LES TROPHÉES NSI

Édition 2023

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET

Ce document est l'un des livrables à fournir lors du dépôt de votre projet : 5 pages maximum (hors documentation).

Pour accéder à la liste complète des éléments à fournir, consultez la page **Préparer votre participation**.

Vous avez des questions sur le concours ? Vous souhaitez des informations complémentaires pour déposer un projet ? Contactez-nous à **info@trophees-nsi.fr**.

NOM DU PROJET : AI-ChessMate

> PRÉSENTATION GÉNÉRALE :

• Idée et objectifs
• Origines et intérêts du projet
• (...)

Idée et objectifs :

Notre idée était de créer un jeu d'échecs où l'on pourrait jouer contre une Intelligence Artificielle (IA) et où l'on pourrait avoir le choix entre plusieurs variantes de jeux. Un autre objectif de ce projet était de se perfectionner dans la maîtrise du langage de programmation JavaScript.

Origines et intérêts du projet :

Plus tôt dans l'année, nous avions déjà fait de nos côtés, dans le cadre d'un projet en NSI, des jeux ayant également pour but de jouer contre une IA, c'était le Tic Tac Toe et le Surakarta. Lors de ces projets, le développement des IA avait été la partie la plus difficile et la moins concluante, en effet, elles fonctionnaient, mais n'étaient pas optimales. Ainsi, nous avons voulu approfondir nos connaissances et notre maîtrise du sujet et pouvoir réussir à intégrer une IA optimale à un jeu. Aussi, sachant que nous sommes tous joueurs d'échecs, quoi de mieux de le faire sur la base de ce jeu, plus complexe que les précédents, qui pourra nous fournir plus de défi. Nous avions tous codé nos précédents projets en Python et nous avons voulu changer pour le JavaScript afin que le jeu soit plus interactif et surtout plus simple d'utilisation. Nous avons par ailleurs étudié Alan Turing en classe, il est considéré comme le père de l'IA et s'était donné pour objectif d'en réaliser une qui pourrait jouer aux échecs, nous avons alors trouvé intéressant de faire ce parallèle.

Mais ce projet ne devait pas avoir d'intérêt seulement pour nous qui l'avons développé, mais également pour toutes les personnes qui voudraient le tester. Cet intérêt est simple : pouvoir apprendre et s'entraîner aux échecs en jouant contre une IA plus ou moins forte et découvrir quelques variantes qui enrichissent le jeu.

> ORGANISATION DU TRAVAIL :

• Présentation de l'équipe (prénom de chaque membre et rôle dans le projet)
• Répartition des tâches
• Organisation du travail (répartition par petits groupes, fréquence de réunions, travail en dehors de l'établissement scolaire, outils/logiciels utilisés pour la communication et le partage du code, etc.)

• Evan : Création du jeu de base en Python

• Baptiste : Création de la variante « Mini-Échec » en Python et d'une IA qui pourrait y jouer. Création de la vidéo.

• Robinson : Création de toutes les autres variantes en JavaScript et conception du dossier de projet. Création du logo.

• Amaury : Conversion du Python en JavaScript et optimisations des fonctions et création de l'IA.

** Il est important de noter que nous pouvons tous accéder au travail des autres puisque nous travaillons sur la plateforme de développement GitHub pour le code source et sur Google Docs pour le dossier de projet*

Pour s'organiser, nous avons créé un groupe sur discord afin de communiquer entre nous, pour le partage de code, nous avons utilisé le site GitHub dans le but de pouvoir d'avoir accès à tous les fichiers simultanément. Presque tout le travail a été réalisé en dehors de l'établissement scolaire. Nous avons cependant eu plusieurs réunions dans l'établissement dans le but de faire le point sur l'avancée du projet, ainsi que sur la répartition des éventuelles nouvelles tâches qui pouvaient faire leur apparition comme la résolution de bugs.

LES ÉTAPES DU PROJET :

• Présenter les différentes étapes du projet (de l'idée jusqu'à la finalisation du projet)

Nous avons tout d'abord commencé par étudier les mouvements de toutes les pièces, puis nous avons créé une fonction qui regroupe tous ces déplacements. Nous avons commencé par créer le jeu de base en python, car c'est le langage que nous maîtrisions le mieux pour ensuite le transférer en JavaScript. Une fois que nous avions le jeu de base et que nous pouvions jouer en joueur vs joueur, nous avons créé notre IA basée sur le minimax puis l'avons intégrée à notre jeu. Afin de la rendre plus ou moins forte, nous jouons sur la profondeur de recherche du Minimax, c'est-à-dire sur combien de coups travaille-t-il en avance. Ensuite, pour gagner en performance, nous avons ajouté la version du minimax avec élagage, ce qui le rend plus performant. Pendant ce temps-là, un des membres du groupe créait le mini-échec en python. En attendant, nous avons ajouté les différentes variantes que nous voulions mettre en plus du mini-échec, à savoir ; le Roi de la Colline, les Échecs Circassiens, les Trois Échecs, et le Chess960 directement en JavaScript. Une fois que le mini-échec fut terminé en python, nous l'avons traduit en JavaScript puis implémenté en tant que dernière variante.
Une fois toutes les variantes ajoutées, il a fallu créer la page d'accueil afin de pouvoir choisir contre quelle variante jouer et sélectionner la profondeur maximum à laquelle elle peut aller (nombre de tours qui seront anticipés par celle-ci). Nous avons également créé des boutons qui permettent de relancer une partie en cours de partie et de retourner à l'accueil. Une fois cela terminé, il ne nous restait plus qu'à tout documenter et le projet était terminé.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

