Projet tutoré : programmation du jeu Push Over

L'objectif de ce projet tutoré est de coder en langage C le jeu Push Over.

Le projet est à faire en binôme en dehors des heures de cours. Le code de ce programme devra impérativement respecter les spécifications données dans ce document. Cela signifie que la structure de données devra être la même que celle donnée ci-dessous et toutes les fonctions demandées (avec les mêmes en-têtes) devront être implémentées. *Il est cependant possible d'ajouter d'autres structures de données et/ou fonctions.* Ce travail sera évalué lors de 4 séances. À chaque séance, chaque binôme aura un créneau de 15 minutes défini préalablement pendant lequel il sera évalué.

La première partie décrit les règles du jeu Push Over. La deuxième partie explique le travail à réaliser et la troisième décrit l'évaluation du projet en indiquant notamment le travail à présenter à chaque séance d'évaluation.

Remarque : Certains tests doivent être évalués avec la fonction `assert`. Cette fonction est une fonction du langage C dont la déclaration se trouve dans le fichier d'en-tête `assert.h`. Elle prend en paramètre un test. Lors de l'appel de cette fonction, si le test est évalué à `0`, un message d'erreur est affiché et le programme s'arrête. Un exemple d'utilisation de la fonction `assert` est donné dans le fichier `utilisation_assert.c` sur la page Web `http://lipn.univ-paris13.fr/~lacroix/enseignement.php`.

1 Description du jeu Push Over

Le Push Over est un jeu à deux joueurs utilisant un plateau carré de `n` * `n` cases (`n` est un entier compris entre 3 et 8 inclus. Il est généralement égal à 5) et deux couleurs de pions. Chaque joueur choisit une couleur de pions. Les deux joueurs insèrent à tour de rôle un pion sur le plateau. Le jeu se termine lorsqu'il existe un alignement horizontal ou vertical de `n` pions d'une même couleur.

1.1 Insertion d'un pion

À tour de rôle, chaque joueur introduit un de ses pions par un des 4 côtés du plateau de jeu. En faisant cela, il décale les pions se trouvant déjà sur cette ligne ou colonne. Il peut même ainsi éjecter du plateau un des pions s'y trouvant déjà, à condition que **ce pion soit de sa couleur**.

Exemple : Considérons la partie en cours suivante. Lorsque le joueur ayant les pions noirs insère son pion sur le côté gauche à l'endroit indiqué par la flèche sur la figure de gauche, il décale tous les pions de la ligne. Après son insertion, le plateau est celui donné par la figure à droite.
Remarque : Le joueur n'aurait pas pu insérer son pion noir sur la troisième colonne (en haut ou en bas) puisqu'en cas d'insertion, le pion éjecté est blanc.

1.2 Fin de partie et gagnant

Une ligne ou colonne est dite *gagnante* pour un joueur si toutes ses cases contiennent un pion du joueur. La partie partie s'arrête avec l'apparition d'une ligne ou d'une colonne gagnante. L'insertion d'un seul pion pouvant entraîner la formation de plusieurs lignes ou colonnes gagnantes, le gagnant est le joueur qui comptabilise le maximum de lignes et de colonnes gagnantes à la fin de la partie. En cas d'égalité, il y a match nul.

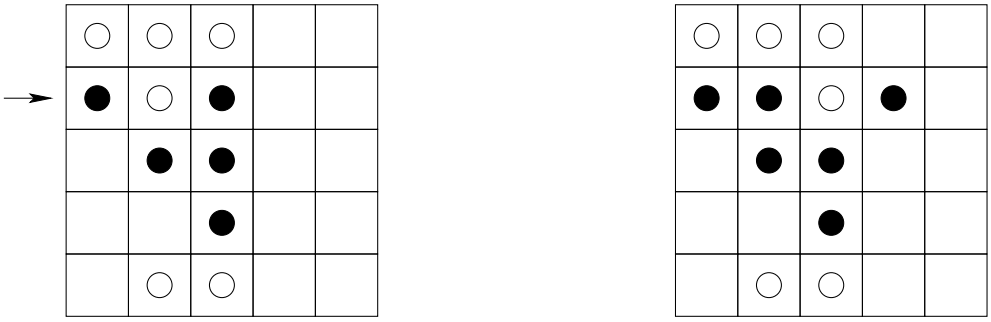

FIGURE 1 – Exemple d'insertion d'un pion noir

2 Travail à réaliser

Le développement du programme se divise en 4 parties. Chaque partie est décrite précisément dans ce qui suit.

