

Projet Othello - Rapport

Benchekroun Yasmine, Charles Lucas, Essakhi Jihane, Foyer Laurent

3 janvier 2023

Chapitre 1

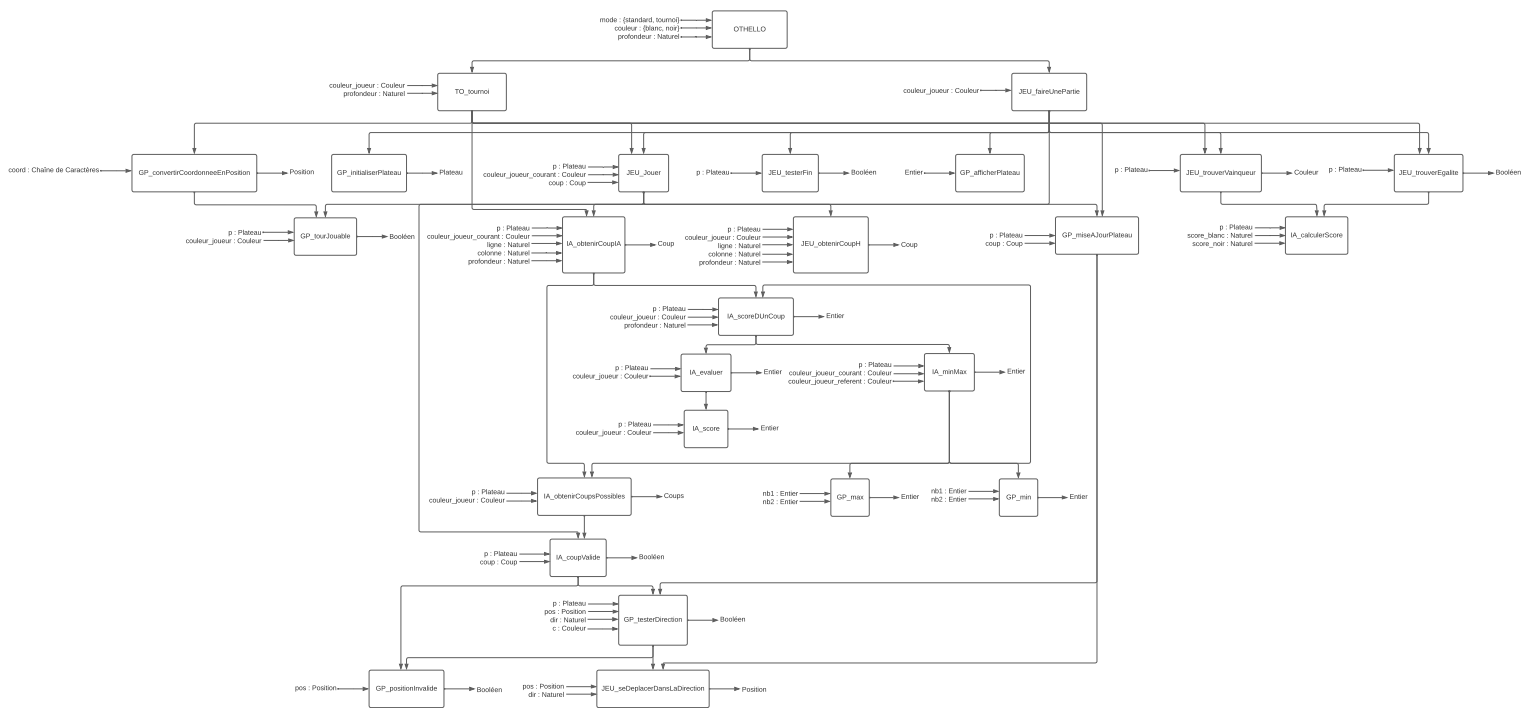
Introduction

L'Othello, une variante du Reversi, est un jeu de stratégie se jouant à deux joueurs sur un plateau de 8x8 cases : l'objectif de ce jeu est de placer des pions tour par tour de manière à contrôler le plus de cases possible sur le plateau.

Le projet détaillé dans ce rapport, réalisé au cours de 10 semaines du 25 octobre 2022 au 3 janvier 2023, consiste à la programmation d'un jeu d'Othello permettant de jouer contre un ordinateur via le développement d'une IA. Celui-ci doit permettre au joueur de choisir sa couleur (noir ou blanc) ainsi que la difficulté de l'IA (la profondeur, qui le plus élevée elle est rend en principe l'IA meilleure au jeu). De plus, le jeu doit pouvoir participer à un mode "tournoi" dans lequel il pourra affronter l'IA d'un autre programme en communiquant via un protocole prédéterminé.

Développé en C, celui-ci a dû être conçu suivant l'ensemble des étapes du cycle en V : analyse, conception préliminaire, conception détaillée, et développement, ainsi que la réalisation de leurs tests associés. Ce rapport détaille dans son ensemble les fonctions et algorithmes utilisées dans la réalisation du jeu d'Othello.

Analyse descendante



Chapitre 3

Analyses

3.1 Présentation des TAD

3.1.1 TAD Couleur

Nom: Couleur
Utilise: **Booleen**
Opérations: couleur: **Booleen** \rightarrow Couleur
couleurOpposée: Couleur \rightarrow Couleur
couleurIdentiques: Couleur \times Couleur \rightarrow **Booleen**
Axiomes: - *couleur*(1) = *BLANC*
- *couleur*(0) = *NOIR*

3.1.2 TAD Pion

Nom: Pion
Utilise: Position, Couleur
Opérations: pion: Couleur \times Position \rightarrow Pion
obtenirPosition: Pion \rightarrow Position
obtenirCouleur: Pion \rightarrow Couleur
changerCouleur: Pion \rightarrow Pion
estPlace: Pion \rightarrow **Booleen**
fixerCouleur: Pion \times Couleur \rightarrow Pion
Axiomes: - si *obtenirCouleur*(*p1*) = *NOIR* alors *obtenirCouleur*(*changerCouleur*(*p1*)) = *BLANC*
- si *obtenirCouleur*(*p1*) = *BLANC* alors *obtenirCouleur*(*changerCouleur*(*p1*)) = *NOIR*

3.1.3 TAD Position

Nom: Position
Utilise: Naturel Non Nul, Booléen, Plateau
Opérations: position: [1..8] \times [1..8] \rightarrow Position
obtenirLigne: Position \rightarrow [1..8]
obtenirColonne: Position \rightarrow [1..8]
Axiomes: - *obtenirLigne*(*position*(*x*, *y*)) = *x*
- *obtenirColonne*(*position*(*x*, *y*)) = *y*

3.1.4 TAD Plateau

Nom: Plateau
Utilise: Naturel Non Nul, Couleur, Naturel , Pion
Opérations: plateau: \rightarrow Plateau
obtenirHauteur: Plateau \rightarrow **Naturel**
obtenirLargeur: Plateau \rightarrow **Naturel**
testerFin: Plateau \rightarrow **Booleen**
estplein: Plateau \rightarrow **Booleen**
obtenirPion: Plateau \times Position \rightarrow Pion
nbPionsDUneCouleur: Plateau \times Couleur \rightarrow **Naturel**
placerPion: Plateau \times Pion \rightarrow Plateau
memePlateau: Plateau \times Plateau \rightarrow Plateau
copier: Plateau \rightarrow Plateau
effacer: Plateau \rightarrow Plateau
mettreÀJourPlateau: Plateau \rightarrow Plateau

3.1.5 TAD Coup

Nom: Coup
Utilise: Pion, Plateau
Opérations: coup: Pion \rightarrow Coup
jouerCoup: Plateau \times Coup \rightarrow Plateau
obtenirPion: coup \rightarrow Pion

3.1.6 TAD Coups

Nom: Coups
Utilise: Coup, Booléen, Naturel , ListeCoups
Opérations: coups: \rightarrow Coups
ajouterCoup: coups \times Coup \rightarrow Coups
estVide: Coups \rightarrow **Booleen**
nbCoups: Coups \rightarrow **Naturel**
obtenirlEmeCoup: Coups \times **Naturel** \rightarrow Coup
Axiomes: - *estVide(coups())*
- non *estVide(ajouterCoup(cps, c))*

3.2 Conception préliminaire

3.2.1 Signatures des fonction / procédures utilisées

— Procédure T0_tournoi(E couleur : Couleur , Profondeur: Naturel)
— Procédure JEU_faireUnePartie(E couleur_Joueur : Couleur)
— fonction GP_convertirCoordonneeEnPosition (coord : Chaîne de caractères) : Position
— fonction GP_intialiserPlateau() : Plateau
— procédure JEU_Jouer (E couleur_joueur_courant : Couleur , coup : Coup , E/S p : Plateau)
— fonction JEU_testerFin(p:Plateau):Booleen
— fonction GP_afficherPlateau(P : plateau)
— fonction JEU_troouverVainqueur(p : Plateau):Couleur

```

— fonction JEU_trouverEgalite(p : Plateau): Booleen
— procédure IA_calculerScore( E p : Plateau , E/S score_blanc , score_noir: Naturel )
— fonction GP_tourJouable( p : Plateau , couleur_joueur : Couleur) : Booleen
— fonction IA_obtenirCoupIA (p:Plateau , couleur_joueur_courant : Couleur , ligne : Naturel,
    colonne : Naturel, profondeur : Naturel):Coup
— fonction IA_scoreDUnCoup (p:Plateau , couleur_joueur : Couleur , profondeur : Naturel):Naturel
— fonction IA_evaluer(p : Plateau, couleur_joueur :Couleur) : Entier
— fonction IA_score(p : Plateau, couleur_joueur :Couleur) : Entier
— fonction minMax(p : Plateau, couleur_joueur_courant , couleur_joueur_referent : Couleur)
    : Entier
— fonction IA_obtenirCoupsPossibles(p : Plateau , couleur_joueur : Couleur):Coups
— fonction GP_max(nb1 , nb2 : Entier):Entier
— fonction GP_min(nb1 , nb2 : Entier):Entier
— fonction IA_coupValide(p : Plateau , coup : Coup):Booleen
— fonction JEU_obtenirCoupH (p : Plateau , couleur_Joueur : Couleur , ligne : Naturel, colonne
    : Naturel, profondeur : Naturel): Coup
— Procédure GP_MiseAJourPlateau( E coup : Coup , E/S p: Plateau)
— fonction GP_testerDirection(p : Plateau pos : Position dir : Naturel c : Couleur)Booleen
— fonction GP_positionInvalide(pos : Position):Booleen
— fonction JEU_seDeplacerDansLaDirection(pos : Position , dir : Naturel): Position

