

## Présentation du sujet

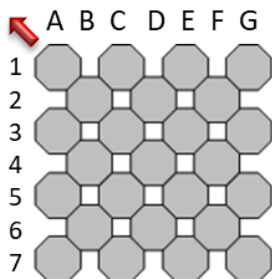
- L'objectif de ce projet est de développer une application qui permette à un utilisateur de jouer via une interface graphique « conviviale » à un jeu nommé « Anti-virus »
- L'application devra proposer à l'utilisateur des parties de différentes complexités, permettre à l'utilisateur de jouer dans le respect des règles du jeu et afficher un score final. Elle pourra être augmentée d'assistances à l'utilisateur, de l'enregistrement de ses scores, de la gestion d'un historique des scores des meilleurs joueurs, etc.
- Votre projet doit aboutir à la production de deux programmes et plusieurs paquetages :
  - ✓ le programme `av_txt.adb` a pour finalité de tester le bon fonctionnement du jeu via une interface texte ;
  - ✓ le programme `av_graph.adb` représentera le jeu doté d'une interface graphique avec laquelle l'utilisateur pourra interagir ;
  - ✓ le paquetage `p_virus` sera partagé par les deux programmes et contiendra les types, sous-types, fonctions et procédures nécessaires au contrôle du déroulement du jeu ;
  - ✓ le paquetage `p_vuetxt` concernera la gestion de l'interface en mode texte ;
  - ✓ le paquetage `p_vuegraph` concernera la gestion de l'interface en mode graphique.

## Règles du jeu

Anti-virus<sup>1</sup> est un jeu diffusé par SmartGames, dans lequel il s'agit de sortir un virus d'une cellule. Le virus et les autres molécules présentes dans la cellule sont représentés par des pièces de couleurs différentes. Ces pièces sont disposées sur un plateau carré alvéolé représentant la cellule. L'objectif est de sortir le virus en un minimum de coups, les pièces se gênant mutuellement dans leurs déplacements. Le jeu est fourni avec un livret de challenges de difficulté croissante.

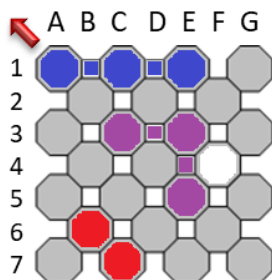


- Le plateau de jeu est une grille carrée de 7 lignes et 7 colonnes contenant 25 cases utilisables sur 49 :



- ✓ 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 et les colonnes paires ont 3 cases, les lignes et les colonnes impaires ont 4 cases.
- ✓ Les cases utilisables sont situées à l'intersection d'une ligne et d'une colonne de même parité

- Le jeu comporte 9 pièces mobiles de couleur : une pièce rouge (représentant le virus) ainsi que 8 pièces 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 lorsqu'elles sont présentes sur la grille.
- Il comporte également 2 pièces blanches fixes, qui occupent chacune une case et gênent le déplacement des pièces de couleur.
- Les pièces de couleur se déplacent une par une<sup>2</sup> en diagonale et ne peuvent pas pivoter. Il y a donc 4 directions possibles : bas/gauche, haut/gauche, bas/droite et haut/droite. Pour que le déplacement d'une pièce mobile soit possible, il faut que les cases de destination existent et soient libres.



### Exemples de déplacement pour la pièce bleue :

- direction haut/gauche, haut/droite ou bas/gauche : impossible (à cause des bords du plateau)
- direction bas/droite : possible

### Exemples de déplacement pour la pièce violette :

- direction haut/gauche ou bas/gauche possible
- direction haut/droite ou bas/droite impossible : à cause de la pièce blanche

- Le début d'une partie est donné par une configuration initiale qui détermine quelles pièces sont utilisées et comment elles sont placées en début de partie. Le virus (pièce rouge) 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.

<sup>1</sup> <http://www.smartgames.eu/fr/smartgames/anti-virus>

<sup>2</sup> Dans le jeu SmartGames, les déplacements groupés sont possibles mais vous ne les gèrerez pas dans ce projet.

# Principes d'implémentation

## Principe général :

- Une pièce placée sur la grille est matérialisée par la couleur des cases qu'elle recouvre.
- Une couleur spéciale (appelée « vide ») permet de savoir si une case utilisable est recouverte ou non.

### Exemple :



	A	B	C	D	E	F	G
1	vide		vide		rouge		vide
2		vide		vide		rouge	
3	vide		vide		vert		vide
4		vert		vert		vide	
5	blanc		orange		marron		vide
6		orange		vide		blanc	
7	vide		orange		marron		vide

## Fichier binaire contenant les différentes configurations initiales de jeu:

- 20 configurations<sup>3</sup> initiales différentes sont stockées les unes à la suite des autres dans un fichier binaire, nommé **Parties**
- Chaque configuration détermine quelles pièces sont utilisées et leur position initiale sur la grille.

Pour une configuration donnée :

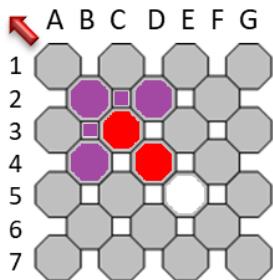
- Il n'y a pas deux pièces mobiles de même couleur.
- Il peut y avoir entre 0 et 2 pièces fixes (couleur blanc).
- Les éléments constituant une pièce mobile sont stockés consécutivement dans le fichier.
- Une configuration se termine toujours par les deux éléments représentant le virus (couleur rouge)

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

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

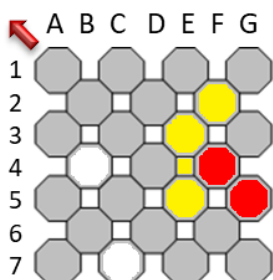
Configuration n° 1
Configuration n° 2

Lors du chargement de la première partie, la grille sera donc initialisée ainsi :



	A	B	C	D	E	F	G
1	vide		vide		vide		vide
2		violet		violet		vide	
3	vide		rouge		vide		vide
4		violet		rouge		vide	
5	vide		vide		vide		vide
6		vide		vide		vide	
7	vide		vide		vide		vide

Lors du chargement de la deuxième partie, la grille sera donc initialisée ainsi :



	A	B	C	D	E	F	G
1	vide		vide		vide		vide
2		vide		vide		jaune	
3	vide		vide		jaune		vide
4		blanc		vide		rouge	
5	vide		vide		jaune		rouge
6		vide		vide		vide	
7	vide		blanc		vide		vide

