Département STPI Lundi 18 juin 2012

I3 - Algorithmique

Durée : *3h*
Documents autorisés : **AUCUN** (calculatrice comprise) [1]

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- N'utilisez pas de crayon à papier.

1 Récursivité (4 points)

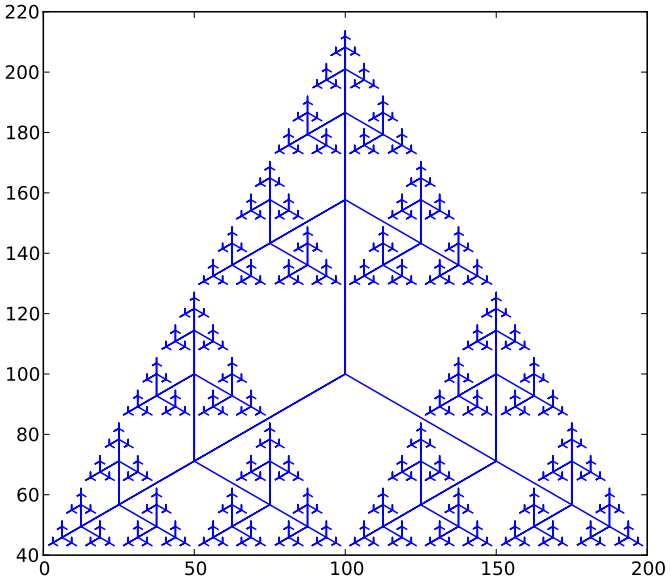


FIGURE 1 – Dessin récursif

Rappels mathématiques

Les trois sommets d'un triangle équilatéral, dont l'un des côté est parallèle à l'axe des abscisses, de centre x_c, y_c de base b ont les coordonnées suivantes :
- $x_c, y_c + 2 * h/3$
- $x_c - b/2, y_c - h/3$
- $x_c + b/2, y_c - h/3$

avec $h = \sqrt{b^2 - (b/2)^2}$

Questions

Supposons que la procédure suivante permette de dessiner un segment sur un graphique (variable de type `Graphique`) :

1. Sauf les dictionnaires pour les étudiants non francophones

– **procédure** ligne **(E/S** g : Graphique,**E** x1,y1,x2,y2 : **Reel)**

L'objectif est de concevoir une procédure `dessinerCroix` qui permet de dessiner sur un graphique des dessins récursifs tels que présentés par la figure 1. La signature de cette procédure est :

– **procédure** dessinerCroix **(E/S** g : Graphique,**E** xCentre,yCentre,base : **Reel**, niveauRecursion : **Naturel)**

1. Lors du premier appel de cette procédure, donnez la valeur des quatre derniers paramètres effectifs afin d'obtenir le graphique de la figure 1.
2. Donnez le corps de cette procédure.

Solution proposée :

1. dessinerCroix(g,100,100,100,5)
2. Algo

```
procédure dessinerCroix (E/S g : Graphique,E xCentre,yCentre,base : Reel, niveauRecursion : Naturel)
    Déclaration  x1,y1,x2,y2,x3,y3,h : Reel
    debut
        h ← racineCarree(b*b-(b/2)*(b/2))
        x1 ← xc
        y1 ← yc+2*h/3
        x2 ← xc-b/2
        y2 ← yc-h/3
        x3 ← xc+b/2
        y3 ← yc-h/3
        ligne(g,xc,yc,x1,y1)
        ligne(g,xc,yc,x2,y2)
        ligne(g,xc,yc,x3,y3)
        si n>0 alors
            dessinerCroix(g,x1,y1,b/2,n-1)
            dessinerCroix(g,x2,y2,b/2,n-1)
            dessinerCroix(g,x3,y3,b/2,n-1)
        finsi
    fin
```

2 Tris (3 points)

On a vu en cours que l'algorithme du tri rapide est :

```
procédure triRapide (E/S t :Tableau[1..MAX] d'Entier,E nb :Naturel)
debut
    triRapideRecursif(t,1,nb)
fin
procédure triRapideRecursif (E/S t :Tableau[1..MAX] d'Entier,E d,f :Naturel)
    Déclaration  indicePivot : Naturel
debut
    si d<f alors
        partionner(t,d,f,indicePivot)
        triRapideRecursif(t,d,indicePivot-1)
        triRapideRecursif(t,indicePivot+1,f)
    finsi
fin
```

Donnez l'algorithme de la procédure `partitionner`.

