Examen (session 1)

7 janvier 2015 - Durée 2h

Documents autorisés : "Fiche traduction Algo-ADA" et "Mémento ADA"

Pour l'ensemble du sujet, nous nous plaçons dans le cadre du ***TP Robot***.
Les annexes présentent différents paquetages et programmes utilisés lors de ce ***TP Robot***.

Les différentes parties sont indépendantes les unes des autres et peuvent être traitées dans un ordre quelconque.

Une partie cachée ou non utile est indiquée par `...`

Une partie à compléter est indiquée par `-- A COMPLETER --`

Penser à utiliser les numéros de ligne pour indiquer les modifications et ajouts éventuels.
Par exemple, si vous voulez utiliser sans modification une partie de code source donnée en annexe, indiquez simplement les numéros de ligne correspondants pour éviter d'avoir à la recopier.

Partie 1 - Génération de terrains et test de performance d'un automate

[*barême indicatif : 5 pts*]

Au lieu d'avoir deux programmes ***generation_terrains*** (cf. annexe 6 - pages 11 à 12) et ***test_performance*** (dans sa version sans interface graphique) (cf. annexe 8 - page 13) , on souhaite avoir un seul programme qui effectue à la fois la génération aléatoire de terrains et le test de performance d'un automate, ce programme s'appellera ***gttp***.
Ce programme devra être appelé avec 6 arguments :

gttp *nb_terrains largeur hauteur pourcentage_cases_occupees fichier_automate fichier_resultat*

Question 1-1 :

Ecrivez un tel programme en vous aidant des codes sources existants.

Question 1-2 :

Complétez le fichier ***Makefile*** (cf. annexe 7 - page 12) afin de pouvoir créer l'exécutable ***gttp***.

Partie 2 - Changement de configuration des terrains

[*barême indicatif : 11 pts*]

Dans cette partie, on considère des terrains différents de ceux utilisés dans le ***TP Robot***.

Un terrain sera défini par :
– une largeur L impaire avec $5 \leq L \leq DIM\_MAX$,
– une hauteur H avec $3 \leq H \leq DIM\_MAX$,
– et un pourcentage d'obstacle p (entier entre 0 et 100).

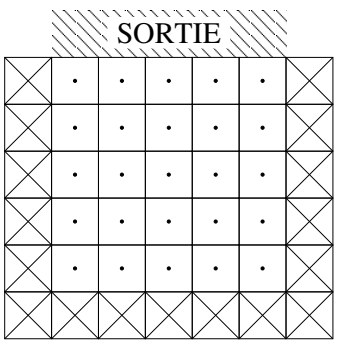


Figure 1

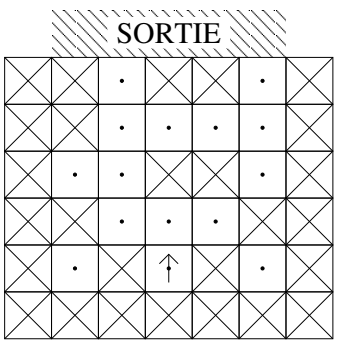


Figure 2

```
7 6
XX.XX.X
XX....X
X..XX.X
XX...XX
X.X.X.X
XXXXXXX
```

Figure 3

Un terrain *vide* consiste en un terrain dont les colonnes 1 et L ainsi que la ligne H sont formées d'obstacles, ces 3 rangées à l'Ouest, à l'Est et au Sud du terrain forment le ***mur*** du terrain.
Par la suite, les obstacles du ***mur*** ne sont pas pris en compte dans le pourcentage d'obstacles à placer.
La figure 1 ci-dessus montre un terrain vide de dimensions $L = 7$ et $H = 6$.

On souhaite modifier le paquetage ***terrain_paq*** (cf. annexe 1 - pages 4 à 6) utilisé lors du ***TP Robot*** mais sans modifier le fichier spécification *terrain_paq.ads* (cf. annexe 1 - page 4) .

Question 2-1 :

Récrivez le corps de la fonction *creer_terrain_vide* (cf. annexe 1 - page 5 - ligne 11) suivant les spécifications ci-dessus.

La position initiale du robot est la case $(x = (L + 1)/2, y = H - 1)$ et son orientation initiale est *Nord*.
Comme dans le ***TP Robot***, le robot doit parvenir à sortir ; dans le cas de ces terrains, la sortie correspond à la partie au Nord du terrain.

La figure 2 ci-dessus montre un terrain de dimensions $L = 7$ et $H = 6$ avec 10 obstacles *hors mur* (soit un pourcentage de 40%) et le robot dans sa position initiale.
La figure 3 ci-dessus à droite montre le fichier texte correspondant.

Question 2-2 :

Modifier le corps de la fonction *lire_terrain* (cf. annexe 1 - page 5 - ligne 13) suivant les spécifications ci-dessus.

Question 2-3 :

Quel est (en fonction de L et H) le nombre maximal d'obstacles qu'on peut placer dans un terrain vide pour assurer l'existence d'un chemin de la case initiale du robot vers la sortie (on rappelle que les obstacles du mur ne comptent pas).

