

# RecuitKP.java

```

1
2 public class RecuitKP {
3     // recuit pour le problÃ¨me du sac Ã dos
4     /*****
5         /* Parametres du recuit */
6         private static final int nbTransitions = 2000;
7         private static final double alpha = 0.995;
8         private static final boolean minimisation = false;
9
10        /****
11        /* dimension du probleme */
12        private static final int DIMENSION = 100;
13
14        // principe d'acceptation
15        private boolean accept(double yi, double yj, double T, boolean min) {
16            boolean isAccepted = false;
17            double dE = yj-yi;
18            double proba = Math.exp(-Math.abs(dE)/T);
19            double tirage = Math.random();
20
21            if(min) {
22                if(dE<0) isAccepted = true;
23                else if (tirage <= proba) isAccepted=true;//acceptation frequente si T
grand car alors proba proche de 1
24            } else { // opt en maximisation
25                if(dE>0) isAccepted = true;
26                else if (tirage <= proba) isAccepted=true;
27            }
28            return isAccepted;
29        }
30        // ****
31        // Chauffage
32        // ****
33        public double chauffage() {
34            int nbAccept = 0;
35            double yi; // critere courant
36            double yj; //critere voisin
37            double T=0.01;
38            double tauxAccept=0.0;
39            EtatKP xi = new EtatKP(DIMENSION);
40
41            do {
42                nbAccept=0;
43                for (int i = 0; i < nbTransitions; i++) {
44                    // generation d'un point de l'espace d'etat
45                    xi.initAleatEtat();
46                    yi = xi.calculCritere();
47                    // generation d'un voisin
48                    xi.genererVoisin();
49                    yj = xi.calculCritere();
50
51                    if(accept(yi,yj,T,minimisation)) nbAccept++;
52
53                    tauxAccept= (double)nbAccept / (double)nbTransitions;
54                }
55                T=1.1*T;
56                System.out.println("T = " + T + " Taux acceptance " + tauxAccept);
57
58            }while (tauxAccept <0.8);
59            return T;
60        } //fin chauffage
61

```

## RecuitKP.java

```

62 // *****
63 // Refroidissement
64 // *****
65 public EtatKP refroidissement(double Tinit) {
66     double yi = 0.0, yj = 0.0; // criteres courant yi et critere voisin mis e 0
67     double T = Tinit;
68     EtatKP xi = new EtatKP(DIMENSION);
69
70     xi.initAleatEtat();
71     yi = xi.calculCritere();
72
73     do {
74
75         for(int i =0; i< nbTransitions;i++) {
76             xi.genererVoisin();
77             yj = xi.calculCritere();
78             if(accept(yi, yj, T, minimisation)) {
79                 yi = yj;
80             }else {
81                 xi.comeBack();
82             }
83         }
84         T = T * alpha;
85         System.out.println("T = " + T + " valeur critere " + yi);
86         // System.out.println(xi.afficherEtat());
87         xi.afficherEtat();
88
89         } while (T > 0.0001 * Tinit || xi.getP()>2000 || xi.getP()<1990);
90     xi.afficherEtat();
91 //     for(int i = 0; i<DIMENSION;i++)
92 //         System.out.println(i + " : "+xi.X[i]+ " ");
93     return xi;
94 }//fin refroidit
95
96 // *****
97 // MAIN
98 // *****
99 public static void main(String args[]) {
100     double temperature;
101     RecuitKP monRecuit = new RecuitKP();
102     // generation des donnees
103     System.out.println("*****Generation des donnees
*****");
104     Data.genererObjets(DIMENSION);
105
106     System.out.println("*****Chauffage *****");
107     temperature = monRecuit.chauffage(); // on recupere la temperature apres
chauffage
108     // i.e. T lorsque le taux d'acceptation est de 0.8
109
110     System.out.println("=====Refroidissement =====");
111     monRecuit.refroidissement(temperature);
112 }//fin main
113 }//fin class Recuit2
114

