

# INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN

EC ALGORITHMIQUE AVANCÉE ET PROGRAMMATION C

---

## Rapport de projet d'algorithmique

---

*Titre du projet :*

« Jeu d'Othello »

*Auteurs :*

Gautier DARCHEN

Romain JUDIC

Riadh KILANI

Claire LOVISA

Sandratra RASENDRASOA

3 janvier 2016



# Introduction

# Table des matières

<b>Introduction</b>	<b>2</b>
<b>I Analyse</b>	<b>4</b>
<b>1 Analyse des TAD</b>	<b>5</b>
1.1 Le TAD « Couleur » . . . . .	5
1.2 Le TAD « Pion » . . . . .	5
1.3 Le TAD « Position » . . . . .	5
1.4 Le TAD « Plateau » . . . . .	5
1.5 Le TAD « Coup » . . . . .	6
1.6 Le TAD « Coups » . . . . .	6
<b>2 Analyse descendante</b>	<b>7</b>
<b>II Conception préliminaire</b>	<b>8</b>
<b>1 Conception préliminaire des TAD</b>	<b>9</b>
1.1 Conception préliminaire du TAD « Couleur » . . . . .	9
1.2 Conception préliminaire du TAD « Pion » . . . . .	9
1.3 Conception préliminaire du TAD « Position » . . . . .	9
1.4 Conception préliminaire du TAD « Plateau » . . . . .	10
1.5 Conception préliminaire du TAD « Coup » . . . . .	10
1.6 Conception préliminaire du TAD « Coups » . . . . .	10
<b>2 Conception préliminaire des fonctions et procédures issues des analyses descendantes</b>	<b>11</b>
2.1 Conception préliminaire de l'analyse descendante de « Faire une partie » . . . . .	11
2.1.1 Types . . . . .	11
2.1.2 Type Direction . . . . .	11
2.1.3 Sous-programmes . . . . .	11
2.2 Conception préliminaire de l'analyse descendante de « obtenirCoupIA » . . . . .	12
<b>III Conception détaillée</b>	<b>13</b>
<b>1 Conception détaillée des TAD</b>	<b>14</b>
1.1 CD du type « Couleur » . . . . .	14
1.2 CD du type « Pion » . . . . .	14
1.3 CD du type « Position » . . . . .	14
1.4 CD du type « Plateau » . . . . .	14

1.5	CD du type « Coup » . . . . .	14
1.6	CD du type « Coups » . . . . .	14
<b>2</b>	<b>Conception détaillée des algorithmes compliqués de l'analyse « faireUnePartie »</b>	<b>16</b>
2.1	La procédure « faireUnePartie » . . . . .	16
2.2	La procédure « jouer » . . . . .	17
2.3	La procédure « jouerCoup » . . . . .	17
2.4	La procédure « inverserPions » . . . . .	17
2.5	La procédure « inverserPionsDir » . . . . .	18
2.6	La procédure « pionEstPresent » . . . . .	18
2.7	La procédure « pionEstPresentRecuratif » . . . . .	19
<b>3</b>	<b>Conception détaillée des algorithmes compliqués de l'analyse « obtenirCoupIA »</b>	<b>20</b>
3.1	La fonction « obtenirCoupIA » . . . . .	20
3.2	La fonction « scoreDUnCoup » . . . . .	21
3.3	La fonction « listeCoupsPossibles » . . . . .	21
3.4	La fonction « coupValide » . . . . .	21
3.5	La fonction « minMax » . . . . .	22
3.6	La fonction « evaluerPlateau » . . . . .	23
3.7	La fonction « evaluerNbCoupsPossiblesAdversaire » . . . . .	23
3.8	La fonction « evaluerNbPionsCouleur » . . . . .	23
3.9	La fonction « evaluerPositionsPionsPlateau » . . . . .	23
<b>IV</b>	<b>Développement</b>	<b>25</b>
<b>1</b>	<b>Les fichiers d'en-têtes (<i>headers</i>)</b>	<b>26</b>
1.1	Le fichier « TAD_Couleur.h » . . . . .	26
1.2	Le fichier « TAD_Coup.h » . . . . .	27
1.3	Le fichier « TAD_Coups.h » . . . . .	28
1.4	Le fichier « TAD_Pion.h » . . . . .	29
1.5	Le fichier « TAD_Plateau.h » . . . . .	30
1.6	Le fichier « TAD_Position.h » . . . . .	32
1.7	Le fichier « Affichage.h » . . . . .	33
1.8	Le fichier « FaireUnePartie.h » . . . . .	34
1.9	Le fichier « FaireUnePartie_Prive.h » . . . . .	35
1.10	Le fichier « ListeCoupsPossibles.h » . . . . .	38
1.11	Le fichier « ListeCoupsPossibles_Prive.h » . . . . .	39
1.12	Le fichier « ObtenirCoupHumain.h » . . . . .	39
1.13	Le fichier « ObtenirCoupIA.h » . . . . .	39
1.14	Le fichier « ObtenirCoupIA_Prive.h » . . . . .	40
<b>2</b>	<b>Les fichiers <i>C</i></b>	<b>42</b>
2.1	Le fichier « TAD_Couleur.c » . . . . .	42
2.2	Le fichier « TAD_Coup.c » . . . . .	42
2.3	Le fichier « TAD_Coups.c » . . . . .	43
2.4	Le fichier « TAD_Pion.c » . . . . .	43
2.5	Le fichier « TAD_Plateau.c » . . . . .	44
2.6	Le fichier « TAD_Position.c » . . . . .	45
2.7	Le fichier « Affichage.c » . . . . .	45
2.8	Le fichier « FaireUnePartie.c » . . . . .	48

2.9	Le fichier « ListesCoupsPossibles.c » . . . . .	54
2.10	Le fichier « ObtenirCoupHumain.c » . . . . .	55
2.11	Le fichier « ObtenirCoupIA.c » . . . . .	56
2.12	Le fichier « main.c » . . . . .	60
<b>3</b>	<b>Les fichiers de test</b>	<b>62</b>
3.1	Le fichier « TestFaireUnePartie.c » . . . . .	62
3.2	Le fichier « TestListeCoupsPossibles.c » . . . . .	68
3.3	Le fichier « TestTADs.c » . . . . .	77
<b>V</b>	<b>Répartition du travail</b>	<b>82</b>
<b>1</b>	<b>Analyse descendante</b>	<b>83</b>
<b>2</b>	<b>Conception préliminaire</b>	<b>84</b>
<b>3</b>	<b>Conception détaillée</b>	<b>85</b>
<b>4</b>	<b>Développement</b>	<b>86</b>
	<b>Conclusion</b>	<b>86</b>

Première partie

*Analyse*

# Chapitre 1

## Analyse des TAD

### 1.1 Le TAD « Couleur »

**Nom:** Couleur  
**Opérations:** blanc:  $\rightarrow$  Couleur  
noir:  $\rightarrow$  Couleur  
changerCouleur: Couleur  $\rightarrow$  Couleur  
**Axiomes:** - *changerCouleur(blanc())=noir()*  
- *changerCouleur(noir())=blanc()*

### 1.2 Le TAD « Pion »

**Nom:** Pion  
**Utilise:** Couleur  
**Opérations:** creerPion: Couleur  $\rightarrow$  Pion  
obtenirCouleurPion: Pion  $\rightarrow$  Couleur  
retournerPion: Pion  $\rightarrow$  Pion  
**Axiomes:** - *obtenirCouleurPion(creerPion(couleur))=couleur*  
- *obtenirCouleurPion(retournerPion(pion))=changerCouleur(obtenirCouleurPion(pion))*

### 1.3 Le TAD « Position »

**Nom:** Position  
**Utilise:** NaturelNonNul  
**Opérations:** obtenirLigne: Position  $\rightarrow$  NaturelNonNul  
obtenirColonne: Position  $\rightarrow$  NaturelNonNul  
fixerPosition: NaturelNonNul  $\times$  NaturelNonNul  $\rightarrow$  Position  
**Axiomes:** - *obtenirLigne(fixerPosition(ligne,colonne))=ligne*  
- *obtenirColonne(fixerPosition(ligne,colonne))=colonne*  
**Préconditions:** fixerPosition(ligne,colonne):  $1 \leq \text{ligne} \leq 8 \ \& \ 1 \leq \text{colonne} \leq 8$

### 1.4 Le TAD « Plateau »

**Nom:** Plateau  
**Utilise:** Booleen, Position, Pion



**Opérations:**  $\text{creerPlateau}: \rightarrow \text{Plateau}$   
 $\text{estCaseVide}: \text{Plateau} \times \text{Position} \rightarrow \mathbf{Booleen}$   
 $\text{viderCase}: \text{Plateau} \times \text{Position} \rightarrow \text{Plateau}$   
 $\text{poserPion}: \text{Plateau} \times \text{Position} \times \text{Pion} \rightarrow \text{Plateau}$   
 $\text{obtenirPion}: \text{Plateau} \times \text{Position} \rightarrow \text{Pion}$   
 $\text{inverserPion}: \text{Plateau} \times \text{Position} \rightarrow \text{Plateau}$

**Axiomes:** -  $\text{estCaseVide}(\text{creerPlateau}(), \text{position}) = \text{VRAI}$   
-  $\text{estCaseVide}(\text{viderCase}(\text{plateau}, \text{position}), \text{position}) = \text{VRAI}$   
-  $\text{estCaseVide}(\text{poserPion}(\text{plateau}, \text{position}, \text{pion}), \text{position}) = \text{FAUX}$   
-  $\text{obtenirPion}(\text{poserPion}(\text{plateau}, \text{position}, \text{pion}), \text{position}) = \text{pion}$   
-  $\text{inverserPion}(\text{inverserPion}(\text{plateau}, \text{position}), \text{position}) = \text{plateau}$

**Préconditions:**  $\text{viderCase}(\text{plateau}, \text{position}): \text{non}(\text{estCaseVide}(\text{plateau}, \text{position}))$   
 $\text{poserPion}(\text{plateau}, \text{position}): \text{estCaseVide}(\text{plateau}, \text{position})$   
 $\text{obtenirPion}(\text{plateau}, \text{position}): \text{non}(\text{estCaseVide}(\text{plateau}, \text{position}))$   
 $\text{inverserPion}(\text{plateau}, \text{position}): \text{non}(\text{estCaseVide}(\text{plateau}, \text{position}))$

## 1.5 Le TAD « Coup »

**Nom:** Coup  
**Utilise:** Position, Pion  
**Opérations:**  $\text{creerCoup}: \text{Position} \times \text{Pion} \rightarrow \text{Coup}$   
 $\text{obtenirPositionCoup}: \text{Coup} \rightarrow \text{Position}$   
 $\text{obtenirPionCoup}: \text{Coup} \rightarrow \text{Pion}$

**Axiomes:** -  $\text{obtenirPositionCoup}(\text{creerCoup}(\text{pos}, \text{pion})) = \text{pos}$   
-  $\text{obtenirPionCoup}(\text{creerCoup}(\text{pos}, \text{pion})) = \text{pion}$

## 1.6 Le TAD « Coups »

**Nom:** Coups  
**Utilise:** Naturel, NaturelNonNul, Coup  
**Opérations:**  $\text{creerCoups}: \rightarrow \text{Coups}$   
 $\text{ajouterCoups}: \text{Coups} \times \text{Coup} \rightarrow \text{Coups}$   
 $\text{nbCoups}: \text{Coups} \rightarrow \mathbf{Naturel}$   
 $\text{iemeCoup}: \text{Coups} \times \mathbf{NaturelNonNul} \rightarrow \text{Coup}$

**Axiomes:** -  $\text{iemeCoup}(\text{ajouterCoups}(\text{cps}, \text{cp}), \text{nbCoups}(\text{cps})) = \text{cp}$   
-  $\text{nbCoups}(\text{creerCoups}()) = 0$   
-  $\text{nbCoups}(\text{ajouterCoups}(\text{cps}, \text{cp})) = \text{nbCoups}(\text{cps}) + 1$

**Préconditions:**  $\text{iemeCoup}(\text{cps}, i): i \leq \text{nbCoups}(\text{cps})$

## Chapitre 2

# Analyse descendante

On insérera ici les images des analyses descendantes (une fois qu'elles seront finies et qu'on n'y touchera plus).

