**PROJET IFB**

**Printemps - 2021**

# PUISSANCE N


**LES MEMBRES DU GROUPE :**

Céliane ALLAIRE / Loïc BOURIEL / Mathis BURGER

# SOMMAIRE

# INTRODUCTION

## Composition de notre équipe

Dès que nous avons appris que nous devions réaliser un projet en groupe, nous avons tout de suite eu à l'esprit que la composition des membres de notre groupe allait être primordiale. En effet, nous avions déjà été confrontés à des travaux de groupe, et nous savions donc toute l'importance quant à l'élaboration de ce dernier.

Peu après l'annonce des sujets, nous nous sommes directement mis ensemble pour choisir le sujet, préparer le projet et commencer à réfléchir… Nous formons donc un trinôme issu du même groupe et du même demi-groupe de TD/TP. Effectivement, ce projet a demandé un temps de travail et de réflexion considérable, c'est pourquoi il était nécessaire d'avoir des emplois du temps concordants pour pouvoir programmer des séances de travail communes à nous 3. Bien que les dernières séances de TD/TP furent consacrées au projet, nous avons beaucoup travaillé ce projet chacun chez nous et en commun dans les locaux de l'UTBM.

## Choix du sujet

Nous avons beaucoup hésité entre les deux projets proposés : « Puissance N » et « Bataille navale ». Instinctivement, nous voulions choisir le projet qui nous parlait le plus et qui plaisait à tous les membres du groupe. Mais aussi et surtout, celui dont nous imaginions déjà les différentes fonctions et la manière dont nous allions nous y prendre.

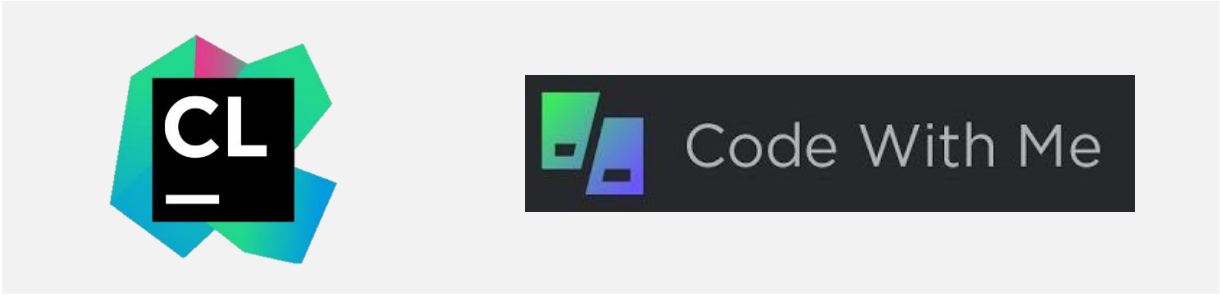
Nous avons pris le temps de bien lire les documents supports des deux projets et de noter ensemble les différentes idées qui nous venaient en tête. Nous avons donc préféré utiliser plus de temps de réflexion algorithmique, avant de se lancer définitivement dans un sujet ou l'autre. Cela nous a permis d'éviter de rester bloqués trop longtemps sur un point en particulier, et donc gagner du temps par la suite.

À la suite de longues conversations, nous avons choisi de réaliser le projet « Puissance N ». Comme évoqué plus haut, nous avions déjà quelques idées sur la structure de certaines fonctions et la manière d'aborder le code lors de la programmation.

Nous voilà maintenant lancés dans le projet : « Puissance N » !

## Les préparations avant le début de la programmation

Avant de nous lancer aveuglément dans la programmation, nous avons pris beaucoup de temps à « préparer le terrain ». Nous avons dû tout d'abord choisir sur quel logiciel nous allions programmer ensemble. Au départ, comme recommandé, nous voulions nous lancer sur GitHub. Cependant, le peu de connaissances que nous avions sur le fonctionnement de GitHub et l'apparence complexe du logiciel nous a (malheureusement) vite découragé. Nous avons donc opté pour l'utilisation de CLion. En effet, tous les 3 très à l'aise sur cet environnement de développement, nous avons découvert l'option « Code With Me », qui permet de coder à plusieurs et simultanément un même projet. Après quelques essais sur le présent plugin, notre groupe était d'accord de réaliser ce projet de la sorte.