<sup>3</sup> La configuration initiale des différentes parties vous est fournie en page 15 ([ANNEXE V](#))

Créez un répertoire **SBAda** dans votre répertoire **M1103**, puis copiez-y le contenu du répertoire :  
/users/info/pub/1a/M1103/SBAda/

### A. Mise en place et contrôle du déroulement du jeu

Vous disposez de la spécification du paquetage **p\_virus** (vous pourrez l'augmenter par la suite si besoin) :

```
with sequential_io; with p_esiut; use p_esiut;

package p_virus is

    ----- Types pour représenter la grille de jeu
    subtype T_Col is character range 'A'..'G';
    subtype T_Lig is integer range 1..7;
    type T_Coul is (rouge, turquoise, orange, rose, marron, bleu, violet, vert, jaune, blanc, vide);

    type TV_Grille is array (T_lig, T_col) of T_Coul;4

    ----- Types pour représenter les pièces du jeu
    subtype T_CoulP is T_Coul range rouge..blanc ; -- couleurs des pièces
    package p_coul_io is new p_enum(T_CoulP) ; use p_coul_io ;

    type TR_ElemP is record
        colonne : T_Col;
        ligne : T_Lig;
        couleur : T_CoulP;
    end record;

    type TV_Pieces is array(T_coulP) of boolean ;

    ---- Instanciation de sequential_io pour un fichier de TR_ElemP ;
    package p_piece_io is new sequential_io (TR_ElemP); use p_piece_io;

    ---- type pour la direction de déplacement des pièces
    type T_Direction is (bg, hg, bd, hd);
    package p_dir_io is new p_enum(T_Direction); use p_dir_io;

    ----- Primitives d'initialisation d'une partie

    procedure InitPartie (Grille : in out TV_Grille ; Pieces : in out TV_Pieces);
    -- {} => {Tous les éléments de Grille ont été initialisés avec la couleur vide
    -- y compris les cases inutilisables
    -- Tous les éléments de Pieces ont été initialisés à false}

    procedure Configurer (f : in out p_piece_io.file_type; nb : in positive;
        Grille : in out TV_Grille ; Pieces : in out TV_Pieces);
    -- {f ouvert,nb est un numéro de configuration (appelé numéro de partie),
    -- une configuration décrit le placement des pièces du jeu, pour chaque configuration :
    -- * les éléments d'une même pièce (même couleur) sont stockés consécutivement
    -- * il n'y a pas deux pièces mobiles ayant la même couleur
    -- * les deux éléments constituant le virus (couleur rouge) terminent la configuration}
    -- => {Grille a été mis à jour par lecture dans f de la configuration de numéro nb
    -- Pieces a été initialisé en fonction des pièces de cette configuration}

    -- pour test configuration...
    procedure PosPiece (Grille : in TV_Grille ; coul : in T_coulP);
    -- {} => {la position de la pièce de couleur coul a été affichée si cette pièce est dans Grille
    -- exemple : ROUGE : F4 G5}

    ----- Contrôle du jeu
    function Possible (Grille : in TV_Grille; coul : T_CoulP; Dir : in T_Direction) return boolean;
    -- {coul /= blanc}
    -- => {résultat = vrai si la pièce de couleur coul peut être déplacée dans la direction Dir}

    procedure MajGrille (Grille : in out TV_Grille; coul : in T_coulP; Dir :in T_Direction);
    -- {la pièce de couleur coul peut être déplacée dans la direction Dir}
    -- => {Grille a été mis à jour suite au déplacement}

    function Guerison (Grille : in TV_Grille) return boolean;
    -- {} => {résultat = le virus (pièce rouge) est prêt à sortir (position coin haut gauche)}

end p_virus;
```

<sup>4</sup> Voir le document « Vecteurs à 2 dimensions » sur Chamilo

## Étape 1 : Initialisation d'une partie

- Créez le corps du paquetage `p_virus`, puis développez :
  - ✓ `InitPartie` : initialisation de la grille de jeu et du vecteur représentant les pièces présentes sur la grille avant configuration
  - ✓ `Configurer` : placement sur la grille des pièces d'une configuration dont le numéro d'ordre dans le fichier de configurations est choisi par l'utilisateur + mise à jour du vecteur représentant les pièces présentes sur la grille
  - ✓ `PosPiece` : affichage de la position dans la grille de chaque élément constitutif d'une pièce de couleur donnée
- Créez une procédure principale `testjeu` dans laquelle vous écrirez :
  - ✓ les clauses d'import et d'utilisation des paquetages nécessaires ainsi que les déclarations qui vous semblent utiles
  - ✓ les instructions permettant :
    - d'initialiser la grille de jeu
    - de demander à l'utilisateur de choisir un numéro de configuration (entre 1 et 20)
    - de configurer le jeu en conséquence
    - de tester la configuration obtenue (position de chaque pièce du jeu)
  - ✓ le traitement des exceptions système pouvant se produire

## Étape 2 : Déplacement d'une pièce sur la grille

- Dans `p_virus`, développez les deux fonctions et la procédure permettant de contrôler le jeu :
  - ✓ fonction `Possible` : Vérification de la possibilité de déplacer une pièce dans une direction donnée.  
*Cette fonction est fondamentale pour contrôler le déplacement d'une pièce. Il s'agit de vérifier avant toute tentative de déplacement que, suite à ce dernier, aucun élément d'une pièce ne recouvre un élément d'une autre pièce ou « sorte » de la grille (rappel : les pièces se déplacent toujours en diagonale).*
  - ✓ procédure `MajGrille` : Mise à jour de la grille suite au déplacement (possible) d'une pièce mobile dans une direction donnée.
  - ✓ fonction `Guerison` : Test du gain d'une partie (le virus doit être positionné sur les cases A1 et B2).
- Dans `testjeu` ajoutez les instructions suivantes :
  - ✓ choix d'une pièce à déplacer (la pièce doit bien sûr être mobile et présente sur la grille préalablement configurée)
  - ✓ test de la possibilité de déplacement de cette pièce dans chaque direction et, pour chaque déplacement possible :
    - mise à jour de la grille suite au déplacement de la pièce
    - affichage de la nouvelle position des pièces de la grille
    - réinitialisation, puis reconfiguration de la grille dans la configuration initiale

