Jeu de la vie

Le jeu de la vie n'est pas vraiment un jeu Il représente l'évolution d'une population de cellules, qui naissent, vivent, et meurent selon des règles établies

A consulter (éventuellement)

- https://fr.wikipedia.org/wiki/Jeu_de_la_vie
- https://youtu.be/S-W0NX97DB0 (traite aussi de sujets plus complexes)

Les règles représentent un algorithme systolique (qui est traité par pulsions/vagues, fait référence à la contraction du muscle cardiaque et/ou flux sanguin), soit par la génération de populations successives.

Il existe plusieurs variantes, la version la plus simple est nomenclaturée sous la norme VIE 2333, soit une cellule survit avec 2 ou 3 voisines et meurent avec plus de 3 voisines Une cellule nait dans une case qui à 3 cellules voisines

Il existe des variantes VIE 4555 ou VIE 5766 ou chaque cellule possède 26 voisines (3D)

Dans VIE 2333, chaque case possède 8 voisines. Une case n'est pas sa voisine

Règles pour VIE 2333

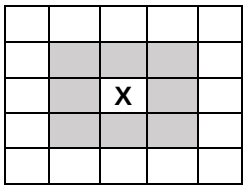
R1 : Une cellule meurt d'ennui (isolement) si elle à moins de 2 voisines

R2 : Une cellule meurt étouffée, sil elle a plus de 3 voisines

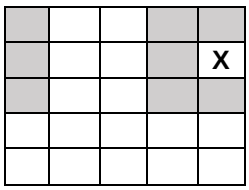
R3 : Une cellule nait dans une case vide ayant 3 cellules voisines

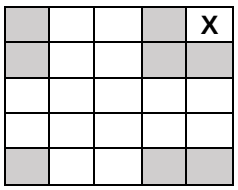
R4 : Les autres situations ne changent pas la population

Les voisines de la cellule X sont grisées



Suppression des effets de bords





La grille représente la « projection » d'une sphère (planète), une cellule a donc toujours 8 cases voisines

Le jeu consiste à saisir une population de départ et à observer son comportement par générations successives

Quelques exemples de générations :

Population initiale

	M	**V**	**M**	

Population générée

		N		
		V		
		N		

M : Meurt (isolement) à la génération suivante

V : Reste en vie à la génération suivante

N : Nait à la génération suivante

Population initiale

	V			
	V	**E**	**V**	
			V	

Population générée

	V			
	V		**V**	
			V	

E : Meurt (étouffement) à la génération suivante

V : Reste en vie à la génération suivante

Population initiale

	V	**V**		
	V	**V**		

Population générée

	V	**V**		
	V	**V**		

Population initiale

	M	**V**	**V**	
			V	
		M		

Population générée

		N		
		V	**V**	
	N		**V**	

Population initiale

		V		
		E	**E**	
	M		**V**	

Population générée

		V	**N**	**N**
				N
			V	

C'est un planeur, soit une population qui se « déplace » dans la grille

Objectif

Ecrire un programme qui permet de visualiser des générations de populations de cellules.

Le programme sera organisé comme suit (proposition de fonctionnement) :

Etape 1 *(Action a choix)*

1 - Charger une population à partir d'un fichier

2 - Saisir manuellement une population

Etape 2 *(Action à choix, sur le choix 1, tourne en affichant 2 populations)*

1 - Générer la population suivante

2 - Stopper la génération et sauver la population

3 - Stopper la génération sans sauver

Structure de données

Le type (proposé) `Tpopulation` permettant de stocker une population est un tableau a 2 dimensions de booléen (`True` si il y a une cellule, `False` si il n'y en a pas)

Génération d'une population

Pour chaque calcul d'une nouvelle génération, il y a deux populations (ancienne, nouvelle) concernées

