École de technologie supérieure
Hugues Saulnier, Anis Boubaker, Éric Thé

INF155 TP2 Automne 2018

Le jeu de Nim

1. Objectifs :

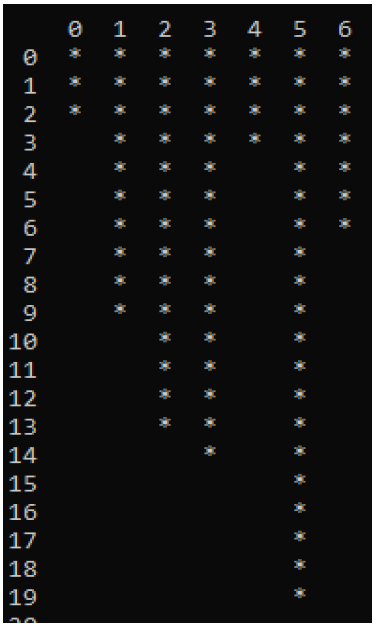
- Renforcer la maitrise des notions de base ;
- Mettre en application les notions relatives aux pointeurs, aux tableaux à une et plusieurs dimensions et aux modules ;
- Mettre en pratique les approches de tests unitaires ;
- Respecter les exigences de programmation : l'indentation, l'utilisation de constantes, l'ajout de commentaires, etc.

2. Description du problème : Le jeu de Nim

Le jeu de Nim est un jeu très célèbre et probablement un des plus anciens jeux du monde[1]. Il en existe plusieurs variantes et nous allons mettre en œuvre une de ces variantes dans un programme où un joueur humain doit s'opposer à un ordinateur.

Les règles du jeu sont très simples : nous disposons un ensemble de pièces sur plusieurs colonnes. Le nombre de pièce et leur disposition en colonne se fait aléatoirement. Dans notre cas, nous nous limiterons à un maximum de 20 colonnes et un maximum de 35 pièces par colonne.

Le joueur commence par décider combien de colonnes il souhaite avoir dans le plateau du jeu. Un ensemble de pièces par colonne est alors déterminé aléatoirement. L'exemple, ci-contre, présente une configuration du jeu à 7 colonnes.



Ensuite le joueur et l'ordinateur jouent tour à tour (le programme détermine aléatoirement qui commence), chacun choisissant une colonne et le nombre de pièce à en retirer (minimum une pièce, maximum toutes les pièces de la colonne). Le jeu se poursuit jusqu'à ce que la dernière pièce soit retirée, et le joueur l'ayant retirée sera déclaré gagnant. L'exécutable du programme attendu vous est fourni sur Moodle.

Afin de simplifier le projet, nous le réaliserons en deux parties :

a) Le programme permettant de jouer contre un ordinateur qui choisit ses coups de façon aléatoire;

b) Compléter la partie (a) en implémentant un algorithme de jeu qui mène l'ordinateur systématiquement à la victoire. Cet algorithme vous est détaillé plus loin.

Le projet doit être réalisé selon la conception qui vous est décrite dans la suite de cet énoncé, et vous devez respecter la subdivision en modules et sous-programmes tels que décrits.

[1] Jorgensen, Anker Helms (2009), "Context and driving forces in the development of the early computer game Nimbi", IEEE Annals of the History of Computing, 31 (3): 44–53

3. Conception du programme

Votre projet doit respecter la conception modulaire décrite ci-dessous. Certains modules vous sont fournis, les autres devront être implémentés par vous.

