Rapport sur le projet "Puissance 4"
Par Timothée Mazzucotelli
Licence Informatique L2 Semestre 3 Automne

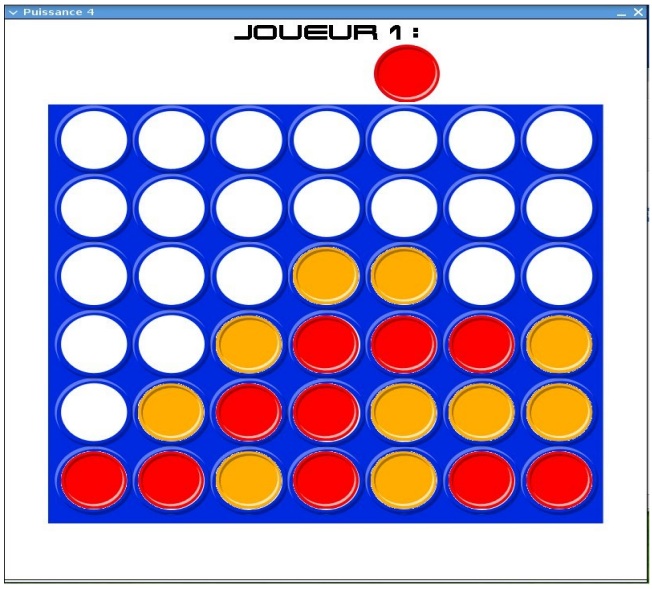
- ***INTRODUCTION*** -

- Le projet :

Projet réalisé dans le cadre des Structures de Données et Algorithmique, matière du premier semestre de la deuxième année de licence d'informatique. Le but de ce projet est de réaliser un jeu de puissance 4 en langage C, et ce en utilisant et spécifiant certaines structures de données.

- But du jeu :

Le but du jeu est d'aligner 4 pions sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une forme ou couleur particulière. En l'occurence il s'agit d'un rond pour le premier joueur, et d'une croix pour le second. Tour à tour les deux joueurs placent un pion dans la colonne de leur choix,
le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne suite à quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) d'au moins quatre pions de sa couleur. Si toutes les cases de la grille de jeu sont remplies alors qu'aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.



- Algorithmique :
-

Certaines stratégies de jeu peuvent s'appliquer, comme par exemple garantir au premier joueur de gagner à coup sûr, à condition qu'il place chacun de ses pions de manière adéquate. Pour ce faire, le premier pion doit être placé dans la colonne centrale. Le jeu de puissance 4 a été résolu de façon exacte en 1988, par James D. Allen, et indépendamment par Victor Allis à quelques jours d'intervalle, grâce à l'informatique. C'est-à-dire qu'un programme doté de la résolution exacte du jeu pourra, s'il joue en premier, s'assurer la victoire, peu importe les placements du joueur.

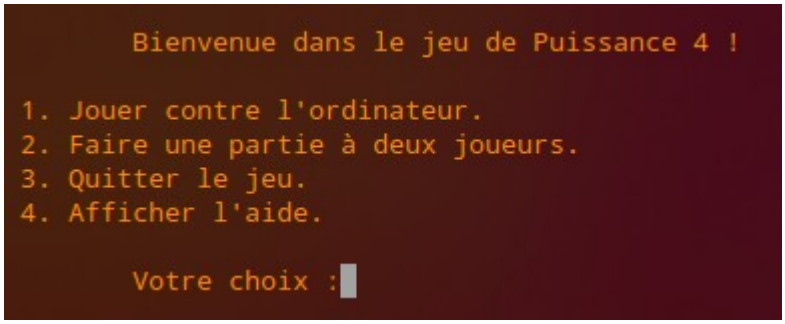
- Intelligence Artificielle :
-

Le développement de l'intelligence artificielle n'étant pas le but de ce projet, aucune résolution parfaite du jeu ne sera proposée dans ce programme. D'ailleurs, même une résolution "partielle" du jeu n'est présentement pas assurée, certains choix de l'ordinateur étant tout à fait douteux... Soyez rassurés tout de même, l'ordinateur est néanmois capable d'obtenir (heureusement !) la victoire. Ce serait tout de suite moins drôle sinon.

