

PG108 UNIX - Langage C

Compte rendu de Projet :

PUISSANCE 4

2024-2025

LATU-FERRAND Paul

Groupe : F

Encadrant : SAUSSERAU Jonathan

Table des matières

[I. Objectifs du projet 3](#__RefHeading___Toc293_384713159)

[II. Travail réalisé 3](#__RefHeading___Toc266_384713159)

[III. Analyse et perspectives 3](#__RefHeading___Toc268_384713159)

[III.1. Estimation de la complexité 3](#__RefHeading___Toc331_384713159)

[III.2. Difficultés rencontrées 3](#__RefHeading___Toc287_384713159)

[III.3. Optimisation du code 4](#__RefHeading___Toc357_384713159)

[III.4. Propositions d’amélioration 4](#__RefHeading___Toc359_384713159)

[IV. Conclusion 4](#__RefHeading___Toc270_384713159)

# Objectifs du projet

Ce projet a pour objectif de réaliser le jeu du puissance 4 en langage C. Le jeu doit permettre de jouer contre un adversaire réel lorsque que l’on rentre le nom des deux joueurs, ou de jouer contre un ordinateur qui joue en mode aléatoire si le nom du joueur 2 est vide.

# Travail réalisé

Le premier choix réalisé concerne la mise en forme du jeu, en effet le choix a été fait d’utiliser un tableau unidimensionnel et de coder une fonction ***indice***permettant de parcourir chaque case du tableau en fonction de la rangée et de la colonne voulue. Ensuite la fonction ***init*** permet d’initialiser la partie en récupérant le nom de chaque joueur et en initialisant la grille de jeu avec des zéros partout. Après avoir initialisé le jeu il faut pouvoir afficher la grille de jeu sur le terminal, c’est la fonction ***display*** qui le permet. Selon la valeur qui est dans une case du tableau, la fonction ***display*** affiche dans la grille de jeu soit un jeton rouge, soit un jeton jaune, soit un point quand la case est vide. Il faut ensuite mettre à jour chaque colonne de jeu avec la fonction ***colonne\_de\_joueur***, qui demande au joueur via le terminal dans quelle colonne il veut jouer, puis met le jeton dans la colonne correspondante en prenant en compte les jetons qui sont potentiellement dessous.

Une fois ces fonctions permettant de mettre en place le jeu, il faut coder des fonctions pour les différentes conditions de victoires ou de fin de partie. D’abord il y a la fonction ***detecte\_plein*** qui détecte lorsque la grille de jeu est pleine et qu’il n’y a pas de vainqueur. J’ai ensuite coder 4 sous-fonctions ***Test\_ligne, Test\_colonne, Test\_diag\_pos et Test\_diag\_neg,*** qui testent à chaque tour si 4 jetons sont alignés respectivement sur une ligne, une colonne, une diagonale positive ou une diagonale négative. Lorsque 4 jetons sont alignés ces fonctions renvoient la valeur lue dans le tableau afin de savoir quel joueur a gagné. C’est la fonction ***Test\_vainqueur*** qui, en s’appuyant surles 4 sous-fonctions précédentes, réalisent les tests à chaque tour et renvoie le joueur gagnant.

Le ***main*** de ce programme est composé d’une boucle *while* qui tourne tant que la fonction

***Test\_vainqueur*** n’a pas renvoyé le joueur gagnant (ou tant qu’elle reste égale à zéro) Une fois sortie de la boucle le joueur gagnant est annoncé et le programme s’arrête.

# Analyse et perspectives

## Estimation de la complexité

Je vais déterminer la complexité temporelle de chaque fonction utilisée dans mon programme.

int indice(char r, char c)

**Meilleur cas :** O(1)

**Pire cas :** O(1)

**Moyenne :** O(1)

En effet, cette fonction effectue uniquement un calcul arithmétique, sa complexité est donc constante.

void ChangementDeJoueur()

**Meilleur cas :** O(1)

**Pire cas :** O(1)

**Moyenne :** O(1)

Il s’agit de vérifier une condition, la complexité est donc constante également.