```

Chapitre 4

Conception détaillée

4.1 Conception détaillée des TAD

- TAD Couleur = Structure
 - couleur = **Booleen**
- TAD Pion = Structure
 - position = Position
 - couleur = Couleur
 - estPlacé = **Booleen**
- TAD Position = Structure
 - ligne = 1..8
 - colonne = 1..8
- TAD Plateau = Structure
 - lesPions = Tableau[8][8] de Pions
 - estPlein = Booléen
 - nbPionsPlaces = 1..64
- Coup = Pion
- Coups = Liste [1..MAX] de Coup

4.2 Conception détaillée des fonctions et procédures

Fonctions et procédures qui permettent de jouer une partie

Procédure JEU_faireUnePartie

procédure JEU_faireUnePartie (**E** couleur_choisie : Couleur)

Déclaration p : Plateau , profondeur : **Naturel**, vainqueur ; Couleur , egalite : **Booleen**, nb_tour : **Naturel**, couleur_joueur_humain , couleur_joueur_courant : Couleur , ligne : **Naturel**, colonne : **Caractere**, str : Tableau[1..2] de **Caractere**

debut

nb_tour ← 0
couleur_joueur_humain ← couleur_choisie
couleur_joueur_courant ← couleur_joueur_humain
p ← GP_initialiserPlateau()
IHM_gras("Vous commencez!")
IHM_gras("Avec quelle profondeur voulez vous que l'IA joue?")

```

lire(profondeur)
tant que non JEU_testerFin(p) faire
    nb_tour ← nb_tour + 1
    IHM_separateur()
    GP_afficherPlateau(p)
    si couleursIdentiques(couleur_joueur_courant,couleur_joueur_humain) alors
        IHM_gras("À vous de jouer!")
        repetier
            IHM_gras("Quel coup voulez-vous jouer? Entrez une lettre suivie d'un chiffre : ")
            lire(str)
            ligne ← str[1]
            colonne ← str[0]
            si non IA_coupValide(p,coup(pion(couleur_joueur_humain,position(ligne,colonne)))) alors
                IHM_gras("Coup non valide, réessayez!")
            finsi
        jusqu'a ce que
            IA_coupValide(p,coup(TADPion_pion(couleur_joueur_humain,Position_position(ligne,colonne))))

        JEU_jouer(p,couleur_joueur_courant,JEU_obtenirCoupH,ligne,colonne,3)
    sinon
        JEU_jouer(p,couleur_joueur_courant, IA_obtenirCoupIA,0,0,profondeur)
    finsi
    couleur_joueur_courant ← couleurOpposee(couleur_joueur_courant)
fintantque
IHM_gras("La partie est terminée! ")
egalite ← JEU_trouverEgalite(p)
si non egalite alors
    vainqueur ← JEU_trouverVainqueur(p)
    ecrire(" Le vainqueur est la couleur",obtenircouleur(vainqueur))
sinon
    GP_afficherPlateau(p)
    ecrire("Pas de vainqueur!")
finsi
fin

```

Procédure JEU_jouer

procédure JEU_jouer (**E/S** p : Plateau **E** couleur_joueur_courant : Couleur)

Déclaration coup_joue : Coup

debut

```

si GP_tourJouable(p, couleur_joueur_courant) alors
    coup_joue ← obtenirCoup(p,couleur_joueur_courant,3)
    JEU_jouerCoup(p,coup_joue)
    GP_miseAJourPlateau(p,coup_joue)

```

sinon

```

    ecrire(" Tour sauté!")

```

finsi

fin

Fonction JEU_obtenirCoupH

fonction JEU_obtenirCoupH (p : plateau , couleur_joueur : Couleur , ligne : **Naturel**, colonne : **Caractere**, profondeur : **Naturel**) : Coup

Déclaration coup : Coup

debut

 coup ← coup(pion(couleur_joueur, position(ligne,colonne)))

retourner coup

fin

Fonction seDeplacerDansLaDirection

procédure seDeplacerDansLaDirection (**E/S** pos : Position, **E** dir : **Naturel**)

Déclaration dirLigne , dirColonne : Tableau [1..8] d'Entier

debut

 dirLigne ← [-1,-1,0,1,1,1,0,-1]

 dirColonne ← [0,1,1,1,0,-1,-1,-1]

retourner Position_position(Position_obtenirLigne(pos) + dirLigne[dir] , Position_obtenirColonne(pos) + dirColonne[dir])

fin

JEU_trouverVainqueur

fonction JEU_trouverVainqueur (p1 : Plateau) : Couleur

Déclaration Taille : **Naturel**, egalite : **Naturel**, **Naturel**egalite , vainqueur : Couleur , score_blanc, score_noir : **Naturel**

debut

 TAILLE ← 8

 egalite ← (score_blanc = score_noir)

si egalite **alors**

 vainqueur ← couleur(Vrai)

sinon

si score_blanc > score_noir **alors**

 vainqueur ← couleur(vrai)

sinon

 vainqueur ← couleur(Faux)

finsi

finsi

retourner vainqueur

fin

fonction JEU_trouverEgalite

fonction JEU_trouverEgalite (p1 : Plateau) : **Booleen**

Déclaration score_blanc , score_noir : **Naturel**, egalite : **Booleen**

debut

```

    JEU_calculerScore(pl,score_blanc,score_noir)
    egalite ← (score_blanc = score_noir)
    retourner egalite
fin

```

JEU_testerFin

```

fonction JEU_testerFin (p : Plateau ) : Booleen
debut
    si (Plateau_estPlein(p)) et (non GP_tourJouable(p, couleur(1))) alors
        retourner 1))
    si non GP_tourJouable(p, couleur(0)) alors
        retourner 1
    sinon
        retourner 0
    finsi
finsi
fin

```

Procédure JEU_calculerScore

procédure JEU_calculerScore (**E** p1 : Plateau **E/S** score_blanc : **Naturel**, score_noir : **Naturel**)

Déclaration TAILLE : **Naturel**, blanc , noir : Couleur , position : Position , pion : Pion

```

debut
    TAILLE ← 8
    blanc ← couleur(Vrai)
    noir ← couleur(Faux)
    score_blanc ← 0
    score_noir ← 0
    pour i ← 1 à TAILLE faire
        pour j ← 1 à TAILLE faire
            position ← position(i,j)
            pion ← obtenirPion(pl, position)
            si estPlace(pion) alors
                si couleursIdentiques(obtenirCouleur(pion),blanc) alors
                    score_blanc ← score_blanc + 1
                finsi
                si couleursIdentiques(obtenirCouleur(pion),noir) alors
                    score_noir ← score_noir + 1
                finsi
            finsi
        finsi
    finpour
finpour
fin

```

Fonctions Procédures liées à la gestion du plateau

Fonction GP_convertirCoordonneeEnPosition

fonction GPconvertirCoordonneeEnPosition (coord : Tableaux[1..Max] de **Caractere**) : Position

Déclaration ligne, colonne : **Naturel**, A , a , zero : **Caractere**

debut

A ← "A"

a ← "a"

0 ← "0"

si coord[1] >= 'A' et coord [1] <= 'H' **alors**

colonne ← coord[2] - A +1

sinon

colonne ← coord[1] - a +1

ligne ← coord - zero

fin

retourner TADPositionposition(ligne, colonne)

fin

Procédure GP_afficherPlateau

procédure GP_afficherPlateau (p : Plateau)

Déclaration i, j : **Naturel**

debut

retourALaLigne()

ecrire(" A B C D E F G H")

retourALaLigne()

pour i ← 0 à 8 **faire**

ecrire(i+1, " ")

pour j ← 0 à 8 **faire**

si TADPlateau_obtenirPion(p, TADPosition_position(i,j)).estPlace **alors**

ecrire(TADPion_obtenirCouleur(TADPlateau_obtenirPion(p, TADPosition_position(i,j))).couleur)

sinon

ecrire(() "- ")

fin

finpour

retourALaLigne()

finpour

retourALaLigne()

fin

Fonction GP_initialiserPlateau

fonction GP_initialiserPlateau () : Plateau

Déclaration p : Plateau

debut

p ← TADPlateau_plateau()

```

    TADPlateau_placerPion(p, TADPion_pion(TADCouleur_couleur(1), TADPosition_position(3, 3)))
    TADPlateau_placerPion(p, TADPion_pion(TADCouleur_couleur(1), TADPosition_position(4, 4)))
    TADPlateau_placerPion(p, TADPion_pion(TADCouleur_couleur(0), TADPosition_position(3, 4)))
    TADPlateau_placerPion(p, TADPion_pion(TADCouleur_couleur(0), TADPosition_position(4, 3)))
    retourner p
fin

```

Fonction GP_positionInvalide

fonction GP_positionInvalide (pos : Position) : **Booleen**

Déclaration

debut

retourner TADPosition_obtenirColonne(pos) < 0 OU TADPosition_obtenirColonne(pos) > 7 OU TADPosition_obtenirLigne(pos) < 0 OU TADPosition_obtenirLigne(pos) > 7

fin

Fonction GP_tourJouable

fonction GP_tourJouable (p : Plateau, couleur_joueur : Couleur) : **Booleen**

Déclaration i, j, dir : **Naturel**

debut

pour i ← 0 à 8 **faire**

pour j ← 0 à 8 **faire**

si NON TADPlateau_obtenirPion(p, TADPosition_position(i, j)).estPlace **alors**

pour dir ← 0 à 8 **faire**

si GP_testerDirection(p, TADPosition_position(i, j), dir, couleur_joueur) **alors**

retourner VRAI

finsi

finpour

finsi

finpour

finpour

retourner FAUX

fin

Fonction GP_testerDirection

fonction GP_testerDirection (p : Plateau, pos : Position, dir : **Naturel**, couleur_joueur : Couleur) : **Booleen**

