Module M1103 – Janvier 2017
Semaine bloquée - Projet « ANTI-VIRUS »

Présentation du sujet

- L'objectif de ce projet est de développer, en binôme, une application qui permette à un utilisateur de jouer à l'anti-virus à l'aide d'une interface graphique « conviviale ». L'application devra proposer à l'utilisateur des grilles de différentes complexités, permettre de jouer en respectant les règles et afficher un score final. Elle pourra également permettre de résoudre ces grilles si l'utilisateur n'y arrive pas. Un historique des scores des meilleurs joueurs pourra également être géré.
- L'application sera organisée en plusieurs paquetages qui vous seront présentés au fur et à mesure.
- Votre projet doit aboutir à la production de deux programmes : le premier, que l'on nommera **antivirus.adb** vous permettra de tester avec une interface classique (en mode texte) l'ensemble de vos fonctions et procédures. Lorsque celles-ci auront été validées, un second programme, **avgraphique.adb** dotera l'application d'une interface graphique qui réagira dynamiquement aux actions de l'utilisateur.

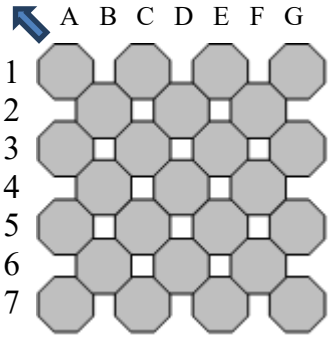
Règles du jeu

Anti-virus est un jeu SmartGames de logique dans lequel il faut sortir le virus d'une cellule pour gagner. Le virus (la pièce rouge) et les autres molécules (les autres pièces de couleur) se gênent dans leurs déplacements dans la cellule (le plateau de jeu) :

http://www.smartgames.eu/fr/smartgames/anti-virus

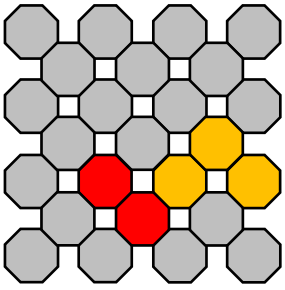
Les pièces glissent en diagonale. Le jeu est fourni avec un livret de 60 challenges triés par difficulté croissante.

- Le plateau de jeu est une grille carrée de 7 lignes et 7 colonnes contenant 25 cases :



- ✓ Les colonnes de la grille sont identifiées par une lettre majuscule de A à G
- ✓ Les lignes de la grille sont identifiées par un entier de 1 à 7
- ✓ La case située à l'intersection de la troisième colonne et de la cinquième ligne sera ainsi identifiée par le couple (C,5)
- ✓ Les lignes paires ont 3 cases et les lignes impaires ont 4 cases. Attention, certaines cases n'existent pas (par exemple (C,4)).

- Le jeu comporte 9 pièces de couleur : la pièce rouge (le virus à éliminer) ainsi que 8 pièces mobiles de couleurs différentes (turquoise, orange, rose, marron, bleu, violet, vert et jaune). Les pièces sont de taille et de formes différentes. Elles occupent 2 ou 3 cases lorqu'elles sont présentes sur la grille.
- Il existe également 2 petites pièces blanches qui occupent chacune une case et qui ne peuvent pas être déplacées au cours de la partie. Elles gênent donc les pièces de couleur dans leurs déplacements sur la grille.
- Les pièces de couleur se déplacent une par une[1] en diagonale (elles ne peuvent pas se déplacer horizontalement ni verticalement, ni pivoter). Il y a donc 4 directions possibles : bas/gauche, haut/gauche, bas/droite et haut/droite. Pour que le déplacement soit possible, il faut que les cases de destination existent et soient libres.



Exemple : pour la pièce orange :

direction haut/gauche : possible
direction bas/gauche : impossible (à cause de la pièce rouge)
direction haut/droite ou bas/droite : impossible (à cause du bord du plateau)

- Le début d'une partie est donné par une configuration initiale qui indique quelles pièces sont utilisées et comment elles sont placées. Le virus (pièce rouge) est bien sûr toujours présent et occupe toujours 2 cases. Une configuration comporte 0, 1 ou 2 pièces blanches fixes. Le but est de déplacer les pièces afin de permettre au virus de sortir par le coin situé en haut à gauche.

[1] *Dans le jeu Smartgames, les déplacements groupés sont possibles mais vous ne les gèrerez pas dans ce projet.*

Partie 1 – Affichage simple de la grille initiale

L'objectif de cette partie est de :

- récupérer dans un fichier une configuration initiale de jeu ;
- stocker dans un vecteur la position des pièces sur le plateau de jeu ;
- afficher la grille de jeu.