CalculeColonneAleatoire(char tableau[NB\_rangees\*NB\_colonnes])

**Meilleur cas :** O(1)

**Pire cas :** O(6)

**Moyenne :** O(1)

Cette fonction est constituée d’une boucle avec au maximum 6 itérations, d’où la valeur de sa complexité.

int init(char J1[30], char J2[30], char tableau[NB\_rangees\*NB\_colonnes])

**Meilleur cas :** O(N)

**Pire cas :** O(N)

**Moyenne :** O(N)

La complexité temporelle est linéaire et dépend de la taille de la grille de jeu.

void display(char tableau[NB\_rangees\*NB\_colonnes])

**Meilleur cas :** O(N)

**Pire cas :** O(N)

**Moyenne :** O(N)

La complexité est linéaire également car la fonction doit parcourir chaque case de la grille.

int colonne\_de\_joueur(int JoueurEnCours, char tableau[NB\_rangees\*NB\_colonnes])

**Meilleur cas :** O(N)

**Pire cas :** O(N)

**Moyenne :** O(N)

Dans le pire cas, la boucle while parcourt l’entièreté du tableau.

int detecte\_plein(char tableau[NB\_rangees\*NB\_colonnes], int JoueurEnCours)

**Meilleur cas :** O(1)

**Pire cas :** O(N)

**Moyenne :** O(N)

Selon le moment où la grille est détectée pleine, la fonction peut parcourir toute la grille.

int TestLigne(char tableau[NB\_rangees\*NB\_colonnes])

int TestColonne(char tableau[NB\_rangees\*NB\_colonnes])

int TestDiagPos(char tableau[NB\_rangees\*NB\_colonnes])

int TestDiagNeg(char tableau[NB\_rangees\*NB\_colonnes])

**Meilleur cas :** O(N)

**Pire cas :** O(N)

**Moyenne :** O(N)

Ces 4 fonctions ont la même complexité puisqu’elles parcourent la grille entière afin de trouver 4 jetons alignés, leur complexité est donc linéaire.

int TestVainqueur(char tableau[NB\_rangees\*NB\_colonnes])

**Meilleur cas :** O(N)

**Pire cas :** O(N)

**Moyenne :** O(N)

Comme les 4 fonctions précédentes ont une complexité linéaire, cette fonction a également une complexité linéaire.

int main()

Pour cette fonction, la complexité dépend de celle de chaque fonction précédemment explicitée, mais dans le pire cas possible, la complexité est en O(N2).

## Difficultés rencontrées

L’étape qui m’a posé le plus de difficulté a été la réalisation de la fonction***Test\_vainqueur*** et des 4 sous-fonctions associées. En effet il fallait vérifier tous les cas possibles et éliminer ceux qui n’étaient pas réalisables, notamment pour les tests sur les diagonales. Il m’a fallu un peu de temps pour déterminer les différents cas possibles puis pour coder les différentes vérifications de chaque hypothèse.

## Optimisation du code

Le sujet étant guidé, chaque fonction nécessaire au fonctionnement du programme est séparée des autres, il est donc assez facile de se retrouver dans la structure générale de ce programme. Concernant le temps d’exécution, certaines actions auraient surement pu être réaliser dans une même fonction mais cela n’a pas d’impact très important sur la rapidité perçue par les joueurs.

## Propositions d’amélioration

Avec plus de temps, la principale amélioration serait une IA efficace pour que le jeu face à l’ordinateur soit plus intéressant. L’amélioration de l’affichage pourrait être aussi pertinent même si j’ai déjà essayé de faire quelques changements avec des jetons de couleurs et la séparation de chaque case.

# Conclusion

Finalement, j’ai pu coder un jeu de puissance 4 fonctionnel, qui peut être joué soit face à un adversaire réel, soit face à l’ordinateur quoi joue des colonnes aléatoires. Cette version de puissance 4 pourrait être amélioré surtout si l’on choisit de jouer contre l’ordinateur avec une intelligence artificielle, et surement d’un point de vue esthétique avec une vraie grille de jeu.