Le programme comprendra les sept modules suivants :

a) **nim** : C'est le « cerveau » du programme. Inclut les fonctions nécessaires à la gestion du plateau de jeu en mémoire et les fonctions de calcul de la stratégie de l'ordinateur. Aucun affichage n'est réalisé dans les fonctions de ce module. Tous les noms des fonctions de ce module sont préfixés par « **nim_** »

b) **nim_test** : Ce module comprend les fonctions de test du modules **nim**. Il doit inclure une fonction de test pour chaque fonction du module ***nim***, ainsi qu'une fonction qui appelle toutes les fonctions de test du module. Les fonctions de ce module doivent être préfixées par « **nim_test_** »

c) **nim_io (NIM Input/Output)** : Ce module inclut les fonctions assurant l'interactivité du jeu avec l'usager. Il comprend les fonctions permettant de gérer le jeu (alterner entre les joueurs), de demander à l'usager de saisir les données nécessaires, et de mettre à jour l'état du plateau du jeu au fur et à mesure. Les fonctions de ce module feront souvent appel aux fonctions du module « nim » ainsi qu'aux fonctions du module « nim_ihm » décrit ci-dessous. Les noms des fonctions de ce module ne sont pas préfixés.

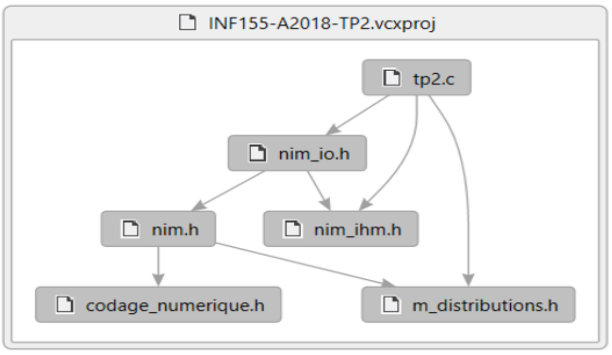
d) **nim_ihm (NIM Interface-Homme-Machine)** : Ce module vous est fourni. Il comprend diverses fonctions permettant l'affichage et la lecture de données dans la fenêtre du jeu. On y retrouve une fonction qui initialise la fenêtre du programme en la subdivisant en 3 zones, une fonctions qui permettent d'afficher une pièce sur une colonne, une fonction qui permet d'afficher du texte (comme ***printf***), ou de saisir du texte (comme ***scanf***), etc. L'utilisation de ce module vous est présentée dans la prochaine section. Tous les noms des fonctions de ce module sont préfixés par « **ihm_** ».

e) **m_distributions :** Ce module fourni est le même module que vous avez utilisé dans le cadre de votre TP1. Tous les noms des fonctions de ce module sont préfixés par « **md_** ».

f) **codage_numerique :** Dans ce module, vous développerez les fonctions permettant d'effectuer des conversions de base (décimal ←→ binaire). Ces fonctions seront utiles au module « nim », particulièrement les fonctions qui calculent la stratégie de l'ordinateur. Tous les noms des fonctions de ce module sont préfixés par « **codage_** ».

g) **codage_numerique_test :** Module qui regroupe toutes les fonctions de test du module ***codage_numerique***. Tous les noms de fonctions doivent être préfixés par « **codage_test_** ».

Le graphique, ci-contre, illustre les relations entre les modules. Une flèche allant du module A vers le module B veut dire que le module A inclut le module B (ou encore : le module A requiert le module B pour fonctionner). Tel qu'expliqué plus haut, on voir que le module nim_io dépend à la fois du module « nim_ihm » que du module « nim ».



Les fonctions qui composent chacun des modules vous seront décrites dans ce qui suit. Nous commençons par comprendre comment utiliser le module **nim_ihm**.

4. Utilisation du module « nim_ihm »

Le module **fourni** « nim_ihm » regroupe un ensemble de fonctions permettant de construire et d'interagir facilement avec l'interface utilisateur de Nim. Une description détaillée de chaque fonction est fournie en commentaire dans le fichier *nim_ihm.h*. Nous fournissons ici une description succincte de chaque fonction, ainsi qu'un exemple simple d'utilisation du module.