Solution proposée :

procédure partitionner **(E/S** t : **Tableau[**1..MAX**] d'Entier** ; **E** debut,fin : **Naturel** ; **S** indicePivot : **Naturel)**

Déclaration i,j,pivot : **Naturel**

debut
pivot ← t[debut]
i ← debut
j ← fin
tant que i≤j **faire**
si t[i]≤pivot **alors**
i ← i+1
sinon
si t[j]>pivot **alors**
j ← j-1
sinon
echanger(t[i],t[j])
finsi
finsi
fintantque
indicePivot ← j
echanger(t[debut],t[j])
fin

3 Le jeu du serpent (13 points)

L'objectif global de cet exercice est de concevoir une partie du jeu du serpent : jeu des années 80, que l'on retrouve sur tous les vieux téléphones, qui consiste à guider un serpent de façon à ce qu'il mange des proies sans qu'il se mange lui même.

Le serpent avance constamment dans une direction. On peut le faire changer de direction. Lorsqu'il arrive au bord de l'air de jeu, il réapparaître de l'autre côté. Enfin lorsqu'il mange une proie, il grandit pendant un certain temps (seule sa tête avance). La partie se termine lorsque le serpent mange une partie de son corps.

Les figures 2 présentent plusieurs états du jeu.

3.1 L'air de jeu

L'ensemble des éléments du jeu, les éléments du serpent et les proies, ont une position. Le serpent se dirige dans une direction.

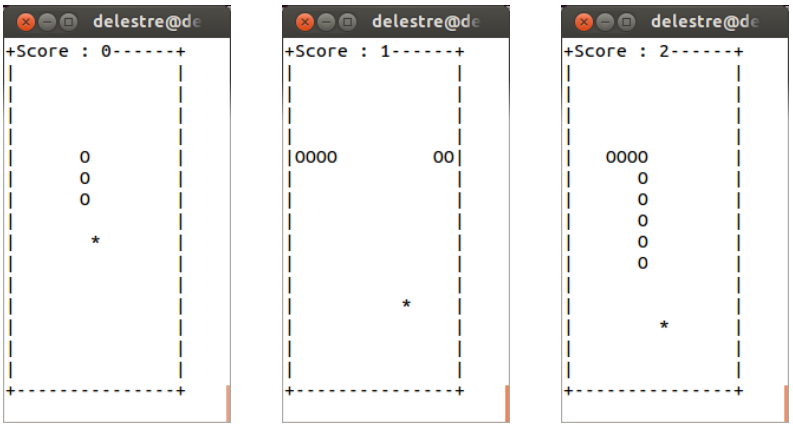
Soit les types `Direction` et `Position` définis de la façon suivante :

Type Direction **=** {nord,est,ouest,sud}
Type Position **= Structure**
x : **NaturelNonNul**
y : **NaturelNonNul**
finstructure

L'air de jeu est caractérisé par quatre constantes `XMIN`, `XMAX`, `YMIN`, `YMAX` (pour toute position p on a : $XMIN \leq p.x \leq XMAX$ et $YMIN \leq p.y \leq YMAX$). On suppose que la position de coordonnées $(XMIN, YMIN)$ se trouve en haut à gauche de l'air de jeu, et celle de coordonnées $(XMAX, YMAX)$ en bas à droite.

Proposez le corps de la fonction suivante qui permet de calculer une nouvelle position à partir d'une position donnée et d'une direction donnée (attention si la position donnée se trouve en bordure de l'air de jeu, la position calculée peut être à l'opposé de l'air de jeu).

– **fonction** nouvellePosition (p : Position, d : Direction**) :** Position



(a) Serpent qui se dirige vers le nord, il n'a encore mangé aucune proie. La position de la tête a pour coordonnées (7,5).

(b) Serpent ayant mangé une proie et qui est passé du côté droit de l'air de jeu, au côté gauche (il se dirige vers l'est). La position de la tête a pour coordonnées (4,5).

(c) Serpent ayant mangé deux proies. Il se dirige vers le sud.