Pour évaluer un automate avec un ensemble de terrains de même largeur, même hauteur et même pourcentage de case occupées, on veut connaitre :
– les terrains réussis (les terrains desquels le robot est sorti),
– pour chaque terrain échoué (terrain dans lequel le robot est resté bloqué), à quelle ligne le robot s'est arrêté.
On veut donc que le programme ***test_performance*** crée un fichier résultat avec :
– le nombre de terrains,
– la hauteur des terrains,
– pour chaque terrain, la ligne à laquelle le robot s'est arrêté (0 indique qu'il est sorti du terrain).
Le fichier ci-dessous montre un exemple de fichier (resultat d'un test de performance avec 50 terrains de hauteur $H = 7$) :

```
50 7
  0 2 6 0 1 0 1 3 0 0 4 0 1 1 3 1 2 0 0 5 0 2 0 1 3 0 0 0 1 0 3 5 0 0 1 2 1 0 3 1 0 1 3 4 0 1 2 0 1 3
```

On souhaite modifier le programme ***test_performance*** (cf. annexe 8 - page 13) afin de générer un tel fichier.

Question 2-4 :

Comment modifier ce programme ***test_performance*** sans modifier les paquetages dont il dépend ?

A partir de ce fichier résultat, on souhaite savoir à quelle ligne s'arrête le robot quand il reste bloqué dans le terrain.
On veut donc représenter ce fichier résultat à l'aide d'un histogramme où chaque classe correspond à un numéro de ligne entre 1 et $H - 1$.

Question 2-5 :

Calculez et dessiner l'histogramme correspondant au fichier ci-dessus.

Partie 3 - Automate

[*barême indicatif : 4 pts*]

Dans le ***TP Robot***, les automates sont déterministes, c'est à dire que pour un terrain donné, un automate créera toujours la même suite de transitions (et dons d'actions) et le résultat final sera toujours le même.
En annexe 4 (pages 8 et 9), se trouve le code source du paquetage ***automate_paq*** utilisé lors du ***TP Robot***.

On souhaite rendre aléatoire le comportement de l'automate ainsi : pour un état et une transition, on définit deux actions possibles, puis lors du fonctionnement de l'automate, pour un état et une transition, on choisit aléatoirement une des deux actions possibles, la probabilité pour choisir l'une ou l'autre action est la même, c'est à dire 0,5 ou 50 %.
Il est possible de garder le caractère déterministe pour un état et une transition donnés en affectant deux fois la même action.

La figure 4 ci-dessous montre un exemple d'un tel automate. Par exemple :
– pour l'état 1 et la transition `LIBRE`, deux actions sont possibles *AVANCE* ou *DROITE*.
– pour l'état 1 et la transition `SORTIE`, la seule action possible est *AVANCE*, on la répète donc deux fois.
– comme dans le cas d'un automate déterministe, on interdit à l'automate d'avoir une action *AVANCE* pour une transition *OCCUPE*.

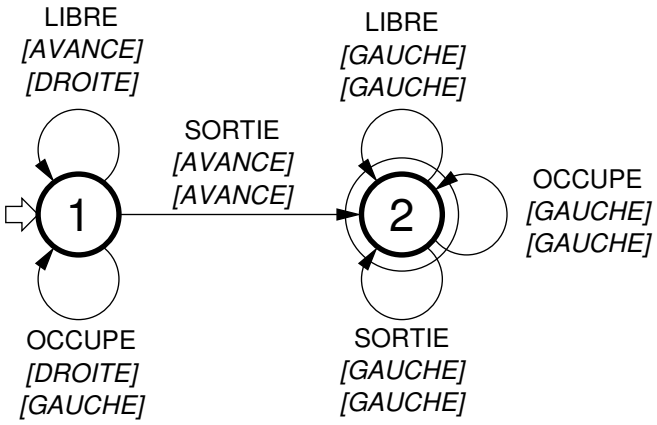


Figure 4

On modifie le format d'un fichier automate ainsi : dans la table des actions, chaque ligne correspondant à un état est formée de 6 valeurs :
– les deux actions possibles correspondant à la transition `LIBRE`,
– les deux actions possibles correspondant à la transition `OCCUPE`,
– les deux actions possibles correspondant à la transition `SORTIE`.

Par exemple, pour l'automate ci-dessus le fichier correspondant sera :

```
2
2

1 1 2
2 2 2

AVANCE DROITE   DROITE GAUCHE   AVANCE AVANCE
GAUCHE GAUCHE   GAUCHE GAUCHE   GAUCHE GAUCHE
```

On modifie le type *automate* du fichier ***automate_paq.ads*** (cf. annexe 4 - page 8) ainsi :

```
MAX_ETATS : constant positive := 50;
subtype NumEtat is positive range 1..MAX_ETATS;
type MatriceEntree is array(NumEtat'range,Entree'range) of NumEtat;
type MatriceSortie is array(NumEtat'range,Entree'range,1..2) of Sortie;
type automate is record
    nb_e : NumEtat;              -- nb d'états
    etat_final : NumEtat;        -- numéro de l'état final
    tab_t : MatriceEntree;       -- table de transition
    tab_s : MatriceSortie;       -- table de sortie
    etat_cour : NumEtat;         -- état courant
end record;
```

Question 3-1 :

Modifier en conséquence le fichier ***automate_paq.adb*** (cf. annexe 4 - pages 8 à 9) pour gérer un tel automate. Les spécifications du paquetage générique standard ADA ***ada.numerics.discrete_random*** sont rappelées en annexe 3, page 7.

