Université Côte d'Azur

Faculté des Sciences

Rapport
Projet : Puissance 4

Projet réalisé par :
BENCHECI Valentin

Année universitaire :
2020 / 2021

SOMMAIRE

1. CONTEXTE

Puissance 4 (appelé aussi parfois **4 en ligne**) est un jeu de stratégie combinatoire abstrait, commercialisé pour la première fois en 1974 par la Milton Bradley Company, plus connue sous le nom de MB et détenue depuis 1984 par la société Hasbro.

Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 7 rangées et 7 colonnes. 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 la dite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle (où égale).

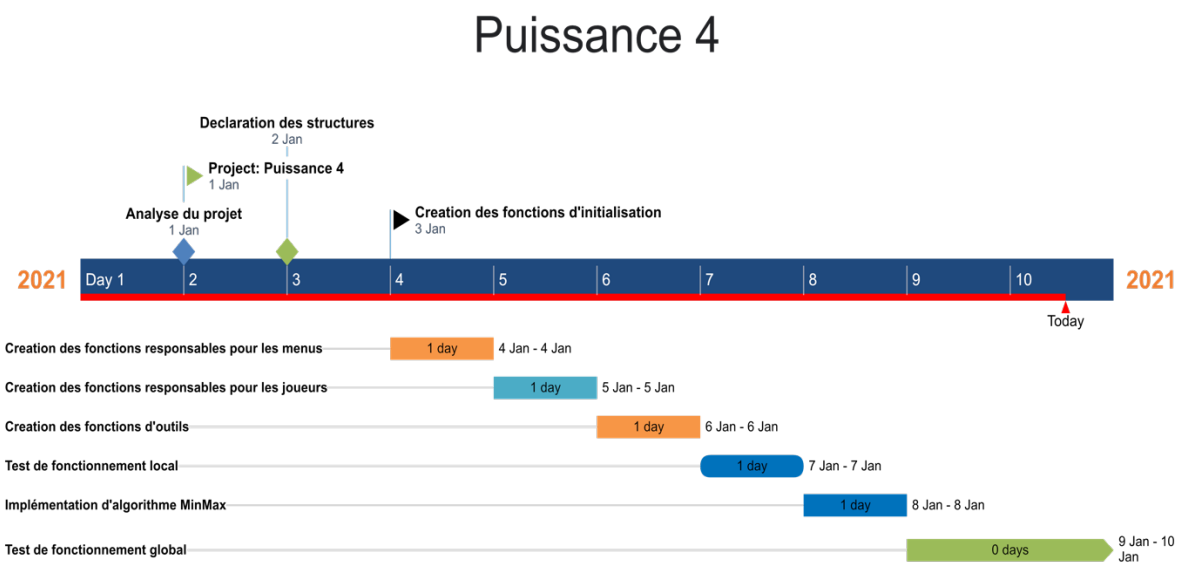
Par rapport au jeu traditionnel, le projet réalisé permet en plus de :

- Tourner le plateau de 90 degrés sur la gauche ;
- Tourner le plateau de 90 degrés sur la droite ;
- Retourner le plateau.

2. REPARTITION DES TACHES ET DES TECHNOLOGIES

Diagramme GANTT

Le diagramme de Gantt, couramment utilisé en gestion de projet, est l'un des outils les plus efficaces pour représenter visuellement l'état d'avancement des différentes activités (tâches) qui constituent un projet. La colonne de gauche du diagramme énumère toutes les tâches à effectuer, tandis que la ligne d'en-tête représente les unités de temps les plus adaptées au projet (jours, semaines, mois etc.). Chaque tâche est matérialisée par une barre horizontale, dont la position et la longueur représentent la date de début, la durée et la date de fin.



3. LES SOLUTIONS LOGICIELLES

Langage C

Le C est un langage impératif, généraliste, issu de la programmation système. Inventé au début des années 1970 pour réécrire UNIX, il est devenu un des langages les plus utilisés.

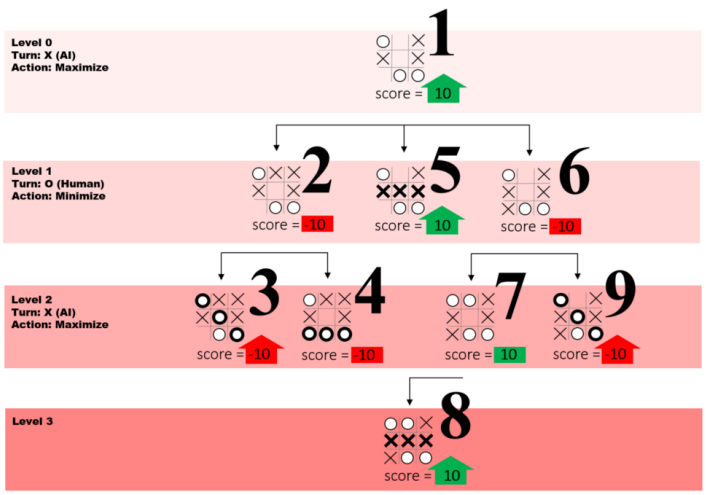
La langue C a été utilisée à 100% dans la création de ce projet. Prouvant ainsi qu'il s'agit d'un langage universel, utilisable dans tous les domaines.

L’algorithme MinMax

L'**algorithme minimax** est un algorithme qui s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle (et à information complète) consistant à minimiser la perte maximum (c'est-à-dire dans le pire des cas). Pour une vaste famille de jeux, le théorème du minimax de von Neumann assure l'existence d'un tel algorithme, même si dans la pratique il n'est souvent guère aisé de le trouver.

Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser (le jeu est à somme nulle).

