# PROJET FINAL : JEU D'ECHEC

## Introduction :

### But :

Le but du projet est de représenter un échiquier et les déplacements de pièce en cours de partie. Des améliorations peuvent être apporter comme l'amélioration de l'interface, la situation du roi, le respect des règles avancées….

## Le programme :

Les dimensions de la fenêtre de jeu sont de 12 cases sur 8 pour permettre d'afficher un échiquier de 8x8 et des informations sur le coter. Chaque case est affichée par 100 pixels.

-*coorClick* est un tuple qui contient les coordonnées x et y.

-*lstPiece* est un dictionnaire qui contient les coordonnées de toutes les pièces avec comme clé la première lettre du nom de la pièce suivie de sa couleur.

-*nomFichier* est une chaine de caractère.

-*moovedPiece* est une liste qui contient la liste des déplacements des pièces.

-*TourDe* est une chaine de caractère.

-*rock* est un dictionnaire où son stocker tous les rocks possibles avec comme clé ses coordonnées et comme valeur True ou False.

-*lstMoov* est une liste qui contient tous les mouvements possibles d'une pièce.

-*echec* est une chaine de caractère qui prends la couleur du joueur en échec.

-*nombreCoup* contient une valeur entière et à qui on ajoute 1 à chaque fois qu'un joueur joue.

-*Nj1* et *Nj2* sont des chaines de caractères qui contiennent le nom des joueurs.

-*sj1* et *sj2* contiennent des nombres décimaux ou des entiers suivant le score de chaque joueur.

-*looserName* est chaine de caractères qui contient le nom du perdant

-*Jeu* est un dictionnaire qui contient comme clé « pièce » qui a pour valeur une copie indépendante du dictionnaire lstPiece, le nom des joueurs, le score, …

-*piece* est une liste qui contient la liste des pièces.

-*coorDeplacement* est un tuple qui prends les coordonnés de la case d'arrivé de la piece

-*needPiece* est un booléen

-*CoorP* est un tuple qui contient les coordonnées de la pièce

-*ColorP* est une chaine de caractère qui contient la couleur d'un pion

-*startLigne* contient une valeur entière et qui prends les coordonnées en Y du pion

-direction contient une valeur entière

-*dicoX* est un dictionnaire qui « convertie » les coordonnées en X qui sont en chiffres en lettre pour que sa correspondent à un vrai échiquier

-*dicoY* est un dictionnaire qui « convertie » les coordonnées en Y de l'échiquier pour qu'elles correspondent à des coordonnées d'un vrai échiquier car elles sont inversées par rapport à notre plateau.

## Les fonctions utilisées :

La fonction *Case_vers_pixel* permet de transformer les coordonnées d'une case en coordonnés du pixel qui se trouve au centre de cette case. Cette fonction renvoie ainsi donc les coordonnées en pixel de la case.

La fonction *recup_case* permet de récupérer la case sur laquelle le joueur à cliquer en fonction des coordonnées en pixels du curseur. On peut dire que c'est l'inverse de la fonction précédente, case_vers_pixel. On définit x, y comme les coordonnées du click. On divise par 100 pour avoir la case ou le joueur clique. S'il clique au-delà de la case 7 c'est qu'il clique sur le menu

La fonction *click_MenuDebut* permet de gérer les cliques dans le menu qui s'affiche au lancement du jeu. Si on clique dans une certaine zone alors la variable menue est égale à False.

La fonction *click_Menu* permet de gérer les cliques dans le menu sur le coter de l'échiquier. Si on clique dans une certaine zone à droite de l'échiquier en jeu alors le jeu se ferme.

La fonction *deter_deplacement* permet de déterminer le déplacement d'une pièce donné grâce au module 'deterPiece' et renvoie la liste des déplacements possible pour la pièce sélectionner.

La fonction *suppr_echec_moov* permet de supprimer les mouvements impossibles lors d'un échec. Tout d'abord on copie les listes mooved_Piece et lstMoov dans deux listes différentes. « Try » et « except » permet d'exécuter une instruction après « try » et si une exception intervient, alors l'instruction est sautée et si le type d'exception levé correspond à un nom indiqué après le mot « except » correspondant alors l'instruction est exécuté, puis l'exécution continue après l'instruction « try ». Donc si la liste PieceBouger d'index 2 possède

un coup on copie le dictionnaire lstPieces et on met à jour la copie du dictionnaire. Mais si une erreur du type « IndexError » apparaît alors ajoute le déplacement possible de la pièce à la liste PieceBouger. Si la fonction deter_echec retourne la même couleur que TourDe alors on retire le déplacement impossible de la liste finalMoov.