Déclaration directionValide : **Booleen**

debut

directionValide ← FAUX

repeter

pos ← JEU_seDeplacerDansLaDirection(pos, dir)

si GP_positionInvalide(pos) **alors**

retourner FAUX

sinon

```

    si sequenceDePions(p, pos, couleur_joueur) alors
        retourner FAUX
    sinon
        si NON sequenceDePions(p, pos, couleur_joueur) alors
            tant que TADPlateau_obtenirPion(p, pos).estPlace ET NON directionValide ET NON
            GP_positionInvalide(pos) faire
                pos ← JEU_seDeplacerDansLaDirection(pos, dir)
                si GP_positionInvalide(pos) alors
                    retourner FAUX
                finsi
            si NON sequenceDePions(p, pos, couleur_joueur) alors
                retourner VRAI
            finsi
        fintantque
    finsi
fin

```

jusqu'a ce que TADPlateau_obtenirPion(p, pos).estPlace ET NON directionValide
retourner directionValide
fin

Fonction GP_miseAJourPlateau

procédure GP_miseAJourPlateau (**E/S** p : Plateau, **E** coup : coup)

Déclaration pos : Position
 dir : **Naturel**
 pion : Pion

debut

```

    pour dir ← 0 à 8 faire
        pos ← TADPion_obtenirPosition(TADCoup_obtenirPion(coup))
        directionValide ← GP_testerDirection(p, pos, dir)
        TADPion_obtenirCouleur(TADCoup_obtenirPion(coup))
        si directionValide alors
            pos ← JEU_seDeplacerDansLaDirection(pos, dir)
            tant que sequenceDePions(p, pos, couleur_joueur) faire
                pion ← TADPlateau_obtenirPion(p, pos)
                TADPion_changerCouleur(pion)
                TADPlateau_placerPion(p, pion)
                pos ← JEU_seDeplacerDansLaDirection(pos, dir)
            fintantque
        finsi
    finpour
fin

```

Fonction GP_max

fonction GP_max (nb1, nb2 : **Entier**) : **Entier**

Déclaration

debut

```

    si nb1>nb2 alors
        retourner nb1
    sinon
        retourner nb1
    finsi
fin

```

Fonction GP_min

fonction GP_min (nb1, nb2 : Entier) : Entier

```

    Déclaration
debut
    si nb1<nb2 alors
        retourner nb1
    sinon
        retourner nb1
    finsi
fin

```

Fonction GP_nbPionsContigus

fonction GP_nbPionsContigus (p : Plateau, pion : pion) : Naturel

```

    Déclaration  resultat, dir : Naturel
                  directionValide : Booleen
                  pos : Position
debut
    pour dir ←0 à 8 faire
        directionValide ← GP_testerDirection(p, pos, dir, TADPion_obtenirCouleur(pion))
        tant que sequenceDePions(p, pos, couleur_joueur) ET directionValide faire
            pos ← JEU_seDeplacerDansLaDirection(pos, dir)
            resultat ← resultat + 1
        fintantque
    finpour
    retourner resultat
fin

```

Fonctions et procédures liés à l'IA

Fonction IA_coupValide

fonction IA_coupValide (p : Plateau , coup : Coup) : Booleen

```

    Déclaration  pos : Position , resultat : Booleen, i : Naturel
debut
    pos ← obtenirPosition(obtenirPion(coup))
    resultat ← 0
    i ← 1
    si obtenirPion(p,coup.pion.position).estPlace et positionInvalide(pos) alors
        resultat ← Faux

```

```

    sinon
        tant que non resultat et  $i \leq 8$  faire
            resultat  $\leftarrow$  testerDirection(p, pos, i, coup.pion.couleur)
            i  $\leftarrow$  i+1
        fintantque
    fin
    retourner resultat
fin

```

Fonction IA_evaluer

```

fonction IA_evaluer (p : Plateau ,couleur_joueur : Couleur) : Naturel
debut
    retourner IA_score(p,couleur_joueur) - IA_score(p,couleurOpposee(couleur_joueur))
fin

```

Fonction IA_minMax

```

fonction IA_minMax (p : Plateau , couleur_joueur_courant : Couleur , couleur_joueur_referent : Couleur
, profondeur : Naturel) : Naturel
    Déclaration resultat : Naturel, coups_possibles : Coups , score,nb_coups_possibles : Naturel, i :
    Naturel
debut
    coups_possibles  $\leftarrow$  IA_obtenirCoupsPossibles(p,couleur_joueur_courant)
    nb_coups_possibles  $\leftarrow$  Coups_obtenirNbCoups(coups_possibles)
    si nb_coups_possibles = 0 alors
        retourner 0
    sinon
        resultat  $\leftarrow$  IA_scoreDUnCoup(p,Coups_obtenirIEmeCoup(coups_possibles,0),couleur_joueur_courant,profondeur)

        pour i  $\leftarrow$  2 à nb_coups_possibles faire
            score  $\leftarrow$  IA_scoreDUnCoup(p,obtenirIEmeCoup(coups_possibles,i),couleur_joueur_courant,profondeur)

            si Couleur_couleursIdentiques(couleur_joueur_courant,couleur_joueur_referent) alors
                resultat  $\leftarrow$  GP_max(resultat,score)
            sinon
                resultat  $\leftarrow$  GP_min(resultat,score)
            finsi
        finpour
        retourner resultat
    finsi
fin

```

Fonction IA_obtenirCoupIA

```

fonction IA_obtenirCoupIA (p : Plateau , couleur_joueur_courant : Couleur , ligne : Naturel, colonne :
Caractere, profondeur : Naturel) : Coup
    Déclaration resultat : Coup , coups_possibles : Coups , score : Naturel, meilleur_score : Naturel,
    i : Naturel

```

```

debut
  coups_possibles ← IA_obtenirCoupsPossibles(p,couleur_joueur_courant)
  resultat ← obtenirIEmeCoup(coups_possibles,0)
  meilleur_score ← IA_scoreDUnCoup(p,resultat,couleur_joueur_courant,profondeur)
  pour i ← 1 à obtenirNbCoups(coups_possibles) faire
    score ← IA_scoreDUnCoup(p,obtenirIEmeCoup(coups_possibles,i),couleur_joueur_courant,profondeur)

    si score>meilleur_score alors
      resultat ← obtenirIEmeCoup(coups_possibles,i)
      meilleur_score ← score
    finsi
  finpour
  retourner resultat
fin

```

Fonction IA_obtenirCoupsPossibles

fonction IA_obtenirCoupsPossibles (p : Plateau , couleur_joueur : Couleur) : Coups

Déclaration coups_possibles : Coups , coup : Coup , i , j : Naturel

```

debut
  coups_possibles ← coups()
  pour i ← 1 à 8 faire
    pour j ← 1 à 8 faire
      si IA_coupValide(p, coup (pion(couleur_joueur, position(i, j)))) alors
        coup ← coup(pion(couleur_joueur, position(i, j)))
        ajouterCoup(coups_possibles, coup)
      finsi
    retourner coups_possibles
  finpour
finpour
fin

```

Fonction IA_score

fonction IA_score (p : Plateau , couleur_joueur : Couleur) : Naturel

Déclaration score, colonne , ligne : Naturel, pion : Pion , couleur : Couleur , position : Position

```

debut
  score ← 0
  pour colonne ← 1 à 8 faire
    pour ligne ← 1 à 8 faire
      position ← position(ligne,colonne)
      pion ← obtenirPion(p,position)
      couleur ← obtenirCouleur(pion)
      si couleursIdentiques(couleur,couleur_joueur) alors
        score ← score + 1
      finsi
    finpour
  finpour
fin

```


Fonction IA_scoreDUnCoup

fonction IA_scoreDUnCoup (p : Plateau , coup : Coup , couleur_joueur : Couleur , profondeur : **Naturel**) : **Naturel**

Déclaration copie_plateau2 : Plateau , copie_plateau : Plateau

debut

 copie_plateau2 ← plateau()

 copie_plateau ← copie_plateau2

 jouerCoup(copie_plateau,coup)

si estPlein(copie_plateau) et profondeur = 0 **alors**

retourner IA_evaluer(copie_plateau,couleur_joueur)

sinon

retourner IA_minMax(copie_plateau,couleur_joueur,couleur_joueur,profondeur)

finsi

fin

Procédure qui permet de faire un Tournoi

Procédure TO_tournoi

procédure TO_tournoi (**E** couleur_choisie : Couleur , profondeur : **Naturel**)

Déclaration p : Plateau , pos : Position , coup : Coup , coup_possible , coup_possible_precedent , a_nous_de_jouer , egalite : **Booleen**, vainqueur : Couleur , pion : Pion , saisie : Tableau [1..8] de caractère

debut

 p ← GP_initialiserPlateau()

 a_nous_de_jouer ← non couleursIdentiques(Couleur_couleur(1),couleur_choisie)

 coup_possible ← Vrai

tant que Vrai **faire**

si a_nous_de_jouer **alors**

 coup_possible_precedent ← coup_possible

 coup_possible ← GP_tourJouable(p,couleur_choisie)

si coup_possible **alors**

 coup ←

 IA_obtenirCoupIA(p,couleur_choisie,0,0,profondeur)

 JEU_jouer(p,couleur_choisie,IA_obtenirCoupIA,0,0,profondeur)

sinon

si coup_possible_precedent **alors**

ecrire("passe")

sinon

 egalite ← JEU_trouverEgalite(p)

 vainqueur ← JEU_trouverVainqueur(p)

si egalite **alors**

ecrire("nulle")

sinon

ecrire(couleursIdentiques(vainqueur,Couleur_couleur(1))) "blanc" : "noir")

finsi

finsi

finsi

sinon

lire(saisie)

```

    si sontEgalesIC(saisie , "passe") alors
        pos ← GP_convertirCoordonneeEnPosition(saisie)
        pion ← pion(TADCouleur_couleurOpposee(couleur_choisie), pos)
        coup ← coup(pion)
        jouerCoup(p,coup)
        GP_miseAJourPlateau(p, coup)
    finsi
finsi
    a_nous_de_jouer ← non a_nous_de_jouer
tantque
fin