L’algorithme en action :



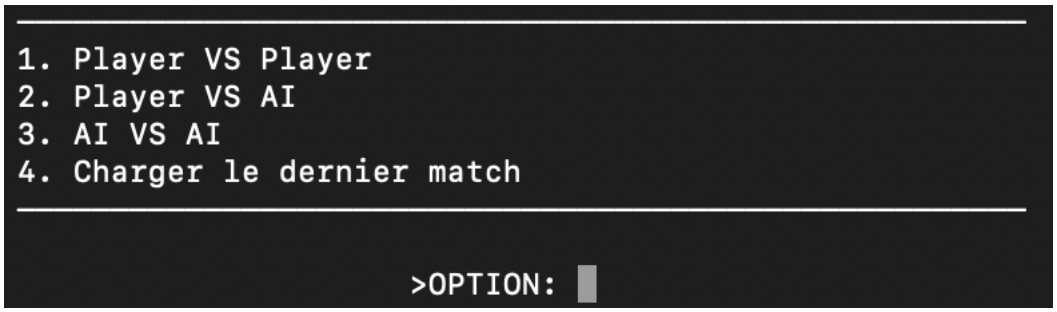
1. L'algorithme est servi par origBoard et aiPlayer. Il dresse une liste de trois cellules vides trouvées, vérifie la finitude de l'état et parcourt toutes les cellules vides. L'algorithme change ensuite newBoard, plaçant aiPlayer dans la première cellule vide. Ensuite, il s'appelle de newBoard et huPlayer et attend que le deuxième appel renvoie une valeur.
2. Pendant que le premier appel de fonction est toujours en cours d'exécution, le second est lancé, créant une liste de deux cellules vides, vérifiant que l'état est fini et bouclant toutes les cellules vides. Ensuite, le deuxième appel modifie newBoard, plaçant huPlayer dans la première cellule vide. Après cela, il s'appelle de newBoard et aiPlayer et attend que le troisième appel renvoie une valeur.
3. L'algorithme compile une liste de cellules vides et corrige la victoire du joueur après avoir vérifié la finitude de l'état. Par conséquent, il renvoie un objet avec un champ de facture (-10).
4. L'algorithme compile une liste de cellules vides et corrige la victoire du joueur après avoir vérifié la finitude de l'état. Par conséquent, il renvoie un objet avec un champ de facture (-10).
5. Dans le cinquième appel de fonction, l'algorithme compile une liste de cellules vides et corrige la victoire de l'IA après avoir vérifié la finitude de l'état. Par conséquent, il renvoie un objet avec un champ de score de +10.
6. Le sixième appel fait une liste de deux cellules vides, vérifie la fin de l'état et boucle sur toutes les cellules vides. Il modifie ensuite newBoard pour placer huPlayer dans la première cellule vide. Ensuite, il s'appelle de newBoard et aiPlayer et attend que le septième appel renvoie une valeur.
7. Le nouvel appel compose une liste d'une cellule vide, vérifie si l'état est fini et change newBoard en plaçant aiPlayer dans une cellule vide. Après cela, il s'appelle de newBoard et huPlayer et attend que cet appel renvoie une valeur.
8. Le huitième appel fait une liste vide de cellules vides et corrige la victoire de aiPlayer après avoir vérifié la finitude de l'état. Par conséquent, il renvoie un objet avec un champ de score de (+10) un niveau supérieur, le septième appel.
9. Après cela, l'algorithme compile une liste de cellules vides et corrige la victoire de aiPlayer après avoir vérifié la finitude de l'état. Par conséquent, il renvoie un objet avec un champ de score de (+10) un niveau supérieur.

