RAPPORT SAé DEV 2024

Jeu de Blocus

EL MCHANTEF Sarah et PEIRO TOMAS Maylee

BUT 1 INFO, Groupe 1

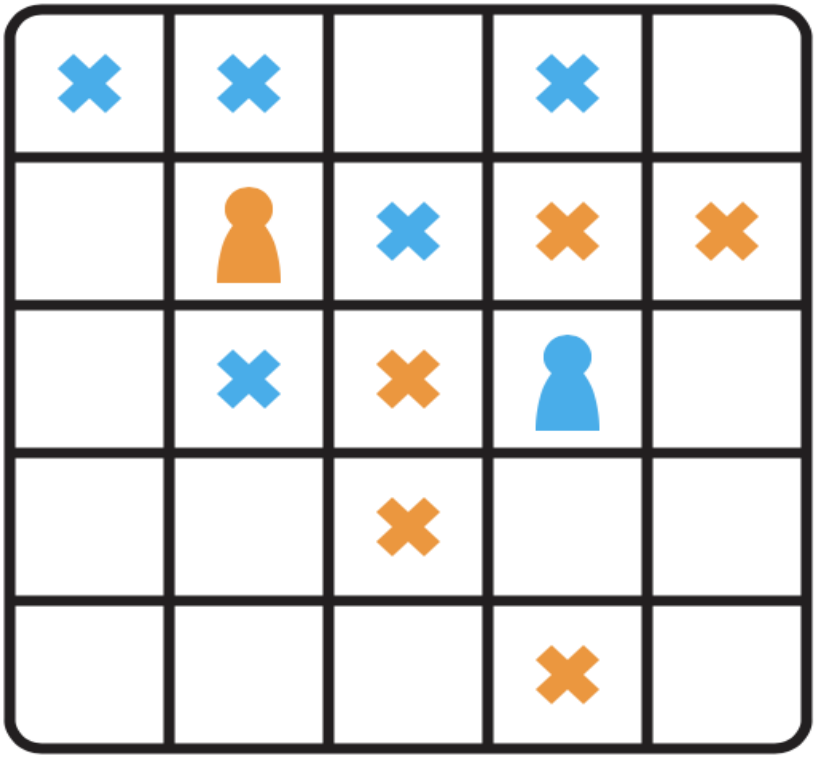
Sommaire :

I. Introduction :

Dans le cadre de notre formation en BUT informatique, nous avons eu à réaliser un projet : développer un jeu nommé Blocus en C89. Ce jeu oppose deux joueurs, ou un joueur contre un bot, sur une grille. Le but est de se déplacer stratégiquement tout en bloquant son adversaire.

Le fonctionnement du jeu est assez simple : chaque joueur déplace son pion sur une case adjacente (horizontalement, verticalement ou en diagonale) et condamne ensuite une case de son choix pour limiter les options de l'autre joueur. Lors du premier tour, les joueurs ne condamnent pas de case, ils placent simplement leur pion. La partie se termine lorsque l'un des deux joueurs ne peut plus bouger.

Ce projet a été une belle opportunité pour renforcer nos compétences en programmation C. En plus de l'aspect technique, cela nous a permis de mieux comprendre l'importance de l'organisation et du travail en équipe.



Le joueur bleu se déplace, puis condamne une case de son choix. Idem pour le joueur orange.

II. Brainstorming

À vrai dire, au tout début, nous ne savions pas vraiment par où commencer ni comment nous y prendre. Ce projet représentait une étape importante pour nous, car il s'agit de notre tout premier projet dans le cadre du BUT informatique. Nous avons donc passé un bon moment à discuter de la manière dont nous allions aborder le travail. Plusieurs questions nous venaient à l'esprit : par où commencer ? Quelles parties étaient les plus complexes ? Comment se répartir les tâches ? Voici un exemple d'une de nos sessions réflexions, nous avions principalement travaillé sous forme de tableaux:

Idée/Problèmes discutés	Solut° proposées	Actions à réaliser
Comment structurer le programme ?	• on fait tout dans un fichier et on divisera plus tard, ou on divise maintenant?	• Découpage du projet en modules
Representation de la grille	• tableau 2D (type int) pour chaques cases • differencier les cases avec des valeurs : par exemple, la case prend la valeur 0 pour le cas ou elle est libre, prend la valeur 1 quand elle est occupée par un pion et prend la valeur 2 quand elle est condamnée	• Définition d'un tableau global • Initialisation au départ de toutes les cases à 0 • Implémentation d'une fonction DeplacementValide pour verifier les deplacements avant l'exécution
Organisation du travail en équipe	- Maylee sur l'IA (1 joueur) - Sarah sur les 2 joueurs - Se voir régulierement pour tester et integrer le code.	• Planification de sessions de codage pendant les vacances et tous les jours (surtout jeudi, vendredi après midi et les week-ends)

