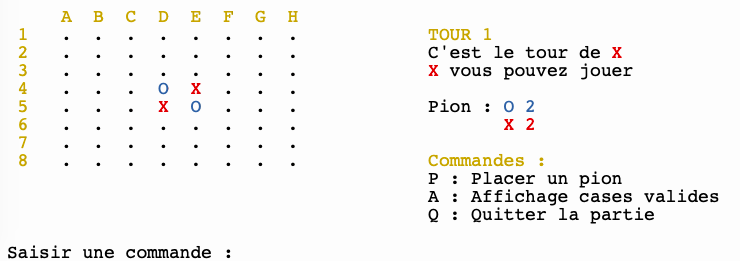
2019 - 2020

PROJET 2 – Black & White



MILANO Célia

DELRIEU Pierrick

L1 - groupe B

**SOMMAIRE**

1. Menu de présentation
2. Les fonctions principales
3. L’affichage sur une interface texte classique
4. L’architecture du programme
5. Les options

**Le jeu Black & White** est inspiré du jeu Othello, aussi appelé Reversi. La première édition de ce jeu est sortie dans les années 80. Il est aujourd’hui un jeu référence du gaming. En effet, depuis quelques années il possède ses propres championnats du mondes (junior, mixte, hommes, femmes et par équipe). Ce jeu est très rependu au Japon mais également en France où de nombreux tournois sont organisés. D’ailleurs, son édition a entrainé dans les années 80 la création de la FFO (Fédération Française d’Othello).

**Règle du jeu :**

Le Black & White se joue sur un plateau carré de taille général 8x8. C’est un jeu qui se joue à deux joueurs, un possédant des pions noirs et l’autre possédant des pions blancs. Les joueurs jouent chacun leur tour (sauf si un joueur doit passer son tour car il ne peut pas jouer).

Le but du jeu est de posséder à la fin de la partie le plus de pion possible. Une partie est finie lorsque le plateau est plein ou lorsque les deux joueurs n’ont plus de possibilité de jeu.

Pour avoir le plus de pion possible il faut capturer les pions de l’adversaire. Un pion adverse se transforme en notre pion lorsque le pion adverse est entouré par deux de nos pions. Il est possible de capturer les pions en ligne et en diagonale.

Les joueurs peuvent poser les pions seulement sur des cases vides et adjacentes a un pion adverse. Ils ne peuvent pas modifier l’emplacement d’un pion en cours de partie ni retirer un pion.

**Le projet :**

L’objectif du projet était de réaliser ce jeu sur une interface texte classique.

Dans le projet, le plateau peut varier de taille (la taille étant choisie par l’utilisateur en début de partie). Nous avons réalisé le jeu avec des croix X et des O pour modéliser les pions. Nous avons établi que X commencera la partie. Le jeu a été réalisé avec deux modes, un multijoueur (2 joueurs) et un mode solo.

**1 - Menu de présentation**

Tout d’abord, nous avons réalisé un menu de présentation lorsque le joueur commence le jeu.

**Le menu de présentation se situe à chaque début de partie et contient :**

* Le nom du jeu
* Les explications rapides du jeu
* Le choix du mode (multijoueur ou solo)
* La saisie du ou des noms d’utilisateurs
* L’attribution des pions au(x) joueur(s)
* La saisie de la taille du plateau

Le choix du mode se fait grâce à la saisie de ‘S’ pour le mode solo ou de ‘M’ pour le mode multijoueur. L’utilisateur peut cependant saisir les lettres en minuscules. En effet, celle-ci sont convertis automatiquement en majuscule grâce à : mode = mode.upper().

Toutes les saisies devant être sécurisées, la saisie du mode est sécurisée grâce à une boucle tant que.

La saisie du ou des noms d’utilisateurs est libre.

Les noms sont convertis en minuscule avec la première lettre en majuscule grâce à : nom=nom.capitalize()

L’attribution des pions au(x) joueur(s) ne se fait pas de manière aléatoire.

Le pion1=’X’ est attribué automatiquement au joueur 1 et le pion2= ’O’ est attribué au joueur 2 ou à l’ordinateur.

Cependant, l’utilisateur peut choisir d’échanger les pions. Pour cela, lorsque le programme lui demande il faut qu’il saisisse ‘change’. Toutes les saisies devant être sécurisé, la saisie est sécurisée grâce à une boucle tant que.