4. IMPLÉMENTATION

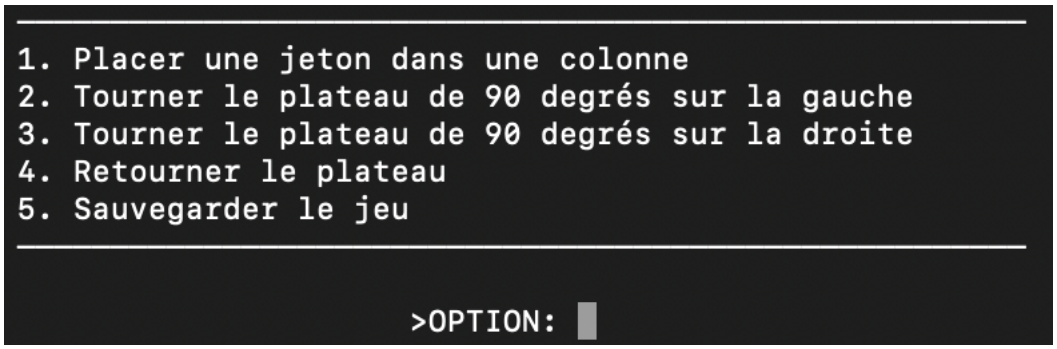
a. Les menus

- **mainMenu**

- **gameTypeMenu**



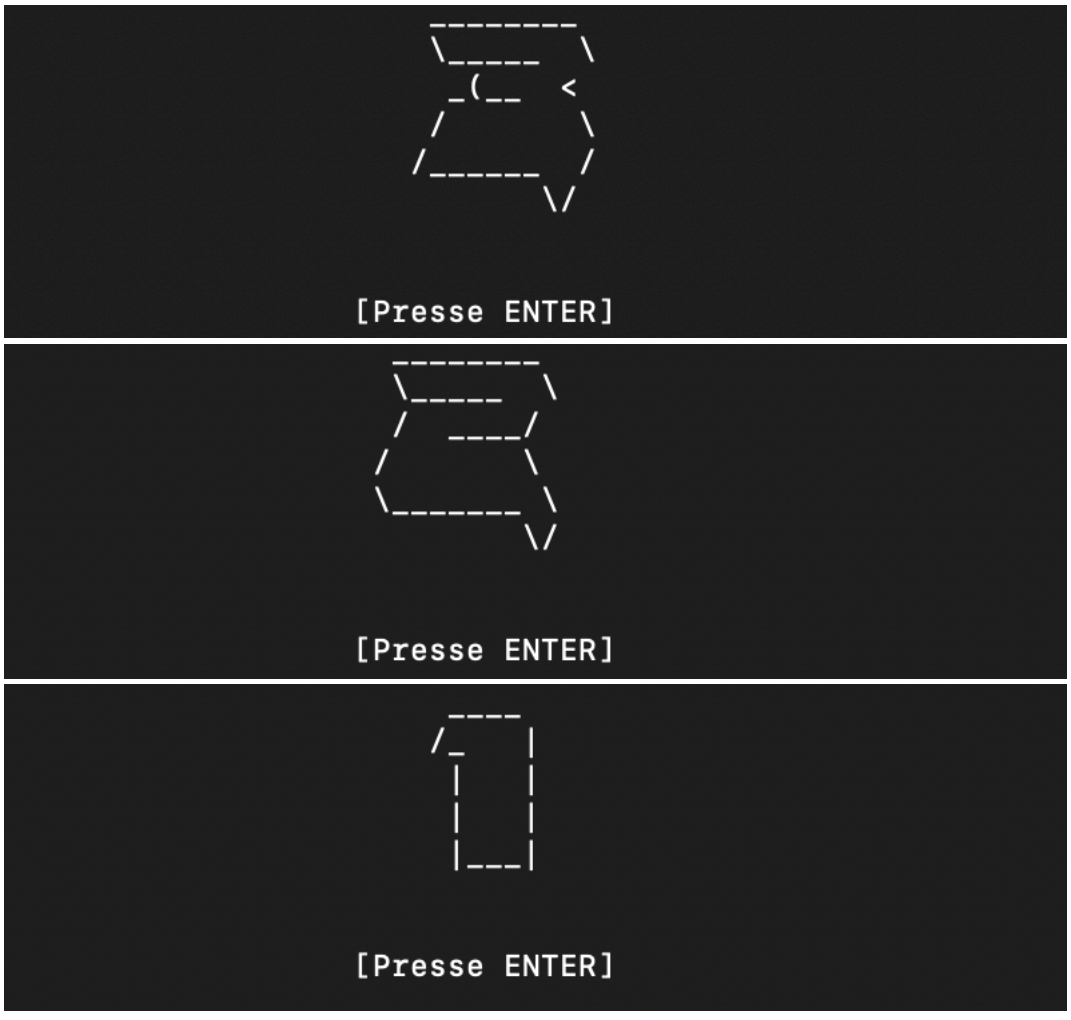
- **gameMoveMenu**



b. Les particularités du projet

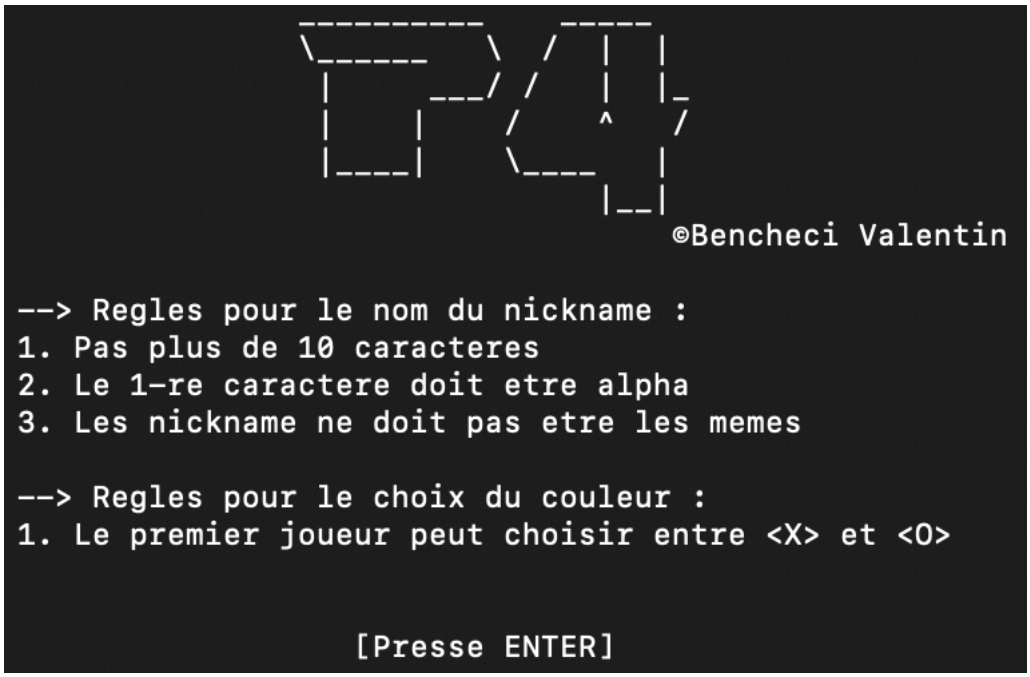
- **Logo :** pour un design plus attractif, j'ai décidé d'utiliser un logo.

- **Compte à rebours :** le jeu étant de type console, j'ai décidé de lui donner "vie" en utilisant le compte à rebours.



- **Presse ENTER :** ainsi, l'utilisateur ne passe à l'étape suivante que lorsqu'il est prêt.