Vous disposez de la spécification du paquetage **p_virus** décrit ci-dessous (le corps sera développé ultérieurement, au fur et à mesure des questions et vous pourrez compléter ce paquetage en fonction de vos besoins) :

```
with sequential_IO;
with p_esiut; use p_esiut;

package p_virus is

--------------- Types pour representer les pieces et la grille de jeu

   subtype T_Col is character range 'A'..'G';
   subtype T_Lig is integer range 1..7;
   type T_Piece is (rouge, turquoise, orange, rose, marron, bleu, violet, vert, jaune, blanc, vide);

   type TR_Piece is record
      colonne    :   T_Col;
      ligne      :   T_Lig;
      couleur    :   T_Piece range rouge..blanc;
   end record;

   ---- Instanciation de sequential_IO pour le fichier de description de la grille
   package p_Piece_IO is new sequential_IO (TR_Piece); use p_Piece_IO;

   ---- type pour le vecteur representant les pièces du jeu
   type TV_Virus is array (T_lig,T_col) of T_Piece;[2]

   ---- type pour la direction des deplacements des pieces
   type T_Direction is (bg, hg, bd, hd);
   package p_Direction_IO is new p_enum(T_Direction);

--------------- Creation et Affichage de la grille

procedure CreeVectVirus (f : in out file_type; nb : in integer; V :out TV_Virus);
-- {f (ouvert) contient des configurations initiales,
-- toutes les configurations se terminent par la position du virus rouge}
-- => {V a été initialisé par lecture dans f de la partie de numéro nb}

procedure AfficheVectVirus (V : in TV_Virus);
-- {} => {Les valeurs du vecteur V sont affichées sur une ligne}

procedure AfficheGrille (V : in TV_Virus);
-- {} => {Le contenu du vecteur V est affiché dans une grille symbolisée (cf. sujet, page 3)
-- Les colonnes sont numérotées de A à G et les lignes sont numérotées de 1 à 7.
-- Dans chaque case : le caractère '.' = case vide
--                    un chiffre = numéro de la couleur de la pièce présente dans la case
--                    le caractère 'B' = pièce blanche fixe
--                    rien = pas une case}


--------------- Fonctions et procedures pour le jeu

function Gueri (V : in TV_Virus) return Boolean;
-- {} => {résultat = la pièce rouge est prête à sortir (coin haut gauche)}

function Presente (V : in TV_Virus; Coul : in T_Piece) return Boolean;
-- {} => {résultat =  la pièce de couleur Coul appartient à V}

function Possible (V : in TV_Virus; Coul : in T_Piece; Dir : in T_Direction) return Boolean;
-- {P appartient à la grille V} => {résultat = vrai si la pièce de couleur Coul peut être
--                                  déplacée dans la direction Dir}

procedure Deplacement(V : in out TV_Virus; Coul : in T_Piece; Dir :in T_Direction);
-- {la pièce de couleur Coul peut être déplacée dans la direction Dir}
--                                        => {V a été mis à jour suite au déplacement}

end p_virus;
```

[2] *Voir le polycopié Ada pour les vecteurs à 2 dimensions. Un élément d'un vecteur V de type TV_Virus se note V(i,j).*

Les configurations initiales sont stockées les unes à la suite des autres dans un fichier **Parties** dont les éléments sont de type **TR_Piece**. Il y a 20 configurations différentes dans le fichier.

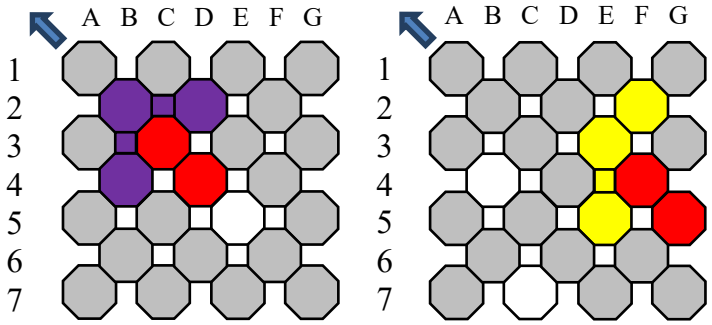
- Les pièces du jeu sont modélisées dans le fichier par un ou plusieurs éléments dont le champ **couleur** est identique.
- Pour une configuration donnée, les constituants de chaque pièce sont stockés consécutivement.
- Il n'y a pas deux pièces mobiles de la même couleur pour une configuration donnée.
- Une configuration se termine toujours par la modélisation de la pièce rouge.

Le fichier **Parties** commence de la façon suivante :

Fin de la 1ère configuration *Fin de la 2ème configuration*

B 4 violet	B 2 violet	D 2 violet	E 5 blanc	C 3 rouge	D 4 rouge	E 5 jaune	E 3 jaune	F 2 jaune	B 4 blanc	C 7 blanc	F 4 rouge	G 5 rouge	•••

Les 2 premières parties sont donc :



Travail à faire :

- Recopiez dans votre répertoire les fichiers du répertoire **/users/info/pub/1a/M1103/antivirus/**
- Ecrivez les fonctions et procédures du paquetage **p_virus** :
  - ✓ procédure **CreeVectVirus** : chargement de la partie demandée à partir du fichier **Parties** dans un vecteur à 2 dimensions de type **TV_Virus** représentant la grille de jeu dans sa situation initiale
  - ✓ procédure **AfficheVectVirus** : affichage simple des valeurs stockées dans le vecteur (pour vérification)
  - ✓ procédure **AfficheGrille** : affichage du vecteur sous forme de grille en indiquant les numéros de lignes et de colonnes. Pour chaque case de la grille, on affichera un caractère représentant ce qu'elle contient :
    - ➢ . = case vide
    - ➢ chiffre = numéro de la couleur de la pièce présente dans cette case (en utilisant sa position dans le type énuméré **T_Piece**)
    - ➢ B = pièce blanche fixe
    - ➢ rien = élément du vecteur qui ne correspond pas à une case
- Testez au fur et à mesure dans un programme **antivirus.adb :**
  - ✓ Saisie d'un numéro de configuration à jouer (entier entre 1 et 20)
  - ✓ Chargement de la configuration demandée
  - ✓ Affichage de la configuration : affichage simple, puis affichage sous forme de grille

