Programmation applicative – L2
Troisième série de TP : Structure de données et graphisme, le jeu des dominos

H. Chahdi {hatim.chahdi@umontpellier.fr}
T. Lorieul {titouan.lorieul@umontpellier.fr}
J. Thiebaut {jocelyn.thiebaut@umontpellier.fr}

1 Introduction

Cet enoncé concerne 3 séances de TP. Pensez à sauvegarder précieusement votre travail au fûr et à mesure des séances, pour pouvoir l'utiliser lors de l'examen final.

On décide de représenter un domino par une liste de deux entiers compris entre 0 et 6, 0 représentant le blanc. Le jeu dont un joueur dispose est représenté par une liste de dominos. Sur la table il y a une chaîne de dominos que nous représenterons aussi par une liste de dominos. Le joueur peut jouer s'il a un domino dont un des entiers est présent à une extrémité de la chaîne. Le cas échéant, la chaîne s'agrandit et le domino placé est retiré du jeu du joueur.

Nous allons mettre l'accent sur les structures de données mises en œuvre. Il est notamment **indispensable** d'identifier dans vos commentaires les accesseurs, constructeurs et fonctions de test pour chaque structure de données que nous utiliserons.

2 Structures de données `domino` et `jeu`

Exercice 1 *Écrire l'interface constructeur et accesseur de la structure de données domino.*

Exercice 2 *Écrire l'interface constructeur et accesseur de la structure de données jeu.*

Exercice 3 *Écrire une fonction* `est_double?` *qui teste si un domino est un double. Exemple :*

```
(est_double? '(3 3)) ==> \#t
```

Exercice 4 *Écrire ensuite une fonction* `doubles` *qui détermine tous les doubles d'un jeu. Exemple :*

```
(doubles '( (1 2) (3 3) (4 4) )) ==> ( (3 3) (4 4)).
```

Exercice 5 *Écrire une fonction* `retourner` *qui retourne un domino. Exemple :*

```
(retourner '(3 4)) ==> (4 3)
```

Exercice 6 *Écrire une fonction* `peut-jouer?` *qui prend en paramètre un entier n compris entre 0 et 6, un jeu j, et qui teste si le jeu j contient au moins un domino qui contient l'entier n. Dans ce cas la fonction rendra #t, sinon #f*

Exercice 7 *Écrire la fonction* `extraire` *qui prend en paramètre un entier n compris entre 0 et 6, un jeu j, et qui retourne le premier domino de j qui contient n. Pour cette fonction on fera l'hypothèse que j contient toujours un domino qui convient. Exemple :*

```
(extraire 3 '( (1 2) (3 3) (3 5))) ==> (3 3).
```

3 Manipulations des jeux et de la chaîne de dominos

Jouer au dominos consiste à bâtir une chaîne de dominos. Cette chaîne ne peut être agrandie qu'en ajoutant un domino compatible au début ou à la fin : si la chaîne commence (ou finit) par un n, on peut ajouter au début (ou à la fin) un domino contenant n. Exemple : si la chaîne est `( (3 0) (0 6) (6 6) (6 5) (5 2) )` alors on peut ajouter le domino `(3 4)` au début de la chaîne pour obtenir la chaîne `( (4 3) (3 0) (0 6) (6 6) (6 5) (5 2) )` ou le domino `(2 1)` à la fin pour obtenir la chaîne `((3 0) (0 6) (6 6) (6 5) (5 2) (2 1))`. Une chaîne est donc valide si le second élément de toute liste représentant un domino est égal au premier élément de la liste suivante (représentant le domino suivant).