Deuxième partie

Conception préliminaire

# Chapitre 1

## Conception préliminaire des TAD

Nous avons mis en place un code d'identification à l'aide de préfixes pour chaque TAD de la manière suivante :

- Couleur : « CL\_ »
- Pion : « PI\_ »
- Position : « POS\_ »
- Plateau : « PL\_ »
- Coup : « CP\_ »
- Coups : « CPS\_ »

### 1.1 Conception préliminaire du TAD « Couleur »

- **fonction** CL\_blanc () : Couleur
- **fonction** CL\_noir () : Couleur
- **fonction** CL\_changerCouleur (couleur : Couleur) : Couleur

Pour la conception détaillée, nous avons ajouté la fonction de comparaison de deux « Couleur » :

- **fonction** CL\_sontEgales (couleur1, couleur2 : Couleur) : **Booleen**

### 1.2 Conception préliminaire du TAD « Pion »

- **fonction** PI\_creerPion (couleur : Couleur) : Pion
- **fonction** PI\_obtenirCouleurPion (pion : Pion) : Couleur
- **procédure** PI\_retournerPion (**E/S** pion : Pion)

Pour la conception détaillée, nous avons ajouté la fonction de comparaison de deux « Pion » :

- **fonction** PI\_sontEgaux (pion1, pion2 : Pion) : **Booleen**

### 1.3 Conception préliminaire du TAD « Position »

- **fonction** POS\_obtenirLigne (position : Position) : **NaturelNonNul**
- **fonction** POS\_obtenirColonne (position : Position) : **NaturelNonNul**
- **procédure** POS\_fixerPosition (**E** ligne, colonne : **NaturelNonNul**, **S** position : Position)  
  |précondition(s)  $1 \leq \text{ligne} \leq 8 \ \& \ 1 \leq \text{colonne} \leq 8$

Pour la conception détaillée, nous avons ajouté la fonction de comparaison de deux « Position » :

- **fonction** POS\_sontEgales (position1, position2 : Position) : **Booleen**

## 1.4 Conception préliminaire du TAD « Plateau »

- **fonction** PL\_creerPlateau () : Plateau
- **fonction** PL\_estCaseVide (plateau : Plateau, position : Position) : Couleur
- **procédure** PL\_viderCase (**E/S** plateau : Plateau, **E** position : Position)  
  |précondition(s) non(estCaseVide(plateau,position))
- **procédure** PL\_poserPion (**E/S** plateau : Plateau, **E** position : Position, pion : Pion)  
  |précondition(s) estCaseVide(plateau,position)
- **fonction** PL\_obtenirPion (plateau : Plateau, position : Position) : Pion  
  |précondition(s) non(estCaseVide(plateau,position))
- **procédure** PL\_inverserPion (**E/S** plateau : Plateau, **E** position : Position)  
  |précondition(s) non(estCaseVide(plateau,position))

## 1.5 Conception préliminaire du TAD « Coup »

- **fonction** CP\_creerCoup (position : Position, pion : Pion) : Coup
- **fonction** CP\_obtenirPositionCoup (coup : Coup) : Position
- **fonction** CP\_obtenirPionCoup (coup : Coup) : Pion

Pour la conception détaillée, nous avons ajouté la fonction de comparaison de deux « Coup » :

- **fonction** CP\_sontEgaux (coup1, coup2 : Coup) : **Booleen**

## 1.6 Conception préliminaire du TAD « Coups »

- **fonction** CPS\_creerCoups () : Coups
- **procédure** CPS\_ajouterCoups (**E/S** coups : Coups, **E** coup : Coup)
- **fonction** CPS\_nbCoups (coups : Coups) : Naturel
- **fonction** CPS\_iemeCoup (coups : Coups, i : **NaturelNonNul**) : Coup  
  |précondition(s)  $i \leq \text{nbCoups}(\text{coups})$

## Chapitre 2

# Conception préliminaire des fonctions et procédures issues des analyses descendantes

### 2.1 Conception préliminaire de l'analyse descendante de « Faire une partie »

#### 2.1.1 Types

- **Type** getCoup = **fonction**(plateau : Plateau, couleur : Couleur) : Coup
- **Type** afficherPlateau = **procédure**(**E** plateau : Plateau, coup : Coup, aPuJoueur, estPartieFinie : **Booleen**)

#### 2.1.2 Type Direction

Pour nous aider dans l'écriture de fonctions de FaireUnePartie et d'ObtenirCoupIA, nous avons décidé de nous aider d'un type énuméré Direction, ainsi que de fonctions encapsulantes.

- **Type** Direction = {GAUCHE,DROITE,HAUT,BAS,DIAGGH,DIAGGB,DIAGDH,DIAGDB}
- **fonction** DIR\_positionSelonDirection (posInit : Position, dirDeplacement : Direction) : Position
- **fonction** DIR\_inverserDirection (dirInit : Direction) : Direction
- **fonction** DIR\_deplacementValide (pos : Position, dirDeplacement : Direction) : **Entier**

#### 2.1.3 Sous-programmes

- **procédure** faireUnePartie (**E** afficher : afficherPlateau, coupJoueur1, coupJoueur2 : getCoup, couleurJoueur1 : Couleur, **S** vainqueur : Couleur, estMatchNul : **Booleen**)
- **procédure** initialiserPlateau (**E/S** plateau : Plateau)
- **procédure** jouer (**E/S** plateau : Plateau, couleurJoueur : Couleur, **E** coupJoueur : getCoup, **S** aPuJouer : **Booleen**, coupJoueur : Coup)
- **procédure** finPartie (**E** plateau : Plateau, aPuJouerJoueur1,aPuJouerJoueur2 : **Booleen**, **S** nbPionsNoirs, nbPionsBlancs : **Naturel**, estFinie : **Booleen**)
- **fonction** plateauRempli (plateau : Plateau) : **Booleen**
- **procédure** nbPions (**E** plateau : Plateau, **S** nbPionsBlancs, nbPionsNoirs : **Naturel**)
- **procédure** jouerCoup (**E** coup : Coup, **E/S** plateau : Plateau)
- **procédure** inverserPions (**E** pos : Position, pionJoueur : Pion, **E/S** plateau : Plateau)

- **procédure** inverserPionsDir (**E/S** plateau : Plateau, **E** posInitiale, posCourante : Position, dirInversion : Direction)
- **procédure** pionEstPresent (**E** pionJoueur : Pion, dirATester : Direction, **E/S** pos : Position, plateau : Plateau, **S** pionPresent : **Booleen**)
- **procédure** pionEstPresentRecuratif (**E** pionJoueur : Pion, dirATester : Direction, **E/S** pos : Position, plateau : Plateau, **S** pionPresent : **Booleen**)

## 2.2 Conception préliminaire de l'analyse descendante de « obtenirCoupIA »

- **fonction** obtenirCoupIA (plateau : Plateau, couleur : Couleur) : Coup
- **fonction** profondeur () : **NaturelNonNul**
- **fonction** listeCoupsPossibles (plateau : Plateau, couleur : Couleur) : Coups
- **fonction** coupValide (plateau : Plateau, coup : Coup) : **Booleen**
- **procédure** copierPlateau (**E** plateauACopier : Plateau, **S** plateauCopie : Plateau)
- **fonction** minMax (plateau : Plateau, couleurRef, couleurCourante : Couleur, profondeurCourante : **Naturel**, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**
- **fonction** scoreDUnCoup (plateau : Plateau, coup : Coup, couleurRef, couleurCourante : Couleur, profondeurCourante : **Naturel**, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**
- **fonction** score (plateau : Plateau, couleur : Couleur, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**
- **fonction** evaluerPlateau (plateau : Plateau, couleur : Couleur, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**
- **fonction** evaluerNbCoupsPossiblesAdversaire (plateau : Plateau, couleur : Couleur) : **Entier**
- **fonction** evaluerNbPionsCouleur (plateau : Plateau, couleur : Couleur) : **Entier**
- **fonction** evaluerPositionsPionsPlateau (plateau : Plateau, couleur : Couleur, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**
- **fonction** initialiserGrilleScore () : grilleScore : **Tableau**[1..8][1..8] **de Entier**

Troisième partie

Conception détaillée



# Chapitre 1

## Conception détaillée des TAD

### 1.1 CD du type « Couleur »

— **Type** Couleur = {blanc, noir}

### 1.2 CD du type « Pion »

— **Type** Pion = **Structure**  
couleur : Couleur  
**finstructure**

### 1.3 CD du type « Position »

— **Type** Position = **Structure**  
ligne : **Naturel**  
colonne : **Naturel**  
**finstructure**

### 1.4 CD du type « Plateau »

— **Type** Position = **Structure**  
pions : **Tableau**[1..8][1..8] de Pion  
presencePions : **Tableau**[1..8][1..8] de Booleen  
**finstructure**

### 1.5 CD du type « Coup »

— **Type** Coup = **Structure**  
position : Position  
pion : Pion  
**finstructure**

### 1.6 CD du type « Coups »

— **Type** Coups = **Structure**  
tabCoups : **Tableau**[1..60] deCoup

nbCps : **Naturel**  
**finstructure**

## Chapitre 2

# Conception détaillée des algorithmes compliqués de l'analyse « faireUnePartie »

### 2.1 La procédure « faireUnePartie »

**procédure** faireUnePartie (**E** afficher : afficherPlateau, obtenirCoupJoueur1, obtenirCoupJoueur2 : getCoup, couleurJoueur1 : Couleur, **S** vainqueur : Couleur, estMatchNul : **Booleen**)

**Déclaration** plateau : Plateau  
aPuJouerJoueur1, aPuJouerJoueur2, estFinie : **Booleen**  
couleurJoueur2 : Couleur  
nbPionsBlancs, nbPionsNoirs : **Naturel**  
positionInitialisation : Position  
coupJoueur1, coupJoueur2 : Coup

**debut**

```
aPuJouerJoueur1 ← VRAI
aPuJouerJoueur2 ← VRAI
couleurJoueur2 ← CL_changerCouleur(couleurJoueur1)
estFinie ← FAUX
nbPionsBlancs ← 2
nbPionsNoirs ← 2
plateau ← initialiserPlateau()
PL_initialiserPlateau(plateau)
POS_fixerPosition(4,4,positionInitialisation)
coupJoueur1 ← CP_creerCoup(positionInitialisation,PI_creerPion(CL_blanc()))
afficher(plateau,coupJoueur1,aPuJouerJoueur1,estFinie)
tant que non(estFinie) faire
    jouer(plateau, couleurJoueur1, obtenirCoupJoueur1, aPuJouerJoueur1, coupJoueur1)
    afficher(plateau,coupJoueur1,aPuJouerJoueur1,estFinie)
    finPartie(aPuJouerJoueur1, aPuJouerJoueur2, plateau, estFinie, nbPionsBlancs, nbPionsNoirs)
    jouer(plateau, couleurJoueur2, obtenirCoupJoueur1, aPuJouerJoueur2, coupJoueur2)
    afficher(plateau,coupJoueur2,aPuJouerJoueur2,estFinie)
    finPartie(plateau, aPuJouerJoueur1, aPuJouerJoueur2, nbPionsBlancs, nbPionsNoirs, estFinie)
fintantque
afficher(plateau,coupJoueur2,aPuJouerJoueur2,estFinie)
si nbPionsBlancs = nbPionsNoirs alors
```

```

    vainqueur ← CL_blanc()
    estMatchNul ← VRAI
sinon
    estMatchNul ← FAUX
    si nbPionsBlancs > nbPionsNoirs alors
        vainqueur ← CL_blanc()
    sinon
        vainqueur ← CL_noir()
    finsi
finsi
fin

```

## 2.2 La procédure « jouer »

**procédure** jouer (**E/S** plateau : Plateau, couleurJoueur : Couleur, **E** obtenirCoupJoueur : getCoup, **S** aPuJouer : **Booleen**, coupJoueur : Coup)

**Déclaration** i : **Naturel**  
coups : Coups  
res : **Entier**

```

debut
    coupJoueur ← obtenirCoupJoueur(plateau,couleurJoueur)
    coups ← listeCoupsPossibles(plateau, couleurJoueur)
    si CPS_nbCoups(coups) > 0 alors
        pour i ← 1 à CPS_nbCoups(coups) faire
            si CP_sontEgaux(CPS_iemeCoup(coups,i),coupJoueur) alors
                jouerCoup(coupJoueur,plateau)
                res ← VRAI
            finsi
        finpour
    finsi
    aPuJouer ← res
fin

```

## 2.3 La procédure « jouerCoup »

**procédure** jouerCoup (**E** coup : Coup, **E/S** plateau : Plateau)

**Déclaration** pos : Position  
pionJoueur : Pion

```

debut
    PL_poserPion(plateau, CP_obtenirPositionCoup(coup), CP_obtenirPionCoup(coup))
    pos ← CP_obtenirPositionCoup(coup)
    pionJoueur ← CP_obtenirPionCoup(coup)
    inverserPions(pos, pionJoueur, plateau)
fin

```

## 2.4 La procédure « inverserPions »

**procédure** inverserPions (**E** pos : Position, pionJoueur : Pion, **E/S** plateau : Plateau)

```

Déclaration  posTmp : Position
                dir : Direction
                pionPresent : Booleen

debut
  pour dir ← GAUCHE à DIAGDB faire
    posTmp ← pos
    pionEstPresent(pionJoueur, dir, posTmp, plateau, pionPresent)
    si pionPresent alors
      inverserPionsDir(plateau, pos, DIR_positionSelonDirection(posTmp, DIR_inverserDirection(dir)),
        DIR_inverserDirection(dir))
    finsi
  finpour
fin

```

## 2.5 La procédure « inverserPionsDir »

**procédure** inverserPionsDir (**E/S** plateau : Plateau, **E** posInitiale, posCourante : Position, dirInversion : Direction)

```

Déclaration  inew, jnew : NaturelNonNul
                posSuivante : Position

debut
  positionSuivante ← posCourante
  inew ← POS_obtenirLigne(DIR_positionSelonDirection(posSuivante, dirInversion))
  jnew ← POS_obtenirColonne(DIR_positionSelonDirection(posSuivante, dirInversion))
  si non(POS_sontEgales(posInitiale, posCourante)) ET DIR_deplacementValide(posCourante, dirInversion)) alors
    PL_inverserPion(plateau, posCourante)
    POS_fixerPosition(inew, jnew, posSuivante)
    inverserPionsDir(plateau, posInitiale, posSuivante, dirInversion)
  finsi
fin

```

## 2.6 La procédure « pionEstPresent »

**procédure** pionEstPresent (**E** pionJoueur : Pion, dirATester : Direction, **E/S** pos : Position, plateau : Plateau, **S** pionPresent : **Booleen**)

```

Déclaration  couleurAdversaire : Couleur

debut
  couleurAdversaire ← CL_changerCouleur(PI_obtenirCouleur(pionJoueur))
  si non(DIR_deplacementValide(pos, dirATester)) alors
    pionPresent ← FAUX
  sinon
    pos ← DIR_positionSelonDirection(pos, dirATester)
    si CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau, pos)), couleurAdversaire) ET
      (non(PL_estCaseVide(plateau, pos))) alors
      pos ← DIR_positionSelonDirection(pos, dirATester)
      pionEstPresentRecursif(pionJoueur, dirATester, pos, plateau, pionPresent)
    sinon