REMARQUES : *N'hésitez pas à ajouter d'autres fonctions ou procédures au paquetage si elles vous semblent nécessaires. Pensez également à traiter si besoin des exceptions (sur le fichier lu, sur le choix de la partie, etc.)*

Exemples d'affichage de la grille (pour les 2 première configurations)

```
Sortie
      \  A B C D E F G
       \ =============
      1| .   .   .   .
      2|   6   6   .
      3| .   0   .   .
      4|   6   0   .
      5| .   .   B   .
      6|   .   .   .
      7| .   .   .   .
```

```
Sortie
      \  A B C D E F G
       \ =============
      1| .   .   .   .
      2|   .   .   8
      3| .   .   8   .
      4|   B   .   0
      5| .   .   8   0
      6|   .   .   .
      7| .   B   .   .
```

La pièce de numéro 0 correspond au virus rouge.

Partie 2 – Déplacement des pièces et jeu en mode texte

L'objectif de cette partie est de :

- déplacer les pièces en fonction des choix de l'utilisateur ;
- afficher la grille au fur et à mesure du jeu ;
- annoncer lorsque la partie est gagnée (c'est-à-dire lorsque le virus est placé en haut à gauche).

Travail à faire :

Complétez le programme principal **antivirus.adb** qui devra (tant que la partie n'est pas gagnée) :

- ✓ demander le numéro de la pièce à déplacer et la direction du déplacement, puis effectuer ce déplacement s'il est possible
- ✓ afficher la grille après le déplacement et détecter si le virus est prêt à être éliminé.

Vous rajouterez pour cela dans le paquetage **p_virus** :

- ✓ une fonction booléenne **Presente** : vérifie si une pièce est sur le plateau de jeu
- ✓ une fonction booléenne **Possible** : indique si le déplacement d'une pièce est possible
- ✓ une procédure **Deplacement** : effectue un déplacement possible
- ✓ une fonction booléenne **Gueri** : indique si la partie est gagnée

Partie 3 – Vue graphique

Après avoir développé et testé les différentes fonctionnalités de votre application grâce aux interfaces "rustiques" en mode texte, vous allez maintenant développer une interface graphique un peu plus agréable.

Pour réaliser cette interface, nous vous proposons un paquetage public[3] **`p_fenbase`** (voir **Annexe I**).

Ce paquetage permet, **après l'appel** d'**`InitialiserFenetres`**, de réaliser les opérations décrites ci-dessous :

- **Création d'une fenêtre** et ajout dans cette fenêtre d'éléments graphiques **de 7 types possibles** : des boutons classiques, des boutons avec des images dessus (au format .xpm), des boîtes à cocher (check box), des textes "fixes" (non modifiables par l'utilisateur de l'interface), des champs de saisie (modifiables eux !), des textes "fixes" avec ascenseur (pour afficher un texte de plusieurs lignes) et des horloges (qui affichent l'heure !).
  - ✓ La création d'une fenêtre commence par l'appel à la fonction **`DebutFenetre`**.
  - ✓ Les ajouts d'éléments se font à l'aide de 7 fonctions relatives à chaque type d'élément (**`AjouterBouton`**, **`AjouterBoutonRond`**, **`AjouterTexte`**, **`AjouterChamp`**, **`AjouterTexteAscenseur`**, **`AjouterHorloge`**, **`AjouterBoiteCocher`**, **`AjouterBoutonImage`**).
  - ✓ La création d'une fenêtre se termine par un appel à la procédure **`FinFenetre`**.
  - ✓ Chaque élément ajouté doit avoir un **nom unique** pour sa fenêtre. C'est grâce à ce nom qu'on pourra le manipuler par la suite.
  - ✓ Attention, il faut construire les fenêtres d'une application **les unes après les autres** (on en commence une, on lui ajoute des éléments, on la termine et seulement alors on passe à une autre fenêtre…).
  - ✓ Les fenêtres ainsi créées peuvent ensuite être **affichées ou masquées** à l'utilisateur (procédures **`MontrerFenetre`** et **`CacherFenetre`**).
  - ✓ On dispose d'une fonction (**`AttendreBouton`**) qui permet **d'attendre jusqu' à ce que l'utilisateur ait pressé un bouton** dans une fenêtre. Cette fonction renvoie en résultat le nom du bouton pressé. On dispose également d'une fonction (**`ClickDroit`**) qui indique si le choix a été effectué avec le bouton droit de la souris.
  - ✓ On dispose d'une fonction pour **consulter le contenu d'un champ** de saisie (**`ConsulterContenu`**) pour savoir ce que l'utilisateur y a tapé (la fonction renvoie une chaîne) et d'une fonction qui permet de consulter si une boite à cocher est activée ou non (**`EtatBoiteCocher`**).
- Enfin on dispose de procédures qui permettent de **modifier l'apparence et le comportement des éléments d'une fenêtre** :
  - ✓ **`ChangerTexte`** qui permet de modifier le texte associé à un élément (par exemple le texte affiché dans un bouton)
  - ✓ **`ChangerContenu`** qui permet de modifier le contenu d'un champ de saisie ou d'une zone texte avec ascenseur
  - ✓ **`ChangerCouleurTexte`** et **`ChangerCouleurFond`** qui permettent de modifier la couleur du contenu (texte) ou du fond d'un élément
  - ✓ **`ChangerStyleTexte`** et **`ChangerTailleTexte`** qui permettent de modifier respectivement le style (*standard, gras, italique,…*) et la taille (*standard, moyen, grand, très grand*) d'un texte de bouton, de champ texte ou de zone de saisie (*avec ou sans ascenseur*). Voir en annexe, un tableau récapitulatif des styles et tailles disponibles.
  - ✓ **`ChangerEtatBouton`** qui permet d'activer/désactiver un bouton (quel que soit son type) dans une fenêtre (un bouton désactivé est toujours affiché mais si l'on clique dessus, c'est sans effet…)
  - ✓ **`ChangerImageBouton`** qui permet de modifier l'image d'un bouton « image »
  - ✓ **`MontrerElem`**, **`CacherElem`** qui permettent respectivement de montrer un élément de la fenêtre préalablement caché ou de cacher un élément de la fenêtre jusque-là visible (par défaut tous les éléments ajoutés à une fenêtre sont visibles).