int ihm_init_ecran(int nb_lignes, int nb_colonnes);
Initialise l'affichage du jeu de NIM en subdivisant l'écran en trois zones : le titre, le plateau de jeu et la console. Les paramètres permettent de définir le nombre de ligne et de colonnes à afficher sur le plateau de jeu.
int ihm_changer_taille_plateau(int nb_lignes, int nb_colonnes);
Change le nombre de colonnes et de lignes du plateau de jeu de NIM après l'avoir initialisé.
int ihm_ajouter_piece(int ligne, int col);
Ajoute une pièce sur le plateau de jeu à la position ayant pour coordonnée (ligne, col)
void ihm_printf(...);
Cette fonction s'utilise exactement comme un ***printf***. Elle permet d'afficher du texte dans la zone console du jeu.
void ihm_scanf(...);
Cette fonction s'utilise exactement comme un ***scanf***. Elle permet de lire une information saisie au clavier.
void ihm_pause(void);
Fonction qui interrompe l'exécution du programme jusqu'à ce que l'utilisateur presse sur une touche.
void ihm_effacer_ecran();
Efface la zone console de l'écran.
int ihm_choisir_colonne(void);
Fonction qui demande à l'usager de choisir une des colonnes du plateau de jeu de façon interactive (en utilisant les touches flêchées) et qui retourne l'indice de la colonne choisie.

Voici un exemple de programme utilisant le module ***nim_ihm***. Il vous est conseillé de l'essayer avant de commencer à travailler sur le TP pour bien saisir le fonctionnement du module.

```
#include <stdlib.h>
#include "nim_ihm.h"

int main(void)
{
  int nb_lignes, //Nombre de lignes du plateau
     nb_colonnes, //Nombre de colonnes du plateau
     choix_colonne; //Colonne choisie par l'usager

  //Itinitalise l'écran du jeu. Dimensions du plateau encore inconnues (0,0).
  if (!ihm_init_ecran(0, 0)) {
     system("pause");
     return EXIT_FAILURE;
  }

  ihm_printf("Combien de colonnes ? ");
  ihm_scanf("%d", &nb_colonnes);
  ihm_printf("Combien de lignes ? ");
  ihm_scanf("%d", &nb_lignes);

  ihm_printf("Je redimentionne a %d lignes et %d cols.\n",
             nb_lignes, nb_colonnes);
  ihm_printf("Pressez une touche.\n\n");
  ihm_pause();
  ihm_changer_taille_plateau(nb_lignes, nb_colonnes);

  ihm_printf("Pressez une touche pour que j'efface l'ecran.\n\n");
  ihm_pause();
  ihm_effacer_ecran();
```

```
  ihm_printf("Ecran efface. Maintenant choissez une colonne. \n\n");
  choix_colonne = ihm_choisir_colonne();

  ihm_printf("Vous avez choisi %d, j'y ajoute 2 pieces.\n", choix_colonne);
  ihm_ajouter_piece(0, choix_colonne);
  ihm_ajouter_piece(1, choix_colonne);

  ihm_pause();
  return EXIT_SUCCESS;
}
```

5. Étapes de réalisation

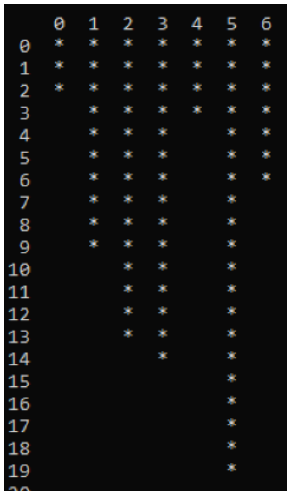
Afin de vous aider à mieux répartir votre travail sur le temps dont vous disposez, il vous est suggéré d'effectuer votre travail en deux étapes. Le travail réalisé lors de la première étape vous servira intégralement pour la deuxième étape.

5.1 Partie 1 : Un ordinateur pas si intelligent

L'objectif de cette partie est d'implémenter un jeu de Nim entre un joueur humain et un ordinateur. Cependant, nous ne nous attarderons pas sur la stratégie de jeu que doit employer l'ordinateur, et on le fera jouer de façon aléatoire. Ainsi, quand c'est le tour de l'ordinateur, ce dernier choisira aléatoirement une des colonnes du plateau, puis choisira un nombre de pièces aléatoire parmi le nombre de pièces présentes sur la colonne choisie.

