Projet jeu de dames

Réalisé par : *FAHSI Hajar*

KRIOUILE Mouaad

GANGAT Bilal

I. Introduction :

Ce projet est un programme C qui offre à l'utilisateur plusieurs fonctionnalités liées au jeu de dames. Nous avons choisi de se réunir tous pour écrire le programme afin de rendre la compilation plus rapide et de s'entraider au cas où l'un de nous rencontre une difficulté.

Le report de date de remise de travail nous a beaucoup aidé, on avait donc l'opportunité de se voir plusieurs fois par semaine durant les vacances. Ça nous a permis de travailler plus rapidement et on a pu s'aider plus facilement.

II. Répartition de travail :

NOM, Prénom	Travail réalisé
Fahsi Hajar	*1.3 La liste des configurations.* *Les fonctions : Affiche_ plateau, partie_detruire, partie_sauvegarder, Partie_charger, partie_jouer, replay_jouer.*
Kriouile Mouaad	*1.2 La liste des coups joues* *Les fonctions : annuler_mouvement, partie_creer, partie_nouvelle, replay_charger, deplacement_valide.*
Gangat Bilal	*la partie 1.1 : les pièces des joueurs.* *Les fonctions : case_vide, modifier_case, saisie case, changer_joueur, modif_damier et la structure partie1.4.* *Le makefile et le main.c*

III. Les difficultés rencontrées :

Durant le projet nous avons rencontrés des problèmes, mais nous avons réussi à en trouver les solutions. Les problèmes majoritaires ont étaient :

-Un moyen de faire bouger les pièces sur le damier.

-**Saisie_case** : Nous avions eu du mal à trouver une technique pour la saisie des coordonnées et surtout pour les déplacements intermédiaires.

-**Partie_sauvegarder** : le fait d'insérer le damier dans le dossier plateau et la partie dans le dossier partie a été difficile car nous n'avions pas vu ça en cours.

-**Deplacement_valide** : cette fonction a été l'une des plus compliquées car elle était longue et demandait beaucoup de conditions, nous nous sommes donc inspiré de plusieurs algorithmes sur internet

-**General** : si nous n'avions pas utilisé ses fichiers, la plupart des fonctions du programme serait beaucoup plus long et illisible.

-Comment lire un fichier et remplir les variables afin de charger la partie.

-Comment créer un système facile pour rejouer le replay.

Certains problèmes sont expliqués dans la partie IV.

IV. Explication des fonctions et résolutions des problèmes :

general.h general.c

C'est des fichiers qui contiennent l'ensemble des fonctions et de structures dont nous nous sommes servi pour définir quelques nouveaux paramètres, faciliter le programme et le rendre plus compréhensible, nous allons expliquer brièvement l'utilité des fonctions et structures.

Le fichier **general.h** contient la déclaration des fonctions utilisé dans general.c de quelques bibliothèques dont nous avons besoin et une structure coord qui représente une coordonnée composé de deux positions x et y.

Le fichier **general.c** contient la fonction effacer écran qui est tout simplement une fonction pour effacer l'écran, son utilité est particulièrement dans la fonction afficher plateau, pour effacer l'écran afin d'afficher un autre plateau.

La fonction **attendre()** est une fonction qui met le programme en attente jusqu'à ce que l'utilisateur clique entrée.

La fonction **caractere_valeur** est pour dégager la valeur du caractère digit, C'est à dire prendre les valeurs (1-9) qui représentent des caractères ASCII et les rendre des entiers (int).

La fonction **nettoyer_chaine** remplace le caractère '\n' par '\0' pour assurer que l'utilisateur a fini d'écrire, et qui lit la commande entrée par l'utilisateur si elle est tapée en majuscule, on a utilisé la fonction (**tolower)** qui convertit n'importe quelle chaine saisie en minuscule si elle est entrée en majuscule, pour cela on a dû appeler la bibliothèque 'ctype.h'

La fonction **coord_creer** prend en argument deux entiers x et y et retourne une coordonnée à partir de deux position x et y.

La fonction **coord_direction_du_mouvement** qui prend deux arguments (coordonnées initiales et finales) et une cordonnée facteur qu'on lui donne la valeur de la différence entre les coordonnées initiales et les coordonnées finales de X puis Y.