FIGURE 2 – Quelques exemples d'état du jeu serpent

Solution proposée :

fonction nouvellePosition (p : Position, d : Direction) **:** Position

Déclaration resultat : Position

debut

resultat ← p

cas où d **vaut**

est:

p.x ← p.x - 1

si p.x < XMIN **alors**

p.x ← XMAX

finsi

ouest:

p.x ← p.x + 1

si p.x > XMAX **alors**

p.x ← XMIN

finsi

nord:

p.y ← p.y - 1

si p.y < YMIN **alors**

p.y ← YMAX

finsi

sud:

p.y ← p.y + 1

si p.y > YMAX **alors**

p.y ← YMIN

finsi

fincas

retourner resultat

fin

3.2 Le serpent

Soit le type `ListeChaineePosition` (défini comme nous l'avons vu en cours pour la liste chaînée d'entiers) qui possède les fonctions et procédures suivantes :

- **fonction** listeVide **() :** ListeChaineePosition
- **fonction** estVide **(**uneListe : ListeChaineePosition**) : Booleen**
- **procédure** ajouter **(E/S** uneListe : ListeChaineePosition,**E** p : Position**)**
- **fonction** obtenirPosition **(**uneListe : ListeChaineePosition**) :** Position

  ⌊**précondition(s)** non estVide(uneListe)
- **fonction** obtenirListeSuivante **(**uneListe : ListeChaineePosition**) :** ListeChaineePosition

  ⌊**précondition(s)** non estVide(uneListe)
- **procédure** fixerListeSuivante **(E/S** uneListe : ListeChaineePosition,**E** nelleSuite : ListeChaineePosition**)**

  ⌊**précondition(s)** non estVide(uneListe)
- **procédure** supprimerTete **(E/S** l :ListeChaineePosition**)**

  ⌊**précondition(s)** non estVide(l)
- **procédure** supprimer **(E/S** uneListe : ListeChaineePosition**)**

On se propose de représenter le serpent à l'aide d'une liste chaînée de positions, tel que la tête de la liste représentera la position de la queue du serpent et le dernier élément de la liste représentera la position de la tête du serpent. Ainsi on représente le type `Serpent` de la façon suivante :

Type Serpent **= Structure**
  tete : ListeChaineePosition
  queue : ListeChaineePosition
finstructure

La figure 3 présente une représentation graphique du serpent de la figure 2(a).

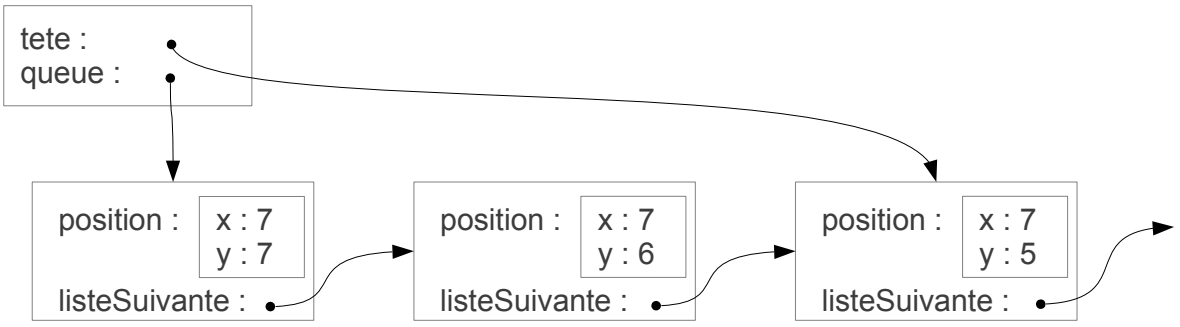
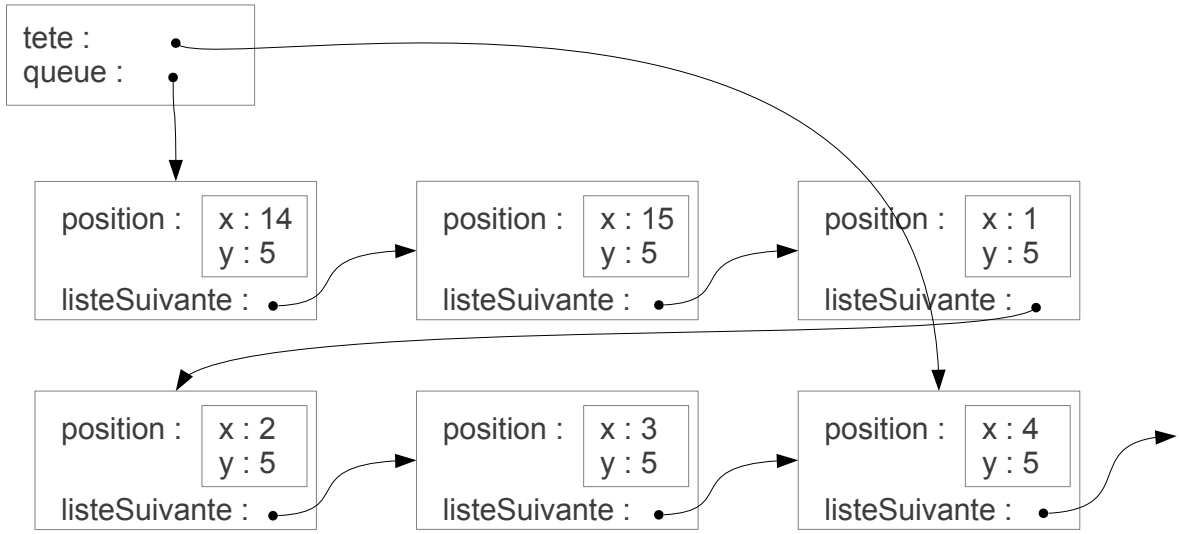