```

# EtatKP.java

```

1 import java.util.Random;
2 // recuit pour le problème du sac à dos
3 public class EtatKP {
4
5     public int dimEtat;
6     public int[] X;
7
8     private static Random generateur = new Random(999);
9     private int oldIndexI;//, oldIndexJ;
10    //private double oldPoids;
11    private double capacite = 2000;
12    private double P, V;
13
14    /**
15     * Methodes locales */
16
17    //echange deux valeurs du vecteur d'etat X
18    public void swap(int a, int b) {
19        int tmp;
20        tmp = X[a];
21        X[a]=X[b];
22        X[b]=tmp;
23    }
24
25    public double getP()
26    { return P;}
27
28    /** Constructeur */
29
30    public EtatKP(int dimEtat) {
31        this.dimEtat = dimEtat;
32        X = new int[dimEtat];
33    }
34
35    /** Initialisation aleatoire de l'etat */
36
37    public void initAleatEtat() {
38        for(int i= 0 ; i<dimEtat;i++)
39            X[i]=generateur.nextInt(2);
40    }
41
42    /** Evaluation des objectifs */
43
44    public double calculCritere() {
45        double valeur = 0;
46        double poids = 0;
47

```

# EtatKP.java

```

48         for (int i=0; i<dimEtat; ++i){
49             valeur += X[i] * Data.tabValeurs[i];
50             poids += X[i] * Data.tabPoids[i];
51         }
52         V = valeur;
53         P = poids;
54         double delta = poids - capacite;
55         double epsilon = delta / capacite;
56         if (delta <= 0){
57             return valeur;
58         }
59         else{
60             //return valeur/(1+epsilon*epsilon);
61             return valeur -10*delta;
62             //return 0;
63         }
64     }
65
66     /* GenererVoisin */
67
68     public void genererVoisin() {
69         int indexI = generateur.nextInt(dimEtat); // on choisit un indice au hasard entre
0 et dimEtat-1
70         X[indexI]=(X[indexI] + 1) % 2;// si X(i)= 0, 0+1=1%2 donne 1, si X[i]= 1, 1+1=2%2
donne 0
71         oldIndexI=indexI; //on memorise l'index pour pouvoir faire un retour arriere le
cas echeant
72     }
73
74     /* Retour a la solution precedente */
75
76     public void comeBack() {
77         X[oldIndexI] = (X[oldIndexI] + 1) % 2;
78     }
79
80     /* Affichage */
81
82     public void afficherEtat() {
83         System.out.println("poids sac = " + P + " valeur sac = " + V);
84     }
85 }
86

```

# Data.java

```

1 import java.util.*;
2
3 public class Data {
4
5     public static double tabPoids[];
6     public static double tabValeurs[];
7     public static double[][] tabVilles;
8     public static int tabAvions[];
9     private static Random generateur=new Random(123);
10
11
12     /* parametres de generation */
13     private static double RAYON=100.0;
14     private static int POIDS=100;
15     private static int VALEURS=100;
16     /* parametres de generation */
17
18
19
20     public static int separation[][]={{4,5,6},
21                                     {3,3,4},
22                                     {3,3,3}};
23
24
25     // generation de n objets pour le probleme du sac a dos
26     public static void genererObjets(int n) {
27         tabPoids=new double[n];
28         tabValeurs=new double[n];
29         for (int i=0;i<n;i++){
30             tabPoids[i]=(double)(1+generateur.nextInt(POIDS));
31             tabValeurs[i]=(double)(1+generateur.nextInt(VALEURS));
32         }
33     }
34
35     // affichage des objets
36     public static void afficherObjets() {
37         System.out.println("*****
Objets*****");
38         for (int i=0;i<tabPoids.length;i++){
39             System.out.println("Objet " + i + " poids= " + tabPoids[i] + " valeur= " +
tabValeurs[i]);
40         }
41     }
42
43
44     // generation de n villes sur le cercle (TSP)
45     public static void genererVillesCercle(int n) {
46         double theta,x,y;
47         tabVilles=new double[n][2];
48         for (int i=0;i<n;i++){
49             theta=2*Math.PI*generateur.nextDouble();
50             x=RAYON*Math.cos(theta);
51             y=RAYON*Math.sin(theta);
52
53             tabVilles[i][0]=x;
54             tabVilles[i][1]=y;
55         }
56     }
57
58
59     // generation de n villes dans un carre RAYONxRAYON (TSP)
60     public static void genererVilles(int n) {

