Toutain Guillaume et Huriez Mathis

# PT 1.1 : Blocus

C89

# Sommaire :

# Introduction :

Le projet "Blocus" est écrit en langage C89. Le but de ce projet est de réaliser un jeu pour un ou deux joueurs. Seul ou en binôme.

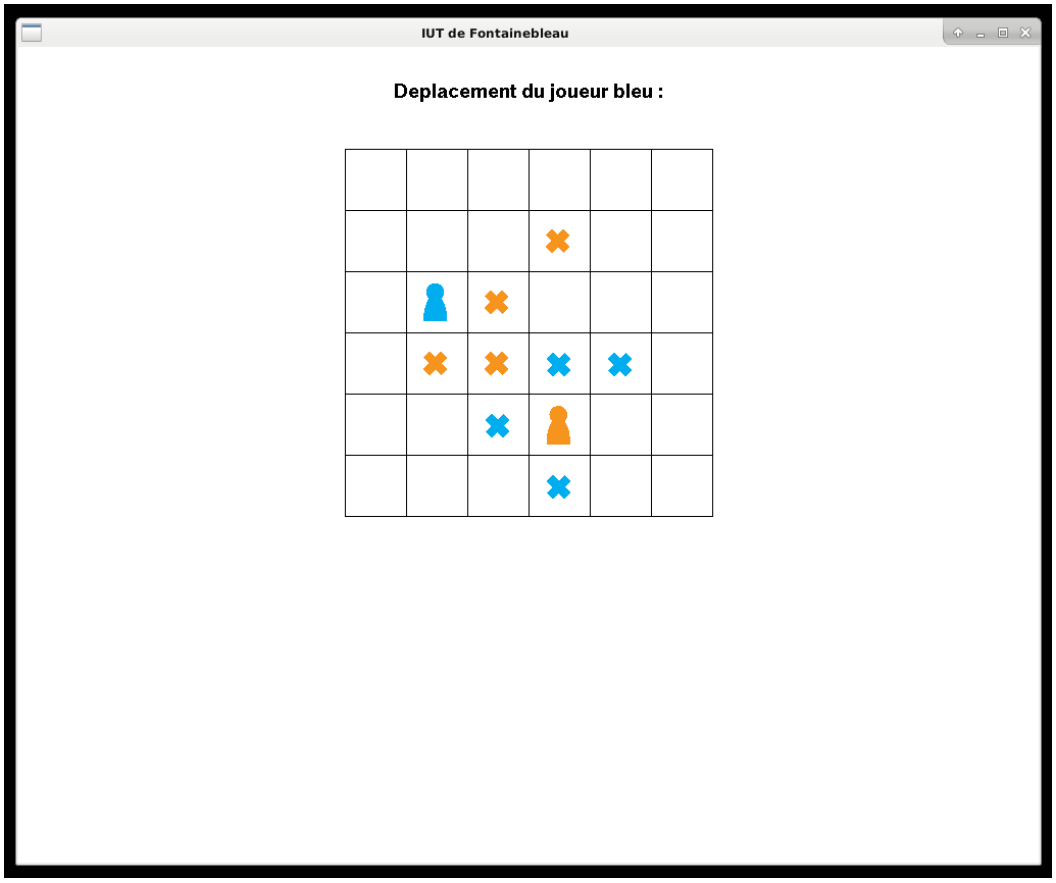
Ce jeu prend la forme d'une grille carré entre 3 et 9 cases de largeur et hauteur, au choix de l'utilisateur. Cette affichage est possible grâce, à la bibliothèque graphique de l'IUT. Aucun autre emprunt extérieur n'est permis pour ce projet. Chaque joueur est représenté par une couleur, bleu pour le premier joueur, et orange pour le second.

L'entièreté du jeu se contrôle uniquement à la souris.

# Fonctionnalités :

## Déroulement :

Lors du lancement du jeu, un premier écran s'affiche, permettant de choisir la taille de la grille entre 3 et 9, ainsi que le mode de jeu : 1 joueur (et donc jouer contre l'ordinateur) ou bien 2 joueurs. On clique sur "jouer", ce qui affiche le deuxième écran, celle de la grille. À la fin de la partie, le dernier écran s'affiche, dévoilant le gagnant, ainsi que la proposition de rejouer ou de quitter le jeu.