Dans le jeu, nous utiliserons une représentation du plateau du jeu en mémoire à l'aide d'un tableau à une dimension. Chaque case du tableau représente une colonne du plateau du jeu et la valeur contenue correspond au nombre de pièces présentes dans la colonne en question. Nous donnons ci-dessous, en exemple, le tableau représentant le plateau du jeu ainsi que la configuration du jeu correspondante :

0	1	2	3	4	5	6
3	10	14	15	4	20	7



Le plateau de jeu peut avoir un maximum de 20 colonnes (**`PLATEAU_MAX_COLONNES`**), chacune pouvant contenir un maximum de 35 pièces (**`PLATEAU_MAX_PIECES`**). Au début du jeu, on demande à l'usager le nombre de colonnes désiré, et on choisit aléatoirement le nombre de pièces par colonne.

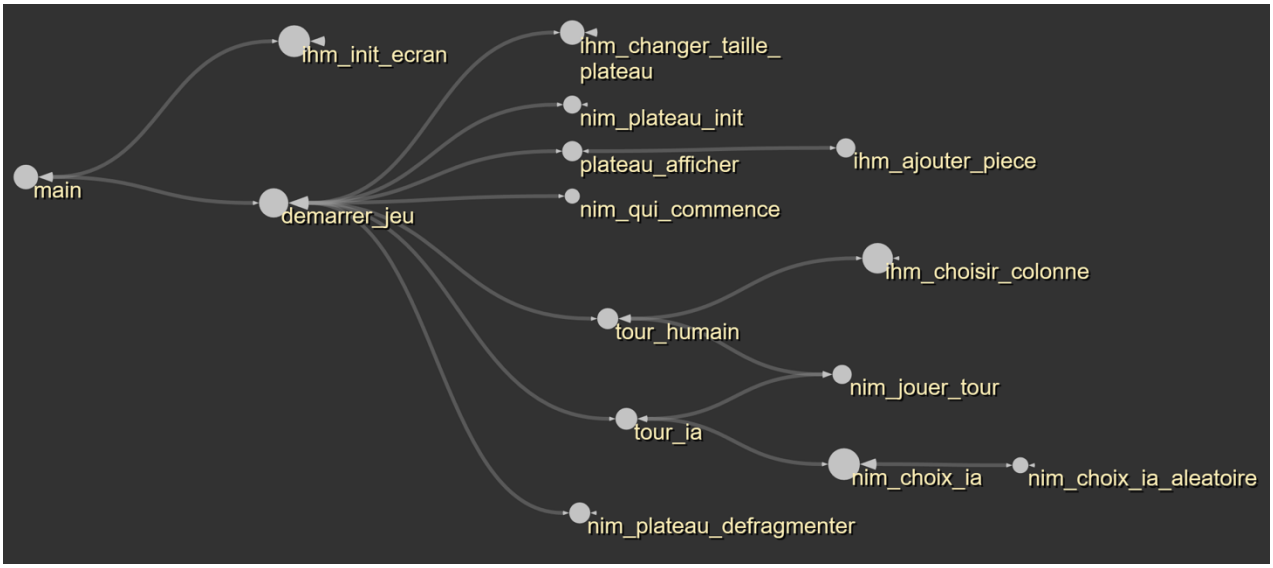
À chaque tour, le plateau du jeu sera mis à jour en conséquence. Par exemple, en partant de la configuration ci-dessous, si le prochain joueur souhaite retirer 5 pièces de la colonne 3, le tableau devient :

0	1	2	3	4	5	6
3	10	14	10	4	20	7

Supposons que le prochain joueur veuille maintenant retirer 4 pièces de la colonne 4, nous nous retrouvons avec une colonne vide. Dans ce cas, votre programme doit « défragmenter » le tableau en éliminant la colonne vide. Ainsi, le tableau modifié ressemblera à ceci :

0	1	2	3	4	5
3	10	14	10	20	7

Dans la suite, nous détaillons les fonctions à implémenter pour cette partie. Pour vous donner une meilleure vue globale de ce que vous devez faire, nous présentons ci-dessous le graphe d'appels des fonctions. Un graphe d'appels permet de représenter les appels entre les fonctions. Le graphe se lit de gauche à droite. Ainsi, la fonction main fait appel aux fonction **`ihm_init_ecran`** (module **`nim_ihm`**) et à la fonction **`demarrer_jeu`** (module **`nim_io`**). Notez toutefois que le graphe comprend que les appels principaux et ne donne aucune indication sur l'ordre d'appel des fonctions.