```

```

61     double x,y;
62
63     tabVilles = new double[n][2];
64     for (int i=0;i<n;i++){
65         x=RAYON*generateur.nextDouble();
66         y=RAYON*generateur.nextDouble();
67         tabVilles[i][0]=x;
68         tabVilles[i][1]=y;
69     }
70 }
71
72 // affichage des villes
73 public static void afficherVilles() {
74     System.out.println("***** Villes
75     *****");
76     for (int i=0;i<tabVilles.length;i++){
77         System.out.println("Ville " + i + " x= " + tabVilles[i][0]+ " y= " +
78         tabVilles[i][1]);
79     }
80
81 // generation de n avions (classe heavy(0), Medium(1), Small(2))
82 public static void genererAvions(int n) {
83     tabAvions=new int[n];
84     for (int i=0;i<n;i++){
85         tabAvions[i]=generateur.nextInt(3);
86     }
87
88 // affichage sequence avions
89 public static void afficherAvions() {
90     System.out.println("***** Avions
91     *****");
92     for (int i=0;i<tabAvions.length;i++){
93         System.out.println("Avion " + i + " classe= " + tabAvions[i]);
94     }
95
96
97 public static void main(String[] args){
98     int dim=100;
99     genererObjets(dim);
100    afficherObjets();
101    genererVillesCercle(dim);
102    afficherVilles();
103    genererVilles(dim);
104    afficherVilles();
105    genererAvions(dim);
106    afficherAvions();
107 }
108 }

```

# RecuitSeq.java

```

1 // sequencement
2 public class RecuitSeq {
3     /******
4     /* Parametres du recuit */
5     private static final int nbTransitions = 2000;
6     private static final double alpha = 0.995;
7     private static final boolean minimisation = true;
8     /******
9     /* dimension du probleme */
10    private static final int DIMENSION = 100;
11
12    // principe d'acceptation
13    private boolean accept(double yi, double yj, double T, boolean min) {
14        boolean isAccepted = false;
15        double dE = yj-yi;
16        double proba = Math.exp(-Math.abs(dE)/T);
17        double tirage = Math.random();
18
19        if(min) {
20            if(dE<0) isAccepted = true;
21            else if (tirage <= proba) isAccepted=true;//acceptation frequente si T grand
22            car alors proba proche de 1
23        } else { // opt en maximisation
24            if(dE>0) isAccepted = true;
25            else if (tirage <= proba) isAccepted=true;
26        }
27        return isAccepted;
28    }
29    // *****
30    // Heat Up
31    // *****
32    public double heatUpLoop() { // HeatUp heat = new HeatUp();
33        int acceptCount = 0;
34        double yi = 0, yj;
35        double T = 0.01, tauxAccept = 0.0;
36        EtatSeq xi = new EtatSeq(DIMENSION);
37
38        do {
39            acceptCount = 0;
40            for (int i = 0; i < nbTransitions; i++) {
41                // generation d'un point de l'espace d'etat
42                xi.initAleatEtat();
43                yi = xi.calculCritere();
44
45                // generation d'un voisin
46                xi.genererVoisin();
47                yj = xi.calculCritere();
48
49                if (accept(yi, yj, T, minimisation))
50                    acceptCount++;
51                tauxAccept = (double) acceptCount / (double) nbTransitions;
52            }
53            T = T * 1.1;
54            System.out.println("T= " + T + " tauxAccept= " + tauxAccept + " currentCost= "
55                + yi);
56        } while (tauxAccept < 0.8);
57        return T;
58    }
59    // *****
60    // COOLING
61    // *****

```

# RecuitSeq.java

```

61 public EtatSeq coolingLoop(double Tinit) { // HeatUp heat = new HeatUp();
62     double yi = 0.0, yj = 0.0;
63     double T = Tinit;
64     EtatSeq xi = new EtatSeq(DIMENSION);
65
66     xi.initAleatEtat();
67     yi = xi.calculCritere();
68     do {
69         for (int i = 0; i < nbTransitions; i++) {
70             xi.genererVoisin();
71             yj = xi.calculCritere();
72             if (accept(yi, yj, T, minimisation)) {
73                 yi = yj;
74             } else {
75                 xi.comeBack();
76             }
77         }
78         T = T * alpha;
79         System.out.println("T= " + T + " valeur critere " + yi);
80     } while (T > 0.0001 * Tinit);
81     xi.afficherEtat();
82     //Data.afficherAvions();
83     return xi;
84 }
85 // *****
86 // MAIN
87 // *****
88 public static void main(String args[]) {
89     double temperature;
90     RecuitSeq monRecuit = new RecuitSeq();
91     // generation des donnees
92     System.out.println("*****Generation des donnee
*****");
93     Data.genererAvions(DIMENSION);
94
95     System.out.println("*****Chauffage *****");
96     temperature = monRecuit.heatUpLoop();
97     System.out.println("=====Refroidissement =====");
98     monRecuit.coolingLoop(temperature);
99 } // end main
100 } // End class HeatUp2
101