```

```

        pionPresent ← FAUX
    fin
    fin
    fin
    fin

```

## 2.7 La procédure « pionEstPresentRecuratif »

**procédure** pionEstPresentRecuratif (**E** pionJoueur : Pion, dirATester : Direction, **E/S** pos : Position, plateau : Plateau, **S** pionPresent : **Booleen**)

**Déclaration** couleurJoueur : Couleur

**debut**

couleurJoueur ← PI\_obtenirCouleurPion(pionJoueur)

**si** estCaseVide(plateau, pos) **alors**

pionPresent ← FAUX

**sinon**

**si** CL\_sontEgales(PI\_obtenirCouleur(PL\_obtenirPion(plateau,pos)),couleurJoueur) **alors**

pionPresent ← VRAI

**sinon**

**si** non(DIR\_deplacementValide(pos,dirATester)) **alors**

pionPresent ← FAUX

**sinon**

pos ← DIR\_positionSelonDirection(pos,dirATester)

pionEstPresentRecuratif(pionJoueur, dirATester, pos, plateau, pionPresent)

**finsi**

**finsi**

**finsi**

**fin**

## Chapitre 3

# Conception détaillée des algorithmes compliqués de l'analyse « obtenirCoupIA »

### 3.1 La fonction « obtenirCoupIA »

**fonction** obtenirCoupIA (plateau : Plateau, couleur : Couleur) : Coup

**Déclaration** i, profondeurMinMax : **Naturel**  
coupsPossibles : Coups  
scoreCourant, meilleurScore : **Entier**  
coupCourant, meilleurCoup : Coup  
grilleScore : **Tableau**[1..8][1..8] **de Entier**

**debut**

```
profondeurMinMax ← profondeur()
coupsPossibles ← listeCoupsPossibles(plateau, couleur)
grilleScore ← initialiserGrilleScore()
si CPS_nbCoups(coupsPossibles) > 0 alors
  meilleurCoup ← CPS_iemeCoup(coupsPossibles, 1)
  meilleurScore ← scoreDUnCoup(plateau, meilleurCoup, couleur, couleur, profondeurMinMax, grilleScore)
  pour i ← 2 à CPS_nbCoups(coupsPossibles) faire
    coupCourant ← CPS_iemeCoup(coupsPossibles, i)
    scoreCourant ← scoreDUnCoup(plateau, coupCourant, couleur, couleur, profondeurMinMax, grilleScore)
    si (scoreCourant > meilleurScore) ET coupValide(plateau, coupCourant) alors
      meilleurCoup ← coupCourant
      meilleurScore ← scoreCourant
  finsi
finpour
finsi
retourner meilleurCoup
```

**fin**

### 3.2 La fonction « scoreDUnCoup »

**fonction** scoreDUnCoup (plateau : Plateau, coup : Coup, couleurRef, couleurCourante : Couleur, profondeurCourante : **Naturel**, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**

**Déclaration** plateauTest : Plateau

**debut**

copierPlateau(plateau, plateauTest)

jouerCoup(coup, plateauTest)

**si** plateauRempli(plateauTest) **OU** (profondeurCourante = 0) **alors**

**retourner** score(plateauTest, couleurRef, grilleScore)

**sinon**

**retourner** minMax(plateauTest, couleurRef, CL\_changerCouleur(couleurCourante), profondeurCourante – 1, grilleScore)

**finsi**

**fin**

### 3.3 La fonction « listeCoupsPossibles »

**fonction** listeCoupsPossibles (plateau : Plateau, couleur : Couleur) : Coups

**Déclaration** coupsPossibles : Coups

    positionTest : Position

    coupTest : Coup

    pionJoueur : Pion

    i,j : **NaturelNonNul**

    nbPionsBlancs, nbPionsNoirs : **Naturel**

**debut**

k ← 0

CPS\_creerCoups(coupsPossibles)

pionJoueur ← PI\_creerPion(couleur)

**pour** i ← 1 à 8 **faire**

**pour** j ← 1 à 8 **faire**

        POS\_fixerPosition(i,j,positionTest)

**si** PL\_estCaseVide(plateau,positionTest) **alors**

            coupTest = CP\_creerCoup(positionTest,pionJoueur)

**si** coupValide(plateau,coupTest) **alors**

                CPS\_ajouterCoups(coupsPossibles,coupTest)

**finsi**

**finsi**

**finpour**

**finpour**

**retourner** coupsPossibles

**fin**

### 3.4 La fonction « coupValide »

**fonction** coupValide (plateau : Plateau, coup : Coup) : **Booleen**

**Déclaration** pos,posTmp : Position

    pionJoueur : Pion



```

        pionPresent : Booleen
        dir : Direction

debut
    pionPresent ← FAUX
    pos ← CP_obtenirPositionCoup(coup)
    pionJoueur ← CP_obtenirPionCoup(coup)
    dir ← GAUCHE
    tant que non(pionPresent) ET (dir ≤ DIAGDB) faire
        posTmp ← pos
        si DIR_deplacementValide(posTmp, dir) ET PL_estCaseVide(plateau, pos) alors
            pionEstPresent(pionJoueur, dir, posTmp, plateau, pionPresent)
        finsi
        dir ← dir + 1
    fintantque
    retourner pionPresent
fin
    
```

### 3.5 La fonction « minMax »

**fonction** minMax (plateau : Plateau, couleurRef, couleurCourante : Couleur, profondeurCourante : **Naturel**, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**

**Déclaration** coupsPossibles : Coups  
 resultat, score : **Entier**  
 i : **Naturel**

```

debut
    coupsPossibles ← listeCoupsPossibles(plateau, couleurCourante)
    si CPS_nbCoups(coupsPossibles) > 0 alors
        resultat ← scoreDUnCoup(plateau, CPS_iemeCoup(coupsPossibles, 1), couleurRef, couleurCourante,
            profondeurCourante, grilleScore)
        pour i ← 2 à CPS_nbCoups(coupsPossibles) faire
            score ← scoreDUnCoup(plateau, CPS_iemeCoup(coupsPossibles, i), couleurRef, couleurCourante,
                profondeurCourante, grilleScore)
            si CL_sontEgales(couleurCourante, couleurRef) alors
                resultat ← max(resultat, score)
            sinon
                resultat ← min(resultat, score)
            finsi
        finpour
    sinon
        si CL_sontEgales(couleurCourante, couleurRef) alors
            resultat ← INFINI
        sinon
            resultat ← - INFINI
        finsi
    finsi
    retourner resultat
fin
    
```

*Remarque : On utilise ici une constante « INFINI », qui représentera un score supérieur à tout autre score, c'est-à-dire un coup gagnant.*

### 3.6 La fonction « evaluerPlateau »

**fonction** evaluerPlateau (plateau : Plateau, couleur : Couleur, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**

**Déclaration** evaluer1, evaluer2, evaluer3, res : **Entier**

**debut**

evaluer1  $\leftarrow$  evaluerNbCoupsPossiblesAdversaire(plateau,couleur)  
 evaluer2  $\leftarrow$  evaluerNbPionsCouleur(plateau,couleur)  
 evaluer3  $\leftarrow$  evaluerPositionsPionsPlateau(plateau,couleur, grilleScore)  
 res  $\leftarrow$  evaluer1 + evaluer2 + evaluer3  
**retourner** res

**fin**

### 3.7 La fonction « evaluerNbCoupsPossiblesAdversaire »

**fonction** evaluerNbCoupsPossiblesAdversaire (plateau : Plateau, couleur : Couleur) : **Entier**

**Déclaration** nbCoupsAdversaire, res : **Entier**  
 coupsAdversaire : Coups  
 couleurAdversaire : Couleur

**debut**

couleurAdversaire  $\leftarrow$  CL\_changerCouleur(couleur)  
 coupsAdversaire  $\leftarrow$  listeCoupsPossibles(plateau, couleurAdversaire)  
 nbCoupsAdversaire  $\leftarrow$  nbCoups(coupsAdversaire)  
 res  $\leftarrow$  60−10×nbCoupsAdversaire  
**retourner** res

**fin**

### 3.8 La fonction « evaluerNbPionsCouleur »

**fonction** evaluerNbPionsCouleur (plateau : Plateau, couleur : Couleur) : **Entier**

**Déclaration** res : **Entier**  
 nbPionsNoirs, nbPionsBlancs : **Naturel**

**debut**

nbPions(plateau,nbPionsNoirs,nbPionsBlancs)  
**si** CL\_sontEgales(couleur,CL\_noir()) **alors**  
 res  $\leftarrow$  nbPionsNoirs−nbPionsBlancs  
**sinon**  
 res  $\leftarrow$  nbPionsBlancs−nbPionsNoirs  
**finsi**  
**retourner** res

**fin**

### 3.9 La fonction « evaluerPositionsPionsPlateau »

**fonction** evaluerPositionsPionsPlateau (plateau : Plateau, couleur : Couleur, grilleScore : **Tableau**[1..8][1..8] **de Entier**) : **Entier**

**Déclaration** res, resJoueur, resAdversaire : **Entier**  
 i, j, x, y : **Naturel**  
 pos : Position

**debut**

resJoueur  $\leftarrow$  0

resAdversaire  $\leftarrow$  0

**pour** i  $\leftarrow$  1 à 8 **faire**

**pour** j  $\leftarrow$  1 à 8 **faire**

        POS\_fixerPosition(i-1,j-1,pos)

**si** non(PL\_estCaseVide(plateau, pos)) ET CL\_sontEgales(PI\_obtenirCouleur(obtenirPion(pos)), couleur) **alors**

            resJoueur  $\leftarrow$  resJoueur+grilleScore[i-1][j-1]

**sinon**

**si** non(PL\_estCaseVide(plateau,pos)) **alors**

                resAdversaire  $\leftarrow$  resAdversaire+grilleScore[i-1][j-1]

**finsi**

**finsi**

**finpour**

**finpour**

retourner res

**fin**

Quatrième partie

Développement

# Chapitre 1

## Les fichiers d'en-têtes (*headers*)

### 1.1 Le fichier « TAD\_Couleur.h »

```
/**
 * \file TAD_Couleur.h
 * \brief Implantation du TAD Couleur
 * \author Groupe 1.5
 * \version 1.0
 * \date 02/12/15
 *
 */

#ifndef __TAD_COULEUR__
#define __TAD_COULEUR__

/**
 * \enum Couleur
 * \brief Le type Couleur représente les deux couleurs possibles
 *
 */
typedef enum{BLANC,NOIR} Couleur;

/**
 * \fn Couleur CL_blanc()
 * \brief Fonction qui retourne la couleur 'blanc'
 *
 * \return Couleur
 */
Couleur CL_blanc();

/**
 * \fn Couleur CL_noir()
 * \brief Fonction qui retourne la couleur 'noir'
 *
 * \return Couleur
 */
Couleur CL_noir();

/**
 * \fn Couleur CL_changerCouleur(couleur : Couleur)
 * \brief Fonction qui retourne l'autre couleur que celle passée en paramètre d'
    entrée
 *
 * \param Couleur couleur : Couleur, la couleur à changer
 * \return Couleur
 */
```

```

*/
Couleur CL_changerCouleur(Couleur couleur);

/**
 * \fn CL_sontEgales
 * \brief Fonction testant l'égalité de deux couleurs
 *
 * \param Couleur couleur1
 * \param Couleur couleur2
 * \return int (Booleen)
 */
int CL_sontEgales(Couleur couleur1, Couleur couleur2);

#endif

```

## 1.2 Le fichier « TAD\_Coup.h »

```

/**
 * \file TAD_Coup.h
 * \brief Implantation du TAD Coup
 * \author Groupe 1.5
 * \version 1.0
 * \date 02/12/15
 *
 */

#ifndef __TAD_COUP__
#define __TAD_COUP__
#include "TAD_Position.h"
#include "TAD_Pion.h"

/**
 * \struct Coup
 * \brief Le type Coup permet de représenter le coup d'un joueur, en regroupant une
 *        position (sur le plateau) et un pion
 *
 */
typedef struct {
    Position position; /**< la largeur de la grille */
    Pion pion; /**< la hauteur de la grille */
} Coup;

/**
 * \fn Coup CP_creerCoup(Position position, Pion pion)
 * \brief Fonction qui retourne un coup à partir d'une position et d'un pion
 *
 * \param Position position : la position à affecter au Coup
 * \param Pion pion : le Pion à affecter au Coup
 * \return Coup
 */
Coup CP_creerCoup(Position position, Pion pion);

/**
 * \fn Position CP_obtenirPositionCoup(Coup coup)
 * \brief Fonction qui retourne la position d'un coup
 *
 * \param Coup coup : le coup dont on veut la position

```

```

    * \return Coup
    */
Position CP_obtenirPositionCoup(Coup coup);

/**
 * \fn Position CP_obtenirPionCoup(Coup coup)
 * \brief Fonction qui retourne le pion d'un coup
 *
 * \param Coup coup : le coup dont on veut le pion
 * \return Coup
 */
Pion CP_obtenirPionCoup(Coup coup);

/**
 * \fn CP_sontEgaux
 * \brief Fonction testant l'égalité de deux coups (type Coup)
 *
 * \param Coup coup1
 * \param Coup coup2
 * \return int (Booleen)
 */
int CP_sontEgaux(Coup coup1, Coup coup2);

#endif