Étant donné qu'il s'agissait de notre tout premier projet, nous avons longuement réfléchi à la meilleure manière de nous organiser. Après plusieurs discussions, nous avons décidé de commencer par travailler ensemble sur les étapes initiales, comme la gestion de la grille et l'affichage du menu principal, afin de nous familiariser avec les bases du projet. Une fois ces fondations posées, nous avons réparti les tâches plus spécifiques : l'implémentation de l'IA pour l'une, et la gestion de l'interface graphique pour l'autre. Grâce à cette organisation, nous avons pu avancer progressivement tout en identifiant rapidement les parties nécessitant des ajustements. Les échanges réguliers sur nos

avancées ont été essentiels pour conserver une vision globale du projet et garantir sa cohérence.

III. Difficultés rencontrées :

Comme pour tout projet, nous avons rencontré plusieurs difficultés au cours du développement du jeu Blocus. Ces obstacles, bien qu'ils aient parfois ralenti notre progression, ont également été une source d'apprentissage. En surmontant ces problèmes, nous avons acquis de nouvelles compétences qui nous seront utiles non seulement dans ce projet, mais aussi dans nos futurs travaux et dans notre vie professionnelle.

Honnêtement, au début, nous étions complètement perdues et paniquées. Nous avons eu du mal à nous organiser et à nous familiariser avec les concepts techniques du projet. Le plus gros obstacle au départ a été lié à l'affichage des différentes tailles de texte et à la gestion de l'interface. En effet, lorsque nous avons tenté d'intégrer des zones de texte interactives, Sarah a rencontré un problème technique sur son ordinateur : lorsqu'elle cliquait sur ces zones, la fenêtre ne réagissait pas comme prévu. Après avoir fait face à cette situation, nous avons décidé de demander de l’aide à notre professeur, Luc Hernandez. Il nous a suggéré d'utiliser des images à la place des zones de texte.

Malheureusement, cela ne s'est pas avéré aussi simple. Nous avons eu énormément de mal à afficher correctement nos images via la fonction ChargerImage. À plusieurs reprises, soit aucune image ne s'affichait, soit seulement une partie de l'image était visible. À ce moment-là, nous avons décidé d'utiliser la fonction ChargerSprite, qui a été bien plus adaptée à nos besoins. Après quelques ajustements, notamment le redimensionnement et le recadrage des images, nous avons réussi à afficher correctement les éléments tels que les grilles 3x3, 4x4, et les boutons représentant les modes de jeu.

Ces images, désormais affichées correctement, ont été utilisées comme des boutons cliquables, et leur interaction avec la souris a été rendue fonctionnelle grâce aux fonctions pour gérer les clics.

Un point important auquel nous avons dû prêter attention a été le choix de la taille des zones cliquables : elles devaient être suffisamment larges pour être facilement accessibles par le joueur, tout en étant suffisamment précises pour que le clic ne doive pas être répété plusieurs fois pour avoir un résultat.

Bien que ces problèmes aient été frustrants à certains moments, ils ont aussi renforcé nos compétences et nous ont appris à être plus rigoureux dans la gestion de l'interface et des entrées utilisateur.

IV. Fonctionnalités de notre programme :

La première fonctionnalité de notre code est évidemment de l'exécuter. En effet, grâce au makefile réalisé, il est possible de compiler notre programme de façon automatique, sans avoir à entrer manuellement les commandes de compilation à chaque modification du code. Cela nous a permis de gagner énormément de temps et d'assurer que le projet soit facilement compilable, même après de nombreuses modifications.