```

Chapitre 5

Implémentation en C

Fonctions C des TAD

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <assert.h>
5  #include <stdbool.h>
6  #include "TAD_Couleur.h"
7  #include "TAD_Pion.h"
8
9
10 Couleur TADCouleur_couleur(bool param){
11     Couleur couleur;
12
13     /* si param est 0 c'est noir, sinon blanc */
14     couleur.couleur = param;
15     return couleur;
16 }
17
18 Couleur TADCouleur_couleurOpposee(Couleur couleur){
19     Couleur couleurOpposee;
20
21     couleurOpposee.couleur = !couleur.couleur;
22     return couleurOpposee;
23 }
24
25 bool TADCouleur_couleursIdentiques(Couleur couleur1, Couleur couleur2){
26     return (couleur1.couleur == couleur2.couleur);
27 }
28
29 char* TADCouleur_afficherCouleur(Couleur couleur){
30     if (couleur.couleur == 1)
31         return "blanc";
32     else
33         return "noir";
34 }
```

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <assert.h>
5  #include <stdbool.h>
6  #include "TAD_Coup.h"
7  #include "TAD_Pion.h"
8  #include "TAD_Plateau.h"
9  #include "IA_CoupValide.h"
10
11
12  /* Partie publique */
13
14  Coup TADCoup_coup (Pion pion){
15      Coup coup;
16      coup.pion = pion;
17      return coup;
18  }
19
20  Pion TADCoup_obtenirPion(Coup coup){
21      return coup.pion;
22  }
23
24
25  void TADCoup_jouerCoup(Plateau *p, Coup coup){
26      TADPlateau_placerPion(p, coup.pion);
27  }

```

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <assert.h>
5  #include <stdbool.h>
6  #include "TAD_Coups.h"
7
8
9  /* Fonctions de Liste Coups (privées)*/
10
11  void ListeCoups_ajouterCoup(ListeCoups *liste_coups, Coup c){
12      ListeCoups noeud = (ListeCoups) malloc (sizeof(Ncoups));
13
14      ListeCoups_fixerCoup(&noeud, c);
15      ListeCoups_fixerListeSuivante(noeud, *liste_coups);
16      *liste_coups = noeud;
17  }
18  /*-----*/
19  void ListeCoups_supprimerCoup(ListeCoups *liste_coups){
20      ListeCoups liste_coups_temp = *liste_coups;
21      *liste_coups = ListeCoups_obtenirListeSuivante(*liste_coups);
22      free(liste_coups_temp);
23  }

```

```

24  /*-----*/
25  void ListeCoups_supprimerIEmeCoupRecuratif(ListeCoups *liste_coups , unsigned
    int i){
26      if(i == 0){
27          ListeCoups_supprimerCoup(liste_coups);
28      }
29      else{
30          ListeCoups liste_coups_suivante = ListeCoups_obtenirListeSuivante(*
            liste_coups);
31          ListeCoups_supprimerIEmeCoupRecuratif(&liste_coups_suivante , i-1);
32      }
33  }
34  /*-----*/
35  Coup ListeCoups_obtenirCoup(ListeCoups liste_coups){
36      return liste_coups->coup;
37  }
38  /*-----*/
39  void ListeCoups_fixerCoup(ListeCoups *liste_coups , Coup c){
40      (*liste_coups)->coup = c;
41  }
42  /*-----*/
43  ListeCoups ListeCoups_obtenirListeSuivante(ListeCoups liste_coups){
44      return (*liste_coups).listeSuivante;
45  }
46  /*-----*/
47  void ListeCoups_fixerListeSuivante(ListeCoups liste_coups , ListeCoups
    liste_coups_suivante){
48      (*liste_coups).listeSuivante = liste_coups_suivante;
49  }
50
51
52  /*-----*/
53  /*-----*/
54
55
56  /* Fonctions de Coups (publiques) */
57  void TADCoups_fixerNbCoups(Coups *coups , unsigned int nb_coups){
58      (*coups).nbCoups = nb_coups;
59  }
60  /*-----*/
61  void TADCoups_fixerListeCoups(Coups *coups , ListeCoups l){
62      (*coups).listecoups = l;
63  }
64  /*-----*/
65  unsigned int TADCoups_obtenirNbCoups(Coups coups){
66      return coups.nbCoups ;
67  }
68  /*-----*/
69  Coups TADCoups_coups () {
70      Coups coups;
71      /* on fixe le nombre de coups à l'origine à 0*/

```

```

72     TADCoups_fixerNbCoups(&coups,0);
73     /*on initialise la liste de coups à la liste vide*/
74     coups.listecoups = NULL;
75     return coups;
76 }
77 /*-----*/
78 ListeCoups *TADCoups_obtenirListeCoups(Coups *coups){
79     return &(coups->listecoups);
80 }
81 /*-----*/
82 void TADCoups_ajouterCoup(Coups *coups, Coup coup){
83
84     ListeCoups_ajouterCoup(TADCoups_obtenirListeCoups(coups),coup);
85     TADCoups_fixerNbCoups(coups,TADCoups_obtenirNbCoups(*coups) + 1);
86 }
87 /*-----*/
88 bool TADCoups_estVide (Coups coups){
89     return (coups.listecoups == NULL);
90 }
91 /*-----*/
92 Coup TADCoups_obtenirIEmeCoup(Coups coups, unsigned int i){
93     /* on vérifie la précondition */
94     assert(TADCoups_obtenirNbCoups(coups) >= i);
95
96     // et on démarre la récursivité
97     return TADCoups_obtenirIEmeCoupRecuratif(*TADCoups_obtenirListeCoups(&coups
98         ), i);
99 }
100 /*-----*/
101 Coup TADCoups_obtenirIEmeCoupRecuratif(ListeCoups liste_coups, unsigned int i){
102     /* si i vaut 0 c'est qu'on est à l'élément qu'on cherche*/
103     if(i == 0){
104         return ListeCoups_obtenirCoup(liste_coups);
105     }
106     /* sinon on obtient la liste suivante en diminuant i*/
107     else{
108         return TADCoups_obtenirIEmeCoupRecuratif(
109             ListeCoups_obtenirListeSuivante(liste_coups), i-1);
110     }
111 }

```

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <stdbool.h>
5  #include "TAD_Pion.h"
6  #include "TAD_Couleur.h"
7
8
9  /* Partie publique */
10

```

```

11 Pion TADPion_pion(Couleur couleur, Position pos){
12     Pion pion;
13     pion.position = pos;
14     pion.couleur = couleur;
15     pion.estPlace = 0;
16     return pion;
17 }
18
19
20
21
22 Position TADPion_obtenirPosition(Pion pion){
23     return pion.position;
24 }
25
26 bool TADPion_estPlace(Pion p) {
27     return p.estPlace;
28 }
29
30 Couleur TADPion_obtenirCouleur(Pion pion){
31     return pion.couleur;
32 }
33
34 void TADPion_changerCouleur(Pion *pion){
35
36     if (TADPion_obtenirCouleur(*pion).couleur == TADCouleur_couleur(0).couleur)
37     {
38         pion->couleur = TADCouleur_couleur(1);
39     }
40     else{
41         pion->couleur = TADCouleur_couleur(0);
42     }
43 }
44
45 void TADPion_fixerCouleur(Pion *pion, Couleur couleur){
46     pion->couleur = couleur;
47 }
48
49 void TADPion_placerPion(Pion pion){
50     pion.estPlace = true;
51 }

```

```

1 #include <stdlib.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <assert.h>
5 #include <stdbool.h>
6 #include "TAD_Plateau.h"
7 #include "TAD_Pion.h"
8 #include "TAD_Position.h"

```

```

9  #include "TAD_Coups.h"
10
11
12
13  /* Partie publique */
14
15  Plateau TADPlateau_plateau() {
16      Plateau p;
17      unsigned int TAILLE;
18      TAILLE = 8;
19      p.estPlein = false;
20      p.nbPionsPlaces = 0;
21
22      for (int i=0; i<TAILLE; i++) {
23          for (int j=0; j<TAILLE; j++) {
24              p.lesPions[i][j].couleur = TADCouleur_couleur(0);
25              p.lesPions[i][j].estPlace = 0;
26              p.lesPions[i][j].position = TADPosition_position(i, j);
27          }
28      }
29      return (p);
30  }
31
32
33  bool TADPlateau_estPlein(Plateau p){
34      unsigned int TAILLE;
35      TAILLE = 8;
36      for(int i=0; i<TAILLE; i++){
37          for(int j=0; j<TAILLE; j++){
38              if (!TADPlateau_obtenirPion(p, TADPosition_position(i, j)).estPlace
39                  ) {
40                  return 0;
41              }
42          }
43      }
44      return 1;
45  }
46
47  Pion TADPlateau_obtenirPion(Plateau p, Position pos) {
48      Pion pion = p.lesPions[TADPosition_obtenirLigne(pos)][
49          TADPosition_obtenirColonne(pos)];
50      return pion;
51  }
52
53  void TADPlateau_placerPion(Plateau *p, Pion pion){
54      int ligne, colonne;
55      ligne = TADPosition_obtenirLigne(pion.position);
56      colonne = TADPosition_obtenirColonne(pion.position);
57      p->lesPions[ligne][colonne].estPlace = 1;
58      p->lesPions[ligne][colonne].couleur = pion.couleur;
59      p->nbPionsPlaces = p->nbPionsPlaces + 1;

```



```

58 }
59
60 /* fonctions liées à l'utilisation de l'allocation dynamique */
61 /*Plateau* TADPlateau_copier(Plateau *p){
62     Plateau* copie;
63     unsigned int TAILLE;
64     TAILLE = 8;
65     Plateau cp = TADPlateau_plateau();
66
67     copie = &cp;
68     for(int i=0; i<TAILLE; i++){
69         for(int j=0; j<TAILLE; j++){
70             if (TADPion_estPlace(*(p->lesPions[i][j]))){
71                 //copie->lesPions[i][j]=p->lesPions[i][j];
72                 memmove(copie->lesPions[i][j],p->lesPions[i][j], sizeof(Pion)*
73                     TAILLE*TAILLE);
74             }
75         }
76     }
77
78     return copie;
79 }*/
80
81 void TADPlateau_copier(Plateau p, Plateau *copie){
82     //Plateau copie = TADPlateau_plateau();
83     for(int ligne = 0; ligne < 8; ligne++){
84         for(int colonne = 0; colonne < 8; colonne++){
85             copie->lesPions[ligne][colonne].estPlace = p.lesPions[ligne][
86                 colonne].estPlace;
87             copie->lesPions[ligne][colonne].couleur = p.lesPions[ligne][
88                 colonne].couleur;
89         }
90     }
91     //return copie;
92 }
93
94
95
96 bool TADPlateau_memePlateau(Plateau p1, Plateau p2){
97     bool res;
98     unsigned int TAILLE;
99     unsigned int i, j;
100     Position pos;
101     TAILLE = 8;
102     res= true;
103     i=0;
104     j=0;
105     pos = TADPosition_position(i, j);