FIGURE 3 – Représentation graphique de l'instance de serpent de la figure 2(a)

1. Donnez le type `ListeChaineePosition`.
   Solution proposée :

   Type ListeChaineeDEntiers **=** ^ NoeudDEntier
   Type NoeudDEntier **= Structure**
     entier : **Entier**
     listeSuivante : ListeChaineeDEntiers
   finstructure

2. Représentez graphiquement (à l'image de la figure 3) l'instance du type serpent correspondant à l'état du jeu représenté par la figure 2(b).
Solution proposée :



3. Donnez le corps des deux fonctions suivantes :
   - **fonction** obtenirPositionTete **(**s : Serpent**) :** Position
   - **fonction** obtenirPositionQueue **(**s : Serpent**) :** Position

   Solution proposée :

   fonction obtenirPositionTete **(**s : Serpent**) :** Position
   debut
   retourner obtenirPosition(s.tete)
   fin
   fonction obtenirPositionQueue **(**s : Serpent**) :** Position
   debut
   retourner obtenirPosition(s.queue)
   fin

4. Donnez le corps de la procédure suivante qui permet de faire avancer un serpent en le faisant grandir d'un élément :
   - **procédure** avancerEnGrandissant **(E/S** s : Serpent, **E** d : Direction**)**

   Solution proposée :

   procédure avancerEnGrandissant **(E/S** s : Serpent, **E** d : Direction**)**
   Déclaration temp : ListeChaineePosition
   debut
   temp ← listeVide()
   ajouter(temp,nouvellePosition(obtenirPositionTete(s),d))
   fixerListeSuivante(s.tete,temp)
   s.tete ← temp
   fin

5. Donnez le corps de la procédure suivante qui permet de faire avancer un serpent sans le faire grandir :
   - **procédure** avancerSansGrandir **(E/S** s : Serpent, **E** d : Direction**)**

   Solution proposée :

   procédure avancerSansGrandir **(E/S** s : Serpent, **E** d : Direction**)**

debut
avancerEnGrandissant(s,d)
supprimerTete(s.queue)
fin

6. Donnez le corps de la procédure suivante qui initialise un serpent (état du serpent en début de partie) tel que la position de sa queue soit au centre de l'aire de jeu et que sa direction initiale soit au nord :
   – **procédure** initSerpent **(E/S** s : Serpent, **E** tailleInitiale : **NaturelNonNul)**
   ⌊**précondition(s)** tailleInitiale>1 et tailleInitiale≤(YMAX-YMIN) div 2

   Solution proposée :

   procédure initSerpent **(E/S** s : Serpent, **E** tailleInitiale : **NaturelNonNul)**
   ⌊**précondition(s)** tailleInitiale>1 et tailleInitiale≤(YMAX-YMIN) div 2
   Déclaration p : Position
   resultat : Serpent
   i : **Naturel**
   debut
   p.x ← (XMAX-XMIN) div 2
   p.y ← (YMAX-YMIN) div 2
   resultat.tete ← listeVide()
   ajouter(resultat.tete,p)
   resultat.queue ← resultat.tete
   pour i ←1 **à** tailleInitiale-1 **faire**
   avancerEnGrandissant(s,nord)
   finpour
   retourner resultat
   fin