La saisie de la taille du tableau peut être comprise entre 6 inclus et 26 inclus. La note de cadrage nous impose une taille minimale de 6x6. Cependant, pas de taille maximale nous a été imposée. La taille maximale que nous avons donc choisie est 26x26 car les noms de colonnes devant être représentés par des lettres et l’alphabet contenant 26 lettres, il n’est pas possible de réaliser un tableau de ce type de taille supérieur à 26.

Toutes les saisies devant être sécurisé, l’utilisateur entre une chaine de caractères et on vérifie ensuite si cette chaine de caractères est un nombre valide. Si cette chaine de caractères est un nombre valide alors on la convertit en un entier grâce à : taille = int(taille).

Exemple pour le mode multijoueur :

Une image contenant capture d’écran

Description générée automatiquement

**2 - Les fonctions principales**

**Tout d’abord, voici les fonctions nécessaires à la création du programme principale :**

* Création d’un plateau de taille choisit par l’utilisateur
* Initialisation des pions
* Saisie coordonnées des pions
* Placement pion sur le plateau
* Liste des cases valides (en fonction du pion X ou O)
* Retournement pions (en fonction du pion X ou O)
* Condition arrêt partie
* Détermination du gagnant

La création du plateau se fait en créant un tableau 2D.

Un tableau 2D consiste à créer des listes dans une liste principale à l’aide d’une liste temporaire. Pour cela, on établit un compteur i de ligne. On ajoute à la liste temporaire les éléments d’une ligne i, puis on ajoute cette liste à la liste principale. Il faut réinitialiser la liste temporaire à chaque itération.

Le nombre de lignes et de colonnes sera établie par un paramètre taille, saisie par l’utilisateur dans le programme principal.

Les boucles pour remplir le tableau 2D vont aller jusqu’à taille+1 car il ne faut pas compter la première ligne ni la première colonne qui vont contenir les numéros de lignes et de colonnes des différentes cases du plateau.

Pour remplir le tableau 2D, il faut donc créer une boucle ‘pour i allant de 0 à (taille+1)’ qui va faire un nombre d’itérations correspondant au nombre de lignes. A chaque itération il faut initialiser une liste temporaire qui va être remplie à l’aide d’une nouvelle boucle ‘pour j allant de 0 à (taille+1)’ à laquelle on ajoute des conditions. Une fois la ligne i parcourue, la liste temporaire est ajoutée à la liste principale. On réinitialise ensuite la liste temporaire.

* La première case du tableau, est toujours vide car elle correspond au croisement de la première ligne et de la première colonne. Il faut donc mettre une condition telle que :

Si (i=0) et (j=0) alors on ajoute un espace.

* La première colonne contient les nombres de 1 à taille, ce qui correspond aux numéros des lignes. Il faut ajouter une condition telle que :

Sinon si (j=0) alors on ajoute i à la liste temporaire.

* La première ligne contient les lettres de A à la lettre correspondant à la taille du tableau, ce qui correspond à l’élément d’indice [j-1] de la liste alphabet initialisée au début. On ajoute alors une condition telle que :

Sinon si (i=0) alors on ajoute la lettre d’indice [j-1] de l’alphabet à la liste temporaire.

* Enfin pour le reste du tableau, on ajoute des points, on met une dernière condition telle que :

Sinon on ajoute ‘.’ à la liste temporaire.

L’initialisation des pions se fait au centre du plateau (sous consigne de nos enseignants de TP). Or, la taille du plateau peut être paire ou impaire. Il n’est donc pas possible de placer les pions au centre du plateau pour une taille impaire. L’initialisation des X et des O se fait en diagonale.

La fonction va donc modifier le plateau, elle retourne donc aucun élément.

On modifie donc la taille du plateau :

plateau [int(taille/2)] [int(taille/2)+1]  'X'

plateau [int(taille/2)+1] [int(taille/2)]  'X'

plateau [int(taille/2)] [int(taille/2)]  'O'

plateau [int(taille/2)+1] [int(taille/2)+1]  'O'

A chaque indice du tableau, on garde la partie entière car la taille du tableau peut être impaire.

La saisie des coordonnées des pions se fait en deux étapes, une saisie utilisateur et la sécurité de la saisie.