Si la valeur absolue du facteur de X et du facteur de Y sont inégaux alors ce n'est pas un déplacement diagonale (pour calculer la valeur absolue on s'est servi de la fonction abs()), sinon on fait des tests pour voir si la direction est à droite ou à gauche grâce à la valeur du facteur de X et de Y (si la valeur du facteur X est positif alors le déplacements est à droite et on lui affecte la valeur 1 sinon le déplacement est à gauche et on lui affecte la valeur -1, si la valeur du facteur Y est positif alors le déplacement est vers le haut et on lui affecte la valeur 1,sinon le déplacement est vers le bas et on lui affecte la valeur -1).

Toute utilisation d'une des fonctions au-dessus dans une autre fonction sera signaler et expliquer dans les fichiers qui correspond.

Piece.c piece.h

Dans le fichier piece.h on a du ajouter la structure **piece_capture**, qui représente une pièce capturée, et se constitue d'une position des cordonnés de la pièce capturée (pour cela on a utilisé la structure coord de fichier general.h) et d'un booléen (statuts) qui permet de savoir s'il a eu une promotion ou pas. Au début nous nous ne sommes pas rendu compte de besoin de cette fonction, mais en avançant dans le programme nous avons décidé de la créer, précisément dans la fonction deplacement_valide.

Dans le fichier piece.c nous avons respecté la façon dont on nous a demandé d'écrire les fonctions, dans la fonction **piece_caractere** nous voulons trouver une méthode plus rapide pour convertir le caractère associé au joueur en majuscule si c'est le deuxième joueur, alors pour cela on a utilisé la fonction 'toupper ' qu'on a trouvé sur internet.

Dans la fonction **piece_identifier**, le cas où le caractère est non valide, c'est-à-dire une ' ' ou '.' alors le joueur prends la valeur -1 et son statut est ' False ' , le joueur -1 servira alors comme méthode pour vérifier les pièces dans le damier.

Partie.h Partie.c

Partie.h

Pour pouvoir stocker la suite des coups joués depuis le début de la partie on avait besoin de créer la **structure déplacements** (qui pointe sur l'élément précédant et l'élément suivant) , cette dernière contient les coordonnées de la position initiale et final et un tableau de cordonnées des positions intermédiaires qui a le nombre de positions intermédiaires comme ta taille, et un autre tableau de cordonnées pour définir le nombre de pièces capturées et sa taille (nombre de pièces capturées). Et un bool pour la promotion de la pièce après le déplacement.

La structure **Liste coups** est une liste doublement chainée qui contient les premiers déplacements et le déplacement final et le nombre de déplacements.

A ce qui concerne la configuration, on a créé d'abord la **structure Configuration** qui contient un damier sous forme d'un tableau de deux dimensions (10x10) et deux pointeurs vers la même liste (esuivant et eprécèdent) pour sauvegarder les configurations successives.

La liste doublement chainée **Liste config** sert à stocké la suite des configurations successives depuis le début de la partie, on a également utilisé deux pointeurs sur la structure configuration. Et un nombre de configuration.

Dans la structure **Partie** on a ajouté un bool plateau charge (qui nous indiquera si le tableau a était chargé à partir d'un fichier plateau). Son utilité sera expliquée dans les prochaines fonctions.

Partie.c :

La fonction **case_vide () :**

Prends en arguments un pointeur vers la partie et les cordonnée d'une position et vérifie si la case contient un point ou non, dans la fonction **modifier_case** on a respecté les instructions, la fonction modifie le damier à l'aide de la fonction piece_caractere.

La fonction **changer_joueur () :**

Change le joueur qui a le trait en inversant la valeur du booléens.

Dans la fonction **afficher_plateau()** :

On s'est permit d'ajouter un pointeur vers la structure configuration, la fonction vérifie au premier temps la liste de configuration, si elle est vide alors elle affiche le plateau d'après le damier courant sinon elle affiche le plateau de la configuration précédente, on a dû faire ce changement quand on est arrivé à la partie **replay**, on devait offrir à l'utilisateur la possibilité de rejoué une partie enregistrée en imprimant les instances du dernier damier joué.

La fonction **modif_damier()** :

prend en arguments un pointeur vers la structure Partie et un pointeur vers la structure déplacement, Cette fonction était un peu compliquée pour nous, vu qu'elle consiste à faire la partition des cas. On a aussi pensé à ajouter la liste des configurations pour simplifier la tâche dans le reste du programme (on enregistre le damier dans la liste de configuration avant la modification grâce à la fonction memcpy ( )). Apres avoir identifié la piece_courante à l'aide de la fonction piece_identifier , On supprime la case de la pièce joue et puis les pièces capturées.

