Programmation 3

L2 Informatique 2014-2015

Fiche de TP 2

Le jeu du Chomp

Ce TP se déroule en deux séances et est à faire par binômes. Le travail réalisé doit être envoyé au plus tard exactement une semaine après le début de la 2e séance de TP traitant ce sujet. Il sera disposé sur la plate-forme prévue à cet effet et constitué des programmes répondant aux questions et des éventuels fichiers annexes qui peuvent être demandés.

L'objectif de ce TP est d'implanter le *Chomp*, un jeu combinatoire abstrait à deux joueurs. Les règles sont les suivantes. Deux joueurs se disputent une tablette de chocolat de dimension $n \times m$ où n et m sont des entiers supérieurs à un (voir figure 1). Chaque joueur choisit alternativement

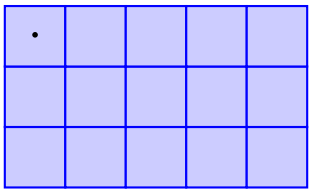

FIGURE 1 – La tablette de chocolat du *Chomp* 3×5 dans sa configuration initiale. Le carré empoisonné contient un point $\cdot$.

un carré de chocolat, le mange, et mange aussi tous les carrés qui se trouvent en bas et à sa droite (voir figure 2). La partie s'arrête lorsque l'un des deux joueurs mange le carré de chocolat en

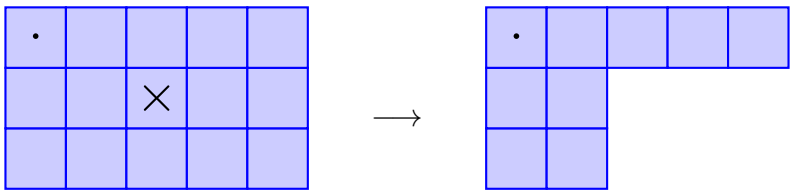

FIGURE 2 – Le joueur mange un carré de chocolat — celui qui contient une croix $\times$. Ceci a l'effet de manger tous les carrés de chocolat situés en bas et à sa droite.

position $(0, 0)$. Ce carré est en effet empoisonné et ce joueur perd la partie.

L'un des objectifs de ce TP est de parfaire l'utilisation de la bibliothèque graphique MLV qui permet d'afficher la tablette de chocolat et de faire en sorte que deux joueurs puissent jouer au *Chomp* à la souris. On suit ici une approche ascendante (dit *bottom-up*). Celle-ci consiste à réaliser le projet en constituant une à une ses pièces pour les assembler finalement.

Conseils :

— pour chaque fonction programmée, s'interroger sur les arguments qui peuvent poser des problèmes. Capturer ces cas à l'aide de pré-assertions ;
— utiliser au maximum les fonctions déjà programmées dans les nouvelles à écrire. Il faut éviter au maximum la duplication de code ;
— il ne suffit pas de programmer uniquement les fonctions qui sont demandées. Pour mener le sujet à bien, des fonctions supplémentaires et non mentionnées explicitement dans le sujet seront utiles. Soigner leur nom ;
— dès qu'une fonction est écrite, la tester de manière à recouvrir tous les cas significatifs possibles et conserver les tests (le but étant de les relancer pour vérifier l'intégrité du programme lors de modifications futures) ;
— commenter le code en évitant absolument les commentaires inutiles. Il faut commenter chaque fonction écrite, juste avant son prototype. Il faut y mentionner les trois points suivants : (1) une description du rôle de la fonction, (2) une description de ses paramètres et (3) une description de ce qu'elle renvoie ;
— préférer la concision au maximum. Un code concis et lisible possède une grande valeur.

Exercice 1. (Définitions des types)

1. Définir un type `Tablette` qui représente une tablette de chocolat de dimension $n \times m$. Utiliser un tableau $n \times m$ qui contient des entiers. Un contenu à `1` signifie que le carré de chocolat correspondant existe encore tandis qu'un contenu à `0` signifie qu'il a été mangé. Toute variable de type `Tablette` connaît sa dimension, c.-à-d. les entiers n et m.
2. Définir un type énuméré `Joueur` qui permet de modéliser les deux joueurs.
3. Définir un type `Position` qui permet de représenter une position de jeu de *Chomp*. Une position est déterminée par une tablette de chocolat et le joueur dont c'est le tour de jouer.
4. Définir un type `Coup` qui permet de modéliser un coup joué. Un coup est entièrement spécifié par les coordonnées `x` et `y` du carré de chocolat que le joueur souhaite manger.

