

Rapport

Nous avons eu pour défi d'encoder une chenille en langage JAVA, qui s'exécutera à l'aide d'une applet HTML.

Pour ce faire, Nous avons défini pas à pas plusieurs objets qui, une fois regroupé, formeront la tête et le corps de notre chenille. Pour finir, nous allons proposer une extension de la chenille prenant en compte de nouveaux paramètres.

Nous allons donc détaillé ci-dessous les différentes étapes par lesquelles nous sommes passées avant d'arriver au script final.

Table des matières

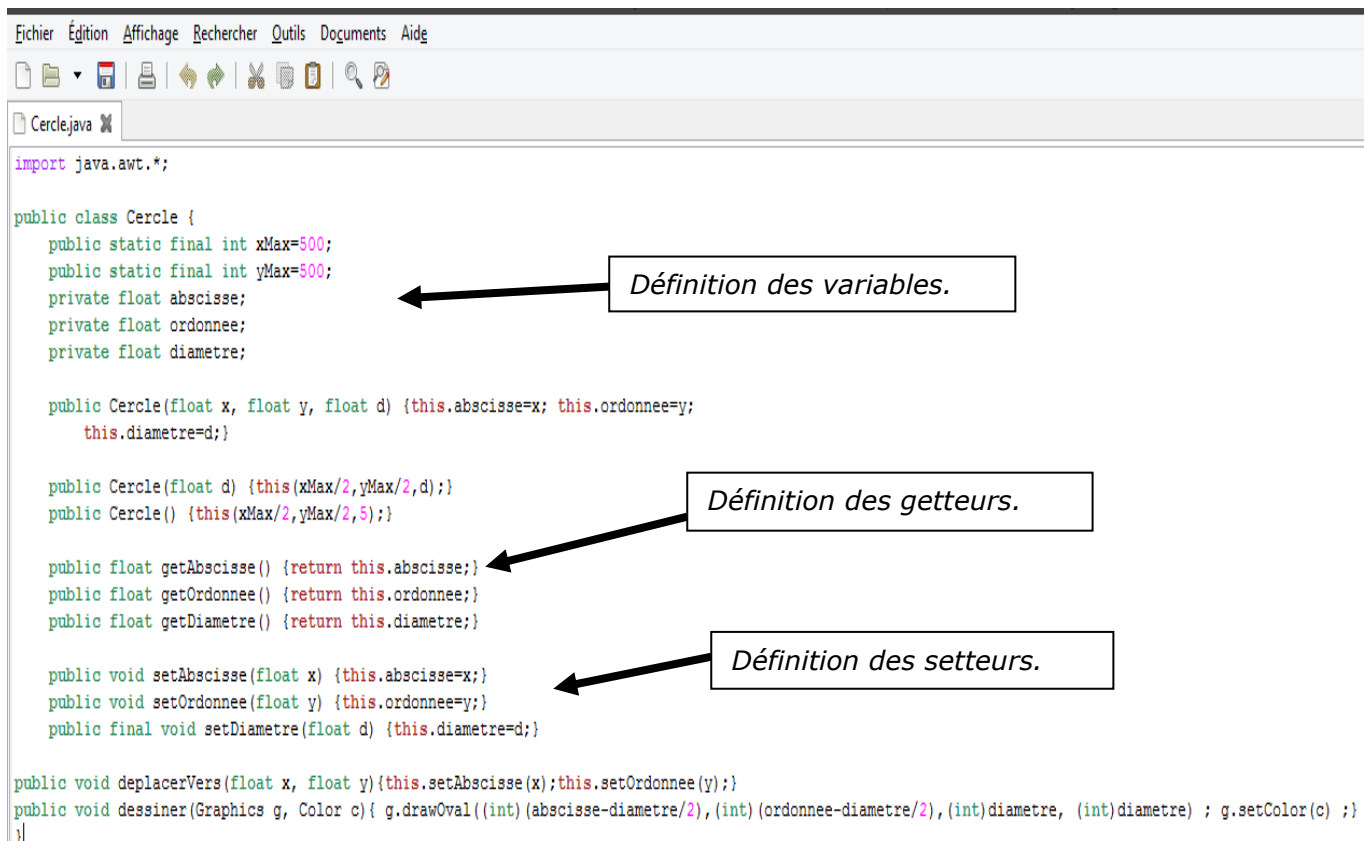
1 – Le Cercle :	2
A – Cercle.java	2
B – TestCercle.java	3
C – Cercle.html	3
D – Compilation et Exécution	4
2 – Le Cercle Animée	5
A – CercleAnime.java	5
B – TestCercleAnime.java	6
3 – La Tête de la chenille	7
A – TeteDeChenille.java	7
B – TestTeteDeChenille.java	8
C – Difficultés et résultats	9
4 – La Chenille	10
A – Chenille.java	10
B – TestChenille.java	11
C – Difficultés et résultats	12
5 – Extension de la chenille	13
A – MaChenille.java	13
B – TestMaChenille.java	14
C – Difficultés et Résultats	15
6 – Relation d'héritage entre les objets	17

1 – Le Cercle :

A – Cercle.java

La première étape du projet consistait à encoder un cercle.

Un cercle se caractérise par son centre (abscisse et ordonnée) et son diamètre tel que :



```
import java.awt.*;

public class Cercle {
    public static final int xMax=500;
    public static final int yMax=500;
    private float abscisse;
    private float ordonnee;
    private float diametre;

    public Cercle(float x, float y, float d) {this.abscisse=x; this.ordonnee=y;
        this.diametre=d;}

    public Cercle(float d) {this(xMax/2,yMax/2,d);}
    public Cercle() {this(xMax/2,yMax/2,5);}

    public float getAbscisse() {return this.abscisse;}
    public float getOrdonnee() {return this.ordonnee;}
    public float getDiametre() {return this.diametre;}

    public void setAbscisse(float x) {this.abscisse=x;}
    public void setOrdonnee(float y) {this.ordonnee=y;}
    public final void setDiametre(float d) {this.diametre=d;}

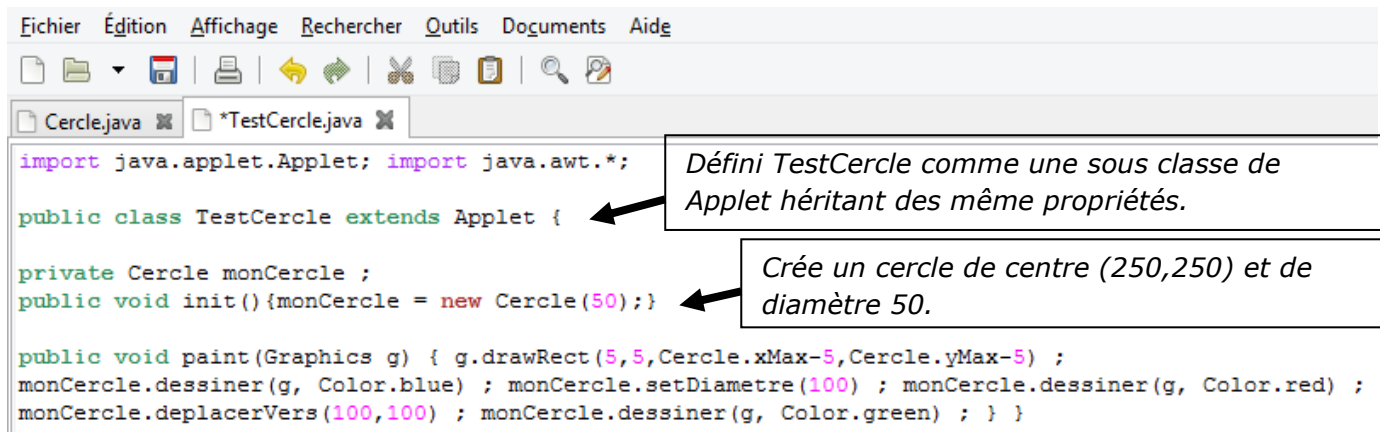
    public void deplacerVers(float x, float y){this.setAbscisse(x);this.setOrdonnee(y);}
    public void dessiner(Graphics g, Color c){ g.drawOval((int)(abscisse-diametre/2),(int)(ordonnee-diametre/2),(int)diametre, (int)diametre) ; g.setColor(c) ;}
}
```

On peut voir sur les deux dernières lignes de code que nous avons défini deux méthodes :

1. *deplacerVers* : Redéfinit l'abscisse et l'ordonnée du centre du cercle afin de modifier la position du cercle dans le plan.
2. *dessiner* : Utilise la méthode *g.drawOval()* prédéfinie dans le package *java.awt* afin de tracer un cercle d'une couleur au choix.