Pour modifier la case de la position finale il y avait deux cas, si la pièce est promu après le déplacement , on crée alors une nouvelle pièce au joueur mais promue, dans l'autre cas on pose le même caractère de la pièce jouée.

Et puis on ajoute les déplacements et la configuration dans les listes doublement chainées en respectant les cas possible (s'il s'agit des premiers déplacements ou s'il s'agit du deuxième déplacements ou plus) tout en augmentant le nombre de déplacements.

Dans la fonction **annuler_mouvement** on a utilisé la méthode de TP pour pouvoir annuler un mouvement (on a précisé entre deux cas, s'il y a un déplacement précédant ou pas).

Une des fonctions les plus délicat était **partie_sauvegarder**, car au début on n'avait pas les bases nécessaires pour la gestion des fichiers, vu le décalage du cours de projet, mais après on s'est débrouiller en faisant des recherches sur internet. Voici donc le déroulement de la fonction, on a commencé par récupérer le nom du fichier (choisi par l'utilisateur) pour cela, on a utilisé la fonction nettoyer chaîne, et **sprintf** qui précise le nom du fichier et son chemin et met le résultat dans la chaîne **'fichier_chemin_complet'**, puis on a créé le fichier en mode écriture **(w)** à l'aide de la fonction **fopen**. On a utilisé la même méthode pour les deux fichiers.

Le fichier se terminant par « .plt » contient alors **PL** à l'entête et le fichier se terminant « .part » contient **PR.**

Le fichier.plt contient les damiers enregistrés depuis le début de la partie, et les fichiers .part contiennent le nombre des positions et les coordonnées des positions initials, intermédiaires et finales.

La fonction **partie_charger () :**

sert à charger un plateau à partir d'un fichier plateau, elle créé une nouvelle partie et commence à la remplir alors, colonne par colonne à partir du fichier, à l'aide d'une boucle while qui vérifie la

condition que la partie a été chargée d'après un fichier plateau, si la condition n'est pas vérifiée le programme ferme le fichier. Dans la boucle while, si on trouve un caractère invalide (différent des caractères que contient le damier ou de (‘’ \n")) le programme affiche une erreur et détruit la partie créer.

La fonction **partie_nouvelle ()** :

Place les pions sur le damier, on s'est inspirée par plusieurs algorithmes sur internet, pour réussir à former l'algorithme le plus simple:

Le placement des pions se fait sur plusieurs étapes :

Si on est situé avant la quatrième ligne et après la 6eme ligne, on place des pions sur les cases qui disposent des cordonnées X et Y, et dans la 5éme ligne et la 6éme ligne on place des '.' sur les cases qui disposent aussi de cordonnées X et Y paires.

Le même algorithme est appliqué au cases qui disposent de cordonnées X et Y impaires.

La fonction **partie_jouer () :**

Cette fonction consiste la boucle principale de jeux ; elle offre plusieurs fonctionnalisé à l’utilisateur.

On a ajouté avant une fonction qui teste si la partie est terminée et une fonction qui propose à l’utilisateur de sauvegarder la partie avant de quitter. **Partie_terminee () ,** cette dernière permet de savoir si la partie est terminée, en analysant toutes les pièces du damier, si un joueur n'a plus de pièce alors la partie est terminée. **Proposer_sauvegarde ()** cette fonction demande à l'utilisateur s'il souhaite sauvegarder la partie. Si le joueur ferme sans avoir joué alors le programme affiche qu'il n'y a rien à sauvegarder, l'utilisateur a également le droit d'annuler le sauvegarde et continuer la partie.

La fonction **partie_jouer** fait appel à plusieurs fonctions du projet, au plus des deux fonctions ci-dessus.

Tant que la partie n'est pas terminer (cf. partie_terminee) on efface à chaque fois l'écran, en utilisant la fonction effacer_ecran (), on affiche le plateau alors, et on attend que l'utilisateur saisie une commande (cf. saisie_case)

Si La commande entré est **"fermer",** on propose à l'utilisateur de sauvegarder la partie grâce à la fonction propose_sauvegarder(), si la partie est sauvegardé alors la partie est terminé.

Si la commande entrée est **"annuler",** on appelle la fonction annuler_mouvement si la liste des coups joués n'est pas vide, dans le cas contraire le programme affiche ‘la liste des coups joués est vide’.

Si la commande est vide on reprend la boucle.

Si la commande n'est pas valide on informe l'utilisateur et on attend une nouvelle réponse.