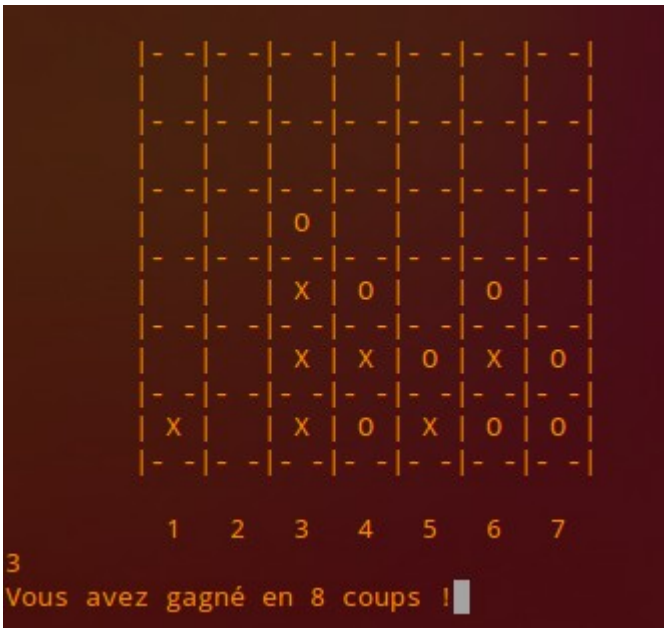
- Interface utilisateur :
-

Au niveau de l'interface, le jeu se déroulera en affichage texte, dans une fenêtre de terminal. Un menu sera proposé, afin de choisir entre afficher l'aide, faire une partie contre l'ordinateur ou jouer contre un deuxième joueur. Pour sélectionner une option, ou une colonne pendant le jeu, il faudra entrer au clavier le numéro de l'option/colonne puis appuyer sur Entrée. La partie se termine quand l'utilisateur l'interrompt, lorsque la grille de jeu est remplie ou encore si un des joueurs aligne 4 pions et gagne. Est alors indiqué le gagnant, ainsi que le nombre de coups utilisés pour gagner.

Menu :



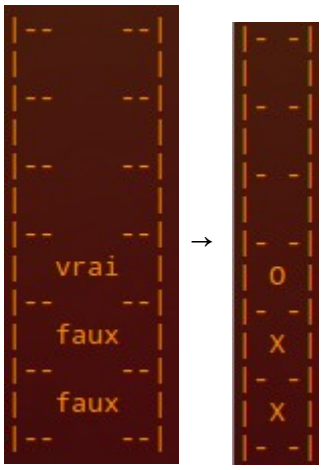
Une partie :



- Structure du jeu :
-

Le langage utilisé est le langage C. La structure principale utilisée est la pile. Pour représenter la grille de jeu, un tableau de sept piles sera donc utilisé. Ces sept piles contiendront des booléens (vrai ou faux) différenciant les pions des deux joueurs. De plus, on ajoutera à la structure de la grille une pile d'entiers, de sorte à mémoriser chaque placement de pion de la partie. Cette pile permettra de compter les pions présents dans la grille ou encore d'annuler une action, et de la répéter. "*Oh non l'ordi va gagner maintenant !! Allez, je change mon pion de colonne.*"
Le programme principal allouera les données puis débutera la récupération des appuis clavier de l'utilisateur, et ceci tant que ce dernier ne quittera pas le jeu depuis le menu.

Pile Booléens → Pile Pions



Pile Historique des étapes :

- ***PROGRAMMATION*** -

- Structuration du code :
-

Fichiers sources (.c) dans le dossier src
Fichiers d'en-tête (.h) dans le dossier include
Fichirs objets (.o) dans le dossier obj
Ficher binaire (exécutable) dans le dossier bin

Les fichiers **sources** comprennent :
-main.c -> le programme principal
-P4.c -> la gestion de la grille de jeu
-PileB.c -> la gestion des piles (colonnes)
-PileH.c -> la gestion de la pile "Historique"
-jeu.c -> la gestion des fonctions permettant de jouer
-io.c -> la gestion des entrées/sorties (appuis clavier, affichage)

Les fichiers d'**en-tête** (typedefs, structures, prototypes et doc) comprennent :
-base.h
-P4.h
-PileB.h
-PileH.h
-jeu.h
-io.h

- Compilation :
-

Le programme est compilé par gcc via la commande make utilisant le Makefile.
La documentation est générée dans le dossier doc par la commande doxygen Doxyfile.