5.2 Fonctions à implémenter pour la partie 1 :

Pour réaliser la première partie, vous aurez à implémenter les fonctions suivantes. Il est important de tester adéquatement vos fonctions. De plus, **vous devez remettre les fonctions de test de toutes les fonctions du module « nim »** que vous regrouperez dans un module nommé « nim_test ».

```
int lire_entier(int min, int max);

Demande à l'usager de saisir un entier entre les bornes min et max
(inclusivement). La fonction doit valider la saisie et redemander à l'usager
de saisir une valeur jusqu'à l'obtention d'une valeur satisfaisante.
(Module NIM_IO)
```

```
void nim_plateau_init(int plateau[], int nb_colonnes);

Initialise le plateau de jeu en remplissant les nb_colonnes d'un nombre
aléatoire de pièces entre 1 et PLATEAU_MAX_PIECES=35. (Module NIM)
```

```
void plateau_afficher(const int plateau[], int nb_colonnes);

Affiche la configuration du plateau à l'écran. Elle affiche chacune des
colonnes en mettant une pièce par ligne, selon le nombre de pièces présentes
dans la colonne en question. (Module NIM_IO)
```

```
int nim_qui_commence(void);

Fonction qui détermine, aléatoirement, qui doit jouer en premier. Elle
retourne l'identifiant du joueur (JOUEUR_HUMAIN==0 ou JOUEUR_IA==1).
(Module NIM)
```

```
int nim_jouer_tour(int plateau[], int nb_colonnes, int colonne,
                   int nb_pieces);
Applique des changements à la configuration du plateau de jeu en retirant
"nb_pieces" de la colonne "colonne" du plateau.

La fonction s'assure que le jeu est valide et renvoie Vrai si le jeu désiré
a pu s'appliquer à la configuration actuelle du jeu. Pour être valide, la
colonne doit exister et nombre de pièces retirées doit être <= au nombre de
pièces actuellement présentes dans la colonne en question. (Module NIM)
```

```
void nim_plateau_supprimer_colonne(int plateau[], int nb_colonnes,
                                   int col_a_supprimer);

Supprime la colonne col_a_supprimer du plateau. (Module NIM)
```

```
int nim_plateau_defragmenter(int plateau[], int nb_colonnes);

Fonction qui supprime les colonnes vides du tableau en utilisant la fonction
nim_plateau_supprimer_colonne. Le nombre de colonnes restant est retourné.
(Module NIM)
```

```
void tour_humain(int plateau[], int nb_colonnes);

Déclenche le tour de l'humain en demandant à l'usager de choisir la colonne
(appel à ihm_choisir_colonne) et le nombre de pièces à retirer du plateau de
jeu.  Une fois le choix effectué, la fonction doit faire appel à
nim_jouer_tour pour appliquer les changements au plateau. (Module NIM_IO)
```

```
void tour_ia(int plateau[], int nb_colonnes, double difficulte);

Déclenche le tour de l'ordinateur. Pour connaitre le choix de l'ordinateur,
on faisant appel à la fonction nim_choix_ia.

Une fois le choix effectué, la fonction doit faire appel à nim_jouer_tour
pour appliquer les changements au plateau. (Module NIM_IO)
```

```
void nim_choix_ia(const int plateau[], int nb_colonnes, double difficulte,
                  int *choix_colonne, int *choix_nb_pieces);

Fonction qui détermine quel doit être le jeu de l'ordinateur. Cette fonction
implémente l'algorithme décrit dans l'énoncé du TP. Le choix de l'ordinateur
sera stocké dans les références choix_colonne et choix_nb_pieces.

Si une erreur de produit, la fonction stocke la valeur aberrante -1 dans les
références choix_colonne et choix_nb_pieces

Important : Dans la première partie du TP, cette fonction se contentera
d'appeler la fonction nim_choix_ia_aleatoire. (Module NIM)
```

```
void nim_choix_ia_aleatoire(const int plateau[], int nb_colonnes,
                            int * choix_colonne, int * choix_nb_pieces);

Fonction qui effectue un jeu aléatoire en choisissant au hasard une colonne,
puis au hasard le nombre de pièces à jouer de cette colonne. (Module NIM)
```

```
void demarrer_jeu(double difficulte);

Fonction qui contrôle le jeu de nim: elle donne la main, tour à tour, à
chacun des deux joueurs et déclare le gagnant une fois la partie terminée.
On quitte cette fonction quand la partie est terminée.

Pour donner la main aux joueurs, on appelle les fonctions tour_humain et
tour_ia. Après chaque tour, cette fonction se charge de défragmenter le
plateau de jeu, de modifier la taille du plateau à l'écran et d'afficher la
nouvelle configuration du plateau de jeu. (Module NIM_IO)
```