Si l'utilisateur entre des cordonnées, on teste si le déplacement est valide (cf. deplacement_valide) si c'est le cas on appelle la fonction modifier_damier (), si c'est la fin de la partie, on affiche le joueur qui a gagner, et on propose le sauvegarde de la partie, mais si le déplacement n'est pas valide on libère la mémoire du déplacement qu'on a alloué avant que l'utilisateur entre des cordonnées, et puis on reprend la boucle.

Afin d'offrir la possibilité de rejouer une partie enregistrée dans un fichier de type ´ .part ' on a créé les deux fonctions replay_charger et replay_jouer.

La fonction **replay_charger () :**

Cette fonction prend en argument un fichier, elle crée une nouvelle partie puis lit les cordonnées contenus dans le fichier, elle vérifie si les déplacements sont corrects et si c'est le cas elle modifie le damier par rapport aux déplacements lus dans le fichier et si le déplacement est invalide alors la fonction affiche une erreur, libère la mémoire et détruit la partie crée.

La fonction **replay_jouer () :**

On commence par afficher la première configuration, c'est-à-dire le premier damier joué, puis on donne à l'utilisateur le droit de circuler entre la configuration c'est-à-dire avancer déplacement par déplacement ([S]suivant / [P]recèdent / [C]ontinuer la partie / [F]ermer) jusqu'à ce qu'il choisit le damier dont veut commencer, l'utilisateur a aussi le droit de fermer le fichier.

Dans le cas [S]uivant, Si on est arrivé au dernier déplacement, il faut afficher le damier final

Mais ce dernier ne se trouve pas dans la liste de configuration mais plutôt dans la configuration->eSuivant alors la fonction **afficher_plateaux()** va afficher le dernier statut du damier.

Pareil dans le cas [P]recèdent, Si la configuration est null (alors le dernier damier après le dernier déplacement est afficher alors l'élément précèdent est la dernière entrée dans la liste de configuration. sinon on charge la configuration précédente.

Le cas [C]ontinuer et [F]ermer permettent à l'utilisateur de continuer à jouer la partie courante ou de la terminer.

Pour la fonction **deplacement _valide (),** on a besoin de saisir des arguments qui sont nécessaires pour récupérer toutes les informations qui vérifie les conditions du deplacement, un tableau de cordonnées du deplacement, le nombre de deplacement, et un pointeur sur un deplacement.

La fonction est divisée par deux parties nécessaires, parlant du premier cas :

Si la pièce est promue:

Tour d'abord on vérifie si c'est un deplacement ou prise, cela est fait en analysant chaque case entre la position initiale et la position finale, si aucune position intermédiaire est détecté alors c'est juste un deplacement, sinon entre chaque déplacement intermédiaire il faut trouvé une et seulement une pièce adversaire, on utilise la fonction **direction_du_mouvment (cf. general.c)** pour savoir si le déplacement est diagonale, on vérifie si la case suivante est vide, si elle n'est pas vide, on identifie la pièce et ensuite on vérifie si la pièce n'appartient pas au joueur qui a le trait, si la condition est vérifié, on vérifie si il n'y a pas de pièce adversaire trouve avant celle-là, si c'est le cas le déplacement est invalide, sinon on vérifie si la pièce n'est pas marqué comme prise, si c'est le cas le déplacement est invalide, on enregistre la position de la pièce capture, est on mentionne qu'on a trouvé une pièce adversaire, si il n'y a aucune pièce adversaire alors c'est un déplacement normal donc le nombre deplacement est égale à un, si c'est pas le cas le déplacement est invalide.

Si la pièce est non promue:

On vérifie si le premier déplacement est un avancement ou une prise. au cas d'un avancement la position X et Y doivent être différentes de l'initiale d'une case diagonale, en cas d'une prise les positions X et Y doivent être différentes de l'initiale de deux cases diagonales uniquement dans chaque deplacement, si c'est un avancement on doit vérifier qu'il est toujours vers le cote de l'adversaire, si c'est le cas on vérifie que la case est vide, si c'est pas le cas le déplacement est invalide, si c'est un avancement mais le joueur a entré plus que deux cordonnées le déplacement est invalide.