En général, cela fonctionnait plutôt bien et cela nous a permis de travailler ensemble, à distance et en présentiel. Le seul vrai inconvénient, était que pour que tout le monde travaille sur notre programme, il fallait que la personne qui héberge notre projet soit connectée. Ceci n'était évidemment pas tout le temps possible. Mais, chaque membre du groupe a bien su s'adapter. Par exemple, un membre du groupe s'occupait d'une partie du code sur son ordinateur personnel et sur un projet brouillon indépendant et le mettait ensuite en commun sur le projet officiel, dès que cela était possible (inutile de préciser que le travail était beaucoup plus efficace en présentiel). De plus, nous avons trouvé une manière astucieuse de synchroniser nos lignes de code (main, les fichiers .c et .h) dans un fichier collaboratif sur Google Drive. Ceci permettait donc à tous les membres du groupe d'avoir accès au code déjà écrit, dès qu'on le voulait.

Une fois l'environnement de développement et notre cadre de travail clairement défini, nous avons commencé à discuter des différentes fonctions que nous voulions écrire et de la répartition des tâches sur l'ensemble du projet. L'utilisation de fonctions nous permettait de mieux diviser le travail. Concrètement, l'un pouvait rédiger l'algorithme du programme de la fonction add_token, un autre travaillait sur la fonction check_winner, pendant que le dernier s'occupait du rendu graphique dans l'interface console, de la clarté globale du code et de ce présent rapport. De cette manière, tout le groupe pouvait travailler simultanément, sans avoir à s'inquiéter de modifier ou gêner autrui. On a été une équipe soudée.

# RÈGLES DU JEU

Notre jeu « Puissance N » est inspiré du mythique jeu de société « Puissance 4 ».

Une partie de puissance N se déroule de la façon suivante :

- L'ordinateur demande au joueur, combien de jetons (N) devront-être alignés pour gagner. La grille de jeu sera alors un carré de côté N + 2. (À noter que dans la version originelle du jeu, il s'agit d'un rectangle de 6x7)
- Ensuite, le joueur qui aura l'initiative (jaune ou rouge) est déterminé par un tirage au sort
- Le joueur tiré au sort choisit le type de jetons qu'il va jouer
- En début de manche, le joueur qui commence doit placer un jeton de sa couleur, dans l'une des N + 2 colonnes. Le premier jeton tombe tout en bas de la grille de jeu, ou prend place juste au-dessus du dernier jeton inséré dans cette colonne
- Le second joueur place alors un de ses jetons dans l'une des N + 2 colonnes
- La partie se termine quand l'un des joueurs a aligné N jetons horizontalement, verticalement, ou en diagonale (ascendante ou descendante) ou quand l'utilisateur choisit de sauvegarder la partie en cours