IUT de Fontainebleau
Deplacement du joueur bleu :


IUT de Fontainebleau
BLOCUS
Partie terminee !
Le gagnant est le joueur 1
Rejouer
Quitter
Projet realise par Mathis Huriez & Guillaume Toutain

L'objectif du jeu est de bloquer le pion ennemi pour gagner, pour se faire, voici le déroulement d'une partie :
Chaque joueur pose son pion sur une case de la grille, puis vient le tour du premier joueur (bleu) se déplace et positionne une croix (bleu) sur la grille pour empêcher l'autre joueurs de se déplacer dessus. C'est ensuite au tour du second joueur (orange) de se déplacer et positionner sa croix (orange). Les joueurs alternent ainsi leur tour, déplacement puis positionnement d'une croix, jusqu'à que l'un deux, ou les deux, ne puisse plus se déplacer.

## Règles :

Le jeu suit des règles précises et définies, les voici :

- Le joueur bleu commence toujours en premier.

- On ne peut pas placer son pion, le déplacer, ou positionner une croix en dehors du terrain délimité par l'utilisateur entre 3 et 9.

- Les pions ou croix ne peuvent pas se superposer. Un pion ne peut pas être placer ou déplacer sur un autre pion ou une croix, une croix ne peut pas être placer sur un pion ou une autre croix (peu importe la couleur).

- Un pion ne peut se déplacer que sur une case adjacente, horizontalement, verticalement ou en diagonale.

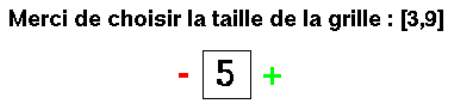
- Une nouvelle croix peut être placé sur n'importe quelle case libre.

- La partie prend fin quand le premier des deux joueur ne peut pas se déplacer.

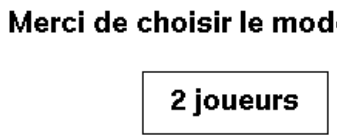
## Détails :

Le jeu possède plusieurs détails pour un confort d'utilisation du jeu.

Dans un premier temps, le menu est intuitif, l'étendu de la grille est affiché ([3,9]) et les signes – et + sont visibles, permettant de modifier la valeur afficher au centre de l'écran qui s'affiche en temps réel.



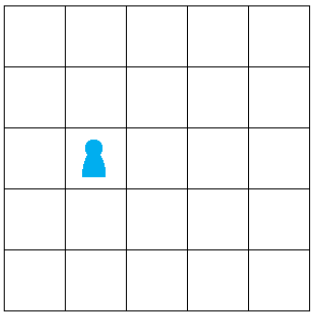

Le mode de jeu se choisis aisément, on clique sur le bouton pour échanger entre le mode 1 joueur et le mode 2 joueurs.



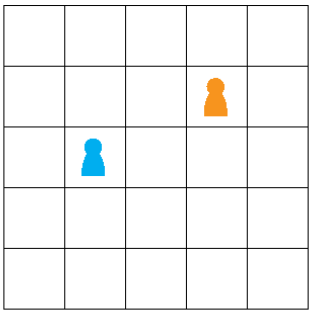

Ce type de menu permet de voir l'entièreté des paramètres de la partie avant de les confirmer en appuyant sur le bouton "Jouer !".

Les textes écrits sur les différents écrans permettent à l'utilisateur de se retrouver à n'importe quel moment, que ce soit sur le menu, mais aussi en jeu, en effet, le statut de la partie est toujours affiché, ce qui permet de savoir quel doit être la prochaine action.

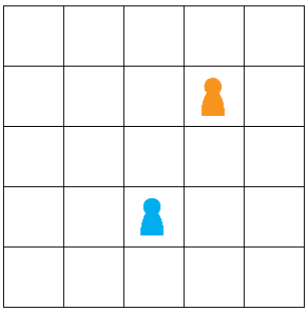
**Positionnement du joueur orange :**



**Deplacement du joueur bleu :**

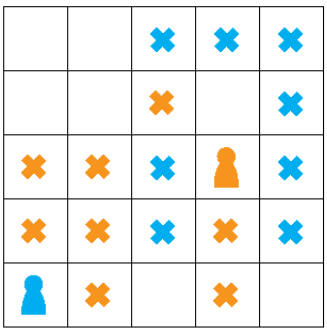


**Positionnement d'une croix du joueur bleu :**



De plus, lors de la fin de la partie (à partir du moment où un joueur ne peut plus se déplacer), le jeu attend un clic de l'utilisateur (indiquez dans le statut du jeu) pour confirmer la fin de la partie et afficher l'écran des scores, cela permet de laisser le temps à l'utilisateur de regarder l'état de la partie tant qu'il le souhaite (pour savoir pourquoi il a perdu ou gagné par exemple).

**>> Fin de partie (cliquez pour continuer) <<**



Le jeu se veut accessible à tous. Ce qui justifie les détails de celui-ci. Dans le mode un joueur, le robot réagit en jouant son tour presque instantanément, pour permettre des parties rapides aux joueurs les plus expérimentés. Robot qui est doté d'une certaine intelligence, il tentera de bloquer le pion du joueur en plaçant une croix sur une des cases adjacentes à celui-ci. Bien entendu, le robot est soumis aux mêmes règles que le joueur et ne trichera pas.