2.1 Partie 1

Cette partie explique la première structure de données à définir modélisant le plateau de jeu du Push Over. Elle donne aussi les fonctions de base à implémenter, permettant de modifier cette structure de données.

Question 1 : Définir le type structuré `plateau` contenant les champs suivants :
- un pointeur d'entiers `tableau`,
- un entier `n`.

Le type structuré `plateau` permettra de représenter le plateau d'une partie de Push Over. Le champ `n` correspondra au nombre de cases par ligne et par colonne. Le champ tableau sera un tableau dynamique à une dimension représentant le plateau de jeu. Les lignes seront codées les unes à la suite des autres. Les `n` premières cases de `tableau` représenteront la première ligne du plateau, les `n` suivantes la deuxième ligne, etc.

La valeur de chaque élément de `tableau` indiquera le contenu de la case du plateau de jeu associée :
- 0 indiquera que la case associée sur le plateau de jeu est libre (*i.e.*, ne contient aucun pion),
- 1 indiquera qu'un pion noir se trouve sur la case du plateau de jeu associée,
- -1 indiquera qu'un pion blanc se trouve sur la case du plateau de jeu associée.

Ainsi, le plateau de gauche de la figure 1 sera représenté par le tableau `tableau` suivant :

-1	-1	-1	0	0	1	-1	1	0	0	0	1	1	0	0	0	0	1	0	0	0	-1	-1	0	0
0					5					10					15					20					

Question 2 : Définir la fonction `creerPlateau` prenant en paramètre un entier. La fonction allouera dynamiquement une variable de type `plateau`, dont le champ `n` sera égal à la valeur passée en paramètre et le champ `tableau` représentera un plateau de taille `n` * `n` ne contenant aucun pion. La fonction retournera l'adresse de la variable de type `plateau` allouée dynamiquement. La fonction devra vérifier à l'aide de la fonction `assert` que l'entier passé en paramètre est compris entre 2 et 8 inclus.

```
1 /*!
2  *  Fonction allouant dynamiquement un plateau dont l'adresse est retournée.
3  *
```

```
 *  \param n : taille d'un côté
 */
plateau* creerPlateau(int n) { ... }
```

Question 3 : Définir la fonction `detruirePlateau` prenant en paramètre un pointeur sur un `plateau` alloué dynamiquement et libérant la mémoire pointée par ce dernier. La fonction devra vérifier, à l'aide de la fonction `assert` que le pointeur ne contient pas la valeur `NULL`.

```
/*!
 *  Fonction désallouant dynamiquement le plateau passé en paramètre
 *
 *  \param p : plateau à désallouer
 */
void detruirePlateau(plateau * p) { ... }
```

Question 4 : Définir la fonction `getCase` prenant en paramètre un pointeur sur une variable de type `plateau`, un indice de ligne `i` et un indice de colonne `j`, et retournant la valeur de la case dans le plateau. La fonction vérifiera d'abord à l'aide de la fonction `assert` que les numéros de ligne et colonne sont valides, c'est-à-dire compris entre 0 et `n`-1 inclus.

```
/*!
 * Fonction retournant la valeur de la case (i,j) du plateau p
 *
 *  \param p : plateau
 *  \param i : entier correspondant au numéro de ligne
 *  \param j : entier correspondant au numéro de colonne
 */
int getCase(plateau * p, int i, int j) { ... }
```