Annexe 1 : paquetage *terrain_paq*

fichier terrain_paq.ads

```
1  -- paquetage terrain_paq
2  with ada.text_io;
3  use ada.text_io;
4
5  package terrain_paq is
6
7      LARGEUR_INCORRECTE, HAUTEUR_INCORRECTE,
8      LECTURE_FICHIER_IMPOSSIBLE, LECTURE_LARGEUR_INCORRECTE,
9      LECTURE_LARGEUR_FORMAT_INCORRECT, LECTURE_HAUTEUR_INCORRECTE,
10     LECTURE_HAUTEUR_FORMAT_INCORRECT, LECTURE_PARTIE_TERRAIN_INCORRECT,
11     LECTURE_CASE_INITIALE_OCCUPEE : exception;
12
13     type Terrain is private;
14
15     -- creer un terrain vide de dimensions L x H
16     -- la fonction leve l'exception LARGEUR_INCORRECTE si L est incorrecte
17     -- la fonction leve l'exception HAUTEUR_INCORRECTE si H est incorrecte
18     function creer_terrain_vide(L, H : positive) return Terrain;
19
20     -- lecture d'un terrain dans le fichier f ouvert en lecture
21     function lire_terrain(f : file_type) return Terrain;
22
23     -- lecture d'un terrain dans un fichier de nom nom_f
24     function lire_terrain(nom_f : string) return Terrain;
25
26     -- retourne la largeur du terrain T
27     function largeur(T : Terrain) return positive;
28     -- retourne la hauteur du terrain T
29     function hauteur(T : Terrain) return positive;
30
31     -- placer un obstacle dans la case de coordonnees (x,y) du terrain T
32     procedure placer_obstacle(T : in out Terrain; x,y : natural);
33
34     -- indique si la case de coordonnees (x,y) est libre
35     -- renvoie VRAI si et ssi :
36     --    1 <= x <= largeur ET 1 <= y <= hauteur ET T.tab(x,y) = '.'
37     function est_case_libre(T : Terrain; x,y : natural) return boolean;
38
39     -- teste si la position de coordonnees (x,y) est dans le terrain
40     function case_dans_terrain(T : Terrain; x,y : natural) return boolean;
41
42     -- renvoie la dimension maximale
43     function dimension_max return positive;
44
45     -- afficher le tableau dans le fichier f ouvert en ecriture
46     procedure ecrire(f : in file_type; T : in Terrain);
47
48 private
49     DIM_MAX : constant positive := 50;
50     type TableauCaracteres is array(1..DIM_MAX,1..DIM_MAX) of character;
51     type Terrain is record
52         largeur, hauteur : positive;
53         tab : TableauCaracteres;
54     end record;
55
56 end terrain_paq;
```

fichier terrain_paq.adb

```
1  -- paquetage terrain_paq
2
3  with ada.text_io, ada.integer_text_io;
4  use ada.text_io, ada.integer_text_io;
5
6  package body terrain_paq is
7
8      -- creer un terrain vide de dimensions L x H
9      -- la fonction leve l'exception LARGEUR_INCORRECTE si L est incorrecte
10     -- la fonction leve l'exception HAUTEUR_INCORRECTE si H est incorrecte
11     function creer_terrain_vide(L, H : positive) return Terrain is
12     ...
13
14     -- lecture d'un terrain dans le fichier f ouvert en lecture
15     function lire_terrain(f : file_type) return Terrain is
16
17         T : Terrain;
18         L, H : positive;   -- dimensions de terrain
19         c : character;
20
21     begin
22         -- lecture de la largeur
23         begin
24             get(f,L);
25         exception
26             when others=> raise LECTURE_LARGEUR_FORMAT_INCORRECT;
27         end;
28         if L>DIM_MAX or L mod 2 = 0 then
29             raise LECTURE_LARGEUR_INCORRECTE;
30         end if;
31         T.largeur := L;
32
33         -- lecture de la hauteur
34         begin
35             get(f,H);
36         exception
37             when others=> raise LECTURE_HAUTEUR_FORMAT_INCORRECT;
38         end;
39         if H>DIM_MAX or H mod 2 = 0 then
40             raise LECTURE_HAUTEUR_INCORRECTE;
41         end if;
42         T.hauteur := H;
43
44         -- lecture du terrain
45         for y in 1..H loop
46             -- se placer en début de la ligne suivante
47             begin
48                 skip_line(f);
49             exception
50                 -- problème : fin de fichier rencontré
51                 when others=> raise LECTURE_PARTIE_TERRAIN_INCORRECT;
52             end;
53             for x in 1..L loop
54                 if end_of_line(f) then
55                     -- problème : ligne incomplète
56                     raise LECTURE_PARTIE_TERRAIN_INCORRECT;
57                 end if;
58                 begin
59                     get(f,c);
60                 exception
61                     -- problème : fin de fichier rencontré
62                     when others=> raise LECTURE_PARTIE_TERRAIN_INCORRECT;
63                 end;
64                 if not(c='.' or c='X') then
65                     -- problème : mauvais caractère
66                     raise LECTURE_PARTIE_TERRAIN_INCORRECT;
67                 end if;
68                 T.tab(x,y) := c;
69             end loop;
70             if not end_of_line(f) then
71                 -- problème : ligne trop longue
72                 raise LECTURE_PARTIE_TERRAIN_INCORRECT;
73             end if;
74         end loop;
75
```

```
76          -- test de la case initiale
77          if T.tab((L+1)/2,(H+1)/2)='X' then
78              raise LECTURE_CASE_INITIALE_OCCUPEE;
79          end if;
80
81          return T;
82      end lire_terrain;
83
84      -- lecture d'un terrain dans un fichier de nom nom_f
85      function lire_terrain(nom_f : string) return Terrain is
86
87          T : Terrain;
88          f : file_type;
89
90      begin
91          -- ouverture du fichier en lecture
92          begin
93              open(f, in_file, nom_f);
94          exception
95              when others=>
96                  raise LECTURE_FICHIER_IMPOSSIBLE;
97          end;
98
99          -- lecture de T dans le fichier f
100         begin
101             T := lire_terrain(f);
102         exception
103             when others=>
104                 close(f);
105                 raise; -- propager l'exception tel quel
106         end;
107
108         -- fermeture du fichier
109         close(f);
110
111         return T;
112     end lire_terrain;
113
114     ...
115
116 end terrain_paq;
```

