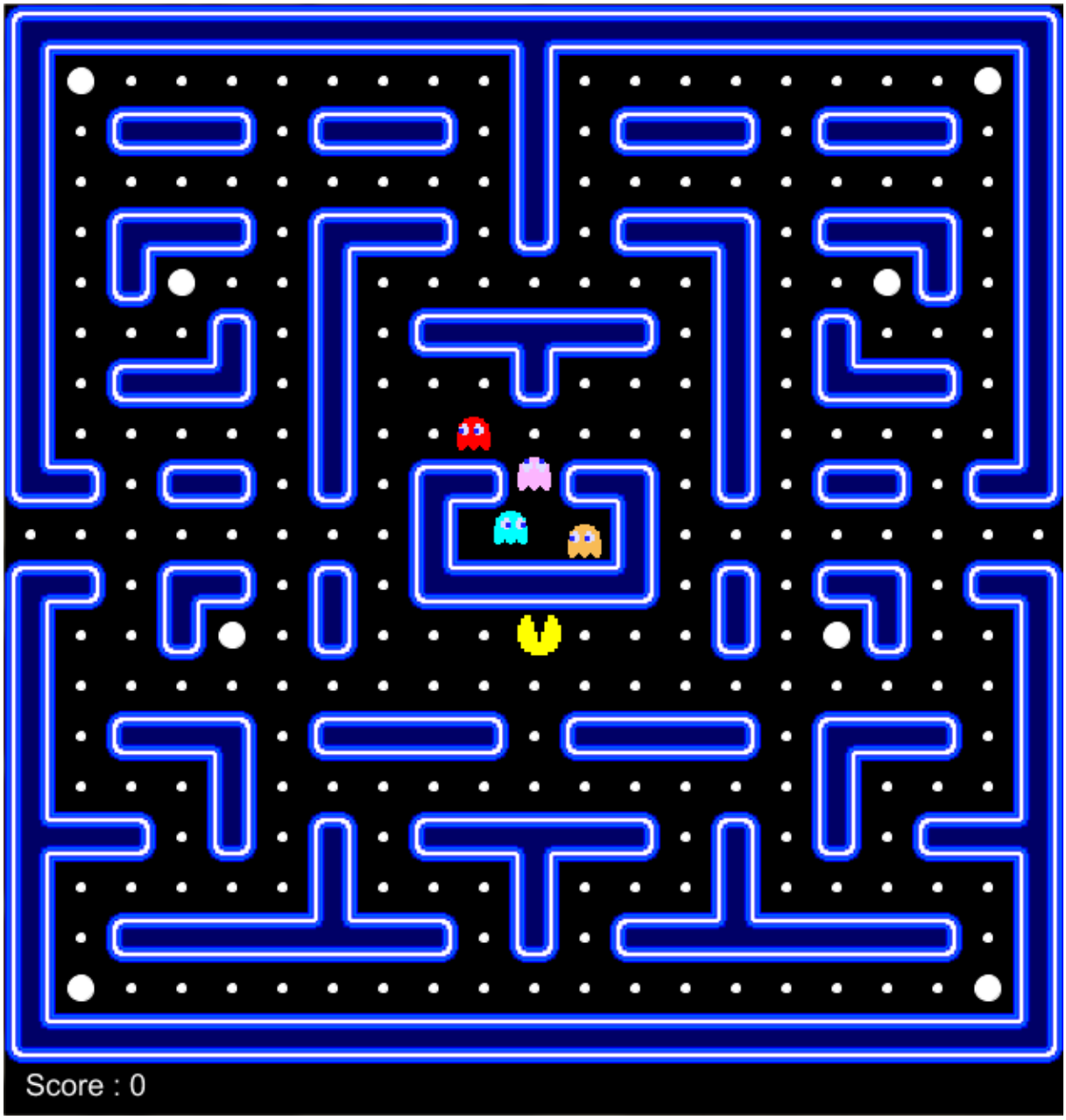
Morgane

KELLER

TS3

Dossier ISN : Jeu de Pacman



En équipe avec Mathilde DUTRIEUX

Sommaire

I. Qu'est-ce qu'un jeu pacman ?

C'est un jeu dans lequel le personnage joué par le joueur (Pacman) doit se déplacer dans un labyrinthe pour ramasser tous les points, sans se faire attraper par les fantômes. Un point, appelé Pacgomme rapporte 10 points. Il y a toutefois quelques gros points dans le jeu, ceux-ci sont appelés Super Pacgomme et rapportent 50 points, elles permettent de geler les fantômes durant un certain temps, les ennemis sont alors plus lents, et surtout : pacman peut les manger ; ils reviennent alors à leur point de départ, au centre du jeu.

Le joueur a la possibilité de passer d'un coté à l'autre du labyrinthe en un instant en passant par les portes laissées sur la droite et sur la gauche.

Le Pacman a 3 vies.

En cas de perte :

Le personnage fait face à un écran indiquant « Game over » et un bouton proposant de rejouer une partie. Le joueur fait alors face à la même situation qu'au tout début.

En cas de victoire :

Le joueur fait face à un écran indiquant « Félicitation ! » et un bouton proposant de passer au niveau suivant.

Le joueur a également la possibilité de quitter le jeu via le deuxième bouton.

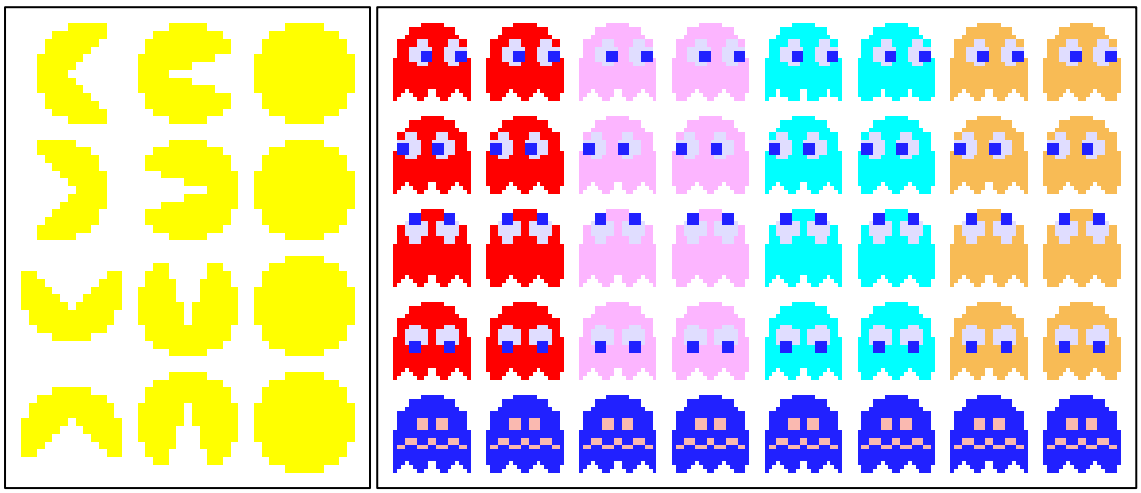
II. Avec quels moyens avons-nous créé ce jeu ?

Pour réaliser ce jeu, nous avons utilisé le logiciel Processing (nous avons personnellement préféré utiliser la version 2.2.1), et nous avons utilisé le Java.

Nous avons également utilisé les bibliothèques control P5 et minim.

