## Projet de Développement [LU2IN013]

Awelo - Projet de jeu à deux joueurs

2022



Awelo - Projet de jeu à deux joueurs

Pour le 2nd semestre de L2 Informatique Mono-disciplinaire, nous avons réalisé le projet ci-contre, dans le cadre de l'UE Projet de Développement (LU2IN013) de Sorbonne Université.

(Accès au projet sur : *https://gitlab.com/ishratslh/awelo*)

Awelo est un projet basé sur la résolution de jeux de plateau qui sont Awele et Othello, à l’aide de Python.

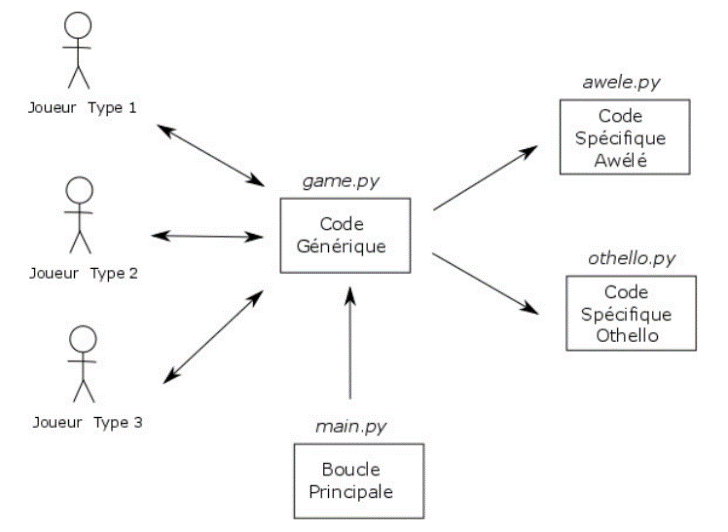
# Chronologie

Au cours des 3 premières semaines, nous avons, dans un premier temps, instauré les bases du projet en implémentant :

* l’interface de jeu pour Othello et Awéle

Cette interface se divise en deux parties :

* Le code générique (*game.py*) ;
* Le code spécifié (ex : *awele.py*, *othello.py*).
* les joueurs (*joueur\_humain*, *joueur\_aleatoire*, *joueur\_1erCoupVal*) ;
* la boucle principale pour une partie (*main.py*) ;
* l’extention pour n parties (dans *main2.py*) ;

Cette partie a pu s’étaler sur les semaines d’après, pour notamment perfectionner l’interface des 2 jeux et l’extension pour n parties afin d’établir des statistiques de victoire quand nous en avions besoin.

L’architecture générale de notre projet se décrit ainsi par ce schéma repris du cours :

Nous avons ensuite dans un 2nd temps, réfléchit à une fonction d’évaluation des coups, et défini un joueur à horizon 1 (*joueur\_h1*).

Ensuite, nous avons découvert le concept de l’algorithme de parcours d’arbre, notamment pour évaluer un coup, et débuté à concevoir l’algorithme MiniMax (*joueur\_Minimax*).

Puis, nous nous sommes intéressés à l’élagage de l’arbre avec l’algorithme Alpha-Bêta (*joueur\_AB*), tout en optimisant le parcours d’arbre.

Enfin, dans les dernières semaines, nous nous sommes penchés sur les techniques d’apprentissage : génétique, supervisé (*joueur\_apprenti*, *joueur\_supervise*).

# Détail du code

### La boucle principale :

* *main.py*

Le **main** va être le centre du code notamment lorsque nous allons devoir :

* définir les deux joueurs que nous voulons voir s’affronter ;
* jouer une ou n partie(s) avec unepartie() ou jouerNParties() ;
* exécuter le fichier dans le terminal.

### Le code générique :

* *game.py*

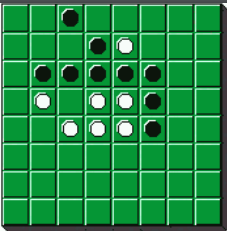
Le **code générique** va permettre à l’implémentation de l’interface de rester valide pour tous les jeux de plateau à 2 joueurs possibles.

En effet, dans ce module sont présents beaucoup de fonctions qui sont communes à tous les jeux (ex : getScore(), getCopieJeu(), changeJoueur(), getJoueur() …).

Mais sont aussi présentes l’appel à des fonctions (comme joueCoup() ) qui vont être différentes selon le jeu auquel on joue. Par exemple, un coup dans Awele ne sera pas joué de la même façon que dans Othello : les conditions, le plateau, les cases, les scores etc. diffèrent.

### Le code spécifique :

Un **code spécifié** va varier selon le jeu auquel on va souhaiter jouer, et contiendra ses spécificités, comme expliqué précédemment.

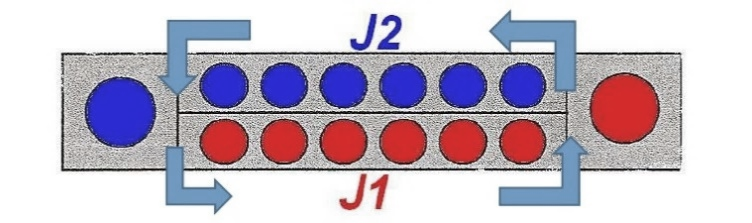
* *Othello.py*

Le célèbre jeu **Othello** se joue sur un plateau de 64 cases (8x8). Au début, 2 pions de chaque couleur sont posés sur les 4 cases centrales.

Chaque joueur va, à son tour, poser un pion de sa couleur afin dans le but d’encadrer un maximum de pions possibles.