Pour vous guider dans l'utilisation du paquetage **`p_fenbase`**, nous vous fournissons le code d'un petit jeu « stupide » : **`marketing.adb`**. Vous pouvez étudier et exécuter ce programme (voir **Annexe II**)… Certaines de ses astuces peuvent vous êtres précieuses pour développer votre programme. En particulier, vous pouvez remarquer que dans la boucle principale de jeu, la déclaration dynamique de la variable **`Bouton`** permet d'effectuer un seul appel à **`AttendreBouton`** et d'éviter ainsi d'avoir à cliquer plusieurs fois …

(*NOTE : Vous pouvez faire nettement mieux en termes de lisibilité et réutilisabilité de votre code !)*

[3] *Il ne faut donc pas recopier ce paquetage…*

Le fonctionnement de votre application pourrait être le suivant :

- Au démarrage, une première fenêtre est affichée pour demander à l'utilisateur de choisir le numéro de la partie à laquelle il souhaite jouer.
- Cette première fenêtre est ensuite masquée et une autre fenêtre (dite de jeu) s'ouvrira. Elle affichera la grille correspondant à la partie demandée. La grille sera représentée par des boutons (un bouton par case) :
  - ✓ Les boutons représentant les pièces mobiles seront en couleur. Ils seront actifs (on pourra cliquer dessus pour choisir la pièce que l'on souhaite déplacer).
  - ✓ Les boutons gris (pas de pièce) ou blanc (pièce fixe) ne seront pas actifs (cliquer dessus n'aura aucun effet) .
- Cette fenêtre de jeu comportera également d'autres éléments pour permettre le jeu :
  - ✓ 4 boutons pour indiquer la direction dans laquelle on souhaite déplacer une pièce préalablement choisie.
  - ✓ un bouton « règles du jeu »
  - ✓ un bouton « quitter » si on souhaite arrêter le jeu avant d'avoir trouvé la solution
  - ✓ une zone de texte pour afficher des messages
- A chaque coup, l'utilisateur pourra donc choisir la pièce et la direction en cliquant sur une case puis sur un des 4 boutons de direction. Si le déplacement est possible, la grille sera mise à jour. Dans le cas contraire, un message d'erreur sera affiché.
- Dans la zone de texte, vous afficherez des messages pour l'utilisateur (aide, erreur, …) ainsi qu'un score composé du nombre de coups joués et du nombre d'erreurs commises depuis le début d'une partie.
- A tout moment, le joueur peut cliquer sur le bouton « règles du jeu », et une 3ème fenêtre doit alors s'afficher avec les règles du jeu. Lorsque le joueur ferme cette fenêtre, il doit pouvoir reprendre sa partie.
- Lorsque le virus est prêt à être éliminé, un message de félicitations est affiché ainsi que le score. Tous les boutons sont alors inactifs sauf le bouton « quitter » (et éventuellement un bouton « rejouer »).

Travail à faire :

- Créez un nouveau paquetage **p_vuegraph** qui contiendra toutes les fonctions et/ou procédures nécessaires à la gestion de la vue graphique. Ce paquetage utilisera bien sûr les types, fonctions et procédures définis dans les paquetages **p_virus** et **p_fenbase**.
- Créez un nouveau programme **avgraphique.adb** qui utilisera ce paquetage ainsi que tous ceux qui lui seront utiles pour :
  - ✓ afficher la fenêtre du choix de la partie,
  - ✓ afficher la partie choisie dans la fenêtre de jeu,
  - ✓ jouer en effectuant des clics sur les boutons,
  - ✓ détecter la fin du jeu,
  - ✓ afficher un score.