Commandes **make**:
"make" pour compiler le projet entier
"make DEBUG=yes" pour compiler le projet avec l'option -g de gcc
"make run" pour lancer le programme
"make debug" pour lancer le programme via gdb
"make valgrind" pour lancer le programme via valgrind
"make clean" pour supprimer les objets et l'exécutable
"make doc" pour générer la documentation
"make cleandoc" pour supprimer la documentation
"make archive" pour archiver le code source et les fichiers Makefile, Doxyfile et Rapport.pdf

- Spécification des structures :
-

****
Spécification de la sorte PileB
(pile de 6 booléen)
****

spéc PileB étend BASE

Opérations :

- pilenouv : _ -> PileB // crée une nouvelle pile vide
- empiler : PileB Bool -> PileB // empile un booléen
- desempiler : PileB -> PileB // retire un booléen de la pile
- sommet : PileB -> Bool // renvoie le sommet de la pile
- kieme : PileB Nat -> Bool // renvoie le k-ième élément de la pile
  // le premier élément étant la base de la pile
- hauteur : PileB -> Nat // renvoie la hauteur de la pile
- vide : PileB -> Bool // test de vacuité

Générateurs de base :
pilenouv et empiler

Constructeurs :
pilenouv, empiler et desempiler

Observateurs :
sommet, kieme, hauteur et vide

Spécification (p: PileB, x: Bool, k: Nat) :

desempiler(pilenouv()) -> PREC1
desempiler(empiler(p,x)) = p

sommet(pilenouv()) -> PREC2
sommet(empiler(p,x)) = x

kieme(pilenouv(),k) -> PREC3
kieme(empiler(p,x),k) = PREC4 +
si k=hauteur(empiler(p,x)) alors
sommet(empiler(p,x))
sinon kieme(p,k)

hauteur(pilenouv()) = 0
hauteur(empiler(p,x)) = 1 + hauteur(p)

vide(pilenouv()) = vrai
vide(empiler(p,x)) = faux

Préconditions :

- PREC1: desempiler(p) = !vide(p)
- PREC2: sommet(p) = !vide(p)
- PREC3: kieme(p,k) = !vide(p)
- PREC4: kieme(p,k) = k>0 && k<=hauteur(p)

Préconditions supplémentaires pour l'implantation en C :

- PREC5: empiler(p,x) = hauteur(p)<6

****
Spécification de la sorte PileH
(pile statique de d'entiers (int)
servant d'historique,
taille = nbr max de coups :
6x7=42)
****

spéc PileH étend BASE

Opérations :

- hpilenouv : _ -> PileH // crée une nouvelle pile vide
- hempiler : PileH Col -> PileH // empile un short int
- hdesempiler : PileH -> PileH // décrémente la hauteur de pile
- hsynchauteur : PileH -> PileH // affecte la hauteur courante à la
  // hauteur max
- hsommet : PileH -> Col // renvoie le sommet de la pile
- hhauteur : PileH -> Nat // renvoie la hauteur de la pile
- hvide : PileH -> Bool // test de vacuité

Générateurs de base :
hpilenouv et hempiler

Constructeurs :
hpilenouv, hempiler, hdesempiler et hsynchauteur

Observateurs :
hsommet, hhauteur et hvide

Spécification (p: PileH, x: Col, k: Nat) :

hdesempiler(hpilenouv()) -> PREC1
hdesempiler(hempiler(p,x)) = p

hsynchauteur(hpilenouv()) = hpilenouv()
hsynchauteur(hempiler(p,x)) = hempiler(hsynchauteur(p),x)

hsommet(hpilenouv()) -> PREC2
hsommet(hempiler(p,x)) = x

hhauteur(hpilenouv()) = 0
hhauteur(hempiler(p,x)) = 1 + hhauteur(p)

hvide(hpilenouv()) = vrai
hvide(hempiler(p,x)) = faux

Préconditions :