B – TestCercle.java

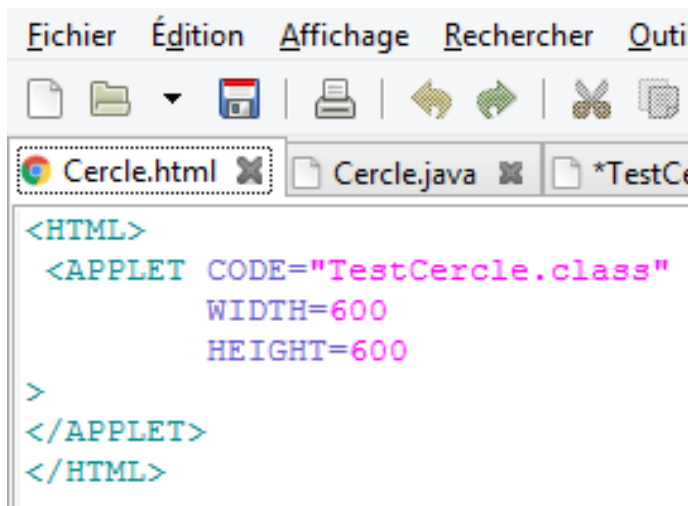
Afin de tester si notre classe Cercle fonctionne correctement, Nous allons encoder un fichier java qui produira différents Cercle en utilisant le fichier ci-dessus :



Nous pouvons observer que la définition de la méthode *paint ()* est une méthode abstraite définie au sein du package *java.applet.Applet*. Elle sert à tracer le cercle obtenu par la méthode *dessiner()* définie dans la classe *cercle* dans une fenêtre graphique html définie comme suit.

C - Cercle.html

Nous allons créer un fichier Cercle.html afin de lancer la classe TestCercle dans une fenêtre graphique



Le fichier ci-contre permet d'exécuter la classe TestCercle dans une fenêtre graphique de longueur=600 et de largeur=600.

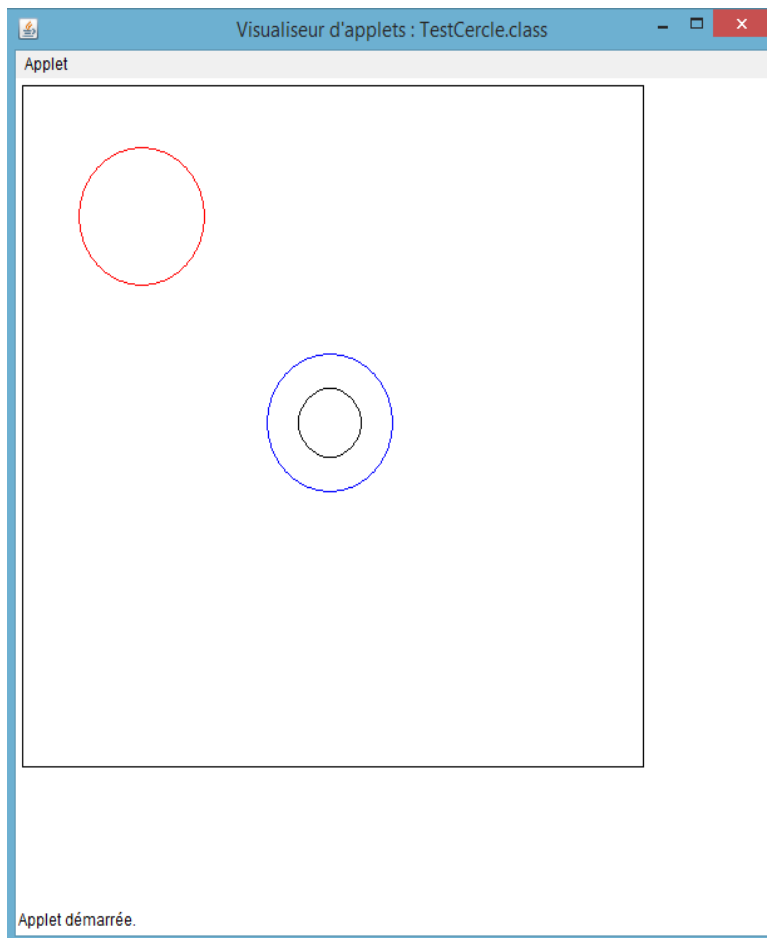
Le langage html fonctionne par balise. Nous lui indiquons ici le type de langage utilisé ainsi que la commande à utiliser.

D – Compilation et Exécution

Afin de pouvoir exécuter nos fichiers, nous allons devoir au préalable les compiler en fichier.class à l'aide de la commande *javac*.

```
C:\Users\Rémi F\Desktop\Travail\LICENCE MASS\L3MASS\INFO\PROJET>javac Cercle.java
C:\Users\Rémi F\Desktop\Travail\LICENCE MASS\L3MASS\INFO\PROJET>javac TestCercle.java
C:\Users\Rémi F\Desktop\Travail\LICENCE MASS\L3MASS\INFO\PROJET>appletviewer htmlCercle.html
```

Une fois l'opération réalisé, nous allons pouvoir exécuter notre classe *TestCercle* dans la fenêtre graphique défini dans *Cercle.java* en faisant appel à la comande *appletviewer*.



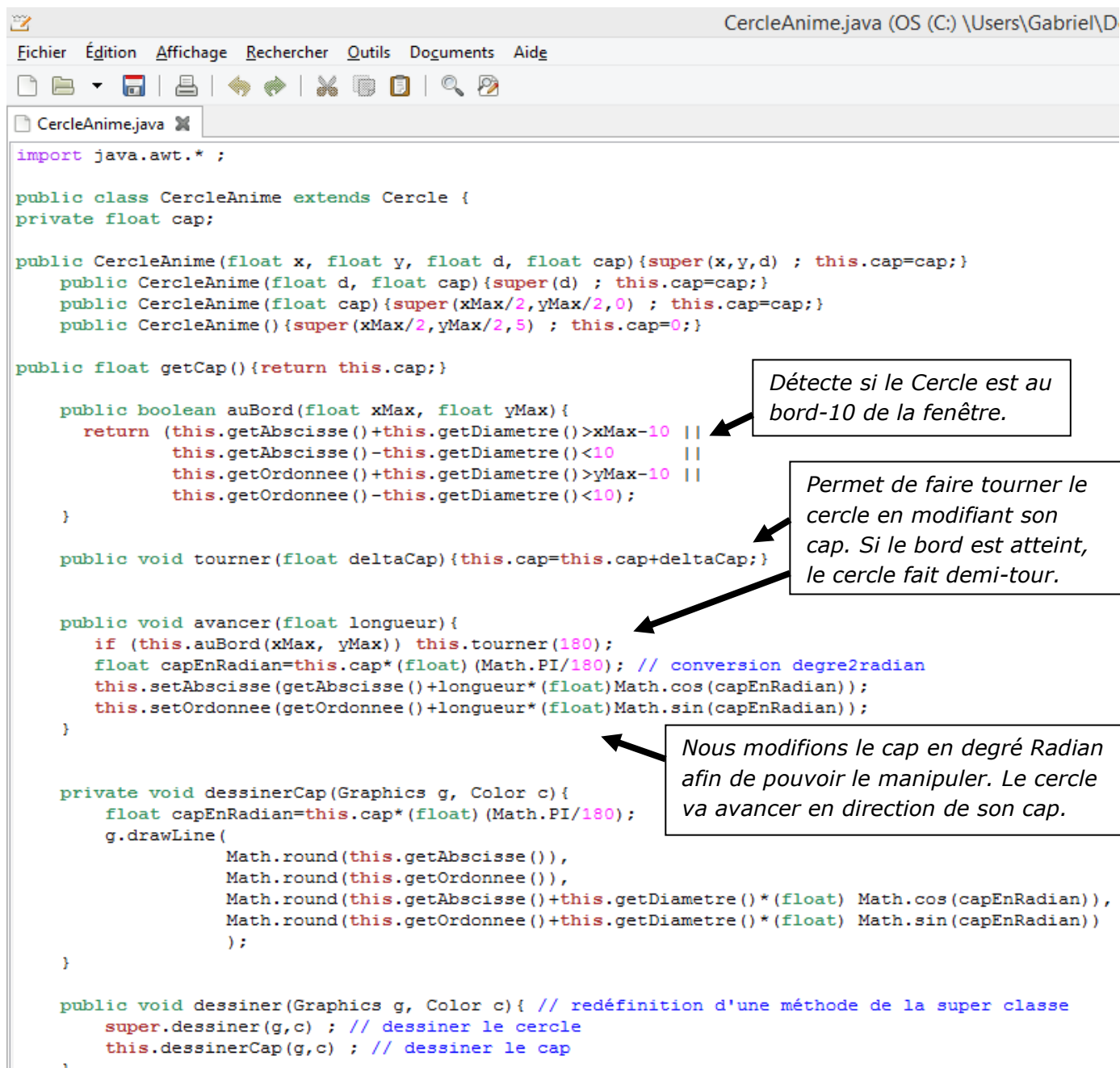
Comme indiqué dans le fichier *Test*, nous allons tracer un premier cercle de centre (250,250) et de diamètre 50 puis un autre de même centre et de diamètre 100.

Pour finir, nous tracerons un dernier cercle de centre (100,100) avec l'appel de la méthode *deplacervers* et de diamètre 100.

