * Dessiner les points actuelles
  */
  public void draw_points(Graphics g) {
    int offsety=95;
    int offsetx=5;
    g.setColor(Color.black);
    g.drawRect(xx*nn+5+offsetx,offsety-5,80,120);
    g.drawString("Lines",xx*nn+10+offsetx,10+offsety);
    g.drawString("1: "+points[0],xx*nn+10+offsetx,25+offsety);
    g.drawString("2: "+points[1],xx*nn+10+offsetx,40+offsety);
    g.drawString("3: "+points[2],xx*nn+10+offsetx,55+offsety);
    g.drawString("4: "+points[3],xx*nn+10+offsetx,70+offsety);
    g.setColor(Color.black);
    g.drawString("Points:                              "                              +
(points[0]*10+points[1]*12*2+points[2]*14*3+points[3]*16*4),xx*nn+10+offsetx,100+of
fsety);
  }


 /**
  * changer la couleur actuelle sur le graphique
  */
  public void set_color(int color, Graphics g) {
    switch(color) {
      case 0:
        g.setColor(Color.lightGray);
        break;
      case 1:
        g.setColor(Color.yellow);
        break;
      case 2:
        g.setColor(Color.green);
        break;
      case 3:
        g.setColor(Color.blue);
        break;
      case 4:
        g.setColor(Color.cyan);
        break;
      case 5:
        g.setColor(Color.darkGray);
        break;
```

```
      case 6:
        g.setColor(Color.magenta);
        break;
      case 7:
        g.setColor(Color.pink);
        break;
    }
  }

 /**
  * Procedure qui initialise le applet et initialise toutes les varibles du jeu
  */
  public void init() {
    // Placement et Creation des Buttons et de l'inputbox
    setLayout(null);
    btnStart = new Button("Start");
    btnStart.reshape(xx*nn/2-35,yy*nn/2+30,70,24);
    btnStart.addActionListener(this);
    btnSubmit = new Button("Submit");
    btnSubmit.addActionListener(this);
    btnSubmit.reshape(xx*nn/2-40,yy*nn/2,80,24);
    edtName = new TextField("Your Name",15);
    edtName.reshape(xx*nn/2-35,yy*nn/2-30,70,24);
    add(edtName);
    add(btnStart);
    add(btnSubmit);
    // Mettre invisible
    btnSubmit.setVisible(false);
    edtName.setVisible(false);
    // Creation du thread
    t = new Thread(this);
    // Ajout des listeneurs
    addKeyListener(this);
    addMouseListener(this);
    busy = false;
    pause = false;
    MyRandom = new Random();
  }

 /**
  * Initialiser le jeu
  */
  public void init_game() {
    active_p  = MyRandom.nextInt(7);
    next_p    = MyRandom.nextInt(7);
    prev_p    = active_p;
    rotate    = 0;
    ax        = xx/2-max_x(rotate)/2;
    ay        = 0;//-min_y(rotate+1);
    start     = false;
    ival      = 500;
    vval      = 0;
    busy = false;
    pause = false;
    gameover  = false;
    maxx = max_x(rotate);
    maxy = max_y(rotate);
    minx = min_x(rotate);
    miny = min_y(rotate);
    int i,j;
    for(i=0;i<4;i++)
      points[i]=0;
```

```
    for(i=0;i<xx;i++) {
      for(j=0;j<yy;j++) {
        col[i][j]=0;
        tetris[i][j]=false;
      }
    }
  }

  public void mouseClicked(MouseEvent me) { }

 /**
  * Procedure qui est appelé par le thread
  */
  public void run() {
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
    while (true) {
      while(start) {
        repaint();
        if(can_down())
          ay++;
        else {
          nextOne();
          if(!can_down()) {
            start=false;
            gameover=true;
            repaint();
          }
        }
        try {
          Thread.sleep(ival);
        } catch (InterruptedException ex) { }
        while(pause || busy) {  }
      }
    }
  }