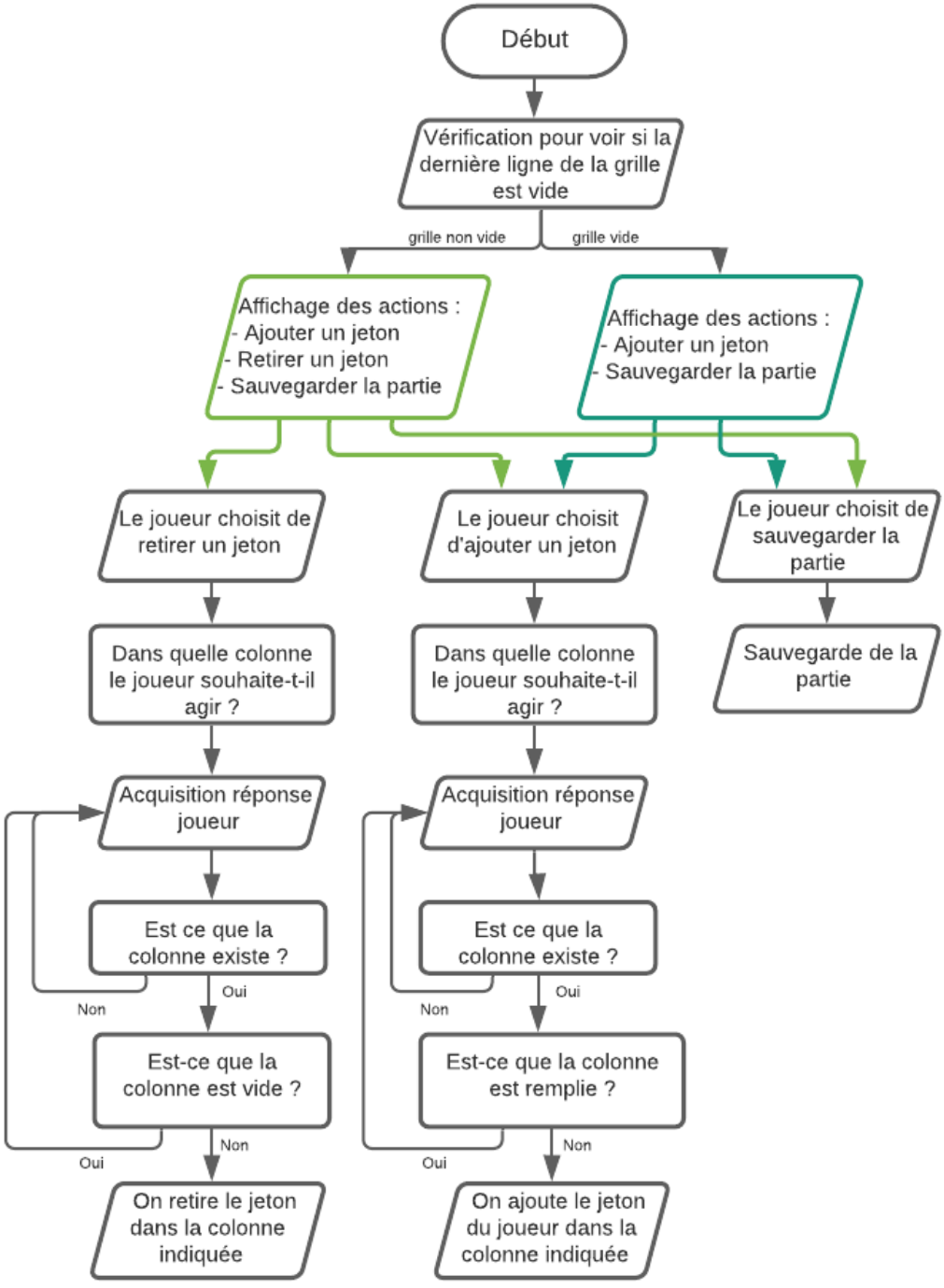
**NB :**

Une règle supplémentaire est rajoutée par rapport au jeu classique : lors de son tour, un joueur peut décider de ne pas poser de jeton, mais au contraire d'en retirer un. Lorsqu'un jeton vient d'être retiré, l'autre joueur ne peut reposer de jetons dans cette colonne pendant sa prochaine action !