Exemple : Supposons que `pEx` est un pointeur sur une variable de type `plateau` dont le champ `tableau` correspond à la représentation mémoire du plateau de gauche de la figure 1. L'exécution du code :

```
printf("Valeur sur la case (0,0) : %d\n", getCase(pEx,0,0));
printf("Valeur sur la case (1,0) : %d\n", getCase(pEx,1,0));
printf("Valeur sur la case (4,4) : %d\n", getCase(pEx,4,4));
```

affichera alors :

```
Valeur sur la case (0,0) : -1
Valeur sur la case (1,0) : 1
Valeur sur la case (4,4) : 0
```

Question 5 : Définir la fonction `setCase` permettant de modifier la valeur d'une case du plateau. La fonction vérifiera d'abord à l'aide de la fonction `assert` que les numéros de ligne et colonne sont valides, et que la valeur que l'on souhaite affecter à la case est un entier compris entre -1 et 1.

```
/*!
 *
 * Fonction modifiant la valeur de la case (i,j) du plateau p
 *
 *  \param p : plateau
 *  \param i : entier compris entre [0,n-1]
 *  \param j : entier compris entre [0,n-1]
 *  \param valeur : entier compris entre [-1,1]
 */
void setCase(plateau *p, int i, int j, int val) { ... }
```

Exemple : Supposons que `pEx` est un pointeur sur une variable de type `plateau` dont le champ `tableau` correspond à la représentation mémoire du plateau de gauche de la figure 1. Le code

```
setCase(pEx,1,1,1);
setCase(pEx,1,2,-1);
setCase(pEx,1,3,1);
```

modifiera le tableau qui correspondra alors au plateau de droite de la figure 1.

Question 6 : Définir la fonction `affichage`. Cette fonction devra afficher à l'écran le plateau de jeu correspondant à la partie passée en paramètre.

```
/*!
 * Fonction affichant le plateau sur le terminal
 *
 * \param p : pointeur sur la partie que l'on souhaite afficher
 */
void affichage(partie * p) { ... }
```

Il existe plusieurs façons de réaliser l'affichage du plateau. On peut distinguer 3 manières :

– Affichage simple : on affiche les valeurs du champ `tableau` sous forme d'un tableau à deux dimensions. Un exemple est donné par la figure 2.

```
-1   -1   -1    0    0
 1   -1    1    0    0
 0    1    1    0    0
 0    0    1    0    0
 0   -1   -1    0    0
```

FIGURE 2 – Affichage simple du damier

– Affichage moyen : on affiche les lettres `N` ou `B` au lieu des valeurs -1 et 1 (rien s'il n'y a pas de pion) et chaque case prend plusieurs lignes et plusieurs colonnes. Un exemple est donné par la figure 3.

```
* * * * * * * * * * * * * * * * * * * * *
*     *     *     *     *     *     *     *
*  B  *  B  *  B  *     *     *     *     *
*     *     *     *     *     *     *     *
* * * * * * * * * * * * * * * * * * * * *
*     *     *     *     *     *     *     *
*  N  *  B  *  N  *     *     *     *     *
*     *     *     *     *     *     *     *
* * * * * * * * * * * * * * * * * * * * *
*     *     *     *     *     *     *     *
*     *  N  *  N  *     *     *     *     *
*     *     *     *     *     *     *     *
* * * * * * * * * * * * * * * * * * * * *
*     *     *     *     *     *     *     *
*     *     *  N  *     *     *     *     *
*     *     *     *     *     *     *     *
* * * * * * * * * * * * * * * * * * * * *
*     *     *     *     *     *     *     *
*     *  B  *  B  *     *     *     *     *
*     *     *     *     *     *     *     *
* * * * * * * * * * * * * * * * * * * * *
```

FIGURE 3 – Affichage moyen du damier

– Affichage difficile : on affiche les pions à l'aide de symboles en utilisant les couleurs du terminal. Un exemple est donné par la figure 4. **Pour utiliser les couleurs du terminal, vous pouvez utiliser les fichiers `couleursTerminal.c` et `couleursTerminal.h` qui se trouvent sur la page** `http://lipn.univ-paris13.fr/~lacroix/enseignement.php`. Les explications se trouvent dans le deuxième fichier en commentaires.