 /**
  * Procedure qui est appelé quand une piece est sur le fond
  * et détermine la prochaine piece
  */
  private void nextOne() {
    int i,j;
    /* Mettre la piece temporaire(active) à la matrice tetris */
    for(i=0;i<4;i++)
      for(j=0;j<4;j++)
        if(pieces[active_p][rotate][i][j]) {
          tetris[i+ax][j+ay]=pieces[active_p][rotate][i][j];
          col[i+ax][j+ay]=active_p+1;
        }
    /* Déterminer les points */
    int lines=seek_full_lines();
    if(lines>0)
      points[lines-1]++;
    if(count_lines() / 10 > vval) {
      vval ++;
      if(ival > 175)
        ival -= 75;
    }
    /* Déterminer la prochaine piece */
    rotate=0;
    prev_p   = active_p;
    active_p = next_p;
```

```
    next_p    = MyRandom.nextInt(7);
    while(prev_p == active_p && next_p == active_p) {
      next_p    = MyRandom.nextInt(7);
    }
    ay=0;//-min_y(rotate);
    ax=xx/2-max_x(rotate)/2;
    maxx = max_x(rotate);
    maxy = max_y(rotate);
    minx = min_x(rotate);
    miny = min_y(rotate);
  }

  public void keyTyped(KeyEvent e) { }

  public void keyPressed(KeyEvent e) {
    busy = true;
    //estr = " keyPressed " + e.getKeyChar();
    if(e.getKeyCode() == KeyEvent.VK_LEFT && !pause)
      if(can_move(-1))
        ax--;
    if(e.getKeyCode()   ==   KeyEvent.VK_ENTER   &&   !pause   ||   e.getKeyCode()   ==
KeyEvent.VK_SPACE) {
      while(can_down())
        ay++;
      nextOne();
    }
    if(e.getKeyCode() == KeyEvent.VK_DOWN && !pause)
      if(can_down())
        ay++;
    if(e.getKeyCode() == KeyEvent.VK_RIGHT && !pause)
      if(can_move(1))
        ax++;
    if(e.getKeyCode() == KeyEvent.VK_P) {
      pause = !pause;

    }
    if(e.getKeyCode() == KeyEvent.VK_UP && !pause)
      if(can_rotate(1)) {
      if(rotate==3)
        rotate=0;
      else
        rotate++;
      }
    maxx = max_x(rotate);
    maxy = max_y(rotate);
    minx = min_x(rotate);
    miny = min_y(rotate);
    busy = false;
    repaint();
  }

  /**
   * Fonction qui détermine si la piece active peut être roté
   */
  public boolean can_rotate(int offset) {
    int rot=rotate+offset;
    if(rot==-1)
      rotate=3;
    else
      if(rot==4)
        rot=0;
    if((ax+max_x(rot) > xx-1) || (ax+min_x(rot) < 0) || ((ay+max_y(rot) >= yy)) ||
```

```
(ay+min_y(rot) < 0) )
      return false;
    else {
      int i, j;
      for(j=min_y(rot);j<max_y(rot)+1;j++)
        for(i=min_x(rot);i<max_x(rot)+1;i++)
          if(tetris[ax+i][ay+j] && pieces[active_p][rot][i][j])
            return false;
      return true;
    }
  }

  /**
   * Fonction qui détermine si la piece active bougé droit ou gauche
   */
  public boolean can_move(int offset) {
    if((offset==1)&&(ax+maxx+1 > xx-1))
      return false;
    else
      if((offset==-1)&&(ax+minx-1 < 0))
        return false;
      else {
      int i, j;
      for(j=miny;j<maxy+1;j++) {
        for(i=minx;i<maxx+1;i++)
          if(tetris[ax+i+offset][ay+j] && pieces[active_p][rotate][i][j])
            return false;
      }
      return true;
      }
  }

  /**
   * Fonction qui détermine si la piece active peut être bougé vers le bas
   */
  public boolean can_down() {
    int i=0, j=0;
    if(ay >= yy-maxy-1)
      return false;
    for(j=miny;j<maxy+1;j++)
      for(i=minx;i<maxx+1;i++)
        if(tetris[ax+i][ay+j+1] && pieces[active_p][rotate][i][j])
          return false;
    return true;
  }

  /**
   * Fonction qui détermine le nombres des lignes complètes
   */
  public int count_lines() {
    int c=0,i;
    for(i=0;i<4;i++)
      c+=points[i];
    return c;
  }

  /**
   * Fonction qui détermine le max en x de la piece active d'une rotation par
paramètre
   */
  public int max_x(int rot) {
    int i,j,max=0;
```

```
    for(i=0;i<4;i++)
      for(j=0;j<4;j++)
        if(pieces[active_p][rot][i][j] && (i > max))
          max=i;
    return max;
  }