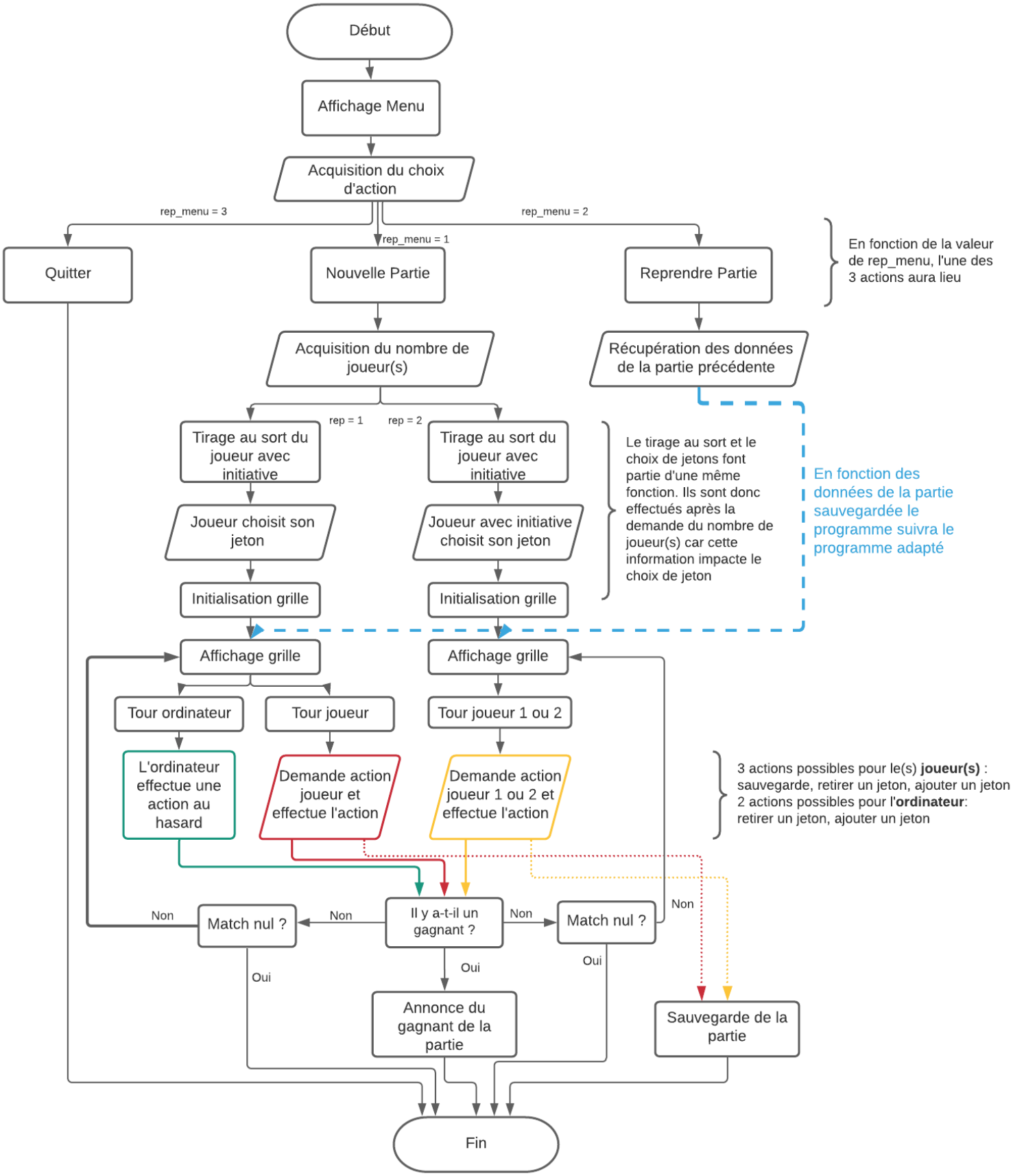
# ALGORIGRAMMES

Pour bien préparer notre projet et avoir un plan avant de nous lancer dans la programmation, nous avons fait des algorigrammes rapidement au brouillon. Nous les avons remis au propre pour pouvoir les présenter ci-dessous et donner un aperçu d'une autre partie de notre processus de travail.