# Organisation du code :

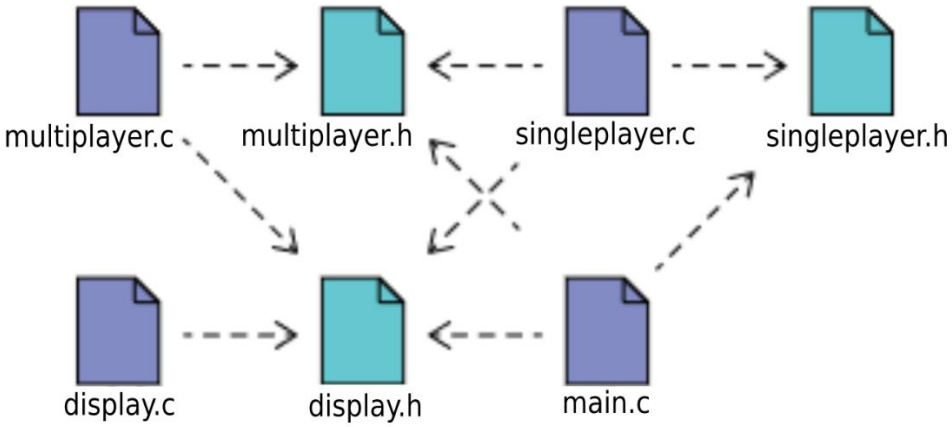
Le code est reparti en plusieurs fichiers sources. Cela nous a permis de se répartir le travail et de s'organiser, voici l'organisation générale du code :

- Main.c : Gère le déroulement du jeu
- Multiplayer.c : Gère les fonctions multijoueur (et aussi les actions du joueur 1 face au robot, inutile de refaire ce qui est déjà fait)
- Singleplayer.c : Ajoute les fonctions manquantes à multiplayer.c pour déplacer automatiquement le joueur 2
- Display.c : Interface avec la librairie graphique adapté au projet

Bien entendu les fichiers .c disposent également d'un fichier .h qui continents les prototypes des fonctions.

## Dépendances :

Voici les dépendances de chaque fichiers sources :




Le fichier Makefile permet de compiler l'ensemble pour former l'exécutable "pt11"

Pour ce faire il compile 4 fichier objets (.o) à l'aide de leurs dépendances respectives :

Multiplayer.o → display.h ; multiplayer.h

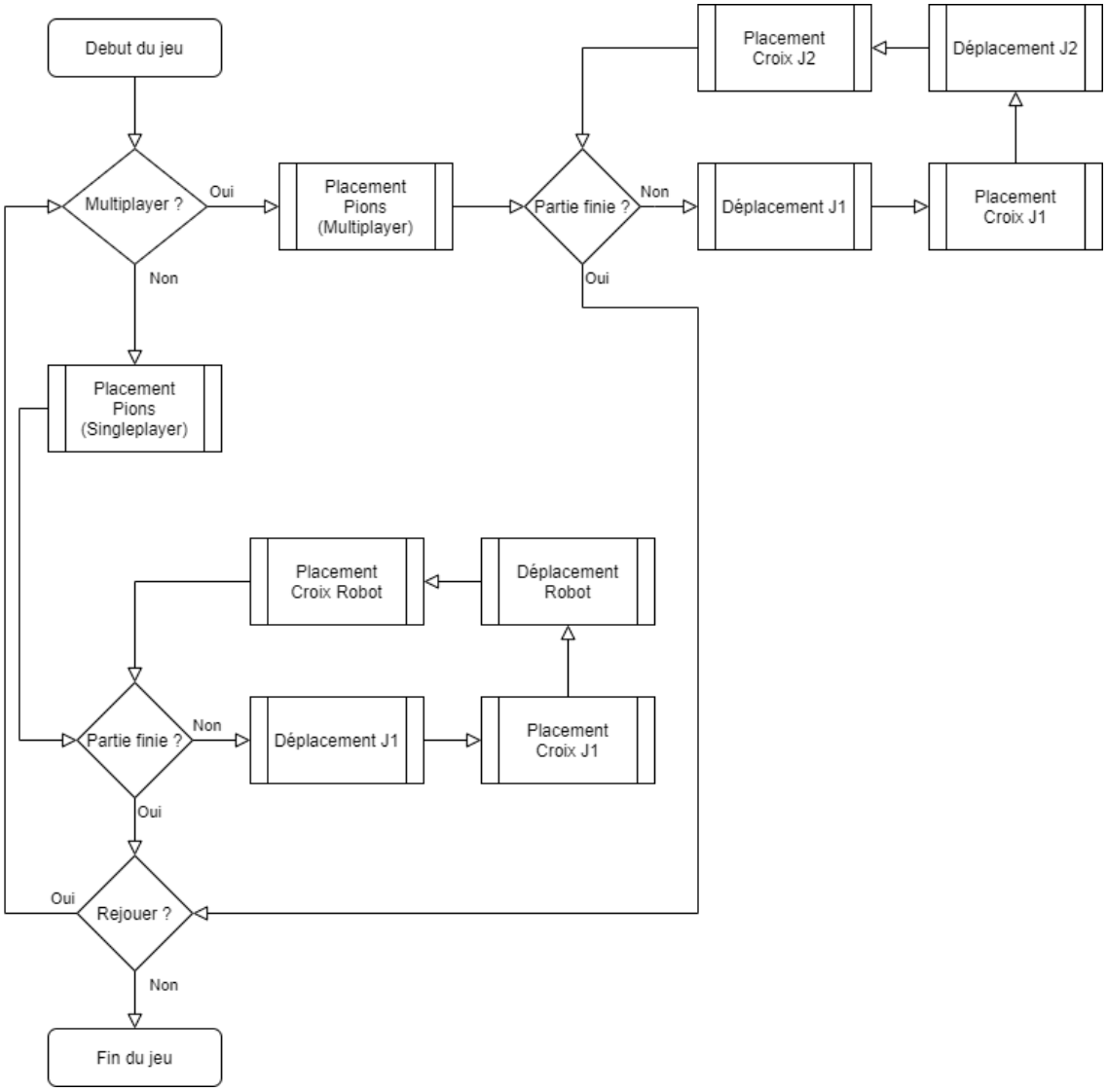
Singleplayer.o → display.h ; multiplayer.h ; singleplayer.h

