

PETILLOT Guillaume
HOT Aurélien
B1

Rapport du Mini-Projet d'Informatique : **PUISSANCE 4**

Introduction :

Notre but dans ce sujet était de pouvoir jouer au jeu « Puissance 4 » à un ou deux joueurs et de pouvoir sauvegarder puis charger une partie. Nous nous sommes servis des outils de base de la programmation, de structures dynamiques (listes chaînées), de la récursivité et avons manipulé des fichiers.

I. Analyse du problème, Cahier des charges

Notre première démarche a été de se représenter les différentes étapes nécessaires à la bonne exécution du programme :

- Créer un jeu
- Afficher le plateau de jeu
- Demander au premier joueur de jouer
- Afficher le pion
- Faire les tests d'alignement
- Si le joueur 1 n'a pas gagné, faire jouer le joueur 2 (dans le mode un joueur, c'est bien sûr l'ordinateur qui joue en tant que joueur 2.)
- Et ainsi de suite...

Cela nous a amené à définir des types spéciaux :

- type collection
- type element_collection
- type joueur
- type jeu
- type grille
- type chaine
- type coor
- type liste_coord

et a créer des sous-programmes, dont voici les principaux :

```

void puissance4() ;
//lance le jeu

jeu creer_jeu_vide() ;
// retourne un jeu vide

void jouer(jeu J, joueur P);
//fait jouer le joueur P dans le jeu J

bool gagner(jeu J, joueur P);
//retourne VRAI si le joueur P a gagné le jeu J
//retourne FAUX sinon

bool grille_pleine(jeu J);
//retourne VRAI si la grille est pleine
//FAUX sinon

void changer_joueur(joueur &P);
//si P=P1 : change le joueur P1 en P2
//sinon change P2 en P1

void afficher_grille(jeu J);
//affiche le plateau du jeu J au dernier coup joué

```

II . Implantation

type collection = adresse de element_collection

```

type element_collection = enregistrement
    char          pion
    collection     suivant

```

```

type joueur = enregistrement
    char          pion
    entier        numéro

```

```

type jeu = enregistrement
    collection    t [7]
    entier        dernier_coup

```

caractère grille [6][9]

caractère chaine[15];

```

type coor = enregistrement
    entier        colonne

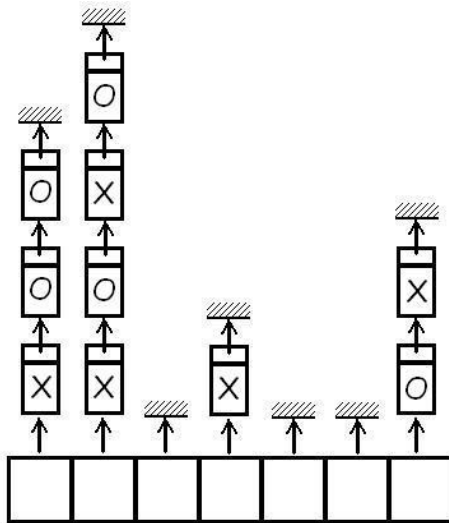
```

```

type liste_coord = enregistrement
    entier        Dim
    coordonees    t[50]

```

Le jeu est modélisé dans la mémoire de l'ordinateur par un tableau de 7 cases contenant chacune une liste chaînée



A chaque nouveau pion joué dans une colonne, on ajoute un nouvel élément à la fin de la liste chaînée correspondante.

Voici quelques sous-programmes utilisés assez souvent lors du jeu :

➤ jouer

procédure jouer(jeu J, joueur P) ;
 //fait jouer le joueur P dans le jeu J

Début

```
entier i ;
i ← 0 ;
afficher « Joueur », P.numero , « : Entrez le numero de colonne : » ;
acquérir i ;

tantque (i > 7 ou i = 0) faire
  afficher en rouge : i « n'est pas une colonne acceptable !! » ;
  afficher « Entrez le numéro de la colonne »
  acquérir i ;
fintantque ;

tantque colonne_pleine(J, J.t[i]) = VRAI faire
  afficher en rouge : « La colonne », i, « est pleine !! » ;
  afficher « Entrez le numéro de la colonne » ;
  acquérir i ;

  tantque (i > 7 ou i = 0) faire
    afficher en rouge : i « n'est pas une colonne acceptable !! » ;
    afficher « Entrez le numéro de la colonne »
    acquérir i ;
  fintantque ;
fintantque ;

ajouter_en_fin(P.pion, J.t[i]) ;
J.dernier_coup ← i ;
```