## Algorigramme n°1 : Fonctionnement Ajout / Retrait jeton



# Algorigramme n°2 : Déroulement d'une partie



# STRUCTURE GENERALE DE NOTRE CODE

## La modularité de notre code

A la vue du projet, nous avons divisé le code source en plusieurs fichiers .c et .h afin d'ordonner et de clarifier le code. Les fonctions sont rassemblées selon leurs thématiques (gestion de la grille, choix de l'ordi, vérification du gagnant…) et son listées ci-dessous :

- gestion_grille.h : gère la grille (initialisation, affichage, sauvegarde, couleurs…)
- ajout_retrait_jetons.h : gère les jetons (ajout, retrait, grille vide/pleine…)
- jeu.h : assure les parties en solo/multi-joueurs
- ordinateur.h : gère l'IA de l'ordinateur
- check_winner.h : vérifie la victoire des joueurs

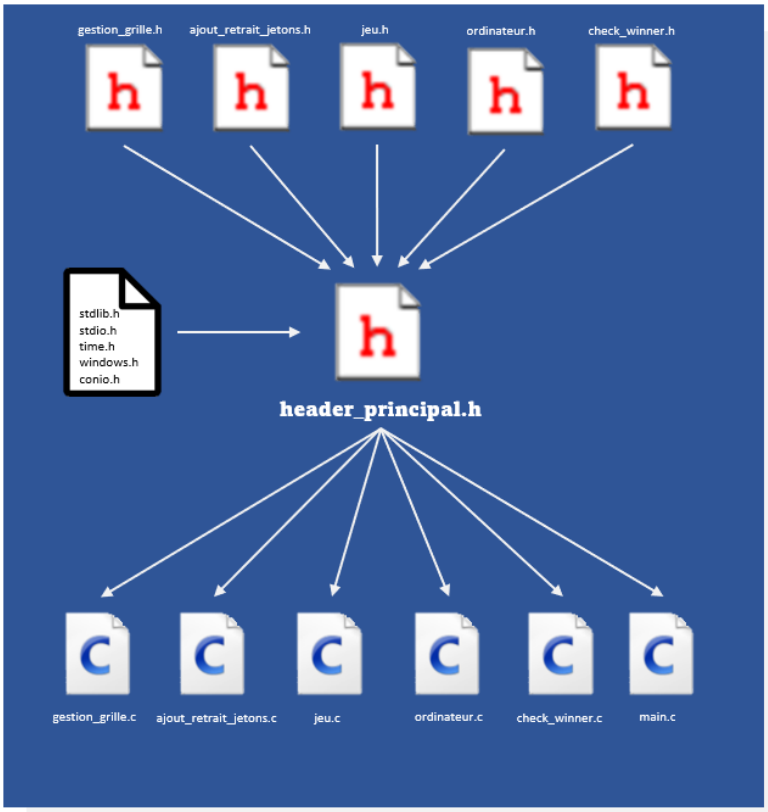
En procédant de la sorte, la maintenance du code et la résolution de problèmes sont facilitées.

## Le header principal

Afin d'inclure dans nos différents fichiers : nos structures, nos déclarations de fonctions et les bibliothèques standards utiles à notre programme, nous avons créé un fichier .h intitulé « header_principal.h ». Chaque fichier .h, associé à un fichier .c, y est donc inclus.

Ce fichier .h est alors inclus dans chaque fichier .c. Cela permet d'assurer les inclusions et donc la compilation des différents fichiers. En effet, cela nous permet alors d'inclure seulement ce fichier header principal dans tous nos fichiers .c. Notre programme fonctionne donc normalement pour les joueurs, mais représente une amélioration d'ergonomie pour gérer notre code source.

Cependant, nous sommes conscients que cette manipulation n'est pas forcément la meilleure car elle inclut des bibliothèques/headers inutiles à certains fichiers. Cela peut donc perturber la compilation à certains moments. Ce n'est donc pas la façon la plus optimale car des portions de code sont incluses alors qu'elles ne seront jamais utilisées, ce qui augmente de façon non négligeable la taille de l'exécutable du programme. Mais, cela reste bien pratique et permet d'avoir la même déclaration dans chacun de nos fichiers. En effet, nous gagnons du temps au niveau des vérifications de nos inclusions dans chacun de nos fichiers.