1. Les personnages

Pour modéliser les personnages nous avons utilisé ces images, trouvés sur internet mais modifiées par nos soins afin qu'elles conviennent à nos exigences :

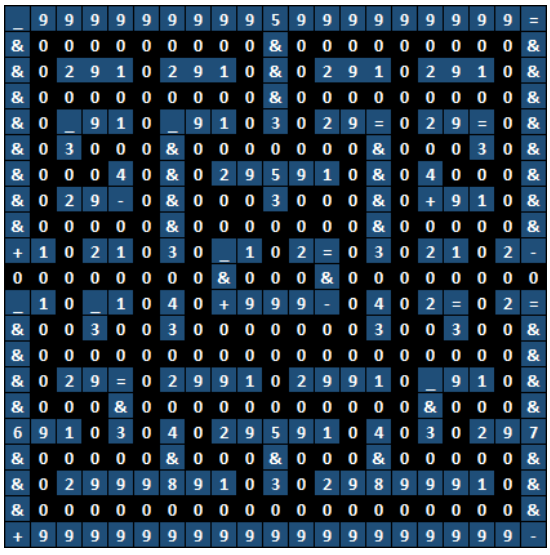


Les personnages ont la possibilité de se déplacer uniquement dans un couloir et les fantômes changent de direction aléatoirement à chaque collision avec un mur.

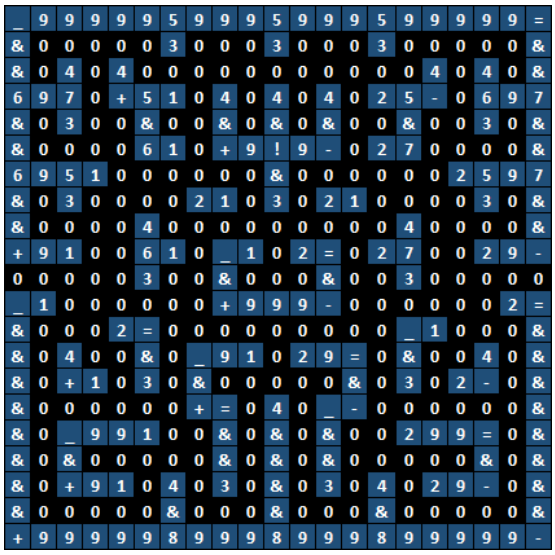
Pacman se déplace grâce aux flèches.

2. Les labyrinthes (lab1.txt et lab2.txt)

Etape 1



// Labyrinthe 1 sur excel



// Labyrinthe 2 sur excel

J'ai tout d'abord utilisé un tableur, coloré en noir ou en bleu les cases selon comment je visualisais les labyrinthes puis j'ai finis par ajouter les correspondances.

Chaque type de mur est caractérisée dans un fichier lab_.txt par un chiffre entre 1 et 9 ou par un symbole entre ‘!’, ‘_’, ‘=’, ‘+’, ‘-’ou ‘&’.
Les couloirs, eux, n’ont pas d’image et sont caractérisés par un 0 dans les fichiers lab.txt, le vide créé de lui-même un couloir.

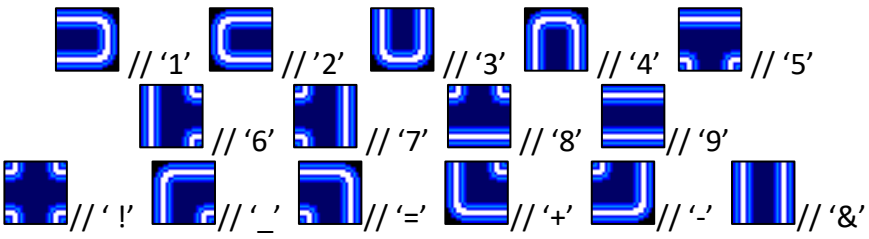
Etape 2

```
_9999999995999999999=
&000000000&000000000&
&029102910&029102910&
&000000000&000000000&
&0_910_9103029=029=0&
&03000&0000000&00030&
&00040&0295910&04000&
&029-0&0003000&0+910&
&00000&0000000&00000&
+1021030_102=0302102-
00000000&000&00000000
_10_1040+999-0402=02=
&0030030000000300300&
&0000000000000000000&
&029=02991029910_910&
&000&00000000000&000&
691030402959104030297
&00000&000&000&00000&
&0299989103029899910&
&0000000000000000000&
+99999999999999999999-
```

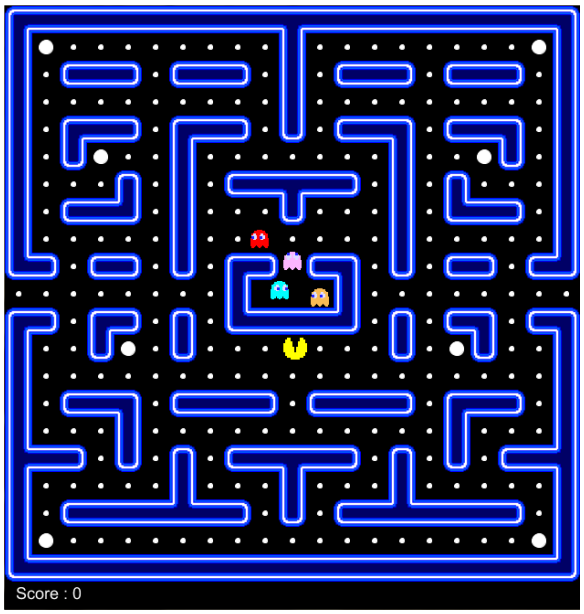
```
_9999959995999599999=
&0000030003000300000&
&0404000000000004040&
6970+51040404025-0697
&0300&00&0&0&00&0030&
&0000+10+9!9-02-0000&
6951000000&0000002597
&03000021030210000030&
&0000400000000040000&
+9100610_102=0270029-
00000300&000&00300000
_1000000+999-0000002=
&0002=000000000_1000&
&0400&0_91029=0&0040&
&0+1030&00000&0302-0&
&000000+=040_-000000&
&0_99100&0&0&00299=0&
&0&00000&0&0&00000&0&
&0+9104030&0304029-0&
&00000&000&000&00000&
+9999989998999899999-
```

Voici le résultat sur les lab1.txt et lab2.txt.

Etape 3



Chaque image mesure 25*25 pixels mais est redimensionnée à 30*30 pixels dans le code.
En faisant correspondre les images avec les fichiers lab_.txt, on obtient ces deux labyrinthes :



// Labyrinthe niveau 1

// Labyrinthe niveau 2

3. La définition des Pacgomme et supers Pacgomme

A. Les fichiers points

```
11111111111111111111
12111111111111111121
11111111111111111111
11111111111111111111
11111111111111111111
11121111111111112111
11111111111111111111
11111111111111111111
11111111111111111111
11111111101111111111
11111111000111111111
11111111111111111111
11112111111111121111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
12111111111111111121
11111111111111111111
```

```
11111111111111111111
11111111111111111111
12111111111111111121
11111111111111111111
11111111212111111111
11111111111111111111
11111111111111111111
11111111111111111111
12111111111111111121
11111111101111111111
11111111000111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11121111111111112111
11111111111111111111
11111111111111111111
11111111111111111111
```

B. Les Pacgommes

Les Pacgommes sont des ellipses de centre de largeur 5 pixels et de hauteur 5 pixels. Elles apparaissent lorsque que le fichier points_.txt affiche un 1 ET qu'il y a un 0 dans le lab_.txt. Une Pacgomme rapporte 10 points et disparaît lorsque Pacman la mange.

C. Les super Pacgommes

Les super Pacgommes sont des ellipses de centre de largeur 15 pixels et de hauteur 15 pixels. Elles sont au nombre de 8. Elles apparaissent lorsque que le fichier points_.txt affiche un 2 ET qu'il y a un 0 dans le lab_.txt. Une super Pacgomme rapporte 50 points et disparait lorsque Pacman la mange, de plus elle gèle les fantômes que pacman peut alors traverser sans perdre.

4. La musique

Le thème connu de Pacman se joue en boucle durant toute la partie.

III. Les améliorations à apporter et bugs à corriger

- Quand les murs forment un coin, le pacman peut entrer en partie dans un mur. Il faudrait redéfinir les valeurs qui permettent de vérifier s'il y a un mur devant le personnage.