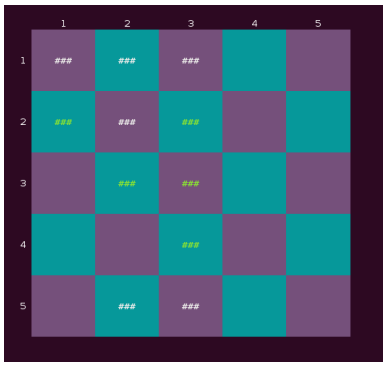

FIGURE 4 – Affichage difficile du damier

Vous pouvez choisir l'une des trois méthodes d'affichage, sachant que plus la méthode est simple, moins elle rapporte de points. (Cependant, il vaut vraiment mieux faire la plus méthode simple correctement que la méthode la plus difficile fausse !)

2.2 Partie 2

Les fonctions à implémenter dans cette partie définissent les mouvements valides et permettent à un joueur de poser un pion.

Question 7 : Définir la fonction `indiceValide` prenant en paramètre un pointeur sur un plateau ainsi qu'un entier et retournant 1 si l'entier correspond à un numéro de ligne ou de colonne valide, et 0 sinon. Voici l'en-tête de la fonction et son commentaire :

```
/*!
 * Fonction retournant 1 si l'indice est valide par rapport au plateau
 *
 *  \param p : plateau
 *  \param indice : entier compris entre [0,n-1]
 */
int  indiceValide(plateau * p, int indice) { ... }
```

Question 8 : Définir la fonction `caseValide` prenant en paramètre un pointeur sur un tableau et deux entiers, et retournant 1 si les deux entiers correspondent à l'indice d'une case du plateau, et 0 sinon.

```
/*!
 * Fonction retournant 1 si la case (indiceLigne,indiceColonne) existe sur le plateau
 *
 *  \param p : plateau
 *  \param indiceLigne : entier correspondant à un indice de ligne
 *  \param indiceColonne : entier correspondant à un indice de colonne
 */
int caseValide(plateau * p, int indiceLigne, int indiceColonne) { ... }
```

Question 9 : Définir la fonction `caseVide` prenant en paramètre un pointeur sur un plateau et les indices d'une case du plateau et retournant 1 si la case correspondante est vide, et 0 sinon.

```
/*!
 * Fonction retournant 1 si la case (i,j) du plateau p est vide
 *
```

```
 *  \param p : plateau
 *  \param i : entier correspondant à un indice de ligne
 *  \param j : entier correspondant à un indice de colonne
 */
int caseVide(plateau * p, int i, int j) { ... }
```

Question 10 : Définir la fonction `insertionPionPossible`. Cette fonction prend en paramètre un pointeur sur un plateau, une case `(ligne,col)` du bord du plateau (définie par des indices de ligne et de colonne), une direction d'insertion et une couleur de pion. La direction est une des 4 directions possibles données dans la figure 5 par les flèches. Informatiquement, chaque direction sera codée à l'aide de deux entiers `di` et `dj` compris entre -1 et 1 (*c.f.* les valeurs données pour chaque direction dans la figure 5). On supposera que la case (`ligne,col`) est une case du bord du plateau et que la direction est une direction d'insertion possible pour la case (`ligne,col`). Excepté pour les 4 coins du plateau, il n'y a qu'une seule direction d'insertion possible pour une case donnée. Le paramètre `pion` vaut -1 ou 1 et représente la couleur du pion que l'on insère. Ainsi, l'insertion donnée par la figure 1 serait donnée par les paramètres (1,0) pour la case, 0,1 pour la direction et 1 pour la couleur du pion. La fonction retournera 1 si le pion peut être inséré dans la direction `di,dj` à partir de la case (`ligne,col`), et 0 sinon.