Si le déplacement est une prise, on vérifie si la case suivante après la pièce à prendre est vide, si ce n'est pas le cas et la pièce est du même joueur ou autre obstacle alors le déplacement est invalide, on vérifie si la pièce capturée est une pièce valide de l'adversaire, si c'est le cas on vérifie si cette dernière n'a pas été marquée pour prise, si c'est le cas le déplacement est invalide, sinon on enregistre la position de la pièce capturée. Les déplacements sont prises valides, alors on copie la liste des positions dans la structure déplacement qui sera ajouté à la liste doublement chainée, on vérifie si c'est une promotion et on marque la promotion dans le déplacement.

Saisie.h Saisie.c

Pour pouvoir créer le jeu correctement, la machine doit être en mesure de lire les commandes entrées par l'utilisateur. Elle doit également vérifiée si les commandes sont valides ou non. Pour cela, nous avons créé la structure Resultat_saisie, cette structure permet de vérifiée la commande entrée puis d'exécuter cette commande.

Pour ce faire, nous lui avons d'abord attribué un bool qui vérifie si la commande est valide. Puis un pointeur char qui vérifie si la commande entrée correspond aux coordonnées permettant de jouer ou s'il s'agit d'une commande permettant d'annuler le coup ou de fermer la partie.

Pour les coordonnées nous avions besoin d'une int permettant de lire les positions x/y initiales et finales et nous avons également utilisé un pointeur pointant vers liste_coordonnees.

Pour finir nous avons ajouté ajouter une structure qui permet d'attendre la commande venant de l'utilisateur.

Saisie.c :

Saisie.c commence avec la structure permettant d'attendre la commande de l'utilisateur. Dans cette structure nous avons la déclaration des variables nécessaires et leurs initialisations. Les **bool** sont initialisés sur false, le **char** à NULL et la **int** à 0.

Une fois les variables nécessaires entrées, on place un while dans lequel on rentre uniquement si une commande est entrée.

Une fois dans la boucle, nous avons un fgets qui lit le contenu de la commande, un nettoyer_chaine de nettoyer la chaîne et on alloue de la mémoire.

Pour la lecture et l'exécution des commandes, nous avons utilisé un switch avec un système d'incrémentation. Plus précisément, si la commande entrée correspond aux coordonnées de x alors la case 0 incrémente **lecture_etape** à 1 (initialisé à 0) sinon **lecture_erreur** devient **true** et on sort de la boucle. De la même manière, case 1 correspond aux coordonnées de y et fonctionne de la même manière, c'est-à-dire **lecture_etape** passe à 2, dans le cas contraire, comme vu précédemment, lecture_erreur devient True et on sort de la boucle. La case 2 est différente car il a été fait pour lire l'espace qui sépare les coordonnées et le « \0 » qui marque la fin de la chaîne.

Pour finir, on a une commande qui permet d'enregistrer les coordonnées et une autre qui renvoie le résultat.

main()

Dans un premier temps, on détecte le type du fichier entré en argument, on ouvre le fichier, si l'en-tête du fichier et "PL" il s'agit d'un fichier plateau, si c'est "PR" il s'agit d'un fichier partie, sinon le fichier est invalide car l'en-tête du fichier n'a pas été reconnue.

On démarre une nouvelle partie, si la partie est terminée alors on affiche à l'utilisateur de cliquer entrer pour quitter.

Il y'a également des petites fonctions utiles pour le jeu, on a une fonction qui, au moment de quitter la partie, permet de la détruire si elle est inchangée. Le main permet également de terminer la partie et affiché 'programme terminer, veuillez cliquer su ENTREE pour quitter'.

Le guide du jeu est expliqué en détail dans le fichier notes.txt

V. Conclusion :

Ce projet fut très intéressant pour chacun d'entre nous puisqu'il nous a d'abord permis d'utiliser les connaissances que nous avons acquises durant notre année. Et de les mettre en application sur un projet concret.

De plus, c'est un projet qui nous a pris beaucoup plus de temps que prévu, il nous a donc demandé une bonne organisation, notamment au niveau de la répartition des tâches et du respect des délais que nous nous étions fixés. Ce projet nous a permis de comprendre l'importance de travailler en groupe (répartition du temps, élaboration d'un rapport, etc.) sur un projet de plus grande importance dans un domaine qui nous intéresse tout particulièrement. Ce projet a donc constitué un très bon entraînement permettant de se faire une idée du travail que peut demander le développement d'un projet du début à la fin. Ce fut donc une expérience très enrichissante pour chacun d'entre nous.

Nous vous remercions pour le décalage de la date final de remise car sans ce décalage nous n'aurions pas pu compléter le projet.

Nous vous remercions de votre lecture, FIN.