Mini-projet 1 (Raffinages)

1 Cahier des charges

Le jeu des 13 allumettes se joue à deux joueurs qui, à tour de rôle, doivent prendre 1, 2 ou 3 allumettes d'un tas qui en contient initialement 13. Le perdant est le joueur qui prend la dernière allumette.

Le but est d'écrire un programme qui permet à un joueur humain d'affronter la machine. Plusieurs niveaux de jeu sont possibles pour l'ordinateur. Le niveau *naïf* consiste à choisir au hasard le nombre d'allumettes qu'il prend. Le niveau *distrait* consiste à choisir au hasard le nombre d'allumettes entre 1 et 3, sans tenir compte de celles qui sont dans le tas. Ainsi, s'il reste 2 allumettes dans le tas, le distrait pourra essayer d'en prendre 3 alors que le naïf en prendra soit 1 soit 2, en aucun cas 3. Le niveau *rapide* consiste à prendre systématiquement le plus grand nombre possible d'allumettes. Enfin, le niveau *expert* correspond à un niveau où l'ordinateur joue du mieux possible.

En plus d'être l'adversaire, l'ordinateur est également l'arbitre. En début de jeu, il demande au joueur humain le niveau de l'ordinateur. L'utilisateur répond avec 'n' pour naïf, 'd' pour distrait, 'r' pour rapide et 'e' pour expert sachant que l'utilisateur peut également utiliser les majuscules. En cas de réponse différente, c'est le niveau expert qui sera choisi. L'ordinateur affiche le niveau choisi. Ensuite, l'ordinateur demande au joueur humain s'il veut commencer ou non. Le joueur humain commence si et seulement s'il répond `'o'` ou `'O'`.

Enfin, l'ordinateur vérifie que les joueurs respectent les règles du jeu. Ainsi, si un joueur essaie de tricher, son coup est annulé et il doit recommencer. En particulier, un joueur doit nécessairement retirer entre 1 et 3 allumettes du tas quand c'est à son tour de jouer.

En fin de partie, l'ordinateur dit qui a gagné.

Avant de demander à un joueur combien il veut prendre d'allumettes, les allumettes restantes sont affichées à l'écran sous la forme de barres verticales (trois l'une sous l'autre pour représenter une allumette). Pour faciliter leur comptage, les allumettes sont groupées par cinq.

L'exemple suivant spécifie l'interface avec l'utilisateur à respecter impérativement (au caractère près !).

```
Niveau de l'ordinateur (n)aïf, (d)istrait, (r)apide ou (e)xpert ? d
Mon niveau est distrait.
Est-ce que vous commencez (o/n) ? f

| | | | |   | | | | |   | | |
| | | | |   | | | | |   | | |
| | | | |   | | | | |   | | |

Je prends 3 allumettes.

| | | | |   | | | | |
| | | | |   | | | | |
```

```
| | | | |   | | | | |

Combien d'allumettes prenez-vous ? 3

| | | | |   | |
| | | | |   | |
| | | | |   | |

Je prends 2 allumettes.

| | | | |
| | | | |
| | | | |

Combien d'allumettes prenez-vous ? 4
Arbitre : Il est interdit de prendre plus de 3 allumettes.
Combien d'allumettes prenez-vous ? 3

| |
| |
| |

Je prends 3 allumettes.
Arbitre : Il reste seulement 2 allumettes.
Je prends 1 allumette.

|
|
|

Combien d'allumettes prenez-vous ? 0
Arbitre : Il faut prendre au moins une allumette.
Combien d'allumettes prenez-vous ? 3
Arbitre : Il reste une seule allumette.
Combien d'allumettes prenez-vous ? 1

J'ai gagné.
```

Indication : En Ada, on utilisera le paquetage `Alea` fourni (fichiers `alea.ads` et `alea.adb` dont il n'est pas utile de regarder le contenu). Le fichier `exemple_alea.adb` (listing 1) est un exemple d'utilisation du paquetage `Alea`. `Alea` est un paquetage générique [1] paramétré par deux entiers correspondant aux bornes de l'intervalle dans lequel les nombres aléatoires seront tirés. Pour l'utiliser, le `use Alea;` en début de fichier n'est pas suffisant. Il faut aussi instancier le paquetage pour donner une valeur aux bornes. Ceci se fait dans la partie déclarations :

```
1  package Mon_Alea is new Alea (5, 15);
2      -- Les nombres aléatoires seront dans [5..15]
3  use Mon_Alea;  -- on peut alors utiliser Get_Random_Number
```