Tout d’abord, l’utilisateur saisi la lettre de la colonne et le numéro de la ligne (ex :’A1’) correspondants aux coordonnées de la case où il veut placer son pion. La variable ‘coord’ récupère la saisie de l’utilisateur. Cette saisie est convertie en majuscule grâce à la fonction ‘upper’, pour faciliter la sécurité de la saisie.

Ensuite, on vérifie si les coordonnées saisies se trouvent dans la liste de cases valides créée grâce à la fonction ‘cases\_valides’ Si elles ne se trouvent pas dans cette liste alors notre fonction demande à l’utilisateur de saisir de nouvelles coordonnées. Puis la fonction retourne ‘coord’ (les coordonnées saisie).

Le placement des pions sur le plateau se fait en récupérant les coordonnées saisies par l’utilisateur et en modifiant le plateau. Le plateau est une donnée modifiée de la fonction. La fonction ne retourne donc rien.

On convertit tout d’abord les coordonnées saisies par l’utilisateur (exemple : ‘A1’) en attribuant a i le numéro de la ligne et a j le numéro de la colonne (exemple : pour A 1, D  4 …).

En paramètre de la fonction on récupère le symbole à placer et on modifie le plateau : plateau [i][j] = symb.

La détermination des cases valides est la fonction principale du projet avec la suivante.

Cette fonction a pour but de retourner les coordonnées des cases valides dans une liste sous forme [‘A1’, …].

Un pion peut être posé si et seulement s’il permet l’encadrement d’un ou de plusieurs pions de l’adversaire.

Pour cela, on parcourt entièrement le tableau a l’aide de deux boucles pour, une comptant les lignes i et une autre comptant les colonnes j. On vérifie à chaque fois si la case contient un ‘.’, c’est-à-dire si elle est vide. Il faut ensuite étudier toute les cases autour de cette case.

Par exemple, si une case au-dessus contient un pion adverse, il faut étudier celle encore au-dessus …, tout en faisant attention de ne pas sortir du tableau. Pour éviter de sortir du tableau nous avons rajouté des conditions.

En algorithme étudions si cette case est valide pour poser une ‘X’ :

Si (plateau[i][j] = ‘.’)

//*Étude de toute les cases autours*

*//Exemple pour les cases au-dessus*

Si (i ≥ 3) et (plateau [i-1] [j] = ‘O’) *//condition pour l’étudier (au moins à la 3ème ligne)*

a  1 *//Initialisation*

*//Tant que la ligne n’est pas celle d’indice 0 et les cases au-dessus sont des ‘O’*

Tant que (i-a ≠ 0) et (plateau [i-a] [j] = ‘O’) :

a  a+1 *//On incrémente 1 pour étudier à chaque itération la ligne au-dessus*

Fin tant que

*//Cette case ne contient pas de ‘O’ car nous sommes sorties de la boucle tant que*

Si (i-a ≠ 0) et (plateau [i-a] [j] = ‘X’) :

On ajoute i et j a une liste

Fin si

Fin si

Fin si

On réalise cet algorithme en l’adaptant aux cases au-dessus à gauche, aux cases au-dessus à droite, aux cases à gauche, aux cases à droite, aux cases en dessous à gauche, aux cases en dessous et aux cases en dessous à droite. Il faut faire attention à bien mettre les conditions pour ne pas sortir du plateau.

La fonction ‘reverse\_pion’ permet le retournement des pions lors de la capture d’un pion.

La fonction est conçue comme la précédente, elle étudie toutes les possibilités dans 8 cases autour de la case de départ. Pour chacune des 8 cases, on initialise ‘a’ : a=1

Détection des pions inverses dans la ligne/ colonne ou diagonale concernée - Il faut réaliser une boucle tant que pour étudier le contenu des cases qui correspondent aux coordonnées de la première case étudiée avec a qui prend la valeur de a+1 à chaque itération. C’est-à-dire tant que la case d’après contient un pion inverse.

Si la dernière case étudiée est occupée par un pion valide (qui joue), alors on crée une boucle « pour » qui va retourner tous les pions entre la case de départ et la dernière case étudiée. Les pions inverses deviennent alors des pions valides (de la forme des pions du joueur qui joue).