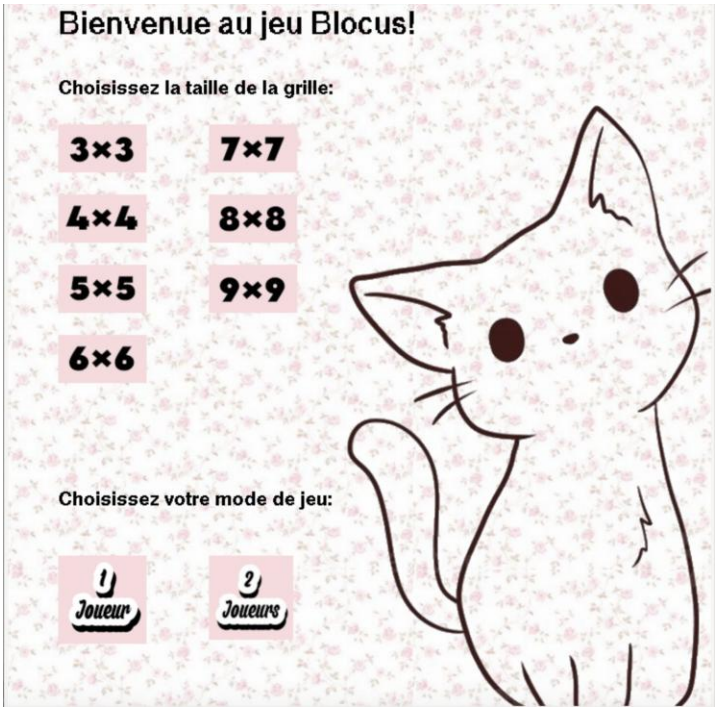
Les autres fonctionnalités dont le programme bénéficie sont les suivantes :

-Affichage du menu principal : Le menu principal est le point de départ du jeu et il permet aux joueurs de sélectionner le mode de jeu (2 joueurs ou 1 joueur) ainsi que la

taille de la grille (par exemple, 3x3 ou 4x4). Ce menu est affiché dans une fenêtre de dimensions raisonnables (700x700), choisies après quelques tests pour éviter que l'affichage soit trop petit ou trop grand. Pour créer ce menu, nous utilisons la fonction InitialiserGraphique() de la bibliothèque graphique, qui prépare l'environnement pour afficher la fenêtre du jeu, avec la fonction CreerFenetre().

-Charger et afficher des sprites: Une des fonctionnalités clés de notre jeu est l'utilisation d'images, ou "sprites", pour représenter les boutons interactifs dans le menu principal et les options de jeu. Nous avons opté pour cette méthode afin de rendre l'interface plus visuelle et interactive. Grâce à l'utilisation des sprites, chaque bouton (comme ceux permettant de choisir la taille de la grille ou le mode de jeu) devient un élément cliquable. Pour ce faire, nous avons utilisé la fonction ChargerSprite() pour charger et afficher les images dans la fenêtre graphique. Ces sprites sont ensuite rendus interactifs grâce à la gestion de la souris, permettant au joueur de cliquer pour effectuer son choix.

Les boutons interactifs représentent différentes tailles de grille (comme 3x3, 4x4) et les modes de jeu (1 joueur, 2 joueurs). Grâce à cette approche visuelle, le joueur peut rapidement s'orienter dans les options proposées sans se perdre dans des menus textuels.



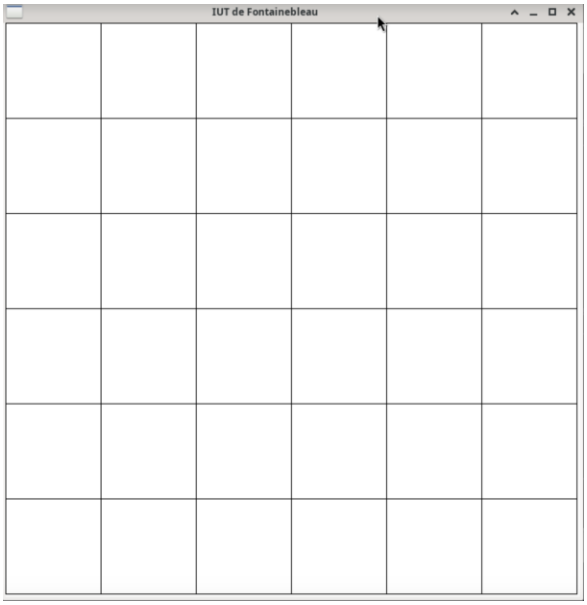
.