Remarques :

- commencez dans un premier temps, avec uniquement une fenêtre simple. Par exemple, développez une fenêtre d'accueil, puis basculez en « vue texte » pour le jeu lui-même. Vous complèterez le paquetage au fur et à mesure de l'écriture de la « vue graphique ».
- pour compiler un programme utilisant les bibliothèques graphiques, utilisez la commande **build** (en ayant au préalable exécuté une fois la commande **export LD_LIBRARY_PATH=/users/info/pub/1a/ada/bib/lib**).
- faites en sorte que votre code soit plus lisible que le programme **marketing** !!

Partie 4 – Extensions

Si votre interface graphique fonctionne correctement, vous pouvez imaginer et programmer des extensions. Par exemple :

- Afficher une horloge et éventuellement intégrer le temps dans le calcul du score
- Gérer un fichier de scores de différents utilisateurs
- Enregistrer les coups joués lors d'une partie
- Donner la possibilité de défaire des coups
- Proposer différents niveaux de jeu. Dans le fichier **Parties**, les configurations initiales sont classées par ordre de difficulté croissante et l'on peut considérer qu'il y a 4 niveaux :
  - ✓ *starter* des parties 1 à 5
  - ✓ *junior* des parties 6 à 10
  - ✓ *expert* des parties 11 à 15
  - ✓ *master* des parties 16 à 18
  - ✓ *wizard* des parties 19 à 20
- Créer un fichier de solutions des parties pour pouvoir les proposer à l'utilisateur
- Permettre de créer de nouvelles parties …