```



## Data.java

```
1 import java.util.Random;
2
3 public class Data {
4
5     public static int tabClasseAvions[];
6     private static Random generateur = new Random(123);
7
8     public static int separation[][] = { { 4, 5, 6 }, { 3, 3, 4 }, { 3, 3, 3 } };
9
10    // generation de n avions (classe heavy(0), Medium(1), Small(2))
11    public static void genererAvions(int n) {
12        tabClasseAvions = new int[n];
13        for (int i = 0; i < n; i++) {
14            tabClasseAvions[i] = generateur.nextInt(3);
15        }
16    }
17
18    // affichage des classes des avions
19    public static void afficherClassesAvions() {
20        System.out.println("***** Avions
21        *****");
22        for (int i = 0; i < tabClasseAvions.length; i++) {
23            System.out.println("Avion " + i + " classe= " + tabClasseAvions[i]);
24        }
25    }
```

# EtatSeq.java

```

1 import java.util.Random;
2 //sÃ©quencement
3 public class EtatSeq {
4
5     public int dimEtat;
6     public int[] idAvions;
7     private static Random generateur = new Random(123);
8     private int oldIndexI;
9     private int oldIndexJ;
10    /*****
11     *
12     */
13    public void exchange(int I, int J) {
14        int buffer;
15        buffer = idAvions[I];
16        idAvions[I] = idAvions[J];
17        idAvions[J] = buffer;
18    }
19    /*****
20     *
21     */
22    public EtatSeq(int dimEtat) {
23        this.dimEtat = dimEtat;
24        idAvions = new int[dimEtat];
25    }
26    /*****
27     *
28     */
29    public void initAleatEtat() {
30        for (int i = 0; i < dimEtat; ++i) {
31            idAvions[i] = i;
32        }
33    }
34    /*****
35     *
36     */
37    public void afficherEtat() {
38        System.out.println("rang d'atterrissage ; id avion");
39        for (int i = 0; i < idAvions.length; i++) {
40            System.out.println(i+1 + " ; "+idAvions[i]);
41        }
42    }
43    /*****
44     *
45     */
46    public void genererVoisin() {
47        // public void genererVoisin2() {
48        //     int indexI = generateur.nextInt(dimEtat); // on prends deux index au hasard entre 0
49        //     // et 99
50        //     int indexJ = generateur.nextInt(dimEtat);
51        //     // on permute les avions i.e leur ordre d'atterrissage compris entre ces deux
52        //     // index de maniere symetrique

```

## EtatSeq.java

```

51//         for (int i = 0; i < Math.ceil(Math.abs(indexI - indexJ) / 2); ++i) {
52//             exchange(Math.min(indexI, indexJ) + i, Math.max(indexI, indexJ) - i);
53//         }
54//         oldIndexI = indexI;
55//         oldIndexJ = indexJ;
56//
57//         for (int i = 0; i < dimEtat - 1; ++i) {
58//             if (idAvions[i] <= i - 4 || idAvions[i] >= i + 4) {
59//                 comeBack();
60//                 break;
61//             }
62//         }
63//     }
64
65     public void genererVoisin() {
66         int indexI = generateur.nextInt(dimEtat); // on prends un index au hasard entre 0
67         et 99 (dimEtat=100=DIMENSION)
68         int indexJ = indexI - 4 + generateur.nextInt(9); // on g n re un indexJ dans
69         indexI +/- 4 pour tenir compte de la
70         //contrainte de vitesse ; un avion ne peut changer de rang au d  l   de +/- 4/
71         if (indexJ < 0) indexJ = 0;
72         if (indexJ >= dimEtat) indexJ = dimEtat - 1;
73         // on permute les avions i.e leur ordre d'atterrissage compris entre ces deux index
74         de maniere symetrique
75         for (int i = 0; i < Math.ceil(Math.abs(indexI - indexJ) / 2); ++i) {
76             exchange(Math.min(indexI, indexJ) + i, Math.max(indexI, indexJ) - i);
77         }
78         oldIndexI = indexI;
79         oldIndexJ = indexJ;
80     }
81
82     /* Retour a la solution pr  c  dente */
83     /*
84     *****
85     */
86     public void comeBack() {
87         for (int i = 0; i < Math.ceil(Math.abs(oldIndexI - oldIndexJ) / 2); ++i) {
88             exchange(Math.min(oldIndexI, oldIndexJ) + i, Math.max(oldIndexI, oldIndexJ) -
89             i);
90         }
91     }
92
93     /* Evaluation des objectifs */
94     /*
95     *****
96     */
97     public double calculCritere() {
98         double cost = 0;
99         // tabClasseAvions contient les classes des 100 avions (0 : Heavy, 1 : Medium, 2 :
100         Light),
101         // l'index de tabAvions correspond   l'identifiant avion de 0   99
102         //idAvions contient les avions identi  s par un chiffre de 0   99
103         // les index du tableau idAvions donne l'ordre d'atterrissage : idAvion[0] donne
104         l'id de l'avion qui atterrit en premier.
105         for (int i = 0; i < dimEtat - 1; ++i) {
106             cost += Data.separation[Data.tabClasseAvions[idAvions[i]]]
107             [Data.tabClasseAvions[idAvions[i + 1]]];
108         }
109         return cost;
110     }
111 }