*Fonctionnement du header_principal.h*

## Le main.c

Notre fichier main.c est le cœur de notre programme. Il fait appel aux différentes fonctions utiles au bon déroulement du jeu. Dans un premier temps, il lance le menu principal et agit selon le choix de l'utilisateur, à savoir : lancer une nouvelle partie, charger une partie sauvegardée, quitter le programme.

Après avoir débuté une partie, et seulement à la fin soit après une victoire, soit après une sauvegarde, le programme libère l'espace mémoire alloué à la grille de jeu.

## Le show_grid

La fonction show_grid occupe une place très importante dans notre programme. En effet, elle gère l'affichage de la grille de jeu. C'est ce qui rend notre jeu agréable et interactif pour les joueurs. Ce n'est que de l'affichage, mais c'est pour que les joueurs aient un bon visuel sur leur partie en cours.

Concrètement, pour la formation de la grille, nous avons utilisé des caractères présents dans la table ASCII, de telle sorte à créer une grille de jeu qui ressemble à celle d'un réel Puissance 4.

## Nos structures personnalisées

### ➢ Structure Grid

```
typedef struct {
    char** grille;
    int largeur;
    int longueur;
}Grid;
```

Nous avons créé la structure « Grid ». Elle est composée d'un tableau de caractères à 2 dimensions qui représente la grille de jeu, d'un entier représentant sa largeur, d'un entier représentant sa longueur.

Modifier cette structure s'effectue simplement en passant en paramètre de fonctions un pointeur sur celle-ci.

### ➢ Structure Joueur

```
typedef struct {
    char couleur;
    int initiative;
    char jeton;
}Joueur;
```

Nous avons créé la structure « Joueur ». Elle permet de caractériser les différents joueurs de la partie.
Elle est composée d'un caractère représentant sa couleur, et d'un entier correspondant à l'initiative, et d'un caractère qui représente le jeton.

### ➢ Structure Origine

```
typedef struct {
    int abscisse;
    int ordonnee;
}Origine;
```

Nous avons créé la structure « Origine ». Elle permet de caractériser le point d'origine de la diagonale (utilisée dans check_winner) dans la grille de jeu.
Elle est composée d'un entier représentant son abscisse, et d'un entier représentant son ordonnée.

# CHOIX REALISES POUR LE DEVELOPPEMENT DU JEU

## Améliorations apportées

Remarque générale : Arbitrairement, dans notre jeu solo contre l'ordi, nous attribuons automatiquement la couleur jaune à l'ordinateur.

### ➢ Choix du jeton

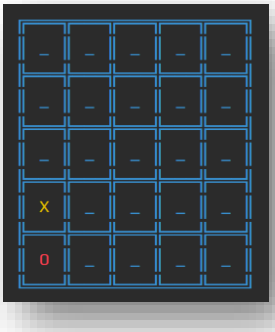
Nous avons proposé une fonctionnalité qui n'était pas exigée dans les consignes (après accord de notre professeur, nous considérons donc ceci comme une amélioration), il s'agit du fait que nous laissons le choix du jeton au joueur ayant l'initiative.

Pour maintenir une certaine logique avec les choix effectués par les joueurs, nous avons aussi pris le temps de faire correspondre la couleur du joueur avec la couleur de son jeton. En effet, si le joueur jaune a l'initiative et qu'il choisit les jetons « X », ces derniers apparaîtront en jaune dans la grille et non en rouge comme demandé dans l'énoncé.

*On demande au joueur ayant l'initiative, quel type de jeton il souhaite utiliser dans sa prochaine partie :*

```
Jeton joueur ROUGE (X ou O) : O
Jeton joueur JAUNE : X
```

*Aperçu de notre grille de jeu avec les couleurs des jetons inversées, comme l'a choisi le joueur :*



### ➢ Couleurs

Pour améliorer l'expérience de jeu de nos adhérents, nous avons ajouté des couleurs et du surlignage pour ajouter du relief à notre zone console. Les interactions avec le joueur sont donc plus sympathiques (messages de félicitations dans la couleur du joueur qui a gagné, messages d'erreurs en rouge...).