2 – Le Cercle Animée

A – CercleAnime.java

Un cercle Animée est une sous classe de Cercle héritant des mêmes propriétés auquel nous ajoutons un cap (en degré) lui permettant de tourner dans l'espace. Il est également capable de détecter les bords de la fenêtre graphique dans laquelle il s'exécute.



```
import java.awt.* ;

public class CercleAnime extends Cercle {
    private float cap;

    public CercleAnime(float x, float y, float d, float cap){super(x,y,d) ; this.cap=cap;}
    public CercleAnime(float d, float cap){super(d) ; this.cap=cap;}
    public CercleAnime(float cap){super(xMax/2,yMax/2,0) ; this.cap=cap;}
    public CercleAnime(){super(xMax/2,yMax/2,5) ; this.cap=0;}

    public float getCap(){return this.cap;}

    public boolean auBord(float xMax, float yMax){
        return (this.getAbscisse()+this.getDiametre()>xMax-10 ||
                this.getAbscisse()-this.getDiametre()<10 ||
                this.getOrdonnee()+this.getDiametre()>yMax-10 ||
                this.getOrdonnee()-this.getDiametre()<10);
    }

    public void tourner(float deltaCap){this.cap=this.cap+deltaCap;}

    public void avancer(float longueur){
        if (this.auBord(xMax, yMax)) this.tourner(180);
        float capEnRadian=this.cap*(float) (Math.PI/180); // conversion degre2radian
        this.setAbscisse(getAbscisse()+longueur*(float)Math.cos(capEnRadian));
        this.setOrdonnee(getOrdonnee()+longueur*(float)Math.sin(capEnRadian));
    }

    private void dessinerCap(Graphics g, Color c){
        float capEnRadian=this.cap*(float) (Math.PI/180);
        g.drawLine(
            Math.round(this.getAbscisse()),
            Math.round(this.getOrdonnee()),
            Math.round(this.getAbscisse()+this.getDiametre()*(float) Math.cos(capEnRadian)),
            Math.round(this.getOrdonnee()+this.getDiametre()*(float) Math.sin(capEnRadian))
        );
    }

    public void dessiner(Graphics g, Color c){ // redéfinition d'une méthode de la super classe
        super.dessiner(g,c) ; // dessiner le cercle
        this.dessinerCap(g,c) ; // dessiner le cap
    }
}
```

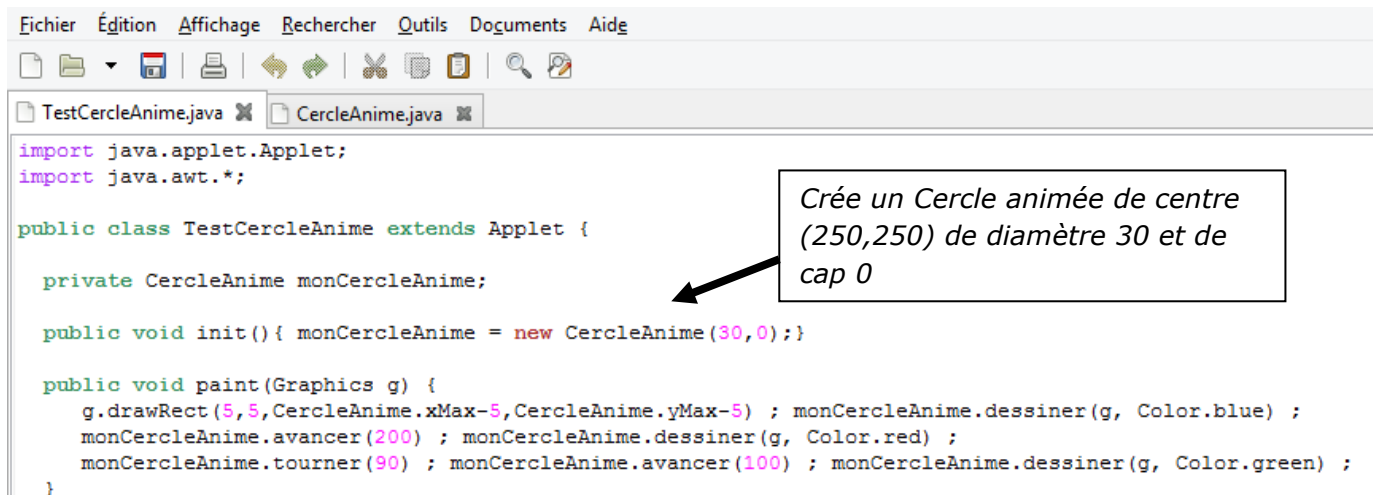
Détecte si le Cercle est au bord-10 de la fenêtre.

Permet de faire tourner le cercle en modifiant son cap. Si le bord est atteint, le cercle fait demi-tour.

Nous modifions le cap en degré Radian afin de pouvoir le manipuler. Le cercle va avancer en direction de son cap.

B – TestCercleAnime.java

Dans le même esprit que pour le Cercle, nous allons créer un fichier test afin de pouvoir vérifier le bon fonctionnement de notre code.



```
import java.applet.Applet;
import java.awt.*;

public class TestCercleAnime extends Applet {

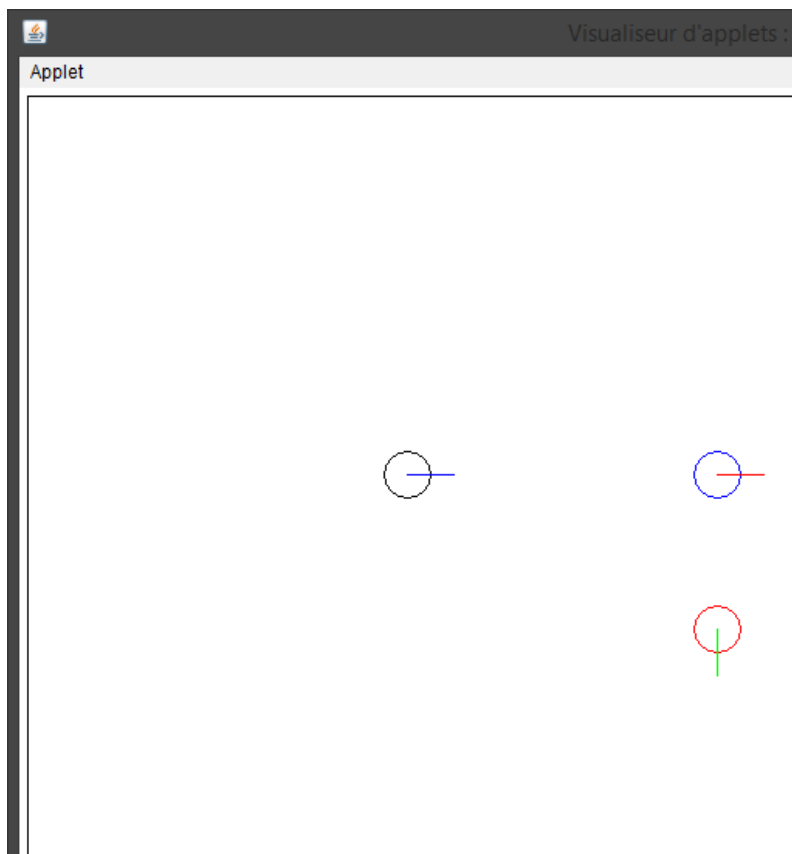
    private CercleAnime monCercleAnime;

    public void init(){ monCercleAnime = new CercleAnime(30,0);}

    public void paint(Graphics g) {
        g.drawRect(5,5,CercleAnime.xMax-5,CercleAnime.yMax-5) ; monCercleAnime.dessiner(g, Color.blue) ;
        monCercleAnime.avancer(200) ; monCercleAnime.dessiner(g, Color.red) ;
        monCercleAnime.tourner(90) ; monCercleAnime.avancer(100) ; monCercleAnime.dessiner(g, Color.green) ;
    }
}
```

L'encodage du fichier html est similaire à celui de Cercle (seul le nom du code est à modifier), De même pour la compilation.

L'exécution du fichier html Nous donne le résultat suivant :