```

### 1.3 Le fichier « TAD\_Coups.h »

```

/**
 * \file TAD_Coups.h
 * \brief Implantation du TAD Coups
 * \author Groupe 1.5
 * \version 1.0
 * \date 02/12/15
 *
 */

#define MAX_COUPS 60

#ifndef __TAD_COUPS__
#define __TAD_COUPS__
#include "TAD_Coup.h"
#include "TAD_Couleur.h"

/**
 * \struct Coups
 * \brief Le type Coups permet de représenter un tableau de Coup et le nombre de Coup
 *        possibles
 *
 */

typedef struct {
    Coup tabCoups[MAX_COUPS];
    unsigned int nbCps;
} Coups;

/**
 * \fn Coups CPS_creerCoups()

```

```

* \brief Fonction qui retourne un Coups (tableau de Coup) vide
*
* \return Coups
*/
void CPS_creerCoups(Coups* coups);

/**
* \fn void CPSajouterCoups(Coups* coups, Coup coup)
* \brief Fonction ajoute le Coup coup à la variable coups
*
* \param Coups* coups : un tableau de Coups
* \param Coup coup : le Coup à ajouter à coups
*/
void CPSajouterCoups(Coups* coups, Coup coup);

/**
* \fn unsigned int CPS_nbCoups(Coups coups)
* \brief Fonction qui renvoie le nombre de Coups d'une variable de type Coups
*
* \param Coups coups : la variable dont on veut compter le nombre de Coups
* \return unsigned int : le nombre de Coups
*/
unsigned int CPS_nbCoups(Coups coups);

/**
* \fn Coup CPS_iemeCoup(Coups coups, unsigned int i)
* \brief Fonction qui ieme Coup de la variable coups
*
* \param Coups coups : la variable dont on veut obtenir le ieme Coup
* \param unsigned int i : indice du Coup à obtenir
* \return Coup : le nombre de Coups
*/
Coup CPS_iemeCoup(Coups coups, unsigned int i);

#endif

```

## 1.4 Le fichier « TAD\_Pion.h »

```

/**
* \file TAD_PION.h
* \brief Implantation du TAD Pion pour le jeu Othello
* \author Groupe 1.5
* \version 1.0
* \date 2/12/2015
*
*/

#ifndef __TAD_PION__
#define __TAD_PION__
#include "TAD_Couleur.h"

/**
* \struct Pion
* \brief Le type Pion permet de représenter un pion
*
*/

typedef struct {
    Couleur couleur; /**< la couleur du pion */

```



```

} Pion;

/**
 * \fn Pion PI_creerPion(Couleur couleur)
 * \brief Fonction de création d'un pion selon une couleur donnée
 *
 * \param Couleur couleur, la couleur à donner au pion
 * \return Pion
 */
Pion PI_creerPion(Couleur couleur);

/**
 * \fn Couleur PI_obtenirCouleur(Pion pion)
 * \brief Fonction permettant d'obtenir la couleur d'un pion
 *
 * \param Pion pion; le pion dont on veut la couleur
 * \return Couleur
 */
Couleur PI_obtenirCouleur(Pion pion);

/**
 * \fn void PI_retournerPion(Pion* pion)
 * \brief Fonction permettant de retourner un pion
 *
 * \param Pion* pion, le pion à retourner
 */
void PI_retournerPion(Pion* pion);

/**
 * \fn int PI_sontEgaux
 * \brief Fonction testant l'égalité de deux pions
 *
 * \param Pion pion1
 * \param Pion pion2
 * \return int (Booleen)
 */
int PI_sontEgaux(Pion pion1, Pion pion2);

#endif

```

## 1.5 Le fichier « TAD\_Plateau.h »

```

/**
 * \file TAD_PLATEAU.h
 * \brief Implantation du TAD Plateau pour le jeu Othello
 * \author Groupe 1.5
 * \version 1.0
 * \date 2/12/2015
 *
 */

#ifndef __TAD_PLATEAU__
#define __TAD_PLATEAU__
#include "TAD_Position.h"
#include "TAD_Pion.h"
#include "TAD_Couleur.h"

```

```

/**
 * \struct Plateau
 * \brief Le type Plateau permet de représenter un plateau
 *
 */

typedef struct {
    Pion pions[8][8]; /**< les pions du plateau */
    int presencePions[8][8]; /**< la case est remplie ou non : 0 si vide, 1 si remplie
    */
} Plateau;

/**
 * \fn Plateau PLATEAU_creerPlateau()
 * \brief Fonction de création d'un plateau de cases vides, sauf les 4 cases
    centrales
 *
 * \return Plateau
 */
Plateau PL_creerPlateau();

/**
 * \fn Couleur estCaseVide(plateau : Plateau, position : Position)
 * \brief Fonction de création d'une grille de cellules mortes
 *
 * \param Plateau plateau, le plateau
 * \param Position position, la position de la case
 * \return int
 */
int PL_estCaseVide(Plateau plateau, Position position);

/**
 * \fn void viderCase(Plateau* plateau, Position position)
 * \brief Procédure permettant de vider une case
 *
 * \param Plateau* plateau, le plateau
 * \param Position position, la position de la case
 */
void PL_viderCase(Plateau* plateau, Position position);

/**
 * \fn void poserPion(Plateau* plateau, Position position, Pion pion)
 * \brief Procédure permettant de poser un pion sur le plateau
 *
 * \param Plateau* plateau, le plateau
 * \param Position position, la position de la case
 * \param Pion pion, le pion à poser
 */
void PL_poserPion(Plateau* plateau, Position position, Pion pion);

/**
 * \fn Pion obtenirPion(Plateau plateau, Position position)
 * \brief Fonction permettant d'obtenir un pion à une position
 *
 * \param Plateau plateau, le plateau
 * \param Position position, la position de la case
 * \return Pion
 */
Pion PL_obtenirPion(Plateau plateau, Position position);

/**

```

```

* \fn void inverserPion(Plateau* plateau, Position position)
* \brief Procédure permettant d'inverser un pion, donc de changer de joueur
*
* \param Plateau* plateau, le plateau
* \param Position position, la position de la case
*/
void PL_inverserPion(Plateau* plateau, Position position);

#endif

```

## 1.6 Le fichier « TAD\_Position.h »

```

/**
* \file TAD_POSITION.h
* \brief Implantation du TAD Position pour le jeu Othello
* \author Groupe 1.5
* \version 1.0
* \date 2/12/2015
*
*/

#ifndef __TAD_POSITION__
#define __TAD_POSITION__

/**
* \struct Position
* \brief Le type Position permet de représenter une position sur le plateau
*
*/

typedef struct {
    unsigned int ligne; /**< l'indice de la ligne du plateau */
    unsigned int colonne; /**< l'indice de la colonne du plateau */
} Position;

/**
* \fn unsigned int POS_obtenirLigne(Position position)
* \brief Fonction d'obtention de l'indice de la ligne
*
* \param Position position, la position dont on veut la ligne
* \return unsigned int
*/
unsigned int POS_obtenirLigne(Position position);

/**
* \fn unsigned int POS_obtenirColonne(Position position)
* \brief Fonction d'obtention de l'indice de la colonne
*
* \param Position position, la position dont on veut la colonne
* \return unsigned int
*/
unsigned int POS_obtenirColonne(Position position);

/**
* \fn void POS_fixerPosition(unsigned int ligne, unsigned int colonne, Position*
    position)
* \brief Fonction permettant de fixer une position en fonction de ses coordonnées
*
*/

```

```

* \param unsigned int ligne, l'indice de la ligne
* \param unsigned int colonne, l'indice de la colonne
* \param Position* position, la position que l'on veut fixer
*/
void POS_fixerPosition(unsigned int ligne, unsigned int colonne, Position* position);

/**
* \fn POS_sontEgales
* \brief Fonction testant l'égalité de deux positions
*
* \param Position pos1
* \param Position pos2
* \return int (Booleen)
*/
int POS_sontEgales(Position pos1, Position pos2);

#endif

```

## 1.7 Le fichier « Affichage.h »

```

/**
* \file Affichage.h
* \brief Fonctions d'affichage d'un plateau et d'affichage de l'aide
* \author Groupe 1.5
* \version 1.0
* \date 09/12/2015
*
*/

#ifndef __AFFICHAGE__
#define __AFFICHAGE__
#include "TAD_Couleur.h"
#include "TAD_Pion.h"
#include "TAD_Plateau.h"
#include "TAD_Position.h"
#include "TAD_Coup.h"

/**
* \fn void afficherAide()
* \brief Fonction qui affiche l'aide du jeu d'Othello (exécutable appelé sans
      argument)
*
*/
void afficherAide();

/**
* \fn void afficherTournoi(Plateau plateau)
* \brief Fonction qui joue en mode tournoi en affichant les coups choisis
*
* \param Plateau plateau, le plateau du jeu en cours
* \param Coup coup, le coup joué
*/
void afficherTournoi(Plateau plateau, Coup coup, int aPuJouer, int estPartieFinie);

/**
* \fn char intToChar(unsigned int i)
* \brief Fonction qui convertit un entier en une lettre
*
* \param unsigned int i, l'entier à convertir

```

```

    * \return char
    */
char intToChar(unsigned int i);

/**
 * \fn void afficherPlateau(Plateau plateau)
 * \brief Fonction qui affiche l'état du plateau à un instant donné
 *
 * \param Plateau plateau, le plateau du jeu en cours
 * \param Coup coup, le coup joué
 */
void afficherPlateau(Plateau plateau, Coup coup, int aPuJouer, int estPartieFinie);

/**
 * \fn void afficherCoup(Couleur couleurJoueur, Coup coupJoueur)
 * \brief Fonction qui affiche le coup d'un joueur
 *
 * \param Couleur couleurJoueur, la couleur du joueur dont on affiche le coup
 * \param Coup coupJoueur, le coup joué par le joueur de couleur couleurJoueur
 */
void afficherCoup(Couleur couleurJoueur, Coup coupJoueur);

#endif

```

## 1.8 Le fichier « FaireUnePartie.h »

```

/**
 * \file FaireUnePartie.h
 * \brief Implantation de faireunepartie-prive pour le projet othello
 * \author groupe 1.5
 * \version 1.0
 * \date 02/12/2015
 *
 */

#ifndef __FAIREUNEPARTIE__
#define __FAIREUNEPARTIE__
#include "FaireUnePartie_Prive.h"

/**
 * \brief FaireUnePartie regroupe la procedure faireUnePartie qui va permettre de
 *        jouer a l'othello
 *
 */

/**
 * \procedure faireUnePartie(void(*afficherPlateau)(Plateau), Coup(*getCoup)(Plateau,
 *        Pion), Coup(*getCoup)(Plateau,Pion), Couleur* vainqueur, int* estMatchNul,
 *        Couleur couleurJoueur1)
 * \brief Procedure permettant de jouer au jeu de l'othello
 *
 * \param void(*afficherPlateau)(Plateau,Coup,int,int) POINTEUR sur une fonction qui
 *        permet d'afficher le plateau a chaque tour
 * \param Coup(*getCoup1)(Plateau,Pion) permet d'obtenir le coup du joueur 1
 * \param Coup(*getCoup2)(Plateau,Pion) permet d'obtenir le coup du joueur 2
 * \param Couleur* vainqueur permet de determiner le gagnant de la partie
 * \param int* estMatchNul booléen qui permet de savoir si aucun joueur n'a gagné la
 *        partie ou il y'a un gagnant
 * \param Couleur couleurJoueur1 permet d'obtenir la couleur choisie par le joueur 1

```

```

* \
*/

void faireUnePartie(void(*afficherPlateau)(Plateau,Coup,int,int), Coup(*getCoup1)(
    Plateau,Couleur), Coup(*getCoup2)(Plateau,Couleur), Couleur* vainqueur, int*
    estMatchNul, Couleur couleurJoueur1);

/**
 * \procedure void pionEstPresent(Pion pionJoueur, unsigned int x, unsigned int y,
 *   Position* pos, Plateau* plateau, int* pionPresent)
 * \brief Procedure qui permet de savoir si un pion est présent sur le plateau selon
 *   une direction, et si oui quelle est sa position
 *
 * \param Pion pionJoueur, le pion représentant le joueur
 * \param Direction dirATester, la direction à tester
 * \param Position* pos, la position initiale du pion qui, à la fin de l'exécution de
 *   la procédure, renvoie la position du pion trouvé
 * \param Plateau* plateau, le plateau de jeu
 * \param int* pionPresent, qui renvoie 0 si aucun pion conforme n'a été trouvé, 1
 *   sinon
 * \
 */
void pionEstPresent(Pion pionJoueur, Direction dirATester, Position* pos, Plateau*
    plateau, int* pionPresent);

/**
 * \procedure void nbPions (Plateau plateau, unsigned int* scoreJoueur1, unsigned int
 *   * scoreJoueur2)
 * \brief Procedure qui permet de compter le nombre de pions des joueurs 1 et 2 sur
 *   le plateau
 *
 * \param Plateau plateau, le plateau de jeu
 * \param int* nbPionsBlancs, le nombre de pions Blanc
 * \param int* nbPionsNoirs, le nombre de pions Noirs
 * \
 */
void nbPions(Plateau plateau, int* nbPionsNoirs, int* nbPionsBlancs);

#endif