Annexe I - Spécification du paquetage p_fenbase

```
with Forms; use Forms; with Interfaces.C;use Interfaces.C;
package  p_fenbase is
  ------------------------------------------------------------------------------
  -- types à connaître pour utiliser le paquetage
  type    TR_Fenetre;                       -- une fenêtre de l'interface graphique
  type    T_EtatBouton is (Marche, Arret);  -- état d'un bouton (actif ou inactif)
  subtype T_Piece is FL_COLOR;             -- couleurs disponibles dans le paquetage forms
  ------------------------------------------------------------------------------
  -- types internes au paquetage, inutile de les connaître !
  type TA_String is access String;       -- pointeur sur une chaîne de caractères
  type T_TypeElement is (Bouton, TexteFixe, ChampDeSaisie, TexteAscenseur, Horloge, Fond, CheckBox, PictBouton);
  type TR_Element;
  type TA_Element is access TR_Element;
  type TR_Element is record
    TypeElement : T_TypeElement;
    NomElement  : TA_String;
    Texte       : TA_String;
    Contenu     : TA_String;
    PElement    : FL_OBJECT_Access;
    Suivant     : TA_Element;
  end record;
  type TR_Fenetre is record
    PFenetre  : FL_FORM_Access;
    Titre     : TA_String;
    PElements : TA_Element;
  end record;
  ------------------------------------------------------------------------------
  -- primitives de la bibliothèque "graphique"
  ------------------------------------------------------------------------------
  procedure InitialiserFenetres;                  -- Initialiser le mode graphique
  ------------------------------------------------------------------------------
  function DebutFenetre (                         -- Créer une nouvelle fenêtre
        Titre   : in     String;                  -- son titre
        Largeur,                                  -- sa largeur en pixels
        Hauteur : in     Positive )               -- sa hauteur en pixels
    return TR_Fenetre;                            -- résultat : la fenêtre créée
  ------------------------------------------------------------------------------
  procedure AjouterBouton (                       -- Ajouter un Bouton
        F          : in out TR_Fenetre;           -- la fenêtre où on ajoute
        NomElement : in     String;               -- le nom du bouton (unique)
        Texte      : in     String;               -- le texte affiché dans le bouton
        X,                                        -- son abscisse en pixels
        Y          : in     Natural;              -- son ordonnée en pixels
        Largeur,                                  -- sa largeur en pixels
        Hauteur    : in     Positive    );       -- sa hauteur en pixels
  ------------------------------------------------------------------------------
  procedure AjouterBoutonRond (                   -- Ajouter un Bouton de forme ronde
        F          : in out TR_Fenetre;           -- la fenêtre où on ajoute
        NomElement : in     String;               -- le nom du bouton (unique)
        Texte      : in     String;               -- le texte affiché dans le bouton
        X,                                        -- son abscisse en pixels
        Y          : in     Natural;              -- son ordonnée en pixels
        Largeur,Hauteur    : in     Positive    );   -- sa largeur et sa hauteur en pixels
  ------------------------------------------------------------------------------
  procedure AjouterTexte (                        -- Ajouter un texte non modifiable
        F          : in out TR_Fenetre;           -- par l'utilisateur dans f
        NomElement : in     String;
        Texte      : in     String;               -- le texte qui sera affiché
        X, Y       : in     Natural;
        Largeur, Hauteur    : in     Positive    );
  ------------------------------------------------------------------------------
  procedure AjouterChamp (                        -- Ajouter un champ de saisie
        F          : in out TR_Fenetre;
        NomElement : in     String;
        Texte      : in     String;               -- le texte affiché en légende
        Contenu    : in     String;               -- le contenu initial du champ
        X, Y       : in     Natural;
        Largeur, Hauteur    : in     Positive    );
  ------------------------------------------------------------------------------
  procedure AjouterTexteAscenseur (               -- Ajouter une zone texte avec
        F          : in out TR_Fenetre;           -- ascenseur (plusieurs lignes)
        NomElement : in     String;
        Texte      : in     String;               -- le texte affiché en légende
        Contenu    : in     String;               -- le contenu de la zone
        X, Y       : in     Natural;
        Largeur, Hauteur    : in     Positive    );
  ------------------------------------------------------------------------------
  procedure AjouterHorloge (                      -- Ajouter une horloge qui affiche
        F          : in out TR_Fenetre;           -- l'heure courante
        NomElement : in     String;
        Texte      : in     String;               -- le texte affiché en légende
        X, Y       : in     Natural;
        Largeur, Hauteur    : in     Positive    );
  procedure AjouterBoiteCocher (                  -- Ajouter une boîte à cocher
        F          : in out TR_Fenetre;
```

```
      NomElement : in     String;
      Texte      : in     String;              -- le texte affiché en légende
      X, Y       : in     Natural;
     Largeur, Hauteur     : in     Positive    );
  ------------------------------------------------------------------------------
  procedure AjouterBoutonImage (                -- Ajouter un bouton avec image
      F          : in out TR_Fenetre;
      NomElement : in     String;
      Texte      : in     String;              -- le texte affiché en légende
      X, Y       : in     Natural;
     Largeur, Hauteur     : in     Positive    );
  ------------------------------------------------------------------------------
  procedure FinFenetre (                        -- Procedure à appeler quand on a
      F : in     TR_Fenetre );                  -- fini d'ajouter des éléments
  ------------------------------------------------------------------------------
  procedure MontrerFenetre (                    -- Afficher unefenêtre
      F : in     TR_Fenetre );
  ------------------------------------------------------------------------------
  procedure CacherFenetre (                     -- Masquer une fenêtre
      F : in     TR_Fenetre );
  ------------------------------------------------------------------------------
  function AttendreBouton (                     -- Attendre qu'un bouton soit pressé
      F : in     TR_Fenetre )                   -- dans la fenetre f
    return String;                              -- Résultat = nom du bouton pressé
  ------------------------------------------------------------------------------
  function ClickDroit return boolean;           -- vrai si on a pressé le bouton droit de la souris
  ------------------------------------------------------------------------------
  function ConsulterContenu (                   -- Permet de récupérer la valeur
      F          : in     TR_Fenetre;          -- saisie par l'utilisateur dans
      NomElement : in     String      )         -- un champ de saisie
    return String;                              -- Résultat : la chaîne saisie !
  ------------------------------------------------------------------------------
  function EtatBoiteCocher (                    -- Permet de savoir si une Checkbox est
      F          : in     TR_Fenetre;          -- cochée ou non
      NomElement : in     String      )
    return boolean;                             -- Résultat : l'état du bouton
  ------------------------------------------------------------------------------
  procedure ChangerTexte (                      -- Permet de modifier l'attribut
      F            : in out TR_Fenetre;         -- texte d'un élément de f
      NomElement   : in     String;             -- Le nom de l'élément à modifier
      NouveauTexte : in     String      );      -- La nouvelle valeur de texte
  ------------------------------------------------------------------------------
  procedure ChangerContenu (                    -- Permet de changer le contenu
      F              : in out TR_Fenetre;       -- d'un champ de saisie ou d'une
      NomElement     : in     String;           -- zone texte avec ascenseur
      NouveauContenu : in     String      );    -- La nouvelle valeur de contenu
  ------------------------------------------------------------------------------
  procedure ChangerCouleurTexte (               -- Permet de changer la couleur
      F               : in out TR_Fenetre;      -- du texte d'un element dans une
      NomElement      : in     String;          -- fenetre
      NouvelleCouleur : in     T_Piece   ) ; --    La nouvelle couleur !
  ------------------------------------------------------------------------------
  procedure ChangerCouleurFond (                -- Permet de changer la couleur
      F               : in out TR_Fenetre;      -- de fond d'un élément dans une
      NomElement      : in     String;          -- fenêtre
      NouvelleCouleur : in     T_Piece   );     -- La nouvelle couleur !
  ------------------------------------------------------------------------------
  procedure ChangerStyleTexte (                 -- Permet de changer le style d'un texte
      F               : in out TR_Fenetre;      -- d'un élément dans une
      NomElement      : in     String;          -- fenêtre
      NouveauStyle : in     FL_TEXT_STYLE   ); -- Le nouveau style !
  ------------------------------------------------------------------------------
  procedure ChangerTailleTexte (                -- Permet de changer la taille d'un texte
      F               : in out TR_Fenetre;      -- d'un élément dans une
      NomElement      : in     String;          -- fenêtre
     Taille : in     X11.Int   );               -- La nouvelle taille !
  ------------------------------------------------------------------------------
  procedure ChangerEtatBouton (                 -- Permet d'activer ou de désactiver un bouton
      F          : in out TR_Fenetre;           -- ou une boîte à cocher. S'il est désactivé, on
      NomElement : in     String;               -- ne peut plus cliquer dessus.
      Etat       : in     T_EtatBouton );       -- valeur : marche (activé) ou arret
  ------------------------------------------------------------------------------
  procedure ChangerImageBouton (                -- Change l'image d'un bouton image
      F          : in out    TR_Fenetre;
      NomElement, NomImage : in String );       -- NomImage est le nom du fichier contenant l'image

  ------------------------------------------------------------------------------
  procedure MontrerElem (F: in out TR_Fenetre; NomElement : in String );                                -
  -- Permet de montrer un élément jusque-là caché
  ------------------------------------------------------------------------------
  procedure CacherElem (F: in out TR_Fenetre; NomElement : in String );                                 -
  -- Permet de cacher un élément jusque-là visible
end p_fenbase;
```