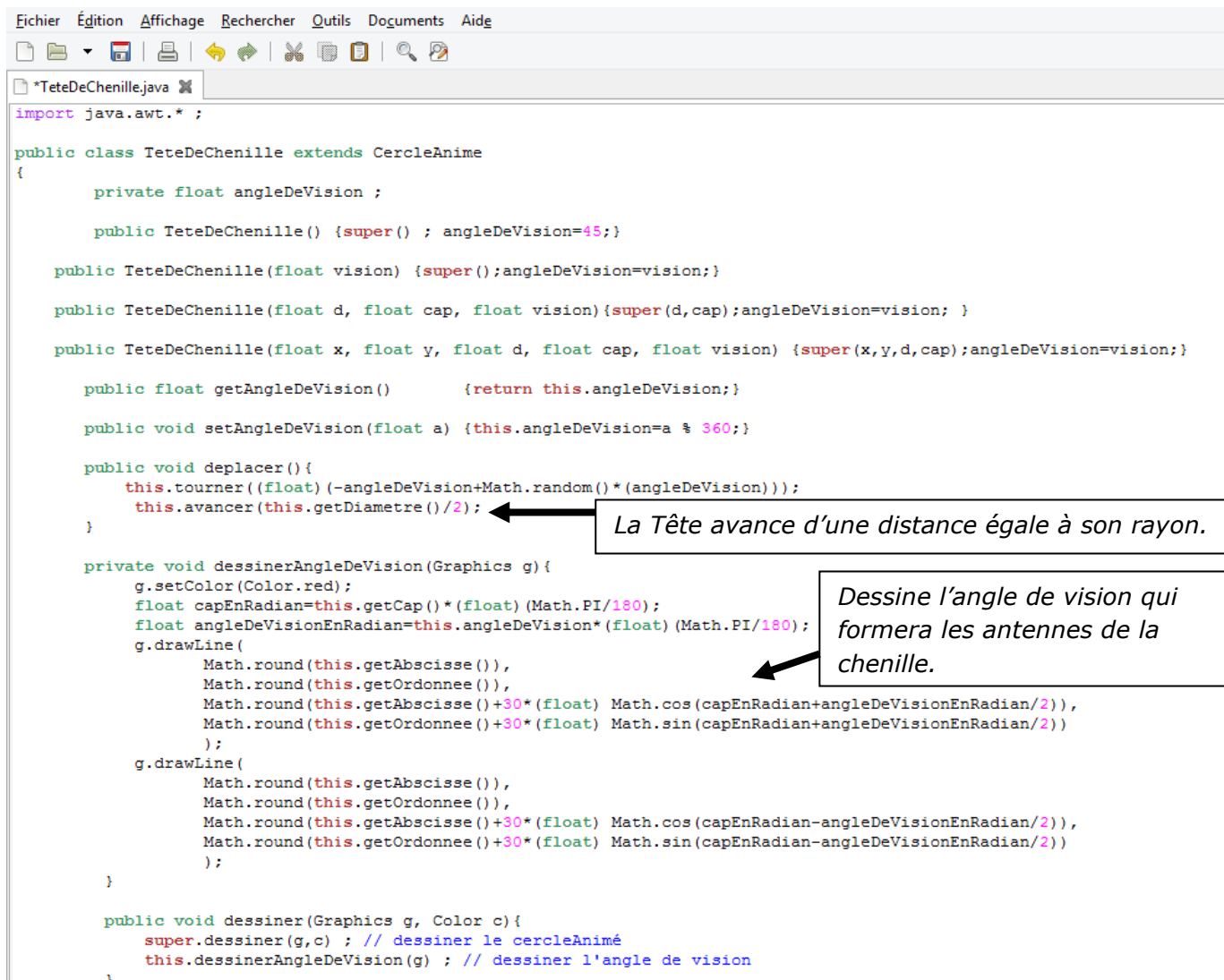
Comme demandé dans les 4 dernières lignes du fichier Test, Le cercle va tout d'abord avancer de 200. Puis, il tournera de 90 degrés avant d'avancer à nouveau de 100.

Voilà donc la première ébauche de ce qui va devenir la tête de notre chenille.

3 – La Tête de la chenille

Jusqu'ici, il n'y avait aucune difficulté notable dans la conception de nos fichiers. Nous allons à présent rentrer dans le vif du sujet en commençant par encoder la tête de la chenille.

A – TeteDeChenille.java



```
import java.awt.* ;

public class TeteDeChenille extends CercleAnime
{
    private float angleDeVision ;

    public TeteDeChenille() {super() ; angleDeVision=45;}

    public TeteDeChenille(float vision) {super();angleDeVision=vision;}

    public TeteDeChenille(float d, float cap, float vision){super(d, cap);angleDeVision=vision; }

    public TeteDeChenille(float x, float y, float d, float cap, float vision) {super(x,y,d, cap);angleDeVision=vision;}

    public float getAngleDeVision()      {return this.angleDeVision;}

    public void setAngleDeVision(float a) {this.angleDeVision=a % 360;}

    public void deplacer(){
        this.tourner((float) (-angleDeVision+Math.random())*(angleDeVision));
        this.avancer(this.getDiametre()/2);
    }

    private void dessinerAngleDeVision(Graphics g){
        g.setColor(Color.red);
        float capEnRadian=this.getCap()*(float) (Math.PI/180);
        float angleDeVisionEnRadian=this.angleDeVision*(float) (Math.PI/180);
        g.drawLine(
            Math.round(this.getAbscisse()),
            Math.round(this.getOrdonnee()),
            Math.round(this.getAbscisse()+30*(float) Math.cos(capEnRadian+angleDeVisionEnRadian/2)),
            Math.round(this.getOrdonnee()+30*(float) Math.sin(capEnRadian+angleDeVisionEnRadian/2))
        );
        g.drawLine(
            Math.round(this.getAbscisse()),
            Math.round(this.getOrdonnee()),
            Math.round(this.getAbscisse()+30*(float) Math.cos(capEnRadian-angleDeVisionEnRadian/2)),
            Math.round(this.getOrdonnee()+30*(float) Math.sin(capEnRadian-angleDeVisionEnRadian/2))
        );
    }

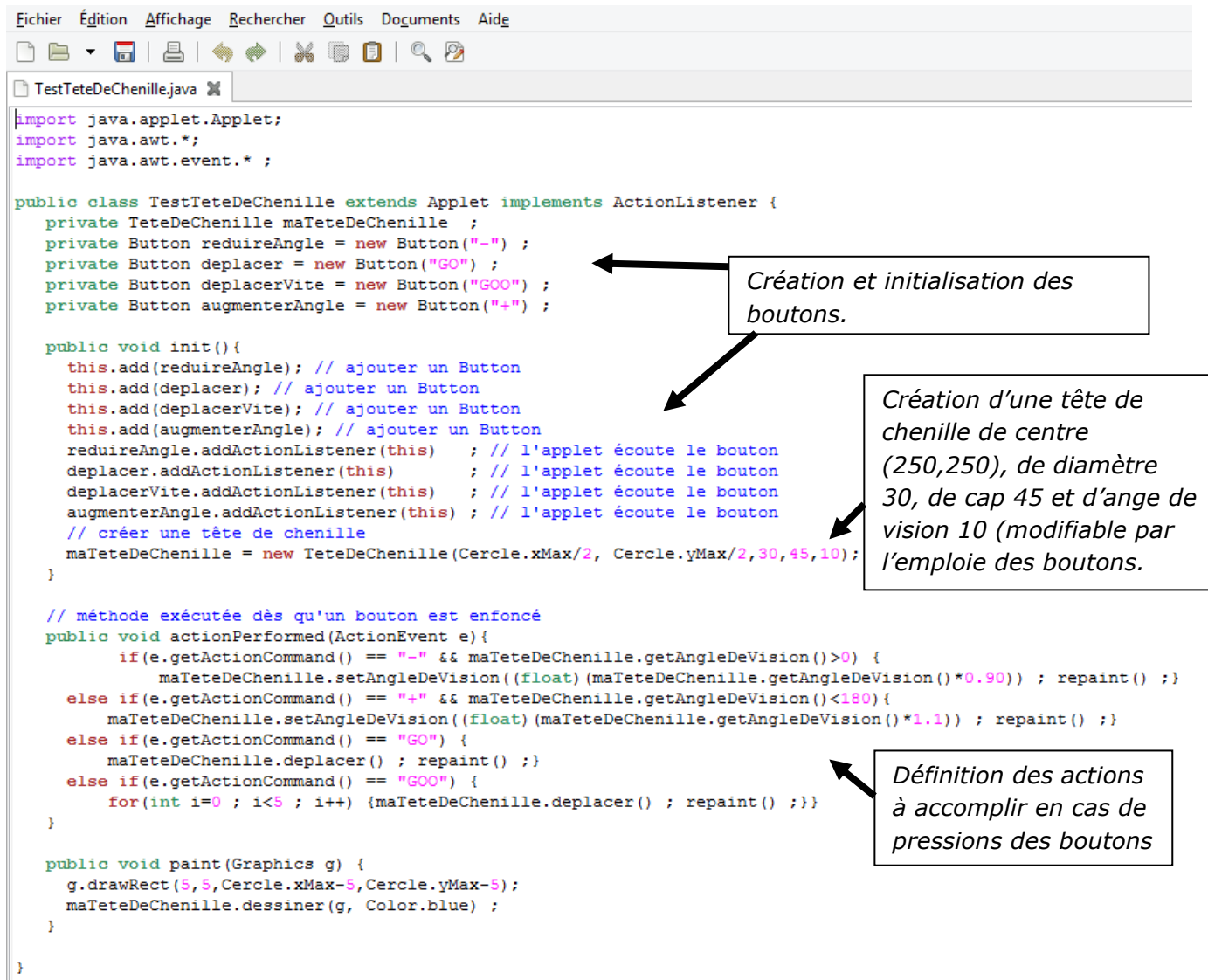
    public void dessiner(Graphics g, Color c){
        super.dessiner(g,c) ; // dessiner le cercleAnimé
        this.dessinerAngleDeVision(g) ; // dessiner l'angle de vision
    }
}
```

La Tête avance d'une distance égale à son rayon.

Dessine l'angle de vision qui formera les antennes de la chenille.

TeteDeChenille est une sous classe de CercleAnime auquel on ajoute un angle de vision qui déterminera la direction de ces déplacements. En effet, la Chenille va se déplacer de manière aléatoire dans son angle de vision. Ces déplacements sont donc imprévisibles dans l'espace défini par $[-\text{AngleDeVision} ; \text{AngleDeVision}]$. Si l'angle de vision vaut 0, la chenille va tout droit.