Par exemple pour la case qui se trouve au-dessus à gauche :

Si (i-1 ≠ 0) et (j-1 ≠ 0) et (plateau[i-1][j-1] = pion\_inv)

a  1

Tant que (i-a  ≠ 0) et (j-a ≠ 0) et (plateau [i-a] [j-a] = pion\_inv)

a  a+1

Fin tant que

Si (i-a ≠ 0) et (j-a ≠ 0) et (plateau [i-a] [j-a] = pion)

Pour b allant de 1 à a

plateau [i-b] [j-b]  pion

Fin pour

Fin si

Fin si

Une fonction permet d’établir les conditions d’arrêts de partie. La fonction retournera 0 si la partie doit continuer et 1 sinon.

Pour qu’une partie s’arrête, il y a trois possibilités :

* le plateau est plein
* les jours sont bloquer
* un des joueurs n’a plus de pions
* l’utilisateur choisit de quitter la partie (cette fonction est mise en place dans l’architecture du programme et non dans cette fonction)

On initialise tout d’abord la variable « point » qui est un compteur qui va permettre de comptabiliser les points présents sur le plateau de jeu à chaque tour. On initialise aussi la variable valide qui sera à 1 si le jeu doit s’arrêter.

On créé tout d’abord deux boucles pour, telle que : ‘Pour i allant de 1 à taille+1’ et ‘Pour j allant de 1 à taille+1’ pour parcourir toutes les lignes et colonnes du plateau de jeu. Dans cette deuxième boucle on met une condition qui compte les points présents (les cases vides) telle que :

Si (plateau[i][j] = '.')

point  point +1

Fin si

Enfin, on ferme les boucles « pour », et on créé une condition qui va attribuer la valeur 1 à la variable valise si les joueurs ne peuvent plus jouer, c’est-à-dire si la fonction cases-valides est égale à 0 pour les deux joueurs ou si il reste 0 point sur le plateau. Cela validera alors l’arrêt de la partie.

L’affichage et la détermination du gagnant est une fonction qui se situe à la fin du programme.

Dans un premier temps, pour réaliser la fonction qui détermine le gagnant de la partie, il faut créer une boucle ‘pour’, dans une autre boucle ‘pour’, qui vont parcourir tout le plateau de jeu. Ces boucles, grâce à des conditions vont compter le nombre de pions ‘O’ et le nombre de pions ‘X’ présents sur le plateau.

En fonction des pions attribuer à chacun des joueurs et s’ils ont demandé à changer, on créé des conditions qui comparent le nombre de pions O et le nombre de pions X. Si les O sont en majorité alors le joueur concerné gagne si ce sont les X qui sont majoritaires alors c’est l’autre joueur qui gagne. Enfin si les nombre de X et de O sont égaux alors il y a match nul.

**3 – Affichages sur une interface texte classique**

L’affichage du jeu se décompose en deux parties, un affichage du menu et un affichage du plateau.

L’affichage doit contenir le plateau, le joueur qui doit jouer, le nombre de pions de chaque joueur, le numéro du tour, la liste des commandes à utiliser.

Nous avons tout d’abord commencé à réaliser deux fonctions différentes d’affichage, une pour le plateau et une pour le menu. Mais nous nous sommes rendus compte qu’il n’est pas possible d’afficher le menu à côté (sur les mêmes lignes que le plateau).

Nous avons donc par la suite créé une seule et unique fonction d’affichage.

Cette fonction permet d’afficher une ligne du plateau puis 10 espaces et enfin une ligne du menu.

Il a fallu cependant créer une autre fonction qui crée un menu en fonction du tour avec comme paramètre ‘pion’, ‘tour’ et ‘plateau’. Cette fonction retourne le menu sous forme d’une liste de taille 11 (car le plateau contient 11 lignes).

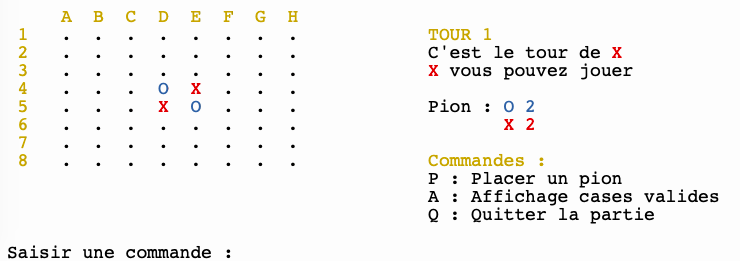
Cette fonction compte le nombre de ‘O’ et de ‘X’ dans le plateau et permet ainsi de l’afficher dans le menu à chaque tour.