Le jeu se termine lorsque les 64 pions sont posés, et le gagnant est celui qui aura le plus de pions de sa couleur.

En plus des fonctions génériques de *game.py*, notre plus grosse fonction est prise(). Cette fonction va repérer les pions qui peuvent être retournées une fois qu’un pion est posé, et les retourner.

* *****Awele.py*

**Awele**, quant à lui, débute en répartissant 4 graines par trou dans les 12 trous du plateau.

Le but du jeu est d'avoir récupéré le plus de graines à la fin de la partie.

A chacun son tour, les joueurs vont choisir un trou de leur camp, y récupérer les graines et égrainer les cases qui suivent dans le sens anti-horaire.

Si sa dernière graine tombe dans un trou du camp adverse et qu'il y a maintenant deux ou trois graines dans ce trou, le joueur récupère ces deux ou trois graines et les met de côté. Ensuite il regarde la case suivante dans le sens horaire : si elle est dans le camp adverse et contient deux ou trois graines, il récupère ces graines, et ainsi de suite jusqu'à ce qu'il arrive à son camp ou jusqu'à ce qu'il y ait un nombre de graines différent de deux ou trois.

D’autres modalités sont décrites dans le cours, mais sont moins pertinentes ici.

La fonction la plus importante ici est egrainer() qui va, comme son nom l’indique, égrainer les cases, et amasser les graines (et le score) dans la mesure du possible.

### L’interface graphique :

* *fenetre*.*py*

Pour le jeu Othello uniquement, nous avons créé une **interface graphique** à l’aide du module Tkinter. Elle est composée d’un échiquier de boutons qui s’actualise à chaque fois qu’on joue un coup, selon les règles citées précédemment. Deux tailles de boutons (dessinés par nous-mêmes) ont été prédéfinies, une sous Linux, l’autre sous Windows car les deux systèmes d’exploitation n’affichent pas les deux tk.buttons de la même manière.

Pour lancer l’interface, il faut aller dans le bon répertoire (Awele>Othello>Fenetre) et exécuter fenetre.py.

### Nos joueurs :

Un jeu de plateau va faire s’affronter deux joueurs, qui auront chacun leurs caractéristiques.

Dans le code se trouvera la principale fonction qu’est saisieCoup(jeu) qui va, comme son nom l’indique, nous permettre de saisir un coup selon le type de joueur :

* *joueur\_humain*.*py*

Le **joueur humain** (éventuellement nous) va devoir insérer au clavier la position (n° de la colonne pour Awele ; ligne et colonne pour Othello) de la case dans laquelle il va vouloir jouer son coup, qui sera ensuite joué.

Un message d’erreur est affiché dans le cas où ce qui est saisi n’est pas un nombre (ex : une lettre).

Ce joueur a été fortement utile, notamment lors des toutes premières séances, lorsqu’il a fallu créer les interfaces des jeux.

* *joueur\_aleatoire.py*

Le **joueur aléatoire** n’a pas pour objectif de jouer le meilleur coup possible mais va, bien au contraire, grâce à random.choice(), choisir un coup aléatoirement parmi la liste des coups valides, et simplement le jouer.

Ce joueur nous a été d’une grande aide lorsqu’il fallait tester la validité de nos joueurs à différents horizons en jouant rapidement plusieurs parties d’affilée.

* *joueur\_1erCoupVal.py*

Le **joueur Premier Coup Valide** va jouer le premier coup de sa liste des coups valides.

Les joueurs suivants vont avoir, comme fonctions dominantes :

* decision()retournant le coup dont le score correspond au score d’évaluation maximal de la liste de coups passée en paramètre ;
* estimation()retournant un score d’utilité estimée pour un état de jeu à partir d’appels de la fonction d’évaluation sur les feuilles de l’arbre ;
* evaluation()retournant un score d’évaluation d’un état de jeu.
* *joueur\_h1.py*

Le **joueur à horizon 1** va se baser sur des estimations effectuées à profondeur 1, à savoir sur les conséquences immédiates d’un coup joué.

Nous avons ensuite, pour réfléchir au fonctionnement des horizons en fonction de la profondeur, créé les 2 fichiers suivants afin de finalement de recentrer sur ce qui a été demandé en cours.

* *joueur\_h2.py*
* *joueur\_h3.py*
* *joueur\_Minimax.py*

Le **joueur MiniMax** a été défini en fonction de l’algorithme MiniMax. L’idée principale de l’algorithme est de choisir le coup avec la meilleure valeur de minimax, soit avec la meilleure situation atteignable contre le meilleur coup.

* *joueur\_AB.py*

Le **joueur Alpha-Bêta** a été défini en fonction de l’algorithme Alpha-Bêta, qui part de l’idée que certaines branches de l’arbre de recherche sont inutiles et qu’on peut ne pas en tenir compte grâce à l’élagage α-β.

* *joueur\_apprenti.py*
* *joueur\_supervisé.py*
* *oracle.py*

L’oracle va donner le coup à préférer comme Alpha-Beta dans le cadre du jeu supervisé/en apprentissage.

# Conclusion :

Sur ces dix semaines, il a été impressionnant et enrichissant de partir de quasiment rien, et d’en arriver au projet que nous avons aujourd’hui.

Ce qui a pris le plus de temps ici aura été, comme prévenu, d’instaurer d’abord les bases du projet. Mais l’on comprend ensuite à quel point on peut se spécifier et aller loin à partir d’un simple objectif, notamment lorsqu’il fallait réfléchir au joueur qui nous imitait ou apprenait de nos coups.