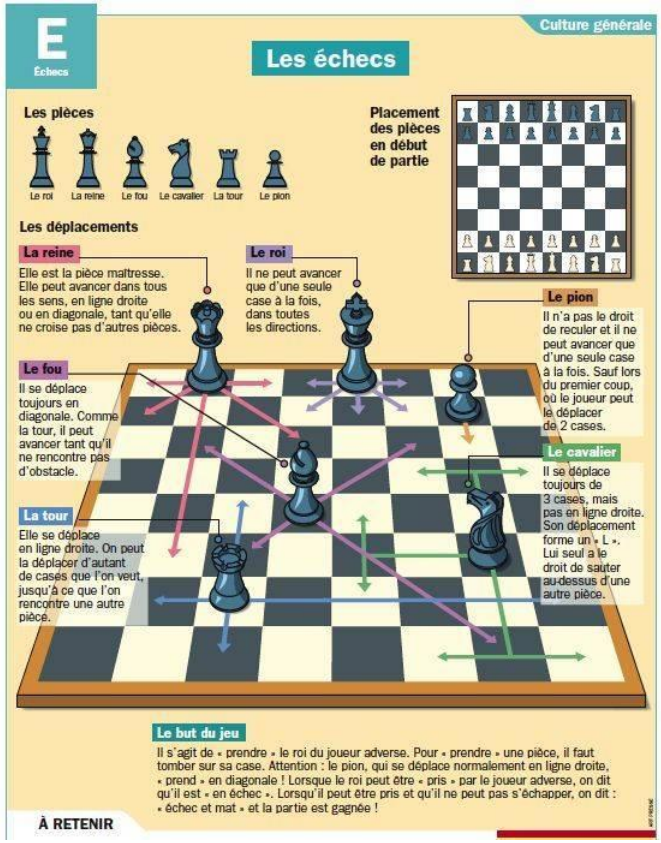
Projet jeu d'échec en langage C

Présentation du jeu

- Les échecs sont un jeu de société stratégique de type jeu de guerre qui se joue à deux joueurs, chaque joueur a chaque extrémité du damier (8x8) dispose de 16 pièces avec plusieurs types de pièces (pion, cavalier, fou, tour, reine, roi) avec un chacun des mouvements possibles. le but étant d'isoler le roi afin de le mettre en échec et mat.

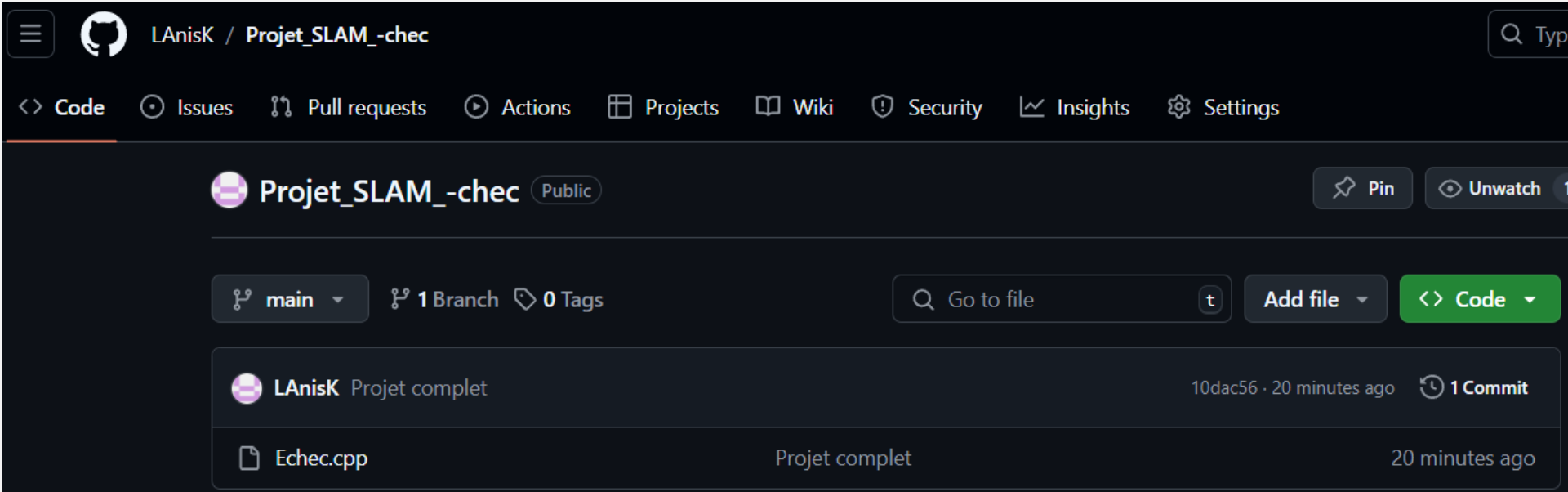


Présentation du Projet

- Le projet a pour but de créer une partie réduite d'un jeu d'échec standard avec seulement 4 pièces pour chaque joueur.
- Mettre en place un système de sauvegarde et de reprise de partie.
- Mettre en place un "IA" permettant de faire une partie contre l'ordinateur.

Gestion de projet avec github

Afin de sauvegarder et mettre à jour le projet à chaque modification on a utilisé GitHub comme outil de versionning.