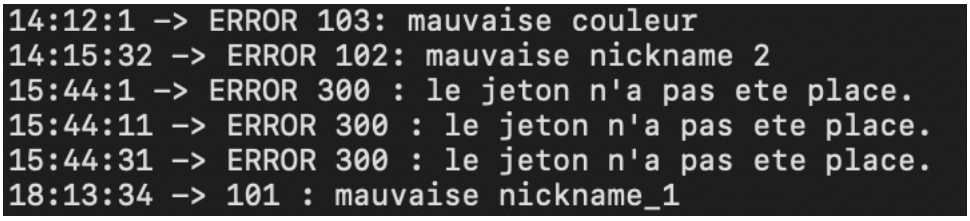
- **Les règles pour le surnom :** avant que l'utilisateur n'entre son pseudo, il est informé des règles à suivre.



- **Type du jeu :** pour distinguer le type de jeu, il est utilisé 3 « flags » pour la variable gameType : 0 – joueur VS joueur, 1 – joueur VS AI, 2 – AI VS AI.
- **Les codes d'erreurs :**

code	erreur
101	mauvaise nickname_1
102	mauvaise nickname_2
103	mauvaise couleur_1
301	le jeton n'a pas été place
401	il n'y a pas de parties sauvegardées
501	il n'y a pas de statistique

- **Journal des erreurs :** chaque erreur est notée dans le fichier « error.log ».



c. Les structures

- **GeneralGameConfig**

```
typedef struct
{
   int gameLenX; //dimension OX
   int gameLenY; //dimension OY

   int gameType; //flag pour le tip du jeu
   char gameBoard[9][9]; //le plateau du jeu

   int firstMove; //la valeur 1/2 | flag pour le joueur qui
                              //va commencer
      int lastMove; // si le jeu a été sauvegardé, nous avons un moyen          //de savoir qui
                              commencera
   int restMoves; //permet de savoir s'il y a encore d'espace libre
} generalGameConfig;
```

- **TerminalUserInterface**

```
typedef struct
{
   char gameLogo[255]; //le logo du jeu
   char developer[255]; //le nom du développeur

   char mainMenu[255]; //les options du menu principale
   char playMenu[255]; //les options que le joueur peut les    //faire pendant le jeu
   char typePlayMenu[255]; //les options pour le type du jeu
   char *allMenu[4]; //les adresses vers mainMenu, playMenu, typePlayMenu

   char nicknameRule[255]; //les réglés pour le surnom
   char gameRule[510]; //les réglés pour le jeu
   char *backCount[3]; //garde les chiffres 3, 2, 1 représentés  //comme chaines des caractères
} terminalUserInterface;
```

- **PlayerConfig**

```
typedef struct
{
   char player1Name[25]; //surnom du premier joueur
   char player2Name[25]; //surnom du deuxième joueur
   char player1Ch; //la couleur du premier joueur
   char player2Ch; //la couleur du deuxième joueur
} playersConfig;
```

- **StatisticConfig**

```
typedef struct
{
   int totalPvsP[3]; //[0] : nombres des jeux PvsP, [1] : jeux gagné par
                                 //le jouer 1, [2] : jeux gagné par le jouer 2
   int totalPvsAI[3]; //[0] : nombres des jeux PvsAI, [1] : jeux gagné   //par le jouer 1, [2] : jeux gagné
                                    par le jouer 2
   int totalAIvsAI[3]; //[0] : nombres des jeux AIvsAI, [1] : jeux gagné //par le jouer 1, [2] : jeux
                                     gagné par le jouer 2
} statisticConfig;
```

- **LogConfig**

```
typedef struct
{
   char errorLog[16]; //le nom du fichier pour les erreurs
   char gameLog[16]; //le nom du fichier pour les informations      //importantes
} logConfig;
```

d. Les fonctions

Compte tenu de la fonctionnalité du projet, les fonctions ont été divisées en 5 catégories.

- **INITALIZING CORE :**

void initializeGeneralGameConfig(generalGameConfig *ggc, int x, int y)

Cette fonction initialise la variable ggc.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param x : la dimension du tableau OX ;
: param y : la dimension du tableau OY.

void initializeTerminalUserInterface(terminalUserInterface *tui)

Cette fonction initialise la variable tui.
: param *tui : le pointeur vers une variable de type terminalUserInterface.

void initializePlayerConfig(playersConfig *plConf, terminalUserInterface *tui, logConfig *lgConf, int gmType)

Cette fonction initialise la variable plConf.
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param *gmType : le type du jeu | 0 -> player VS player, 1 -> player VS AI, 2 -> AI VS AI .

void initializeStatisticConfig(statisticConfig *stConf)

Cette fonction initialise la variable stConf.
: param *stConf : le pointeur vers une variable de type statisticConfig.

void initializeLogConfig(logConfig *lgConf)

Cette fonction initialise la variable lgConf.
: param *lgConf : le pointeur vers une variable de type logConfig.

- **MENU CORE :**

void mainMenu(generalGameConfig *ggc ,terminalUserInterface *tui, playersConfig *plConf, logConfig *lgConf, statisticConfig *stConf)

Cette fonction vise à rediriger le joueur concernant le menu principal(jouer, regarder la statistique, lire les regles du jeu). Tout d'abord, elle réinitialise tous les paramètres liés au jeu, puis affiche les options de menu. En utilisant le « switch » elle arrive à rediriger le joueur selon son choix.

: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig.

void typePlayMenu(generalGameConfig *ggc ,terminalUserInterface *tui, playersConfig *plConf, logConfig *lgConf, statisticConfig *stConf)

Cette fonction vise à rediriger le joueur vers le type du jeu possible.

: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig.

int printMenu(generalGameConfig *ggc, terminalUserInterface *tui, int optMenu, int optMax, char *player)

Cette fonction est particulièrement importante car elle vérifie que l'utilisateur entre de vraies données. Il faut mentionner que la fonction est générale et en fonction des paramètres qu'elle reçoit, elle peut afficher différents menus. Une fois que l'utilisateur a saisi les données, la fonction vérifie si elles ne sont pas négatives ou si elles ne dépassent pas la valeur du paramètre optMax.