```



# RecuitTSP.java

```

1 // TSP
2 public class RecuitTSP {
3     /******
4     /* Parametres du recuit */
5     private static final int nbTransitions=2000;
6     private static final double alpha=0.995;
7     private static final boolean minimisation=true;
8     /******
9     /* dimension du probleme */
10    private static final int DIMENSION=100;
11
12    // principe d'acceptation
13    private boolean accept(double yi, double yj, double T, boolean min) {
14        boolean isAccepted = false;
15        double dE = yj-yi;
16        double proba = Math.exp(-Math.abs(dE)/T);
17        double tirage = Math.random();
18
19        if(min) {
20            if(dE<0) isAccepted = true;
21            else if (tirage <= proba) isAccepted=true;//acceptation frequente si T
grand car alors proba proche de 1
22        } else { // opt en maximisation
23            if(dE>0) isAccepted = true;
24            else if (tirage <= proba) isAccepted=true;
25        }
26        return isAccepted;
27    }
28    /******
29    // Heat Up
30    /******
31    public double heatUpLoop()
32    { //HeatUp heat = new HeatUp();
33        int acceptCount = 0;
34        double yi=0,yj;
35        double T=0.01, tauxAccept=0.0;
36        EtatTSP xi=new EtatTSP(DIMENSION);
37        do {
38            acceptCount=0;
39            for (int i=0; i< nbTransitions;i++)
40            {
41                //generation d'un point de l'espace d'etat
42                xi.initAleatEtat();
43                yi=xi.calculCritere();
44
45                //generation d'un voisin
46                xi.genererVoisin();
47                yj=xi.calculCritere();
48
49                if (accept(yi,yj,T,minimisation)) acceptCount++;
50                tauxAccept=(double)acceptCount/(double)nbTransitions;
51            }
52            T=T*1.1;
53            System.out.println("T= " + T + " tauxAccept= " + tauxAccept + " currentCost= "
+ yi);
54        } while(tauxAccept<0.8);
55        return T;
56    }
57    /******
58    // COOLING
59    /******
60    public EtatTSP coolingLoop(double Tinit)

```

# RecuitTSP.java

```

61  { //HeatUp heat = new HeatUp();
62    double yi=0.0,yj=0.0;
63    double T=Tinit;
64    EtatTSP xi=new EtatTSP(DIMENSION);
65
66    xi.initAleatEtat();
67    yi=xi.calculCritere();
68    do {
69        for (int i=0; i< nbTransitions;i++)
70        {
71            xi.genererVoisin();
72            yj=xi.calculCritere();
73            if (accept(yi,yj,T,minimisation)){
74                yi=yj;
75            }
76            else
77            {
78                xi.comeBack();
79            }
80        }
81        T=T*alpha;
82        System.out.println("T= " + T + " valeur critere " + yi);
83        System.out.println(xi.afficherEtat());
84    } while(T>0.0001*Tinit);
85    return xi;
86 }
87 //*****
88 //      MAIN
89 //*****
90 public static void main( String args[] )
91 {
92     double temperature;
93     RecuitTSP monRecuit = new RecuitTSP();
94     //generation des donnees
95     System.out.println("*****Generation des donnee
*****");
96     Data.genererVillesCercle(DIMENSION);
97     //Data.genererVilles(DIMENSION);
98
99     System.out.println("*****Chauffage *****");
100    temperature = monRecuit.heatUpLoop();
101    System.out.println("=====Refroidissement =====");
102    monRecuit.coolingLoop(temperature);
103 } //end main
104 } //End class HeatUp2
105