Annexe 2 : paquetage *robot_paq*

fichier robot_paq.ads

```
1  -- paquetage robot_paq
2
3  package robot_paq is
4
5      type Orientation is (Nord,Est,Sud,Ouest);
6      type Robot is private;
7
8      -- initialiser un robot en position (x,y) et orientation o
9      function init(x,y : natural; o : orientation) return Robot;
10
11     -- faire avancer le robot d'une case
12     procedure avancer(r : in out Robot);
13
14     -- faire tourner le robot d'un quart de tour sur sa gauche
15     procedure tourner_a_gauche(r : in out Robot);
16
17     -- faire tourner le robot d'un quart de tour sur sa droite
18     procedure tourner_a_droite(r : in out Robot);
19
20     -- recupere la position en abscisse de la case du robot
21     function abscisse(r : in Robot) return natural;
22
23     -- recupere la position en ordonnee de la case du robot
24     function ordonnee(r : in Robot) return natural;
25
26     -- recupere l'orientation du robot
27     function orient(r : in Robot) return Orientation;
28
29     -- recupere la position de la case devant le robot
30     procedure position_devant(r : in Robot; x,y : out natural);
31
32 private
33     ...
34 end robot_paq;
```

Annexe 3 : paquetage *ada.numerics.discrete_random*

extrait du **fichier ada.numerics.discrete_random.ads**

```
1  generic
2      type Type_Resultat is (<>); -- le résultat est de type discret (énuméré ou entier)
3  package Ada.Numerics.Discrete_Random is
4
5      type Generator is private;
6
7      -- initialisation du générateur G avec l'horloge système
8      procedure Reset (G : in Generator);
9
10     -- initialisation du générateur G avec la valeur X0
11     procedure Reset (G : in Generator; X0 : in Integer);
12
13     -- renvoie une valeur aléatoire de type Type_Resultat
14     function Random (G : Generator) return Type_Resultat;
15
16    ...
17 end Ada.Numerics.Discrete_Random;
```

Annexe 4 : paquetage automate_paq

fichier automate_paq.ads

```
1  -- paquetage de manipulation d'automate
2
3  with ada.text_io;
4  use ada.text_io;
5
6  package automate_paq is
7
8      type Entree is (CASE_LIBRE, CASE_OCCUPEE, SORTIE_TERRAIN);
9      type Sortie is (AVANCE, GAUCHE, DROITE);
10
11     type automate is private;
12
13     -- lecture d'un automate dans un fichier
14     function lire_automate(nom_f : string) return automate;
15
16     -- remet l'automate dans son état initial = 1
17     procedure reinitialiser(a : in out automate);
18
19     -- renvoie l'état courant de l'automate
20     function etat_courant(a : automate) return positive;
21
22     -- renvoie le nombre d'états de l'automate
23     function nb_etats(a : automate) return positive;
24
25     -- renvoie vrai si l'automate est dans son état final
26     function est_etat_final(a : automate) return boolean;
27
28     -- modifie l'automate suivant la transition t
29     -- la sortie correspondante est renvoyée par s
30     procedure faire_transition(a : in out automate; t : in Entree; s : out Sortie);
31
32 private
33     MAX_ETATS : constant positive := 50;
34     subtype NumEtat is positive range 1..MAX_ETATS;
35     type MatriceEntree is array(NumEtat'range,Entree'range) of NumEtat;
36     type MatriceSortie is array(NumEtat'range,Entree'range) of Sortie;
37     type automate is record
38         nb_e : NumEtat;              -- nb d'états
39         etat_final : NumEtat;        -- numéro de l'état final
40         tab_t : MatriceEntree; -- table de transition
41         tab_s : MatriceSortie;       -- table de sortie
42         etat_cour : NumEtat;         -- état courant
43     end record;
44 end automate_paq;
```