• Avancement du projet (ce qui est terminé, en cours de réalisation, reste à faire)
• Approches mises en œuvre pour vérifier l'absence de bugs et s'assurer de la facilité d'utilisation du projet
• Difficultés rencontrées et solutions apportées

Le projet est terminé, nous avons réussi à faire tout ce que nous voulions et même plus puisque les variantes n'étaient presque que du bonus.

Afin de vérifier l'absence de bugs et de s'assurer de la facilité d'utilisation, nous testions nos modifications quasiment à chaque fois. Pour s'assurer de la confortabilité visuelle de l'affichage et des couleurs, nous testions les couleurs en direct. De plus, nous avons partagé notre jeu à quelques personnes afin qu'elles le testent et nous fassent un retour sur les éventuels bugs que nous n'aurions pas détectés et sur le jeu en lui-même.

Nos principales difficultés ont été :

- Faire une copie totale d'un tableau : pour pouvoir déboguer plus facilement, par exemple "let tableauCopie = ancienTableau", ne feras pas une vraie copie de ancienArray, car si nous changeons par la suite "tableauCopie" alors "ancienTableau" change aussi. Nous avons mis du temps à trouver une solution qui est la suivante : JSON.parse(JSON.stringify(ancienArray).

- Bien comparer deux tableaux, en JavaScript. Exemple : "[1,2] === [1,2]" ne retourne pas true, pour cela, nous avons été obligés de créer une fonction spéciale, la même chose s'est produite pour savoir si un tableau est dans un autre tableau.
- Comprendre comment fonctionnait le minimax, ce fut difficile de réussir à bien visualiser ce que faisait le minimax
- Faire une bonne fonction d'évaluation, le minimax repose énormément sur la fonction d'évaluation, nous avons mis du temps à en trouver une qui fonctionne correctement.
- La récursivité nous a posé quelques légers problèmes au début du projet car elle ne fait pas partie du programme de première.

> OUVERTURE :

• Idées d'améliorations (nouvelles fonctionnalités)
• Stratégie de diffusion pour toucher un large public (faites preuve d'originalité !)
• Analyse critique du résultat (si c'était à refaire, que changeriez-vous dans votre organisation, les fonctionnalités du projet et les choix techniques ?)

Améliorations :

Nous avons déjà apporté des améliorations en proposant des variantes au jeu classique, nous pourrions ajouter un système qui affiche les pièces capturées sur le côté, un compteur de temps et un compteur de point.

Diffusions :

Nous avons étendu notre projet à un large public de plusieurs manières ; les personnes âgées sont concernées, car elles savent souvent jouer aux échecs. Le fait que le jeu soit sur internet peut limiter l'accès de certaines personnes, mais peut favoriser celui des jeunes. Plusieurs niveaux de difficulté sont disponibles : la variante mini échec pour apprendre à jouer ainsi que trois niveaux de difficulté pour l'IA. Les niveaux qui pourraient être intéressés par ce jeu vont de débutants totaux à joueurs confirmés. Une stratégie de diffusion pourrait être de poster le jeu sur le web et de donner le lien d'accès sur des forums et des serveurs liés aux échecs afin que les personnes intéressées puissent s'entraîner facilement.

Résultat :

Nous sommes satisfaits du résultat final, bien sûr rien n'est parfait et nous pourrions toujours améliorer et approfondir notre projet, mais si nous devions le refaire, nous ne changerions pas grand-chose. La première serait d'améliorer notre organisation et la deuxième de s'inspirer directement de notre programme Python alors que nous avons perdu du temps en essayant de le faire directement en JavaScript.