```

## EtatTSP.java

```

1 import java.util.*;
2 // TSP
3 public class EtatTSP {
4     public int dimEtat;
5     public int[] vecteur;
6
7     private static Random generateur = new Random(123);
8     private int oldIndexI;
9     private int oldIndexJ;
10    private static final int methode =2;
11
12    /**
13     *
14     *
15     *
16     *
17     *
18     *
19     *
20     *
21     *
22     *
23     *
24     *
25     *
26     *
27     *
28     *
29     *
30     *
31     *
32     *
33     *
34     *
35     *
36     *
37     *
38     *
39     *
40     *
41     *
42     *
43     *
44     *
45     *
46     *
47     *
48     *
49     *
50     *
51     *
52     *
53     *
54     *
55     *
56     *
57     *
58     *
59     *
60     *
61     *
62     *
63     *
64     *
65     *
66     *
67     *
68     *
69     *
70     *
71     *
72     *
73     *
74     *
75     *
76     *
77     *
78     *
79     *
80     *
81     *
82     *
83     *
84     *
85     *
86     *
87     *
88     *
89     *
90     *
91     *
92     *
93     *
94     *
95     *
96     *
97     *
98     *
99     *
100    */
101    /**
102     *
103     *
104     *
105     *
106     *
107     *
108     *
109     *
110     *
111     *
112     *
113     *
114     *
115     *
116     *
117     *
118     *
119     *
120     *
121     *
122     *
123     *
124     *
125     *
126     *
127     *
128     *
129     *
130     *
131     *
132     *
133     *
134     *
135     *
136     *
137     *
138     *
139     *
140     *
141     *
142     *
143     *
144     *
145     *
146     *
147     *
148     *
149     *
150     *
151     *
152     *
153     *
154     *
155     *
156     *
157     *
158     *
159     *
160     *
161     *
162     *
163     *
164     *
165     *
166     *
167     *
168     *
169     *
170     *
171     *
172     *
173     *
174     *
175     *
176     *
177     *
178     *
179     *
180     *
181     *
182     *
183     *
184     *
185     *
186     *
187     *
188     *
189     *
190     *
191     *
192     *
193     *
194     *
195     *
196     *
197     *
198     *
199     *
200     *
201     *
202     *
203     *
204     *
205     *
206     *
207     *
208     *
209     *
210     *
211     *
212     *
213     *
214     *
215     *
216     *
217     *
218     *
219     *
220     *
221     *
222     *
223     *
224     *
225     *
226     *
227     *
228     *
229     *
230     *
231     *
232     *
233     *
234     *
235     *
236     *
237     *
238     *
239     *
240     *
241     *
242     *
243     *
244     *
245     *
246     *
247     *
248     *
249     *
250     *
251     *
252     *
253     *
254     *
255     *
256     *
257     *
258     *
259     *
260     *
261     *
262     *
263     *
264     *
265     *
266     *
267     *
268     *
269     *
270     *
271     *
272     *
273     *
274     *
275     *
276     *
277     *
278     *
279     *
280     *
281     *
282     *
283     *
284     *
285     *
286     *
287     *
288     *
289     *
290     *
291     *
292     *
293     *
294     *
295     *
296     *
297     *
298     *
299     *
300     *
301     *
302     *
303     *
304     *
305     *
306     *
307     *
308     *
309     *
310     *
311     *
312     *
313     *
314     *
315     *
316     *
317     *
318     *
319     *
320     *
321     *
322     *
323     *
324     *
325     *
326     *
327     *
328     *
329     *
330     *
331     *
332     *
333     *
334     *
335     *
336     *
337     *
338     *
339     *
340     *
341     *
342     *
343     *
344     *
345     *
346     *
347     *
348     *
349     *
350     *
351     *
352     *
353     *
354     *
355     *
356     *
357     *
358     *
359     *
360     *
361     *
362     *
363     *
364     *
365     *
366     *
367     *
368     *
369     *
370     *
371     *
372     *
373     *
374     *
375     *
376     *
377     *
378     *
379     *
380     *
381     *
382     *
383     *
384     *
385     *
386     *
387     *
388     *
389     *
390     *
391     *
392     *
393     *
394     *
395     *
396     *
397     *
398     *
399     *
400     *
401     *
402     *
403     *
404     *
405     *
406     *
407     *
408     *
409     *
410     *
411     *
412     *
413     *
414     *
415     *
416     *
417     *
418     *
419     *
420     *
421     *
422     *
423     *
424     *
425     *
426     *
427     *
428     *
429     *
430     *
431     *
432     *
433     *
434     *
435     *
436     *
437     *
438     *
439     *
440     *
441     *
442     *
443     *
444     *
445     *
446     *
447     *
448     *
449     *
450     *
451     *
452     *
453     *
454     *
455     *
456     *
457     *
458     *
459     *
460     *
461     *
462     *
463     *
464     *
465     *
466     *
467     *
468     *
469     *
470     *
471     *
472     *
473     *
474     *
475     *
476     *
477     *
478     *
479     *
480     *
481     *
482     *
483     *
484     *
485     *
486     *
487     *
488     *
489     *
490     *
491     *
492     *
493     *
494     *
495     *
496     *
497     *
498     *
499     *
500     *
501     *
502     *
503     *
504     *
505     *
506     *
507     *
508     *
509     *
510     *
511     *
512     *
513     *
514     *
515     *
516     *
517     *
518     *
519     *
520     *
521     *
522     *
523     *
524     *
525     *
526     *
527     *
528     *
529     *
530     *
531     *
532     *
533     *
534     *
535     *
536     *
537     *
538     *
539     *
540     *
541     *
542     *
543     *
544     *
545     *
546     *
547     *
548     *
549     *
550     *
551     *
552     *
553     *
554     *
555     *
556     *
557     *
558     *
559     *
560     *
561     *
562     *
563     *
564     *
565     *
566     *
567     *
568     *
569     *
570     *
571     *
572     *
573     *
574     *
575     *
576     *
577     *
578     *
579     *
580     *
581     *
582     *
583     *
584     *
585     *
586     *
587     *
588     *
589     *
590     *
591     *
592     *
593     *
594     *
595     *
596     *
597     *
598     *
599     *
600     *
601     *
602     *
603     *
604     *
605     *
606     *
607     *
608     *
609     *
610     *
611     *
612     *
613     *
614     *
615     *
616     *
617     *
618     *
619     *
620     *
621     *
622     *
623     *
624     *
625     *
626     *
627     *
628     *
629     *
630     *
631     *
632     *
633     *
634     *
635     *
636     *
637     *
638     *
639     *
640     *
641     *
642     *
643     *
644     *
645     *
646     *
647     *
648     *
649     *
650     *
651     *
652     *
653     *
654     *
655     *
656     *
657     *
658     *
659     *
660     *
661     *
662     *
663     *
664     *
665     *
666     *
667     *
668     *
669     *
670     *
671     *
672     *
673     *
674     *
675     *
676     *
677     *
678     *
679     *
680     *
681     *
682     *
683     *
684     *
685     *
686     *
687     *
688     *
689     *
690     *
691     *
692     *
693     *