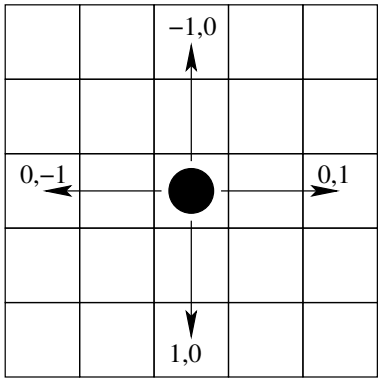

FIGURE 5 – Directions possibles et valeurs associées

Voici l'en-tête de la fonction :

```
/*!
 * Fonction retournant 1 si le pion peut être ajouté à partir de la case (ligne,col)
 * dans la direction donnée par di,dj
 *
 *  \param p : plateau
 *  \param ligne : ligne de la case où l'on insère le pion
 *  \param col : colonne de la case où l'on insère le pion
 *  \param di : direction sur les lignes
 *              // -1 si l'on insère vers le haut,
 *              //  0 si l'on reste sur la même ligne,
 *              //  1 si l'on insère vers le haut
 *  \param dj : direction sur les colonnes
 *              // -1 si l'on insère vers la gauche,
 *              //  0 si l'on reste sur la même colonne,
 *              //  1 si l'on insère vers la droite
 *  \param pion : 1 ou -1 suivant le joueur
 */
int  insertionPionPossible (plateau * p, int ligne, int col, int di, int dj, int pion) { ... }
```

Question 11 : Définir la fonction `insertionPion`. Cette fonction prend les mêmes paramètres que la fonction `insertionPionPossible` et doit effectuer l'insertion du pion sur le plateau. On supposera ici que les paramètres correspondent à une insertion possible.

```
/*!
 * Fonction insérant le pion à partir de la case (ligne,col)
 * dans la direction donnée par di,dj
 *
 *  \param p : plateau
 *  \param ligne : ligne de la case où l'on insère le pion
 *  \param col : colonne de la case où l'on insère le pion
 *  \param di : direction sur les lignes
 *              // -1 si l'on insère vers le haut,
 *              //  0 si l'on reste sur la même ligne,
 *              //  1 si l'on insère vers le haut
 *  \param dj : direction sur les colonnes
 *              // -1 si l'on insère vers la gauche,
 *              //  0 si l'on reste sur la même colonne,
 *              //  1 si l'on insère vers la droite
 *  \param pion : 1 ou -1 suivant le joueur
 */
void insertionPion(plateau * p, int ligne, int col, int di, int dj, int pion) { ... }
```

Question 12 : Définir la fonction `gagne` prenant en paramètre un pointeur sur un plateau. La fonction retournera 0 si la partie n'est pas finie, -1 si le premier joueur a gagné, 1 si le deuxième joueur a gagné, et 2 s'il y a match nul.

```
/*!
 * Fonction retournant 1 ou -1 si l'un des joueurs a gagné, 2 si match nul et 0 si la partie
 * n'est pas finie.
 *
 *  \param p : plateau
 */
int gagne(plateau * p) { ... }
```

2.3 Partie 3

Question 13 : Définir le type structuré `partie` représentant une partie et contenant les champs suivants :

- une chaîne de caractères `nomJoueur1` permettant de stocker 127 caractères,
- une chaîne de caractères `nomJoueur2` permettant de stocker 127 caractères,
- un entier `pionJoueurCourant`,
- un pointeur de `plateau` `p`.

Les champs `nomJoueur1` et `nomJoueur2` contiendront respectivement les noms des deux joueurs. L'entier `pionJoueurCourant` contiendra la valeur -1 ou 1 suivant si c'est au tour du premier ou du deuxième joueur. Le pointeur de type `plateau` contiendra l'adresse de la variable du plateau de la partie.