```

```

106     while ((res==1) && (j<=TAILLE)) {
107         if (!TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(
            TADPlateau_obtenirPion(p1,pos)),TADPion_obtenirCouleur(
            TADPlateau_obtenirPion(p2,pos))))
108             res=0;
109
110         else {
111             i++;
112             if (i>TAILLE) {
113                 j++;
114                 i=1;
115             }
116
117         }
118     }
119 }
120     return res;
121 }
122
123 void TADPlateau_effacer(Plateau* p) {
124     unsigned int TAILLE = 8;
125     for(int i=0; i<TAILLE; i++){
126         for(int j=0; j<TAILLE; j++){
127             //free(p->lesPions[i][j]);
128
129         }
130     }
131     //free(p);
132 }

```

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <assert.h>
5  #include <stdbool.h>
6  #include "TAD_Position.h"
7
8
9  Position TADPosition_position(unsigned int ligne, unsigned int colonne){
10     Position pos;
11
12     pos.ligne = ligne;
13     pos.colonne = colonne;
14     return pos;
15 }
16
17 unsigned int TADPosition_obtenirLigne(Position pos){
18     return pos.ligne;
19 }
20
21 unsigned int TADPosition_obtenirColonne(Position pos){

```

```

22     return pos.colonne;
23 }

```

Fonctions en C liées à la gestion du plateau

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <stdbool.h>
4  #include "TAD_Plateau.h"
5  #include "TAD_Coup.h"
6  #include "TAD_Coups.h"
7  #include "TAD_Position.h"
8  #include "TAD_Couleur.h"
9  #include "TAD_Pion.h"
10 #include "JEU_SeDeplacerDansLaDirection.h"
11 #include "UNIT_GestionPlateau.h"
12
13
14 Position GP_convertirCoordonneeEnPosition(char str[]) {
15     int ligne, colonne;
16     ligne = atoi((const char*)&str[1]);
17     colonne = str[0];
18     if(colonne > 96 && colonne < 105) colonne = colonne - 32;
19
20     ligne = ligne - 1;
21     colonne = (int) colonne - 65;
22     return(TADPosition_position(ligne, colonne));
23 }
24
25
26 void GP_afficherPlateau(Plateau p){
27     printf("\n\n  A B C D E F G H\n\n");
28     for(int i=0; i<8; i++){
29         printf("%d  ", i+1);
30         for(int j=0; j<8; j++){
31             if(TADPlateau_obtenirPion(p, TADPosition_position(i,j)).estPlace){
32                 printf("%d  ", TADPion_obtenirCouleur(TADPlateau_obtenirPion(p,
33                     TADPosition_position(i,j))).couleur);
34             }
35             else{
36                 printf("— ");
37             }
38         }
39         printf("\n");
40     }
41     printf("\n");
42 }
43
44 Plateau GP_initialiserPlateau(){
45     Plateau p;
46     p = TADPlateau_plateau();

```

```

46 // pions blancs (1)
47 TADPlateau_placerPion(&p, TADPion_pion(TADCouleur_couleur(1),
48   TADPosition_position(3, 3)));
49 TADPlateau_placerPion(&p, TADPion_pion(TADCouleur_couleur(1),
50   TADPosition_position(4, 4)));
51 // pions noirs (0)
52 TADPlateau_placerPion(&p, TADPion_pion(TADCouleur_couleur(0),
53   TADPosition_position(3, 4)));
54 TADPlateau_placerPion(&p, TADPion_pion(TADCouleur_couleur(0),
55   TADPosition_position(4, 3)));
56
57 return (p);
58 }
59
60 bool GP_positionInvalide(Position pos){
61   return (TADPosition_obtenirColonne(pos) < 0 ||
62     TADPosition_obtenirColonne(pos) > 7 ||
63     TADPosition_obtenirLigne(pos) < 0 ||
64     TADPosition_obtenirLigne(pos) > 7);
65 }
66
67 bool GP_tourJouable(Plateau p, Couleur couleur_joueur){
68   for(unsigned int i=0; i<8; i++){
69     for(unsigned int j=0; j<8; j++){
70       if(!TADPlateau_obtenirPion(p, TADPosition_position(i, j)).estPlace
71         ){
72         for(unsigned int dir=0; dir<8; dir++){
73           if(GP_testerDirection(p, TADPosition_position(i, j), dir,
74             couleur_joueur)){
75             return 1;
76           }
77         }
78       }
79     }
80   }
81   return 0;
82 }
83
84 bool GP_testerDirection(Plateau p, Position pos, unsigned int dir, Couleur
85   couleur_joueur) {
86   bool directionValide = 0;
87
88   do{
89     pos = JEU_seDeplacerDansLaDirection(pos, dir);
90     if(GP_positionInvalide(pos)){
91
92       return 0;
93     }
94   }
95   else{

```

```

89         if (TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(
          TADPlateau_obtenirPion(p, pos)), couleur_joueur) &&
          TADPlateau_obtenirPion(p, pos).estPlace){
90             return 0;
91         }
92         else{
93         if (!TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(
          TADPlateau_obtenirPion(p, pos)), couleur_joueur) &&
          TADPlateau_obtenirPion(p, pos).estPlace) {
94             while(TADPlateau_obtenirPion(p, pos).estPlace && !directionValide
              && !GP_positionInvalide(pos)) {
95
96
97
98
99                 pos = JEU_seDeplacerDansLaDirection(pos, dir);
100                 if (GP_positionInvalide(pos)){
101                     return 0;
102                 }
103                 if (TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(
                  TADPlateau_obtenirPion(p, pos)), couleur_joueur) &&
                  TADPlateau_obtenirPion(p, pos).estPlace){
104                     directionValide = 1;
105                 }
106
107             }
108         }
109     }
110 } while(TADPlateau_obtenirPion(p, pos).estPlace && !directionValide);
111
112 return directionValide;
113 }
114
115 void GP_miseAJourPlateau(Plateau *p, Coup coup) {
116     bool directionValide;
117     Position pos;
118
119
120
121     for(unsigned int dir=0; dir<8; dir++) {
122         pos = TADPion_obtenirPosition(TADCoup_obtenirPion(coup));
123         directionValide = GP_testerDirection(*p, pos, dir,
          TADPion_obtenirCouleur(coup.pion));
124         if(directionValide){
125             pos = JEU_seDeplacerDansLaDirection(pos, dir);
126             while (!TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(
              TADPlateau_obtenirPion(*p, pos)), TADPion_obtenirCouleur(coup.
              pion)) && TADPlateau_obtenirPion(*p, pos).estPlace){
127
128                 Pion pion = TADPlateau_obtenirPion(*p, pos);
129                 TADPion_changerCouleur(&pion);

```

```

130         TADPlateau_placerPion(p, pion);
131
132         pos = JEU_seDeplacerDansLaDirection(pos, dir);
133     }
134 }
135 }
136 }
137
138
139 int GP_max(int nb1, int nb2){
140     if (nb1>nb2)
141         return nb1;
142     else
143         return nb2;
144 }
145
146 int GP_min(int nb1, int nb2){
147     if (nb1<nb2)
148         return nb1;
149     else
150         return nb2;
151 }
152
153 unsigned int GP_nbPionsContigus(Plateau p, Pion pion){
154     int resultat;
155     bool directionValide;
156     Position pos;
157     for (unsigned int i=0; i<8;i++){
158         directionValide = GP_testerDirection(p, pos, i, TADPion_obtenirCouleur
            (pion));
159
160         while ((! TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(
            TADPlateau_obtenirPion(p, pos)), TADPion_obtenirCouleur(pion))) &&
            directionValide) {
161             pos = JEU_seDeplacerDansLaDirection(pos, i);
162             resultat =resultat +1;
163         }
164     }
165     return resultat;
166 }
167 }

```

Fonctions en C qui permettent de jouer une partie

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <stdbool.h>
4 #include "TAD_Plateau.h"
5 #include "TAD_Couleur.h"
6 #include "TAD_Coups.h"
7 #include "TAD_Coup.h"

```

```

8  #include "JEU_ObtenirCoupH.h"
9  #include "IA_ObtenirCoupsPossibles.h"
10 #include "IA_CoupValide.h"
11 #include "UNIT_GestionPlateau.h"
12 #include "JEU_Jouer.h"
13
14
15 // ici *obtenirCoup permet d'obtenir un pointeur de fonction de type
    obtenirCoup, pour utiliser obtenirCoupH ou obtenirCoupIA selon le contexte
16 void JEU_jouer(Plateau *p, Couleur couleur_joueur_courant, Coup (*obtenirCoup)(
    Plateau, Couleur, int, char, unsigned int /* la profondeur */), int ligne,
    char colonne, unsigned int profondeur){
17     Coup coup_joue;
18
19     // si le tour est jouable on agit normalement, sinon on saute le tour
20     if(GP_tourJouable(*p, couleur_joueur_courant)){
21         coup_joue = obtenirCoup(*p, couleur_joueur_courant, ligne, colonne,
            profondeur);
22         TADCoup_jouerCoup(p, coup_joue);
23         GP_miseAJourPlateau(p, coup_joue);
24     }
25     else{
26         printf("MASTER : Tour sauté !\n");
27     }
28 }

```

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include "JEU_ObtenirCoupH.h"
5  #include "TAD_Plateau.h"
6  #include "TAD_Pion.h"
7  #include "TAD_Couleur.h"
8  #include "TAD_Position.h"
9  #include "TAD_Coup.h"
10
11 Coup JEU_obtenirCoupH(Plateau p, Couleur couleur_joueur, int ligne, char
    colonne, unsigned int profondeur){
12     Coup coup;
13
14     // on crée le coup choisi par le joueur humain
15     coup = TADCoup_coup(TADPion_pion(couleur_joueur, TADPosition_position(ligne
        , colonne)));
16     return coup;
17 }