Fin

➤ ajouter_en_fin

procédure ajouter_en_fin(char X (E), collection C (E/S)) ;

// ajoute l'element X en fin de la collection C

Début

si est_vide(C) = VRAI alors

 ajouter_en_tete(X,C) ;

sinon ajouter_en_fin(X,C→suivant) ;

finsi ;

Fin

➤ test_vert

fonction boolean test_vert(jeu J, collection C);

//teste si 4 pions au moins dans la collection C sont alignés verticalement

//dans le jeu J à la dernière colonne jouée

Début

 entier i, Nb ;

 bool B ;

 i ← J.dernier_coup;

 B ← FAUX ;

 C ← J.t[i];

 Nb ← 1;

tantque C → suivant ≠ NULL alors

si C → pion = (C → suivant) → pion alors

 C ← C → suivant ;

 Nb ← Nb + 1;

sinon Nb ← 1 ;

finsi

si Nb ≥ 4 alors B ← VRAI;

finsi

 retourner B ;

Fin

Vous trouverez dans

- chargement.h :

jeu charger(int m);

//retourne un jeu enregistré précédement et donne le choix dans le mode de jeu m (un ou deux joueurs)

void deux_joueur_charge(jeu J, grille G);

//lance un jeu J, et la grille G, à deux joueurs enregistré précédement

void ordi_charge(jeu &J, joueur P, liste_coord C, int j);

//charge une partie enregistrée

void un_joueur_charge(jeu J, grille G);

//lance un jeu, et la grille G, contre l'ordinateur enregistré précédement

- collection.h :

typedef struct element_collection * collection ;

typedef struct element_collection {

 char pion ;

 collection suivant ;

} element_collection ;

//définition des types

collection creer_liste_vide() ;

// retourne une collection vide

char ieme_element(int i , collection C) ;

// retourne le ieme element de la collection C

bool est_vide(collection C) ;

// retourne true si C est vide ; false sinon

void ajouter_en_tete(char X , collection & C) ;

// ajoute l'element X en tete de la collection C, X : entrée et C : entrée/sortie

void ajouter_en_fin(char X , collection & C) ;

// ajoute l'element X en fin de la collection C, X : entrée et C : entrée/sortie

int nb_element(collection C);

//retourne le nombre d'éléments de la collection C

- Divers.h

#define ALEA_MAX 2147418112

// constante définissant le plus grand entier aléatoire que peut fournir la fonction aleatoire

void init_aleatoire(void);

// initialise le générateur de nombres aléatoire procédure à appeler une seule fois avant le premier appel de la fonction "aleatoire(N)"

int aleatoire(int N);

// retourne un entier généré aléatoirement compris entre 0 et N. Attention : N doit être <= à ALEA_MAX

void attendre(void);

// attend la frappe d'une touche au clavier

- graphique.h

typedef char grille [6][9];

//définit un tableau de 6x9 cases

void afficher_grille_vide(grille G);

//affiche une grille vide

void afficher_grille(grille G);

//affiche la grille du jeu en cours

- jouer.h :

void puissance4();

//lance le jeu

void jouer(jeu &J, joueur P);

//fait jouer le joueur P dans le jeu J

void deux_joueur();

//lance un jeu Humain VS Humain

void ordi_joue(jeu &J, joueur P);

//fait jouer l'ordinateur dans une colonne aléatoire

```
void un_joueur();  
//lance un jeu Humain VS Ordinateur
```

- outils_jeu.h :

```
typedef struct jeu{  
    collection t[8];  
    int dernier_coup; //dernier_coup indique le numéro de la dernière colonne jouée  
}jeu;  
//définit le type jeu
```

```
typedef struct joueur{  
    char pion; //retourne X comme pion pour le joueur 1, O pour le joueur 2  
    int numero; //retourne le numéro du joueur (1 ou 2)  
}joueur;  
//définit le type joueur
```

```
jeu creer_jeu_vide() ;  
//retourne un jeu vide
```

```
void changer_joueur(joueur &P);  
//si P=P1 : change le joueur P1 en P2, sinon change P2 en P1
```

```
char pion_case(jeu J, int l, int c);  
//retourne le pion ("X" ou "O") placé à la l ième ligne et à la c ième colonne du jeu J,  
retourne "." si la case est vide
```

```
int ligne(jeu J);  
//retourne la ligne du dernier pion joué
```

```
int colonne(jeu J);  
//retourne la colonne du dernier pion joué
```