  /**
   * Fonction qui détermine le min en x de la piece active d'une rotation par
paramètre
   */
  public int min_x(int rot) {
    int i,j,min=3;
    for(i=0;i<4;i++)
      for(j=0;j<4;j++)
        if(pieces[active_p][rot][i][j] && (i < min))
          min=i;
    return min;
  }

  /**
   * Fonction qui détermine le max en y de la piece active d'une rotation par
paramètre
   */
  public int max_y(int rot) {
    int i,j, max=0;
    for(j=0;j<4;j++) {
      for(i=0;i<4;i++) {
        if(pieces[active_p][rot][i][j] && (j > max))
          max=j;
      }
    }
    return max;
  }

  /**
   * Fonction qui détermine le min en y de la piece active d'une rotation par
paramètre
   */
  public int min_y(int rot) {
    int i,j, min=3;
    for(j=3;j>=0;j--) {
      for(i=0;i<4;i++) {
        if(pieces[active_p][rot][i][j] && (j < min))
          min=j;
      }
    }
    return min;
  }

  /**
   * Procédure qui élimine une ligne compète passé par paramètre
   */
  public void del_line(int y) {
    int i,j;
    for(i=y;i>1;i--)
      for(j=0;j<xx;j++) {
      tetris[j][i]=tetris[j][i-1];
      col[j][i]=col[j][i-1];
      }
  }
```

```
/**
 * Fonction qui compte le nombre des lignes compète et les élémine
 */
public int seek_full_lines() {
  boolean line;
  int i, j, count=0;
  busy = true;
  for(i=0;i<yy;i++) {
    line=true;
    j=0;
    while(line && (j < xx)) {
      if(!tetris[j][i])
        line=false;
      j++;
    }
    if(line &&(j > 1)) {
      del_line(i);
      count++;
    }
  }
  busy = false;
  return count;
}

/**
 * Procédure qui diminue le clignotage
 */
public void update(Graphics g) {
  if (dbImage == null) {
    dbImage = createImage(this.getSize().width, this.getSize().height);
    dbg = dbImage.getGraphics();
  }
  dbg.setColor(getBackground());
  dbg.fillRect(0, 0, this.getSize().width, this.getSize().height);
  dbg.setColor(getForeground());
  paint(dbg);
  g.drawImage(dbImage, 0, 0, this);
}

/**
 * Procédure qui dessine un rectangle sur une position spécéfique
 */
public void draw_rect(int x, int y, Graphics g) {
  Color acol=g.getColor();
  g.fillRect((x)*nn,(y)*nn,nn,nn);
  g.setColor(Color.black);
  g.drawRect((x)*nn,(y)*nn,nn,nn);
  g.setColor(acol);
}

/**
 * Procédure qui dessine tout le jeu noir(En pause)
 */
public void drawBlack(Graphics g) {
  int i,j;
  g.setColor(Color.black);
  for(i=0;i<xx;i++)
    for(j=0;j<yy;j++)
      draw_rect(i,j,g);
}

/**
```

```
   * Procédure qui dessine le jeu complet
   */
  public void paint(Graphics g) {
    if(!start) {
      if(!gameover) {
        draw_grid(g);
        draw_piece(ax,ay,g);
        g.setColor(Color.black);
        g.drawString("Arrows to move!",xx*nn+5,150);
        g.drawString("<<Enter>>  to fall!",xx*nn+5,170);
        g.drawString("<<P>> to Pause!",xx*nn+5,190);
        draw_next(g);
      }else {
        g.setColor(Color.black);
        g.drawString("Game Over!!!",60,100);
        g.drawString("Your                Score:                "                +
(points[0]*10+points[1]*12*2+points[2]*14*3+points[3]*16*4),60,120);
        btnSubmit.setVisible(true);
        btnStart.setVisible(true);
        edtName.setVisible(true);
      }
      set_color(0,g);
      g.drawString("© GENGOUX Pascal 2007",5,nn*yy+15);
    } else {

      draw_grid(g);
      if(pause)
        drawBlack(g);
      else {
        draw_piece(ax,ay,g);
        draw_tetris(g);
      }
      set_color(0,g);
      g.drawString("© GENGOUX Pascal 2007",5,nn*yy+15);
      draw_points(g);
      draw_next(g);
    }
  }
}
```