Présentation du groupe

- Nous sommes un groupe de trois éléves de l'école IRIS
  - Théo ROUABLE : Chef de projet, a affecter les tâches aux autres membres ainsi que le développement du code de base du jeu d'échec en C
  - Anis KEDDAR : Développeur C chargé de l'étude et du développement du mouvement des pièces sur l'échecier
  - Chams-Eddin BOUCHOUARI : Développeur C chargé du développement du système de sauvgarde et de reprise de la partie.

Objectif

- • Présenter les deux fonctions pour sauvegarder et charger une partie d’un jeu.
- • Fonction `saveGame` : Enregistre l'état du jeu dans un fichier texte.
- • Fonction `loadGame` : Récupère les données sauvegardées pour les réinjecter dans le jeu.
- • Mise en place d'une "IA" pour jouer contre l'ordinateur.

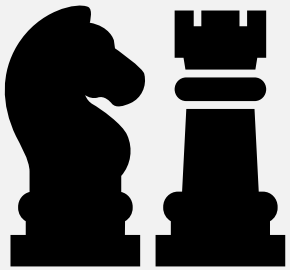
Fonction `isMoveValid`

- Cette fonction vérifie à ce que les mouvements des pièces soit respecté et non aléatoire

```c
// Vérifier si un mouvement est valide selon les règles d'échecs pour chaque pièce
int isMoveValid(char piece, int sx, int sy, int dx, int dy) {
    int deltaX = dx - sx;
    int deltaY = dy - sy;

    switch (piece) {
        case 'R':  // Tour
        case 'r':
            return (sx == dx || sy == dy);
        case 'C':  // Cavalier
        case 'c':
            return (abs(deltaX) == 2 && abs(deltaY) == 1) || (abs(deltaX) == 1 && abs(deltaY) == 2);
        case 'B':  // Fou
        case 'b':
            return abs(deltaX) == abs(deltaY);
        case 'Q':  // Reine
        case 'q':
            return (sx == dx || sy == dy || abs(deltaX) == abs(deltaY));
        default:
            return 0;
    }
}
```

Code de la fonction `isMoveValid`



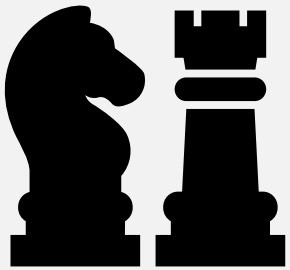
Fonction `saveGame`

- • Cette fonction sauvegarde l'état du jeu (ici, une grille) dans un fichier texte.
- • Chaque élément du tableau est écrit dans le fichier.
- • Un message est affiché si la sauvegarde réussit ou échoue.

```c
// Fonction de sauvegarde
void saveGame() {
    FILE *file = fopen("savegame.txt", "w");
    if (!file) {
        printf("Erreur : Impossible de sauvegarder la partie.\n");
        return;
    }

    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            fprintf(file, "%c", board[i][j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("Partie sauvegardee avec succès !\n");
}
```

Code de la fonction `saveGame`



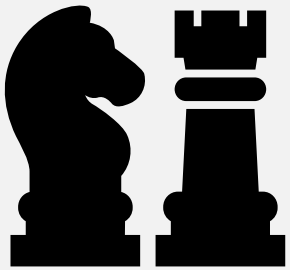
Fonction `loadGame`

- • Cette fonction lit les données sauvegardées dans un fichier texte.
- • Elle remplit le tableau de jeu avec les informations sauvegardées.
- • Un message est affiché si le chargement réussit ou échoue.

```c
// Fonction pour charger une partie sauvegardée
void loadGame() {
    FILE *file = fopen("savegame.txt", "r");
    if (!file) {
        printf("Erreur : Impossible de charger la partie.\n");
        return;
    }

    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            fscanf(file, " %c", &board[i][j]);
        }
    }
    fclose(file);
    printf("Partie chargee avec succes !\n");
}
```

Code de la fonction `loadGame`



Points Clés

- • Ouverture et fermeture du fichier avec `fopen` et `fclose`.
- • Utilisation de `fprintf` pour afficher les données dans le fichier.
- • Utilisation de `fscanf` pour écrire dans le fichier.
- • Gestion des erreurs (fichier non trouvé ou erreur d'ouverture).

Conclusion

- • Les fonctions de sauvegarde et de chargement sont essentielles pour permettre à un joueur de reprendre une partie.
- • Les fichiers texte peuvent être utilisés comme une solution simple et efficace pour stocker des données temporaires.

Vision d'amélioration

- Ce projet à été très intéressant à aboutir et on souhaiterais l'améliorer au fur et à mesure avec plusieurs idées en tête :
  - Créer une interface plus intuitif qu'un terminal, avec des boutons pour manipuler les pièces au lieu d'écrire les cases qu'on souhaite modifier.
  - Éclairage des cases possibles à atteindre avec une pièce une fois qu'on clique dessus.
  - Utiliser plus de pièces.
  - Une IA ayant des décisions plus stratégiques avec un filtrage de cas par cas selon les mouvements faits par le joueur.

Merci à vous pour votre attention !

—

Suivez-nous sur GitHub et consultez nos projets entamés !