La fonction *save_partie* permet de sauvegarder une partie dans un fichier texte. Tout d'abord on ouvre un fichier avec le nom du fichier qu'on a rentré dans un système de saisie en mode écriture. On écrit toute les informations d'une partie dans le document à l'aide de « save.write ». On a des boucles « for » pour sauvegarder les informations telles que les coordonnées des pièces ou la valeur des booléen des rocks.

La fonction *charger_partie* permet de charger une partie sauvegarder. On ouvre le nom du fichier rentrer dans la saisie. On lit la première ligne sans rien faire car on sait qu'il est écrit « Jeu ». On récupère toutes les informations qui touches les joueurs comme leur pseudo, leur nombre de coups et leurs scores. On récupère les booléen des rocks avec une boucles « for ». On récupère la couleur des pièces et on transforme leur coordonnée pour qu'elles correspondent aux coordonnées de notre jeu et on les ajoute au dictionnaire des pieces

La fonction *recup_piece* permet de regarder si une pièce se trouve sur la case ou vient de cliquer le joueur, si oui on retourne la pièce et son indice dans la liste des pièces sinon on ne retourne rien. On regarde chaque pièce et ses coordonnées dans la liste des pièces. Puis pour chaque coordonnée dans la liste des coordonnés des pièce et si les coordonnées du click est égale à l'une des coordonnés d'une pièce on retourne cette pièce et ces coordonnées

La fonction *maj_dic* permet de mettre à jour le dictionnaire. Dans un premier temps on retire l'ancienne coordonnés avec « remove » et on ajoute la nouvelle avec « append ». Et enfin on vérifie si une pièce a été mangé.

La fonction *maj_rock* permet de voir si le rock est toujours possible. Aux échecs on peut rocker que si le Roi et la Tour n'ont pas bouger et qu'aucun des deux n'est menacé ou sera menacé. On regarde donc tout simplement dans la liste des coups si le roi à bouger s'il a bougé, la variable rock est égal à False. Pareil pour la Tour.

La fonction *deplace_rock* permet de rocker, Dans une première condition on regarde si le roi n'a pas bougé puis dans une seconde condition on vérifie qu'il passe de sa case d'origine à la case de petit ou grand rock alors la tour change elle aussi de place pour se placer à sa gauche.

La fonction *dessiner_menu* permet d'afficher des informations comme le tour de la personne à jouer, le score ou le bouton pour quitter le jeu qui fonctionne avec la fonction click_menu. Pour les champs de saisie on a utilisé tkinter et on a modifier fltk en modifiant le renvoie du canvas créer lorsqu'on créer une fenêtre.

La fonction *deplacer_piece* permet de regarder si la case sur laquelle le joueur souhaite déplacer sa pièce est déjà occuper par une pièce de sa couleur ou non. Si c'est une de ses pièces qui se trouve sur cette case alors c'est cette pièce qui devient la pièce à déplacer sinon la pièce se déplace sur cette case. Si moovedPiece est égale à « None » alors celle-ci est égale aux valeurs retourner par recup_piece. Si moovedPiece n'est pas égale à « None » et que moovedPiece est égale à sa couleur. Alors on return « False et la liste des mouvements des pièces ». Pour chaque pièce dans la liste de pièce, si les coordonnées de la clique sont

dans la liste des pièces, on regarde si les pièces sont de la même couleur et si oui on retourne « False » et la liste des mouvements des pièces. On ajoute les coordonnés de la clique à moovedPiece et on retourne « True » et la liste des mouvements des pièces.

La fonction *manger_piece* permet de regarder si la case d'arriver de la pièce voulant être déplacer contient une pièce de la couleur opposée. Si oui on supprime la pièce de la couleur opposée de la liste des pièces sinon on ne fait rien. Pour chaque pièce et coordonnés des pièce dans la liste des pièces, On regarde les coordonnés de du click dans la liste des cordonnées des pièces, si dans la liste des mouvements des pièces d'index 2 est égale au cordonnées du click et que le mouvement d'index 1 n'est pas égale à la pièce.

La fonction *deter_echec* permet de déterminé s'il y un échec. On définit les variables attackColor et echecColor par blanc et noir. Pour i dans un intervalle de 2, on regarde chaque pièce et coordonnés dans la liste des pièces pour chaque tuple de coordonnés de pièce dans un liste de coordonnés de pièce. Si tout ça existe c'est qu'il y a échec et coorR est égale à position du roi. Si la couleur de la pièce est égale à la variable attackColor alors la liste des coups possible est actualisé et si les coordonnés du roi se trouves dans la liste de coups alors on retourne la couleur de la personne en échec. On inverse les valeurs attackColor et echecColor car ça sera au noir de jouer. Si on a n'a pas retournée une couleur alors c'est qu'il n'y a pas d'échec donc on retourne « None »

La fonction *dessine_pieces* permet de représenter les pièces sur l'échiquier en fonction de la liste des pièces sur le plateau

La fonction *dessiner_moov* permet de dessiner des mouvements possibles à l'aide de petit cercle en récupérant la liste des mouvements possible de la pièce sélectionner

La fonction *dessiner_echiquier* permet de représenter l'échiquier en alternant les couleurs des cases comme sur un vrai échiquier.