6. Partie 2 : Un ordinateur (trop?) compétitif !

Maintenant que nous disposons d'un programme qui nous permet de jouer au jeu de *Nim*, nous allons nous intéresser à l'algorithme de jeu de l'ordinateur et implémenter un algorithme qui fait gagner l'ordinateur assurément sous certaines conditions. En effet, le jeu de Nim fait partie d'une classe de jeux, dans la théorie des jeux, où il est possible de gagner de façon systématique si le joueur se trouve dans une situation particulière :

- Le jeu peut se trouver dans un de deux états : un état pair ou un état impair.
- Si le jeu se trouve dans un état impair, le prochain joueur a la certitude de gagner s'il suit un algorithme de façon systématique en ramenant le jeu à un état pair à chaque tour.
- Depuis un état pair, le jeu se trouvera obligatoirement dans un état impair une fois que le joueur aura joué.
- La fin du jeu, lorsque toutes les pièces auront été retirées du plateau de jeu (i.e. situation perdante), est un état pair.

Ainsi, la stratégie de l'ordinateur consiste à :

- Si l'état du jeu est dans un état impair, le ramener vers un état pair (perdant pour l'adversaire);
- Sinon, jouer aléatoirement en espérant que l'adversaire nous redonne la main avec un jeu dans un état impair.

Pour déterminer l'état du jeu, nous allons construire une matrice d'entiers ou chaque ligne représente le nombre de pièces, en binaire, d'une colonne du plateau de jeu. Ensuite nous faisons la somme des bits de la matrice, colonne par colonne. Prenons un exemple :

Plateau de jeu :

0	**1**	**2**	**3**	**4**	**5**	**6**
19	10	14	13	4	20	7

Algorithme – Étape 1 (construction de la matrice binaire): La première étape consiste à construire la matrice binaire du jeu. La matrice binaire correspondante à l'exemple est :

0	0	0	0	1	0	0	1	1	= 19 décimal
1	0	0	0	0	1	0	1	0	= 10 décimal
2	0	0	0	0	1	1	1	0	=14 décimal
3	0	0	0	0	1	1	0	1	=13 décimal
4	0	0	0	0	0	1	0	0	=4 décimal
5	0	0	0	1	0	1	0	0	=20 décimal
6	0	0	0	0	0	1	1	1	=7 décimal

Algorithme – Étape 2 (Déterminer l'état du jeu) : Nous construisons ensuite un tableau comprenant la somme des bits des colonnes de la matrice. Dans notre exemple, on obtient le vecteur :

0	0	0	2	3	5	4	3

Le jeu se trouve dans un état pair si toutes les sommes sont paires. Sinon, il est dans un état impair. Dans notre exemple, le jeu se trouve dans un état impair. Ceci est parfait, car c'est une situation gagnante!

Algorithme – Étape 3 (calcul du coup): Si le jeu est dans un état pair, on jouera de façon aléatoire en espérant que l'adversaire commette une erreur dans un prochain tour. Si, par contre, nous sommes dans un état impair, nous allons calculer le mouvement qui amènera le jeu dans un état pair pour l'adversaire.