B – TestTeteDeChenille.java



Nous avons ajouté 4 boutons à notre fenêtre graphique afin de pouvoir «contrôler» la Tête de la chenille à l'aide de l'applet *ActionListener* servant à récupérer pour traitement les actions (prédéfinie effectuées par l'utilisateur :

1. Bouton «-» : Réduit l'angle de vision de 10%
2. Bouton «+» : Augmente l'angle de vision de 10%
3. Bouton «go» : Déplace la chenille d'une distance égale à son rayon
4. Bouton «goo» : Déplace la chenille d'une distance égale à 5 fois son rayon afin de donner une impression de vitesse à l'utilisateur

C – Difficultés et résultats

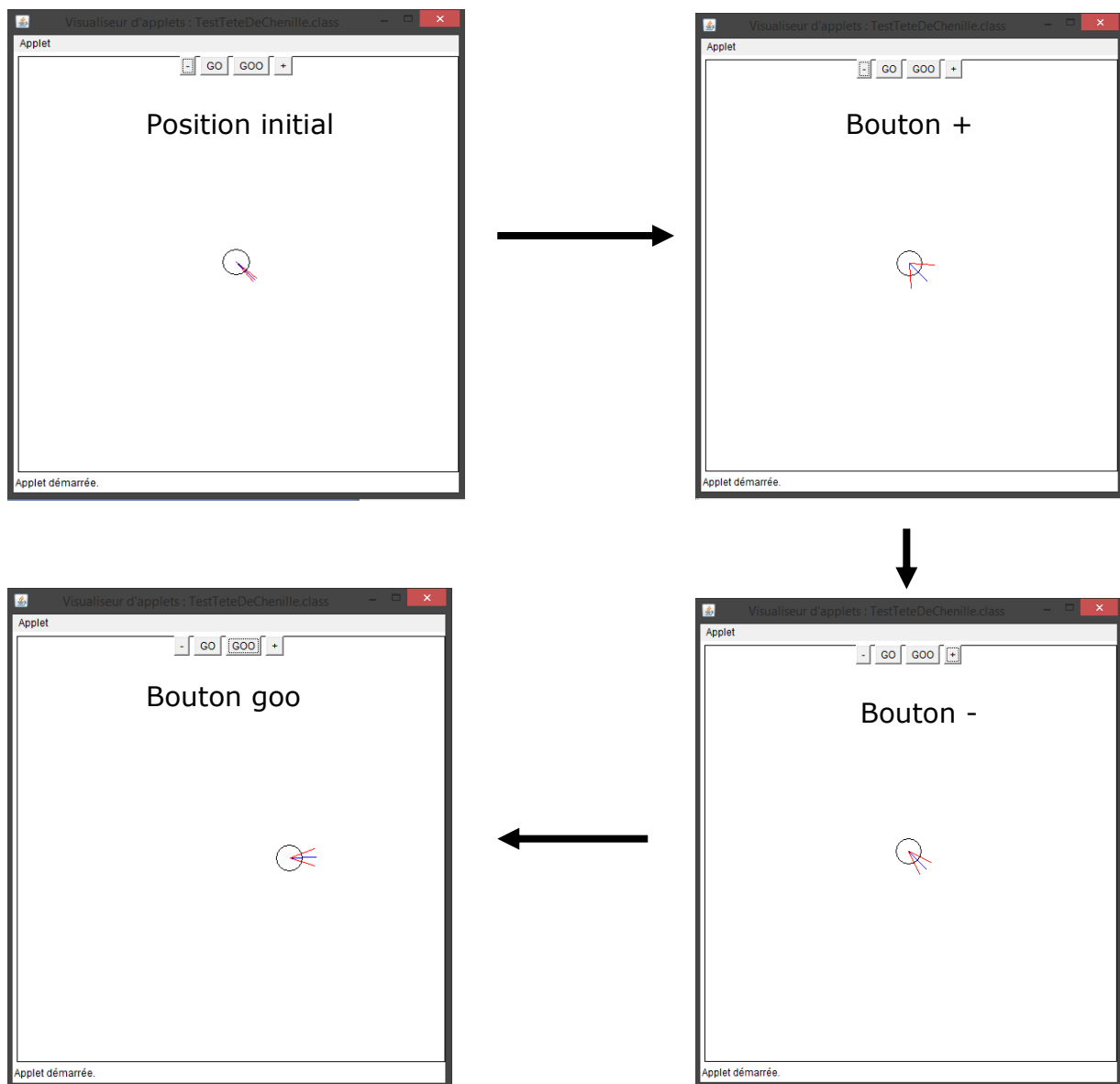
Ici, une des difficultés était d'introduire la notion de déplacement aléatoire en fonction de l'angle de vision.

```
this.tourner((float) (-angleDeVision+Math.random()*(angleDeVision)) );
```

En effet, *Math.random()* nous donne un nombre aléatoire entre 0 et 1. Nous voulons un nombre aléatoire entre $-\text{angleDeVision}$ et angleDeVision qui s'opère grâce à la manipulation effectuée ci-dessus.

Une autre était de mettre en place et définir les boutons de la fenêtre graphique particulièrement pour + et -. Il fallait, avant de redéfinir la valeur de *angleDeVision*, transformer sa nouvelle valeur en nombre flottant par l'utilisation de *float*. Nous avons choisi de définir sa nouvelle valeur à l'aide d'un modulo (choix complètement personnel).

Ci-suit, voici les résultats obtenus une fois le fichier html correspondant exécuté dans l'invite de commande :

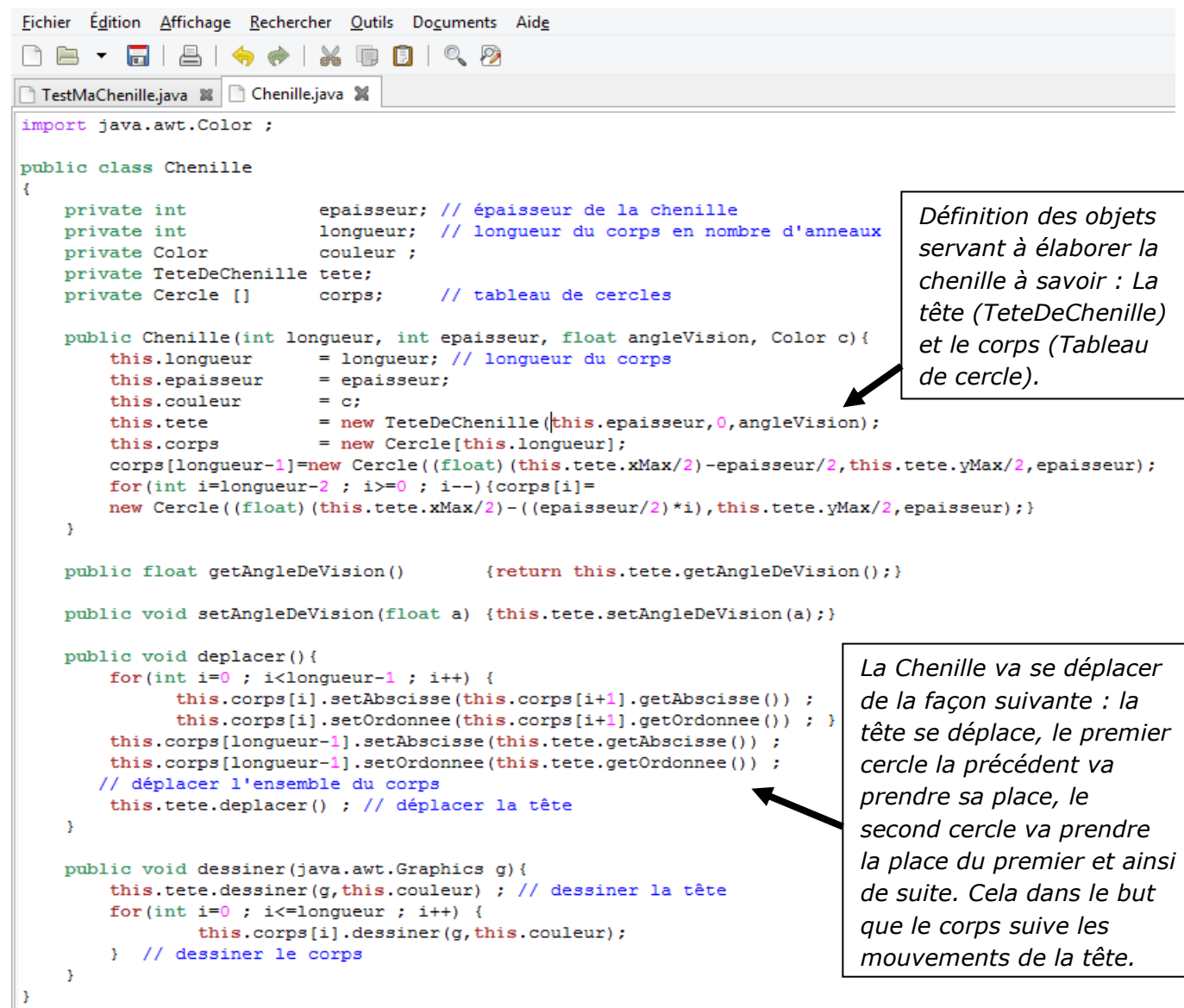


4 – La Chenille

A présent, nous avons en main tous les objets nécessaires à l'encodage de la chenille.

Le corps de la chenille sera formé d'un ensemble de cercle s'emboitant les uns dans les autres par rapport à leurs centres respectifs. Celui-ci sera rattaché à la tête comme défini précédemment.