Annexe II – Marketing.adb

```
with p_fenbase ; use p_fenbase ;
with Forms ; use Forms;
with p_esiut; use p_esiut;
with ada.calendar; use ada.calendar;
procedure marketing is
  FSaisieNom,  FJeu,  FResultat  : TR_Fenetre;   -- l'application comporte 3 fenetres
  Compteur    : Natural;
  Touche      : Character;
  I,  J       : Natural;
  Nombouton   : String(1..2);
  HeureDeb,HeureFin : Time;
  NewLine     : constant Character  := Character'Val (10);   -- retour chariot
begin
   -- on initialise l'interface graphique (a ne faire qu'une fois en début de programme)
   InitialiserFenetres;
   -- on crée la fenêtre pour saisir le nom du "gogo"
   FSaisieNom:=DebutFenetre("Nom du Joueur",400,70);
      AjouterChamp(FSaisieNom,"ChampNom","Votre Nom","quidam",100,10,280,30);
      AjouterBouton(FSaisieNom,"BoutonValider","valider",100,50,70,30);
      AjouterBouton(FSaisieNom,"BoutonAnnuler","annuler",180,50,70,30);
   FinFenetre(FSaisieNom);
   -- on crée la fenêtre pour "jouer"
   FJeu:=DebutFenetre("CHANCE",300,400);
      AjouterTexte(FJeu,"message1","TAPE SUR 3 TOUCHES AU HASARD...",10,10,250,30);
      AjouterTexte(FJeu, "message2","Tente ta chance !",10,50,280,30);
      -- on ajoute une grille de 3x3 boutons affichant les valeurs de 1 à 9
      -- on donne à chaque bouton un nom d'élément qui représente sa position dans la grille
      -- le bouton de la ligne 2 colonne 2 s'appellera donc "23"
      for I in 1..3 loop
         for J in 1..3 loop
            nombouton := Integer'Image(I)(2..2) & Integer'Image(J)(2..2);
            -- le bouton en i,j s'appelle "ij" et affiche la valeur (i-1)*3+J
            AjouterBouton(FJeu,nombouton,integer'image((i-1)*3+J),(J-1)*60+40,(I-1)*60+90,60,60);
            ChangerCouleurFond(FJeu,nombouton,FL_DARKGOLD);
            ChangerTailleTexte(Fjeu,nombouton,FL_Large_Size);
            ChangerStyleTexte(Fjeu,nombouton,FL_Bold_Style);
          end loop;
      end loop;
      AjouterTexte(FJeu,"BarreDEtat","Aucune touche pressee",10,300,250,30);
      AjouterBouton(FJeu,"Fin","FIN",55,390,70,30);
      ChangerStyleTexte(FJeu, "Fin", FL_BOLD_Style);
      ChangerTailleTexte(Fjeu,"Fin",FL_medium_size);
      AjouterHorloge(Fjeu,"Clock","",150,320,100,100);
      AjouterBouton(FJeu,"BoutonAbandonner","abandon",55,350,70,30);
   FinFenetre(FJeu);
   -- on crée la fenêtre pour afficher le "gain"
   Fresultat:=Debutfenetre("Votre gain ! ",300,150);
      AjouterTexteAscenseur(FResultat,"message","","",10,10,280,100);
      AjouterBouton(FResultat,"BoutonFin","The End",115,140,70,30);
   FinFenetre(FResultat);
   --------------------------------------------------------------------------
   -- début du jeu, on montre la première fenêtre et on attend un bouton
   MontrerFenetre(FSaisieNom);
   if AttendreBouton(FSaisieNom)/="BoutonAnnuler" then
      CacherFenetre(FSaisieNom);
      Compteur:=0; -- nombre de touches pressées
      ChangerTexte(Fjeu,"message2", ConsulterContenu(FSaisieNom,"ChampNom")& " tente ta chance !");
      MontrerFenetre(FJeu);
      HeureDeb:=clock;
      -- interdiction de cliquer sur Fin tant qu'on n'a pas joué ou abandonné
      ChangerEtatbouton(FJeu,"Fin",Arret);
      loop -- BOUCLE jusqu'a la fin du jeu ou l'abandon
         declare
            Bouton : String := (Attendrebouton(Fjeu));
         begin
            if Bouton /= "BoutonAbandonner" and Bouton /= "Fin" then - c'est-à-dire clic sur un bouton "ij"
               if compteur < 3 then
                  Compteur:=Compteur+1; -- une nouvelle touche a été pressée
                  ChangerEtatBouton(FJeu, Bouton, Arret); -- bouton rendu inactif
                  ChangerCouleurFond(Fjeu, Bouton, FL_RED); -- et mis en rouge
                  ChangerTailleTexte(Fjeu,Bouton,FL_normal_size); -- pour que le texte rentre dans la case
                  ChangerTexte(Fjeu, Bouton, "GAGNE"); -- et c'est l'arnaque !!!
                  -- on calcule les coordonnées (i,j) dans la grille du bouton pressé grâce au nom du bouton...
                  I:=Character'Pos(Bouton(Bouton'First)) - Character'Pos('0');
                  J:=Character'Pos(Bouton(Bouton'Last))  - Character'Pos('0');
                  -- on calcule la "valeur" du bouton pressé et on la convertit en char
                  Touche:=Character'Val(((I-1)*3+J)+Character'Pos('0'));
                  -- on changer l'objet texte "BarreDetat" en fonction de la touche pressée
                  ChangerTexte(Fjeu,"BarreDEtat","La touche " & Touche & " a ete pressee");
                  if compteur = 3 then -- C'est gagné !!!
                     for I in 1..3 loop -- on désactive toutes les cases du jeu (sans faire de détail !...)
                        for J in 1..3 loop
                           nombouton := Integer'Image(I)(2..2) & Integer'Image(J)(2..2);
                           ChangerEtatBouton(FJeu,nombouton,Arret);
                        end loop;
                     end loop;
```

```
                    HeureFin:= clock;
                    ChangerTexte(Fjeu,"message2","BRAVO !!! tu as gagne en"
                    & natural'image(natural(Heurefin-HeureDeb))& "'");
                    ChangerStyleTexte(FJeu, "message2", FL_BOLD_Style);
                    ChangerTailleTexte(Fjeu,"message2",FL_medium_Size);
                    ChangerEtatBouton(FJeu,"Fin",Marche);
                 end if;
              end if;
           elsif Bouton = "Fin" then
              exit;
           else -- Bouton Abandonner
              compteur := 0;
              exit;
           end if;
        end;
     end loop;
     CacherFenetre(FJeu);
     if Compteur=0 then -- le joueur a abandonné
        ChangerCouleurFond(FResultat,"fond",FL_BLACK);-- fond noir pour accabler
        ChangerContenu(FResultat,"message","Joueur " & ConsulterContenu(FSaisieNom,"ChampNom") & NewLine &
              "qui ne tente rien n'a rien !!!");
     else -- le joueur a tenté sa chance
        ChangerCouleurFond(FResultat,"fond",FL_RED); -- fond rouge pour encenser
        ChangerContenu(FResultat,"message","Joueur " & ConsulterContenu(FSaisieNom,"ChampNom") &
             ", NOUS T'ATTENDONS ! "  & NewLine & "RDV au magasin, " & NewLine & "TU GAGNERAS ENCORE !!!");
     end if;
     Montrerfenetre(Fresultat); -- on affiche la fenêtre des gains
     loop
        exit when Attendrebouton(Fresultat) = "BoutonFin";
     end loop;
     Cacherfenetre(Fresultat);
  end if;
end marketing;
```