7. Donnez le corps de la fonction suivante qui permet de savoir si une position donnée est une position d'un élément du serpent :
   – **fonction** estUnePositionDuSerpent (s : Serpent, p : Position**) : Booleen**

   Solution proposée :

   fonction estUnePositionDuSerpent (s : Serpent, p : Position**) : Booleen**
   Déclaration temp : ListeChaineePosition
   trouve : **Booleen**
   debut
   temp ← s.queue
   trouve ← FAUX
   tant que non estVide(temp) et non trouve **faire**
   si obtenirPosition(temp)=p **alors**
   trouve ← VRAI
   sinon
   temp ← obtenirListeSuivante(temp)
   finsi
   fintantque
   retourner trouve
   fin

3.3 Jeu du serpent

Soit les types suivants :

- **Type** T1 **= fonction**(d : Direction) : Direction
- **Type** T2 **= procédure**(**E** s : Serpent ; positionProie : Position ; score : **Naturel**)

Soit la procédure `jouer` suivante qui permet de jouer au jeu du serpent :

```
procédure jouer (f1 : T1, p2 : T2, tailleInitiale,tailleAgrandissementSiProieMange : NaturelNonNul)
  |précondition(s)  tailleInitiale>1 et tailleInitiale≤(YMAX-YMIN) div 2
  Déclaration  s : Serpent
               pProie, p : Position
               tailleAgrandissementRestant,score : Naturel
               dir : Direction
               jeuFini : Booleen
debut
  initSerpent(s)
  tailleAgrandissementRestant ← 0
  score ← 0
  dir ← nord
  pProie ← positionAleatoireProie(s)
  jeuFini ← FAUX
  tant que non jeuFini faire
    dir ← f1(dir)
    p ← calculerNellePosition(obtenirPositionTete(s),dir)
    si estUnePositionDuSerpent(s,p) alors
      jeuFini ← VRAI
    sinon
      si p = pProie alors
        pProie ← positionAleatoireProie(s)
        tailleAgrandissementRestant ← tailleAgrandissementRestant + tailleAgrandissementSiProieMange

        score ← score + 1
      finsi
      si tailleAgrandissementRestant > 0 alors
        avancerEnGrandissant(s,dir)
        tailleAgrandissementRestant ← tailleAgrandissementRestant - 1
      sinon
        avancerSansGrandir(s,dir)
      finsi
    finsi
    p2(s,pProie,score)
  fintantque
  supprimerSerpent(s)
fin
```

1. Quels sont les rôles des fonction `f1` et procédure `p2` ? Pourquoi sont-elles passées en paramètre de la procédure `jouer` ?

   Solution proposée :

   Le but de la fonction `f1` est de connaître la prochaine direction du serpent. Le but de la procédure `p2` est d'afficher le serpent, la proie et le score. Les deux sont passées en paramètre de la procédure `jouer` car cela permet de séparer le logique métier de l'IHM.

2. La fonction `positionAleatoireProie` permet d'obtenir la position d'une nouvelle proie (ses coordonnées sont tirées au hasard). Lors de l'appel, pourquoi doit-on passer le serpent comme paramètre effectif ?

   Solution proposée :

   Afin que la position de la proie ne soit pas une position d'un des éléments du serpent.

3. En supposant que l'on possède la fonction suivante qui permet d'obtenir un naturel aléatoire compris entre deux bornes, donnez l'algorithme de la fonction `positionAleatoireProie`.
   - **fonction** naturelAleatoire (borneInf, borneSup : **Naturel) : Naturel**

     ⌊**précondition(s)** borneInf $<$ borneSup

   Solution proposée :

   fonction positionAleatoireProie (s : Serpent) **:** Position

   Déclaration resultat : Position

   debut
   repeter
   resultat.x ← naturelAleatoire(XMIN,XMAX)
   resultat.y ← naturelAleatoire(YMIN,YMAX)
   jusqu'a ce que non estUnePositionDuSerpent(s,resultat)
   retourner resultat
   fin