fichier automate_paq.adb

```
1  -- paquetage de manipulation d'automate
2
3  with ada.text_io, ada.integer_text_io;
4  use ada.text_io, ada.integer_text_io;
5
6  package body automate_paq is
7
8      package Entree_IO is new ada.text_io.enumeration_io(Entree);
9      package Sortie_IO is new ada.text_io.enumeration_io(Sortie);
10
11     -- lecture d'un automate dans un fichier nommé nom_f
12     function lire_automate(nom_f : string) return automate is
13
14         a : automate;
15         f : file_type;
16
17     begin
18         -- ouverture du fichier
19         open(f, in_file, nom_f);
20
21         -- lecture et vérification du nombre d'états
22         get(f, a.nb_e);
23
```

```
24          -- lecture et vérification de l'état final
25          get(f, a.etat_final);
26
27          -- lecture de la table de transition
28          for i in 1..a.nb_e loop
29              for j in Entree'range loop
30                  get(f, a.tab_t(i,j));
31              end loop;
32          end loop;
33
34          -- lecture de la table de sortie
35          for i in 1..a.nb_e loop
36              for j in Entree'range loop
37                  Sortie_IO.get(f, a.tab_s(i,j));
38              end loop;
39          end loop;
40
41          -- le mettre dans l'état initial
42          a.etat_cour := 1;
43
44          -- fermer le fichier
45          close(f);
46
47          return a;
48      end lire_automate;
49
50      -- remet l'automate dans son état initial = 1
51      procedure reinitialiser(a : in out automate) is
52
53      begin
54          a.etat_cour := 1;
55      end reinitialiser;
56
57      -- renvoie l'état courant de l'automate
58      function etat_courant(a : automate) return positive is
59
60      begin
61          return a.etat_cour;
62      end etat_courant;
63
64      -- renvoie le nombre d'états de l'automate
65      function nb_etats(a : automate) return positive is
66
67      begin
68          return a.nb_e;
69      end nb_etats;
70
71      -- renvoie vrai si l'automate est dans son état final
72      function est_etat_final(a : automate) return boolean is
73
74      begin
75          return a.etat_cour=a.etat_final;
76      end est_etat_final;
77
78      -- modifie l'automate suivant la transition t
79      -- la sortie correspondante est renvoyée par l'action s
80      procedure faire_transition(a : in out automate; t : in Entree; s : out Sortie) is
81
82      begin
83          s           := a.tab_s(a.etat_cour,t);
84          a.etat_cour := a.tab_t(a.etat_cour,t);
85      end faire_transition;
86
87  end automate_paq;
```

Annexe 5 : paquetage *systeme_robot*

fichier systeme_robot.ads

```
1  -- paquetage systeme_robot - fichier spécification
2
3  with automate_paq, terrain_paq, robot_paq;
4  use automate_paq, terrain_paq, robot_paq;
5
6  with ada.text_io;
7  use ada.text_io;
8
9  package systeme_robot is
10
11     -- type ConfigCase correspondant au type Entree défini
12     -- dans le paquetage automate_paq
13     subtype ConfigCase is Entree;
14
15     -- type ActionRobot correspondant au type Sortie défini
16     -- dans le paquetage automate_paq
17     subtype ActionRobot is Sortie;
18
19     -- initialiser un SystemeRobot avec un automate lu dans un fichier nommé nom_fa
20     -- et un fichier terrain :
21     -- . soit le terrain t0 est donné directement
22     -- . soit il est lu dans un fichier ouvert dons le descripteur est ft
23     -- . soit il est lu dans le fichier nommé nom_ft
24     procedure init(nom_fa : in string; t0 : in terrain);
25     procedure init(nom_fa : in string; ft : in file_type);
26     procedure init(nom_fa : in string; nom_ft : in string);
27
28     -- faire avancer le SystemeRobot d'une case
29     procedure avancer;
30
31     -- faire tourner le SystemeRobot 'a gauche
32     procedure tourner_a_gauche;
33
34     -- faire tourner le SystemeRobot 'a droite
35     procedure tourner_a_droite;
36
37     -- détermine l'état suivant du SystemeRobot suivant son automate
38     procedure suivant;
39
40     -- indique si le SystemeRobot est sorti
41     function est_sorti return boolean;
42
43     -- indique si le SystemeRobot a atteint son nombre max de transitions
44     function max_transitions_atteint return boolean;
45
46     -- indique le nombre de transitions effectu'ees
47     function nb_transitions return natural;
48
49     -- indique le nombre maximum de transitions
50     function nb_max_transitions return natural;
51
52     -- recupere la position de la case du SystemeRobot
53     procedure position(x,y : out natural);
54
55     -- recupere la position en abscisse de la case du SystemeRobot
56     function abscisse return natural;
57
58     -- recupere la position en ordonnee de la case du SystemeRobot
59     function ordonnee return natural;
60
61     -- recupere l'orientation du SystemeRobot
62     function orient return Orientation;
63
64     -- recupere la config de la case devant le SystemeRobot
65     function config_case_devant return ConfigCase;
66
67     -- ecrire à l'écran le SystemeRobot
68     procedure ecrire;
69
70 end systeme_robot;
```