Display.o → display.h

Et enfin main.o qui utilise tous les fichiers objets précédents.

# Structure du code :

## Diagrammes :



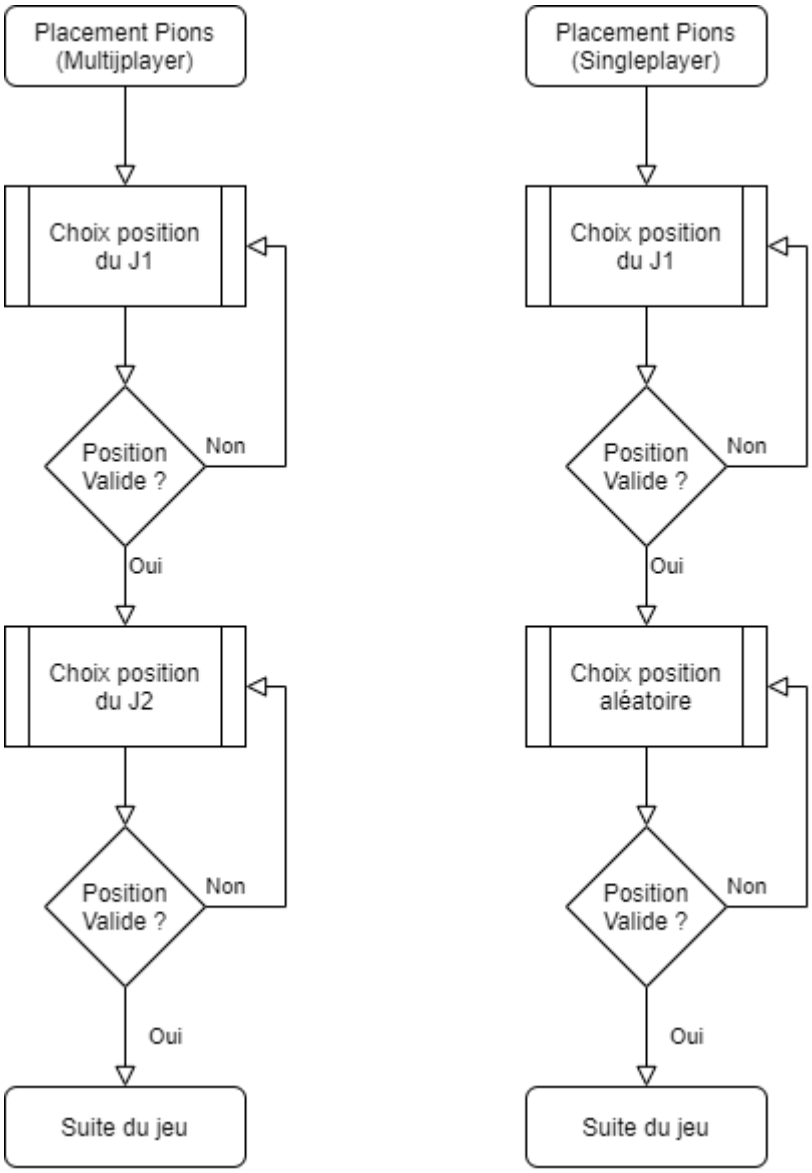



Schéma « flowchart » de la mise en place des pions.