Exercice 8 *Écrire une fonction* `chaine_valide?` *qui teste si une chaîne est valide.*

Exercice 9 *Écrire les fonctions* `ext_g` *et* `ext_d` *qui calculent respectivement la valeur de l'extrémité gauche et droite d'une chaîne. Exemple :*

```
(ext_g '( (3 0) (0 6) (6 6) (6 5) (5 2) ) ) ==> 3.
```

Exercice 10 *Écrire une fonction* `supprimer` *qui prend comme paramètre un domino d et un jeu j, et qui rend le jeu j duquel on a retiré d. Le domino d est supposé exister dans le jeu j.*

Exercice 11 *Écrire une fonction* `ajouter` *qui prend comme paramètre un domino d et une chaîne, et qui ajoute de façon cohérente le domino à la chaîne ch. On supposera que le domino peut toujours être ajouté à la chaîne. Exemples :*

```
(ajouter '(3 4) ' ((3 0) (0 6) (6 6) (6 5) (5 2) ))
  ==> ( (4 3) (3 0) (0 6) (6 6) (6 5) (5 2) )
(ajouter '(6 2) ' ((3 0) (0 6) (6 6) (6 5) (5 2) ))
  ==> ( (3 0) (0 6) (6 6) (6 5) (5 2) (2 6))
```

Exercice 12 *Écrire une fonction* `pose` *qui prend comme paramètre une liste (j ch) composée d'un jeu j et d'une chaîne ch et qui calcule la liste (jp chp) obtenue en ajoutant (si cela est possible) un domino d du jeu j à la chaîne ch. Le cas échéant, jp est j duquel on a retiré le domino d, et chp est ch à laquelle on a ajouté de manière cohérente le domino d. Dans le cas où aucun domino de j ne peut être ajouté à ch, le résultat est la liste (j ch).*

4 Partie graphique

Pour cette partie du TP, le niveau de langage utilisé doit être « niveau avancé ». D'autre part, choisissez l'item d'ajout d'un TeachPack dans le menu Langage, cliquez deux fois sur htdp, choisissez draw.ss et cliquez sur ok. Pressez ensuite le bouton executer.

4.1 Ouvrir une fenêtre

```
(start nombre1 nombre2) ==> void
```

Cette fonction ouvre une fenêtre dans laquelle pourront s'exécuter les fonctions de dessin [1]. Les deux arguments sont des nombres représentant respectivement la longueur et la hauteur de la fenêtre en pixels [2]. Cette fonction est évaluée à void, c'est-à-dire qu'elle ne retourne aucune valeur. Elle a cependant un effet sur l'écran puisqu'elle affiche une fenêtre. Exemple :

```
(start 320 200)
```

Toutes les fonctions suivantes auront besoin qu'une zone pour dessiner (canvas en anglais), ouverte avec la fonction start.

4.2 Les positions

Les « positions » sont des objets de scheme qui possèdent au moins deux valeurs. Les deux valeurs représentent des coordonnées dans le plan (exprimées en nombre de pixels) ; autrement dit, une position. Pour créer un objet « position » vous devez utiliser la fonction suivante :

```
(make-posn nombre1 nombre2) ==> position
```

Le repère utilisé est raccordé au coin supérieur gauche de la fenêtre ouverte. L'axe des abscisses est orienté de la gauche vers la droite et l'axe des ordonnées est orienté du haut vers le bas. Exemple :

```
(make-posn 20 34)
```

Étant donnée une position vous disposez également de deux fonctions, `posn-x` et `posn-y` qui permettent d'accéder aux valeurs des coordonnées en x et y de cette position.

```
(posn-x (make-posn 4 5)) ==> 4
(posn-y (make-posn 4 5)) ==> 5
```

4.3 Tracer une ligne

```
(draw-solid-line position1 position2) ==> true
```

Cette fonction trace une ligne entre les deux positions passées en arguments.

```
(draw-solid-line (make-posn 20 25) (make-posn 24 34))
```

1. aussi appelée « canvas »
2. L'image affichée à l'écran est constituée d'un assemblage de points minuscules appelés « pixels ». Le pixel est la plus petite surface de l'écran sur laquelle il peut agir.