- L'état gelé des fantômes nous permet pour l'instant uniquement de les traverser, or pacman est censé pouvoir les manger, ils retournent alors au centre du jeu et rapportent un certain nombre de points.

- Dans le jeu Pacman, il y a un système de fruit qui apparaissent un fois par niveau et qui valent un certain nombre de points.

- Les fantômes, dans le jeu original, ont chacun une « personnalité » qui définit leurs mouvements, or dans notre code, les fantômes se déplacent aléatoirement.

- Le système de meilleur score que nous n'avons pas eu le temps de programmer…

```
//------------------------------------------------- DECLARATIONS ---------
------------------------------------------------------------------------
--
PImage  BHD, BHG, BVB, BVH, C, CB, CD, CG, CH, MH, MV, VBD, VBG, VHD, VHG,
PC, EN; // Variables de type PImage pour gérer les images
import controlP5.*; // On importe la bibliothèque CP5
ControlP5 cp5; // On déclare l'objet CP5
import ddf.minim.*; // On importe la bibliothèques pour les sons
Minim minim ; // Déclaration de l'objet minim pour les sons
AudioPlayer son ; // Déclaration de l'objet son
String lignes [ ] ; // On déclare le tableau de lignes pour le labyrinthe
String points [ ] ; // On déclare le tableau de lignes pour les pac-gommes
Personnage P1; // On déclare l'objet P1 de type Personnage
Ennemi E1; // On déclare l'objet E1 de type Ennemi (Rouge)
Ennemi2 E2; // On déclare l'objet E2 de type Ennemi (Orange)
Ennemi3 E3; // On déclare l'objet E3 de type Ennemi3 (Rose)
Ennemi4 E4; // On déclare l'objet E4 de type Ennemi4 (Bleu)
int t = 630; // Variable de la taille du labyrinthe
int tab1[][] = new int [21][21]; // Tableau déterminant l'état initial 0 ou
l'état trouvé 1 des (super)pacgommes
int tab2[][] = new int [21][21]; // Tableau déterminant l'état initial 1
(pacgomme) ou 2 (superpacgomme) et l'état trouvé en 0 (couloir) lié au
points.txt
Textarea zoneTexte; // On déclare les zones de texte
Textarea zoneGG; // On déclare les zones de texte
Textarea zoneGO; // On déclare les zones de texte
int score = -10; // On initialise le score à -10 car Pacman apparait sur
une pac-gomme
boolean visible = true; // Pacman est visible
Button bouton1; // On déclare un bouton
Button bouton2; // On déclare un bouton
Button bouton3; // On déclare un bouton
color rouge = color(255, 0, 0); // On initialise la couleur rouge
int etat = 0; // Variable de l'état des fantômes initialisée à non gelés
int timer1; // Variable qui permet de calculer le temps de "gelage"
int timer2; // 2e variable qui permet de calculer le temps de "gelage"
//------------------------------------------------- FONCTION  SETUP () ---
------------------------------------------------------------------------
--------
void  setup ( )
{
  frameRate(12); // On fixe le nombre de fois où la fonction draw() est
exécutée en 1 seconde
  background ( 0 ) ; // Couleur du fond de la fenêtre : noir
  size ( 630, 660 ) ; // Taille de la fenêtre : 630 pixels en largeur et
660 pixels en hauteur
  lignes = loadStrings ( "lab1.txt" ) ; // On charge le fichier "lab.txt"
dans le tableau de lignes
  points = loadStrings ("points1.txt"); // On charge le fichier
"points1.txt" dans le tableau de lignes
  BHD = loadImage("BHD.png"); // BHD = Bout Hoizontal dirigé vers la Droite
  BHG = loadImage("BHG.png"); // BHG = Bout Hoizontal dirigé vers la Gauche
  BVB = loadImage("BVB.png"); // BVB = Bout Vertical dirigé vers le Bas
  BVH = loadImage("BVH.png"); // BVH = Bout Vertical dirigé vers le Haut
  C = loadImage("C.png"); // C = Croisement
  CB = loadImage("CB.png"); // CB = Croisement dirigé vers le Bas
  CD = loadImage("CD.png"); // CD = Croisement dirigé vers la Droite
  CG = loadImage("CG.png"); // CG = Croisement dirigé vers la Gauche
  CH = loadImage("CH.png"); // CH = Croisement dirigé vers le Haut
```

```
  MH = loadImage("MH.png"); // MH = Mur Horizontal
  MV = loadImage("MV.png"); // MV = Mur Vertical
  VBD = loadImage("VBD.png"); // VBD = Virage en Bas Droite
  VHG = loadImage("VHG.png"); // VHG = Virage en Haut Gauche
  VBG = loadImage("VBG.png");// VBG = Virage en Bas Gauche
  VHD = loadImage("VHD.png"); // VHD = Virage en Haut Droite
  PC = loadImage("PCspritesheet.png"); // PC = Pacman
  EN = loadImage("Ennemispritesheet.png"); // EN = Ennemi
  P1 = new Personnage(305, 365, 4, 64, PC); // On définit le personnage
(abs, ord, x, y, image)
  E1 = new Ennemi(305, 245, 4, 52, EN); // On définit l'ennemi (abs, ord,
x, y, image)
  E2 = new Ennemi2(335, 305, 148, 52, EN); // On définit l'ennemi (abs,
ord, x, y, image)
  E3 = new Ennemi3(305, 305, 52, 52, EN); // On définit l'ennemi (abs, ord,
x, y, image)
  E4 = new Ennemi4(275, 305, 100, 52, EN); // On définit l'ennemi (abs,
ord, x, y, image)
  for (int i=0; i<21; i++) // Boucle for nombre de lignes verticales du
lab.txt
  {
    for (int j=0; j<21; j++) // Boucle for nombre de lignes horizontales du
lab.txt
    {
      tab1[i][j]=0; // On initialise l'état des points à "non trouvés"
      tab2[i][j]=points[j].charAt(i); // tab2 correspond au points.txt
chargé
    }
  }
  minim = new Minim ( this ) ; // Implémentation de l'objet minim
  son = minim.loadFile ( "Theme.mp3" ) ; // On charge le fichier du son à
exécuter
  son.rewind () ; // On recharge le son
  son.play () ; // On exécute le son
  son.loop(); // On répète le son en boucle
  cp5 = new ControlP5(this); // Implémentation de l'objet cp5
  zoneTexte = cp5.addTextarea("zone")
    .setPosition(10, 630) // Abscisse et ordonnées du coin supérieur gauche
      .setSize(630, 90) // Largeur et hauteur de la zone d'affichage
        .setFont(createFont("arial", 18)) // type et taille de la police
          .setLineHeight(30) // hauteur de chaque ligne dans la zone
d'affichage
            .setColor(color(255)) // Couleur de la police
              .setColorBackground(color(0)) // Couleur du fond
                .setColorForeground(color(0)) // Couleur de premier plan
                  .setText("Score : 0") // Ce qui est écrit
                    .setVisible(true) // Visibilité de la zone
                      ;
  zoneGO = cp5.addTextarea("fin") // On affiche le message de fin
    .setPosition(175, 250) // Abscisse et ordonnées du coin supérieur
gauche
      .setSize(280, 90) // Largeur et hauteur de la zone d'affichage
        .setFont(createFont("arial", 50)) // type et taille de la police
          .setLineHeight(30) // hauteur de chaque ligne dans la zone
d'affichage
            .setColor(color(255)) // Couleur de la police
              .setColorBackground(color(0)) // Couleur du fond
                .setColorForeground(color(0)) // Couleur de premier plan
                  .setText("Game Over") // Ce qui est écrit
                    .setVisible(false) // Visibilité de la zone
                      ;
```

```
  zoneGG = cp5.addTextarea("gg") // On affiche le message de fin
    .setPosition(175, 250) // Abscisse et ordonnées du coin supérieur
gauche
      .setSize(630, 90) // Largeur et hauteur de la zone d'affichage
        .setFont(createFont("arial", 50)) // type et taille de la police
          .setLineHeight(30) // hauteur de chaque ligne dans la zone
d'affichage
            .setColor(color(255)) // Couleur de la police
              .setColorBackground(color(0)) // Couleur du fond