- sauvegarde.h :

```
typedef char chaine[15];  
//définit le type chaine
```

```
typedef struct coor {int colonne;}coordonees;  
//définit le type coor
```

```
typedef struct liste_coord {int Dim; coordonees t[50];}liste_coord;  
//définit le type liste_coord
```

```
void enregistrer(chaine Nom_fichier, jeu J);  
//ecris les coups joués au cours du jeu J dans le fichier texte intitulé Nom_fichier
```

```
liste_coord lire_coord(chaine Nom_fichier);  
//lis le fichier texte intitulé Nom_fichier
```

```
coor fabriquer(jeu J, int c);  
//créé une liste de "coordonnées" donnant la colonne du dernier coup joué du jeu J
```

```
void ecrire_coord(chaine Nom_fichier, liste_coord C);  
//écrit les coordonnées de tous les coups joués dans un fichier nommé Nom_fichier
```

```
liste_coord creer_liste_coord_vide();  
//créé une liste de coordonnées vide
```

```
void ajouter_en_fin_liste_coord(coor E, liste_coord &C);  
//ajoute en fin la coordonnée E dans C
```

- tests.h

```
bool test_vert(jeu J, collection C);  
//teste si 4 pions au moins dans la collection C sont alignés verticalement dans le jeu J à la dernière colonne  
jouée
```

```
bool test_horiz(jeu J, collection C);  
//teste si 4 pions au moins dans la collection C sont alignés horizontalement dans le jeu J à la dernière colonne  
jouée
```

```
bool test_diag1(jeu J, collection C);  
//teste si 4 pions au moins dans la collection C sont alignés dans la diagonale / dans le jeu J à la dernière colonne  
jouée
```

```
bool test_diag2(jeu J, collection C);  
//teste si 4 pions au moins dans la collection C sont alignés dans la diagonale \ dans le jeu J à la dernière colonne  
jouée
```

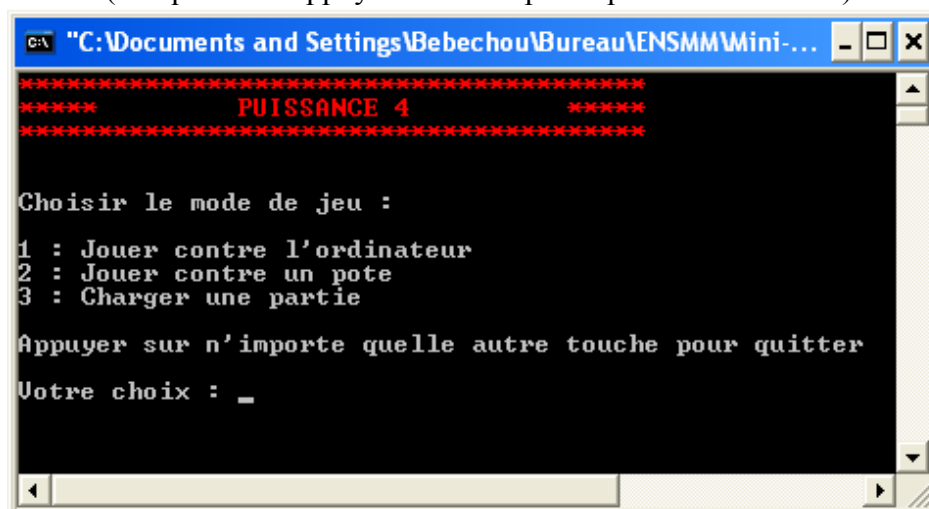
```
bool gagner(jeu J, joueur P);  
//retourne VRAI si le joueur P a gagné le jeu J, retourne FAUX sinon
```

```
bool grille_pleine(jeu J);  
//retourne VRAI si la grille est pleine, FAUX sinon
```