Annexe 6 : programme *generation_terrains*

fichier generation_terrains.adb

```
1  -- génération aléatoire de terrains
2  -- le programme génère n terrains de largeur et hauteur fixes
3  -- avec largeur et hauteur impaires et inférieures à terrain_paq.dimension_max
4  -- avec une probabilité p (exprimee en pourcentage) que chaque case
5  -- autre que la case initiale soit occupée
6  -- l'appel du programme se fait avec 5 arguments :
7  --  generation_terrains nb_terrains largeur hauteur pourcentage_cases_occupees fichier_T
8  -- la sortie se fait dans le fichier de terrains nommé fichier_T
9
10 with ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
11 with terrain_paq;
12 use ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
13 use terrain_paq;
14 with ada.numerics.discrete_random;
15
16 procedure generation_terrains is
17
18     package Alea is new ada.numerics.discrete_random(natural);
19     use Alea;
20
21     G : generator;
22
23     DIM_MAX : constant positive := terrain_paq.dimension_max;
24
25     -- generation aleatoire d'un terrain de dimensions L colonnes et
26     -- H lignes avec le pourcentage p de cases occupées.
27     -- Préconditions : L et H impaires et inférieures à terrain_paq.dimension_max
28     --                 p : entier entre 1 et 100
29     function generation_aleatoire(L, H : positive; p : natural)
30       return Terrain is
31
32         T : Terrain;
33         boucle : boolean := true;
34
35     begin
36         while boucle loop
37
38             -- creer un terrain vide de dimensions L x H
39             T := creer_terrain_vide(L,H);
40
41             -- placer les cases occupées
42             for x in 1..L loop
43                 for y in 1..H loop
44                     -- pour chaque case autre que la case centrale
45                     if not (x=(L+1)/2 and y=(H+1)/2) then
46                         -- la case (x,y) est occupee si random(G) mod 100 < p
47                         if random(G) mod 100 < p then
48                             placer_obstacle(T, x, y);
49                         end if;
50                     end if;
51                 end loop;
52             end loop;
53
54             -- test d'existence d'un chemin
55             boucle := not existe_chemin_vers_sortie(T);
56         end loop;
57         return T;
58     end generation_aleatoire;
59
60     n, L, H, p : positive;
61     f : file_type;
62     T : Terrain;
63
64 begin
65     if argument_count /= 5 then
66         put_line("ERREUR : nb d'arguments incorrect");
67
68     else
69         -- récupération des arguments
70         n := positive'value(argument(1)); -- nb de terrains
71         L := positive'value(argument(2)); -- largeur des terrains
72         H := positive'value(argument(3)); -- hauteur des terrains
73         p := positive'value(argument(4)); -- pourcentage de cases occupees
74
```

```
75          -- tests sur L, H et p
76          if L>DIM_MAX or L mod 2 = 0 then
77              put_line("Dimension L incorrecte");
78              return;
79          end if;
80          if H>DIM_MAX or H mod 2 = 0 then
81              put_line("Dimension H incorrecte");
82              return;
83          end if;
84          if p>100 then
85              put_line("Pourcentage p trop grand");
86              return;
87          end if;
88
89          -- ouverture du fichier des terrains
90          create(f, out_file, argument(5));
91
92          -- ecriture du nombre de terrains
93          put(f, n, width=>1);
94          new_line(f);
95
96          -- creation des terrains
97          reset(G);
98          for i in 1..n loop
99              -- generer aleatoirement un terrain
100             T := generation_aleatoire(L,H,p);
101
102             -- ecrire le terrain généré dans f
103             ecrire(f, T);
104
105         end loop;
106
107         -- fermeture du fichier des terrains
108         close(f);
109
110     end if;
111 end generation_terrains;
```

Annexe 7 : *Makefile*

fichier Makefile

```
1  ##############################################################################
2  # Fichier Makefile
3  ##############################################################################
4
5
6  ##############################################################################
7  # règles de compilation séparée
8  ##############################################################################
9
10 terrain_paq.ali : terrain_paq.adb terrain_paq.ads
11     gnat compile terrain_paq.adb
12
13 generation_terrains.ali : generation_terrains.adb terrain_paq.ali
14     gnat compile generation_terrains.adb
15
16 ##########################################################
17 # regles de creation des executables
18
19 generation_terrains : generation_terrains.ali
20     gnat bind -x generation_terrains.ali
21     gnat link generation_terrains.ali -o generation_terrains
```

Annexe 8 : programme *test_performance*

fichier test_performance.adb

```
1  -- programme pour évaluer un robot automate dans une suite de terrains
2  -- syntaxe : test_performance fichier_automate fichier_terrains fichier_resultat
3  --  avec : fichier_automate = fichier contenant la description d'un automate
4  --         fichier_terrains = fichier contenant une suite de terrains
5  --         fichier_resultat = fichier dans lequel sont ecrits les statistiques du robot.
6  with ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
7  use ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
8  with systeme_robot;
9  use systeme_robot;
10
11 procedure test_performance is
12
13     type EtatRobot is (Marche,Bloque,Sorti);
14
15     fT : file_type; -- fichier des terrains
16     fR : file_type; -- fichier des résultats
17     n  : positive;  -- nb de terrains
18     er : EtatRobot;
19
20 begin
21     if argument_count /= 3 then
22         put_line("ERREUR : nb arguments incorrect");
23     else
24         -- ouverture du fichier des terrains
25         open(fT, in_file, argument(2));
26
27         -- ouverture du fichier des resultats
28         create(fR, out_file, argument(3));
29
30         -- lecture du nombre de terrains et écriture dans le fichier résultat
31         get(fT, n);
32         put(fR, n); new_line(fR);
33
34         for i in 1..n loop
35
36             -- initialiser le SystemeRobot avec l'automate et le terrain
37             systeme_robot.init(argument(1), fT);
38
39             -- le SystemeRobot est en état de marche
40             er := Marche;
41
42             -- boucle sur les transitions du robot
43             while er=Marche loop
44
45                 -- effectuer une transition
46                 systeme_robot.suivant;
47
48                 -- verifier si le robot est sorti ou
49                 --  si le nb max de transitions est atteint
50                 if systeme_robot.est_sorti then
51                     er := Sorti;
52                 elsif systeme_robot.max_transitions_atteint then
53                     er := Bloque;
54                 end if;
55
56             end loop;
57
58             -- ecrire le resultat dans le fichier
59             if er=Sorti then
60                 put(fR, systeme_robot.nb_transitions,width=>1);
61             else
62                 put(fR, "-1");
63             end if;
64             new_line(fR);
65
66         end loop;
67
68         -- fermeture des fichiers
69         close(fR);
70         close(fT);
71     end if;
72 end test_performance;
```