                .setColorForeground(color(0)) // Couleur de premier plan
                  .setText("Félicitations") // Ce qui est écrit
                    .setVisible(false) // Visibilité de la zone
                      ;
  bouton1 = cp5.addButton("recommencer") // On affiche le bouton pour
recommencer
    .setPosition(260, 350) // Abscisse et ordonnées du coin supérieur
gauche
      .setSize(100, 50) // Largeur et hauteur de la zone d'affichage
        .setColorBackground(color(rouge)) // Couleur du fond
          .setVisible(false) // Visibilité du bouton
            ;
  bouton2 = cp5.addButton("niveausuivant") // On affiche le bouton pour
passer au niveau suivant
    .setPosition(185, 350) // Abscisse et ordonnées du coin supérieur
gauche
      .setSize(100, 50) // Largeur et hauteur de la zone d'affichage
        .setColorBackground(color(rouge)) // Couleur du fond
          .setVisible(false) // Visibilité du bouton
            ;
  bouton3 = cp5.addButton("quitter") // On affiche le bouton pour quitter
le jeu
    .setPosition(335, 350) // Abscisse et ordonnées du coin supérieur
gauche
      .setSize(100, 50) // Largeur et hauteur de la zone d'affichage
        .setColorBackground(color(rouge)) // Couleur du fond
          .setVisible(false) // Visibilité du bouton
            ;
}    // Fin de la fonction setup ( )
//------------------------------------------------- FONCTION  DRAW () ------
--------------------------------------------------------------------------
--------
void  draw ( )    // Fonction qui tourne en boucle : même vide elle doit
être présente
{
  if (etat == 1) // Si les fantômes sont gelés
  {
    timer2 = millis()-timer1; // timer2 prend la valeur du nouveau temps
écoulé moins l'ancien
    if ( timer2>10000 ) // quand timer2 est supérieur à 10 secondes
    {
      etat = 0; // Les fantômes redeviennent normaux
    }
  }
  afficherLab(); // On appelle la fonction qui affiche le labyrinthe
  if (visible==true) // Lorsque Pacman est visible
  {
    if (rencontre()==false) // Si Pacman ne touche pas de fantôme
    {
      P1.deplacer(); // On affiche Pacman qui se déplace
    } else
    {
```

```
      visible = false; // On efface Pacman
    }
  }
  E1.deplacer(); // On affiche l'ennemi
  E2.deplacer(); // On affiche l'ennemi
  E3.deplacer(); // On affiche l'ennemi
  E4.deplacer(); // On affiche l'ennemi
  if (score == 2620) // Si on a gagné
  {
    delay(700); // Délais de 0,7 seconde
    background ( 0 ) ; // Couleur du fond de la fenêtre
    size ( 630, 660 ) ; // Taille de la fenêtre : 630 pixels en largeur et
660 pixels en hauteur
    visible = false; // On fait disparaitre Pacman
    zoneGG.setVisible(true); // On fait afficher le message Félicitations
    bouton2.setVisible(true); // On fait afficher le bouton niveau suivant
    bouton3.setVisible(true); // On fait afficher le bouton quitter
  }
  if (visible == false && score<2620) // Si Pacman a disparu et qu'on a pas
récupéré toutes les pacgommes donc qu'on a perdu
  {
    delay(700); // Délais de 0,7 seconde
    background ( 0 ) ; // Couleur du fond de la fenêtre
    size ( 630, 660 ) ; // Taille de la fenêtre : 630 pixels en largeur et
660 pixels en hauteur
    bouton1.setVisible(true); // On fait afficher le bouton recommencer
    zoneGO.setVisible(true); // On fait afficher le message Game Over
  }
}// Fin de la fonction draw ( )
//------------------------------------- FONCTION  KEYPRESSED ( ) --------
--------------------------------------------------------------------------
-------
void keyPressed()
{

  if (keyCode==RIGHT) // Quand on appuie sur la flèche droite
  {
    P1.setY(4); // Renvoie au case vers la droite
  }
  if (keyCode==LEFT) // Quand on appuie sur la flèche gauche
  {
    P1.setY(34); // Renvoie au case vers la gauche
  }
  if (keyCode==UP) // Quand on appuie sur la flèche haut
  {
    P1.setY(64); // Renvoie au case vers le haut
  }
  if (keyCode==DOWN) // Quand on appuie sur la flèche bas
  {
    P1.setY(94); // Renvoie au case vers le bas
  }
}
//------------------------------------- FONCTION  AFFICHERLAB ( ) -------
--------------------------------------------------------------------------
-----
void afficherLab()
{
  background(0);
  fill(0);
  stroke(0);
```

```
  for (int i=0; i<21; i++) // Boucle for nombre de lignes verticales du
lab.txt
  {
    for (int j=0; j<21; j++) // Boucle for nombre de lignes horizontales du
lab.txt
    {
      if (lignes[j].charAt(i) == '&') // Boucle for affichage '&' = MV
      {
        image(MV, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '-') // Boucle for affichage '-' = VBD
      {
        image(VBD, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '_') // Boucle for affichage '_' = VHG
      {
        image(VHG, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '+') // Boucle for affichage '+' = VBG
      {
        image(VBG, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '=') // Boucle for affichage '=' = VHD
      {
        image(VHD, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '1') // Boucle for affichage '1' = BHD
      {
        image(BHD, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '2') // Boucle for affichage '2' = BHG
      {
        image(BHG, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '3') // Boucle for affichage '3' = BVB
      {
        image(BVB, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '4') // Boucle for affichage '4' = BVH
      {
        image(BVH, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '5') // Boucle for affichage '5' = CB
      {
        image(CB, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '6') // Boucle for affichage '6' = CD
      {
        image(CD, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '7') // Boucle for affichage '7' = CG
      {
        image(CG, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '8') // Boucle for affichage '8' = CH
      {
        image(CH, 30*i, 30*j, 30, 30);
      }
      if (lignes[j].charAt(i) == '9') // Boucle for affichage '9' = MH
      {
        image(MH, 30*i, 30*j, 30, 30);
```

```
      }
      if (lignes[j].charAt(i) == '!') // Boucle for affichage '!' = C
      {
        image(C, 30*i, 30*j, 30, 30);
      }
      if (tab1[i][j] == 0 && lignes[j].charAt(i) == '0' &&
points[j].charAt(i) == '1'  )
      {
        pacgomme();
        stroke ( 255 ) ; // Définit la couleur du bord de l'ellipse en
"RGB"
        fill ( 255 ) ; // Définit la couleur de remplissage de l'ellipse
        ellipse ( 15+30*i, 15+30*j, 5, 5 ) ; // trace des ellipses de
centre O ( 15+30*i ; 15+30*j     ) , de largeur 6 et de hauteur 6
      }
      if (tab1[i][j] == 0 && lignes[j].charAt(i) == '0' &&
points[j].charAt(i) == '2'  )
      {
        superpacgomme();
        stroke ( 255 ) ; // Définit la couleur du bord de l'ellipse en
"RGB"
        fill ( 255 ) ; // Définit la couleur de remplissage de l'ellipse
        ellipse ( 15+30*i, 15+30*j, 15, 15 ) ; // trace des ellipses de
centre O ( 15+30*i ; 15+30*j     ) , de largeur 6 et de hauteur 6
      }
    }
  }
}
void pacgomme()
{
  for (int i=0; i<21; i++) // Boucle for nombre de lignes verticales du
lab.txt
  {
    for (int j=0; j<21; j++) // Boucle for nombre de lignes horizontales du
lab.txt
    {
      if ( points[j].charAt(i) == '1'  && tab1[i][j] == 0 &&
((P1.abs+13)/30 ) == i && ((P1.ord+13)/30) == j)