```

## 1.9 Le fichier « FaireUnePartie\_Prive.h »

```

/**
 * \file FaireUnePartie_Prive.h
 * \brief Implantation de faireunepartie-prive pour le projet othello
 * \author groupe 1.5
 * \version 1.0
 * \date 02/12/2015
 *
 */

#ifndef __FAIREUNEPARTIE_PRIVÉ__
#define __FAIREUNEPARTIE_PRIVÉ__
#include "TAD_Plateau.h"
#include "TAD_Couleur.h"
#include "TAD_Position.h"
#include "TAD_Pion.h"
#include "TAD_Coup.h"

```

```

#include "TAD_Coups.h"
#include "FaireUnePartie.h"

/**
 * \brief FaireUnePartie-prive regroupe seulement les fonctions et procédures qu'on
 * va utiliser dans FAIREUNEPARTIE
 *
 */

/* Introduction d'un type privé Direction */

typedef enum {GAUCHE,DROITE,HAUT,BAS,DIAGGH,DIAGGB,DIAGDH,DIAGDB} Direction;

Position DIR_positionSelonDirection(Position posInit, Direction dirDeplacement);
Direction DIR_inverserDirection(Direction dirInit);
int DIR_deplacementValide(Position pos, Direction dirDeplacement);

/**
 * \fn Plateau InitialiserPlateau()
 * \brief Procedure permettant d'initialiser le plateau (place quatre pions au centre
 * )
 *
 * \
 */
void initialiserPlateau(Plateau *plateau);

/**
 * \fn void jouer(Plateau* plateau , Couleur* couleurJoueur, GETCOUP(*
 * obtenirCoupJoueur)(Plateau,Couleur,Coup), int* aPuJouer)
 * \brief Procedure qui permet à un joueur de jouer
 *
 * \param Plateau* plateau, le plateau de l'othello
 * \param Couleur* couleurJoueur, la couleur du joueur qui joue le tour
 * \param GETCOUP(*obtenirCoupJoueur)(Plateau,Couleur,Coup), permet d'obtenir le coup
 * du joueur
 * \param int* aPuJouer, boolean qui permet de savoir si le joueur a pu placer son
 * pion ou pas.
 * \param Coup* coupJoueur, le coup choisi et joué
 * \
 */
void jouer(Plateau* plateau , Couleur* couleurJoueur, Coup(*obtenirCoupJoueur)(
    Plateau,Couleur), int* aPuJouer, Coup* coupJoueur);

/**
 * \fn void jouerCoup (Coup coup, Plateau* plateau)
 * \brief Procedure qui permet de jouer un coup sur le plateau
 *
 * \param Coup coup , le coup que le joueur souhaite jouer
 * \param Plateau* plateau, le plateau de l'othello
 * \
 */
void jouerCoup(Coup coup, Plateau* plateau);

/**
 * \fn void inverserPions(Position pos, Pion pionJoueur, Plateau* plateau)
 * \brief Procedure qui permet de retourner les pions dans toutes les directions si
 * possible, après le coup
 *
 * \param Position pos, la position du coup

```

```

* \param Pion pionJoueur, le pion du coup
* \param Plateau* plateau, le plateau sur lequel est joué le coup
* \
*/
void inverserPions(Position pos, Pion pionJoueur, Plateau* plateau);

/**
* \fn inverserPionsDir(Plateau* plateau, Position posInitiale, Position posCourante,
    Direction dirInversion);
* \brief Procedure qui permet de retourner les pions sur le plateau selon une
    direction donnée
*
* \param Plateau* plateau, le plateau de jeu
* \param Position posInitiale, la position du coup joué
* \param Position posCourante, la position courante sur le plateau
* \param Direction dirInversion, la direction d'inversion
* \
*/
void inverserPionsDir(Plateau* plateau, Position posInitiale, Position posCourante,
    Direction dirInversion);

/**
* \fn void pionEstPresentRecuratif(Pion pionJoueur, Direction dirATester, Position*
    pos, Plateau* plateau, int* pionPresent);
* \brief Procedure qui permet de savoir si un pion est présent sur le plateau selon
    une direction, et si oui quelle est sa position, de manière récursive à partir
    de la case à côté de la position initiale
*
* \param Pion pionJoueur, le pion représentant le joueur
* \param Direction dirATester, la direction de recherche
* \param Position* pos, la position initiale du pion qui, à la fin de l'exécution de
    la procédure, renvoie la position du pion trouvé
* \param Plateau* plateau, le plateau de jeu
* \param int* pionPresent, qui renvoie 0 si aucun pion conforme n'a été trouvé, 1
    sinon
* \
*/
void pionEstPresentRecuratif(Pion pionJoueur, Direction dirATester, Position* pos,
    Plateau* plateau, int* pionPresent);

/**
* \fn finPartie (Plateau plateau, int aPuJouerJoueur1, aPuJouerJoueur2 , unsigned int
    * scoreJoueur1, unsigned int* scoreJoueur2 , int* estFinie)
* \brief Procedure qui permet de déterminer si la partie est finie ou non.
*
* \param Plateau plateau, le plateau de jeu
* \param int aPuJouerJoueur1, 1 si le joueur 1 a pu jouer à son dernier tour, 0
    sinon
* \param int aPuJouerJoueur2, 1 si le joueur 2 a pu jouer à son dernier tour, 0
    sinon
* \param int* nbPionsBlancs, le nombre de pions Blanc
* \param int* nbPionsNoirs, le nombre de pions Noirs
* \param int* estFinie, 1 si la partie est finie, 0 sinon
* \
*/
void finPartie (Plateau plateau, int aPuJouerJoueur1, int aPuJouerJoueur2 , int*
    nbPionsNoirs, int* nbPionsBlancs , int* estFinie);

/**
* \fn int plateauRempli (Plateau plateau)
* \brief Fonction qui renvoie un booléen indiquant si le plateau est rempli ou non.

```



```

*
* \param Plateau plateau, le plateau à tester.
* \return int, le booléen indiquant si le plateau est rempli.
*/
int plateauRempli(Plateau plateau);

#endif

```

## 1.10 Le fichier « ListeCoupsPossibles.h »

```

/**
 * \file ListeCoupsPossibles.h
 * \brief Implantation et signatures des fonctions publiques de ListeCoupsPossibles
 * \author Groupe 1.5
 * \version 1.0
 * \date 02/12/2015
 *
 */

#ifndef __LISTES_COUPS_POSSIBLES__
#define __LISTES_COUPS_POSSIBLES__
#include "TAD_Coup.h"
#include "TAD_Coups.h"
#include "TAD_Plateau.h"
#include "TAD_Couleur.h"
#include "FaireUnePartie.h"

/**
 * \fn Coups listeCoupsPossibles(Plateau plateau, Couleur couleur)
 * \brief Fonction qui retourne un ensemble de coups possibles
 *
 * \param Plateau plateau, le plateau
 * \param Couleur couleur, couleur du joueur courant
 * \return Coups
 */
Coups listeCoupsPossibles(Plateau plateau, Couleur couleur);

/**
 * \fn void copierPlateau(Plateau plateauACopier, Plateau* plateauCopie)
 * \brief Procédure qui copie un plateau sur un autre
 *
 * \param Plateau plateauACopier, le plateau à copier
 * \param Plateau* plateauCopie, le plateau copié
 */
void copierPlateau(Plateau plateauACopier, Plateau* plateauCopie);

/**
 * \fn int coupValide(Plateau plateau, Coup coup)
 * \brief Fonction qui vérifie qu'un coup est valide
 *
 * \param Plateau plateau, le plateau
 * \param Coup coup, le coup à vérifier
 */
int coupValide(Plateau plateau, Coup coup);

#endif

```

### 1.11 Le fichier « ListeCoupsPossibles\_Prive.h »

```
#ifndef __LISTES_COUPS_POSSIBLES_PRIVÉ__
#define __LISTES_COUPS_POSSIBLES_PRIVÉ__
#include "ListeCoupsPossibles.h"
#include "TAD_Coup.h"
#include "TAD_Plateau.h"
#include "FaireUnePartie.h"
#include "FaireUnePartie_Prive.h"

#endif
```

### 1.12 Le fichier « ObtenirCoupHumain.h »

```
/**
 * \file ObtenirCoupHumain.h
 * \brief Implantation et signatures des fonctions publiques d'ObtenirCoupHumain
 * \author Groupe 1.5
 * \version 1.0
 * \date 02/12/2015
 *
 */

#ifndef __OBTENIR_COUP_HUMAIN__
#define __OBTENIR_COUP_HUMAIN__
#include "TAD_Coup.h"
#include "TAD_Plateau.h"
#include "TAD_Couleur.h"

/**
 * \fn Coup obtenirCoupIA(plateau Plateau, couleur Couleur)
 * \brief Fonction permettant de récupérer le coup joué par l'humain
 *
 * \param Plateau plateau, le plateau du jeu en cours
 * \param Couleur couleur, la couleur représentée par l'humain
 * \return Coup
 */
Coup obtenirCoupHumain(Plateau plateau, Couleur couleur);

#endif
```

### 1.13 Le fichier « ObtenirCoupIA.h »

```
/**
 * \file ObtenirCoupIA.h
 * \brief Implantation et signatures des fonctions publiques d'ObtenirCoupIA
 * \author Groupe 1.5
 * \version 1.0
 * \date 02/12/2015
 *
 */

#ifndef __OBTENIR_COUP_IA__
```

```

#define __OBTENIR_COUP_IA__
#include "TAD_Coup.h"
#include "TAD_Plateau.h"
#include "TAD_Couleur.h"

// /**
//  * \def PROFONDEUR 5
//  * \brief Profondeur d'exploration de scoreDUnCoup et minMax
//  *
//  */
//
// #define PROFONDEUR 4

/**
 * \fn Coup obtenirCoupIA(plateau Plateau, couleur Couleur)
 * \brief Fonction permettant de récupérer le coup joué par l'IA
 *
 * \param Plateau plateau, le plateau du jeu en cours
 * \param Couleur couleur, la couleur représentée par l'IA
 * \return Coup
 */
Coup obtenirCoupIA(Plateau plateau, Couleur couleur);

#endif

```

## 1.14 Le fichier « ObtenirCoupIA\_Prive.h »

```

#ifndef __OBTENIR_COUP_IA_PRIVÉ__
#define __OBTENIR_COUP_IA_PRIVÉ__
#include "TAD_Plateau.h"
#include "TAD_Couleur.h"

/* Récupération de la profondeur maximale d'exploration */
unsigned int profondeur(void);

/* Score associé à un coup joué par le joueur de la couleur couleurCourante */
int scoreDUnCoup(Plateau plateau, Coup coup, Couleur couleurRef, Couleur
    couleurCourante, unsigned int profondeurCourante, int** grilleScore);

/* Permet de "remonter" le score du meilleur coup dans l'arbre des possibilités de
    jeu */
int minMax(Plateau plateau, Couleur couleurRef, Couleur couleurCourante, unsigned int
    profondeurCourante, int** grilleScore);

/* Permet d'obtenir le score de la partie pour la couleur donnée en entrée */
int score(Plateau plateau, Couleur couleur, int** grilleScore);

/* Evaluation du plateau selon différentes règles de jeu et de priorité */
int evaluerPlateau(Plateau plateau, Couleur couleur, int** grilleScore);

/* Evaluation du nombre de coups possibles de l'adversaire */
int evaluerNbCoupsPossiblesAdversaire(Plateau plateau, Couleur couleur);

/* Evaluation de la différence du nombre de pions possédés et du nombre de pions
    possédés par l'adversaire */
int evaluerNbPionsCouleur(Plateau plateau, Couleur couleur);

/* Evaluation du plateau selon la position des différents pions */

```

```
int evaluerPositionsPionsPlateau(Plateau plateau, Couleur couleur, int** grilleScore)
;

/* Initialisation de la grille qui attribue à chaque position un poids */
int** initialiserGrilleScore();

int min(int a, int b);

int max(int a, int b);

#endif
```

## Chapitre 2

# Les fichiers *C*

### 2.1 Le fichier « TAD\_Couleur.c »

```
#include "TAD_Couleur.h"

/* Partie publique */

Couleur CL_blanc(){
    return BLANC;
}

Couleur CL_noir(){
    return NOIR;
}

Couleur CL_changerCouleur(Couleur couleur){
    if (CL_sontEgales(couleur, CL_blanc())){
        return CL_noir();
    }
    else {
        return CL_blanc();
    }
}

int CL_sontEgales(Couleur couleur1, Couleur couleur2){
    return (couleur1==couleur2);
}
```

### 2.2 Le fichier « TAD\_Coup.c »

```
#include "TAD_Coup.h"
#include "TAD_Position.h"
#include "TAD_Pion.h"

/* Partie publique */

Coup CP_creerCoup(Position position, Pion pion){
    Coup coup;
    coup.position = position;
    coup.pion = pion;
    return coup;
}
```

```

Position CP_obtenirPositionCoup(Coup coup){
    return coup.position;
}

Pion CP_obtenirPionCoup(Coup coup){
    return coup.pion;
}

int CP_sontEgaux(Coup coup1, Coup coup2){
    return (POS_sontEgales(coup1.position, coup2.position) && PI_sontEgaux(coup1.pion, coup2.pion));
}

```

## 2.3 Le fichier « TAD\_Coups.c »

```

#include "TAD_Coups.h"
#include <assert.h>

/* Partie publique */

void CPS_creerCoups(Coups* coups){
    coups->nbCps = 0;
}

void CPSajouterCoups(Coups* coups, Coup coup){
    coups->tabCoups[coups->nbCps] = coup;
    coups->nbCps = (coups->nbCps)+1;
}

unsigned int CPS_nbCoups(Coups coups){
    return coups.nbCps;
}

Coup CPS_iemeCoup(Coups coups, unsigned int i){
    assert(i>=0 && i<coups.nbCps);
    return coups.tabCoups[i];
}