-Gestion de l’interface avec la souris : L'un des éléments essentiels dans ce processus est la fonction SourisCliquee(). Cette fonction permet de tester si un bouton de la souris a été cliqué. Elle renvoie soit 1 (si un clic est détecté), soit 0 (sinon), ce qui permet de savoir si l'utilisateur a interagi avec l'interface graphique. Lorsqu'un clic est détecté, les coordonnées du clic sont également récupérées et stockées dans les variables x et y. Cela nous permet de déterminer précisément où l'utilisateur a cliqué dans la fenêtre graphique. Ces informations sont ensuite utilisées pour vérifier si le clic se situe dans la zone d'un bouton interactif, et en fonction des coordonnées du clic, le programme exécute les actions appropriées.

-Initialisation de la grille : La grille est générée dynamiquement en fonction de la taille choisie par le joueur (par exemple, 3x3, 4x4, jusqu'à 9x9). La fonction InitialiserGrille() joue un rôle central dans cette étape. Définie dans le fichier grille.c et déclarée dans grille.h, elle permet d'initialiser toutes les cases de la grille à une valeur de 0, indiquant que les cases sont vides au départ. Le fonctionnement de la fonction repose sur deux boucles imbriquées utilisant les compteurs i et j, qui parcourent respectivement les lignes et les colonnes de la grille. Cette approche garantit que chaque case est correctement initialisée. Nous avons choisi cette convention de valeurs pour représenter l'état des cases de manière claire :

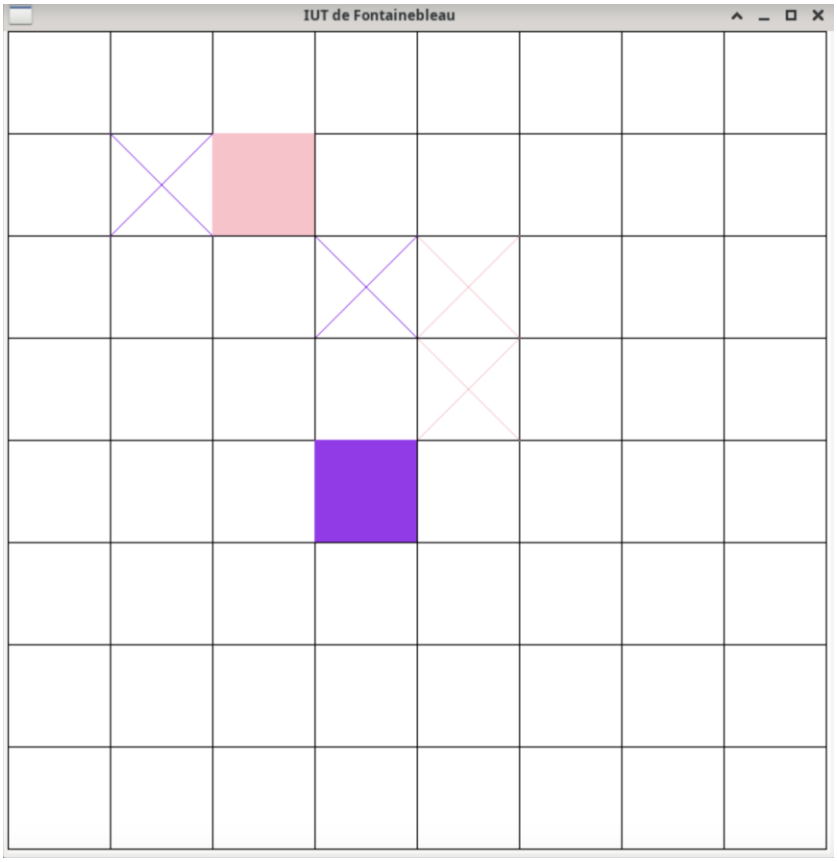
- Une case contenant la valeur 0 est vide (libre).
- Une case contenant la valeur 1 indique qu'elle est occupée par un pion.
- Une case contenant la valeur 2 signifie qu'elle est condamnée et ne peut plus être utilisée.

-Dessin de la grille de jeu : La grille de jeu selon la taille choisie se fait grâce à la fonction DessinerGrille( ) que nous avons déclaré dans DessinerGrille.h. Elle utilise la fonction de la bibliothèque graphique DessinerRectangle( ) pour tracer chaque case de la grille. Comme pour InitialiserGrille( ), elle utilise deux boucles implémentées avec i et j parcourant les lignes et les colonnes de la grille.