      {
        tab1[i][j]=1;
        tab2[i][j]=0;
        score = score+10; // Une pacgomme rapporte 10 points
        zoneTexte.setText ( "Score : " + score) ; // On affiche le score
dans la zone d'affichage
      }
    }
  }
}
void superpacgomme()
{
  for (int i=0; i<21; i++) // Boucle for nombre de lignes verticales du
lab.txt
  {
    for (int j=0; j<21; j++) // Boucle for nombre de lignes horizontales du
lab.txt
    {
      if (  points[j].charAt(i) == '2' && tab1[i][j] == 0 &&
((P1.abs+13)/30 ) == i && ((P1.ord+13)/30) == j)
      {
        tab1[i][j]=1;
        tab2[i][j]=0;
```

```
        score = score+50; // Une super pacgomme rapporte 50 points
        etat = 1; // On change l'état des fantômes (en gelés)
        timer1 = millis(); //timer1 prend la valeur du temps écoulé depuis
le lancement de la partie
        zoneTexte.setText ( "Score : " + score) ; // On affiche le score
dans la zone d'affichage
      }
    }
  }
}
//--------------------------- FONCTION RENCONTRE() ------------------------
-------------------------------------------------------------------------
----------
boolean rencontre()
{
  boolean touche = false; // On part du principe que Pacman n'a pas été
touché
  if (abs(P1.getAbs() - E1.getAbs()) <=24 && abs(P1.getOrd() - E1.getOrd())
<=24 && etat==0 || abs(P1.getAbs() - E2.getAbs()) <=24 && abs(P1.getOrd() -
E2.getOrd()) <=24 && etat==0 || abs(P1.getAbs() - E3.getAbs()) <=24 &&
abs(P1.getOrd() - E3.getOrd()) <=24 && etat==0 || abs(P1.getAbs() -
E4.getAbs()) <=24 && abs(P1.getOrd() - E4.getOrd()) <=24 && etat==0) // Si
l'écart entre les coordonnées de Pacman et d'un ennemi est inférieur ou
égal à environ la taille de Pacman et que les fantômes ne sont pas gelés
  {
    touche = true; // Pacman a été touché
  }
  return touche; // On transmet que Pacman a été touché
}
//--------------------------- FONCTION RECOMMENCER() ----------------------
-------------------------------------------------------------------------
----------
void recommencer()
{
  score = -10; // On initialise le score à -10 car Pacman apparait sur une
pacgomme
  visible = true; // Pacman est visible
  clear(); // On efface tout ce qui se trouve dans la fenêtre d'affichage
  background ( 0 ) ; // Couleur du fond de la fenêtre
  size ( 630, 660 ) ; // Taille de la fenêtre : 630 pixels en largeur et
660 pixels en hauteur
  lignes = loadStrings ( "lab1.txt" ) ; // On charge le fichier "lab.txt"
dans le tableau de lignes
  points = loadStrings ("points1.txt"); // On charge le fichier
"points1.txt" dans le tableau de points
  bouton1.setVisible(false); // Le bouton1 disparaît
  zoneGO.setVisible(false); // La zone de texte disparaît
  P1 = new Personnage(305, 365, 4, 64, PC); // On définit le personnage
(abs, ord, x, y, image)
  E1 = new Ennemi(305, 245, 4, 52, EN); // On définit l'ennemi (abs, ord,
x, y, image)
  E2 = new Ennemi2(305, 305, 148, 52, EN); // On définit l'ennemi (abs,
ord, x, y, image)
  E3 = new Ennemi3(305, 305, 52, 52, EN); // On définit l'ennemi (abs, ord,
x, y, image)
  E4 = new Ennemi4(305, 305, 100, 52, EN); // On définit l'ennemi (abs,
ord, x, y, image)
  for (int i=0; i<21; i++) // Boucle for nombre de lignes verticales du
lab.txt
  {
```

```
    for (int j=0; j<21; j++) // Boucle for nombre de lignes horizontales du
lab.txt
    {
      tab1[i][j]=0;
      tab2[i][j]=points[j].charAt(i);
    }
  }
  zoneTexte.setVisible(true); // La zone de texte apparaît
}
//------------------------------------ FONCTION QUITTER () ------------
--------------------------------------------------------------------------
--------
void quitter()
{
  exit(); // On fait fermer la fenêtre
}
//------------------------------------ FONCTION NIVEAU SUIVANT () ------
--------------------------------------------------------------------------
--------
void niveausuivant()
{
  score = -10; // On initialise le score à -10 car Pacman apparait sur une
pacgomme
  visible = true; // Pacman est visible
  clear();  // On efface tout ce qui se trouve dans la fenêtre d'affichage
  background ( 0 ) ; // Couleur du fond de la fenêtre
  size ( 630, 660 ) ; // Taille de la fenêtre : 630 pixels en largeur et
660 pixels en hauteur
  lignes = loadStrings ( "lab2.txt" ) ; // On charge le fichier "lab.txt"
dans le tableau de lignes
  points = loadStrings ("points2.txt"); // On charge le fichier
"points1.txt" dans le tableau de points
  bouton2.setVisible(false); // Le bouton2 disparaît
  bouton3.setVisible(false); // Le bouton3 disparaît
  zoneGG.setVisible(false); // La zone de texte disparaît
  P1 = new Personnage(305, 365, 4, 64, PC); // On définit le personnage
(abs, ord, x, y, image)
  E1 = new Ennemi(305, 245, 4, 52, EN); // On définit l'ennemi (abs, ord,
x, y, image)
  E2 = new Ennemi2(335, 305, 148, 52, EN); // On définit l'ennemi (abs,
ord, x, y, image)
  E3 = new Ennemi3(305, 305, 52, 52, EN); // On définit l'ennemi (abs, ord,
x, y, image)
  E4 = new Ennemi4(275, 305, 100, 52, EN); // On définit l'ennemi (abs,
ord, x, y, image)
  for (int i=0; i<21; i++) // Boucle for nombre de lignes verticales du
lab.txt
  {
    for (int j=0; j<21; j++) // Boucle for nombre de lignes horizontales du
lab.txt
    {
      tab1[i][j]=0;
    }
  }
  zoneTexte.setVisible(true); // La zone de texte apparaît
}
//------------------------------------------------------------------ FIN ---
--------------------------------------------------------------------------
-------
```

```
//-------------------------- CLASSE ENNEMI ------------------------------
--
class Ennemi // On déclare la classe Ennemi
{
  // Attributs de la classe
  protected int abs; // Attribut pour l absisse du Personnage
  protected int ord; // Attribut pour l ordonnée du Personnage
  protected int x; // Attributs pour l abscisse dans la planche d images
  protected int y; // Attributs pour l ordonnée dans la planche d images
  protected PImage ima, im; // Attribut pour l image du personnage
  // Constructeur de la classe
  Ennemi(int _abs, int _ord, int _x, int _y, PImage _ima)
  {
    abs = _abs; // On définit l absisse du Personnage
    ord = _ord; // On définit l ordonnée du Personnage
    x = _x; // On définit l abscisse dans la planche d images
    y = _y; // On définit l ordonnée dans la planche d images
    ima = _ima; // On définit l image du Personnage
  } // Fin du constructeur
  // Accesseur pour l'attribut abs
  int getAbs()
  {
    return abs;
  }
  // Mutateur pour l'attribut abs
  void setAbs (int _abs)
  {
    abs = _abs ;
  }
  // Accesseur pour l'attribut ord
  int getOrd()
  {
    return ord;
  }
  // Mutateur pour l'attribut abs
  void setOrd (int _ord)
  {
    ord = _ord ;
  }
  // Accesseur pour l'attribut abs
  int getX()
  {
    return x;
  }
  // Mutateur pour l'attribut abs
  void setX (int _x)
  {
    x = _x ;
  }
  // Accesseur pour l'attribut abs
  int getY()
  {
    return y;
  }
  // Mutateur pour l'attribut abs
  void setY (int _y)
  {
    y = _y ;
  }
  // Méthode d affichage
```

```
  void afficher()
  {
    if (etat == 0) // Si le fantôme n'est pas gelé
    {
      im = ima.get(x, y, 20, 20); // On extrait l image à afficher dans la
planche
    } else
    {
      im = ima.get(x, 100, 20, 20); // On extrait l image à afficher dans
la planche en bloquant dans le bas de la planche pour toutes les directions
    }