La fonction *regarde_piece* permet de savoir s'il y a une pièce sur la case sélectionner. On peut aussi récupérer la pièce qui se trouve sur la case en question

La fonction *mat_ou_pat* permet de déterminé s'il y a échec et mat ou si la partie est nulle. Pour chaque pièce et coordonnés de pièce dans le dictionnaire et pour tuple des coordonnés de pièce dans la liste des coordonnés des pièce, on vérifie que la pièce est de la couleur de la personne qui doit jouer et alors la liste des coups possible est égale au nouveau coup possible. Si la liste des coups est vide on retourne « None ». Si l'échec est égal à la couleur de la personne qui doit jouer alors ou retourne la couleur de la personne qui joue sinon on retourne le mot pat. Pat signifie la nulle, c'est qu'un joueur n'a plus de coup à jouer alors que la situation d'échec et mat correspond à une mise en échec et être dans l'impossibilité de jouer un coup légal

La fonction *restartPartie* contient toutes les variables dont on a besoin pour relancer une partie qui a été jouer.

La fonction *deterRock* permet de déterminer le mouvement rock, si la piece est blanche et que rock est différent de None alors on passe à une autre condition qui vérifie qu'il n'y a pas de piece qui empercherait le rock (piece qui soit entre le roi et la tour). On le refait 4 fois en changeant les coordonnées et la couleur pour chaque rock. Et on retourne la liste des coups possibles.

La fonction deterLignediag permet de déterminer les lignes diagonales. On définit les coordonnées de la pièce comme étant Xp et Yp et la direction comme étant x et y. On ajoute la direction aux coordonnées. Moov0k prends la valeur de regarde_piece. S'il est vrai alors on ajoute les coordonnées à la liste de coups et puis on réajoute la direction. On trouve une autre condition qui veut que savoir si on détermine qu'une seule fois ce mouvement comme pour le roi par exemple et si c'est vrai alors moov0k passe à « False ». Si attack_piece est différent de « None » alors on regarde si la couleur de la pièce est différente de la pièce qu'on veut attaquer et si c'est vrai alors on ajoute le coup ou les coups à la liste de coups possibles. On retourne la liste pour terminer.

La fonction DeterP permet de déterminé les mouvements possibles d'un pion, on créer une liste vide lstMoov, Si la couleur prend la valeurs « white » on utilise la fonction deterPintermediaire pour calculer les coups possibles sinon on fait la même chose pour les noirs et on retourne la liste des coups possibles.

La fonction DeterC permet de déterminé les mouvements possibles des cavaliers, on utilise des boucles « for » pour tester des valeurs qui correspond au mouvement que ferait un cavalier et si toutes les conditions sont réunies, on supprime toutes les fois ou y et x sont égale à 0 ou sont égale entre elle. On ajoute ça aux coordonnées du cavaliers pour connaître ses mouvements possibles. On ajoute les coups qui permettent de manger une pièce d'un adversaire et on retourne la liste de coup.

La fonction DeterF permet de determiné les mouvements possibles des fous, pour i et j qui cooresponde à x et y, on ajoute les mouvements possibles grâce à la fonction deterLignediag. On retourne la liste de coup.

La fonction deterPintermedaire permet de calculer les déplacements possibles d'un pion noir ou blanc. Si les coordonnées en Y du pion sont égale aux coordonnées du pion en début de partie alors on ajoute les coups qui lui permettent d'avancées de deux ou d'une case sinon il ne peut avancer que d'une case. Pour la condition suivante on regarde s'il y une pièce dans les zones ou le pion peut attaquer qui sont les diagonales, s'il y a une pièce et que sa couleur est l'inverse de la sienne on ajoute le mouvement à la liste des mouvements possibles. On retourne pour finir la liste des coups possibles.

La fonction deterR permet de déterminer les mouvements des rois, on créer une liste lstMoov vide et pour i et j on calcule tous les coups possibles à l'aide de la boucle for … in range (-1,2) de x et y qui corresponde à x et y et on ajoute tout ça à lstMoov. On ajoute le rock s'il est possible à la liste des coups et on la renvoie.