: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param optMenu : le menu qu'il faut afficher ;
: param optMax : le nombre totale des options pour ce menu ;
: param *player : le nom du joueur.

- **PLAY CORE :**

void resetGameBoard(generalGameConfig *ggc)

Cette fonction réinitialise le plateau de jeu, le nombre de pas restants et la valeur de la variable lastMove.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

int evaluate(generalGameConfig *ggc, playersConfig *plConf, statisticConfig *stConf)

Cette fonction vérifie si l'un des 2 joueurs a gagné. Si le premier joueur gagne augmente la valeur de la variable de score de 5, sinon la diminue de 5, puis renvoie cette variable.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig ;
: return score : la valeur calculée en dépendance si un des joueurs a gagné.

int * minMax(generalGameConfig *ggc, playersConfig *plConf, statisticConfig *stConf, int deepth, int player)

Cette fonction utilise l'algorithme MiniMax expliqué ci-dessus. Tous les cas possibles sont divisés dans un arbre, puis le cas le plus favorable est sélectionné(un joueur essaie de prendre le nombre maximal des points et l'autre le nombre minimal). Il faut mentionner que juste les feuille sont évalués et la fonction s'appelle de façon récursive.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig ;
: param deepth : le nobre total des pas a verifier (hauteur) ;
: param player : l'ordre du joueur ;
: return *best : retourne dans best[0] le meilleure de la position et dans best[1] le score accumulé pour cette position.

void startGamePvsAI(generalGameConfig *ggc, terminalUserInterface *tui, playersConfig *plConf, logConfig *lgConf, statisticConfig *stConf)

Cette fonction lance le jeu entre le joueur et la machine et fonctionne tant que le plateau n'est pas plein ou que l'un des joueurs n'a pas gagné. Tout d'abord, elle appelle la fonction « decideFirstMove » pour décider au hasard lequel des joueurs commencera le jeu. Ensuite, elle affiche le menu du jeu et, selon le choix du joueur, exécute certaines fonctions.

: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig.

void startGameAIvsAI(generalGameConfig *ggc, terminalUserInterface *tui, playersConfig *plConf, logConfig *lgConf, statisticConfig *stConf)

Cette fonction lance le jeu entre la machine et la machine et fonctionne tant que le plateau n'est pas plein ou que l'un des joueurs n'a pas gagné. Tout d'abord, elle appelle la fonction « decideFirstMove » pour décider au hasard lequel des joueurs commencera le jeu. Il faut mentionner que le choix pour cet type du jeu est toujours 1.

: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig.

void checkScore(generalGameConfig *ggc, char gameBoard[9][9], int yCoord, int xCoord, char color, int *score)

Cette fonction vérifie si le jeton de couleur reçu par le paramètre est dans la position indiquée. S'il existe, le paramètre score est incrémenté, sinon il reçoit la valeur 0.

: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param gameBoard[9][9] : un tableau avec le plateau du jeu;
: param yCoord : la position OY ;
: param xCoord : la position OX ;
: param color : la couleur (X ou O) ;
: param *score : pointer vers une variable int (dans notre cas, vers score).

void showWinner(generalGameConfig *ggc, terminalUserInterface *tui, playersConfig *plConf, statisticConfig *stConf, int chWin)

Cette fonction reçoit par paramètre l'ordre du joueur qui a gagné et affiche le résultat.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param chWin : l'ordre du joueur qui a gagné.

int checkWinner(generalGameConfig *ggc, playersConfig *plConf, statisticConfig *stConfig)

Cette fonction vérifie si l'un des joueurs a gagné. Il vérifie d'abord verticalement et horizontalement, puis il vérifie également les diagonales. L'algorithme de vérification est le suivant: vérifier chaque élément augmente la valeur de la variable de score, si elle est supérieure ou égale à 4, alors le joueur a gagné.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig ;
: return *best : retourne 1 si le premier joueur a gagné, 2 si le deuxieme, 3 si c'est egalité et -1 si aucun n'a pas gagné.

void transpose(generalGameConfig *ggc)

Cette fonction transpose le plateau de jeu.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

void reverseRow(generalGameConfig *ggc)

Cette fonction inverse les lignes du plateau de jeu.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

void reverseCol(generalGameConfig *ggc)

Cette fonction inverse les colonnes du plateau de jeu.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

void turnLeft(generalGameConfig *ggc)

Cette fonction vérifie d'abord si tous les éléments sont placés le plus gauche possible sur le plateau et après elle appelle les fonctions « transpose » et « reverseCol ».
: param *ggc : le pointeur vers une variable de type generalGameConfig.

void turnRight(generalGameConfig *ggc)

Cette fonction vérifie d'abord si tous les éléments sont placés le plus droit possible sur le plateau et après elle appelle les fonctions « transpose » et « reverseRow ».
: param *ggc : le pointeur vers une variable de type generalGameConfig.

void turn180(generalGameConfig *ggc)

Cette fonction appelle les fonctions « reverseCol » et « reverseRow ». Apres, elle met tous les jeton le plus bas possible.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

void decideFirstMove(generalGameConfig *ggc)

La fonction vérifie si la variable lastMove est différente de -1, dans ce cas, le programme doit générer une nouvelle partie et sélectionne au hasard lequel des 2 joueurs commencera en premier. Sinon, la fonction définit que le joueur qui a fait la dernière étape démarrera.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

int checkMove(generalGameConfig *ggc, int coordinate)

La fonction vérifie si la variable lastMove est différente de -1, dans ce cas, le programme doit générer une nouvelle partie et sélectionne au hasard lequel des 2 joueurs commencera en premier. Sinon, la fonction définit que le joueur qui a fait la dernière étape démarrera.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param coordinate : la valeur OX ou le joueur veut ajouter le jeton ;
: return : retourne 1 si c'est possible d'ajouter le jeton et 0 sinon.