```

## 2.4 Le fichier « TAD\_Pion.c »

```

#include "TAD_Couleur.h"
#include "TAD_Pion.h"
#include <stdio.h>

/* Partie publique */

Pion PI_creerPion(Couleur couleurPion){
    Pion pion;
    pion.couleur=couleurPion;
    return pion;
}

Couleur PI_obtenirCouleur(Pion pion){
    return pion.couleur;
}

```

```

void PI_retournerPion(Pion* pion){
    Couleur couleurAChanger,nouvelleCouleur;
    couleurAChanger=PI_obtenirCouleur(*pion);
    nouvelleCouleur=CL_changerCouleur(couleurAChanger);
    pion->couleur=nouvelleCouleur;
}

int PI_sontEgaux(Pion pion1, Pion pion2){
    return CL_sontEgales(pion1.couleur,pion2.couleur);
}

```

## 2.5 Le fichier « TAD\_Plateau.c »

```

#include "TAD_Position.h"
#include "TAD_Pion.h"
#include "TAD_Couleur.h"
#include "TAD_Plateau.h"

Plateau PL_creerPlateau(){
    Plateau plateau;
    unsigned int i,j;
    for(i=1;i<9;i++){
        for(j=1;j<9;j++){
            plateau.pions[i-1][j-1]=PI_creerPion(CL_blanc());
            plateau.presencePions[i-1][j-1]=0;
        }
    }
    return plateau;
}

int PL_estCaseVide(Plateau plateau, Position position){
    unsigned int i,j;
    i=POS_obtenirLigne(position);
    j=POS_obtenirColonne(position);
    if((plateau.presencePions[i][j])==0){
        return 1;
    }
    else{
        return 0;
    }
}

void PL_viderCase(Plateau* plateau, Position position){
    unsigned int i,j;
    i=POS_obtenirLigne(position);
    j=POS_obtenirColonne(position);
    plateau->presencePions[i][j]=0;
}

void PL_poserPion(Plateau* plateau, Position position, Pion pion){
    unsigned int i,j;
    i=POS_obtenirLigne(position);
    j=POS_obtenirColonne(position);
    plateau->pions[i][j]=pion;
    plateau->presencePions[i][j]=1;
}

Pion PL_obtenirPion(Plateau plateau, Position position){

```

```

    Pion pion;
    unsigned int i,j;
    i=POS_obtenirLigne(position);
    j=POS_obtenirColonne(position);
    pion=plateau.pions[i][j];
    return pion;
}

void PL_inverserPion(Plateau* plateau, Position position){
    Pion pion;
    unsigned int i,j;
    i=POS_obtenirLigne(position);
    j=POS_obtenirColonne(position);
    pion=plateau->pions[i][j];
    PI_retournerPion(&pion);
    plateau->pions[i][j]=pion;
}

```

## 2.6 Le fichier « TAD\_Position.c »

```

#include "TAD_Position.h"
#include <assert.h>

/* Partie publique */

unsigned int POS_obtenirLigne(Position position){
    return position.ligne;
}

unsigned int POS_obtenirColonne(Position position){
    return position.colonne;
}

void POS_fixerPosition(unsigned int ligne, unsigned int colonne, Position* position){
    assert((ligne>=0) && (ligne<8) && (colonne>=0) && (colonne<8));
    position->ligne=ligne;
    position->colonne=colonne;
}

int POS_sontEgales(Position pos1, Position pos2){
    return (((pos1.ligne) == (pos2.ligne)) && ((pos1.colonne) == (pos2.colonne)));
}

```

## 2.7 Le fichier « Affichage.c »

```

#include <stdio.h>
#include <stdlib.h>
#include "TAD_Couleur.h"
#include "TAD_Pion.h"
#include "TAD_Coup.h"
#include "TAD_Plateau.h"
#include "TAD_Position.h"
#include "Affichage.h"
#include "FaireUnePartie.h"

void afficherAide(){

```



```

printf("Aide du programme othello \n"
"Les options possibles sont : \n"
"\t othello standard blanc|noir [profondeur>2] \n"
"\t\t permet de jouer contre l'ordinateur en lui donnant les blancs \n"
"\t\t ou les noirs \n"
"\t\t par défaut la profondeur d'analyse est égale à 4 \n \n"
"\t othello tournoi blanc|noir [profondeur>2] \n"
"\t\t permet de faire jouer le programme dans un mode tournoi en \n"
"\t\t lui donnant les blancs ou les noirs\n"
"\t\t par défaut la profondeur d'analyse est égale à 5\n\n");
}

void afficherTournoi(Plateau plateau, Coup coup, int aPuJouer, int estPartieFinie){
    int nbPionsNoirs, nbPionsBlancs, ligne;
    unsigned int i, j;
    char colonne;

    if(estPartieFinie){
        nbPions(plateau, &nbPionsNoirs, &nbPionsBlancs);
        if(nbPionsNoirs==nbPionsBlancs){
            printf("nulle\n");
        }
        else{
            if(nbPionsNoirs>nbPionsBlancs){
                printf("noir\n");
            }
            else{
                printf("blanc\n");
            }
        }
    }
    else{
        if(!aPuJouer){
            printf("passe\n");
        }
        else{
            i=POS_obtenirLigne(CP_obtenirPositionCoup(coup));
            j=POS_obtenirColonne(CP_obtenirPositionCoup(coup));
            colonne=intToChar(i);
            ligne=j+1;
            printf("%c%d\n", colonne, ligne);
        }
    }
}

char intToChar(unsigned int i){
    char res;
    switch(i){
        case 0 :
            res='a';
            break;
        case 1 :
            res='b';
            break;
        case 2 :
            res='c';
            break;
        case 3 :
            res='d';
            break;
        case 4 :

```

```

        res='e';
        break;
        case 5 :
            res='f';
            break;
        case 6 :
            res='g';
            break;
        case 7 :
            res='h';
            break;
    }
    return(res);
}

void afficherPlateau(Plateau plateau, Coup coup, int aPuJouer, int estPartieFinie){
    Couleur couleurDernierJoueur;
    couleurDernierJoueur = PI_obtenirCouleur(CP_obtenirPionCoup(coup));
    unsigned int i,j;
    int nbPionsNoirs,nbPionsBlancs;
    Couleur couleurBlanc, couleurNoir;
    couleurBlanc=CL_blanc();
    couleurNoir=CL_noir();

    if (aPuJouer && !((POS_obtenirColonne(CP_obtenirPositionCoup(coup))==4) && (
        POS_obtenirLigne(CP_obtenirPositionCoup(coup))==4))){
        afficherCoup(couleurDernierJoueur, coup);
    }

    if(!estPartieFinie){
        printf("      1  2  3  4  5  6  7  8 \n");
        printf("      \n");
        Position position;
        for(i=1;i<9;i++){
            printf("  %d |", i);
            for (j=1;j<9;j++){
                POS_fixerPosition(i-1,j-1,&position);
                if (!PL_estCaseVide(plateau,position)){
                    if (CL_sontEgales((PI_obtenirCouleur(PL_obtenirPion(plateau,position))),
                        couleurBlanc)){
                        printf("  ");
                        printf("|");
                    }
                    else
                    {
                        if (CL_sontEgales((PI_obtenirCouleur(PL_obtenirPion(plateau,position))),
                            couleurNoir)){
                            printf("  |");
                        }
                    }
                }
            }
            printf("\n      \n");
        }
    }
    else{
        nbPions(plateau, &nbPionsNoirs, &nbPionsBlancs);
    }
}

```

```

    if(nbPionsNoirs==nbPionsBlancs){
        printf("La partie est nulle\n");
    }
    else{
        if(nbPionsNoirs>nbPionsBlancs){
            printf("Le joueur ayant les pions a gagné\n");
        }
        else{
            printf("Le joueur ayant les pions a gagné\n");
        }
    }
}

void afficherCoup(Couleur couleurJoueur, Coup coupJoueur){
    Position pos = CP_obtenirPositionCoup(coupJoueur);
    unsigned int ligne = POS_obtenirLigne(pos);
    unsigned int colonne = POS_obtenirColonne(pos);
    ligne ++; // pour l'affichage
    colonne ++; // pour l'affichage

    if (CL_sontEgales(couleurJoueur,CL_blanc())){
        printf("\n\n\n Le joueur joue en : \n \tligne : %u colonne : %u \n\n",ligne,
            colonne);
    }
    else{
        if (CL_sontEgales(couleurJoueur,CL_noir())){
            printf("\n\n\n Le joueur joue en : \n \tligne : %u colonne : %u \n\n",ligne,
                colonne);
        }
    }
}

```

## 2.8 Le fichier « FaireUnePartie.c »

```

#include "FaireUnePartie.h"
#include "FaireUnePartie_Prive.h"
#include "ListeCoupsPossibles.h"
#include "Affichage.h"
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <ncurses.h>
#define TRUE 1
#define FALSE 0

/* Partie publique */

void faireUnePartie(void(*afficher)(Plateau,Coup,int,int), Coup(*getCoup1)(Plateau,
Couleur), Coup(*getCoup2)(Plateau,Couleur), Couleur *vainqueur, int* estMatchNul,
Couleur couleurJoueur1)
{
    Plateau plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);
    int aPuJouerJoueur1=TRUE, aPuJouerJoueur2=TRUE;
    int estFinie=FALSE;
    Couleur couleurJoueur2=CL_changerCouleur(couleurJoueur1);
    int nbPionsBlancs=2, nbPionsNoirs=2;
    Coup coupJoueur1, coupJoueur2;

```

```

    Position positionInitialisation;
    POS_fixerPosition(4,4,&positionInitialisation);
    coupJoueur1 = CP_creerCoup(positionInitialisation,PI_creerPion(CL_blanc()));
    coupJoueur2 = CP_creerCoup(positionInitialisation,PI_creerPion(CL_blanc()));

    afficher(plateau,coupJoueur1,aPuJouerJoueur1,estFinie);
    while (!estFinie) {
        jouer(&plateau,&couleurJoueur1,getCoup1,&aPuJouerJoueur1,&coupJoueur1);
        afficher(plateau,coupJoueur1,aPuJouerJoueur1,estFinie);
        jouer(&plateau,&couleurJoueur2,getCoup2,&aPuJouerJoueur2,&coupJoueur2);
        afficher(plateau,coupJoueur2,aPuJouerJoueur2,estFinie);
        finPartie(plateau,aPuJouerJoueur1,aPuJouerJoueur2,&nbPionsNoirs,&
            nbPionsBlancs,&estFinie);
    }
    afficher(plateau,coupJoueur2,aPuJouerJoueur2,estFinie);
    if (nbPionsBlancs==nbPionsNoirs){
        *vainqueur=CL_blanc();
        *estMatchNul=TRUE;
    }
    else (*estMatchNul=FALSE);{
        if (nbPionsBlancs>nbPionsNoirs){
            *vainqueur=CL_blanc();
        }
        else {
            *vainqueur=CL_noir();
        }
    }
}

/* Partie privée */

void initialiserPlateau(Plateau *plateauDeJeu){
    Pion pionNoir;
    Pion pionBlanc;
    Position positionPion;

    *plateauDeJeu=PL_creerPlateau();
    pionNoir=PI_creerPion(CL_noir());
    pionBlanc=PI_creerPion(CL_blanc());
    POS_fixerPosition(3,3,&positionPion);
    PL_poserPion(plateauDeJeu, positionPion, pionBlanc);
    POS_fixerPosition(3,4,&positionPion);
    PL_poserPion(plateauDeJeu, positionPion, pionNoir);
    POS_fixerPosition(4,3,&positionPion);
    PL_poserPion(plateauDeJeu, positionPion, pionNoir);
    POS_fixerPosition(4,4,&positionPion);
    PL_poserPion(plateauDeJeu, positionPion, pionBlanc);
}

void jouer(Plateau* plateau , Couleur* couleurJoueur , Coup(*getCoup)(Plateau,Couleur)
, int* aPuJouer , Coup* coupJoueur){
    unsigned int i;
    int res;
    Coups coups;
    res=FALSE;
    *coupJoueur=getCoup(*plateau,*couleurJoueur);
    coups=listeCoupsPossibles(*plateau,*couleurJoueur);
    printf("%d \n",CPS_nbCoups(coups));

```

```

    if (CPS_nbCoups(coups)>0){
        printf("OK 1 \n");
        for(i=0;i<CPS_nbCoups(coups);i++){
            if (CP_sontEgaux(CPS_iemeCoup(coups,i),*coupJoueur)) {
                jouerCoup(*coupJoueur,plateau);
                res=TRUE;
            }
        }
    }
    *aPuJouer=res;
}

void jouerCoup (Coup coup, Plateau* plateau){
    Position pos;
    Pion pionJoueur;

    PL_poserPion(plateau,CP_obtenirPositionCoup(coup),CP_obtenirPionCoup(coup));
    pos=CP_obtenirPositionCoup(coup);
    pionJoueur=CP_obtenirPionCoup(coup);
    inverserPions(pos,pionJoueur,plateau);
}

void inverserPions(Position pos, Pion pionJoueur, Plateau* plateau){
    Position posTmp;
    Direction dir;
    int pionPresent;
    for (dir = GAUCHE; dir <= DIAGDB; dir++){
        posTmp = pos;
        pionEstPresent(pionJoueur,dir,&posTmp,plateau,&pionPresent);
        if (pionPresent) {
            inverserPionsDir(plateau,pos,DIR_positionSelonDirection(posTmp,
                DIR_inverserDirection(dir)),DIR_inverserDirection(dir));
        }
    }
}

void inverserPionsDir(Plateau* plateau, Position posInitiale, Position posCourante,
    Direction dirInversion){
    Position posSuivante=posCourante;
    unsigned int inew,jnew;
    inew=POS_obtenirLigne(DIR_positionSelonDirection(posSuivante,dirInversion));
    jnew=POS_obtenirColonne(DIR_positionSelonDirection(posSuivante,dirInversion));

    if (!(POS_sontEgales(posInitiale,posCourante)) && DIR_deplacementValide(
        posCourante, dirInversion)){
        PL_inverserPion(plateau,posCourante);
        POS_fixerPosition(inew,jnew,&posSuivante);
        inverserPionsDir(plateau,posInitiale,posSuivante,dirInversion);
    }
}

void pionEstPresent(Pion pionJoueur, Direction dirATester, Position* pos, Plateau*
    plateau, int* pionPresent){
    Couleur couleurAdversaire;
    couleurAdversaire = CL_changerCouleur(PI_obtenirCouleur(pionJoueur));
    if (!DIR_deplacementValide(*pos,dirATester)) {
        *pionPresent = FALSE;}
    else {