## B. Interface et jeu en mode texte

- Créez un paquetage `p_vuetxt`, et développez dans ce paquetage la procédure suivante :

```
procedure AfficheGrille(Grille : in TV_Grille) ;
-- {} => {la grille a été affichée selon les spécifications suivantes :
--      * la sortie est indiquée par la lettre S
--      * une case inactive ne contient aucun caractère
--      * une case de couleur vide contient un point
--      * une case de couleur blanche contient le caractère F (Fixe)
--      * une case de la couleur d'une pièce mobile contient le chiffre correspondant à la
--      position de cette couleur dans le type T_Coul}
```

Exemples d'affichage de la grille pour les deux premières configurations :

Configuration n°1								Configuration n° 2							
	A	B	C	D	E	F	G		A	B	C	D	E	F	G
S	-	-	-	-	-	-	-	S	-	-	-	-	-	-	-
1		.	.	.	.	.	.	1		.	.	.	.	.	.
2			6		6		.	2		.	.	.		8	.
3		.		0		.	.	3		.	.	.	8		.
4			6		0		.	4			F	.	.	0	.
5		.	.	.		F	.	5		.	.	.		8	0
6		.	.	.	.	.	.	6		.	.	.	.	.	.
7		.	.	.	.	.	.	7		.	.	F	.	.	.

- Complétez `p_vuetxt` avec tout ce que vous jugerez utile et ne concernant que le jeu en interface texte.  
*Ajoutez par exemple des fonctions ou des procédures guidant l'utilisateur sur ce qu'il doit saisir au clavier pour : choisir une pièce à déplacer, la déplacer dans une direction donnée, annuler le dernier déplacement, changer de pièce à déplacer, ou encore, abandonner la partie...*
- Créez une procédure principale `av_txt` qui doit permettre à l'utilisateur de choisir une configuration de jeu et de jouer dans le but de sortir le virus. Vous veillerez à ce que le jeu soit le plus convivial possible (l'utilisateur doit toujours avoir un retour visuel sur l'effet de ses actions...)

Vous gérerez toutes les causes d'erreur système de façon à ce que le programme se déroule correctement, y compris s'il est lancé par un utilisateur qui ne lit pas les consignes qu'on lui donne ☺

## Partie 2 – Développement d'une interface graphique

Après avoir développé et testé les différentes fonctionnalités de votre application via des interfaces "rustiques", vous allez maintenant développer une interface graphique un peu plus agréable...

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

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

- **Créer une fenêtre :**
  - ✓ La création d'une fenêtre **commence** par l'appel à la fonction `DebutFenetre`.
  - ✓ Une fenêtre a un nom unique et est composée d'un ensemble d'éléments graphiques. Lors de sa création elle est dotée par défaut, d'un élément nommé "fond". Les éléments qui peuvent y être ajoutés sont de différents types :
    - bouton classique rectangulaire, bouton rond, bouton décoré d'une image (au format .xpm)
    - boîte à cocher, bouton radio
    - image
    - texte (non modifiable par l'utilisateur) avec ou sans ascenseur, champ de saisie
    - horloge analogique ou digitale, minuteur, chronomètre
  - ✓ L'ajout d'un élément se fait à l'aide de la procédure relative à son type (cf. [primitives d'ajout d'éléments graphiques](#)).
    - un élément doit avoir un nom unique pour sa fenêtre. : c'est grâce à ce nom qu'on peut le manipuler.
    - un élément a une apparence graphique par défaut (couleur de texte et de fond, style et taille de texte).
    - un élément a un comportement par défaut (par exemple, un bouton est actif par défaut).
    - on peut modifier l'apparence et le comportement d'un élément dès sa création (cf. plus bas)
    - un élément est défini une fois pour toutes : on ne peut pas le supprimer de sa fenêtre mais on peut le rendre temporairement visible ou invisible (cf. `Montrer/Cacher un élément`)
  - ✓ Des éléments d'une même fenêtre peuvent être groupés de façon à ce que l'activation de l'un d'entre eux désactive les éléments du même groupe. C'est particulièrement utile pour gérer des boutons radio ou des boîtes à cocher alternatives. Pour cela, il faut encadrer l'ajout des éléments à grouper par les appels des procédures `DebutGroupe` et `FinGroupe`.
  - ✓ La création d'une fenêtre **se termine** par un appel à la procédure `FinFenetre`.

**Attention** : il faut construire les fenêtres d'une application indépendamment les unes des autres.

- **Afficher / masquer une fenêtre :**  
Une fenêtre peut être affichée ou masquée à l'utilisateur (procédures `MontrerFenetre` / `CacherFenetre`).
- **Modifier le comportement d'un élément d'une fenêtre** (dès sa création ou au cours de l'exécution) :  
On peut activer/désactiver un bouton, cacher/montrer un élément, forcer l'activation d'un bouton radio ou boîte à cocher ou encore modifier le comportement d'un timer (cf. [primitives de modification du comportement d'éléments graphiques existants](#))
- **Modifier l'apparence d'un élément d'une fenêtre** (dès sa création ou au cours de l'exécution) :  
De nombreuses primitives permettent de modifier un ou plusieurs critères d'apparence d'un élément, conformément à son type (cf. [primitives de modification de l'apparence ou du contenu d'un élément](#)).
- **Exploiter les éléments d'une fenêtre par programme :**  
Chaque élément d'une fenêtre a sa raison d'être : purement décorative (l'élément doit être désactivé) ou fonctionnelle si le programmeur a développé du code exécutable si l'utilisateur clique dessus. (cf. [primitives d'accès à un élément graphique existant](#)). Parmi les primitives proposées, trois d'entre elles vous seront particulièrement utiles :
  - ✓ La fonction `AttendreBouton` bloque l'exécution jusqu'à ce que l'utilisateur ait cliqué sur un bouton actif de la fenêtre courante. Cette fonction a pour résultat le nom du bouton.
  - ✓ La fonction `ClickDroit` permet de savoir si l'utilisateur a utilisé le bouton droit de la souris pour cliquer sur un bouton actif (on peut ainsi effectuer un traitement différencié suivant la nature du click).
  - ✓ La fonction `ConsulterContenu` permet de récupérer ce qu'a tapé l'utilisateur dans un élément de type champ de saisie.