A – Chenille.java



```
import java.awt.Color ;

public class Chenille
{
    private int          epaisseur; // épaisseur de la chenille
    private int          longueur;  // longueur du corps en nombre d'anneaux
    private Color        couleur ;
    private TeteDeChenille tete;
    private Cercle []    corps;     // tableau de cercles

    public Chenille(int longueur, int epaisseur, float angleVision, Color c){
        this.longueur      = longueur; // longueur du corps
        this.epaisseur     = epaisseur;
        this.couleur       = c;
        this.tete          = new TeteDeChenille(this.epaisseur,0,angleVision);
        this.corps         = new Cercle[this.longueur];
        corps[longueur-1]=new Cercle((float) (this.tete.xMax/2)-epaisseur/2,this.tete.yMax/2,epaisseur);
        for(int i=longueur-2 ; i>=0 ; i--){corps[i]=
            new Cercle((float) (this.tete.xMax/2)-((epaisseur/2)*i),this.tete.yMax/2,epaisseur);}
    }

    public float getAngleDeVision()      {return this.tete.getAngleDeVision();}

    public void setAngleDeVision(float a) {this.tete.setAngleDeVision(a);}

    public void deplacer(){
        for(int i=0 ; i<longueur-1 ; i++) {
            this.corps[i].setAbscisse(this.corps[i+1].getAbscisse()) ;
            this.corps[i].setOrdonnee(this.corps[i+1].getOrdonnee()) ; }
        this.corps[longueur-1].setAbscisse(this.tete.getAbscisse()) ;
        this.corps[longueur-1].setOrdonnee(this.tete.getOrdonnee()) ;
        // déplacer l'ensemble du corps
        this.tete.deplacer() ; // déplacer la tête
    }

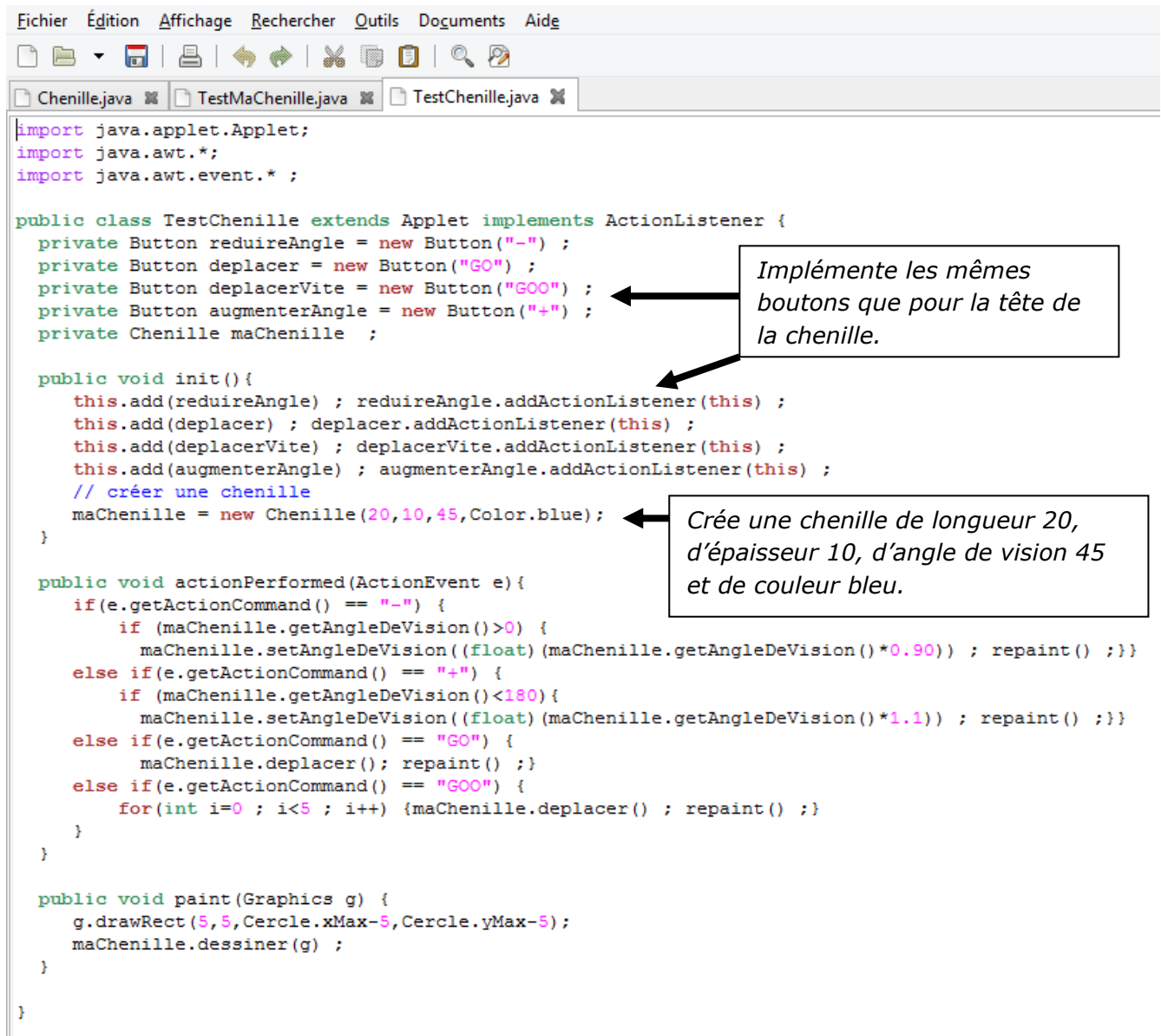
    public void dessiner(java.awt.Graphics g){
        this.tete.dessiner(g,this.couleur) ; // dessiner la tête
        for(int i=0 ; i<=longueur ; i++) {
            this.corps[i].dessiner(g,this.couleur);
        } // dessiner le corps
    }
}
```

Définition des objets servant à élaborer la chenille à savoir : La tête (TeteDeChenille) et le corps (Tableau de cercle).

La Chenille va se déplacer de la façon suivante : la tête se déplace, le premier cercle la précédant va prendre sa place, le second cercle va prendre la place du premier et ainsi de suite. Cela dans le but que le corps suive les mouvements de la tête.

Comme nous pouvons le voir, la Chenille n'est pas une sous classe et elle n'hérite donc d'aucune propriété. Elle va alors réutiliser les objets précédemment définie (TeteDeChenille et Cercle) afin de se former. Sa longueur va dépendre du nombre de cercle qui la compose.

B – TestChenille.java



Comme nous pouvons le remarquer, ce script ressemble étrangement à celui créé pour la tête de la chenille : mêmes boutons, même traitement...

C – Difficultés et résultats

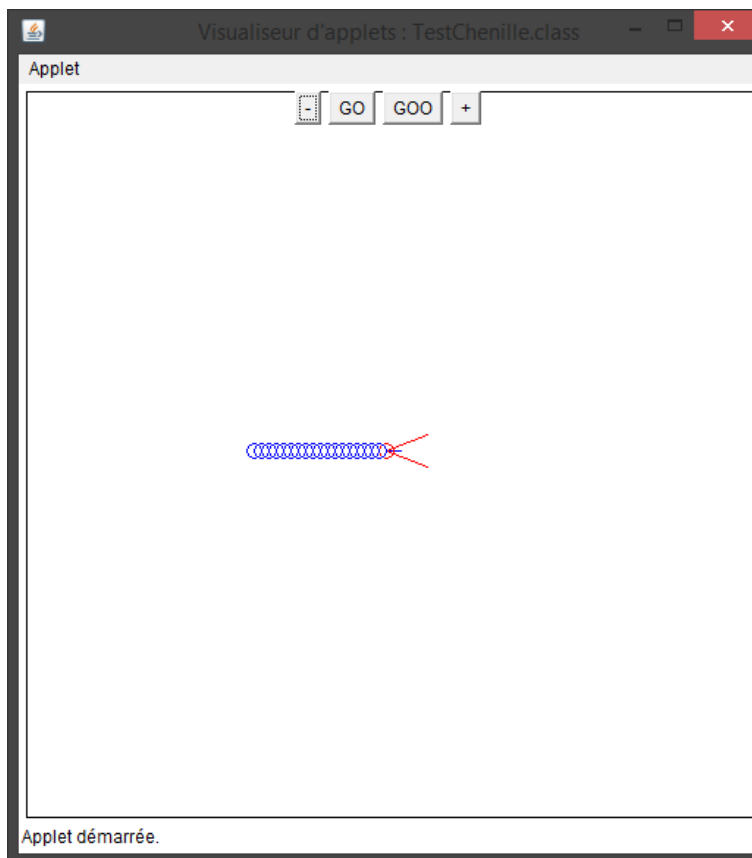
Ici, la grosse difficulté que nous avons rencontrée a été d'encoder le déplacement du corps de telle sorte qu'il suive celui de la chenille. La solution résidait dans le fait que chaque élément prenne la place de celui que le précède.

```
public void deplacer() {  
    for(int i=0 ; i<longueur-1 ; i++) {  
        this.corps[i].setAbscisse(this.corps[i+1].getAbscisse()) ;  
        this.corps[i].setOrdonnee(this.corps[i+1].getOrdonnee()) ; }  
    this.corps[longueur-1].setAbscisse(this.tete.getAbscisse()) ;  
    this.corps[longueur-1].setOrdonnee(this.tete.getOrdonnee()) ;  
    // déplacer l'ensemble du corps  
    this.tete.deplacer() ; // déplacer la tête  
}
```

En effet, il fallait se balader dans les indices du tableau Cercle et changer les abscisses des uns avec celui des autres et faire de même pour les ordonnées.