Annexe III – Couleurs disponibles dans le paquetage Forms

COULEUR
FL_TOMATO, FL_INDIANRED, FL_SLATEBLUE, FL_COL1, FL_RIGHT_BCOL, FL_BOTTOM_BCOL FL_TOP_BCOL, FL_LEFT_BCOL, FL_MCOL, FL_INACTIVE, FL_PALEGREEN, FL_ORCHID FL_DARKCYAN, FL_DARKTOMATO, FL_WHEAT, FL_DARKORANGE, FL_DEEPPINK, FL_CHARTREUSE FL_DARKVIOLET, FL_SPRINGGREEN, FL_DODGERBLUE, FL_DARKGOLD, FL_BLACK, FL_RED FL_GREEN, FL_YELLOW, FL_BLUE, FL_MAGENTA, FL_CYAN, FL_WHITE

Annexe IV : Styles et tailles disponibles dans le paquetage Forms

TAILLE	*STYLE*
FL_TINY_SIZE (8 pt) FL_SMALL_SIZE (10 pt) FL_NORMAL_SIZE (12 pt) (valeur par défaut) FL_MEDIUM_SIZE (14 pt) FL_LARGE_SIZE (18 pt) FL_HUGE_SIZE (24 pt)	FL_NORMAL_STYLE – valeur par défaut FL_BOLD_STYLE (gras) FL_ITALIC_STYLE (italique) FL_BOLDITALIC_STYLE (gras italique) FL_FIXED_STYLE (tous les caractères occupent le même espace) FL_FIXEDBOLD_STYLE FL_FIXEDITALIC_STYLE FL_FIXEDBOLDITALIC_STYLE FL_TIMES_STYLE (police ⇔ Times ou Times New Roman déclinable en BOLD, ITALIC, BOLDITALIC) FL_SHADOW_STYLE (ombré) / EMBOSSED (relief) / ENGRAVED (gravé) *Remarque : attention, certains choix de style sont sans effet.*