    image(im, abs, ord, 20, 20); // On affiche l image en A(abs;ord) de
taille 20 * 20
  } // Fin de la méthode d affichage
  boolean PasDeMurDevant (int _y)
  {
    boolean val = true;
    int xA = 0;
    int xB = 0;
    int yA = 0;
    int yB = 0;
    if (_y == 28) // Si le personnage va à gauche
    {
      xA = (abs-4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs-4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 4) // Si le personnage va à droite
    {
      xA = (abs+20+4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs+20+4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 52) // Si le personnage va en haut
    {
      xA = abs+4; // Abscisse du point A
      yA = ord-4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord-4; // Ordonnée du point B
    }
    if (_y == 76) // Si le personnage va en bas
    {
      xA = abs+4; // Abscisse du point A
      yA = ord+20+4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord+20+4; // Ordonnée du point B
    }
    int i1 = int(yA/30); // Numéro de ligne dans le labyrinthe pour A
    int j1 = int(xA/30); // Numéro de colonne dans le labyrinthe pour A
    int i2 = int(yB/30); // Numéro de ligne dans le labyrinthe pour B
    int j2 = int(xB/30); // Numéro de colonne dans le labyrinthe pour B
    if (lignes[i1].charAt(j1)!='0' && lignes[i2].charAt(j2)!='0')
      // Si il y a un mur devant le personnage
    {
      val=false;
    } else // Sinon si il n'y a pas de mur devant le personnage
    {
      val=true;
    }
```

```
    return val;
  }
  void deplacer()
  {
    switch (y)
    {
    case 4 : // Vers la droite
      if (PasDeMurDevant(4)) // Si le personnage va vers la droite
      {
        setAbs((abs+4+t)%t); // On augmente l'abscisse
        x=(x+24)%48; // On enchaine les images sur la planche initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement, 4+24 c'est la taille du perso avec sa marge + la marge et
c'est un multiple de 52 et 76
      }
      break;
    case 28 : // Vers la gauche
      if (PasDeMurDevant(28))
      {
        setAbs((abs-4+t)%t); // On diminue l'abscisse
        x=(x+24)%48; // On enchaine les images sur la planche initale
      } else
      {

        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }
      break;
    case 76 : // Vers le bas
      if (PasDeMurDevant(76))
      {
        setOrd(ord+4); // On augmente l'ordonnée
        x=(x+24)%48; // On enchaine les images sur la planche initale
        if (ord<260 && ord>220 && abs>300 && abs<340)
        {
          y=4+24*int(random(4));
        }
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }

      break;
    case 52 : // Vers le haut
      if (PasDeMurDevant(52))
      {
        setOrd(ord-4); // On diminue l'ordonnée
        x=(x+24)%48; // On enchaine les images sur la planche initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement OU P1.Y
      }
      break;
    }

    afficher();
  }
```

```
}



//------------------------- CLASSE ENNEMI2 ------------------------------
---
class Ennemi2 // On déclare la classe Ennemi
{
  // Attributs de la classe
  protected int abs; // Attribut pour l absisse du Personnage
  protected int ord; // Attribut pour l ordonnée du Personnage
  protected int x; // Attributs pour l abscisse dans la planche d images
  protected int y; // Attributs pour l ordonnée dans la planche d images
  protected PImage ima, im; // Attribut pour l image du personnage
  // Constructeur de la classe
  Ennemi2(int _abs, int _ord, int _x, int _y, PImage _ima)
  {
    abs = _abs; // On définit l absisse du Personnage
    ord = _ord; // On définit l ordonnée du Personnage
    x = _x; // On définit l abscisse dans la planche d images
    y = _y; // On définit l ordonnée dans la planche d images
    ima = _ima; // On définit l image du Personnage
  } // Fin du constructeur
  // Accesseur pour l'attribut abs
  int getAbs()
  {
    return abs;
  }
  // Mutateur pour l'attribut abs
  void setAbs (int _abs)
  {
    abs = _abs ;
  }
  // Accesseur pour l'attribut ord
  int getOrd()
  {
    return ord;
  }
  // Mutateur pour l'attribut abs
  void setOrd (int _ord)
  {
    ord = _ord ;
  }
  // Accesseur pour l'attribut abs
  int getX()
  {
    return x;
  }
  // Mutateur pour l'attribut abs
  void setX (int _x)
  {
    x = _x ;
  }
  // Accesseur pour l'attribut abs
  int getY()
  {
    return y;
  }
  // Mutateur pour l'attribut abs
```

```
  void setY (int _y)
  {
    y = _y ;
  }
  // Méthode d affichage
  void afficher()
  {
    if (etat == 0)
    {
      im = ima.get(x, y, 20, 20); // On extrait l image à afficher dans la
planche
    } else
    {
      im = ima.get(x, 100, 20, 20); // On extrait l image à afficher dans
la planche en bloquant dans le bas de la planche pour toutes les directions
    }
    image(im, abs, ord, 20, 20); // On affiche l image en A(abs;ord) de
taille 20 * 20
  } // Fin de la méthode d affichage
  boolean PasDeMurDevant (int _y)
  {
    boolean val = true;
    int xA = 0;
    int xB = 0;
    int yA = 0;
    int yB = 0;
    if (_y == 28) // Si le personnage va à gauche
    {
      xA = (abs-4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs-4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 4) // Si le personnage va à droite
    {
      xA = (abs+20+4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs+20+4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 52) // Si le personnage va en haut
    {
      xA = abs+4; // Abscisse du point A
      yA = ord-4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord-4; // Ordonnée du point B
    }
    if (_y == 76) // Si le personnage va en bas
    {
      xA = abs+4; // Abscisse du point A
      yA = ord+20+4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord+20+4; // Ordonnée du point B
    }
    int i1 = int(yA/30); // Numéro de ligne dans le labyrinthe pour A
    int j1 = int(xA/30); // Numéro de colonne dans le labyrinthe pour A
    int i2 = int(yB/30); // Numéro de ligne dans le labyrinthe pour B
    int j2 = int(xB/30); // Numéro de colonne dans le labyrinthe pour B
    if (lignes[i1].charAt(j1)!='0' && lignes[i2].charAt(j2)!='0')
      // Si il y a un mur devant le personnage
    {
```

```
      val=false;
    } else // Sinon si il n'y a pas de mur devant le personnage
    {
      val=true;
    }
    return val;
  }

  void deplacer()
  {
    switch (y)
    {

    case 4 : // Vers la droite
      if (PasDeMurDevant(4)) // Si le personnage va vers la droite
      {
        setAbs((abs+4+t)%t); // On augmente l'abscisse
        x=(x-148+24)%48+148; // On enchaine les images sur la planche
initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement dans la planche d'image
      }
      break;
    case 28 : // Vers la gauche
      if (PasDeMurDevant(28))
      {
        setAbs((abs-4+t)%t); // On diminue l'abscisse
        x=(x-148+24)%48+148; // On enchaine les images sur la planche
initale
        if (abs<305 && abs>275 && ord>300 && ord<340)
        {
          y=52;
        }
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }
      break;
    case 76 : // Vers le bas
      if (PasDeMurDevant(76))
      {
        setOrd(ord+4); // On augmente l'ordonnée
        x=(x-148+24)%48+148; // On enchaine les images sur la planche
initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }
      break;
    case 52 : // Vers le haut
      if (PasDeMurDevant(52))
      {
        setOrd(ord-4); // On diminue l'ordonnée
        x=(x-148+24)%48+148; // On enchaine les images sur la planche
initale
      } else
      {
```

```
        if (ord<260 && ord >220 && abs>300 && abs<340)
        {
          y=4+24*int(random(2));
        } else {
          y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement OU P1.Y
        }
      }

      break;
    }
    afficher();
  }
}



//------------------------- CLASSE ENNEMI3 ------------------------------
---
class Ennemi3 // On déclare la classe Ennemi
{
  // Attributs de la classe
  protected int abs; // Attribut pour l absisse du Personnage
  protected int ord; // Attribut pour l ordonnée du Personnage
  protected int x; // Attributs pour l abscisse dans la planche d images
  protected int y; // Attributs pour l ordonnée dans la planche d images
  protected PImage ima, im; // Attribut pour l image du personnage
  // Constructeur de la classe
  Ennemi3(int _abs, int _ord, int _x, int _y, PImage _ima)
  {
    abs = _abs; // On définit l absisse du Personnage
    ord = _ord; // On définit l ordonnée du Personnage
    x = _x; // On définit l abscisse dans la planche d images
    y = _y; // On définit l ordonnée dans la planche d images
    ima = _ima; // On définit l image du Personnage
  } // Fin du constructeur
  // Accesseur pour l'attribut abs
  int getAbs()
  {
    return abs;
  }
  // Mutateur pour l'attribut abs
  void setAbs (int _abs)
  {
    abs = _abs ;
  }
  // Accesseur pour l'attribut ord
  int getOrd()
  {
    return ord;
  }
  // Mutateur pour l'attribut abs
  void setOrd (int _ord)
  {
    ord = _ord ;
  }
  // Accesseur pour l'attribut abs
  int getX()
  {
    return x;
  }
```

```
  // Mutateur pour l'attribut abs
  void setX (int _x)
  {
    x = _x ;
  }
  // Accesseur pour l'attribut abs
  int getY()
  {
    return y;
  }
  // Mutateur pour l'attribut abs
  void setY (int _y)
  {
    y = _y ;
  }
  // Méthode d affichage
  void afficher()
  {
    if (etat == 0)
    {
      im = ima.get(x, y, 20, 20); // On extrait l image à afficher dans la
planche
    } else
    {
      im = ima.get(x, 100, 20, 20); // On extrait l image à afficher dans
la planche en bloquant dans le bas de la planche pour toutes les directions
    }
    image(im, abs, ord, 20, 20); // On affiche l image en A(abs;ord) de
taille 20 * 20
  } // Fin de la méthode d affichage
  boolean PasDeMurDevant (int _y)
  {
    boolean val = true;
    int xA = 0;
    int xB = 0;
    int yA = 0;
    int yB = 0;
    if (_y == 28) // Si le personnage va à gauche
    {
      xA = (abs-4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs-4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 4) // Si le personnage va à droite
    {
      xA = (abs+20+4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs+20+4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 52) // Si le personnage va en haut
    {
      xA = abs+4; // Abscisse du point A
      yA = ord-4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord-4; // Ordonnée du point B
    }
    if (_y == 76) // Si le personnage va en bas
    {
      xA = abs+4; // Abscisse du point A
```

```
      yA = ord+20+4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord+20+4; // Ordonnée du point B
    }
    int i1 = int(yA/30); // Numéro de ligne dans le labyrinthe pour A
    int j1 = int(xA/30); // Numéro de colonne dans le labyrinthe pour A
    int i2 = int(yB/30); // Numéro de ligne dans le labyrinthe pour B
    int j2 = int(xB/30); // Numéro de colonne dans le labyrinthe pour B
    if (lignes[i1].charAt(j1)!='0' && lignes[i2].charAt(j2)!='0')
      // Si il y a un mur devant le personnage
    {
      val=false;
    } else // Sinon si il n'y a pas de mur devant le personnage
    {
      val=true;
    }
    return val;
  }
  void deplacer()
  {
    switch (y)
    {
    case 4 : // Vers la droite
      if (PasDeMurDevant(4)) // Si le personnage va vers la droite
      {
        setAbs((abs+4+t)%t); // On augmente l'abscisse
        x=(x-52+24)%48+52; // On enchaine les images sur la planche initale
        if (abs<335 && abs>305 && ord>300 && ord<340)
        {
          y=52;
        }
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement dans la planche d'image
      }
      break;
    case 28 : // Vers la gauche
      if (PasDeMurDevant(28))
      {
        setAbs((abs-4+t)%t); // On diminue l'abscisse
        x=(x-52+24)%48+52; // On enchaine les images sur la planche initale
        if (abs<305 && abs>275 && ord>300 && ord<340)
        {
          y=52;
        }
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }
      break;
    case 76 : // Vers le bas
      if (PasDeMurDevant(76))
      {
        setOrd(ord+4); // On augmente l'ordonnée
        x=(x-52+24)%48+52; // On enchaine les images sur la planche initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
```

```
      }
      break;
    case 52 : // Vers le haut
      if (PasDeMurDevant(52))
      {
        setOrd(ord-4); // On diminue l'ordonnée
        x=(x-52+24)%48+52; // On enchaine les images sur la planche initale
      } else
      {
        if (ord<260 && ord >220 && abs>300 && abs<340)
        {
          y=4+24*int(random(2));
        } else {
          y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement OU P1.Y
        }
      }
      break;
    }
    afficher();
  }
}



//-------------------------- CLASSE ENNEMI4 ------------------------------
---
class Ennemi4 // On déclare la classe Ennemi
{
  // Attributs de la classe
  protected int abs; // Attribut pour l absisse du Personnage
  protected int ord; // Attribut pour l ordonnée du Personnage
  protected int x; // Attributs pour l abscisse dans la planche d images
  protected int y; // Attributs pour l ordonnée dans la planche d images
  protected PImage ima, im; // Attribut pour l image du personnage
  // Constructeur de la classe
  Ennemi4(int _abs, int _ord, int _x, int _y, PImage _ima)
  {
    abs = _abs; // On définit l absisse du Personnage
    ord = _ord; // On définit l ordonnée du Personnage
    x = _x; // On définit l abscisse dans la planche d images
    y = _y; // On définit l ordonnée dans la planche d images
    ima = _ima; // On définit l image du Personnage
  } // Fin du constructeur
  // Accesseur pour l'attribut abs
  int getAbs()
  {
    return abs;
  }
  // Mutateur pour l'attribut abs
  void setAbs (int _abs)
  {
    abs = _abs ;
  }
  // Accesseur pour l'attribut ord
  int getOrd()
  {
    return ord;
  }
  // Mutateur pour l'attribut abs
  void setOrd (int _ord)
```

```
  {
    ord = _ord ;
  }
  // Accesseur pour l'attribut abs
  int getX()
  {
    return x;
  }
  // Mutateur pour l'attribut abs
  void setX (int _x)
  {
    x = _x ;
  }
  // Accesseur pour l'attribut abs
  int getY()
  {
    return y;
  }
  // Mutateur pour l'attribut abs
  void setY (int _y)
  {
    y = _y ;
  }
  // Méthode d affichage
  void afficher()
  {
    if (etat == 0)
    {
      im = ima.get(x, y, 20, 20); // On extrait l image à afficher dans la
planche
    } else
    {
      im = ima.get(x, 100, 20, 20); // On extrait l image à afficher dans
la planche en bloquant dans le bas de la planche pour toutes les directions
    }