DOCUMENTATION

• Spécifications fonctionnelles (guide d'utilisation, déroulé des étapes d'exécution, description des fonctionnalités et des paramètres)
• Spécifications techniques (architecture, langages et bibliothèques utilisés, matériel, choix techniques, format de stockage des données, etc)
• Illustrations, captures d'écran, etc

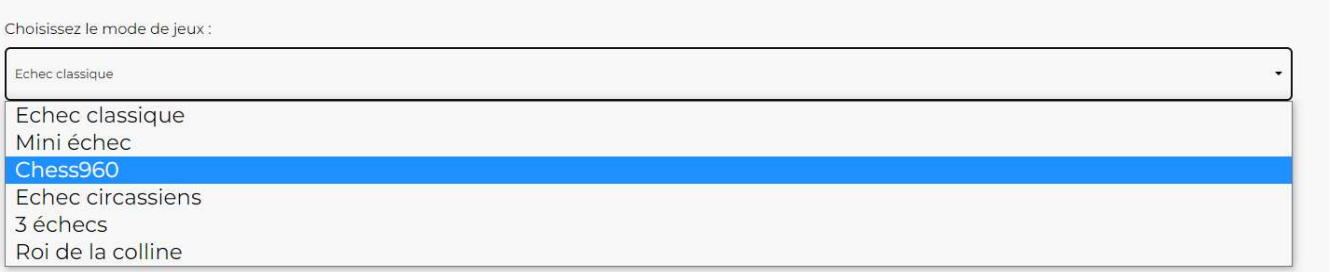
Spécifications fonctionnelles :

Les étapes pour utiliser le jeu sont les suivantes :

1. Télécharger le dossier "AI-Chessmate" :

2. Ouvrir "index.html" dans un navigateur web :

3. Choisir un mode de jeu :



4. Choisir le niveau de l'IA :

5. Cliquer sur le bouton jouer :

6. Vous pouvez retourner à l'accueil ou relancer une partie :

Spécifications techniques :

Langages utilisés :
- JavaScript 72,2 %
- Python 25,5 %
- CSS 1,9 %
- HTML 0,8 %

Matériel utilisé :
- Visual Studio Code
- GitHub
- Google Docs

Fonctions :

Communes à tous les fichiers :

```
arrayEqual(a, b)
/*
 * Fonction qui compare 2 arrays simple pour savoir s'ils sont égaux
 *
 * - @param {Array} a: L'array qui se fait comparer
 * - @param {Array} b: L'array qui compare
 *
 * @returns {Boolean} - true si les 2 array sont égaux - false sinon
 */
```

```
elementInArray (element, array)
/*
 * Fonction qui regarde si un élément est compris dans un autre array
 *
 * - @param {Array} élément: L'élément que l'on cherche
 * - @param {Array} array: l'array dans lequel on regarde si l'élément est
dedans
 * - @retrurns {Boolean}: true si élément est dans array - false sinon
 */
```

Mini-échecs :

```
function mini_chess()
/*
 * Fonction qui crée entièrement la variante spéciale “Mini Échecs”
 */
```