Question 14 : Définir la fonction `creerPartie`. Cette fonction demandera les noms des deux joueurs et la taille du plateau de jeu sur lequel ils souhaitent disputer la partie. Elle allouera dynamiquement une variable de type `partie` avec les informations saisies au clavier (le plateau devra aussi être créé en fonction de la taille saisie au clavier).

```
/*!
 * Fonction allouant dynamiquement une partie et renvoyant l'adresse de celle-ci.
```

```
 * La taille du plateau et les noms des joueurs sont saisis par l'utilisateur.
 *
 */
partie * creerPartie () { ... }
```

Question 15 : Définir la fonction `detruirePartie`. Cette fonction prendra en paramètre un pointeur sur une partie et libérera toute la mémoire allouée pour cette partie.

```
/*!
 * Fonction désallouant dynamiquement la partie passée en paramètre
 *
 * \param pa : partie à désallouer
 */
void detruirePartie (partie * pa) { ... }
```

Question 16 : Définir la fonction `changePionJoueurCourant` prenant en paramètre un pointeur sur une partie et modifiant le joueur courant, c'est-à-dire le joueur dont c'est le tour de jouer, de la partie.

```
/*!
 * Fonction changeant le pion du joueur courant (1 ou -1)
 *
 * \param pa : partie en cours
 */
void changePionJoueurCourant(partie * pa) { ... }
```

Question 17 : Définir la fonction `saisieJoueur` permettant à l'utilisateur de saisir un mouvement, c'est-à-dire une insertion, ou de retourner au menu principal. La fonction retournera 0 si l'utilisateur souhaite retourner au menu principal, et 1 s'il saisit un mouvement. La saisie sera répétée jusqu'à ce que le joueur saisisse un mouvement possible.

```
/*!
 * Fonction modifiant les variables ligne, col, di et dj passées en paramètre avec
 * les valeurs saisies par l'utilisateur. Elle retourne 0 si le joueur a appuyé
 * sur la lettre M et 1 sinon.
 *
 *  \param p : partie
 *  \param ligne : ligne du plateau où l'on souhaite insérer le pion du joueur courant de la partie
 *  \param col : colonne du plateau où l'on souhaite insérer le pion du joueur courant de la partie
 *  \param di : direction d'insertion (ligne)
 *  \param dj : direction d'insertion (colonne)
 */
int saisieJoueur (partie * pa, int * ligne, int * col, int * di, int * dj) { ... }
```

Il y a plusieurs façons de réaliser cette saisie. On peut distinguer deux manières (une facile et une plus compliquée) :

- Cas simple : demander à l'utilisateur s'il souhaite retourner au menu principal ou saisir une case. Dans ce dernier cas, on demande le numéro de ligne puis le numéro de colonne, et la direction (à l'aide de deux entiers).
- Cas difficile : l'utilisateur saisit une chaîne de caractères : `M` s'il souhaite retourner au menu principal ou une insertion définie par une lettre suivi d'un chiffre. La lettre (`H`, `B`, `G` ou `D`) indique sur quel bord du plateau il souhaite insérer son pion (en haut, en bas, à gauche ou à droite). Le chiffre donne le numéro de ligne ou de colonne dans lequel le joueur souhaite insérer son pion. On suppose dans ce cas que le numéro est compris entre 1 et `n` inclus. À titre d'exemple, si le joueur souhaite saisir l'insertion donnée par la figure 1, il saisit alors `G2`.

Vous pouvez choisir l'une des deux méthodes, sachant que la méthode simple rapporte un peu moins de points que la manière difficile. (Cependant, il vaut vraiment mieux faire la méthode simple correctement que la manière difficile fausse !)

Question 18 : Définir la fonction `tourJoueur`. Cette fonction prend en paramètre une partie. Elle commence par afficher le plateau puis le nom du joueur courant. Elle demande ensuite au joueur de saisir son mouvement. Si l'utilisateur souhaite accéder au menu, elle retourne 0. Sinon, elle effectue l'insertion du pion du joueur et retourne 1.

```
/*!
 * Fonction permettant au joueur courant de jouer.
 * La fonction retourne 1 si le joueur joue, et 0 sinon (s'il appuie sur la touche M).
 *
 * \param pa : partie en cours
 */
int tourJoueurCourant(partie * pa) { ... }
```

Question 19 : Définir la fonction `jouer`. Cette fonction prend en paramètre une partie. Chaque joueur joue à tour de rôle jusqu'à ce que la partie soit finie, ou qu'un joueur souhaite accéder au menu principal. Si la partie est terminée, la fonction retourne 1. Si un joueur souhaite accéder au menu principal, elle retourne 0.

```
/*!
 * Fonction permettant de jouer à Push over.
 *
 * \param pa : pointeur sur une partie en cours (elle doit être allouée). La partie peut ne pas être
 * un début de partie.
 *
 * Retourne 0 si un joueur veut accéder au menu principal, 1 si la partie s'est terminée normalement
 * (Partie gagnée ou nulle)
 */
int jouer( partie * pa) { ... }
```

2.4 Partie 4

Cette partie introduit les fonctions de menu, de sauvegarde et de chargement de partie.