Rappel des règles de génération

Table de décision

		R1	R2	R3	R4	R5	R6
C1	Est une cellule	O	O	O	O	N	N
C2	Nb de voisines	<2	>3	=3	=2	<>3	=3
A1	Cellule meurt	X	X				
A2	Cellule nait						X
A3	Reste identique			X	X	X	

Exemple de signature :

```
nouvelleGeneration( ancienne : Tpopulation(E) ):Tpopulation
```

Cette fonction fera appel à une autre fonction qui détermine le nombre de cellules voisines vivantes d'une case (*ligne,colonne*) dans une population.

```
nbVoisines( ligne      : Entier (E),
            colonne    : Entier (E),
            population : Tpopulation (E) ) : Entier
```

Pour déterminer le nombre de voisines en supprimant les effets de bords, on peut utiliser une fonction qui corrige la position (ligne ou colonne)

```
corrige( posi      : Entier (E),
         max       : Entier (E) ) : : Entier
```

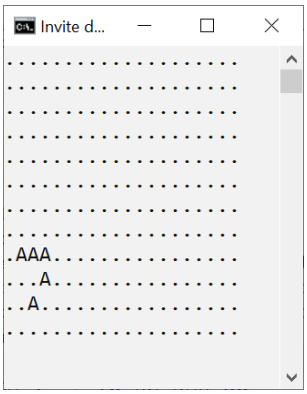
Si `posi` est plus grand que `max`, elle retourne `1` et si `posi = 0`, elle retourne `max`

Affichage de populations

L'affichage, sans environnement graphique, est simple (mais moche) Il peut être réalisé avec des `Write` et des `Writeln` La représentation d'une cellule peut être fait avec un caractère, l'absence de cellule par un autre.

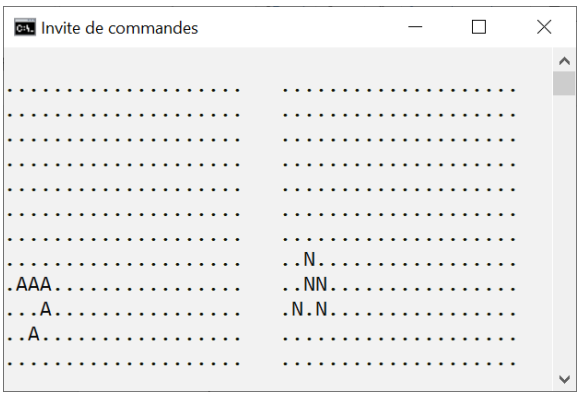
```
Affiche1Generation( population : Tpopulation  (E),
                    carVivant : caractere (E),
                    carVide   : caractere (E) )
```

Exemple d'affichage (pour un carré de 20 par 20) avec les valeurs `carVivant = 'A', carVide = '.'`



Cette affichage est ici à titre d'exemple car il est plus intéressant de disposer d'un service qui affiche 2 générations (ancienne/nouvelle) cote à cote.

```
affiche2Generations( ancPopulation : Tpopulation (E),
                     carAncien     : Caractere (E),
                     nouvPopulation: Tpopulation (E),
                     carNouveau    : Caractere (E),
                     carVide       : Caractere (E))
```



Remarque : L'affichage cellule par cellule avec des `Write` et des `Writeln` est possible, mais il est très lent (20x20x2 = 800 `Write`).

Si vous construisez une ligne complète (`String`) en mémoire en concaténant les cellules et vous l'afficher ensuite (20 `Writeln`), l'affichage sera beaucoup plus rapide

Sérialisation (sauvegarde) d'une population

Certaines populations sont susceptibles d'être sauvegardées, respectivement chargées

La structure des fichiers est à choix, par exemple (ligne et colonne, séparé par une virgule) :

PLANEUR.TXT

```
17,2
17,3
17,4
18,4
19,3
```

Exemple de signatures pour la sérialisation/chargement

```
serializePopulation (population : Tpopulation (E)) ;

chargePopulation    (population : Tpopulation (E/S)) ;
```