```

## EtatTSP.java

```

43     }
44
45     /*
46     /*
47     public void genererVoisin()
48     {
49         int indexI = generateur.nextInt(dimEtat);
50         int indexJ = generateur.nextInt(dimEtat);
51
52         if(methode ==1)
53             exchange(indexI, indexJ); /**1er methode pour changer la position
54             aleatoirement
55             if (methode ==2){
56                 for (int i=0; i<Math.ceil(Math.abs(indexI-indexJ)/2); ++i){
57                     exchange(Math.min(indexI, indexJ)+i, Math.max(indexI, indexJ)-i);
58                     //System.out.println(i+ " " + (Math.min(indexI, indexJ)+i) + " " +
59                     (Math.max(indexI, indexJ)-i) );
60                 }
61             }
62             oldIndexI = indexI;
63             oldIndexJ = indexJ;
64         }
65
66     /*
67     /*
68     public void comeBack()
69     {
70         if(methode ==1) exchange(oldIndexI, oldIndexJ); /**1er methode pour changer la
71         position aleatoirement
72         if (methode ==2){
73             for (int i=0; i<Math.ceil(Math.abs(oldIndexI-oldIndexJ)/2); ++i){
74                 exchange(Math.min(oldIndexI, oldIndexJ)+i, Math.max(oldIndexI,
75                 oldIndexJ)-i);
76             }
77         }
78
79     /*
80     /*
81     public double calculCritere(){
82         double cost=0;
83         double dx,dy;
84         for (int i=0; i<dimEtat-1; ++i){
85             dx = Data.tabVilles[vecteur[i+1]][0]-Data.tabVilles[vecteur[i]][0];
86             dy = Data.tabVilles[vecteur[i+1]][1]-Data.tabVilles[vecteur[i]][1];
87             cost += Math.sqrt(dx*dx+dy*dy);
88         }