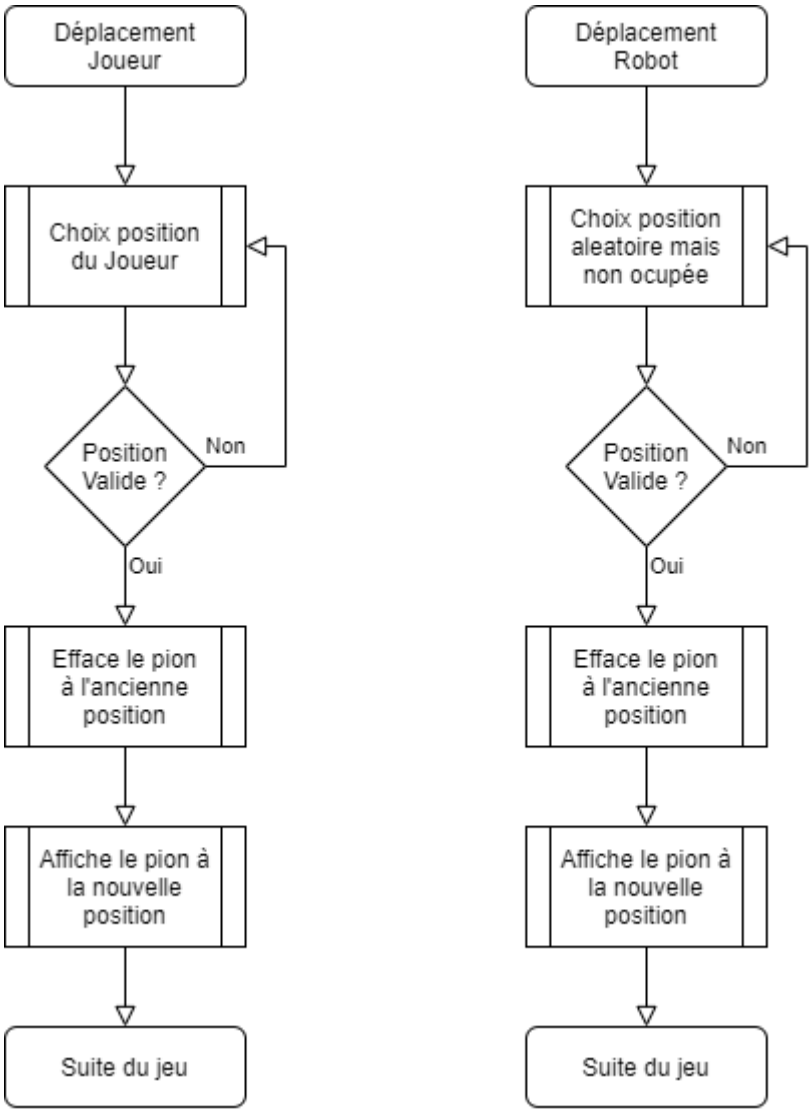



Schéma « flowchart » du déplacement des pions

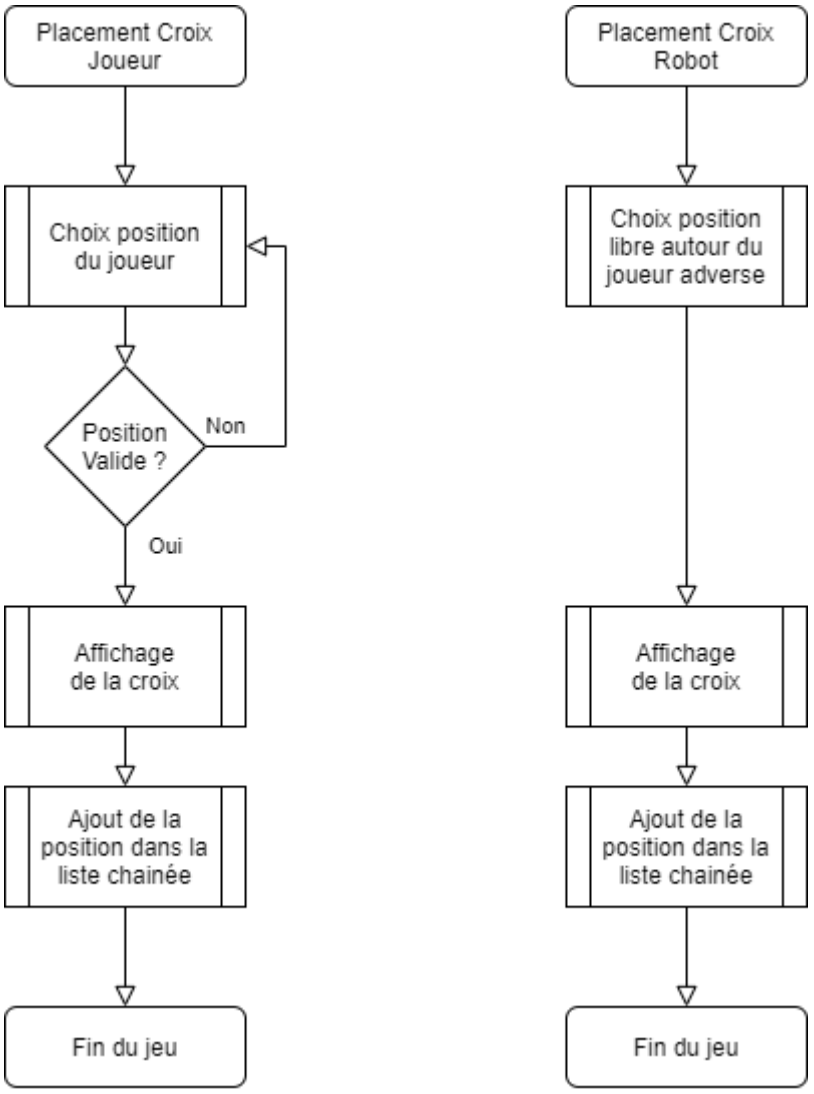



Schéma « flowchart » du placement des croix

## Fonctions :

Nous allons ici voir les différentes fonctions de chaque fichier. Nous regarderons leurs signatures et leurs utilités.

### Main.c :

Le fichier main.c utilise les fonctions des autres fichiers, il n'a pas de fonctions propres.

### Display.c :

- InitDisplay : Permet d'initialiser le module graphique
- CloseDisplay : Permet de fermer le module graphique (evite une inclusion de graph.h dans le main, et aussi de regrouper les fonctions graphiques)
- EcrireTexteCentre : Permet d'afficher du texte centré horizontalement sur la fenêtre.
- ShowSelectDisplay : Affiche l'écran de sélection. Il change d'écran virtuel de la librairie, efface l'écran. Ensuite il affiche touts les textes / images statiques. Ensuite il attend un click dans une boucle while qui s'arrête à l'appui du bouton jouer. A chaque clic l'écran actualise les valeurs dynamiques en fonction de la position du clic.
- ShowGameDisplay : Affiche une grille de la bonne taille à l'écran après voir changé d'écran virtuel et effacé l'écran.
- ShowResultDisplay : Encore une fois change d'écran virtuel et efface l'écran. Puis affiche le gagnant puis attends le clic de l'utilisateur sur un des deux boutons. Cette fonction retourne le choix de la personne.
- DisplayCross : Affiche une croix selon la position choisie par rapport à la taille de la grille.
- DisplayPlayer : Affiche un joueur selon la position choisie par rapport à la taille de la grille.
- EraseCell : Efface une cellule par rapport à la taille de la grille.
- GetClickPos : Attends un clic et retourne sa position relative à grille de taille choisie.
- SetGameStatus : Affiche un message en haut de l'écran.