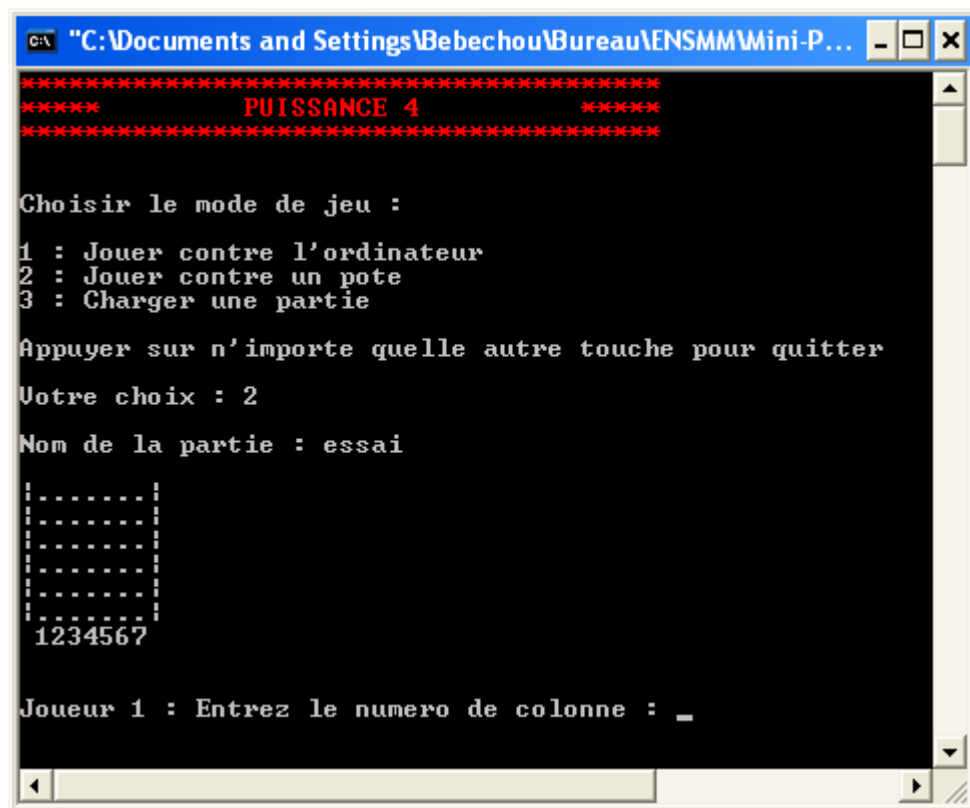
```
bool colonne_pleine(jeu J, collection C);  
//retourne VRAI si la dernière colonne jouée est pleine, FAUX sinon
```

III . Jeux d'essais :

L'écran d'accueil du jeu dans lequel vous pouvez choisir entre trois mode de jeu.
(Ou quitter en appuyant sur n'importe quelle autre touche)



Rentrez les différentes données demandées



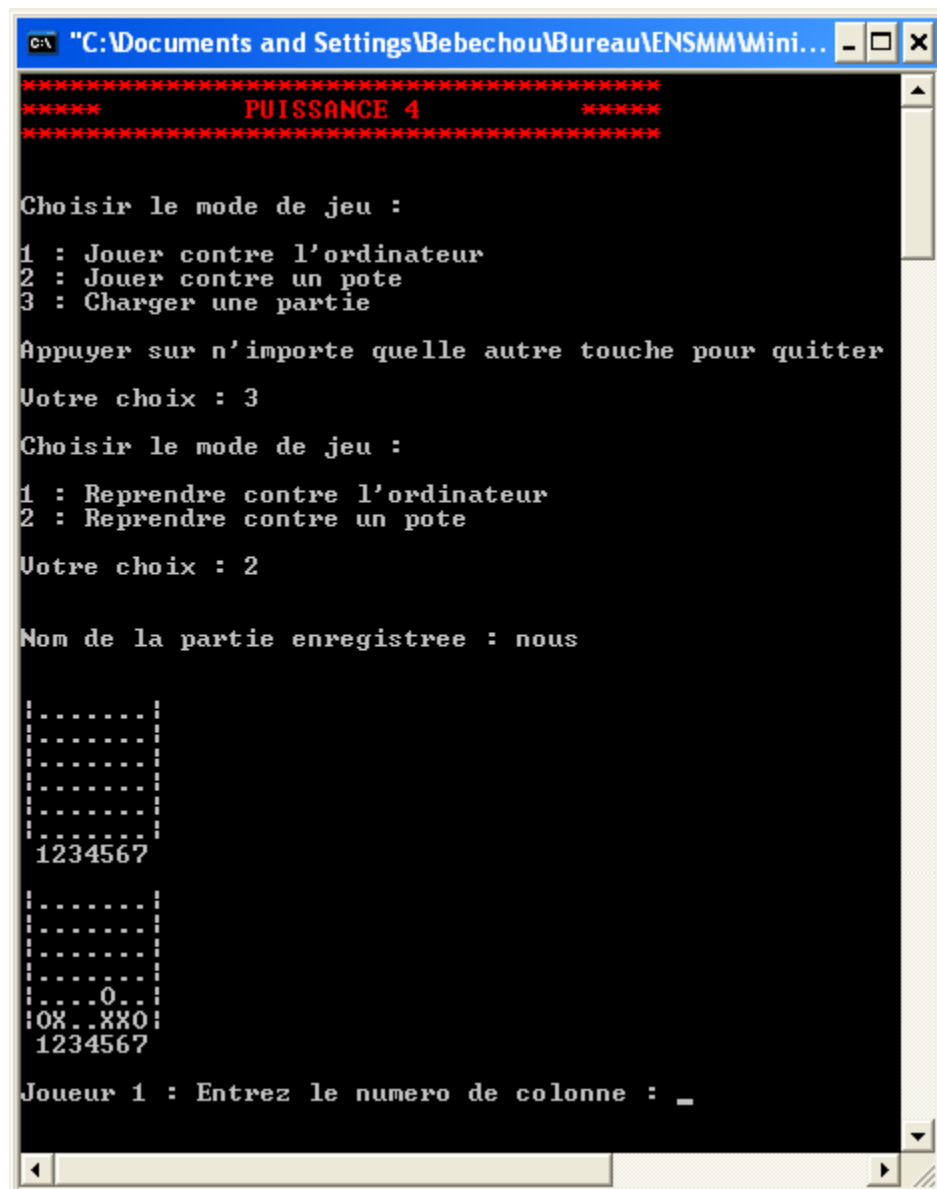
Jouez ! et essayez de gagner.


```
C:\ "C:\Documents and Settings\Bebechou\Bureau\ENSMM\Mini-P... - _ X
|-----|
|.XX.X|
|.X00X|
|.00X00X|
|1234567|
Joueur 2 : Entrez le numero de colonne : 2
|-----|
|.XX.X|
|.0X00X|
|.00X00X|
|1234567|
Joueur 1 : Entrez le numero de colonne : 6
|-----|
|.X|
|.XX.X|
|.0X00X|
|.00X00X|
|1234567|
Joueur 1 A GAGNE!!!!
```