```

```

        *pos = DIR_positionSelonDirection(*pos, dirATester);
        /* On doit regarder que la case n'est pas vide car le plateau est rempli des
           pions blancs par défaut */
        if (CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(*plateau, *pos)),
            couleurAdversaire) && (!PL_estCaseVide(*plateau, *pos))) {
            *pos = DIR_positionSelonDirection(*pos, dirATester);
            pionEstPresentRekursif(pionJoueur, dirATester, pos, plateau, pionPresent);
        }
        else {
            *pionPresent = FALSE;
        }
    }
}

void pionEstPresentRekursif(Pion pionJoueur, Direction dirATester, Position* pos,
    Plateau* plateau, int* pionPresent)
{
    Couleur couleurJoueur;
    couleurJoueur = PI_obtenirCouleur(pionJoueur);
    if (PL_estCaseVide(*plateau, *pos)) {
        *pionPresent = FALSE;
    }
    else {
        if (CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(*plateau, *pos)),
            couleurJoueur)) {
            *pionPresent = TRUE;
        }
        else {
            if (!DIR_deplacementValide(*pos, dirATester)) {
                *pionPresent = FALSE;
            }
            else {
                *pos = DIR_positionSelonDirection(*pos, dirATester);
                pionEstPresentRekursif(pionJoueur, dirATester, pos, plateau, pionPresent);
            }
        }
    }
}

void finPartie (Plateau plateau, int aPuJouerJoueur1, int aPuJouerJoueur2, int*
    nbPionsNoirs, int* nbPionsBlancs, int* estFinie)
{
    printf("J1 : %d, J2 : %d, rempli : %d \n", aPuJouerJoueur1, aPuJouerJoueur2,
        plateauRempli(plateau));
    if (((aPuJouerJoueur1 == FALSE) && (aPuJouerJoueur2 == FALSE)) || (plateauRempli(
        plateau))) {
        nbPions(plateau, nbPionsNoirs, nbPionsBlancs);
        *estFinie = TRUE;
    }
}

void nbPions (Plateau plateau, int* nbPionsNoirs, int* nbPionsBlancs)
{
    *nbPionsNoirs = 0;
    *nbPionsBlancs = 0;
    Position pos;
    Couleur couleur = CL_noir();
    unsigned int i, j;
    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {

```

```

        POS_fixerPosition(i,j,&pos);
        if((!PL_estCaseVide(plateau,pos))){
            if (CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau,pos)),
                couleur)){
                *nbPionsNoirs=*nbPionsNoirs+1;
            }
            else {
                *nbPionsBlancs=*nbPionsBlancs+1;
            }
        }
    }
}

int plateauRempli(Plateau plateau){
    int res = TRUE;
    unsigned int i=0,j=0;
    Position position;
    while(res && (i<8)){
        while(res && (j<8)){
            POS_fixerPosition(i,j,&position);
            if (PL_estCaseVide(plateau,position)){
                res = FALSE;
            }
            j=j+1;
        }
        j=0;
        i=i+1;
    }
    return res;
}

/* Introduction d'un type privé Direction */

Direction DIR_inverserDirection(Direction dirInit){
    Direction newDir;
    switch(dirInit){
        case GAUCHE :
            newDir = DROITE;
            break;
        case DROITE :
            newDir = GAUCHE;
            break;
        case HAUT :
            newDir = BAS;
            break;
        case BAS :
            newDir = HAUT;
            break;
        case DIAGGH :
            newDir = DIAGDB;
            break;
        case DIAGGB :
            newDir = DIAGDH;
            break;
        case DIAGDH :
            newDir = DIAGGB;
            break;
        case DIAGDB :

```

```

        newDir = DIAGGH;
        break;
    }
    return newDir;
}

Position DIR_positionSelonDirection(Position posInit, Direction dirDeplacement){
    Position newPos;
    unsigned int i,j;
    i = POS_obtenirLigne(posInit);
    j = POS_obtenirColonne(posInit);
    if (DIR_deplacementValide(posInit,dirDeplacement)){
        switch(dirDeplacement){
            case GAUCHE :
                POS_fixerPosition(i,j-1, &newPos);
                break;
            case DROITE :
                POS_fixerPosition(i,j+1, &newPos);
                break;
            case HAUT :
                POS_fixerPosition(i-1,j, &newPos);
                break;
            case BAS :
                POS_fixerPosition(i+1,j, &newPos);
                break;
            case DIAGGH :
                POS_fixerPosition(i-1,j-1, &newPos);
                break;
            case DIAGGB :
                POS_fixerPosition(i+1,j-1, &newPos);
                break;
            case DIAGDH :
                POS_fixerPosition(i-1,j+1, &newPos);
                break;
            case DIAGDB :
                POS_fixerPosition(i+1,j+1, &newPos);
                break;
        }
    }
    return newPos;
}

int DIR_deplacementValide(Position pos, Direction dirDeplacement){
    int valide;
    unsigned int i,j;
    i = POS_obtenirLigne(pos);
    j = POS_obtenirColonne(pos);
    switch(dirDeplacement){
        case GAUCHE :
            valide = (j >= 1);
            break;
        case DROITE :
            valide = (j <= 6);
            break;
        case HAUT :
            valide = (i >= 1);
            break;
        case BAS :
            valide = (i <= 6);

```



```

        break;
    case DIAGGH :
        valide = ((i >= 1) && (j >= 1));
        break;
    case DIAGGB :
        valide = ((i <= 6) && (j >= 1));
        break;
    case DIAGDH :
        valide = ((i >= 1) && (j <= 6));
        break;
    case DIAGDB :
        valide = ((i <= 6) && (j <= 6));
        break;
    }
    return valide;
}

/* Fin du type privé Direction */

```

## 2.9 Le fichier « ListesCoupsPossibles.c »

```

#include "TAD_Plateau.h"
#include "TAD_Coup.h"
#include "TAD_Couleur.h"
#include "ListesCoupsPossibles_Prive.h"
#include "ListesCoupsPossibles.h"
#include "FaireUnePartie.h"
#include "FaireUnePartie_Prive.h"
#include <math.h>
#include <string.h>
#include <stdio.h>

/* Partie publique */

Coups listeCoupsPossibles(Plateau plateau, Couleur couleur){
    Coups coupsPossibles;
    Position positionTest;
    Coup coupTest;
    Pion pionJoueur;
    unsigned int i,j;
    int nbPionsBlancs,nbPionsNoirs;
    CPS_creerCoups(&coupsPossibles);
    pionJoueur = PI_creerPion(couleur);
    for (i = 0; i < 8; i++){
        for (j = 0; j < 8; j++){
            POS_fixerPosition(i,j,&positionTest);
            if(PL_estCaseVide(plateau,positionTest)) {
                coupTest = CP_creerCoup(positionTest,pionJoueur);
                if (coupValide(plateau,coupTest)){
                    CPSajouterCoups(&coupsPossibles,coupTest);
                }
            }
        }
    }
    return coupsPossibles;
}

```

```

/* Partie privée */

int coupValide(Plateau plateau, Coup coup) {
    int pionPresent;
    Position pos, posTmp;
    Pion pionJoueur;
    pionPresent = 0;
    pos = CP_obtenirPositionCoup(coup);
    pionJoueur = CP_obtenirPionCoup(coup);
    Direction dir = GAUCHE;
    while(!(pionPresent) && (dir <= DIAGDB)) {
        posTmp = pos;
        if(DIR_deplacementValide(posTmp, dir) && PL_estCaseVide(plateau, pos)){
            pionEstPresent(pionJoueur, dir, &posTmp, &plateau, &pionPresent);
        }
        dir++;
    }
    return pionPresent;
}

void copierPlateau(Plateau plateauACopier, Plateau* plateauCopie){
    *plateauCopie = PL_creerPlateau();
    memcpy(&(plateauCopie->pions), &(plateauACopier.pions), sizeof(Pion)*8*8);
    memcpy(&(plateauCopie->presencePions), &(plateauACopier.presencePions), sizeof(int)*8*8);
}

```

## 2.10 Le fichier « ObtenirCoupHumain.c »

```

#include "TAD_Couleur.h"
#include "TAD_Coup.h"
#include "TAD_Plateau.h"
#include "ListeCoupsPossibles.h"
#include <stdio.h>

Coup obtenirCoupHumain(Plateau plateau, Couleur couleur){
    unsigned int i=9, j=9;
    int estValide = 0;
    Coup coup;
    Position position;
    Pion pion;
    if(CL_sontEgales(couleur, CL_blanc())){
        printf("Joueur : \n");
    }
    else {
        printf("Joueur : \n");
    }
    while(estValide==0) {
        while((i > 8) || (i < 1)){
            printf("Veuillez saisir un numéro de ligne (de 1 à 8) : \n");
            scanf("%u", &i);
        }
        while((j > 8) || (j < 1)){
            printf("Veuillez saisir un numéro de colonne (de 1 à 8) : \n");
            scanf("%u", &j);
        }
    }
}

```

```

    i=i-1;
    j=j-1;
    POS_fixerPosition(i,j,&position);
    pion=PI_creerPion(couleur);
    coup=CP_creerCoup(position,pion);
    estValide=coupValide(plateau,coup);
    i=9;
    j=9;
    if(!estValide){
        printf("Coup non valide, recommencez \n");
    }
}
return coup;
}

```

## 2.11 Le fichier « ObtenirCoupIA.c »

```

#include "ObtenirCoupIA.h"
#include "ObtenirCoupIA_Prive.h"
#include "FaireUnePartie.h"
#include "ListeCoupsPossibles.h"
#include "TAD_Coups.h"
#include <stdlib.h>
#include <stdio.h>

#define INFINI 10000 /* Valeur affectée pour signifier qu'un coup est gagnant. */
#define PROFONDEUR 2

/* Partie publique */

Coup obtenirCoupIA(Plateau plateau, Couleur couleur){
    Coups coupsPossibles;
    unsigned int i,profondeurMinMax=profondeur();
    int scoreCourant, meilleurScore;
    Coup coupCourant, meilleurCoup;
    int** grilleScore=initialiserGrilleScore();
    coupsPossibles=listeCoupsPossibles(plateau,couleur);
    if (CPS_nbCoups(coupsPossibles) > 0) {
        meilleurCoup = CPS_iemeCoup(coupsPossibles,0); // Le premier coup de la liste à l'
        // indice 0 ici, contrairement au pseudo-code
        meilleurScore = scoreDUnCoup(plateau,meilleurCoup,couleur,couleur,
        profondeurMinMax, grilleScore);
        for (i=1;i<CPS_nbCoups(coupsPossibles);i++) { // cf remarque précédente : le 2nd
        // coup est à l'indice 1 etc...
            coupCourant = CPS_iemeCoup(coupsPossibles,i);
            scoreCourant = scoreDUnCoup(plateau,coupCourant,couleur,couleur,
            profondeurMinMax, grilleScore);
            if ((scoreCourant > meilleurScore) && coupValide(plateau,coupCourant)) {
                meilleurCoup = coupCourant;
                meilleurScore = scoreCourant;
            }
        }
    }
    free(grilleScore);
    return meilleurCoup;
}

```

```

/* Partie privée */

unsigned int profondeur(void){
    return PROFONDEUR;
}

int scoreDUnCoup(Plateau plateau, Coup coup, Couleur couleurRef, Couleur
couleurCourante, unsigned int profondeurCourante, int** grilleScore){
    Plateau plateauTest;
    copierPlateau(plateau,&plateauTest);
    jouerCoup(coup, &plateauTest);
    if (plateauRempli(plateauTest) || profondeurCourante==0){
        return score(plateauTest, couleurRef, grilleScore);
    }
    else{
        return minMax(plateauTest, couleurRef, CL_changerCouleur(
couleurCourante), profondeurCourante-1, grilleScore);
    }
}

int minMax(Plateau plateau, Couleur couleurRef, Couleur couleurCourante, unsigned int
profondeurCourante, int** grilleScore){
    Coups coupsPossibles;
    int resultat, score;
    unsigned int i;

    coupsPossibles = listeCoupsPossibles(plateau, couleurCourante);
    if (CPS_nbCoups(coupsPossibles) > 0){
        resultat = scoreDUnCoup(plateau, CPS_iemeCoup(coupsPossibles, 0),
couleurRef, couleurCourante, profondeurCourante, grilleScore);
        for (i=0 ; i<CPS_nbCoups(coupsPossibles);i++){
            score = scoreDUnCoup(plateau, CPS_iemeCoup(coupsPossibles, i), couleurRef,
couleurCourante, profondeurCourante, grilleScore);
            if (CL_sontEgales(couleurCourante,couleurRef)){
                resultat = max(resultat, score);
            }
            else{
                resultat = min(resultat, score);
            }
        }
    }
    else{
        if (CL_sontEgales(couleurCourante,couleurRef)){
            resultat = INFINI;
        }
        else{
            resultat = -1*INFINI;
        }
    }
    return(resultat);
}

int score(Plateau plateau, Couleur couleur, int** grilleScore){
    return evaluerPlateau(plateau,couleur, grilleScore);
}

int evaluerPlateau(Plateau plateau, Couleur couleur, int** grilleScore){
    int evaluer1,evaluer2,evaluer3,res;
    evaluer1=evaluerNbCoupsPossiblesAdversaire(plateau,couleur);
    evaluer2=evaluerNbPionsCouleur(plateau,couleur);