4.4 Dessiner un disque

```
(draw-solid-disk position nombre) ==> true
```

dessine un disque centré autour de la position passée en premier argument et dont le rayon est spécifié par le nombre passé en second argument. Exemple :

```
(draw-solid-disk (make-posn 20 25) 5)
```

Commencez par ouvrir une fenêtre de 342 pixels sur 256 pixels

Exercice 13 *Écrire une fonction* `(dessiner-gros-point position) ==> true` *qui, étant donné une position, dessine un disque centré autour de cette position avec un rayon de 2 pixels.*

Exercice 14 *Écrire une fonction* `(dessiner-rectangle position1 position2) ==> true` *qui étant donné deux positions représentant respectivement le coin supérieur gauche et inférieur droit d'un rectangle, trace ce rectangle.*

Exercice 15 *Écrire une fonction* `(dessiner-demi-dominos position nombre) ==> true` *qui dessine un carré de 24 pixels de côté, centré autour de la position, avec autant de point(s) à l'intérieur que le nombre passé en second argument, compris entre 1 et 6, le spécifie.*

Exercice 16 *Écrire une fonction* `(dessiner-dominos position domino) ==> true` *qui, étant donné un domino, dessine ce domino autour de la position précisée. Le domino est dessiné horizontalement : un demi-domino à gauche et un demi-domino à droite.*

Exercice 17 *Écrire une fonction* `(dessiner-jeu-dominos jeu nombre) ==> true` *qui, étant donné un jeu de dominos (autrement dit une liste de dominos) et un numéro de joueur, affiche ces dominos empilés horizontalement en bas à gauche de la fenêtre pour le joueur 1 ou en bas à droite de la fenêtre pour le joueur 2. Chaque colonne ne peut contenir plus de 5 dominos. Si la colonne est pleine, on continuera d'empiler juste à côté. Un espace vide et large d'un pixel sera laissé entre les colonnes.*

Exercice 18 *Écrire une fonction* `(dessiner-chaîne-dominos chaîne) ==> true` *qui, étant donné une chaîne de dominos (autrement dit une liste de dominos), affiche ces dominos en ligne, tout en haut de la fenêtre. Si la longueur de la fenêtre est trop courte la chaîne doit continuer juste en dessous. Un espace vide et large d'un pixel sera laissé entre les colonnes.*

Exercice 19 *l'expression* `(random 7)` *renvoie un nombre au hasard entre 0 et 6. Utiliser cette expression pour écrire la fonction* `(générer-jeu x)` *qui génère au hasard un jeu de x dominos (donc une liste de x dominos). Écrire aussi une fonction* `InitChaîne` *qui génère une liste contenant un domino choisi au hasard pour servir de point de départ à la chaîne de jeu.*

Exercice 20 *Écrire la fonction* `début-jeu` *qui renvoie une liste composée de trois éléments : deux jeux et une chaîne de jeu générés au hasard. La chaîne chaîne de jeu initiale ne contient qu'un domino.*

Exercice 21 *Écrire une fonction* `(jouer j1 j2 ch)` *qui prend en entrée deux jeux de dominos et une chaîne, puis fait jouer alternativement chaque joueur jusqu'à ce que l'un d'entre eux n'ait plus de dominos dans son jeu, ou que l'un d'entre eux ne puisse plus poser de dominos. Graphiquement, les jeux et la chaîne seront redessinés dès qu'un nouveau domino est posé (on utilisera la fonction* `clear-all` *qui efface tout ce qui a été dessiné) après un temps d'attente de quelques secondes (on utilisera la fonction* `(sleep- for-a-while x)` *qui permet d'attendre x secondes avant l'évaluation de l'expression suivante).*

Exercice 22 *il ne reste plus qu'à écrire la fonction* `jeu` *qui fait appel à la fonction début- jeu et appelle, avec les jeux et la chaîne obtenus, l'expression* `(jouer j1 j2 ch)`*. Admirer le (votre) travail!!!*