```



# EtatTSP.java

```
86     dx = Data.tabVilles[vecteur[dimEtat-1]][0]-Data.tabVilles[vecteur[0]][0];
87     dy = Data.tabVilles[vecteur[dimEtat-1]][1]-Data.tabVilles[vecteur[0]][1];
88     cost += Math.sqrt(dx*dx+dy*dy);
89     return cost;
90 }
91 }//fin EtatTSP
92
93
94
```

# Data.java

```

1 import java.util.*;
2
3 public class Data {
4
5     public static double tabPoids[];
6     public static double tabValeurs[];
7     public static double[][] tabVilles;
8     public static int tabAvions[];
9     private static Random generateur=new Random(123);
10
11
12     /** *****/
13     /*  parametres de generation  */
14     private static double RAYON=100.0;
15     private static int POIDS=100;
16     private static int VALEURS=100;
17     /** *****/
18
19
20     public static int separation[][]={{4,5,6},
21                                     {3,3,4},
22                                     {3,3,3}};
23
24
25     // generation de n objets pour le probleme du sac a dos
26     public static void genererObjets(int n) {
27         tabPoids=new double[n];
28         tabValeurs=new double[n];
29         for (int i=0;i<n;i++){
30             tabPoids[i]=(double)(1+generateur.nextInt(POIDS));
31             tabValeurs[i]=(double)(1+generateur.nextInt(VALEURS));
32         }
33     }
34
35     // affichage des objets
36     public static void afficherObjets() {
37         System.out.println("*****
Objets*****");
38         for (int i=0;i<tabPoids.length;i++){
39             System.out.println("Objet " + i + " poids= " + tabPoids[i] + " valeur= " +
tabValeurs[i]);
40         }
41     }
42
43
44     // generation de n villes sur le cercle (TSP)
45     public static void genererVillesCercle(int n) {
46         double theta,x,y;
47         tabVilles=new double[n][2];
48         for (int i=0;i<n;i++){
49             theta=2*Math.PI*generateur.nextDouble();
50             x=RAYON*Math.cos(theta);
51             y=RAYON*Math.sin(theta);
52
53             tabVilles[i][0]=x;
54             tabVilles[i][1]=y;
55         }
56     }
57
58
59     // generation de n villes dans un carre RAYONxRAYON (TSP)
60     public static void genererVilles(int n) {

```

```

61     double x,y;
62
63     tabVilles = new double[n][2];
64     for (int i=0;i<n;i++){
65         x=RAYON*generateur.nextDouble();
66         y=RAYON*generateur.nextDouble();
67         tabVilles[i][0]=x;
68         tabVilles[i][1]=y;
69     }
70 }
71
72 // affichage des villes
73 public static void afficherVilles() {
74     System.out.println("***** Villes
75     *****");
76     for (int i=0;i<tabVilles.length;i++){
77         System.out.println("Ville " + i + " x= " + tabVilles[i][0]+ " y= " +
78         tabVilles[i][1]);
79     }
80
81 // generation de n avions (classe heavy(0), Medium(1), Small(2))
82 public static void genererAvions(int n) {
83     tabAvions=new int[n];
84     for (int i=0;i<n;i++){
85         tabAvions[i]=generateur.nextInt(3);
86     }
87
88 // affichage sequence avions
89 public static void afficherAvions() {
90     System.out.println("***** Avions
91     *****");
92     for (int i=0;i<tabAvions.length;i++){
93         System.out.println("Avion " + i + " classe= " + tabAvions[i]);
94     }
95 }
96
97 public static void main(String[] args){
98     int dim=100;
99     genererObjets(dim);
100    afficherObjets();
101    genererVillesCercle(dim);
102    afficherVilles();
103    genererVilles(dim);
104    afficherVilles();
105    genererAvions(dim);
106    afficherAvions();
107 }
108 }

```