Corrigé

Partie 1 - Génération de terrains et test de performance d'un automate

[*barême indicatif : 5 pts*]

Question 1-1 :

```
-- programme gttp

-- @@@@@ : utiliser les instructions with et use des deux programmes
with ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
with terrain_paq, systeme_robot;
with ada.numerics.discrete_random;
-- @@@@@

procedure gttp is

-- @@@@@ : lignes 18-58 de generation_terrains
    package Alea is new ada.numerics.discrete_random(natural);
    use Alea;

    G : generator;

    DIM_MAX : constant positive := terrain_paq.dimension_max;

    -- generation aleatoire d'un terrain de dimensions L colonnes et
    -- H lignes avec le pourcentage p de cases occupées.
    -- Préconditions : L et H impaires et inférieures à terrain_paq.dimension_max
    --                 p : entier entre 1 et 100
    function generation_aleatoire(L, H : positive; p : natural)
      return Terrain is

       T : Terrain;
       boucle : boolean := true;

    begin
        while boucle loop

            -- creer un terrain vide de dimensions l x h
            T := creer_terrain_vide(l,h);

            -- placer les cases occupées
            for x in 1..l loop
                for y in 1..h loop
                    -- pour chaque case autre que la case centrale
                    if not (x=(l+1)/2 and y=(h+1)/2) then
                        -- la case (x,y) est occupee si random(G) mod 100 < p
                        if random(G) mod 100 < p then
                            placer_obstacle(T, x, y);
                        end if;
                    end if;
                end loop;
            end loop;

            -- test d'existence d'un chemin
            boucle := not existe_chemin_vers_sortie(T);
        end loop;
        return T;
    end generation_aleatoire;
-- @@@@@

    type EtatRobot is (Marche,Bloque,Sorti);
    n, L, H, p : positive;
    T : Terrain;
    er : EtatRobot;
    fR : file_type;




begin
    if argument_count /= 6 then
```

```
        put_line("ERREUR : nb arguments incorrect");
    else

-- @@@@@ : lignes 69-73 de generation_terrains
        -- récupération des arguments
        n := positive'value(argument(1));
        L := positive'value(argument(2));
        H := positive'value(argument(3));
        p := positive'value(argument(4));
-- @@@@@

        -- ouverture du fichier des resultats
        create(fR, out_file, argument(6));

        -- lecture du nombre de terrains et écriture dans le fichier résultat
        put(fR, n); new_line(fR);

        -- initialiser le générateur aléatoire
        reset(G);

        for i in 1..n loop

            -- générer un terrain
            T := generation_aleatoire(L, H, p);

            -- initialiser le SystemeRobot avec l'automate et le terrain
            systeme_robot.init(argument(5), T);

-- @@@@@ : lignes 39-66 de test_performance
            -- le SystemeRobot est en état de marche
            er := Marche;

            -- boucle sur les transitions du robot
            while er=Marche loop

                -- effectuer une transition
                systeme_robot.suivant;

                -- verifier si le robot est sorti ou
                --  si le nb max de transitions est atteint
                if systeme_robot.est_sorti then
                    er := Sorti;
                elsif systeme_robot.max_transitions_atteint then
                    er := Bloque;
                end if;

            end loop;

            -- ecrire le resultat dans le fichier
            if er=Sorti then
                put(fR, systeme_robot.nb_transitions,width=>1);
            else
                put(fR, "-1");
            end if;
            new_line(fR);

        end loop;
-- @@@@@

        -- fermeture des fichiers
        close(fR);
    end if;
end gttp;
```

Question 1-2 :

Il faut rajouter les règles suivantes :

```
robot_paq.ali : robot_paq.adb robot_paq.ads
    gnat compile robot_paq.adb

automate_paq.ali : automate_paq.adb automate_paq.ads
    gnat compile automate_paq.adb

systeme_robot.ali : systeme_robot.adb systeme_robot.ads \
 terrain_paq.ali robot_paq.ali automate_paq.ali
    gnat compile systeme_robot.adb

gttp.ali : gttp.adb terrain_paq.ali systeme_robot.ali
    gnat compile gttp.adb

gttp : gttp.ali
    gnat bind -x gttp.ali
    gnat link gttp.ali -g -o gttp
```

Partie 2 - Changement de configuration des terrains

[*barême indicatif : 10 pts*]

Question 2-1 :

```
    -- creer un terrain vide de dimensions L x H
    function creer_terrain_vide(L, H : positive) return Terrain is

        T : Terrain;

    begin
        if L<5 or L>DIM_MAX or L mod 2 = 0 then
            raise LARGEUR_INCORRECTE;
        end if;

        -- lecture de la hauteur
        if H<3 or H>DIM_MAX then
            raise HAUTEUR_INCORRECTE;
        end if;

        T.largeur := L;
        T.hauteur := H;
        for x in 1..L loop
            for y in 1..H loop
                if x=1 or x=L or y=H then
                    T.tab(x,y) := 'X';
                else
                    T.tab(x,y) := '.';
                end if;
            end loop;
        end loop;
        return T;

    end creer_terrain_vide;
```

Question 2-2 :

• Remplacer la ligne 28

```
        if L>DIM_MAX or L mod 2 = 0 then
```

par

```
        if L<5 or L>DIM_MAX or L mod 2 = 0 then
```

• Remplacer la ligne 39

```
        if H>DIM_MAX or H mod 2 = 0 then
```

par

```
        if H<3 or H>DIM_MAX then
```

• Avant la ligne 68

```
    T.tab(x,y) := c;
```

ajouter les instructions

```
        -- test du mur
        if (x=1 or x=L or y=H) and c='.' then
                raise LECTURE_PARTIE_TERRAIN_INCORRECT;
        end if;
```

• Remplacer la ligne 77

```
        if T.tab((L+1)/2,(H+1)/2)='X' then
```

par

```
        if T.tab((L+1)/2,H-1)='X' then
```

Question 2-3 :

Un terrain vide contient $(L-2)\times(H-1)$ cases vides.