**Attention** : pour qu'une fenêtre reste affichée après l'appel de `MontrerFenetre`, le programme doit comporter au moins un appel à `AttendreBouton`

## A. Attendus

L'interface texte étant unique (une seule fenêtre), l'utilisateur était contraint de suivre la séquence d'actions prévue par le programmeur pour tenter de dégager le virus. Grâce à la possibilité de multifenêtrage dont vous disposez maintenant, le jeu peut prendre une forme nettement moins contraignante pour l'utilisateur et être facilement doté de fonctionnalités qui le rendront plus attrayant.

Libre à vous, d'étendre les fonctionnalités du jeu, tout en gardant ses principes initiaux (organisation et configuration de la grille de jeu, règles relatives au déplacement des pièces, gain d'une partie).

**Les modalités minimales qui vous sont demandées sont les suivantes :**

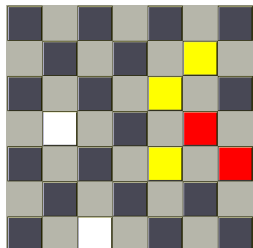
- ✓ Fenêtre d'accueil de l'utilisateur avec saisie d'un pseudo et choix de la partie qu'il veut jouer, avec possibilité de quitter immédiatement l'application ou de jouer la partie choisie.
- ✓ Fenêtre de jeu où la grille configurée sera affichée et mise à jour et dans laquelle l'utilisateur pourra à sa guise, déplacer une pièce, recommencer ou quitter la partie courante (s'il quitte une partie, l'utilisateur doit pouvoir en choisir une autre via la fenêtre d'accueil sans avoir à relancer l'application).
- ✓ Dans chaque fenêtre créée : présence d'éléments dédiés à l'information de l'utilisateur (exemple pour la fenêtre de jeu : nombre de coups qu'il a joués depuis le début d'une partie, nombre d'erreurs (tentatives de déplacements interdits), gain de la partie courante....)

Une fois ces fonctionnalités développées et bien testées, vous pourrez les enrichir (voir **B. Extensions**).

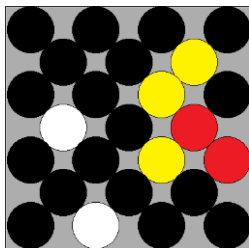
**Représentation de la grille de jeu :**

- ✓ La grille sera représentée par des boutons : un bouton par case utilisable, la couleur des boutons dépendant de s'ils sont ou non recouverts par une pièce. Au fil d'une partie, les boutons de la grille seront activés ou désactivés selon leur couleur du moment (couleur d'une pièce mobile ou d'une case vide).
- ✓ Vous pouvez opter pour le choix de boutons carrés (plus simples à mettre en place) ou de boutons ronds (plus jolis, mais moins simples à mettre en place et plus délicats à manipuler lors du jeu, la zone cliquable étant moins bien délimitée que celle des boutons carrés -défaut inhérent à la bibliothèque graphique utilisée).

Grille à boutons carrés



Grille à boutons ronds



**Charte de développement :**

- ✓ Vous devez impérativement faire appel aux fonctions ou procédures de `p_virus` pour le contrôle du jeu (chargement d'une partie, déplacement d'une pièce, gain de la partie).
- ✓ La définition des fenêtres du jeu, les fonctionnalités de mise à jour graphique de la grille ou de tout élément d'une fenêtre dépendant des actions de l'utilisateur seront développées dans le paquetage `p_vue_graph`.
- ✓ La gestion du jeu lui-même (enchaînements d'affichage ou masquage de fenêtres, prise en compte des actions de l'utilisateur) sera prise en charge par la procédure principale `av_graph`. Vous veillerez à ce que chaque action de l'utilisateur ait l'effet attendu et notamment à ce que l'utilisateur n'ait pas à cliquer plusieurs fois sur un bouton pour que le programme réagisse...). Et bien sûr, vous ferez tout ce qu'il faut pour que le jeu ne s'arrête pas brutalement !

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 IV**)...

Certaines de ses astuces peuvent vous être 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 !)

## Méthodologie conseillée :

### 1. Étudiez le programme `marketing` et penchez-vous sur :

- la spécification des procédures du paquetage `p_fenbase` auxquelles il fait appel
- la création de ses différentes fenêtres
- la création de la grille de boutons de la fenêtre `FJeu`
- sa structure (boucles et déclaration dynamique)
- etc.

### 2. Maquettez sur papier la fenêtre de jeu (celle qui affiche la grille), puis :

- dans le paquetage `p_vue_graph`, ajoutez la procédure de création de cette fenêtre
- dans un programme de test :
  - ✓ chargez la grille avec la configuration 1
  - ✓ affichez la fenêtre jusqu'à ce que l'utilisateur ait cliqué sur un bouton qui provoque sa fermeture
- corrigez ou complétez votre code jusqu'à ce que cette première étape donne les résultats attendus

**ATTENTION** : Pour compiler une procédure principale utilisant les paquetages `forms` et `p_fenbase`, utilisez la commande `build`.  
Avant la première exécution, puis à chaque nouvelle connexion, tapez dans votre terminal la commande :  
`export LD_LIBRARY_PATH=/users/info/pub/1a/ada/bib/lib`

### 3. Spécifiez le comportement de chaque élément de la fenêtre de jeu et complétez en conséquence le paquetage `p_vue_graph`, puis testez.

### 4. Procédez de façon similaire avec les autres fenêtres

### 5. Spécifiez les conditions d'enchaînement des différentes fenêtres

### 6. Développez (ou complétez) le programme `av_graph`.

## B. Extensions (suivant le temps que vous aurez...)

Vous pouvez doter votre jeu de nouvelles fonctionnalités, comme par exemple :

- ✓ Créer une fenêtre de règles du jeu, accessible à tout moment
  - ✓ Permettre l'annulation du dernier déplacement (ou de déplacements successifs)
  - ✓ Gérer et afficher le temps qui s'écoule entre le premier déplacement et le gain d'une partie
  - ✓ Gérer les scores effectués par les différents utilisateurs pour chaque partie qu'ils ont jouée et en corollaire :
    - afficher en début d'une partie, le meilleur score qu'a réalisé le joueur pour cette partie par le passé, et s'il fait mieux, mettre à jour son score en fin de partie
    - afficher en début de partie, le meilleur score obtenu pour cette partie par les joueurs qui l'ont pratiquée
- NOTE : les scores peuvent éventuellement intégrer le temps qu'a mis le joueur pour gagner une partie...*
- ✓ Proposer différents niveaux de jeu. Dans le fichier `Parties`, les configurations initiales sont ordonnées par ordre de difficulté croissante et on peut considérer qu'il y a 4 niveaux :
    - starter (parties 1 à 5)
    - junior (parties 6 à 10)
    - expert (parties 11 à 15)
    - wizard (parties 19 à 20)
  - ✓ Créer un fichier de solutions de certaines parties pour pouvoir les proposer à l'utilisateur
  - ✓ ... ou tout autre bonne idée que vous aurez envie d'implémenter !