- PREC1: hdesempiler(p) = !hvide(p)
- PREC2: hsommet(p) = !hvide(p)

Préconditions supplémentaires pour l'implantation en C :

- PREC3: hempiler(p,x) = hhauteur(p)<42

****
Spécification de la sorte P4
(Jeu Puissance 4, 7 PileB + 1 PileH)
****

spéc P4 étend PileB, PileH

Opérations :

- jeunouv : _ -> P4 // crée un nouveau jeu vide
- ajouterPion : P4 Nat -> P4 // ajoute un pion dans une colonne donnée
- retirerPion : P4 -> P4 // retire le dernier pion placé
- aquiletour : P4 -> Bool // indique à qui est le tour (vrai:joueur)
- colonne : P4 -> Nat // récupère l'indice de colonne du dernier pion
  // placé
- nbEtapes : P4 -> Nat // indique le nombre actuel de pions dans la
  //grille
- alignHoriz : P4 Nat -> Nat // nombre max de pions alignés
  // horizontalement incluant le pion au sommet de la colonne donnée
- alignVert : P4 Nat -> Nat // nombre max de pions alignés verticalement
  // incluant le pion au sommet de la colonne donnée

- alignDiagGauche : P4 Nat -> Nat // nombre max de pions alignés sur la //diagonale gauche incluant le pion au sommet de la colonne donnée
- alignDiagDroite : P4 Nat -> Nat // nombre max de pions alignés sur la // diagonale droite incluant le pion au sommet de la colonne donnée

Générateurs de base :
jeunouv et ajouterPion

Constructeurs :
jeunouv, ajouterPion, retirerPion

Observateurs :
aquiletour, colonne, nbEtapes, alignHoriz, alignVert, alignDiagGauche et alignDiagDroite

Spécification (j: P4, p: PileB, x: Bool, k/c: Nat) :

retirerPion(jeunouv()) = jeunouv()
retirerPion(ajouterPion(j,c)) = j

aquiletour(jeunouv()) = vrai
aquiletour(ajouterPion(j,c)) = !aquiletour(j)

colonne(jeunouv()) -> PREC1
colonne(ajouterPion(j,c)) = c

nbEtapes(jeunouv()) = 0
nbEtapes(ajouterPion(j,c)) = 1 + nbEtapes(j)

alignHoriz(jeunouv(),k) = 0
alignHoriz(ajouterPion(j,c),k) = ////////////////////////

alignVert(jeunouv(),k) = 0
alignVert(ajouterPion(j,c),k) =
si (c=k et sommet(pile(j,k)) = aquiletour(j))
alors alignVert(j,k) + 1 sinon alignVert(j,k)

alignDiagGauche(jeunouv(),k) = 0
alignDiagGauche(ajouterPion(j,c),k) = ////////////////////////

alignDiagDroite(jeunouv(),k) = 0
alignDiagDroite(ajouterPion(j,c),k) = ////////////////////////

Préconditions :

- PREC1: colonne(j) = nbEtapes(j)>0

- ***DIFFICULTES RENCONTREES*** -

La seule réelle difficulté rencontrée a été la programmation de l'intelligence artificielle... Je n'ai pas réussi à donner à l'ordinateur assez de potentiel pour jouer « intelligemment ».

- ***DIVERS*** -

La presque totalité de la programmation est de style fonctionnel. Les deux seules fonctions codées avec effets de bord sont les fonctions « unJoueur » et « deuxJoueurs ». En effet, la fonction « unJoueur » affiche des données à l'écran, mais modifie également la grille de jeu (lorsque l'ordinateur joue). Or j'avais besoin qu'elle renvoie un entier pour savoir si la boucle devait s'interrompre ou non, je ne pouvais donc conserver les changements apportés à la grille qu'en passant un pointeur de celle-ci à la fonction. De plus, le choix du type de jeu (un ou deux joueurs) modifie la valeur d'un pointeur de fonction dans le main, il fallait donc que ces deux fonctions aient le même prototype. J'ai donc également modifié la fonction « unJoueur » afin qu'elle s'accorde avec le pointeur de fonction « modeDeJeu ».