Le plus court chemin possible correspond à la case initiales et toutes les cases vides au-dessus soit $H-1$ cases vides.

On peut donc placer au plus $(L-2)\times(H-1)-(H-1)=(L-3)\times(H-1)$ obstacles.

Question 2-4 :

La difficulté est de pouvoir écrire la hauteur de l'ensemble des terrains dans le fichier résultat.
La solution est pour chaque terrain de d'abord le lire, récupérer la hauteur du premier terrain lu, puis d'initialiser le système robot avec le terrain lu précédemment.
Ce qui donne les modifications suivantes.

• Ajouter le paquetage *terrain_paq* dans les dépendances : remplacer les lignes 8 et 9

```
with systeme_robot;
use  systeme_robot;
```

par

```
with terrain_paq, systeme_robot;
use  terrain_paq, systeme_robot;
```

• A la ligne 19 ajouter la déclaration d'une variable de type `terrain` :

```
    T : terrain;
```

• Remplacer les lignes 36 et 37

```
            -- initialiser le SystemeRobot avec l'automate et le terrain
            systeme_robot.init(argument(1), fT);
```

par

```
            -- lire le terrain
            T := lire_terrain(fT);

            -- écrire la hauteur du premier terrain lu dans fR
            if i=1 then
                put(fR, hauteur(T)); new_line(fR);
            end if;

            -- initialiser le SystemeRobot avec l'automate et le terrain T
            systeme_robot.init(argument(1), T);
```

• Remplacer les lignes 60 et 62

```
                put(fR, systeme_robot.nb_transitions,width=>1);
```

```
                put(fR, "-1");
```

par

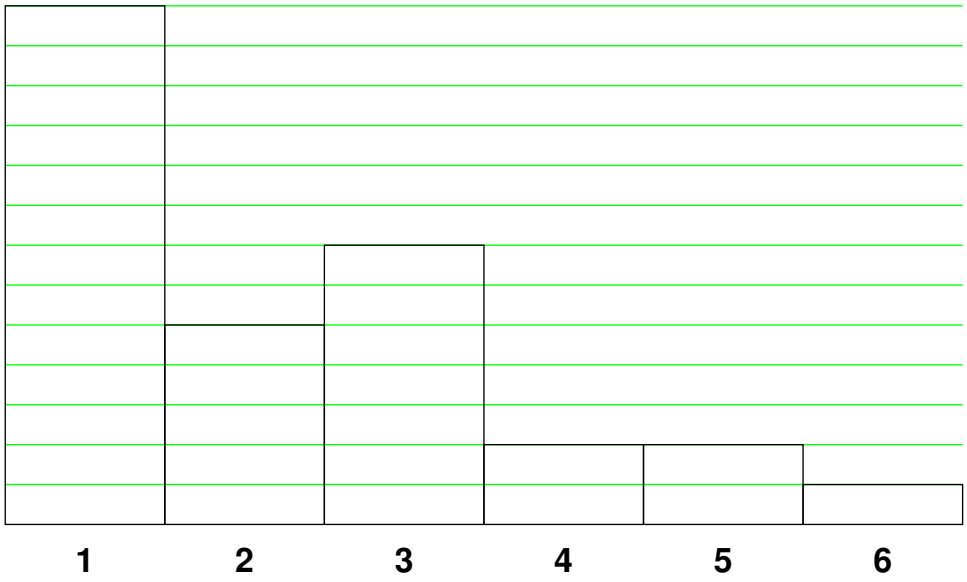
```
                put(fR, 0, width=>1);
```

```
                put(fR, systeme_robot.ordonnee, width=>1);
```

Question 2-5 :

L'histogramme a donc 6 classes :

Classe	1	2	3	4	5	6
Effectif	13	5	7	2	2	1



Partie 3 - Automate

[*barême indicatif : 4 pts*]

Question 3-1 :

• En ligne 5, ajouter l'instruction suivante :

```
with ada.numerics.discrete_random;
```

• Ajouter dans le corps du paquetage, les instructions suivantes :

```
subtype UnDeux is integer range 1..2;
package alea_1_2 is new ada.numerics.discrete_random(UnDeux);
use alea_1_2;
G : generator;
```

• Dans la fonction `lire_automate`, remplacer la ligne 37

```
Sortie_IO.get(f, a.tab_s(i,j));
```

par

```
Sortie_IO.get(f, a.tab_s(i,j,1));
Sortie_IO.get(f, a.tab_s(i,j,2));
```

• Dans la fonction `reinitialiser`, ajouter en ligne 54, l'instruction

```
reset(G);
```

• Dans la procédure `faire_transition`, remplacer la ligne 83

```
s := a.tab_s(a.etat_cour,t);
```

par

```
s := a.tab_s(a.etat_cour,t,random(G));
```