

# Rapport de Projet : Wakachess

## Introduction

Ce document détaille la conception du projet Wakachess. Nous aborderons les étapes clés du développement ainsi que les fonctions structurantes qui permettent au programme de simuler une partie d'échecs.

Lien git du projet : <https://github.com/wakadu13/WakaChess.git>

(Ce document a été amélioré par IA)

### Conventions de données

- Représentation des pièces : Les majuscules représentent les pièces blanches et les minuscules les pièces noires.
- Structure du plateau : Le jeu est modélisé par un tableau (matrice) de cases, dont l'indexation commence à 0.

## PARTIE 1 : Logique de jeu (Fichier geo.py)

La première phase a consisté à concevoir les fonctions de déplacement théoriques. À ce stade, le programme ne vérifie pas encore la légalité absolue du coup (par exemple, si le roi est en échec).

### Déplacements par pièce

Les fonctions suivantes calculent les cases accessibles pour chaque type de pièce :

- `positionPossiblePion(x, y, plateau)`
- `positionPossibleTour(x, y, plateau)`
- `positionPossibleCavalier(x, y, plateau)`
- `positionPossibleFou(x, y, plateau)`
- `positionPossibleReine(x, y, plateau)`

- `positionPossibleRoi(x, y, plateau)` : Cette fonction est la plus complexe en raison des règles spécifiques au Roi et des contraintes de sécurité.

Note technique : Pour optimiser le code, j'ai utilisé les méthodes `str.isupper` et `str.islower` (suggérées par l'IA) afin d'identifier dynamiquement si une pièce est alliée ou ennemie sans dupliquer les fonctions.

### Vérification de la légalité

Pour s'assurer qu'un mouvement ne place pas le roi en échec, le programme utilise :

1. Généralisation : Un dictionnaire `index_fonction` lie chaque caractère (P, T, C, etc.) à sa fonction de mouvement respective.
2. Analyse de menace : Les fonctions `attaquesRoi` et `attaquesPion` modularisent la détection des menaces.
3. Validation finale : La fonction `CoupLegal(plateau, posInit, posFut)` centralise ces vérifications. C'est l'une des parties les plus complexes du code en raison de l'interdépendance des fonctions `trouverRoi` et `estEchec`.

Enfin, la gestion des tours de jeu et de la victoire est assurée par `ensembleCoupsLegauxMin/Maj` et `finDePartie`. La partie se termine dès qu'aucun coup légal n'est disponible.

## PARTIE 2 : Gestion du plateau

Cette section gère la partie matérielle et visuelle du jeu :

- `generationPlateau()` : Initialise la matrice de l'échiquier.
- `deplacer_piece_physique()` : Met à jour les coordonnées d'une pièce sur la matrice.
- `affichagePlateau()` : Gère l'interface textuelle ou graphique du plateau.
- `CopiePlateau()` : Fonction cruciale qui permet de simuler un coup sur une copie virtuelle pour vérifier sa légalité avant de l'appliquer réellement.

## PARTIE 3 : Fichiers annexes

- utilitaire.py : Regroupe les fonctions transversales et les outils de calcul mineurs.
- ia.py : Ce module a été généré via l'IA. Il contient les algorithmes de décision (Minimax ou autre) que je n'ai pas pu développer manuellement faute de temps.

## PARTIE 4 : Squelette du programme (main.py)

Le fichier principal orchestre le déroulement de la partie. Il contient la boucle de jeu et la fonction changementPosition, qui fait le pont entre l'entrée utilisateur et les règles de légalité. La logique du bot adverse a été affinée par l'IA pour garantir une meilleure réactivité.

## Conclusion et Perspectives

Le projet Wakachess est actuellement un prototype fonctionnel. Bien que les bases soient solides, certaines règles avancées comme le roque, la prise en passant ou la promotion du pion restent à implémenter.

Axes d'amélioration :

1. Optimisation de l'IA : Au niveau 5, le temps de calcul est trop important. L'implémentation d'un élagage Alpha-Bêta plus performant ou l'utilisation d'une base de données d'ouvertures permettrait d'accélérer la réflexion.
2. Refactorisation du code : Une simplification de l'architecture permettrait de réduire la redondance et d'améliorer la vitesse d'exécution globale.