void startGamePvsP(generalGameConfig *ggc, terminalUserInterface *tui, playersConfig *plConf, logConfig *lgConf, statisticConfig *stConf)

Cette fonction lance le jeu entre le joueur et l'autre joueur et fonctionne tant que le plateau n'est pas plein ou que l'un des joueurs n'a pas gagné. Tout d'abord, elle appelle la fonction « decideFirstMove » pour décider au hasard lequel des joueurs commencera le jeu. Ensuite, elle affiche le menu du jeu et, selon le choix du joueur, exécute certaines fonctions.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param *stConf : le pointeur vers une variable de type statisticConfig.

void printGameBoard(generalGameConfig *ggc)

La fonction affiche le tableau du jeu avec les coordonnés.
: param *ggc : le pointeur vers une variable de type generalGameConfig.

- **USER CORE :**

int getPlayerOrderWithColor(playersConfig *plConf, char color)

Cette fonction vérifie si le paramètre « color » est identique à celui du premier joueur ou à celui du joueur 2 et renvoie le resultat.
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param color : la couleur du jeton, X ou O ;
: return : retourne 1 si le paramètre « color » est identique avec la couleur choisi par le joueur 1 et 0 sinon.

int getAnotherPlayer(int player)

Cette fonction renvoie l'ordre du joueur opposé à celui donné dans le paramètre.
: param player : l'ordre du joueur, 1 où 2 ;
: return : retourne 1 si le paramètre « player » est egale à 2 et 2 sinon.

char getPlayerColor(playersConfig *plConf, int player)

La fonction renvoie la couleur choisie par le joueur donnée dans le paramètre.
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param player : l'ordre du joueur, 1 où 2 ;
: return playerCh : retoune la couleur choisi par le joueur passé en parametre.

int checkNickname(char input[])

La fonction vérifie si la longueur du paramètre est comprise entre 1 et 9 et si le premier caractère est alpha.
: param input[] : le chaine des caracteres ;
: return : retoune 0 si le parametre est bon comme nickname et 1 sinon.

char getOppositColor(char ch)

La fonction renvoie la couleur opposée.
: param ch : X ou O ;
: return : X si le parametre est O, O sinon.

char * getPlayerWithColor(playersConfig *plConf, char ch)

La fonction renvoie le nom du joueur qui a choisi la couleur identique au paramètre ch.
: param *plConf : le pointeur vers une variable de type playerConfig ;
: param ch : X ou O ;
: return playerName : retoune le nom du joueur qui a choisi la couleur identique au paramètre ch.

- **UTILITY CORE :**

void assign2Arrays(int a1[], int a2[])

La fonction affecte les valeurs du paramètre 2 au paramètre 1.
: param a1[] : un tableau ;
: param a2[] : un tableau.

void printHeader(terminalUserInterface *tui)

La fonction supprime la console puis affiche le logo du jeu et le nom du développeur.
: param *tui : le pointeur vers une variable de type terminalUserInterface.

void showStatistic(statisticConfig *stConf, terminalUserInterface *tui)

Cette fonction affiche les données relatives aux statistiques du jeu sous forme de tableau.
: param *stConf : le pointeur vers une variable de type statisticConfig ;
: param *tui : le pointeur vers une variable de type terminalUserInterface.

void loadStatistic(statisticConfig *stConf)

Cette fonction lit les informations du fichier statistics.txt et les charge dans la variable stConf.
: param *stConf : le pointeur vers une variable de type statisticConfig.

void updateStatistic(statisticConfig *stConf, int gmType, int playerOrder)

Cette fonction incrémente la valeur de la variable statistiquement liée en fonction des paramètres reçus.
: param *stConf : le pointeur vers une variable de type statisticConfig ;
: param gmType : le type du jeu, 0/1/2 ;
: param playerOrder : l'order du joueur, 1 où 2.

int checkFileExists(char fileName[])

La fonction renvoie 1 si elle a réussi à ouvrir le fichier (c'est-à-dire qu'il existe) 0 sinon.
: param fileName[] : le nom du fichier.

void loadSavedGame(generalGameConfig *ggc, playersConfig *plConf)

La fonction lit dans le fichier "savedGames.txt" les informations sur la partie sauvegardée et les charge dans les paramètres reçus.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *plConf : le pointeur vers une variable de type playerConfig.

void saveStatistic(statisticConfig *stConf)

La fonction enregistre les informations de paramètre dans le fichier "statistics.txt".
: param *stConf : le pointeur vers une variable de type statisticConfig.

void saveGame(generalGameConfig *ggc, playersConfig *plConf)

Cette fonction enregistre dans le fichier les informations sur les joueurs, les étapes restantes, le joueur à suivre et le plateau de jeu.
: param *ggc : le pointeur vers une variable de type generalGameConfig ;
: param *plConf : le pointeur vers une variable de type playerConfig.

void freeBuffer(FILE* f)

La fonction libère les informations qui se trouvent dans le buffer.
: param *f : le pointeur vers le stream ou le fichier.

void waitEnter(void)

La fonction est exécutée jusqu'à ce que l'utilisateur appuie sur ENTER.

void backCount(terminalUserInterface *tui)

La fonction affiche les variables backCount, c'est-à-dire 3/2/1.
: param *tui : le pointeur vers une variable de type terminalUserInterface.

void logInfo(char fileNom[], char information[])

La fonction enregistre dans le fichier les informations fournies dans le paramètre, aidant ainsi l'heure actuelle.
: param fileNom[] : le nom du fichier où il faut sauvegarder l'information ;
: param information[] : l'information qu'il faut sauvegarder.

void warnError(terminalUserInterface *tui, logConfig *lgConf, char error[])