-Placer son pion : Grâce à la fonction PlacerPionInitial(), définie dans le fichier PlacerPionInitial.c, le joueur peut placer son pion (représenté graphiquement par une case colorée en rose ou en violet avec la fonction RemplirRectangle()) sur la grille, à condition que la case sélectionnée soit vide (valeur 0). Cette fonctionnalité vérifie systématiquement la validité du placement avant de mettre à jour l'état de la grille. En plus de gérer le placement manuel pour une partie en mode deux joueurs, cette fonction prend également en charge le placement automatique des pions par le bot si le mode solo est sélectionné.

-Gestion des tours des joueurs : Lorsqu'un joueur doit déplacer son pion, le programme détecte la case choisie grâce au clic de souris. La fonction EffacerCase() est alors appelée pour réinitialiser la valeur de cette case à 0, indiquant qu'elle est à nouveau vide et disponible. Une fois la case libérée, elle peut être réutilisée pour placer un pion ou pour être condamnée, selon les actions du joueur. Pour actualiser visuellement la grille, la fonction RemplirRectangle() est utilisée pour recolorier la nouvelle case où le joueur décide de déplacer son pion. Cependant, les déplacements des joueurs ne sont pas libres : ils doivent respecter les règles du jeu. C'est ici qu'intervient la fonction EstAdjacente(). Cette fonction vérifie si la case choisie est adjacente à celle où se trouve actuellement le pion. Si la case est valide, le déplacement est autorisé ; sinon, il est bloqué.



-Jouer contre un “bot” : Le programme offre la possibilité de jouer contre une intelligence artificielle. Celle-ci a été développée en utilisant la fonction rand() pour générer des coordonnées aléatoires, tout en s’assurant qu’elles restent pertinentes grâce à une combinaison avec la fonction EstAdjacente(). Cette approche permet au bot de respecter les règles du jeu en plaçant son pion uniquement sur une case adjacente à sa position actuelle. Pour rendre l’interaction plus naturelle, nous avons ajouté un léger délai avant que l’IA ne joue, à l’aide des fonctions de gestion du temps. Cela simule un comportement humain et évite que le bot ne réagisse de manière instantanée.

-Vérifier si un des joueurs a gagné : La victoire est déterminée lorsqu’un joueur est totalement bloqué, c'est-à-dire qu’il ne peut plus déplacer son pion. Pour cela, la fonction EstBloque() entre en jeu. Cette dernière analyse les huit cases adjacentes au pion du joueur (grâce à la fonction EstAdjacente()) pour vérifier si elles sont toutes occupées, c’est-à-dire différentes de 0. Si aucune case valide n’est disponible, la fonction confirme que le joueur est bloqué, et son adversaire est alors déclaré gagnant.

-Afficher une page finale : Le fichier AfficherFin.c gère l'affichage de l'écran final, qui annonce le vainqueur grâce à un simple appel à la fonction EcrireTexte(). Cette page inclut également des options permettant au joueur de rejouer ou de quitter le jeu. Dans le fichier main.c, une boucle while est utilisée pour contrôler la relance ou l'arrêt du programme. La variable recommencer, initialisée à 1 au début, permet au jeu de tourner jusqu'à ce qu'une condition de fin soit atteinte. Si le joueur choisit de rejouer en cliquant sur le bouton approprié, la valeur de recommencer reste à 1, et la boucle continue de s'exécuter pour relancer une nouvelle partie. En revanche, si le joueur clique sur "Fermer", la variable recommencer est mise à 0, ce qui arrête l'exécution de la boucle et ferme le jeu grâce à la fonction FermerGraphique(). Enfin, lorsque l'un des joueurs est bloqué et ne peut plus se déplacer, la partie se termine automatiquement. Un message final s'affiche pour désigner le gagnant.

Affichage de la 3ème fenêtre dans le cas où le joueur 2 (pion violet) gagne.

Affichage de la 3eme fenêtre dans le cas où le joueur 1 (pion rose) gagne.

Toutes ces fonctionnalités ont été développées dans le but de respecter les règles du jeu. Chaque aspect, de l'affichage dynamique de la grille à la gestion des déplacements et des victoires, a été pensé pour offrir un jeu fonctionnel et agréable à utiliser.

V. Structure du programme :

Le code contient 12 fichiers sources (.c) avec le code exécutable :

-main.c qui gère la boucle principale du jeu.

-menu.c qui présente le jeu ainsi que ses différents paramètres.

-grille.c qui initialise la grille à 0.

-PlacerPionInitial.c qui donne le premier placement des pions.