# 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 TA_String is access String;          -- pointeur sur une chaîne de caractères
type TR_Fenetre;                          -- une fenêtre de l'interface graphique
type T_EtatBouton is (Marche, Arrêt);     -- état d'un bouton (actif ou inactif)
subtype T_Couleur is FL_COLOR;            -- couleurs disponibles dans le paquetage forms
-- FL_BLACK, FL_RED, FL_GREEN, FL_YELLOW, FL_BLUE, FL_MAGENTA, FL_CYAN,
-- FL_WHITE, 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_DARKGOLD,
-- FL_ORCHID, FL_DARKCYAN, FL_DARKTOMATO, FL_WHEAT, FL_DARKORANGE, FL_DEEPPINK,
-- FL_CHARTREUSE, FL_DARKVIOLET, FL_SPRINGGREEN, FL_DODGERBLUE
-----
-- types internes au paquetage, inutile de les connaître !
-----
type T_TypeElement is (Bouton, TexteFixe, ChampDeSaisie, TexteAscenseur, Horloge, Fond, CheckBox,
    PictBouton, BoutonRond, Image, Timer, BoutonRadio, Groupe );
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;

-----
-- Initialisation du mode graphique, définition fenêtres et groupes d'éléments
-----
-- initialiser le mode graphique
-----
procedure InitialiserFenetres;

-- définir une nouvelle fenêtre
-----
function DebutFenetre (
    Titre       : in String;          -- nom de la fenêtre
    Largeur,    : in Positive;        -- sa largeur en pixels
    Hauteur     : in Positive;        -- sa hauteur en pixels
    return TR_Fenetre;                -- résultat : la fenêtre créée

-- terminer la définition d'une nouvelle fenêtre
-----
procedure FinFenetre (
    F           : in TR_Fenetre );    -- nom de la fenêtre

-- définir un groupe d'éléments graphiques
-----
procedure DebutGroupe (
    F           : in out TR_Fenetre;  -- la fenêtre du groupe
    NomGroupe   : in String );        -- le nom du groupe

-- terminer un groupe d'éléments
-----
procedure FinGroupe;                  -- clôt le groupe précédemment défini

-- primitives d'ajout d'éléments graphiques dans une fenêtre déjà définie
-----
-- ajouter un bouton de forme rectangulaire
-----
procedure AjouterBouton (
    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,          : in Natural;         -- abscisse coin HG en pixels
    Y           : in Natural;         -- ordonnée coin HG en pixels
    Largeur,    : in Positive;        -- largeur du bouton en pixels
    Hauteur     : in Positive;        -- hauteur du bouton en pixels
    TypeBouton  : in FL_Button_Type := FL_NORMAL_BUTTON); -- type du bouton (optionnel)
-- types de boutons : cf documentation XForms
-- FL_NORMAL_BUTTON (retourne une valeur quand relâché)
-- FL_PUSH_BUTTON (retourne une valeur si relâché, reste enfoncé jusqu'à nouveau clic)
-- FL_MENU_BUTTON (retourne une valeur quand pressé)
-- FL_TOUCH_BUTTON (retourne une valeur tant que l'utilisateur le presse)
-- FL_RETURN_BUTTON (bouton classique mais qui réagit à la touche ENTER du clavier)
-- ajouter un bouton de forme circulaire
-----
procedure AjouterBoutonRond (
    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           : in Natural;         -- abscisse coin HG du carré englobant en pixels
    Y           : in Natural;         -- ordonnée coin HG du carré englobant en pixels
    Diam        : in Positive );      -- diamètre du bouton en pixels
```



```

-- ajouter une boîte à cocher -----
procedure AjouterBoiteCocher (
    F          : in out TR_Fenetre;    -- la fenêtre où on ajoute
    NomElement : in    String;          -- le nom de la boîte à cocher (unique)
    Texte      : in    String;          -- le texte affiché en légende
    X,          -- abscisse coin HG rectangle englobant en pixels
    Y          : in    Natural;          -- ordonnée coin HG rectangle englobant en pixels
    Largeur,   -- sa largeur en pixels
    Hauteur    : in    Positive );      -- sa hauteur en pixels

-- ajouter un bouton radio -----
procedure AjouterBoutonRadio (
    F          : in out TR_Fenetre;    -- la fenêtre où on ajoute
    NomElement : in    String;          -- le nom du bouton radio (unique)
    Texte      : in    String;          -- le texte affiché en légende
    X,          -- abscisse coin HG du carré englobant en pixels
    Y          : in    Natural;          -- ordonnée coin HG du carré englobant en pixels
    Diam       : in    Positive;        -- son diamètre en pixels
    Actif      : in    Boolean := false; -- inactivé par défaut
    -- NOTE : un bouton radion est intéressant dans un groupe d'autres boutons radio

-- ajouter un bouton décoré d'une image (format XPM) -----
procedure AjouterBoutonImage (
    F          : in out TR_Fenetre;    -- la fenêtre où on ajoute
    NomElement : in    String;          -- le nom du bouton (unique)
    Texte      : in    String;          -- le texte affiché en légende
    NomImage   : in    String;          -- le nom de l'image décor
    X,          -- abscisse du bouton en pixels
    Y          : in    Natural;          -- ordonnée du bouton en pixels
    Largeur,   -- largeur du bouton en pixels
    Hauteur    : in    Positive );      -- hauteur du bouton en pixels

-- ajouter une zone de texte non modifiable -----
procedure AjouterTexte (
    F          : in out TR_Fenetre;    -- la fenêtre où on ajoute
    NomElement : in    String;          -- le nom de la zone de texte
    Texte      : in    String;          -- son contenu
    X,          -- abscisse du coin HG de la zone de texte
    Y          : in    Natural;          -- ordonnée du coin HG de la zone de texte
    Largeur,   -- largeur de la zone de texte
    Hauteur    : in    Positive );      -- hauteur de la zone de texte