2 Contraintes

1. Le programme doit fonctionner sur les salles d'enseignement de l'ENSEEIHT.

1. Cette notion sera vue plus tard...

```ada
with Ada.Text_IO;           use Ada.Text_IO;
with Ada.Integer_Text_IO;   use Ada.Integer_Text_IO;
with Alea;

-- Procédure qui illustre l'utilisation du paquetage Alea.
procedure Exemple_Alea is

    package Mon_Alea is
        new Alea (5, 15);  -- générateur de nombre dans l'intervalle [5, 15]
    use Mon_Alea;

    Nombre: Integer;
begin
    -- Afficher 10 nombres aléatoires
    Put_Line ("Quelques nombres aléatoires : ");
    for I in 1..10 loop
        Get_Random_Number (Nombre);
        Put (Nombre);
        New_Line;
    end loop;
end Exemple_Alea;
```

Listing 1 – Le fichier `exemple_alea.adb`

2. **Il est interdit de définir d'autres sous-programmes que ceux founis et d'utiliser des tableaux.**
3. On suppose que l'utilisateur saisit bien un entier quand on le lui demande (pas de robustesse à réaliser sur ce point).

3 Documents à rendre

Les **documents à rendre**, dits *livrables*, seront rendus via votre projet Git (dans le dossier pr01).

Les *livrables* sont :

1. un **rapport sous la forme d'un document partagé de type Google Doc** [2] pour les raffinages et l'autoévaluation de ces raffinages et du code. Il faudra faire une **copie** du document `https://docs.google.com/document/d/1dIRXO0JrCEX1IQCx5uTHeOXdvw1AXpV1ZVTl6DENeHg`

   Ce document doit être partagé avec votre enseignant de TP en lui donnant au moins les droits de commentateur. Il pourra ainsi vous faire des commentaires sur vos raffinages.

   Les différentes rubriques de ce document devront être remplies.

2. Nous utilisons un Google Doc pour avoir un document partagé avec lequel il sera facile pour l'enseignant de faire des commentaires/remarques sur les raffinages proposés par un étudiant. Raffiner est difficile, en particulier car c'est relativement subjectif. Faire simplement des retours rapides est un atout...

   Si vous ne souhaitez pas utiliser un *Google Doc*, échangez avec votre enseignant de TP pour trouver une solution. Sans solution, c'est le PDF rendu sur Git qui sera évalué.

Il inclut en particulier la grille d'évaluation des raffinages et la grille d'évaluation du code. Ces deux grilles seront complétées par l'étudiant et prises en compte dans la notation.
Les explications sur la grille des raffinages est à cette URL : `https://docs.google.com/spreadsheets/d/1A42zW2va97LPAyp27Ue6v16QKlQhqbsiLh0oG8NekpM`.

2. Un **export PDF du document partagé** sera rendu. Ceci permet de garder une trace du travail rendu.
3. Les **fichiers sources du programme**. Ce sont les fichiers déjà présents sur le dépôt Git. Le seul fichier source à modifier est **`allumettes.adb`**.
4. Le fichier **`LISEZ-MOI.txt` complété**, en particulier le temps passé sur le projet et les **réponses aux questions**.

4 Principales dates

Voici les **principales dates** du projet (les modalités de remise des différents éléments sont décrits sur le site Web, à consulter impérativement) :

— lundi 23 septembre : **mise en ligne du sujet**,

— fin première séance de projet : remise des **raffinages réalisés en séance de projet** (document `raffinages.pdf`, nom à respecter impérativement, sur Gitlab),

— jeudi 26 septembre : remise des **raffinages (version finale)**, y compris la **grille d'évaluation** des raffinages (document `raffinages.pdf`).

— fin deuxième séance de projet : remise du **code source écrit pendant la séance de projet** (ne pas oublier de pousser les modifications faites sur Gitlab),

— mardi 8 octobre, 8 h : remise de la **version finale des livrables** (toujours via Git), y compris les raffinages (si modifiés), les grilles d'évaluations des raffinages et du code et le fichier `LISEZ-MOI.txt` complété (avec en particulier la réponses aux questions).

5 Barème (indicatif)

1. Raffinages : 8 points
2. Programmation : 6 points
3. Correction du programme : 3 points
4. Réponses aux questions : 3 points

Attention, les pénalités suivantes peuvent s'appliquer sur la note obtenue :

— 5 points : le programme ne compile pas.

— 2 points : le programme rendu contient des messages d'avertissement.

— 2 points : mauvaise présentation du rapport. Il s'agit de conserver la structure du rapport et de le compléter.