Vous rencontrerez peut-être différents types d'erreurs :
Rejouez à nouveau, en essayant de les éviter...

```
C:\ "C:\Documents and Settings\Bebechou\Bureau\ENSMM\Mini-P... - _ X
Joueur 1 : Entrez le numero de colonne : 5
|.X|
|.0|
|.X.X|
|.0..0|
|.X0X0X|
|1234567|
Joueur 2 : Entrez le numero de colonne : 2
|.0|
|.X|
|.0|
|.X.X|
|.0..0|
|.X0X0X|
|1234567|
Joueur 1 : Entrez le numero de colonne : 2
La colonne 2 est pleine !!
Entrez le numero de colonne : 9
9 n'est pas une colonne acceptable !!
Entrez le numero de colonne :
```

Pour charger une partie, entrez 3 à l'écran d'accueil et suivez les instructions
En choisissant 1, vous reprendrez contre l'ordinateur une partie, qu'elle ait été enregistrée en
mode un joueur ou non. De même en tapant 2.



Conclusion :

- Lorsque le jeu demande le numéro de colonne, il ne faut pas entrer un caractère autre qu'un chiffre, sinon le programme s'emballe et l'erreur « 0 n'est pas une colonne acceptable ! » tourne en boucle.
- Lorsqu'on charge une partie, le programme affiche une grille vide avant la grille de jeu sauvegardée, que l'on aurait souhaité supprimer. Mais elle est nécessaire au bon affichage de la suite. (voir la dernière capture d'écran.)
- Lorsqu'on charge une partie qui n'existe pas, le programme plante.

Ensuite, nous allons donner quelques améliorations possibles :

- Faire une **vraie** intelligence artificielle. En effet, pour l'instant, l'ordinateur se contente de jouer aléatoirement.
- Rafraichir l'écran après chaque coup joué. Pour l'instant, les grilles se suivent dans la console DOS.
- Faire une interface graphique décente.
- Pouvoir sauvegarder à n'importe quel moment du jeu. Puisque le programme sauvegarde le jeu après chaque coup.

Ce projet nous a permis de nous familiariser avec certains outils informatiques vu en cours. C'est le premier programme complet que nous avons réalisé et il nous a permis de bien comprendre toutes les étapes nécessaires à l'élaboration d'un programme informatique. De plus, le fait de voir notre programme fonctionner nous a procuré une certaine satisfaction. Enfin, nous aurions pu réaliser toutes les améliorations citées ci-dessus, mais le temps nous a manqué.