La fonction affiche le message d'erreur et appelle la fonction logInfo pour l'enregistrer dans le fichier.
: param *tui : le pointeur vers une variable de type terminalUserInterface ;
: param *lgConf : le pointeur vers une variable de type logConfig ;
: param error[] : le chaine des caracteres concernant l'information d'erreur.

void printLine(int n)

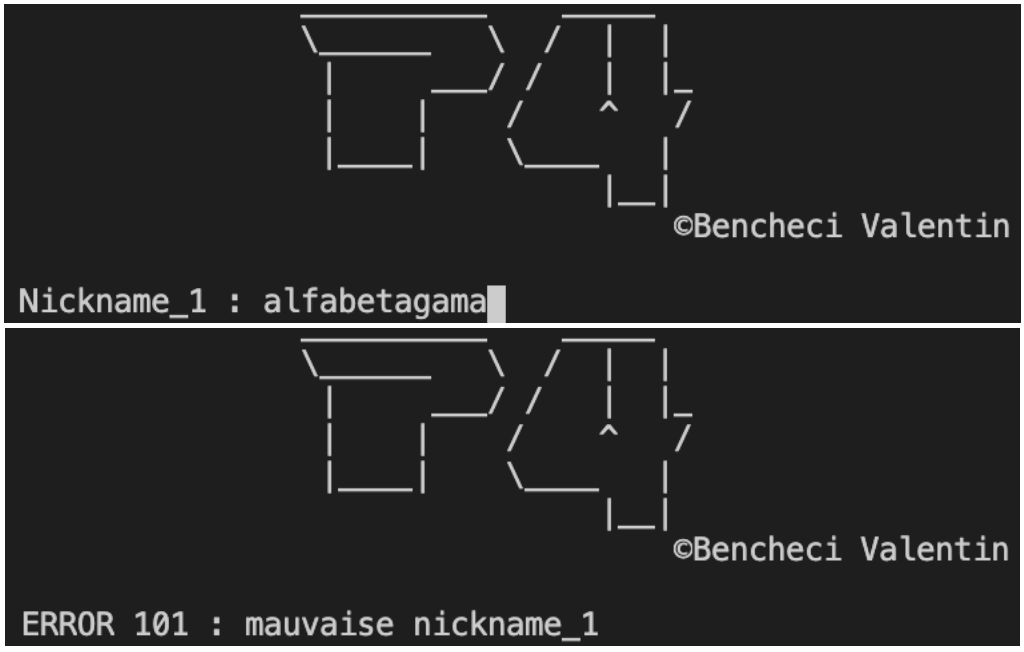
La fonction affiche le caractère "-" n fois, formant ainsi une ligne.
: param n : combien de fois le caractère doit être affiché.

void printCoordinates(int x)

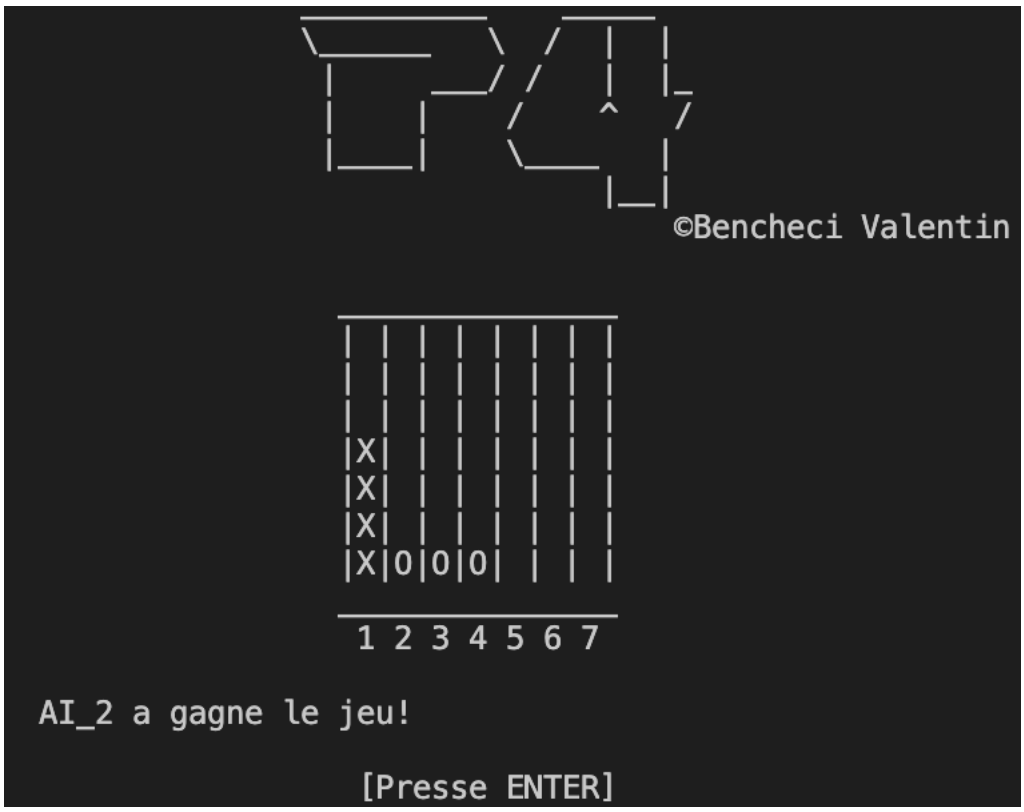
La fonction affiche les coordonnées de 1 à x.
: param x : combien de coordonnées doivent être affichées.