    image(im, abs, ord, 20, 20); // On affiche l image en A(abs;ord) de
taille 20 * 20
  } // Fin de la méthode d affichage
  boolean PasDeMurDevant (int _y)
  {
    boolean val = true;
    int xA = 0;
    int xB = 0;
    int yA = 0;
    int yB = 0;
    if (_y == 28) // Si le personnage va à gauche
    {
      xA = (abs-4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs-4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 4) // Si le personnage va à droite
    {
      xA = (abs+20+4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs+20+4+t)%t; // Abscisse du point B
      yB = ord+20-4; // Ordonnée du point B
    }
    if (_y == 52) // Si le personnage va en haut
    {
```

```
      xA = abs+4; // Abscisse du point A
      yA = ord-4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord-4; // Ordonnée du point B
    }
    if (_y == 76) // Si le personnage va en bas
    {
      xA = abs+4; // Abscisse du point A
      yA = ord+20+4; // Ordonnée du point A
      xB = abs+20-4; // Abscisse du point B
      yB = ord+20+4; // Ordonnée du point B
    }
    int i1 = int(yA/30); // Numéro de ligne dans le labyrinthe pour A
    int j1 = int(xA/30); // Numéro de colonne dans le labyrinthe pour A
    int i2 = int(yB/30); // Numéro de ligne dans le labyrinthe pour B
    int j2 = int(xB/30); // Numéro de colonne dans le labyrinthe pour B
    if (lignes[i1].charAt(j1)!='0' && lignes[i2].charAt(j2)!='0')
      // Si il y a un mur devant le personnage
    {
      val=false;
    } else // Sinon si il n'y a pas de mur devant le personnage
    {
      val=true;
    }
    return val;
  }
  void deplacer()
  {
    switch (y)
    {
    case 4 : // Vers la droite
      if (PasDeMurDevant(4) ) // Si le personnage va vers la droite
      {
        setAbs((abs+4+t)%t); // On augmente l'abscisse
        x=(x-100+24)%48+100; // On enchaine les images sur la planche
initale
        if (abs<335 && abs>305 && ord>300 && ord<340)
        {
          y=52;
        }
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement dans la planche d'image
      }
      break;
    case 28 : // Vers la gauche
      if (PasDeMurDevant(28))
      {
        setAbs((abs-4+t)%t); // On diminue l'abscisse
        x=(x-100+24)%48+100; // On enchaine les images sur la planche
initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }
      break;
    case 76 : // Vers le bas
      if (PasDeMurDevant(76))
      {
```

```
        setOrd(ord+4); // On augmente l'ordonnée
        x=(x-100+24)%48+100; // On enchaine les images sur la planche
initale
      } else
      {
        y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement
      }
      break;
    case 52 : // Vers le haut
      if (PasDeMurDevant(52))
      {
        setOrd(ord-4); // On diminue l'ordonnée
        x=(x-100+24)%48+100; // On enchaine les images sur la planche
initale
      } else
      {
        if (ord<260 && ord >220 && abs>300 && abs<340)
        {
          y=4+24*int(random(2));
        } else {
          y=4+24*int(random(4)); // On recalcule une autre direction
aléatoirement OU P1.Y
        }
      }
      break;
    }
    afficher();
  }
}



// -------------------- CLASSE PERSONNAGE ----------------
class Personnage // On déclare la classe Personnage
{
  // Attributs de la classe
  private int abs; // Attribut pour l absisse du Personnage
  private int ord; // Attribut pour l ordonnée du Personnage
  private int x; // Attributs pour l abscisse dans la planche d images
  private int y; // Attributs pour l ordonnée dans la planche d images
  private PImage ima, im; // Attribut pour l image du personnage
  // Constructeur de la classe
  Personnage(int _abs, int _ord, int _x, int _y, PImage _ima)
  {
    abs = _abs; // On définit l absisse du Personnage
    ord = _ord; // On définit l ordonnée du Personnage
    x = _x; // On définit l abscisse dans la planche d images
    y = _y; // On définit l ordonnée dans la planche d images
    ima = _ima; // On définit l image du Personnage
  } // Fin du constructeur
  // Accesseur pour l'attribut abs
  int getAbs()
  {
    return abs;
  }
  // Mutateur pour l'attribut abs
  void setAbs (int _abs)
  {
    abs = _abs ;
  }
```

```
  // Accesseur pour l'attribut ord
  int getOrd()
  {
    return ord;
  }
  // Mutateur pour l'attribut abs
  void setOrd (int _ord)
  {
    ord = _ord ;
  }
  // Accesseur pour l'attribut abs
  int getX()
  {
    return x;
  }
  // Mutateur pour l'attribut abs
  void setX (int _x)
  {
    x = _x ;
  }
  // Accesseur pour l'attribut abs
  int getY()
  {
    return y;
  }
  // Mutateur pour l'attribut abs
  void setY (int _y)
  {
    y = _y ;
  }
  // Méthode d'affichage
  void afficher()
  {
    im = ima.get(x, y, 26, 26); // On extrait l image à afficher dans la
planche
    image(im, abs, ord, 26, 26); // On affiche l image en A(abs;ord) de
taille 26*26
  } // Fin de la méthode d affichage
  boolean PasDeMurDevant (int _y)
  {
    boolean val = true;
    int xA = 0;
    int xB = 0;
    int yA = 0;
    int yB = 0;
    if (_y == 34) // Si le personnage va à gauche
    {
      xA = (abs-4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs-4+t)%t; // Abscisse du point B
      yB = ord+26-4; // Ordonnée du point B
    }
    if (_y == 4) // Si le personnage va à droite
    {
      xA = (abs+26+4+t)%t; // Abscisse du point A
      yA = ord+4; // Ordonnée du point A
      xB = (abs+26+4+t)%t; // Abscisse du point B
      yB = ord+26-4; // Ordonnée du point B
    }
    if (_y == 64) // Si le personnage va en haut
    {
```

```
    xA = abs+4; // Abscisse du point A
    yA = ord-4; // Ordonnée du point A
    xB = abs+26-4; // Abscisse du point B
    yB = ord-4; // Ordonnée du point B
  }
  if (_y == 94) // Si le personnage va en bas
  {
    xA = abs+4; // Abscisse du point A
    yA = ord+26+4; // Ordonnée du point A
    xB = abs+26-4; // Abscisse du point B
    yB = ord+26+4; // Ordonnée du point B
  }
  int i1 = int(yA/30); // Numéro de ligne dans le labyrinthe pour A
  int j1 = int(xA/30); // Numéro de colonne dans le labyrinthe pour A
  int i2 = int(yB/30); // Numéro de ligne dans le labyrinthe pour B
  int j2 = int(xB/30); // Numéro de colonne dans le labyrinthe pour B
  if (lignes[i1].charAt(j1)!='0' && lignes[i2].charAt(j2)!='0')
    // Si il y a un mur devant le personnage
  {
    val=false;
  } else // Sinon si il n'y a pas de mur devant le personnage
  {
    val=true;
  }
  return val;
}
// Méthode polymorphe deplacer()
void deplacer()
{
  switch (y)
  {
  case 4 : // Vers la droite
    if (PasDeMurDevant(4))
    {
      setAbs((abs+4+t)%t); // On augmente l'abscisse
      x=(x+30)%90; // On enchaine les images sur la planche initiale
    }
    break;
  case 34 : // Vers la gauche
    if (PasDeMurDevant(34))
    {
      setAbs((abs-4+t)%t); // On diminue l'abscisse
      x=(x+30)%90; // On enchaine les images sur la planche initiale
    }
    break;
  case 94 : // Vers le bas
    if (PasDeMurDevant(94))
    {
      setOrd(ord+4); // On augmente l'ordonnée
      x=(x+30)%90; // On enchaine les images sur la planche initiale
    }
    break;
  case 64 : // Vers le haut
    if (PasDeMurDevant(64))
    {
      setOrd(ord-4); // On diminue l'ordonnée
      x=(x+30)%90; // On enchaine les images sur la planche initiale
    }
    break;
  }
  afficher();
```

```
  } // Fin de la méthode deplacer()
}
```