## Multiplayer.c :

- FirstsPlayersPositions : cette fonction récupère la taille de la grille pour pouvoir adapter faire les calculs en fonction du choix de la taille par le joueur. Elle renvoie les coordonnées (variable de type "pos", une structure) avec x et y (entiers) de chacun des deux joueurs (grâce à une structure de structure).
- MovementPlayer1 : elle prend la taille de la grille, la position des 2 joueurs ainsi que la liste des croix posé sur la grille. Cette fonction sert à déplacer le joueur 1 en respectant les différentes règles, puis elle renvoi la position des deux joueurs.
- CrossPlayer1 : elle récupère la même chose que la fonction MovementPlayer1, Cette fonction place une croix sur la grille à l'endroit où le joueur clique tant que la position ne respecte pas les différentes règles. Puis elle ajoute la position de la nouvelle croix à la liste chaînée grâce à la fonction AddElement et renvoi cette liste chaînée.
- MovementPlayer2 : comme MovementPlayer1 mais pour le joueur 2.
- CrossPlayer2 : comme CrossPlayer1 mais pour le joueur 2.
- AddElement : prend comme paramètres une coordonnée ainsi que l'adresse du maillon/de l'élément précédent. Elle ajoute la coordonnée récupéré à la liste chaînée et renvoi cette liste.
- DeleteList : Récupère une liste chaînée et la supprime.
- BlockedCase : prend comme paramètres le premier élément de la liste chaînée et une coordonnée. Elle test si la case regardé (celle donnée en paramètre) est occupée ou non par une croix, grâce à une boucle qui parcourt la liste chaînée pour faire les tests. Elle renvoie 1 si la case est occupée est 0 si elle est libre.
- EndOfGame : récupère les mêmes paramètres que MovementPlayer1 plus une coordonnée (pos -> x et y). La coordonnée donnée est soit la position du joueur 1, soit celle du joueur 2, la fonction test si toute les 8 cases adjacentes sont inaccessibles (en dehors de la grille, possède un pion ou possède une croix), donc permet de tester si le joueur donné en paramètre est bloqué ou non. Elle renvoi 1 si le joueur est bloqué et 0 dans le cas contraire.