-gestion_joueur1.c gérant déplacements et condamnations du joueur 1 (pion rose).

-gestion_joueur2.c gérant déplacements et condamnations du joueur 2(pion violet).

-Estbloque.c qui vérifie tout au long de la partie si un joueur est bloqué, c'est à dire qu'il est entouré de cases dont la valeur est différente de 0.

-EstAdjacente.c qui vérifie si une case est adjacente à une autre.

-EffacerCase.c est chargé de réinitialiser les cases précédemment occupées (celles où un pion était placé) en leur attribuant la valeur 0. Cela permet de rendre ces cases à nouveau "jouables».

-DessinerGrille.c qui dessine la grille, en fonction de la taille choisie.

-AttendreMicrosecondes.c qui donne de courts délais d'attente pour le tour du bot.

-AfficherFinDePartie.c gère l'affichage d'un menu de fin de partie. Cette dernière fenêtre annonce le vainqueur de manière claire et propose au joueur deux options : rejouer ou quitter le jeu.

Et 11 fichiers d'en-tête (.h) correspondant aux déclarations de fonctions :

-menu.h qui déclare fonctions et structures de données du menu.c

-grille.h, idem pour grille.c

-init.h , idem pour PlacerPionInitial.c

-gestion_joueur1.h

-gestion_joueur2.h

-EstBloque.h

-EstAdjacente.h

-EffacerCase.h

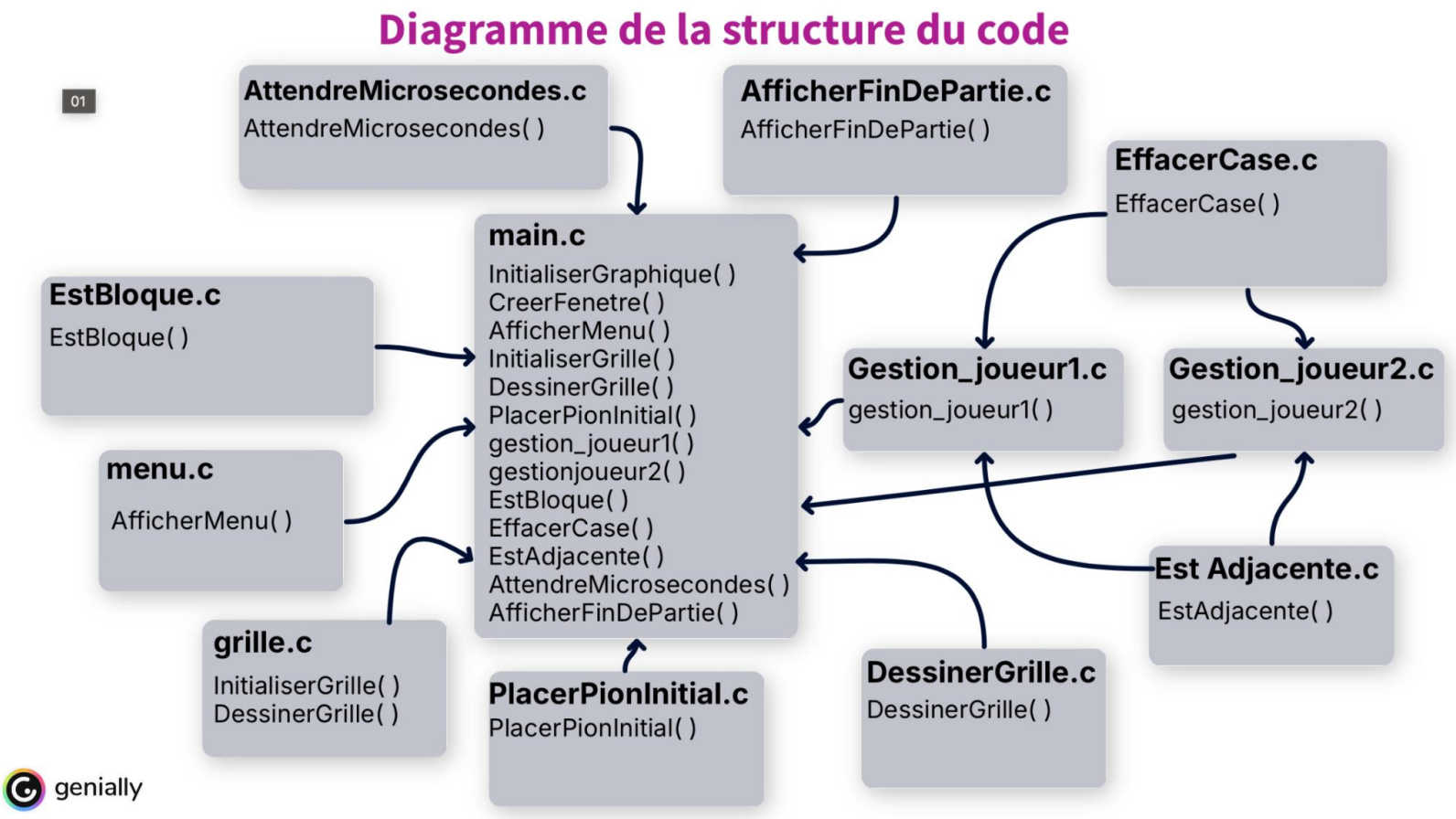
-DessinerGrille.h

-AttendreMicrosecondes.h

-AfficherFinDePartie.h

Chaque fichier d'en-tête contient les déclarations de son fichier.c. Le code est découpé en fonction des différentes étapes, au fur et à mesure de notre avancée. Cela a permis une bien meilleure organisation et ainsi éviter toute confusion.

Nous avons essayé de diviser ce code de façon pertinente, de la première à la dernière étape qui est la page de fin. L'objectif était aussi de savoir où se trouvait chaque élément et ainsi gagner du temps.



VI.Données:

Le menu principal est le point de départ du jeu. Il permet aux joueurs de choisir la taille de la grille ainsi que le mode de jeu, que ce soit pour jouer à 2 joueurs ou contre un bot. Dans le menu, deux variables sont importantes :

- int taille_valide = 0 : Cette variable indique si une taille valide a été sélectionnée. La boucle continue de s'exécuter jusqu'à ce que l'utilisateur fasse un choix valide.
- int mode_valide = 0; : Cette variable indique si un mode de jeu valide a été sélectionné (1 joueur ou 2 joueurs).