Exercice 2. (Manipulation des objets)

1. Écrire une fonction

```
1 Tablette creer_tablette(int n, int m);
```

qui crée et renvoie une variable de type `Tablette` de dimension $n \times m$. La tablette renvoyée possède tous ses carrés de chocolat.

2. Écrire une fonction

```
void manger(Tablette *t, int x, int y);
```

qui modifie la tablette `t` de sorte à manger le carré de chocolat en position (x, y) ainsi que tous ceux qui sont en dessous de lui et à sa droite.

3. Un coup dans une position donnée est légal s'il ordonne de manger un carré de chocolat qui existe encore. Écrire une fonction

```
int est_legal(Position *pos, Coup *coup);
```

qui renvoie `1` si le coup `coup` est légal dans la position `pos` et `0` sinon.

4. La partie est terminée lorsque le carré de chocolat empoisonné vient d'être mangé. Dans ce cas, c'est le joueur qui vient de jouer qui a perdu et l'autre qui a gagné. Écrire une fonction

```
int est_jeu_termine(Position *pos, Joueur *joueur_gagnant);
```

qui renvoie `1` si la partie représentée par la position `pos` est terminée et `0` sinon. Dans le cas où la partie est terminée, la fonction affecte à la variable pointée par `joueur_gagnant` le joueur qui gagne la partie.

5. Écrire une fonction

```
void jouer_coup(Position *pos, Coup *coup);
```

qui joue le coup `coup` dans la position `pos`. Il ne faut pas oublier de modifier le champ qui contient le joueur dont c'est le tour de jouer.

Exercice 3. (Assemblage du jeu)

1. Écrire une fonction

```
void afficher_position(Position *pos);
```

qui affiche en utilisant la bibliothèque MLV la position `pos`.

2. Écrire une fonction

```
Coup lire_coup(Position *pos);
```

qui attend un clic de l'utilisateur sur la tablette de chocolat dans la fenêtre graphique et calcule et renvoie le coup spécifié par l'utilisateur. Si l'utilisateur ne clique pas sur la tablette, ou bien clique sur un carré de chocolat déjà mangé, la fonction ne fait rien et attend de traiter le prochain clic.

3. Utiliser les fonctions précédentes pour construire le programme `Chomp.c` qui permet de jouer au *Chomp*. Utiliser pour cela l'algorithme suivant :

   (1) lire la dimension de la tablette de chocolat sur laquelle le jeu va se dérouler (passée en argument au programme) ;

   (2) initialiser la position `pos` ;

   (3) tant que la position `pos` ne représente pas une partie terminée :

(a) afficher la position `pos` sur la fenêtre graphique ;

(b) lire un coup `c` sur la fenêtre graphique ;

(c) jouer le coup `c` dans la position `pos` ;

(4) afficher le numéro du joueur gagnant.

Exercice 4. (Améliorations)

Cet exercice est optionnel, il n'est a envisager que lorsque tout le reste fonctionne parfaitement.

1. Proposer une option au programme pour gérer des *matchs*. Un match est une succession de parties dans laquelle le nombre de victoires de chaque joueur est pris en compte. Le déroulement est le suivant : chacun des deux joueurs entre un nombre souhaité de parties ; ensuite, un nombre de parties égal à la moyenne (arrondie à l'entier supérieur) entre ces deux entiers sont jouées. À l'issue de ces parties, le résultat du match est affiché (victoire pour l'un ou l'autre des joueurs ou bien match nul).
2. On souhaite concevoir un système de sauvegarde de partie. Une partie est encodée par la taille du plateau et la suite des coups joués depuis le début de la partie. Proposer un format de fichier pour coder une partie de cette façon. Ajouter au programme la possibilité de sauvegarder une partie (et donc écrire son encodage dans un fichier) et la possibilité de charger une partie (et donc de lire son encodage depuis un fichier).
3. Proposer un mode de jeu contre l'ordinateur. Suggérer plusieurs niveaux de difficulté : dans le plus facile, l'ordinateur joue ses coup au hasard ; dans l'autre, imaginer un moyen de faire jouer l'ordinateur avec une certaine force.