Ensuite, la fonction affichage se compose tout d’abord de **l’appel de la fonction cases\_valides**. On convertit cette liste sous forme [‘D1’, …] en une liste [[1,4], …]. On convertit cette liste afin de pouvoir afficher les cases valides par des ‘ ?’.

Voici ci-dessous l’architecture principale de la fonction afficher :

Pour i allant de (0, taille+1) //Parcours du plateau

**//Affichage du plateau**

 Pour j (allant de 0, taille+1)

Afficher (plateau[i][j])

Afficher (10 \* ‘ ‘)

Fin pour

**//Affichage du menu à la suite du plateau**

Si (i ≤ 11)

Pour j allant de (0, taille(menu)-1)

Afficher (menu [i][j])

Fin pour

Fin si

Fin pour

**//Affichage des éléments du menu manquants**

Pour j allant de (i+1, 11)

Afficher (‘ ‘ + (taille\*3) \* ‘ ‘ + menu [j])

Fin pour

La fonction contient en réalité plus de condition pour l’affichage de certains éléments en couleurs comme les numéros de ‘ligne’ et ‘colonne’, les pions (‘O’ et ‘X’) et les ‘?’ des cases valides et certains éléments du menu.

Pour l’affichage du plateau et du menu, il a fallu adapter le programme avec les couleurs. Il a fallu noter que l’affichage d’un élément en couleur affiche automatiquement un espace avant l’élément.

On initialise d’ailleurs les couleurs au début de la fonction.

**4 – L’architecture du programme**

**L’architecture du programme se décompose en 3 parties :**

* Menu de présentation
* Initialisation des données
* L’imbrication dans une boucle while

Premièrement, lors du lancement du programme l’utilisateur est amené au menu de présentation (vu précédemment).

Deuxièmement, l’initialisation des données avant d’exécuter le programme tour à tour est nécessaire.

On initialise donc le plateau et les pions au centre de celui-ci. On initialise également la valeur de la variable ‘tour’ qui va compter les tours à 0 et la variable ‘action’ où sera contenue la saisie de la commande par l’utilisateur. Cette dernière saisie est nécessaire pour rentrer dans la boucle ‘while’ que nous allons voir.

Troisièmement, le contenu du programme qui sera exécuté à chaque tour est contenu dans une boucle **while**.

Les conditions de cette boucle while sont :

* la saisie de l’utilisateur de la lettre ‘Q’ lors de la saisie de la commande pour quitter la partie
* la fonction condition arrêt (1 : on arrête et 0 : on continue)

Le contenu de la boucle while :

tour  tour + 1 //Compteur de tour

**//Attribution du joueur qui doit jouer en fonction du tour (1 tour sur 2 en alternance)**

Si (tour%2 = 0)

pion  'O'

Sinon

pion  'X'

Fin si

**//Affichage plateau + menu**

menu  creation\_menu(pion,tour,plateau) //Création d'un nouveau menu personnalisé à chaque tour

affichage (pion,plateau,taille,action,menu) //Affichage du plateau à chaque tour

**//Si le joueur doit passer son tour**

Si (taille (cases\_valides (pion,plateau,taille) ) = 0)

Afficher ("Vous ne pouvez pas jouer !")

Afficher ("Vous passez votre tour")

**//Si le joueur ne passe pas son tour alors il joue**

Sinon

**//Saisie de la commande**

Afficher ("Saisir une commande : ")

action  Saisir //P : Placer un pion, A : Afficher aide, Q : Quitter

Saisie sécurisée de ‘P’, ‘A’ ou ‘Q’

**//Placement d’un pion**

Si (action = 'P')

coordonnees  saisie\_coordonnees\_pions(pion,plateau,taille)

placement\_pion (pion,coordonnees,plateau)

reverse\_pion (plateau,coordonnees,pion,taille)

**//Affichage case valide + placer un pion ou quitter**

Sinon si (action = 'A')

affichage (pion,plateau,taille,action,menu) //Réaffichage du plateau avec les cases valides

Afficher ("Saisir une commande : ")

action  Saisir //P : Placer un pion, Q : Quitter

Saisie sécurisée de ‘P’ ou ‘Q’

Si (action=='P'): #placer un pion

coordonnees  saisie\_coordonnees\_pions(pion,plateau,taille)

placement\_pion(pion,coordonnees,plateau)

reverse\_pion(plateau,coordonnees,pion,taille)

Fin si

Fin si

Fin si

A noter

La variable ‘action’ permet la saisie de la commande. Les commandes possibles sont ‘P’ pour placer un pion, ‘A’ pour réafficher le tableau avec une aide et ‘Q’ pour quitter.