```

```

1  #include <stdbool.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include "TAD_Plateau.h"
5  #include "TAD_Position.h"

```

```

6  #include "TAD_Coup.h"
7  #include "TAD_Couleur.h"
8  #include "TAD_Pion.h"
9  #include "JEU_CalculerScore.h"
10
11
12 void JEU_calculerScore(Plateau pl, unsigned int* score_blanc, unsigned int*
    score_noir) {
13     unsigned int TAILLE = 8;
14     Couleur blanc = TADCouleur_couleur(true);
15     Couleur noir = TADCouleur_couleur(false);
16     Position position;
17     *score_blanc = 0;
18     *score_noir = 0;
19
20     for(unsigned int i=1; i<TAILLE; i++) {
21         for(unsigned int j=1; j<TAILLE; j++) {
22             position = TADPosition_position(i, j) ;
23             Pion pion = TADPlateau_obtenirPion(pl, position);
24             if (TADPion_estPlace(pion)) {
25                 if (TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(pion)
26                     , blanc))
27                     (*score_blanc)++;
28                 if (TADCouleur_couleursIdentiques(TADPion_obtenirCouleur(pion)
29                     , noir))
30                     (*score_noir)++;
31             }
32         }
33     }

```

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include "TAD_Plateau.h"
5  #include "TAD_Couleur.h"
6  #include "TAD_Coup.h"
7  #include "JEU_TesterFin.h"
8  #include "IA_ScoreDUnCoup.h"
9  #include "IA_ObtenirCoupIA.h"
10 #include "JEU_FaireUnePartie.h"
11 #include "IA_CoupValide.h"
12 #include "JEU_ObtenirCoupH.h"
13 #include "JEU_Jouer.h"
14 #include "UNIT_IHM.h"
15 #include "UNIT_GestionPlateau.h"
16 #include "UNIT_GestionPlateau.c"
17 #include "JEU_TrouverEgalite.h"
18 #include "JEU_TrouverVainqueur.h"
19

```



```

20 void JEU_faireUnePartie(Couleur couleur_choisie, unsigned int profondeur){
21     Plateau p;
22     Couleur *vainqueur;
23     bool egalite;
24
25
26     // compteur de tour
27     int nb_tour = 0;
28
29
30     // on initialise la couleur du joueur avec la couleur choisie
31     Couleur couleur_joueur_humain, couleur_joueur_courant;
32     couleur_joueur_humain = couleur_choisie;
33
34
35     // l'humain commence
36     couleur_joueur_courant = couleur_joueur_humain;
37     p = GP_initialiserPlateau();
38
39
40
41     IHM_gras("Vous commencez !\n");
42     while (!JEU_testerFin(p)){
43         nb_tour = nb_tour + 1;
44
45         IHM_separateur();
46         printf("Tour numéro : %d\n", nb_tour);
47         printf("C'est au joueur %s de jouer !\n", TADCouleur_afficherCouleur(
            couleur_joueur_courant));
48         GP_afficherPlateau(p);
49
50
51
52         // si c'est l'humain qui joue
53         int ligne;
54         char colonne;
55         char str[2];
56
57         if (TADCouleur_couleursIdentiques(couleur_joueur_courant,
            couleur_joueur_humain)){
58             IHM_gras("A vous de jouer !\n");
59             do{
60                 do{
61                     IHM_gras("Quel coup voulez-vous jouer ? \nEntrez une
                        lettre suivie d'un chiffre : ");
62                     scanf("%s", str);
63
64                     ligne = atoi((const char*)&str[1]);
65                     colonne = str[0];
66                     if (colonne > 96 && colonne < 105) colonne = colonne - 32;
67

```

```

68         ligne = ligne -1;
69         colonne = (int) colonne - 65;
70         if (ligne < 0 || ligne > 7 || colonne < 0 || colonne > 7)
71             IHM_gras("<!\> Coup non valide , réessayez !\n");
72     }while(ligne < 0 || ligne > 7 || colonne < 0 || colonne > 7);
73
74     // tant que le coup passé en entrée n'est pas valide , on
75     // continue à demander d'entrer un coup
76     if (!IA_coupValide(p,TADCoup_coup(TADPion_pion(
77         couleur_joueur_humain ,TADPosition_position(ligne ,colonne))
78         )))
79         IHM_gras("<!\> Coup non valide , réessayez !\n");
80     } while (!IA_coupValide(p,TADCoup_coup(TADPion_pion(
81         couleur_joueur_humain ,TADPosition_position(ligne ,colonne)))));
82     JEU_jouer(&p,couleur_joueur_courant ,JEU_obtenirCoupH,ligne ,colonne
83         ,3);
84 }
85 // si c'est l'IA qui joue
86 else{
87     printf("ORDINATEUR : A moi, je joue ici ! \n");
88
89     JEU_jouer(&p,couleur_joueur_courant , IA_obtenirCoupIA,0,0 ,
90         profondeur);
91 }
92 // à la fin d'un tour , on change la couleur du joueur qui joue
93 couleur_joueur_courant = TADCouleur_couleurOpposee(
94     couleur_joueur_courant);
95 }
96
97 // à la fin de la partie
98
99 IHM_gras("<##> La partie est terminée ! <##>\n");
100 egalite = JEU_trouverEgalite(&p);
101 if (!egalite){
102     vainqueur = JEU_trouverVainqueur(&p);
103     GP_afficherPlateau(p);
104     printf("<##> Le vainqueur est la couleur %d ! <##>\n",vainqueur->
105         couleur);
106 }
107 else {
108     GP_afficherPlateau(p);
109     printf("<##> Pas de vainqueur ! <##>\n");
110 }
111 }

```

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "TAD_Position.h"
4 #include "JEU_SeDeplacerDansLaDirection.h"

```

```

5
6 Position JEU_seDeplacerDansLaDirection(Position pos, unsigned int dir) {
7     int dirLigne[8] = {-1,-1,0,1,1,1,0,-1};
8     int dirColonne[8] = {0,1,1,1,0,-1,-1,-1};
9
10    Position position = TADPosition_position(TADPosition_obtenirLigne(pos)+
        dirLigne[dir], TADPosition_obtenirColonne(pos)+dirColonne[dir]);
11    return position;
12 }

```

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <stdbool.h>
4 #include "TAD_Plateau.h"
5 #include "TAD_Couleur.h"
6 #include "TAD_Coups.h"
7 #include "IA_ObtenirCoupsPossibles.h"
8 #include "JEU_TesterFin.h"
9 #include "UNIT_GestionPlateau.h"
10
11 bool JEU_testerFin(Plateau p){
12     // si le plateau est plein ou que le joueur noir ne peut pas jouer, c'est
        fini
13     if(TADPlateau_estPlein(p) || (!GP_tourJouable(p, TADCouleur_couleur(1))))
14         return 1;
15     // si le joueur blanc ne peut pas jouer non plus, c'est fini
16     else if (!GP_tourJouable(p, TADCouleur_couleur(0)))
17         return 1;
18     // sinon ce n'est pas fini
19     else
20         return 0;
21 }

```

```

1 #include <stdbool.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include "TAD_Plateau.h"
5 #include "TAD_Position.h"
6 #include "TAD_Coup.h"
7 #include "IA_CoupValide.h"
8 #include "UNIT_GestionPlateau.h"
9 #include "IA_Score.h"
10 #include "JEU_TrouverVainqueur.h"
11 #include "JEU_CalculerScore.h"
12
13
14 bool* JEU_trouverEgalite(Plateau* pl){
15     bool* egalite = (bool*)malloc(sizeof(bool));
16     unsigned int score_blanc, score_noir;
17
18     JEU_calculerScore(*pl,&score_blanc,&score_noir);

```

```

19     *egalite = (score_blanc==score_noir);
20     return egalite;
21 }

```

```

1  #include <stdbool.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include "TAD_Plateau.h"
5  #include "TAD_Position.h"
6  #include "TAD_Coup.h"
7  #include "IA_CoupValide.h"
8  #include "UNIT_GestionPlateau.h"
9  #include "IA_Score.h"
10 #include "JEU_TrouverVainqueur.h"
11 #include "JEU_CalculerScore.h"
12
13 Couleur* JEU_trouverVainqueur(Plateau* pl) {
14     unsigned int TAILLE = 8;
15     int* egalite = (int*)malloc(sizeof(int));
16     Couleur* vainqueur = (Couleur*)malloc(sizeof(Couleur)*TAILLE*TAILLE);
17     unsigned int score_blanc, score_noir;
18     JEU_calculerScore(*pl,&score_blanc,&score_noir);
19     *egalite = (score_blanc==score_noir);
20
21     if ((*egalite)){
22         *vainqueur = TADCouleur_couleur(true) ;
23     }
24     if (!(*egalite)){
25         if (score_blanc>score_noir) {
26             *vainqueur = TADCouleur_couleur(true);
27         } else {
28             *vainqueur = TADCouleur_couleur(false);
29         }
30     }
31     return vainqueur;
32 }

```

Fonction liée à l'interface Homme Machine

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "UNIT_IHM.h"
4
5  // le préfixe à mettre dans un printf pour le gras
6
7  void IHM_gras(char* texte){
8     printf("\033[1m%s\033[0m", texte);
9 }
10
11 void IHM_separateur(){

```

```

12     printf("_____\\n");
13 }
14
15 void IHM_aide() {
16     IHM_separateur();
17     IHM_gras("<#> Aide du programme othello <#>\\n");
18     printf("Les options possibles sont :\\n");
19     printf("\\t <1> othello standard blanc | noir [profondeur >2]\\n");
20     printf("\\t\\t permet de jouer contre l'ordinateur en lui donnant les blancs
        ou les noirs\\n");
21     printf("\\t <2> othello tournoi blanc | noir [profondeur >2]\\n");
22     printf("\\t\\t permet de faire jouer le programme dans un mode tournoi en
        lui donnant les blancs ou les noirs\\n");
23     IHM_separateur();
24 }