-- ajouter un champ de saisie -----
procedure AjouterChamp (
    F          : in out TR_Fenetre;    -- la fenêtre où on ajoute
    NomElement : in    String;          -- le nom du champ
    Texte      : in    String;          -- le texte affiché en légende
    Contenu    : in    String;          -- le contenu initial du champ
    X,          -- abscisse du coin HG du champ
    Y          : in    Natural;          -- ordonnée du coin HG du champ
    Largeur,   -- largeur du champ
    Hauteur    : in    Positive );      -- hauteur du champ

-- ajouter une zone de texte avec ascenseur -----
procedure AjouterTexteAscenseur (
    F          : in out TR_Fenetre;    -- la fenêtre où on ajoute
    NomElement : in    String;          -- le nom de la zone de texte
    Texte      : in    String;          -- le texte affiché en légende
    Contenu    : in    String;          -- le contenu de la zone
    X,          -- abscisse du coin HG de la zone
    Y          : in    Natural;          -- abscisse du coin HG de la zone
    Largeur,   -- largeur de la zone de texte
    Hauteur    : in    Positive );      -- hauteur de la zone de texte

-- ajouter une horloge analogique -----
procedure AjouterHorlogeAna (
    F          : in out TR_Fenetre;    -- la fenêtre où on joute
    NomElement : in    String;          -- le nom de l'horloge
    Texte      : in    String;          -- le texte affiché en légende
    X,          -- abscisse du coin HG
    Y          : in    Natural;          -- ordonnée du coin HG
    Largeur,   -- largeur
    Hauteur    : in    Positive );      -- hauteur

-- ajouter une horloge digitale -----
procedure AjouterHorlogeDigi (
    F          : in out TR_Fenetre;    -- la fenêtre où on joute
    NomElement : in    String;          -- le nom de l'horloge
    Texte      : in    String;          -- le texte affiché en légende
    X,          -- abscisse du coin HG
    Y          : in    Natural;          -- ordonnée du coin HG
    Largeur,   -- largeur
    Hauteur    : in    Positive );      -- hauteur

```

```

-- ajouter un minuteur -----
procedure AjouterMinuteur (
    F          : in out TR_Fenetre;      -- la fenêtre où on ajoute
    NomElement : in      String;          -- le nom du minuteur
    Texte      : in      String;          -- le texte affiché en légende
    X,         : in      Natural;         -- abscisse du coin HG
    Y          : in      Natural;         -- ordonnée du coin HG
    Largeur,   : in      Positive );      -- largeur
    Hauteur    : in      Positive );      -- hauteur

-- ajouter un chronomètre -----
procedure AjouterChrono (
    F          : in out TR_Fenetre;      -- la fenêtre où on ajoute
    NomElement : in      String;          -- le nom du chronomètre
    Texte      : in      String;          -- le texte affiché en légende
    X,         : in      Natural;         -- abscisse du coin HG
    Y          : in      Natural;         -- ordonnée du coin HG
    Largeur,   : in      Positive );      -- largeur
    Hauteur    : in      Positive );      -- hauteur

-- ajouter une image -----
procedure AjouterImage (
    F          : in out TR_Fenetre;      -- la fenêtre où on ajoute
    NomElement : in      String;          -- le nom de l'élément
    NomImage   : in      String;          -- le nom de l'image
    Text       : in      String;          -- le texte affiché en légende, en bas de l'image
    X,         : in      Natural;         -- son abscisse en pixels
    Y          : in      Natural;         -- son ordonnée en pixels
    Largeur,   : in      Positive );      -- largeur
    Hauteur    : in      Positive );      -- hauteur

-----
-- primitives d'accès à un élément graphique existant
-----
-- récupérer le nom du bouton cliqué -----
function AttendreBouton (
    F          : in      TR_Fenetre )      -- nom de la fenêtre
return String;                             -- Résultat = nom du bouton pressé

-- récupérer le contenu d'un champ de saisie -----
function ConsulterContenu (
    F          : in      TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String )          -- nom du champ de saisie
return String;                             -- Résultat : la chaîne saisie !

-- récupérer le temps résiduel d'un minuteur -----
function ConsulterTimer (
    F          : in      TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String )          -- nom du minuteur
return Float;                             -- Résultat : le temps restant !

-- récupérer l'état d'une boîte à cocher ou d'un bouton radio -----
function ConsulterEtatBCRB (
    F          : in      TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String )          -- nom de l'élément
return boolean;                           -- Résultat : true si coché, false sinon

-----
-- primitives de modification du comportement d'éléments graphiques existants
-----
-- rendre visible une fenêtre -----
procedure MontrerFenetre (
    F          : in      TR_Fenetre );      -- nom de la fenêtre

-- masquer une fenêtre -----
procedure CacherFenetre (
    F          : in      TR_Fenetre );      -- nom de la fenêtre

-- rendre visible un élément graphique caché jusque là -----
procedure MontrerElem (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String );          -- nom de l'élément

-- rendre invisible un élément graphique jusque-là visible -----
procedure CacherElem (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String );          -- nom de l'élément

-- activer ou désactiver un bouton -----
procedure ChangerEtatBouton (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom du bouton
    Etat       : in      T_EtatBouton );    -- valeur : marche (cliquable) ou arrêt (désactivé)

-- forcer l'état d'un bouton radio -----
procedure ChangerEtatBoutonRadio (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom du bouton
    Etat       : in      Boolean );          -- valeur : true (coché) ou false (non coché)

```

```

-- forcer l'état d'une boîte à cocher -----
procedure ChangerEtatBoiteCocher (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de la boîte à cocher
    Etat       : in      Boolean          ); -- valeur : true (coché) ou false (non coché)

-- passer d'un mode minuteur à un mode chronomètre -----
-- le temps s'incrémente au lieu de se décrémenter
procedure ChangerMinuteurEnChrono (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String          ); -- nom de l'élément

-- passer d'un mode chronomètre à un mode minuteur -----
-- le temps se décrémente au lieu de s'incrémenter
procedure ChangerChronoEnMinuteur(
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String          ); -- nom de l'élément

-- mettre en pause en timer en marche -----
procedure PauseTimer (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String          ); -- nom de l'élément

-- mettre en marche en timer en pause -----
procedure RepriseTimer (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String          ); -- nom de l'élément

-----
-- primitives de modification de l'apparence ou du contenu d'un élément
-----
-- modifier la couleur du fond d'un élément graphique -----
procedure ChangerCouleurFond (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    NouvelleCouleur : in      T_Couleur  ); -- La nouvelle couleur !