Pour pouvoir accéder à cette palette de couleurs, nous avons ajouté la bibliothèque <windows.h> et nous avons utilisés une fonction récupérée en ligne. Le site où nous avons trouvé toutes les informations sur l'utilisation de cette fonction couleur est cité plus bas dans nos sources.

## La propreté du code

### ➢ L'indentation

Nous n'allons pas nous étendre sur ce sujet, car c'est presque la base de tout bon code. Les retraits et indentations permettent de voir clairement les différents blocs d'instructions dans une fonction et les boucles. C'est un élément essentiel pour bien comprendre l'algorithme.

Dans notre projet, nous avons utilisé un style d'indentation similaire au modèle classique K&R. Contrairement au modèle K&R, nous plaçons nos accolades ouvrantes sur la même ligne que la déclaration.

*Extrait de notre programme présentant notre manière d'indenter et d'aérer notre code :*

```
if (joueur_jaune.initiative == 0) {
    color(14, 0);
    printf("\nTOUR JOUEUR JAUNE:");
    color(15,0);
} else {
```

## ➢ L'aération

Nous avons choisi de bien aérer notre code. En effet, après les conseils de nos professeurs et d'anciens élèves, il nous a été fortement recommandé d'aérer notre code pour faciliter sa lisibilité. Nous avons ainsi suivi les règles typographiques françaises classiques :

- Un espace après les virgules (et points-virgules hors fin de ligne)
- Un espace avant et après un opérateur binaire (+ - * /), sauf dans certains cas où ne pas les mettre peut aider la lisibilité
- Un espace après les mots-clés : for (, if (, else {

En effet, nous avions déjà pris cette bonne habitude dès le début du semestre, cela ne nous a donc pas vraiment perturbé, et cela rend l'aspect global du code vraiment moins repoussant.

Les retours à la ligne peuvent eux aussi, aider à distinguer efficacement différents blocs dans une même fonction. Nous avons donc effectué des retours à la ligne pour ainsi distinguer différentes sections du code.

## ➢ Le nommage des fonctions

L'ensemble du groupe s'est mis d'accord sur un seul type de nommage de fonction. En effet, sur un projet (relativement) conséquent comme celui-ci, il est presque obligatoire d'adopter un même type de nommage, afin d'appliquer une certaine cohérence générale. Nous avons choisi d'utiliser le type : snake_case.

Toutes nos fonctions ont un nom explicite et français. Seules les fonctions imposées sont en anglais.

# CONCLUSION

## Résultat final

Finalement, notre projet propose une expérience de jeu fonctionnelle et agréable. En effet, le jeu est facile à prendre en main, les messages sont cohérents et ludiques, il y a la présence de couleurs, une grille de jeu pratique et intuitive, ne comporte pas de bugs détectés à ce jour, et le code est facile à maintenir puisqu'il est dûment commenté et aéré. (Oui, nous sommes tout proches de le vendre aux plus grands éditeurs de jeux-vidéo !)

## Bilan cahier des charges

Nous nous sommes appliqués à respecter l'entièreté des consignes présentées. Toutes les fonctions imposées ont été correctement écrites. Nous avons ajouté beaucoup d'éléments d'affichage et d'interactions avec les joueurs : comme des couleurs, une grille de jeu intuitive, des félicitations et bien d'autres. Nous avons tout mis en œuvre pour améliorer l'expérience de jeu.

Nous avons bien sûr respecté les règles du jeu officiel du Puissance 4 (car c'est quand même de là d'où découle le projet), hormis la règle du retrait de jeton qui n'est pas présente normalement.

Nous avons répondu à la proposition d'intelligence artificielle optionnelle « Ordi ». Nous avons même rendu l'expérience de jeu plus offensive et ouverte, en augmentant les probabilités de placer un jeton plutôt que d'en retirer un.