```

Fonctions/Procédure en C liées à l'IA

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include "TAD_Plateau.h"
4  #include "TAD_Couleur.h"
5  #include "IA_Score.h"
6  #include "IA_Evaluer.h"
7
8  int IA_evaluer(Plateau p, Couleur couleur_joueur){
9      // on soustrait le score du joueur adverse à celui du joueur référence
10     return IA_score(p, couleur_joueur) - IA_score(p, TADCouleur_couleurOpposee(
        couleur_joueur));
11 }

```

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include "TAD_Couleur.h"
4  #include "TAD_Plateau.h"
5  #include "TAD_Coups.h"
6  #include "IA_MinMax.h"
7  #include "IA_ObttenirCoupsPossibles.h"
8  #include "IA_ScoreDUnCoup.h"
9  #include "UNIT_GestionPlateau.h"
10
11 int IA_minMax(Plateau p, Couleur couleur_joueur_referent, Couleur
    couleur_joueur_courant, unsigned int profondeur){
12     Coups coups_possibles;
13     int score, nb_coups_possibles, resultat;
14     unsigned int i;
15
16
17     coups_possibles = IA_obtenirCoupsPossibles(p, couleur_joueur_courant);
18     nb_coups_possibles = TADCoups_obtenirNbCoups(coups_possibles);

```

```

19
20
21
22     resultat = IA_scoreDUnCoup(p, TADCoups_obtenirIEmeCoup(coups_possibles, 0),
23         couleur_joueur_referent, couleur_joueur_courant, profondeur);
24
25     for (i = 1; i < nb_coups_possibles; i++) {
26         score = IA_scoreDUnCoup(p, TADCoups_obtenirIEmeCoup(coups_possibles, i),
27             couleur_joueur_referent, couleur_joueur_courant, profondeur);
28         if (TADCouleur_couleursIdentiques(couleur_joueur_courant,
29             couleur_joueur_referent)) {
30             resultat = GP_max(resultat, score);
31         }
32         else {
33             resultat = GP_min(resultat, score);
34         }
35     }
36
37     return resultat;
38 }

```

```

1  #include <stdlib.h>
2  #include <stdbool.h>
3  #include <stdio.h>
4  #include "TAD_Coups.h"
5  #include "TAD_Plateau.h"
6  #include "TAD_Couleur.h"
7  #include "IA_ObtenirCoupsPossibles.h"
8  #include "IA_ScoreDUnCoup.h"
9  #include "IA_ObtenirCoupIA.h"
10 #include "UNIT_GestionPlateau.h"
11
12 Coup IA_obtenirCoupIA(Plateau p, Couleur couleur_joueur_courant, int ligne, char
13     colonne, unsigned int profondeur) {
14     Coup coup;
15     Coups coups_possibles;
16     int score, meilleur_score;
17     unsigned int i;
18
19     coups_possibles = IA_obtenirCoupsPossibles(p, couleur_joueur_courant);
20
21     coup = TADCoups_obtenirIEmeCoup(coups_possibles, 0);
22
23     meilleur_score = IA_scoreDUnCoup(p, coup, couleur_joueur_courant,
24         couleur_joueur_courant, profondeur);
25
26     for (i = 1; i < (TADCoups_obtenirNbCoups(coups_possibles)); i++) {
27         score = IA_scoreDUnCoup(p, TADCoups_obtenirIEmeCoup(coups_possibles, i),
28             couleur_joueur_courant, couleur_joueur_courant, profondeur);

```

```

26         if (score>meilleur_score){
27             coup = TADCoups_obtenirIEmeCoup(coups_possibles , i);
28             meilleur_score = score;
29         }
30     }
31     return coup;
32 }

```

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include "TAD_Plateau.h"
4  #include "TAD_Pion.h"
5  #include "TAD_Coups.h"
6  #include "TAD_Coup.h"
7  #include "TAD_Position.h"
8  #include "IA_CoupValide.h"
9  #include "IA_ObttenirCoupsPossibles.h"
10
11 Coups IA_obtenirCoupsPossibles(Plateau p,Couleur couleur_joueur){
12     Coups coups_possibles = TADCoups_coups();
13     Coup coup;
14
15
16     // on parcourt l'ensemble des coups du plateau, si un coup est valide, on
17     // l'ajoute à la liste
18     for(int i=0; i<8; i++){
19         for(int j=0; j<8; j++){
20             if(IA_coupValide(p, TADCoup_coup(TADPion_pion(couleur_joueur ,
21                 TADPosition_position(i , j))))) {
22                 coup = TADCoup_coup(TADPion_pion(couleur_joueur ,
23                     TADPosition_position(i , j)));
24                 TADCoups_ajouterCoup(&coups_possibles , coup);
25             }
26         }
27     }
28     return coups_possibles;
29 }

```

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <TAD_Plateau.h>
4  #include <TAD_Couleur.h>
5  #include <IA_Score.h>
6
7 int IA_score(Plateau p,Couleur couleur_joueur){
8     int score = 0;
9     int colonne , ligne;
10     Pion pion;
11     Couleur couleur;
12     Position position;
13

```

```

14 // boucle sur la largeur
15 for (colonne = 0;colonne<=7;colonne++){
16     // boucle sur la hauteur
17     for (ligne = 0;ligne<=7;ligne++){
18         position = TADPosition_position(ligne,colonne);
19         pion = TADPlateau_obtenirPion(p,position);
20         couleur = TADPion_obtenirCouleur(pion);
21         // si la couleur du pion est la couleur du joueur passée en entrée
22         // , on ajoute 1 au score
23         if (TADCouleur_couleursIdentiques(couleur,couleur_joueur) && pion.
24             estPlace){
25             score = score + 1;
26         }
27     }
28 }
29 return score;
30 }

```

```

1 #include <stdlib.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include "TAD_Plateau.h"
5 #include "TAD_Coup.h"
6 #include "TAD_Couleur.h"
7 #include "IA_Evaluer.h"
8 #include "IA_MinMax.h"
9 #include "UNIT_GestionPlateau.h"
10 #include "IA_ScoreDUnCoup.h"
11
12 int IA_scoreDUnCoup(Plateau p,Coup coup, Couleur couleur_joueur_referent,
13     Couleur couleur_joueur_courant, unsigned int profondeur){
14     Plateau copie_plateau = TADPlateau_plateau();
15     TADPlateau_copier(p,&copie_plateau);
16     // on copie le plateau pour ne pas le modifier
17
18     TADCoup_jouerCoup(&copie_plateau,coup);
19     GP_miseAJourPlateau(&copie_plateau,coup);
20
21
22     if ( TADPlateau_estPlein(copie_plateau) || profondeur <= 0){
23         int evaluation = IA_evaluer(copie_plateau,couleur_joueur_referent);
24         return evaluation;
25     }
26     else{
27         return IA_minMax(copie_plateau,couleur_joueur_referent,
28             TADCouleur_couleurOpposee(couleur_joueur_courant),profondeur-1);
29     }
30 }

```

Fonction en C qui permet de faire un Tournoi

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include <string.h>
5 #include "TAD_Couleur.h"
6 #include "JEU_TesterFin.h"
7 #include "TAD_Coups.h"
8 #include "IA_ObtenirCoupIA.h"
9 #include "JEU_ObtenirCoupH.h"
10 #include "IA_ObtenirCoupsPossibles.h"
11 #include "JEU_Jouer.h"
12 #include "UNIT_IHM.h"
13 #include "TAD_Coup.h"
14 #include "JEU_TrouverEgalite.h"
15 #include "UNIT_GestionPlateau.h"
16 #include "JEU_TrouverVainqueur.h"
17
18
19 // fonction pour lancer le couleur_joueur_courantnoi
20 void TO_tournoi(Couleur couleur_choisie, unsigned int profondeur){
21     Plateau p = GP_initialiserPlateau();
22     bool a_nous_de_jouer = !(TADCouleur_couleursIdentiques(TADCouleur_couleur
23         (1), couleur_choisie));
24     Position pos;
25     Coup coup;
26     bool coup_possible = true;
27     bool coup_possible_precedent;
28     Couleur vainqueur;
29     bool egalite;
30     Pion pion;
31     char saisie[7];
32     while(true){
33         if (a_nous_de_jouer) {
34             coup_possible_precedent = coup_possible;
35             coup_possible = GP_tourJouable(p, couleur_choisie);
36             if (coup_possible){
37
38                 coup = IA_obtenirCoupIA(p, couleur_choisie, 0, 0, profondeur);
39                 JEU_jouer(&p, couleur_choisie, IA_obtenirCoupIA, 0, 0, profondeur);
40                 printf("%c%d\n", TADPosition_obtenirColonne(
41                     TADPion_obtenirPosition(TADCoup_obtenirPion(coup))) + 'a',
42                     TADPosition_obtenirLigne(TADPion_obtenirPosition(
43                         TADCoup_obtenirPion(coup)))+1);
44             }
45             else {
46                 if (coup_possible_precedent)
47                     printf("passe\n");
48                 else{
```

```

47         egalite = JEU_trouverEgalite(&p);
48         vainqueur = *JEU_trouverVainqueur(&p);
49         if (egalite)
50             printf("nulle\n");
51         else
52             printf((TADCouleur_couleursIdentiques(vainqueur,
53                 TADCouleur_couleur(1))) ? "blanc\n" : "noir\n");
54     }
55 }
56 else{
57     scanf("%s",saisie);
58     if (strcmp(saisie, "passe\n") != 0){ //s'il ne passe pas son tour
59         pos = GP_convertirCoordonneeEnPosition(saisie);
60         pion = TADPion_pion(TADCouleur_couleurOpposee(couleur_choisie)
61             , pos);
62         coup = TADCoup_coup(pion);
63         TADCoup_jouerCoup(&p,coup);
64         GP_miseAJourPlateau(&p,coup);
65     }
66 }
67
68     fflush(stdout);
69     a_nous_de_jouer = !a_nous_de_jouer;
70
71 }
72 }

```

Tests Unitaires des TAD

```

1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <assert.h>
5  #include <stdbool.h>
6  #include "TAD_Coup.h"
7  #include "TAD_Coups.h"
8  #include "TAD_Couleur.h"
9  #include "TAD_Pion.h"
10 #include "TAD_Plateau.h"
11 #include "TAD_Position.h"
12 #include "time.h"
13 #include <CUnit/CUnit.h>
14 #include <CUnit/Basic.h>
15 #include <CUnit/Automated.h>
16 #include <CUnit/Console.h>
17
18
19
20 int init_suite_success(void) {

```