- Identifier, dans la matrice des sommes, la première somme impaire en partant de la gauche. Dans notre exemple, c'est la case ayant pour indice 4 :

0	0	0	2	**3**	5	4	3

- Trouver, dans la matrice binaire à la colonne ayant le même indice, la première ligne ayant un bit à 1. Dans notre exemple :

0	0	0	0	1	0	0	1	1	= 19 décimal
1	0	0	0	0	**1**	0	1	0	= 10 décimal
2	0	0	0	0	1	1	1	0	=14 décimal
3	0	0	0	0	1	1	0	1	=13 décimal
4	0	0	0	0	0	1	0	0	=4 décimal
5	0	0	0	1	0	1	0	0	=20 décimal
6	0	0	0	0	0	1	1	1	=7 décimal
Sommes	0	0	0	2	**3**	5	4	3	

Nous avons ainsi identifié la **ligne numéro 1**. C'est la ligne sur laquelle nous allons travailler. **Ça correspond également à la colonne que l'ordinateur doit jouer (colonne 1**). Il reste à déterminer le nombre de pièces à retirer.

- Inverser les bits de la ligne identifiée, pour chaque colonne où la somme est impaire :

0	0	0	0	1	0	0	1	1	= 19 décimal
1	0	0	0	0	**0**	1	1	1	**Anciennement 10**
2	0	0	0	0	1	1	1	0	=14 décimal
3	0	0	0	0	1	1	0	1	=13 décimal
4	0	0	0	0	0	1	0	0	=4 décimal
5	0	0	0	1	0	1	0	0	=20 décimal
6	0	0	0	0	0	1	1	1	=7 décimal
Sommes	0	0	0	2	**3**	**5**	4	**3**	

- Convertir la ligne qui a été modifiée en décimal : (0000 0111) $_{\text{binaire}}$ = (7) $_{\text{décimal}}$

- La ligne 1 valait initialement 10, et nous souhaitons la transformer en 7. Nous devons donc retirer 3 pièces.

Ainsi, nous avons déterminé qu'il faut retirer **3 pièces** de la **colonne 1** du plateau de jeu. Après le tour de l'ordinateur, le plateau de jeu deviendra :

0	**1**	**2**	**3**	**4**	**5**	**6**
19	7	14	13	4	20	7

Pour implémenter cet algorithme, nous aurons besoins de fonctions qui convertissent un nombre décimal en binaire et un nombre binaire en décimal. La technique de conversion vous sera expliquée en laboratoire.

6.1 Fonctions à implémenter pour la partie 2 :

Voici la liste de fonctions qu'il sera nécessaire d'implémenter pour réaliser la partie 2. N'oubliez pas que vous devez fournir les fonctions de tests pour les modules « **codage_numerique** » et « **nim** »

```
int codage_inverser_tab_bits(int tab_bits[], int nb_bits);

Fonction qui inverse les nb_bits premières valeurs d'un tableau de bits : le
premier bit devient la dernier (et inversement), le deuxième devient l'avant
dernier, etc. Cette fonction est utilisée par la fonction codage_dec2bin.
(Module CODAGE_NUMERIQUE)

Note: nb_bits ne doit pas excéder CODAGE_NB_BITS (constante fixée à 8).
```

```
int codage_dec2bin(int nombre, int resultat[]);

Traduit un nombre décimal en binaire. Le résultat est stocké dans le tableau
"resultat" et le nombre de bits utilisés est renvoyé. Le codage du nombre
décimal doit se faire en un maximum de CODAGE_NB_BITS (fixée à 8). La
fonction renvoie le nombre de bits qui a été nécessaire pour coder le nombre
en binaire. Si le nombre requiert plus que CODAGE_NB_BITS, la fonction
renvoie 0. (Module CODAGE_NUMERIQUE)
```

```
void codage_afficher_tab_bits(const int tab_bits[], int nb_bits);

Affiche un tableau contenant des bits à l'écran. Cette fonction est utilisée
pour des fins de test.

NOTE: On assume ici que le tableau de bits est de taille exactement
CODAGE_NB_BITS. (Module CODAGE_NUMERIQUE)
```

```
int codage_bin2dec(const int tab_bits[]);

Traduit un tableau de bits, représentant un nombre en binaire, vers sa
représentation décimale. La valeur décimale est retournée par la fonction.

NOTE: On assume ici que le tableau de bits est de taille exactement
CODAGE_NB_BITS et que le bit de poids fort est à la case 0 du tableau.
(Module CODAGE_NUMERIQUE)
```

```
void nim_construire_mat_binaire(const int plateau[], int nb_colonnes,
                                int matrice[][CODAGE_NB_BITS]);
```