Question 20 : Définir les fonctions de `sauvegarde` et `chargement` d'une partie.

```
/*!
 *  Fonction sauvegardant la partie passée en paramètre. Retourne 0 en cas de problème, 1 sinon.
 *
 */
int sauvegardePartie( partie *pa) { ... }
```

```
/*!
 *  Fonction chargeant la partie en cours. Retourne l'adresse de la partie nouvellement créée.
 *
 *  Si aucune partie n'a été sauvegardée, une nouvelle partie est créée.
 */
partie * chargementPartie() { ... }
```

On supposera qu'une seule partie peut être sauvegardée à la fois. Le chargement et la sauvegarde se feront à l'aide du fichier `sauvegarde.txt`. Vous choisirez vous-même le format de ce fichier.

Question 21 : Définir la fonction `menu`. La fonction doit afficher la liste des choix possibles. Si une partie est en cours, c'est-à-dire que le pointeur passé en paramètre est différent de `NULL`, les choix sont :

– quitter le jeu/programme (retourne 0),
– commencer une nouvelle partie (retourne 1),
– charger la dernière partie sauvegardée (retourne 2),
– sauvegarder la partie courante (retourne 3),
– reprendre la partie en cours (retourne 4).

Si aucune partie n'est en cours, les deux derniers choix ne sont pas possibles et ne doivent pas être proposés à l'utilisateur. La saisie est répétée jusqu'à ce que l'utilisateur choisisse une des options proposées.

```
/*!
 * Fonction affichant le menu (si p!=NULL, on permet de reprendre une partie et de
 * sauvegarder la partie)
 *
 * \param p : pointeur sur la partie courante.
 * Valeur de retour :
 *      0 si les joueurs veulent quitter le jeu,
 *      1 s'ils veulent commencer une nouvelle partie,
 *      2 s'ils veulent charger la dernière partie sauvegardée,
 *      3 s'ils veulent sauvegarder la partie courante,
 *      4 s'ils veulent reprendre la partie courante.
 */
int menu(partie *pa) { ... }
```

Question 22 : Écrire le programme principal permettant de jouer à Push Over.

Question 23 : Séparer le code en plusieurs fichiers sources et créer un Makefile permettant la compilation du programme.

Améliorations : Il est possible bien sûr d'améliorer le code de ce programme en ajoutant d'autres fonctionnalités telles que :
– plusieurs sauvegardes possibles (par exemple 5 sauvegardes différentes),
– affichage de la liste des insertions possibles pour un joueur afin qu'il puisse choisir le mouvement qu'il souhaite parmi ceux possibles,
– permettre la possibilité de jouer des parties en plusieurs manches. Les deux joueurs s'affrontent. Le gagnant est le premier à remporter 3 parties,
– etc.

Vous pouvez si vous le souhaitez (et si tout le reste a été réalisé) ajouter ces fonctionnalités (ou d'autres) dans votre programme. Ces ajouts pourront faire l'objet de quelques points bonus.

3 Évaluation du projet

Le projet tutoré sera évalué au cours de 4 séances, chaque séance correspondant à l'évaluation d'une partie (autrement dit, le travail à présenter lors de la première séance correspond à celui décrit dans la partie 1, etc). L'évaluation portera sur :
– **l'implémentation** des fonctions demandéees à chaque séance,
– **la pertinence des jeux d'essai réalisés** permettant de tester les différents cas de figure pour chaque fonction demandée,
– **les réponses aux questions** posées lors de l'évaluation.