5. CAS DE TEST

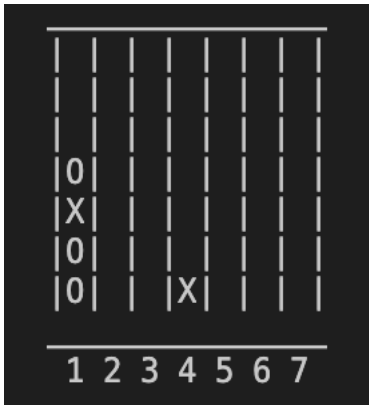
- **Surnom qui ne respecte pas les règles**

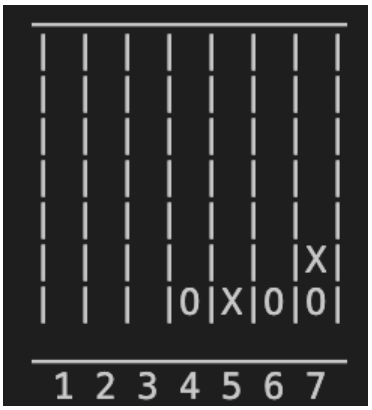


- **Situations où l'un des joueurs gagne**

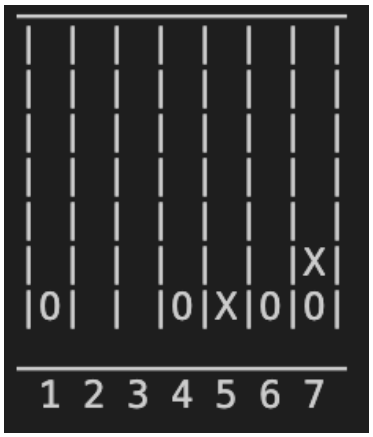


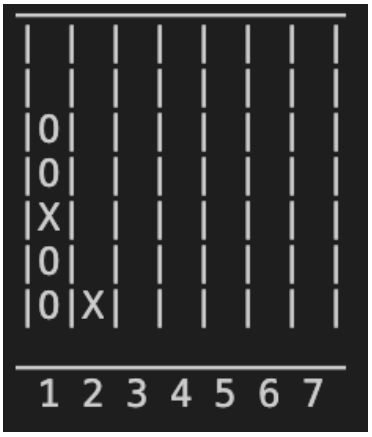
- **Tourner le plateau 90 a gauche**



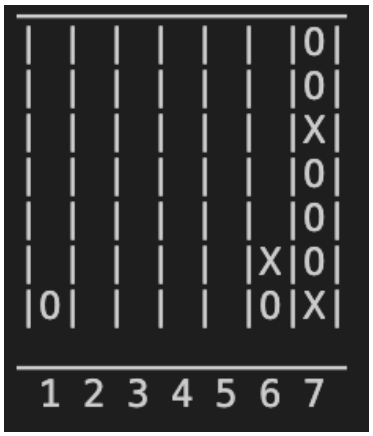


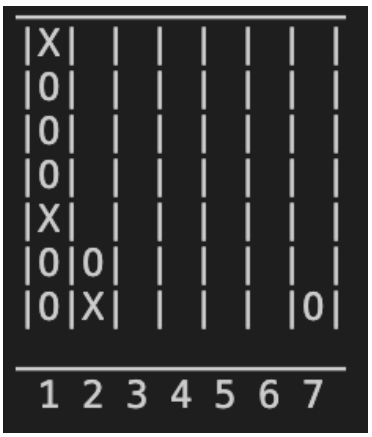
- **Tourner le plateau 90 a droite**





- **Tourner le plateau 180 a gauche**





- **Le fichier "statistics.txt" n'existe pas**

```
ERROR 501: Il n'y a pas de statistique
```

- **Le fichier "savedGames.txt" n'existe pas**

```
ERROR 401: Il n'y a pas de parties sauvegardees
```

- **Sauvegarde des journaux(errors.log)**

```
19:8:34 -> 301 : le jeton n'a pas ete place. Player: AI_1
19:8:38 -> 301 : le jeton n'a pas ete place. Player: AI_1
19:24:24 -> 301 : le jeton n'a pas ete place. Player: AI_2
21:3:1 -> 301 : le jeton n'a pas ete place. Player: Vlad
21:3:15 -> 301 : le jeton n'a pas ete place. Player: Vlad
21:14:26 -> 301 : le jeton n'a pas ete place. Player: Valentin
21:14:31 -> 301 : le jeton n'a pas ete place. Player: Valentin
21:15:13 -> 301 : le jeton n'a pas ete place. Player: Valentin
21:38:47 -> 101 : mauvaise nickname_1
21:52:51 -> 103 : mauvaise couleur_1
22:23:26 -> 102 : mauvaise nickname_2
22:28:19 -> 101 : mauvaise nickname_1
22:55:13 -> 501: Il n'y a pas de statistique
22:57:0 -> 401: Il n'y a pas de parties sauvegardees
```

- **Affichage des statistiques**

```
---> Statistiques

-------------------------------------------------------------------------------------------------------------------
| Total jeux player VS player :    0   | Victoires player1 :            0   | Victoires player2 :            0   |
| Total jeux player VS AI :        0   | Victoires player1 :            0   | Victoires player2 :            0   |
| Total jeux AI VS AI :            5   | Victoires player1 :            4   | Victoires player2 :            1   |
-------------------------------------------------------------------------------------------------------------------
```

6. CONCLUSION

Du début à la fin, j'ai été très enthousiasmé par ce projet. J'avoue qu'il y a eu des moments difficiles, certains même très difficiles, mais en analysant mieux les problèmes j'ai réussi à les résoudre. Ce projet m'a aidé à mieux comprendre comment le code doit être structuré et comment l'optimiser.

Enfin, les temps impartis à la réalisation du système furent bref et il a fallu faire preuve de flexibilité et de persévérance, parfois pour respecter les délais, parfois pour respecter les contraintes technologiques imposées par le projet.