JEU :

```
switchPlayer()
/*
 * Change le joueur qui doit jouer
 */

legalMove (board, coup_precedant, color)
/*
 * Fonction qui récupère tous les coups possibles et légaux pour tous les pions d'une couleur
 *
 * - @param {Array} board: représente l'état actuel du jeu
 * - @param {Array} coup_precedant: représente le coup qui vient d'être joué
 * - @param {string} color: (soit 'white' soit 'black') qui nous dit la couleur des pièces qu'on veut avoir
 *
 *
 * - @returns {array}: contient tous les coups possibles et légaux pour les pièces de `color`
 */
```

```
endgame(winner, cause)
/*
 * Fonction qui affiche le pop up qui nous indique comment et qui a gagné la partie
 *
 *
 * - @param {String} winner: nous dit la couleur du gagnant ou s'il y a nulle (soit 'white' soit 'black' soit 'draw')
 * - @param {String} cause: nous indique pour quelle raison le gagnant à gagner
 *
 *
 * - @returns {fonction} appel la fonction game avec comme paramètre true pour que ça arrête le jeu
 */
```

```
playMove(move)
/*
 * Fonction qui fait bouger une pièce en fonction de ses mouvements associés
 *
 * - @params {array} move: les coordonnées de départ et d'arrivée du mouvement
 */
```

```
game (endgame)
// fonction principale du jeu
```

```
init()
// fonction qui initialise le plateau de base, virtuel et non virtuel
```

```
refreshBoard (chessboard)
// Fonction qui actualise le plateau de la page web avec @param {array}
chessboard
```

```
function win_nul (board, coup_precedant, coup)
/*
 * Fonction qui compare 2 arrays simple pour savoir s'ils sont égaux
 *
 * - @param {Array} board:  représente le plateau de jeu
 * - @param {Array} coup_precedant: représente le coup qui vient d'être joué
 * - @param {Array} coup: représente tous les coups possibles de la couleur
étudiée
 * - @returns {Number} -  -1 si nul -  0 si rien - 1 si victoire
 */
```

```
move (board, row, col, prevMove){
/*
 * Fonction qui renvoie tous les coups qu'une pièce peut faire selon les règles
des échecs
 *
 * - @param {Array} board: plateau du jeu
 * - @param {Number} row: la ligne de board où se trouve la pièce à étudier
 * - @param {Number} col: la colonne de board où se trouve la pièce à étudier
 * - @param {Array} prevMove: coordonner du coup précédant
 *
 * - @returns {Array} contient tous les coups d'une pièce
 */
```

```
promotion(color)
/*
 * Fonction qui détermine la valeur de la promotion
 *
 * - @param {String} color: la couleur de la promotion
 *
 * - @returns {Number}: la valeur de la promotion
 */
```

```
grand_rook (board, nb_m_roi, nb_m_tour0, color)
/*
 * Fonction qui détermine si le grand_rook est possible ou non
 *
 * - @param {Array} board: représente le plateau d'échecs
 * - @param {Number} nb_m_roi: nombre de mouvements du roi
 * - @param {Number} nb_m_tour0: nombre de mouvements de la tour à gauche du roi
 * - @param {String} color: représente la couleur pour laquelle on vérifie le
grand rook
 *
 * - @returns {Boolean}: true si le grand rook est possible - false s'il ne
l'est pas
 */
```

```
petit_rook (board, nb_m_roi, nb_m_tour7, color)
/*
 * Fonction qui détermine si le petit rook est possible ou non
 *
 * - @param {Array} board: représente le plateau d'échecs
 * - @param {Number} nb_m_roi: nombre de mouvements du roi
 * - @param {Number} nb_m_tour7: nombre de mouvements de la tour à droite du roi
 * - @param {String} color: représente la couleur pour laquelle on vérifie le
petit rook
 *
 * - @returns {Boolean}: true si le petit rook est possible - false s'il ne
l'est pas
 */
```

```
clouage (board, color, coup_precedant)
/*
 * Fonction qui vérifie qu'un coup ne mette pas son roi en échec
 *
 * - @param {Array} board: plateau de jeu
 * - @param {String} color: couleur des coups qu'on vérifie
 * - @param {Array} coup_precedant: le coup qui a été jouer
 *
 * - @returns {Array}: la liste des coups qui sont bons, qui ne mettent pas en
échec leur roi
 */
```

```
anti_suicide (board, coup_precedant, color)
/*
* Fonction qui empêche le roi d'aller sur une case ou il est en échec
*
* - @param {Array} board: le plateau de jeu
* - @param {Array} coup_precedant: le dernier coup joué
* - @param {String} color: la couleur du roi


* - @retrurns {Array}: retourne les coordonnées des coups ou le roi peut aller
sans être échec
*/
```

```
check (board, coup_precedant, color)
/*
 * Fonction qui détermine s'il y a échec après avoir joué un coup
 *
 * - @param {Array} board: plateau de jeu
 * - @param {Array} coup_precedant: le coup a été joué précédemment
 * - @param {String} color: la couleur du coup qui a été joué
 *
 * - @returns {Boolean}: true si il y a echec - false si il n'y a pas
 */
```

IA :

```
minimax (game, depth, alpha, beta, isMaximizingPlayer, sum, color,
coup_precedant)
/*
 * Explore récursivement tous les coups possibles jusqu'à une certaine
profondeur, et évalué le plateau au niveau des feuilles
 *
 * Idée principale : maximise la valeur minimum de la position résultant des
coups possibles de l'adversaire.
 *
 * Paramètre :
 *   - game :                 the game object.
 *   - depth:                la profondeur à laquelle le minimax va au maximum.
 *   - alpha & beta:         sert à élaguer les branches inutiles
 *   - isMaximizingPlayer:   true si on maximise, false si on minimise.
 *   - sum:                  la somme (l'évaluation) du plateau actuel
 *   - color:                la couleur de l'IA
 *   - coup_precedant:       le coup joué juste avant
 *
 *   retourne le meilleur coup à jouer
 */
```

```
evaluateBoard (move, board_initial, prevSum, color, win)
/*
 * Fonction qui explore de manière récursive tous les coups possibles jusqu'à
une certaine profondeur, et évalue le plateau au niveau des feuilles en
utilisant l'algorithme minimax grâce à élagage alpha-bêta.
 *
 * @param {Array} move - mouvement que fait une pièce
 * @param {Number} board_intial - le plateau initial (sans le move appliquer)
 * @param {Number} prevSum - True si l'IA cherche à maximiser sa valeur, False
si elle cherche à minimiser la valeur de l'adversaire.
 * @param {Number} prevSum - la valeur de la précédente évaluation
 * @param {String} color - La couleur de l'IA ('b' pour black, 'w' pour white).
 * @param {Number} win - la valeur de win quand on effectue ce coup
 * @param {Boolean} check - true si échec - false si pas
 *
 * @returns {Number} - la nouvelle évaluation du tableau
 */
```