## Améliorations envisageables

Après avoir présenté et fait le point sur tout ce qui a été effectivement réalisé et réussi, il est temps de voir ce que l'on aurait pu envisager comme potentielles améliorations à notre code.

Premièrement, nous aurions pu coder une intelligence artificielle avec plusieurs niveaux de difficulté. En effet, nous avons coder une intelligence artificielle, qui n'est pas strictement aléatoire et qui a plus de probabilité d'ajouter un jeton que d'en enlever un, pour essayer d'équilibrer l'expérience de jeu. Si l'ordinateur retire trop régulièrement des jetons, la partie n'avance pas.

Cependant, cette I.A. est directement dépendante de la fonction rand(). En conséquence, le jeu n'est pas réellement réfléchi comme contre un joueur physique, car ses actions sont purement aléatoires.

Nous aurions aussi pu ajouter un mode de jeu « Impossible », avec une Intelligence Artificielle qui ne fait que retirer les jetons de son adversaire. Dans cette situation-là, les joueurs ne pourraient pas aligner N jetons, et la partie serait alors sans fin.

Deuxièmement, pour rendre l'expérience de jeu encore plus plaisante, nous aurions pu envisager de créer une interface graphique, plutôt que d'utiliser la sortie console. Malheureusement, nous n'avons pas eu assez de temps pour nous lancer concrètement dans la création d'une belle interface graphique.

Troisièmement, nous aurions pu attribuer des pseudos aux joueurs, pour ensuite créer un tableau récapitulatif des statistiques. On aurait pu retrouver par exemple : le nombre de parties jouées, le nombre de parties gagnées par joueur, le nombre de jetons placés, le temps en partie, …

Quatrièmement, nous avons envisagé (sur la toute fin du projet, qui n'a pas pu aboutir : faute de temps) de proposer aux joueurs de lancer une nouvelle partie suite à la fin de la partie précédente. De la sorte, les joueurs pourraient enchaîner les parties sans avoir à relancer le programme à chaque fin de partie.

Enfin, il aurait été possible d'améliorer la vitesse d'exécution de notre programme en optimisant certaines boucles et conditions, ou en enlevant des éléments inutiles. Ce problème ne nous concerne pas réellement puisque notre programme se lance rapidement et ne présente pas de problème. Cela aurait été une bonne démarche à suivre si le projet avait été encore plus important. Par exemple, simplifier les fonctions utiles à « check_winner » fluidifierait la compilation.

## Remarques et remerciements

En conclusion, nous voulons préciser que nous avons pris beaucoup de plaisir à réaliser ce projet. C'était une très bonne manière de nous inciter à nous améliorer en programmation en langage C. De plus, le format du projet, prenant la forme d'un jeu, a rendu la tâche ludique.

Enfin, nous tenons à remercier l'ensemble de l'équipe pédagogique encadrante, qui nous a permis de réaliser ce projet dans le cadre de l'UV IFB2, et qui a toujours bien répondu à nos questions.

# SOURCES

## Généralités

- Astuces et conseils généraux pour la programmation : https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c

## Couleurs

- Couleurs d'affichage (bibliothèque <windows.h> requise) : https://www.developpez.net/forums/d309614/c-cpp/c/couleurs-c/

## Trophée des Champions

- Trophée en ASCII Art : https://ascii.co.uk/art/trophy

## Fichiers

- Gestion des fichiers : https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/16421-lire-et-ecrire-dans-des-fichiers

## Documentation

- Exportation de la documentation du code : https://franckh.developpez.com/tutoriels/outils/doxygen/#LIII-D

## Page de garde

- Image de Puissance 4 mise en illustration : https://www.asdirect.fr/jeux-de-patience-publicitaire/jeu-puissance-4-personnalise-jsps40-10413.php

# LIEN UTILE

Lien : http://212.195.66.71:5000/sharing/vkXsNINMy

Mot de passe : Céliane_Loic_Mathis_P21