Le programme vérifie les coordonnées du clic (x, y) par rapport aux zones définies pour chaque option dans le menu. Si l'utilisateur clique sur une image correspondant à une taille de grille ou à un mode de jeu, les variables taille et mode_de_jeu sont mises à jour. Pour chaque taille de grille (de 3 à 9), le programme vérifie si le clic se trouve dans la zone correspondante, définie par les coordonnées x et y et la taille de l'image. Une fois qu'une taille est sélectionnée, taille_valide devient égal à 1, et la taille choisie est enregistrée dans la variable *taille. Le même principe est appliqué pour les modes de jeu (1 joueur ou 2 joueurs). Le programme vérifie si le clic est dans la zone de l'image correspondant à chaque mode et met à jour la variable mode_de_jeu. Ainsi :

- int* taille : Cette variable stocke la taille de la grille choisie par l'utilisateur (valeurs entre 3 et 9).
- int* mode_de_jeu : Cette variable indique le mode de jeu sélectionné (1 ou 2 joueurs).

Le programme commence donc par afficher un menu via la fonction AfficherMenu(&taille, &mode_de_jeu). Une fois que les paramètres sont récupérés, la taille de chaque case est calculée. Ce calcul se fait dans main.c, avec la formule suivante : taille_case = 700 / taille ; (car la taille de la fenêtre graphique est fixée à 700 pixels). Cette valeur détermine la taille de chaque case en pixels sur l'écran. Ensuite, la fonction DessinerGrille(int taille, int taille_case) est appelée pour dessiner la grille sur l'écran. Pour ce faire, deux boucles imbriquées sont utilisées :

- La boucle extérieure parcourt les lignes de la grille, allant de i = 0 à i = taille.

- La boucle intérieure parcourt les colonnes de chaque ligne, allant de j = 0 à j = taille.

Pour chaque combinaison de i et j, un rectangle représentant une case est dessiné à l'écran.

Lors de l'appel de la fonction InitialiserGrille, la grille (int grille[9][9]) est initialisée. Cela signifie que, par défaut, toutes les cases de la grille sont vides. Au fur et à mesure que le jeu progresse, les valeurs dans ce tableau 2D seront modifiées pour refléter l'état de chaque case selon trois entiers:

- 0 : Si la case est vide
- 1 : Lorsqu'un joueur place un pion sur une case, ce qui indique que la case est occupée par un
  pion.
- 2 : Lorsqu'une case est condamnée (par exemple, lorsqu'un joueur bloque une case), et ne peut plus être utilisée.