Toutefois, Notre script contient un léger bug, qui disparaît après deux itérations d'un bouton pour avancer, que nous ne sommes pas arrivés à résoudre. Le corps de la chenille se disloque en deux parties qui finissent par se reformer d'eux même.

Voici ce que nous obtenons en exécutant le fichier html :



Nous obtenons ainsi notre chenille alignée par rapport à la tête qui a comme position initial (250,250) à savoir le centre de la fenêtre graphique.

5 – Extension de la chenille

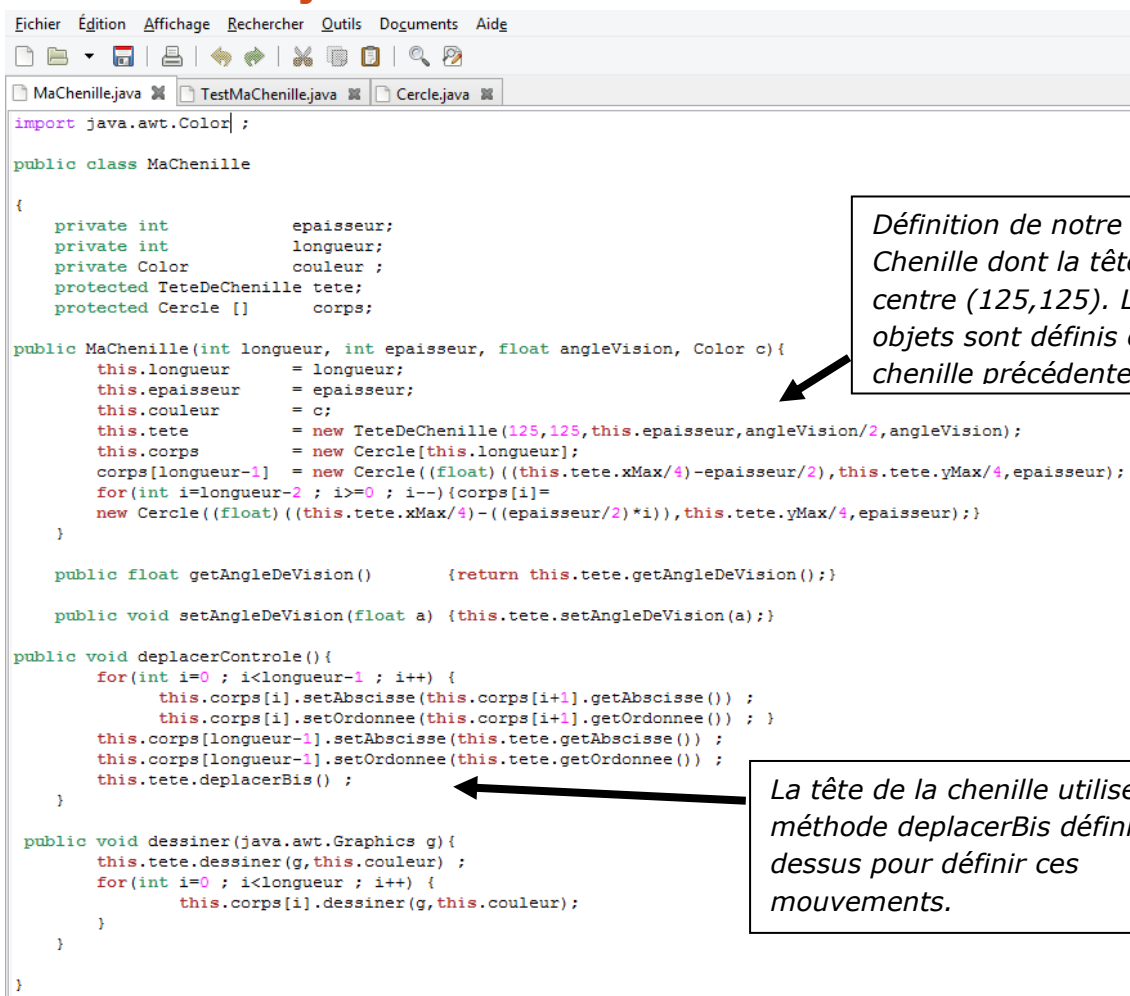
Pour l'extension, nous avons imaginé une sorte de jeu. Sur la fenêtre graphique va s'afficher deux chenilles, une que nous contrôlons totalement et une autre dont les déplacements sont totalement aléatoires (à la façon de notre chenille précédente). Le but étant d'arriver à attraper la chenille hasardeuse.

Afin de pouvoir contrôler les déplacements de la chenille, nous avons du au préalable définir une méthode *déplacerBis* au sein de la classe *TeteDeChenille* tel que :

```
public void deplacerBis() {  
    this.tourner(angleDeVision);  
    this.avancer(this.getDiametre()/2);  
}
```

Comme nous pouvons le voir, *tourner()* n'est plus fonction du hasard mais uniquement de l'angle de vision qui est manipulable avec les boutons – et +.

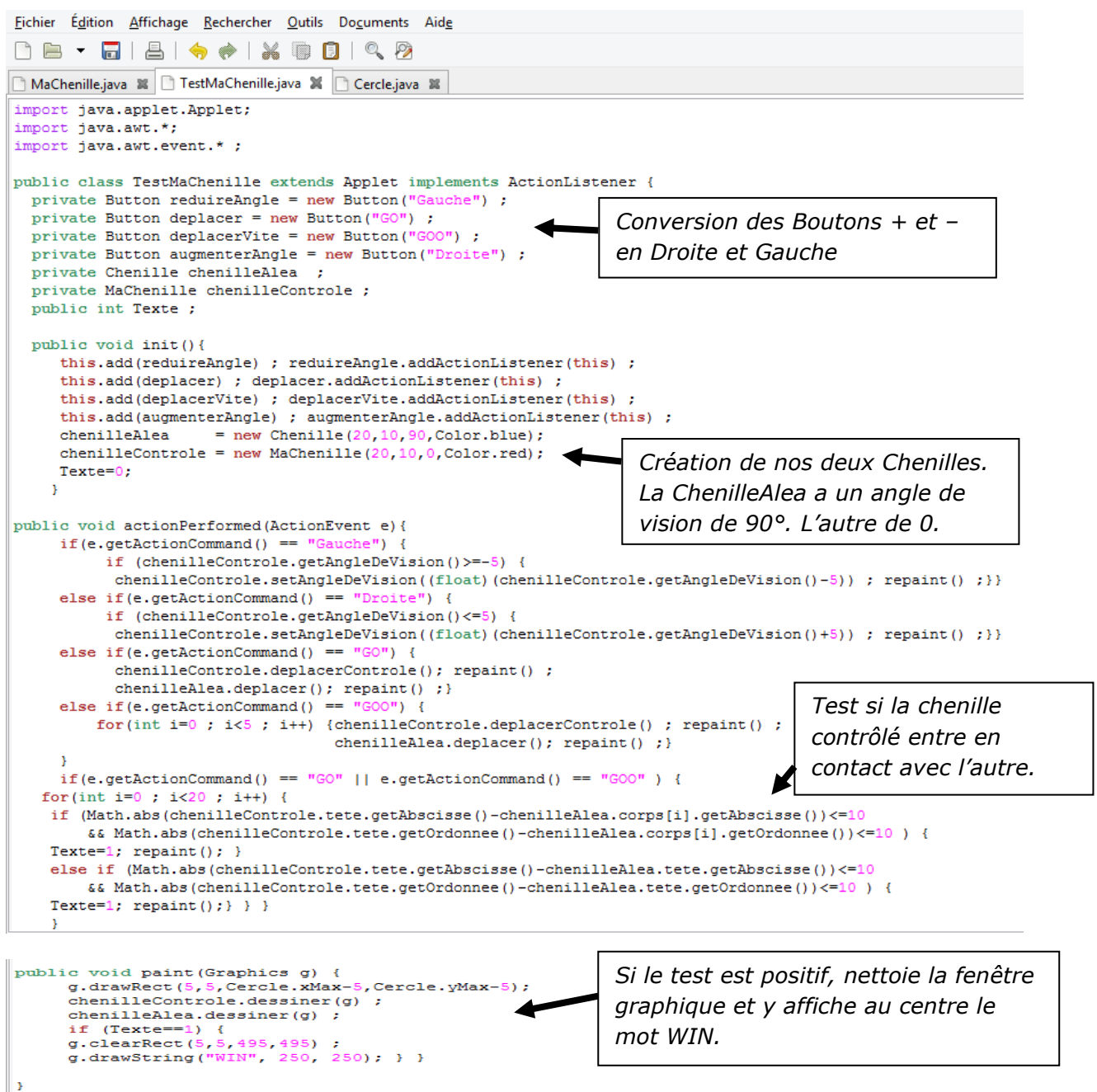
A – MaChenille.java



Afin de définir notre nouvelle chenille contrôlée, nous avons recyclé les éléments de définition de la chenille. Nous avons modifié la position initial de la chenille afin d'éviter que les deux soit superposé. De plus, le déplacement de la tête s'effectue par la méthode *deplacerBis* définie ci-dessus.

Nous avons également du modifier la déclaration en variable privé de tête et corps en variable protégé afin de pouvoir les manipuler dans le fichier test. Manipulation appliqué également à la classe Chenille.