— 1 point : orthographe, construction des phrases...

6 Qualités d'un raffinage

6.1 Règles

1. Un raffinage doit être bien présenté (indentation).

```
Ri : Comment « Action complexe » ?
     actions...
     liées par des...
     structures de contrôle
```

2. **Rappel :** Un raffinage explique **comment** réaliser une action (ou un expression) complexe sous forme d'une **décomposition** en actions liées par des structures de contrôle (séquence, conditionnelle, répétition).
3. Le raffinage d'une action (sous-actions et structures de contrôle) doit décrire complètement cette action.
4. Le raffinage d'une action ne doit décrire que cette action.
5. Les structures de contrôle sont celles du langage algorithmique en respectant les normes de présentation vues en cours.
6. Éviter les structures de contrôle déguisées (si, tant que) : les expliciter en utilisant la structure de contrôle du langage algorithmique (Si, TantQue...)
7. Une action complexe commence par un verbe à l'infinitif.
8. La formulation d'une expression complexe doit décrire la valeur de cette expression (par exemple, « somme stricts des diviseurs du nombre », « le nombre est parfait », « le plus grand de a et b », etc.).
9. Attention : dans une séquence, l'exécution d'une action complexe ne commencera que lorsque l'action précédente sera terminée.

6.2 Compréhension

1. Le vocabulaire utilisé doit être précis et clair.
2. La formulation d'une action (ou une expression) complexe doit être concise et précise. Tout ne peut pas être écrit dans l'intitulé de l'action mais l'essentiel doit être présent et permettre d'identifier les éléments correspondants dans le cahier des charges ou la spécification.
3. Expliciter les données manipulées par une action permet de la préciser (flot de données).
4. Utiliser une contrainte (entre accolades) entre deux actions permet de préciser les obligations de l'action précédente et les hypothèses pour l'action suivante. Par exemple :

```
    Demander un numéro de mois      Mois : out Entier
    { Mois >= 1 ET Mois <= 12 }
    ...
```

5. Chaque niveau de raffinage doit apporter suffisamment d'information (mais pas trop). Il faut trouver le bon équilibre ! Idéalement, on introduit plusieurs actions complexes qui devraient pouvoir être décomposées par des personnes différentes.

6. Les actions introduites devraient avoir un niveau d'abstraction homogène.
7. Ne pas utiliser de formulation « Comparer » car si on compare, c'est pour faire quelque chose ! On n'a pas exprimé l'intention mais le moyen.
8. Il ne devrait y avoir qu'une structure de contrôle (hors séquence) par raffinage. On peut en avoir deux (voire plus) si et seulement si l'introduction d'une nouvelle action complexe est trop artificielle. C'est par exemple le cas si on écrit un algorithme qui donne le nombre de coefficients nuls d'une matrice : il est légitime d'avoir deux **`Pour`** et un **`Si`**.
9. On doit comprendre l'objectif d'une action complexe juste en lisant son intitulé et les données qu'elle manipule sans avoir à lire sa décomposition !

6.3 Remarque

— Certaines de ces règles sont subjectives !

— L'important est de pouvoir expliquer et justifier les choix faits

7 Vérification d'un raffinage

7.1 Nom des actions

— A-t-on utilisé des verbes à l'infinitif pour nommer les actions ?

7.2 Action et sous-actions

On appelle *sous-action* une action qui apparaît dans la décomposition d'une action complexe.

1. Pour être correct, un raffinage doit répondre à la question « COMMENT ? » !
2. Vérifier l'appartenance d'une action A au raffinage d'une action C en demandant « POURQUOI A ? ». Si la réponse n'est pas C :
   — soit on a identifié une action complexe intermédiaire
   — soit A n'est pas à sa place (ne fait pas partie des objectifs de C)

7.3 Flots de données

Utiliser les flots de données pour vérifier :

1. les communications entre niveaux :
   — les sous-actions doivent produire les résultats de l'action ;
   — les sous-actions peuvent (doivent) utiliser les entrées de l'action.
2. l'enchaînement des actions au sein d'un niveau :
   — une donnée ne peut être utilisée (**in**) que si elle a été produite (**out**) par une action précédente (ou si elle était en **in** de l'action complexe en cours de décomposition).

7.4 Raffinages et variables

1. Ne pas oublier de faire apparaître les flots de données.
2. On ne déclare pas une variable dans un raffinage. Elles apparaissent seulement dans les flots de données. Ensuite, on peut les rassembler dans un dictionnaire des données et, dans le cadre de ce mini-projet, elles seront déclarées comme variables locales du programme principal.