Quatrièmement, une fois sortie de la boucle while (c’est-à-dire du jeu) alors on détermine le gagnant en appelant la fonction affichage\_résultat\_multi.

**5 – Les options**

**Les options :**

* Mode solo
* Revenir en arrière de N coup
* Possibilité de recommencer ou quitter à chaque fin de partie
* Supprimer la console à chaque tour

Le mode solo consiste à jouer face à l’ordinateur.  
L’utilisateur choisit le mode en début de partie. En fonction du mode choisit, le programme sera celui du mode solo ou du mode multijoueur (grâce à une condition si) :  
  
Si (mode = ‘multi’)

Programme multijoueur

Sinon

Programme solo

Fin si

**L’architecture du programme** pour le mode solo est identique à celle du mode multijoueur mis à part qu’un tour sur deux l’ordinateur joue.

Si (tour%2 = 0)

Programme ordinateur

Sinon

Programme utilisateur

Fin si

**Le programme ordinateur** consiste à choisir de manière aléatoire une case parmi la liste cases valides créée grâce à la fonction ‘cases\_valides’. Le pion est ensuite placé dans le tableau et les pions sont retournés.

Nous avons désiré créer une **fonction timer** pour pouvoir laisser quelques secondes d’attente avant que l’ordinateur joue. Pour cela nous avons créé une fonction qui met environ quatre secondes à s’exécuter. Cette fonction réalise 100 000 000 fois x 🡨 1. Cette fonction n’est pas parfaite, nous pensons néanmoins que la fonction timer préprogrammée en python est presque similaire à celle-ci. Nous avons donc préféré créer une fonction nous même plutôt qu’utiliser une fonction déjà existante.

Il a également fallu créer une **autre fonction pour l’affichage du gagnant** car les paramètres de la fonction ne sont pas les mêmes que pour le mode solo. La fonction est identique à celle du mode multijoueur (cependant elle a en paramètre un seul nom d’utilisateur et non deux).

Une des options était de laisser la possibilité à l’utilisateur de revenir en arrière de N coups. Pour cela, nous avons eu l’idée de créer une liste qui a chaque tour stocke l’état du plateau. Cependant, le plateau était pris comme une donnée modifiée, cela n’a donc pas été possible. Nous avons donc essayé de créer une fonction qui crée une liste identique, élément par élément. Cependant, nous avons bizarrement rencontré le même problème.  
Cette option n’a donc pas aboutie.

Nous avons rajouté deux options en plus :

* **La possibilité à chaque fin de partie de quitter le jeu ou de recommencer une partie**.   
  Pour cela, nous avons mis tout le programme dans une boucle tant que.

choix = ‘ R’

Tant que (choix = ‘R’)

Programme

Afficher « Pour quitter taper ‘Q’ et pour recommencer taper ‘R’ »

choix 🡨 saisir  
Fin tant que

* La deuxième est une option d’affichage pour **supprimer la console** à chaque tour.

Différentes fonctions avec l’import de librairie permettent ceci sous Windows et linux. Cependant, travaillant sur mac cette fonction ne fonctionnait pas. Nous avons donc créé une fonction qui réalise 100 retours à la ligne.

**Conclusion**

Le projet nous a tout de suite intéressé. Nous avons ainsi pris par la suite du plaisir à le réaliser.

Ce projet d’équipe nous a initié au monde de l’entreprise et nous a appris à mieux manipuler les fonctions. La mise en relation des différentes fonctions dans le projet principal nous a permis de travailler notre esprit de synthèse et notre réflexion informatique.

Les heures de travail consacrées à ce projet nous ont permis d’approfondir notre apprentissage de la programmation en python ainsi que nos connaissances, telles que les tableaux 2D, les différents affichages complexes, l’optimisation des fonctions …

Ce projet nous a permis de comprendre que la programmation est aujourd’hui derrière toutes nos utilisations informatiques, nous sommes donc fiers d’avoir pu créer, de nos propres mains, un jeu similaire au jeu Othello.