Mini-projet 1 (Raffinages)

1 Cahier des charges

Le jeu du devin se joue à deux joueurs. Le premier joueur choisit un nombre compris entre 1 et 999. Le second doit le trouver en un minimum d'essais. À chaque proposition, le premier joueur indique si le nombre proposé est plus grand ou plus petit que le nombre à trouver. À la fin, le nombre d'essais nécessaires est affiché.

Vous devez écrire un programme qui permet à un joueur humain de jouer au jeu du devin contre l'ordinateur. En début de partie, le programme doit laisser le choix au joueur humain entre trouver un nombre choisi par l'ordinateur ou choisir un nombre et le faire deviner à l'ordinateur. En fin de partie, le programme propose à nouveau au joueur de faire une partie.

Quant c'est à l'ordinateur de trouver le nombre, le joueur humain doit lui indiquer si le nombre proposé est trop petit (`'p'` ou `'P'`), trop grand (`'g'` ou `'G'`) ou trouvé (`'t'`, `'T'`). L'ordinateur utilisera une recherche par dichotomie pour trouver le nombre.

Enfin, si l'ordinateur se rend compte que le joueur humain triche, la partie est arrêtée avec le message « Vous trichez. J'arrête cette partie. ». Bien sûr l'ordinateur ne connaît pas (avant de l'avoir deviné) le nombre choisi par l'utilisateur.

Le listing 1 présente un exemple d'utilisation du programme à développer. Il constitue une spécification des interactions attendues avec l'utilisateur.

Listing 1 – Exemple d'utilisation du programme

```
1
2  1- L'ordinateur choisit un nombre et vous le devinez
3  2- Vous choisissez un nombre et l'ordinateur le devine
4  0- Quitter le programme
5  Votre choix : 1
6
7  J'ai choisi un nombre compris entre 1 et 999.
8  Proposition 1 : 900
9  Trop petit.
10 Proposition 2 : 10000
11 Trop grand.
12 Proposition 3 : 990
13 Trop grand.
14 Proposition 4 : 988
15 Trouvé.
16 Bravo. Vous avez trouvé 988 en 4 essais.
17
18 1- L'ordinateur choisit un nombre et vous le devinez
19 2- Vous choisissez un nombre et l'ordinateur le devine
20 0- Quitter le programme
21 Votre choix : 2
22
23 Avez-vous choisi un nombre compris entre 1 et 999 (o/n) ? o
24 Proposition 1 : 500
25 Trop (g)rand, trop (p)etit ou (t)rouvé ? g
26 Proposition 2 : 250
```

```
Trop (g)rand, trop (p)etit ou (t)rouvé ? x
Je n'ai pas compris. Merci de répondre :
   g si ma proposition est trop grande
   p si ma proposition est trop petite
   t si j'ai trouvé le nombre
Trop (g)rand, trop (p)etit ou (t)rouvé ? t
J'ai trouvé 250 en 2 essais.

1- L'ordinateur choisit un nombre et vous le devinez
2- Vous choisissez un nombre et l'ordinateur le devine
0- Quitter le programme
Votre choix : 2

Avez-vous choisi un nombre compris entre 1 et 999 (o/n) ? x
J'attends...
Avez-vous choisi un nombre compris entre 1 et 999 (o/n) ? O
J'attends...
Avez-vous choisi un nombre compris entre 1 et 999 (o/n) ? O
Proposition 1 : 500
Trop (g)rand, trop (p)etit ou (t)rouvé ? 0
Voulez-vous vraiment abandonner (o/*) ? n
Proposition 1 : 500
Trop (g)rand, trop (p)etit ou (t)rouvé ? T
J'ai trouvé 500 en 1 essai.

1- L'ordinateur choisit un nombre et vous le devinez
2- Vous choisissez un nombre et l'ordinateur le devine
0- Quitter le programme
Votre choix : 5

Choix incorrect.

1- L'ordinateur choisit un nombre et vous le devinez
2- Vous choisissez un nombre et l'ordinateur le devine
0- Quitter le programme
Votre choix : 0

Au revoir...
```

Indications techniques 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 :

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

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 Contraintes

1. Le programme doit fonctionner sur les salles d'enseignement de l'ENSEEIHT.
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).
4. On impose de structurer le programme précédent en 3 exercices. Dans le premier exercice, l'ordinateur choisit un nombre et le fait deviner à l'utilisateur. Dans le deuxième exercice, c'est l'utilisateur qui choisit un nombre et l'ordinateur qui doit le trouver. Le troisième exercice correspond au programme complet.

   Le projet sera fait en équipe de deux étudiants. L'un des deux fera les exercices 1 et 3 et l'autre l'exercice 2. Chaque étudiant évaluera les raffinages faits par l'autre étudiant en remplissant la grille correspondante (voir section 3). Bien sûr les remarques doivent être prises en compte pour améliorer les raffinages et l'auto-évaluation.

   Les deux étudiants devront remplir ensemble la grille d'évaluation du code (voir section 3).

3 Documents à rendre

Les **documents à rendre**, dits *livrables*, seront rendus via un projet Git (qui sera créé une fois toutes les équipes connues) ou Moodle. Les modalités seront précisées ultérieurement.

Les *livrables* sont :

1. un rapport sous la forme d'un Google Doc. Il faudra faire une **copie** du document `https://docs.google.com/document/d/1Owph33lUgZ6ZKaaf3k5OZ_T02ciAckjMuop2HmyRxoQ`

   Ce document doit être partagé par les deux membres de l'équipe et avec votre enseignant de TP en lui donnant au moins les droits de commentateur. Il pourra ainsi vous faire des commentaires sur vos raffinages. Vous ne devez pas fermer ces commentaires mais répondre à chaque commentaire en indiquant comment il a été traité (ou en demandant des précisions ou compléments). C'est l'enseignant de TP qui pourra éventuellement fermer un commentaire une fois satistait de son traitement.

   Les différentes rubriques de ce document devront être remplies.

   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 Google Doc sera rendu. Ceci permet de garder une trace du travail rendu.
3. Les fichiers sources du programme.
4. Le fichier `questions.txt` complété.

4 Principales dates

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

— Mardi 19 septembre 2023 : publication du sujet et des fichiers fournis (Moodle).

— Séance TP 2 : réalisation des raffinages sur Google Doc partagé avec l'enseignant de TP.

— Lundi 25 septembre 2023 : finalisation des raffinages (Google Doc) pour validation par l'enseignant de TP.

— Séance TP 4 retour global par le chargé de TP sur les raffinages.

— Samedi 7 octobre 2023 : date limite pour rendre la version finale des livrables : toutes les rubriques du rapport, les fichiers sources, le questionnaire, la version PDF du rapport.

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).

```
1 Ri : Comment « Action complexe » ?
2     actions...
3     liées par des...
4     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. Une expression complexe décrit la valeur de cette expression
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 :

```
1   Demander un numéro de mois      Mois : out Entier
2   { Mois >= 1 ET Mois <= 12 }
3   ...
```

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.
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.