La fonction deterD permet de déterminer les mouvements des dames, on créer une liste lstMoov vide et pour i et j on calcule tous les coups possibles à l'aide de la boucle for … in range (-1,2) de x et y qui corresponde à x et y et on ajoute tout ça à lstMoov

La fonction deterT permet de calculer tous les coups possibles pour une tour, on crée une liste vide lstMoov et on ajoute tous les coups possibles dans toutes les directions

La fonction InfoToEchec permet de transformer les coordonnées du plateau en coordonées d'un plateau d'échec normal. Tout d'abord on définit les coordonnées comme étant x et y et on retourne les noms de pièce en majuscule et leurs coordonnées à l'aide du dictionnaire pour qu'elle soit en notation « pa2 » par exemple, ce qui correspond à Pion en a2.

La fonction EchecToInfo fait l'inverse de la fonction InfoToEchec. On retourne les pièces en minuscule et leur couleur, avec leur coordonnée en utilisant la fonction trouve_cle.

La fonction trouve_cle permet de connaître la clé en fonction de la valeur du dictionnaire. On parcourt k et val dans le dictionnaire et si « cle » est égale à « val » alors on retourne k.

# Programme principale :

Dans un premier temps on initialise le dictionnaire contenant les pièces et leur coordonné, on créer les fenêtres de saisie pour que le joueur puisse rentrer leur pseudo. On initialise les variables rock avec tous les rocks possibles pour chaque joueur. On affiche la fenêtre et le menu du lancement du jeu. On attend que des deux joueurs rentres leur pseudo et cliques sur jouer ou rentres le nom d'une partie enregistrée et appuie sur Load. La variable tourDe est initialisée sur blanc car c'est toujours les blancs qui commencent aux échecs et le nombre de coup est mis à 0. On dessine l'échiquier, on créer la condition pour que le nombre de coups augmente à chaque fois que les blancs rejouent et le menu s'affiche. Si le joueur charge une partie alors on charge le fichier de partie car la variable load ne sera pas vide

On initialise les variables qui se réinitialise à chaque coups, moovedPiece et ValidMoov sont égale « None » car aucune pièce n'est sélectionnée. On créer une liste vide pour accueillir les futurs coups possibles de la pièce sélectionner.

On regarde si le rock est possible et si le joueur à rocker en appelant les fonction maj_rock et deplace_rock. On met le dictionnaire à jour avec le dernier coup valide fait.

On change la personne qui doit jouer avec une boucle « if, else »

# Manuel utilisateur :

**ATTENTION : Il faut exécuter le programme avec le fltk envoyer dans le fichier .zip car comme nous l'avons légèrement modifié pour intégrer un système de saisie il ne marchera pas sans celui-ci.**

Quand vous lancez le programme, un menu s'affiche, après avoir rentrez vos pseudos dans le saisie correspondant si vous cliquez sur jouer alors le jeu se lance. Si vous voulez jouer une partie sauvegarder il vous suffit de renter son nom dans le champ de saisie et d'appuie sur le bouton jouer. Une aide est disponible si possible en cliquant sur le bouton « aide »

Ensuite le joueur qui choisis les blancs commences, les noirs joues ensuite et ainsi de suite. Pour jouer une pièce il suffit de cliquer dessus et de cliquer sur la case où vous voulez qu'elle aille. Tout mouvements possibles sera indiquer par un cercle orange. Le programme fait en sorte qu'aucun coups impossibles soit fait et lorsqu'il y aura un échec vous ne pouvez bouger que des pièces qui empêches la situation d'échec pratique non ?

Sur la droite de l'échiquier on peut voir le nombre de coups fait et le tour de la personne qui doit jouer. Si vous voulez arrêter de jouer quoi de plus simple que de cliquer sur le bouton quitter en bas à droite du menu, ou si vous êtes en pleine partie mais que vous voulez la reprendre plus tard sauvegarder là en rentrant un nom dans le champ de saisie et cloquer

sur « save ». Dès que la partie est gagné pour un joueur ou qu'elle est nulle, une autre partie se relance en inversant les couleurs et en mettant à jour le score.

## Principale difficulté rencontrée :

Nous avons beaucoup de temps à trouver un moyen de coder la fonction mate_ou_pat mais c'est en se demandant qu'est ce qui faisait qu'il y avait mat ou pat qu'on a trouvé. Pour sauvegarder les parties, nous avons dû regarder qu'elle OS l'ordinateur utilisait pour lire un fichier car la méthode changeait pour Mac et Windows.

## Répartition du travail :

Nous avons codé ensemble la base du jeu jusqu'à ce qu'un pion se déplace correctement et puisse manger une autre pièce. Sébastien s'est ensuite attaché à améliorer le jeu en ajoutant les déplacements possibles. Pendant ce temps, Léo avançait sur le rapport.
Nous avons ensuite codé ensemble les fonctions restantes comme la détection d'échec, de mat, de pat, les menus et les modes de jeux, et terminé la rédaction du rapport