```

21     return 0;
22 }
23
24 int clean_suite_success(void) {
25     return 0;
26 }
27
28 bool egalite_pion(Pion p1, Pion p2) {
29     bool res;
30     Position pos1, pos2;
31     res = false;
32     Couleur c1;
33     Couleur c2;
34     c1 = p1.couleur;
35     c2 = p2.couleur;
36     pos1 = p1.position;
37     pos2 = p2.position;
38     if ((c1.couleur == c2.couleur) && (p1.estPlace == p2.estPlace) && (pos1.
        ligne == pos2.ligne) &&(pos1.colonne == pos2.colonne)){
39         res = true;
40     }
41     return (res);
42 }
43
44 bool egalite_position(Position pos1, Position pos2) {
45     return (pos1.ligne == pos2.ligne && pos1.colonne == pos2.colonne);
46 }
47
48 bool egalite_coup(Coup cp1, Coup cp2) {
49     return (egalite_pion(TADCoup_obtenirPion(cp1), TADCoup_obtenirPion(cp2)))
        ;}
50
51
52 void TAD_test_couleur() {
53     Couleur blanc, noir;
54     char *affichage_couleur_blanc;
55     char * affichage_couleur_noir;
56     blanc = TADCouleur_couleur(true);
57     noir = TADCouleur_couleur(false);
58     affichage_couleur_blanc =TADCouleur_afficherCouleur(blanc);
59     affichage_couleur_noir = TADCouleur_afficherCouleur(noir);
60     CU_ASSERT_TRUE(TADCouleur_couleurOpposee(blanc).couleur ==
        TADCouleur_couleur(false).couleur);
61     CU_ASSERT_FALSE((int)TADCouleur_couleursIdentiques(blanc, noir));
62     CU_ASSERT_TRUE((int)TADCouleur_couleursIdentiques(blanc, TADCouleur_couleur
        (true)));
63     CU_ASSERT_STRING_EQUAL(affichage_couleur_blanc, "blanc");
64     CU_ASSERT_STRING_EQUAL(affichage_couleur_noir, "noir");
65 }
66
67 void TAD_test_position() {

```

```

68     Position pos;
69     unsigned int x = rand() % 8;
70     unsigned int y = rand() % 8;
71
72     pos = TADPosition_position(x, y);
73     CU_ASSERT_TRUE(TADPosition_obtenirLigne(pos) == x);
74     CU_ASSERT_TRUE(TADPosition_obtenirColonne(pos) == y);
75 }
76
77 void TAD_test_pion() {
78     Pion p;
79     bool c = true;
80     Couleur blanc;
81     blanc = TADCouleur_couleur(c);
82     Position pos = TADPosition_position(rand() % 8, rand() % 8);
83     p = TADPion_pion(blanc, pos);
84     CU_ASSERT_EQUAL(TADPion_obtenirCouleur(p).couleur, blanc.couleur);
85     CU_ASSERT_EQUAL(p.estPlace, false);
86     TADPion_changerCouleur(&p);
87     CU_ASSERT_EQUAL(TADPion_obtenirCouleur(p).couleur,
88                     TADCouleur_couleurOpposee(blanc).couleur);
89     TADPion_fixerCouleur(&p, blanc);
90     CU_ASSERT_EQUAL(TADPion_obtenirCouleur(p).couleur, blanc.couleur);
91 }
92
93 void TAD_test_plateau() {
94     Plateau plateau = TADPlateau_plateau();
95     CU_ASSERT_FALSE(TADPlateau_estPlein(plateau));
96 }
97
98 void TAD_test_coup() {
99     Pion p;
100    Coup c;
101    Couleur blanc = TADCouleur_couleur(true);
102    Position pos = TADPosition_position(rand() % 8, rand() % 8);
103    p = TADPion_pion(blanc, pos);
104    Plateau pl = TADPlateau_plateau();
105    c = TADCoup_coup(p);
106    CU_ASSERT_TRUE(egalite_pion(TADCoup_obtenirPion(c), p));
107    TADCoup_jouerCoup(&pl, c);
108    CU_ASSERT_TRUE(TADPion_estPlace(TADPlateau_obtenirPion(pl,
109                                    TADPion_obtenirPosition(c.pion))));
110 }
111
112
113 void TAD_test_coups() {
114     Couleur blanc = TADCouleur_couleur(true);
115     Couleur noir = TADCouleur_couleur(false);
116     Coups coups = TADCoups_coups();

```

```

117     Pion p1 = TADPion_pion( blanc , TADPosition_position(1, 1));
118     Pion p2 = TADPion_pion( noir , TADPosition_position(1, 2));
119     Coup coup1 = TADCoup_coup(p1);
120     Coup coup2 = TADCoup_coup(p2);
121
122     CU_ASSERT_TRUE(TADCoups_obtenirNbCoups(coups) == 0);
123     CU_ASSERT_TRUE(TADCoups_estVide(coups));
124     TADCoups_ajouterCoup(&coups, coup1);
125     CU_ASSERT_FALSE(TADCoups_estVide(coups));
126     TADCoups_ajouterCoup(&coups, coup2);
127     CU_ASSERT_TRUE(TADCoups_obtenirNbCoups(coups) == 2);
128
129     CU_ASSERT_TRUE(egalite_coup(TADCoups_obtenirIEmeCoup(coups, 1), coup1));
130     CU_ASSERT_TRUE(egalite_coup(TADCoups_obtenirIEmeCoup(coups, 0), coup2));
131 }
132
133
134 int main(int argc, char** argv){
135     CU_BasicRunMode mode = CU_BRM_VERBOSE;
136     CU_pSuite pSuite = NULL;
137     srand(time(NULL));
138
139     /* initialisation du registre de tests */
140     if (CUE_SUCCESS != CU_initialize_registry()){
141         return CU_get_error();
142     }
143
144     /* ajout d'une suite de test */
145     pSuite = CU_add_suite("Test othello : TADs", init_suite_success,
146                          clean_suite_success);
147     if (NULL == pSuite) {
148         CU_cleanup_registry();
149         return CU_get_error();
150     }
151
152     /* Ajout des tests */
153     if ((NULL == CU_add_test(pSuite, "Test TAD Couleur", TAD_test_couleur))
154         || (NULL == CU_add_test(pSuite, "Test TAD Position", TAD_test_position))
155         || (NULL == CU_add_test(pSuite, "Test TAD Pion", TAD_test_pion))
156         || (NULL == CU_add_test(pSuite, "Test TAD Coup", TAD_test_coup))
157         || (NULL == CU_add_test(pSuite, "Test TAD Coups", TAD_test_coups))
158         || (NULL == CU_add_test(pSuite, "Test TAD Plateau",
159                                 TAD_test_plateau)))
160     {
161         CU_cleanup_registry();
162         return CU_get_error();
163     }
164
165     /* Lancement des tests */
166     CU_basic_set_mode(mode);
167     CU_basic_run_tests();

```

```
166     printf ( "\n" );
167     CU_basic_show_failures ( CU_get_failure_list ( ) );
168     printf ( "\n\n" );
169
170     /* Nettoyage du registre */
171     CU_cleanup_registry ( ) ;
172     return CU_get_error ( ) ;
173 }
```

Tests Unitaires

Chapitre 6

Conclusion

Après 10 semaines de travail, nous sommes parvenus à l'élaboration d'un jeu d'Othello pouvant être joué sur terminal. Celui-ci a été réalisé via la conception de TAD représentant l'ensemble des éléments nécessaires pour le fonctionnement du jeu : Couleur, Position, Pion, Plateau, Coup, et Coups. De plus, un nombre d'algorithmes représentant le fonctionnement logique du plateau, la gestion d'une partie dont la répartition des tours et le comptage des points, ainsi que la réalisation d'une IA basée sur l'algorithme MinMax ont notamment été élaborés pour mener à fin ce projet.

D'un point de vu personnel, ce projet nous aura permis :

- D'exercer la gestion de projet via la plateforme collaborative Gitlab
- Mettre en application l'ensemble des étapes du cycle en V
- Maîtriser l'élaboration et l'usage de types abstraits de données
- Mettre en oeuvre des fonctions et procédures avancées
- Comprendre et réaliser une IA basée sur l'algorithme MinMax
- Approfondir et aiguïser notre connaissance du langage C
- Concevoir un ensemble de tests associées aux algorithmes développés
- Apprendre la rédaction d'un makefile
- Rédiger de la documentation via Doxygen et un rapport rédigé en LaTeX

Le projet a été réparti au sein du groupe selon le tableau suivant :

	Yasmine	Lucas	Jihane	Laurent
Analyse	En commun : analyse du problème, conception des TAD à utiliser, réalisation d'une analyse descendante			
Conception préliminaire	Signature de certaines fonctions liées aux TAD	Signatures de l'ensemble des fonctions préfixées IA, IHM et JEU	Signatures de fonctions préfixées TAD et JEU	Implémentation des signatures des fonctions liées aux TAD
Conception détaillée	Elaboration de certaines fonctions liées aux TAD ainsi que la correction de certaines fonctions en terme d'encapsulation	Élaboration des fonctions liées à l'intelligence artificielle, à l'interface homme machine et les fonctions qui permettent de lancer une partie.	Elaboration des fonctions liées aux TAD Coups, Plateau, et fonctions de JEU.	Elaboration des fonctions générales liées à la gestion du plateau (GP) et des TAD
Développement	Implémentation de certaines fonctions liées aux TAD et correction de certaines fonctions (ajout de pointeurs ...)	Implémentation des fonctions citées précédemment, ainsi que correction des TAD pour s'adapter à cette implémentation.	Implémentation des fonctions citées et modifications apportées aux fonction préfixées TAD et IA.	Implémentation des fonctions citées ci-dessus, intercompatibilité
Développement et tests unitaires	Tests liées à la gestion du plateau	Débogage et tests d'intégration.	Tests unitaires liés aux fonctions préfixées JEU et TAD, débogage au niveau des fonctions IA.	Débogage des fonctions et structures associées aux TAD pour assurer l'implémentation des fonctions de jeu