-- modifier la couleur du label d'un élément ou d'un texte fixe -----
procedure ChangerCouleurTexte (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    NouvelleCouleur : in      T_Couleur  ); -- La nouvelle couleur !

-- modifier le style du label d'un élément ou d'un texte fixe -----
procedure ChangerStyleTexte (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    NouveauStyle : in      FL_TEXT_STYLE  ); -- Le nouveau style !

-- modifier la taille du label d'un élément ou d'un texte fixe -----
procedure ChangerTailleTexte (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    Taille     : in      X11.Int         ); -- La nouvelle taille !

-- modifier l'alignement du label d'un élément ou d'un texte fixe -----
procedure ChangerAlignementTexte (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    Alignement  : in      FL_ALIGN        ); -- Le nouvel alignement !

-- modifier le label d'un élément graphique ou la valeur d'un texte fixe-----
procedure ChangerTexte (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- Le nom de l'élément à modifier
    NouveauTexte : in      String         ); -- La nouvelle valeur de légende

-- modifier le style du contenu d'une zone de texte ascenseur -----
procedure ChangerStyleContenu (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de la zone de texte
    NouveauStyle : in      FL_TEXT_STYLE  ); -- Le nouveau style !

-- modifier la taille du contenu d'une zone de texte ascenseur -----
procedure ChangerTailleContenu (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    Taille     : in      X11.Int         ); -- La nouvelle taille !

-- modifier le contenu d'un champ de saisie ou d'une zone de texte ascenseur---
procedure ChangerContenu (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String;          -- nom de l'élément
    NouveauContenu : in      String       ); -- La nouvelle valeur de contenu

-- effacer le contenu d'un champ de saisie ou d'une zone de texte ascenseur---
procedure EffacerContenu (
    F          : in out TR_Fenetre;      -- nom de la fenêtre
    NomElement : in      String          ); -- nom de l'élément

```

```

-- ajouter une nouvelle ligne dans une zone de texte avec ascenseur-----
procedure AjouterNouvelleLigne (
    F          : in out TR_Fenetre;    -- nom de la fenêtre
    NomElement  : in    String;         -- nom de l'élément
    NouvelleLigne : in    String       ); -- Le contenu de la nouvelle ligne

-- modifier une ligne dans une zone de texte avec ascenseur-----
procedure ChangerLigne (
    F          : in out TR_Fenetre;    -- nom de la fenêtre
    NomElement  : in    String;         -- nom de l'élément
    NumeroLigne  : in    Natural;       -- numéro de ligne à modifier
    NouveauTexte : in    String       ); -- Le nouveau contenu de la ligne

-- supprimer une ligne dans une zone de texte avec ascenseur-----
procedure SupprimerLigne (
    F          : in out TR_Fenetre;    -- nom de la fenêtre
    NomElement  : in    String;         -- nom de l'élément
    NumeroLigne  : in    Natural       ); -- Le numéro de ligne à supprimer

-- changer la durée du temps résiduel d'un minuteur -----
procedure ChangerTempsMinuteur (
    F          : in out TR_Fenetre;    -- nom de la fenêtre
    NomElement  : in    String;         -- nom du minuteur
    NouveauTemps : in    Float         ); -- La durée du décompte !


-- changer l'image d'un bouton image -----
procedure ChangerImageBouton (
    F          : in out TR_Fenetre;    -- nom de la fenêtre
    NomElement  : in    String;         -- nom de l'élément
    NomImage     : in    String       ); -- nom de la nouvelle image décor

-- changer l'image d'un bouton image -----
procedure ChangerImage (
    F          : in out TR_Fenetre;    -- nom de la fenêtre
    NomElement  : in    String;         -- nom de l'élément
    NomImage     : in    String       ); -- nom de la nouvelle image

-----
-- Autre primitive : tester l'utilisation du click droit de la souris
-----
function ClickDroit return boolean; -- vrai si on a pressé le bouton droit de la souris
-----
end p_fenbase;

```

## Annexe II – Couleurs disponibles dans le paquetage Forms

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

## Annexe III : 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) -- <i>valeur par défaut</i> FL_MEDIUM_SIZE (14 pt) FL_LARGE_SIZE (18 pt) FL_HUGE_SIZE (24 pt)	FL_NORMAL_STYLE -- <i>valeur par défaut</i> 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 (↔ Times et déclinable en BOLD, ITALIC, BOLDITALIC) FL_SHADOW_STYLE (ombré) / EMBOSSED (relief) / ENGRAVED (gravé) <i>Remarque : certains choix de style peuvent rester sans effet (Pb. implémentation)</i>

## Annexe IV – Marketing.adb

```
with p_fenbase ; use p_fenbase ;
with Forms ; use Forms;
with ada.calendar; use ada.calendar;

procedure marketing is
  FSaisieNom, FJeu, FResultat : TR_Fenetre; -- l'application comporte 3 fenêtres
  Compteur : Natural;
  Touche : Character;
  I, J : Natural;
  Nombouton : String(1..2);
  HeureDeb,HeureFin : Time;
  NewLine : constant Character := Character'Val (10); -- retour chariot
begin
  -- initialisation de l'interface graphique (à ne faire qu'une fois en début de programme)
  InitialiserFenetres;

  -- création de la fenêtre de saisie du nom du "gogo"
  FSaisieNom:=DebutFenetre("Nom du Joueur",400,90);
  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);

  -- création de la fenêtre pour "jouer"
  FJeu:=DebutFenetre("CHANCE",300,450);
  AjouterTexte(FJeu,"message1","TAPER SUR 3 TOUCHES...",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 "ij" 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);
      -- modification de l'apparence du bouton (couleur de fond, taille et style du texte)
      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);
  AjouterHorlogeDigi(FJeu,"Clock","",150,350,100,70);
  ChangerStyleTexte(FJeu,"Clock", FL_BOLD_Style);
  ChangerTailleTexte(FJeu,"Clock", FL_medium_size);
  ChangerCouleurFond(FJeu,"Clock",FL_WHITE);
  AjouterBouton(FJeu,"BoutonAbandonner","abandon",55,350,70,30);
  FinFenetre(FJeu);