## Singleplayer.c :

- IsCellFree : Détecte si une case est libre ou non. La fonction vérifie cela grâce à la liste chainée des cases qui continent des croix. Elle vérifie également si la case est en dehors des limites du plateau. Elle retourne un int qui représente l'état de la case
- MovementRobot : Reprends le même principe que les autres fonctions pour le mouvement. Son but est de déplacer le robot en respectant les règles du jeu. Premièrement elle construit un tableau de int qui contient l'état des cas (i.e. Sont-elles libres ?) autour du robot. Ensuite elle choisit aléatoirement une des cases libres. Puis

elle cherche à quel mouvement cela correspond-il. Et pour finir elle déplace le pion et retourne la variable de suivi de l'état du jeu.

- SetupPositions : Permet de placer les pions au début de la partie, le joueur le place à la main avec un clic de la souris, le robot est placé aléatoirement sur la grille. Elle retourne la variable de suivi de l'état du jeu.
- CrossRobot : Permet au robot de placer une croix. Premièrement la fonction crée un tableau qui représente l'état des cases autour du joueur adverse. Elle choisit ensuite une case vide autour du joueur adverse. Elle cherche ensuite la position relative de cette case par rapport au joueur adverse, et enfin elle pose la croix et retourne l'adresse du premier élément de la liste chainée qui contiens la position des croix.

# Conclusion :

## Conclusion de Guillaume Toutain :

Personnellement j'ai bien aimé travailler sur ce projet pour plusieurs raisons.

Premièrement je préfère travailler sur un gros projet complexe (comme un petit jeu comme blocus), plutôt de que de faire plusieurs petits exercices comme en TP. On ressent une satisfaction une fois le projet terminé et fonctionnel, qui n'est pas présent sur un petit travail.

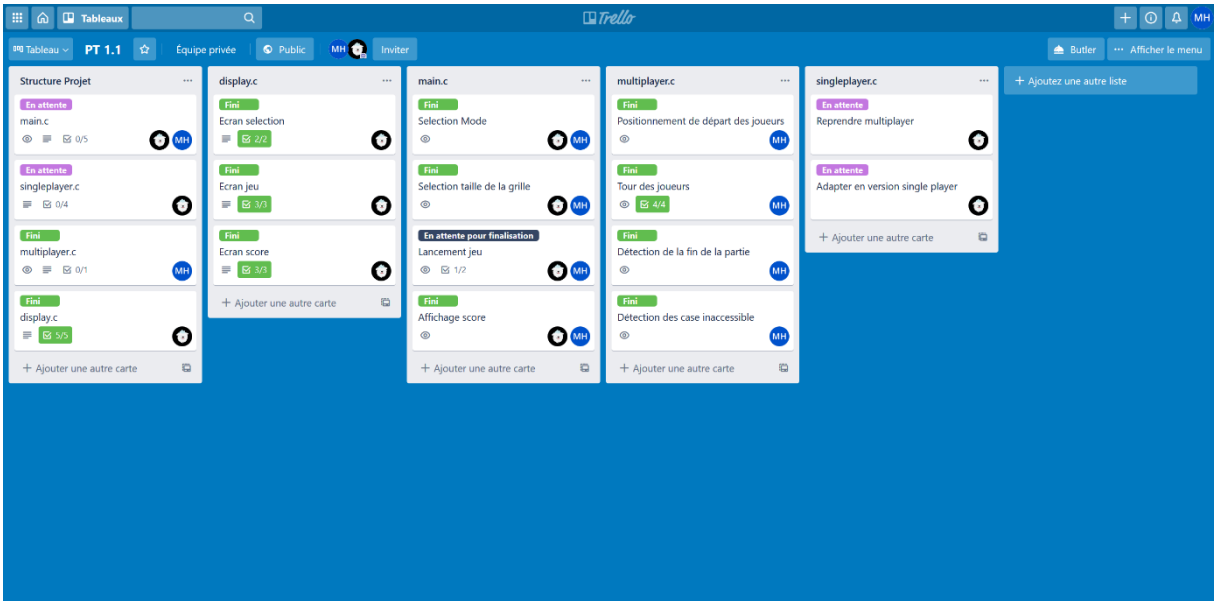
Également devoir travailler sur un jeu, et avec une partie graphique est pour moi plus enrichissante, qu'un banal projet console, car la partie graphique permet de s'intéresser plus en détail à l'ergonomie et à l'utilisation de l'utilisateur. Or avec un programme console, l'expérience utilisateur ne rentre pas en compte du processus de conception.