Les variables pos_rose[2] et pos_vio[2] sont deux tableaux stockant les positions respectives des pions rose et violet dans la grille (les coordonnées sont enregistrées sous forme de colonnes et lignes). Lorsque l'utilisateur clique, les coordonnées du clic sont récupérées via _X et _Y (variables fournies par la bibliothèque graphique utilisée). Ces coordonnées sont ensuite converties en indices de la grille (colonnes et lignes) en divisant les coordonnées par la taille de la case. Le programme vérifie si la case est libre en consultant grille[ligne][colonne] (en regardant si sa valeur est de 0). Lorsque l'utilisateur déplace un pion, l'ancienne position du pion doit être réinitialisée en 0 pour indiquer qu'elle est désormais vide. Ensuite, la nouvelle case où le pion est déplacé doit être mise à 1 pour marquer qu'elle est occupée par un pion. Le tableau grille est mis à jour à chaque mouvement.

Page de fin : Elle s’ouvre dès qu’un des joueurs est bloqué. Elle utilise des variables simples telles que x et y correspondant aux coordonnées de la souris, mais aussi des sprites, des écritures, etc. Dans notre programme, nous avons utilisé des pointeurs, notamment avec la variable *recommencer dans notre fichier AfficherFinDePartie.c. Ce pointeur est une variable de type Int, initialisée à 1. Il permet de gérer l'action choisie par le joueur à la fin de la partie :

Si le clic est dans la zone "Rejouer", la valeur pointée par *recommencer reste à 1, indiquant que le joueur souhaite rejouer.

Si le clic est dans la zone "Quitter", la valeur pointée par *recommencer est mise à 0, ce qui entraîne la fermeture du jeu.

Pour le gagnant, nous avons créé la variable gagnante initialisée à 0 car aucun joueur n'a gagné au début du jeu. Grâce à la fonction EstBloque au cours de la partie, la variable gagnant reçoit 1 ou 2 en fonction de quel joueur gagne la partie. Si le joueur 1 gagne alors gagnant = 1. Ainsi, si gagnant = 1, le texte "Le joueur 1 gagne !" s'affiche, et inversement pour le cas où le joueur 2 gagne la partie, gagnant prendra la valeur 2.

Au sujet des sprites, comme l'indique la bibliothèque graphique, si le chargement des sprites échoue (mauvais format, mauvais chemin, etc…) la fonction retourne -1 et ferme la fenêtre graphique.

VII. Conclusion :

Maylee: Finalement, ce projet m'aura beaucoup stressée et mis la pression. Au final pour une bonne raison car cela nous a empêché de procrastiner. A première vue, la bibliothèque graphique de l'iut permet vraiment de faire de belles choses, on s'est beaucoup amusées avec Sarah sur la partie décoration. Au sujet du code, je dirai que ce projet m'aura fait intégrer encore plus de notions de développement en C, même si nous les avons vues en TP, c'est un bon entraînement. Je suis quand même fière du travail que Sarah et moi avons fourni car tout s'est bien passé finalement et je dirai même que je suis satisfaite de notre jeu de blocus. Quelque chose qui nous a énormément aidé à avancer rapidement est que nous passions la majorité de notre temps de travail ensemble à l'IUT. La communication était bien plus facile et nous avions donc moins de travail à la maison.

Sarah : Ce projet a marqué une étape importante dans mon apprentissage de la programmation, me permettant de mettre en pratique de nombreux concepts étudiés en cours. Certaines parties, comme la gestion des déplacements, ont présenté des défis, mais j'ai réussi à les surmonter grâce à une bonne organisation et une collaboration efficace avec Maylee. Comparé aux projets que j'ai pu réaliser lors de ma formation précédente, je suis particulièrement satisfaite du résultat cette fois-ci. Le fait d'anticiper et de nous organiser correctement nous a permis de consacrer du temps aux détails et de proposer une version du jeu bien plus aboutie. Cette expérience m'a également montré l'importance de la planification et de la collaboration. En résumé, travailler sur

Blocus a renforcé mes compétences en C, notamment dans la gestion des entrées/sorties et l’optimisation des algorithmes, tout en m’apprenant à mieux planifier et à travailler efficacement en équipe.