  -- création de la fenêtre d'affichage du "gain"
  FResultat:=Debutfenetre("Votre gain ! ",300,200);
  AjouterTexteAscenseur(FResultat,"message","", "",10,10,280,100);
  AjouterBouton(FResultat,"BoutonFin","The End",115,140,70,30);
  FinFenetre(FResultat);

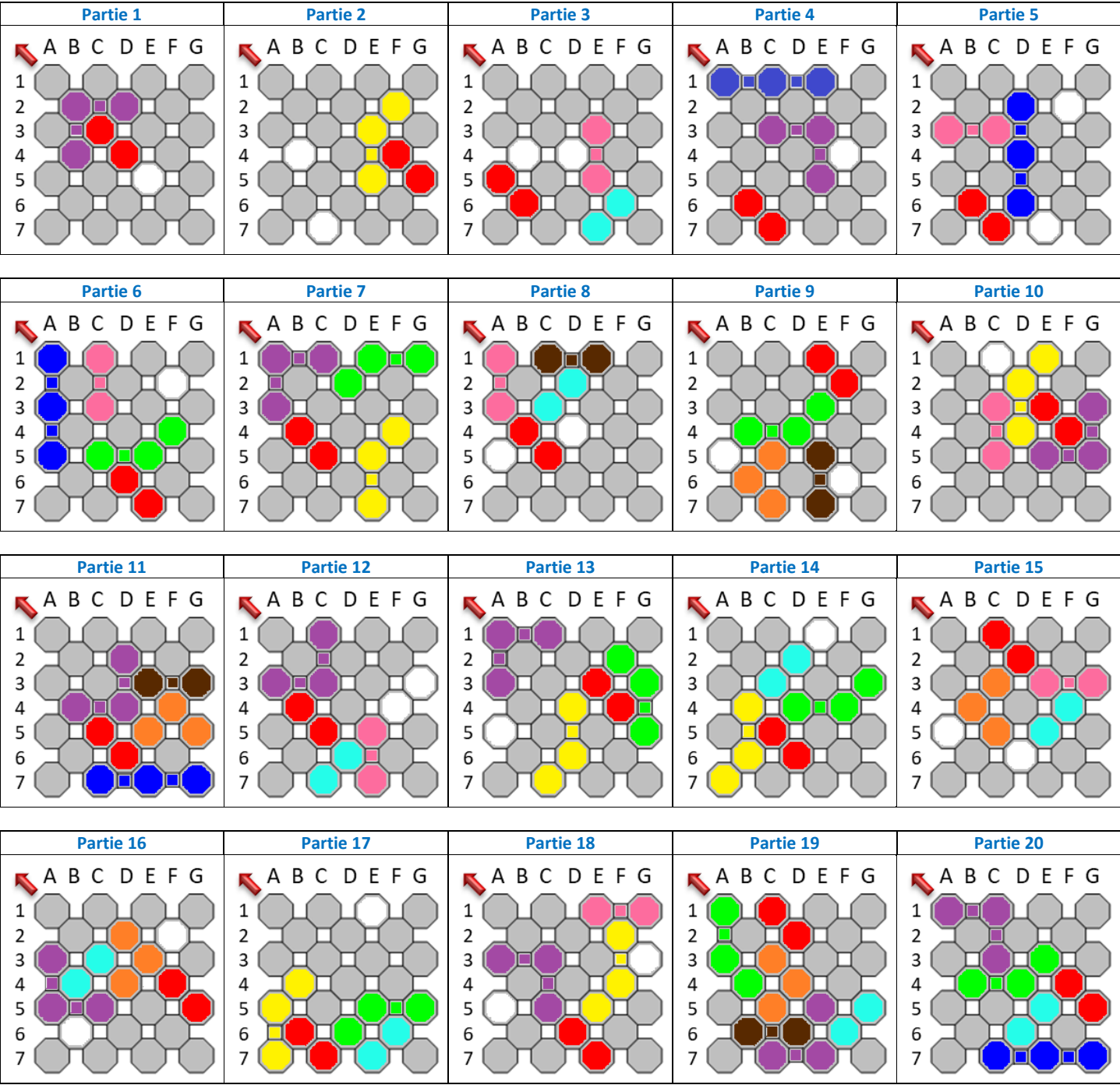
  -----
  -- Affichage de la fenêtre de saisie et attente d'un bouton pressé
  MontrerFenetre(FSaisieNom);
  if AttendreBouton(FSaisieNom)/="BoutonAnnuler" then
    CacherFenetre(FSaisieNom);
    Compteur:=0; -- nombre de touches pressées
    -- modification du texte "message2" de la fenêtre de jeu :
    -- concaténation du contenu du champ de saisie du nom avec " tente ta chance !"
    ChangerTexte(FJeu,"message2", ConsulterContenu(FSaisieNom,"ChampNom")& " tente ta chance !");
    -- Affichage de la fenêtre de jeu
    MontrerFenetre(FJeu);
    HeureDeb:=clock;
    -- Pour que le "gogo" soit obligé de cliquer sur 3 touches, on rend le bouton "Fin" inactif
    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 -- 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_DEEPPINK); -- et mis en rose
            ChangerTailleTexte(FJeu,Bouton,FL_SMALL_SIZE); -- pour que le texte rentre...
            ChangerTexte(FJeu, Bouton, "GAGNE"); -- et c'est l'arnaque !!!
          end if;
        end if;
      end begin;
    end loop;
  end if;
end marketing;
```

```

-- 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 caractère
Touche:= Character'Val(((I-1)*3+J)+Character'Pos('0'));
-- on change l'objet texte "BarreEtat" en fonction de la touche pressée
ChangerTexte(FJeu,"BarreEtat","La touche " & Touche & " a ete pressee");
if compteur = 3 then -- C'est gagné !!!
    -- on désactive toutes les touches (sans faire de détail !...)
    for I in 1..3 loop
        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;
    -- on calcule le temps qu'a mis le gogo pour "gagner"
    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);
    -- réactivation du bouton "Fin"
    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 : message "commercial"
    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, autre message "commercial"
    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;
-- Affichage de la fenêtre des gains
Montrerfenetre(Fresultat);
loop
    exit when Attendrebouton(Fresultat) = "BoutonFin";
end loop;
end if;
end marketing;

```

Annexe V – Configurations du jeu Antivirus



Légende des couleurs

rouge	
turquoise	
orange	
rose	
marron	
bleu	
violet	
vert	
jaune	
blanc	