Je pense pouvoir dire que nous avons bien réussi à s'organiser grâce a la mise ne place de plusieurs outils tels que Trello et Discord. Discord nous à permis d'avoir une bonne communication. Notamment sur l'avancement des différents fichiers. Trello nous a permis d'avoir une « roadmap », dans le but de visuellement voir où on en est dans le projet et les risques de retard.

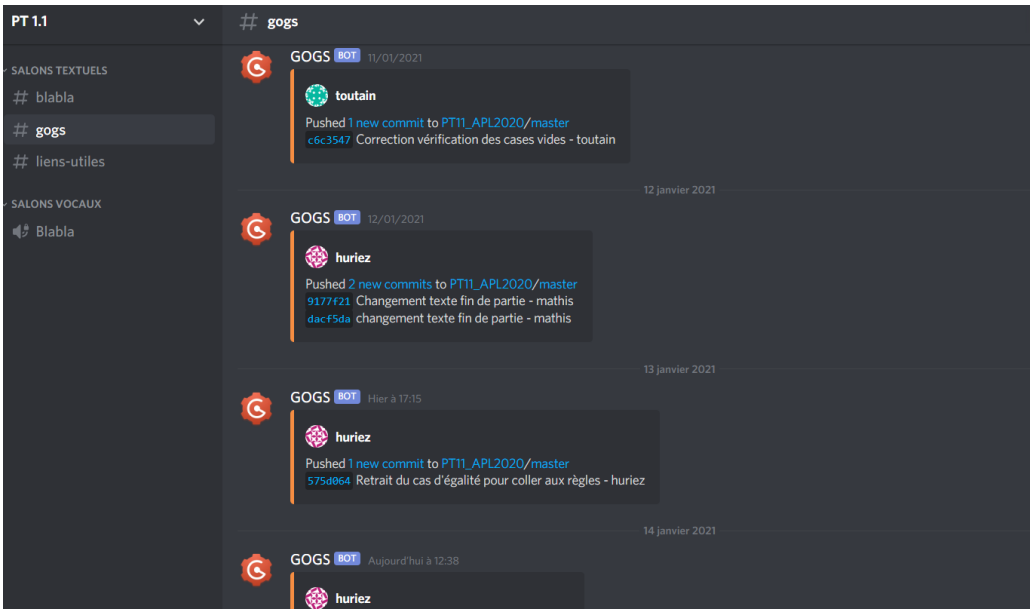
Pour finir, j'aime bien ce type de travail en mode « projet », et je suis curieux de savoir (s'il y aura) quel sera le prochain projet.

## Conclusion de Mathis Huriez :

Pour conclure sur mon expérience personnelle de ce projet, je trouve que l'organisation a été un point fort. Nous avons bien profité de GIT (avec plus de 30 commits), nous avons utilisé Trello pour pouvoir connaître les attributions de chacun, l'avancement du projet et les différentes tâches restantes.



Un serveur Discord a aussi été mis en place, permettant de discuter du projet, de répertorier les différents liens, ainsi que de créer un webhook de gogs, c'est-à-dire que nous avions un message automatique dés qu'un commit était effectué.




De plus, le sens dans lequelle le projet à été construit a été utile et bien pensé, c'est-à-dire, en premier les fonction du fichier display.c, qui servent pour tous les autres fichiers, puis les fonctions de multiplayer.c qui servent pour le singleplayer.c. Le fichier main.c étant construit au fur et à mesure des besoins, mais le corps de celui-ci ayant été pensé à l'avance.

Cette organisation m'a permis de réaliser ma partie du projet assez rapidement, connaissant la direction dans laquelle j'allais. Ce projet, à part au début le temps de bien comprendre toutes les choses, ne m'as pas posé de grandes difficultées et j'ai pris plaisir à le coder.