Construit la matrice binaire nécessaire à l'algorithme de choix de jeu de l'ordinateur. Chaque ligne de la matrice correspond à une colonne du plateau de jeu et contient la représentation binaire du nombre de pièces présentes sur la colonne en question. (Module NIM)

Note: Le résultat est stocké dans le tableau « matrice » et on assume que le tableau comprend au moins "nb_colonnes" lignes et CODAGE_NB_BITS colonnes.

```
void nim_sommes_mat_binaire(const int matrice[][CODAGE_NB_BITS],
                            int nb_lignes, int sommes[]);
```

Calcule les sommes des colonnes d'une matrice binaire de taille nb_lignes*CODAGE_NB_BITS. (Module NIM)

Note: Les résultats sont stockés dans le tableau "sommes", on assume que le tableau est au moins de taille CODAGE_NB_BITS.

```
int nim_position_premier_impaire(const int tab[]);
```

Recherche la première valeur impaire d'un tableau tab et retourne son indice.

Si le tableau ne contient aucune valeur impaire, la fonction retourne -1. (Module NIM)

```
void nim_choix_ia(const int plateau[], int nb_colonnes, double difficulte,
                  int *choix_colonne, int *choix_nb_pieces);
```

(**Note** : Vous devez modifier votre implémentation faite à la partie 1)

Fonction qui détermine quel doit être le jeu de l'ordinateur. Cette fonction implémente l'algorithme décrit dans l'énoncé du TP. Le choix de l'ordinateur sera stocké dans les références choix_colonne et choix_nb_pieces.

La valeur de difficulté (entre 0 et 1) détermine le niveau de difficulté que doit avoir l'ordinateur. Par exemple, si le niveau de difficulté est de 0.7, l'ordinateur joue en utilisant son algorithme dans 70% du temps. Le reste du temps, l'ordinateur joue aléatoirement. On utilise un tirage aléatoire pour savoir comment jouer.

Si une erreur se produit, la fonction stocke la valeur aberrante -1 dans les références choix_colonne et choix_nb_pieces. (Module NIM)

7. Remise et contraintes de l'enseignant

La remise du travail complet devra être effectuée sur Moodle à la date indiquée et respecter les exigences de remise des travaux pratiques (sur le site Moodle dans la section *Travaux pratiques*). De plus :

- La conception de ce travail (découpage en fonctions et modules) qui vous est décrite n'est ni la seule conception possible, ni la meilleure. Cependant, celle-ci répond à des objectifs pédagogiques et d'évaluation. Il vous est donc demandé de la respecter (modules, fonctions, paramètres, spécifications des fonctions, etc.). Tout manquement pourrait affecter votre note.
- Aucune variable globale ne sera acceptée – L'utilisation d'une seule variable globale entraine une pénalité de 25%.
- L'instruction `GOTO` est interdite – L'utilisation de cette instruction entraine une pénalité de 25%.
- Votre programme doit se comporter de la même façon que l'exécutable fourni avec cet énoncé.
- La remise se fait électroniquement sur Moodle. Le site sera configuré **pour refuser tout travail en retard**. Aucun travail ne pourra être remis en retard et, par conséquent, la note de 0 sera attribuée.
- Un programme remis qui ne compile pas se verra attribuer la note de 0.

BON TRAVAIL !