B – TestMaChenille.java



```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class TestMaChenille extends Applet implements ActionListener {
    private Button reduireAngle = new Button("Gauche");
    private Button deplacer = new Button("GO");
    private Button deplacerVite = new Button("GOO");
    private Button augmenterAngle = new Button("Droite");
    private Chenille chenilleAlea;
    private MaChenille chenilleControle;
    public int Texte;

    public void init(){
        this.add(reduireAngle); reduireAngle.addActionListener(this);
        this.add(deplacer); deplacer.addActionListener(this);
        this.add(deplacerVite); deplacerVite.addActionListener(this);
        this.add(augmenterAngle); augmenterAngle.addActionListener(this);
        chenilleAlea = new Chenille(20,10,90,Color.blue);
        chenilleControle = new MaChenille(20,10,0,Color.red);
        Texte=0;
    }

    public void actionPerformed(ActionEvent e){
        if(e.getActionCommand() == "Gauche") {
            if (chenilleControle.getAngleDeVision() > -5) {
                chenilleControle.setAngleDeVision((float)(chenilleControle.getAngleDeVision()-5)); repaint();
            }
        }
        else if (e.getActionCommand() == "Droite") {
            if (chenilleControle.getAngleDeVision() < 5) {
                chenilleControle.setAngleDeVision((float)(chenilleControle.getAngleDeVision()+5)); repaint();
            }
        }
        else if (e.getActionCommand() == "GO") {
            chenilleControle.deplacerControle(); repaint();
            chenilleAlea.deplacer(); repaint();
        }
        else if (e.getActionCommand() == "GOO") {
            for(int i=0; i<5; i++) {chenilleControle.deplacerControle(); repaint();
                                chenilleAlea.deplacer(); repaint();}
        }
        if(e.getActionCommand() == "GO" || e.getActionCommand() == "GOO") {
            for(int i=0; i<20; i++) {
                if (Math.abs(chenilleControle.tete.getAbscisse()-chenilleAlea.corps[i].getAbscisse())<=10
                    && Math.abs(chenilleControle.tete.getOrdonnee()-chenilleAlea.corps[i].getOrdonnee())<=10) {
                    Texte=1; repaint();
                }
                else if (Math.abs(chenilleControle.tete.getAbscisse()-chenilleAlea.tete.getAbscisse())<=10
                    && Math.abs(chenilleControle.tete.getOrdonnee()-chenilleAlea.tete.getOrdonnee())<=10) {
                    Texte=1; repaint();
                }
            }
        }
    }

    public void paint(Graphics g) {
        g.drawRect(5,5,Cercle.xMax-5,Cercle.yMax-5);
        chenilleControle.dessiner(g);
        chenilleAlea.dessiner(g);
        if (Texte==1) {
            g.clearRect(5,5,495,495);
            g.drawString("WIN", 250, 250);
        }
    }
}
```

Conversion des Boutons + et - en Droite et Gauche

Création de nos deux Chenilles. La ChenilleAlea a un angle de vision de 90°. L'autre de 0.

Test si la chenille contrôlé entre en contact avec l'autre.

Si le test est positif, nettoie la fenêtre graphique et y affiche au centre le mot WIN.

Comme nous pouvons le voir, nous avons conservé le même système de boutons que pour la chenille, à la différence près que le bouton - est devenu le bouton Gauche et le bouton + est devenu le bouton Droite.

Afin de pouvoir tourner de façon à contrôler les mouvements, nous avons restreint l'angle de vision dans l'intervalle $[-5 ; 5]$. Le fait de varier l'angle entre négatif et positif permet à la chenille de pouvoir faire demi-tour soit par la droite, soit par la gauche.

Le test mis en place comme ci-dessus vise à détecter si la chenille que nous contrôlons touche soit une partie du corps, soit la tête de l'autre chenille (Si $\text{abs}[\text{l'abscisse de ChenilleControle} - \text{l'abscisse de ChenilleAlea}] \leq 10$ et de même pour l'ordonnée) . Pour ce faire, nous avons déclaré une variable Texte. Celle-ci prend la valeur 0 par défaut. Si le test s'avère positif, elle prend la valeur 1 et la méthode *paint()* est appelé par l'utilisation de *repaint()* qui va afficher la chaîne de caractères WIN.

C – Difficultés et Résultats

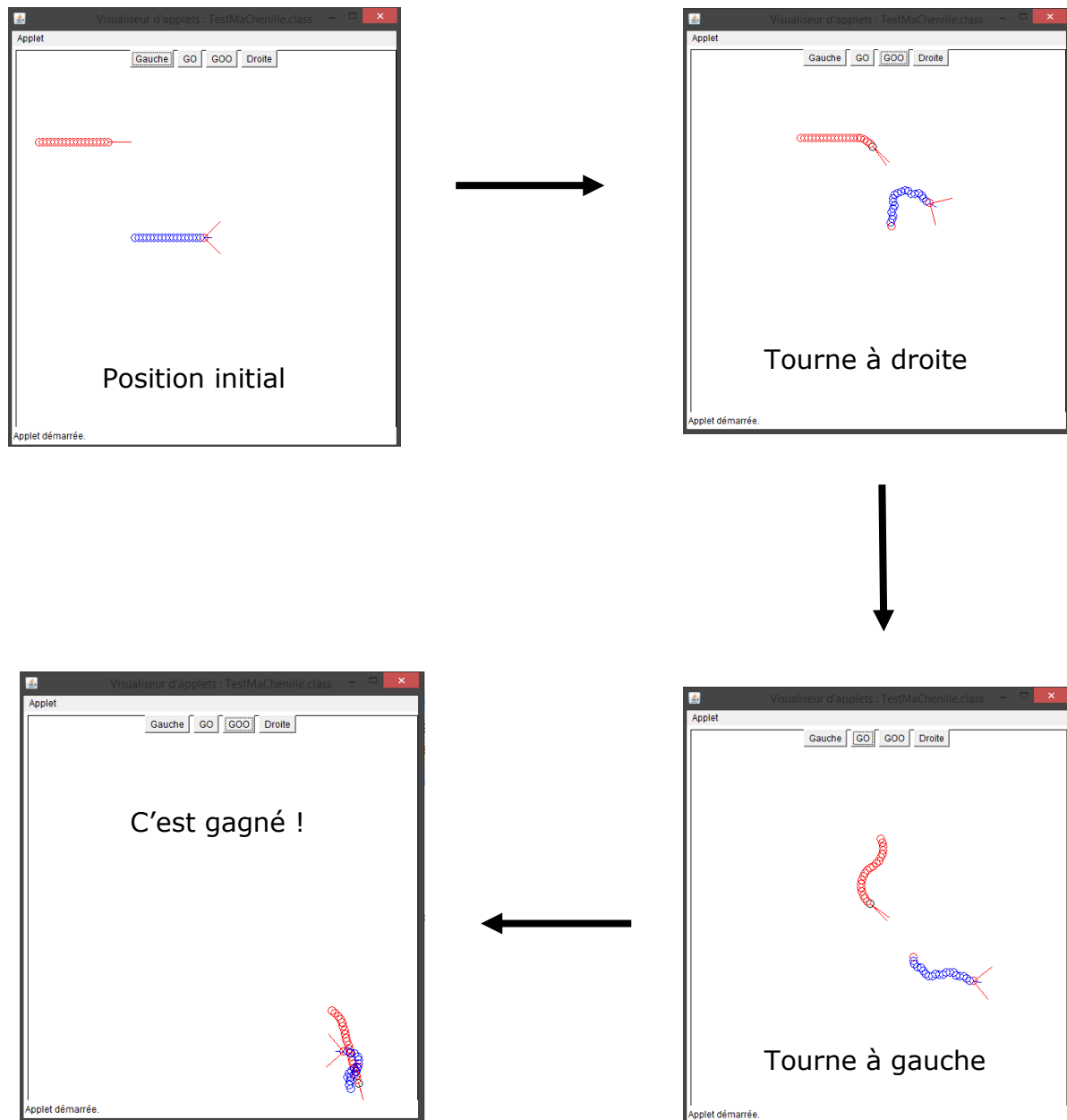
Comme vous vous en êtes probablement aperçu, Notre test présente une erreur et n'affiche pas WIN lorsque l'on touche la chenille. Cela à été notre principale difficulté que nous n'avons malheureusement pas réussi à résoudre. Néanmoins, cela n'affecte pas le bon déroulement du jeu mais peu frustré légèrement l'utilisateur qui n'arrivera jamais à finir la partie.

Nous avons également eu quelques difficultés à positionner correctement la chenille Contrôlé par rapport aux objets qui la compose et à la chenille Aléatoire. Nous avons résolu cela en se servant des propriétés de TêteDeChenille qui est une sous classe de CercleAnime qui est lui-même une sous classe de Cercle qui possède les coordonnées du plan ($x_{\text{Max}}=500$ et $y_{\text{Max}}=500$). Nous avons donc crée notre Tête comme suit.

```
this.tete = new TeteDeChenille(125,125,this.epaisseur,angleVision/2,angleVision);
```

En effet, nous avons choisi de la faire démarrer au point de coordonnée (125,125) qui délimite le bord inférieur droit du quart du plan.

Après exécution du fichier html, nous obtenons le résultat suivant :



6 – Relation d'héritage entre les objets