```

```

    evaluer3=evaluerPositionsPionsPlateau(plateau,couleur,grilleScore);
    res=evaluer1+evaluer2+evaluer3; /* Il serait peut-être utile de donner un
        coefficient à chaque evaluation ? */

    return res;
}

int evaluerNbCoupsPossiblesAdversaire(Plateau plateau, Couleur couleur){
    Coups coupsAdversaire;
    Couleur couleurAdversaire;
    int nbCoupsAdversaire,res;

    couleurAdversaire=CL_changerCouleur(couleur);
    coupsAdversaire=listeCoupsPossibles(plateau, couleurAdversaire);
    nbCoupsAdversaire=CPS_nbCoups(coupsAdversaire);

    res=(60-10*nbCoupsAdversaire); /* Le mieux est que l'adversaire ait 0 coups
        possibles. Plus il en a, moins l'évaluation est bonne. */
    return res;
}

int evaluerNbPionsCouleur(Plateau plateau, Couleur couleur){
    int nbPionsNoirs,nbPionsBlancs;
    int res;

    nbPions(plateau,&nbPionsNoirs,&nbPionsBlancs);
    if (CL_sontEgales(couleur,CL_noir())) {
        res=nbPionsNoirs-nbPionsBlancs;
    }
    else {
        res=nbPionsBlancs-nbPionsNoirs;
    }
    return res;
}

int evaluerPositionsPionsPlateau(Plateau plateau, Couleur couleur, int** grilleScore)
{
    unsigned int i,j;
    Position pos;
    int resJoueur,resAdversaire,res;

    resJoueur=0;
    resAdversaire=0;
    for(i=1;i<9;i++){
        for(j=1;j<9;j++){
            POS_fixerPosition(i-1,j-1,&pos);
            if(!PL_estCaseVide(plateau,pos) && CL_sontEgales(PI_obtenirCouleur(
                PL_obtenirPion(plateau,pos)),couleur)) {
                resJoueur=resJoueur+grilleScore[i-1][j-1];
            }
            else if(!PL_estCaseVide(plateau,pos)) {
                resAdversaire=resAdversaire+grilleScore[i-1][j-1];
            }
        }
    }
    res=resJoueur-resAdversaire;
    return res;
}

/* Tirée de http://emmanuel.adam.free.fr/site/IMG/pdf/jeuP.pdf */

```

```

int** initialiserGrilleScore(){
    unsigned int i,j;
    int** grilleScore=(int**)malloc(8*sizeof(int*)); /* allocation des colonnes */

    for(i=0;i<8;i++){
        grilleScore[i] = (int*) malloc(8*sizeof(int)); /* allocation du
            nombre de cases par colonnes */

    for(i=1;i<9;i++){
        for(j=1;j<9;j++){
            if(((i==1) && (j==1)) || ((i==1) && (j==8)) || ((i==8) && (j==1)) || ((i==8) &&
                (j==8))){
                grilleScore[i-1][j-1]=500;
            }
            else{
                if(((i==1) && (j==2)) || ((i==2) && (j==1)) || ((i==8) && (j==7)) || ((i==7)
                    && (j==8))
                    || ((i==7) && (j==1)) || ((i==8) && (j==2)) || ((i==1) && (j==7)) || ((i
                        ==2) && (j==8))){
                    grilleScore[i-1][j-1]=-150;
                }
                else {
                    if(((i==1) && (j==3)) || ((i==3) && (j==1)) || ((i==8) && (j==6)) || ((i
                        ==6) && (j==8))
                        || ((i==6) && (j==1)) || ((i==8) && (j==3)) || ((i==1) && (j==6)) || ((i
                            ==3) && (j==8))){
                        grilleScore[i-1][j-1]=30;
                    }
                    else {
                        if(((i==1) && (j==4)) || ((i==4) && (j==1)) || ((i==8) && (j==5)) || ((i
                            ==5) && (j==8))
                            || ((i==5) && (j==1)) || ((i==8) && (j==4)) || ((i==1) && (j==5)) || ((i
                                ==4) && (j==8))){
                                grilleScore[i-1][j-1]=10;
                            }
                        else {
                            if(((i==2) && (j==2)) || ((i==2) && (j==7)) || ((i==7) && (j==2)) || ((
                                i==7) && (j==7))){
                                grilleScore[i-1][j-1]=-250;
                            }
                            else {
                                if(((i==3) && (j==3)) || ((i==3) && (j==6)) || ((i==6) && (j==3)) ||
                                    ((i==6) && (j==6))){
                                    grilleScore[i-1][j-1]=1;
                                }
                                else {
                                    if(((i==4) && (j==4)) || ((i==4) && (j==5)) || ((i==5) && (j==4))
                                        || ((i==5) && (j==5))){
                                        grilleScore[i-1][j-1]=16;
                                    }
                                    else {
                                        if(((i==4) && (j==3)) || ((i==5) && (j==3)) || ((i==3) && (j==4))
                                            || ((i==3) && (j==5))
                                            || ((i==6) && (j==4)) || ((i==6) && (j==5)) || ((i==4) && (j==6))
                                                || ((i==5) && (j==6))){
                                                grilleScore[i-1][j-1]=2;
                                            }
                                        else {
                                            grilleScore[i-1][j-1]=0;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
} /*FinSi*/

```

```

        } /*FinSi*/
    } /*FinSi*/
} /*FinSi*/
} /*FinSi*/
} /*FinSi*/
} /*FinSi*/
} /*FinSi*/
} /*FinSi*/
} /*FinSi*/
} /*FinSi*/
return(grilleScore);
}

int min(int a, int b){
    if (a<b){
        return a;
    }
    else{
        return b;
    }
}

int max(int a, int b){
    if (a>b){
        return a;
    }
    else{
        return b;
    }
}

```

## 2.12 Le fichier « main.c »

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "FaireUnePartie.h"
#include "Affichage.h"
#include "ObtenirCoupIA.h"
#include "ObtenirCoupHumain.h"

int main(int argc, char** argv){
    Couleur vainqueur=CL_blanc();
    int matchNul = 0;

    if(argc!=3){
        afficherAide();
    }
    else
    {
        if (argc==3 && (strcmp(argv[1],"standard")==0)){
            if (strcmp(argv[2],"blanc")==0){
                faireUnePartie(afficherPlateau,obtenirCoupHumain,
                               obtenirCoupIA,&vainqueur,&matchNul,CL_blanc());
            }
            else{
                if (strcmp(argv[2],"noir")==0){
                    faireUnePartie(afficherPlateau,
                                   obtenirCoupHumain,obtenirCoupIA,&
                                   vainqueur,&matchNul,CL_noir());
                }
            }
        }
    }
}

```

```
        }
    }
}
else
{
    if (argc==3 && (strcmp(argv[1], "tournoi")==0)){
        if (strcmp(argv[2], "blanc")==0){
            faireUnePartie(afficherTournoi, obtenirCoupIA,
                obtenirCoupIA, &vainqueur, &matchNul,
                CL_blanc());
        }
        else{
            if (strcmp(argv[2], "noir")==0){
                faireUnePartie(afficherTournoi,
                    obtenirCoupIA, obtenirCoupIA, &
                    vainqueur, &matchNul, CL_blanc());
            }
        }
    }
}

return EXIT_SUCCESS;
}
```



# Chapitre 3

## Les fichiers de test

### 3.1 Le fichier « TestFaireUnePartie.c »

```
#include <stdlib.h>
#include <CUnit/Basic.h>
#define TRUE 1
#define FALSE 0
#include "TAD_Couleur.h"
#include "TAD_Couleur.h"
#include "TAD_Pion.h"
#include "TAD_Coup.h"
#include "TAD_Coups.h"
#include "TAD_Plateau.h"
#include "FaireUnePartie.h"
#include "FaireUnePartie.h"

int init_suite_success(void) {
    return 0;
}

int clean_suite_success(void) {
    return 0;
}

/* Tests relatifs à pionEstPresent */

void test_pionEstPresent(void) {
    int pionPresent;
    Plateau plateau;
    Direction dirTest = BAS;
    Pion pionJoueurCourant, pionATrouver, pionAdverse;
    Position posJoueurCourant, posATrouver, posAdverse;

    plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);
    pionJoueurCourant=PI_creerPion(CL_blanc());
    pionAdverse=PI_creerPion(CL_noir());
    pionATrouver=PI_creerPion(CL_blanc());
    pionPresent = 0;

    POS_fixerPosition(0,1,&posJoueurCourant);
    PL_poserPion(&plateau, posJoueurCourant, pionJoueurCourant);
    POS_fixerPosition(1,1,&posAdverse);
    PL_poserPion(&plateau, posAdverse, pionAdverse);
    POS_fixerPosition(2,1,&posATrouver);
```

```

    PL_poserPion(&plateau, posATrouver, pionATrouver);

    pionEstPresent(pionJoueurCourant, dirTest, &posJoueurCourant, &plateau, &pionPresent);

    CU_ASSERT_TRUE(pionPresent==TRUE);
}

/* Tests relatifs à inverserPionsDir */

void test_inverserPionsDir(void) {
    int res;
    Plateau plateau;
    Position positionTeste, positionChangee, positionArret;
    Pion pionBlanc=PI_creerPion(CL_blanc());
    Direction dirTest = BAS;

    plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);

    POS_fixerPosition(2,4,&positionTeste);
    PL_poserPion(&plateau, positionTeste, pionBlanc);

    POS_fixerPosition(3,4,&positionChangee);
    POS_fixerPosition(4,4,&positionArret);

    inverserPionsDir(&plateau, positionArret, DIR_positionSelonDirection(positionTeste,
        dirTest), dirTest);
    res = (CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau, positionChangee)),
        CL_blanc())
        && CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau, positionArret)),
        CL_blanc())
        && CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau, positionTeste)),
        CL_blanc()));

    CU_ASSERT_TRUE(res==TRUE);
}

/* Tests relatifs à inverserPions */

void test_inverserPions(void){
    Position positionPion;
    Position positionTest1, positionTest2, positionJoue;
    int res;
    /* Initialisation situation */
    Pion pionNoir=PI_creerPion(CL_noir());
    Pion pionBlanc=PI_creerPion(CL_blanc());
    Plateau plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);
    POS_fixerPosition(3,2,&positionPion);
    PL_poserPion(&plateau, positionPion, pionNoir);
    POS_fixerPosition(3,3,&positionPion);
    PL_poserPion(&plateau, positionPion, pionNoir);
    POS_fixerPosition(4,2,&positionPion);
    PL_poserPion(&plateau, positionPion, pionBlanc);
    POS_fixerPosition(4,3,&positionPion);
    PL_poserPion(&plateau, positionPion, pionBlanc);

    /* Test */
    POS_fixerPosition(2,2,&positionJoue);
    PL_poserPion(&plateau, positionJoue, pionBlanc);
    inverserPions(positionJoue, pionBlanc, &plateau);

```

```

    POS_fixerPosition(3,2,&positionTest1);
    POS_fixerPosition(3,3,&positionTest2);
    res = (CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau,positionTest1)),
        CL_blanc()))
        && (CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau,positionTest2)),
        CL_blanc()));

    CU_ASSERT_TRUE(res==TRUE);
}

/* Tests relatifs à pionEstPresent */

/* Tests relatifs à initialiserPlateau */

void test_initialiserPlateau(void){
    Plateau plateau;
    int res=TRUE;
    Position pos;
    unsigned int i,j,x,y;
    plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);

    for(i=1;i<9;i++){
        for(j=1;j<9;j++){
            x=i-1;
            y=j-1;
            POS_fixerPosition(x,y,&pos);
            if ((x==3 && y==3) || (x==4 && y==4)){
                if (PL_estCaseVide(plateau,pos)){
                    res=FALSE;
                }
                else if (!CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau,pos)),
                    CL_blanc())){
                    res=FALSE;
                }
            }
            else if ((x==3 && y==4) || (x==4 && y==3)){
                if (PL_estCaseVide(plateau,pos)){
                    res=FALSE;
                }
                else if (!CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau,pos)),CL_noir
                    ()))){
                    res=FALSE;
                }
            }
            else{
                if (!PL_estCaseVide(plateau,pos)){
                    res=FALSE;
                }
            }
        }
    }
    CU_ASSERT_TRUE(res==TRUE);
}

/* Tests relatifs à plateauRempli */
void test_plateauRempliVrai(void){
    unsigned int i,j,x,y;
    Position position;
    Pion pion=PI_creerPion(CL_noir());
    Plateau plateau;

```

```

    plateau=PL_creerPlateau();
    for(i=1;i<9;i++){
        for(j=1;j<9;j++){
            x=i-1;
            y=j-1;
            POS_fixerPosition(x, y, &position);
            PL_poserPion(&plateau,position,pion);
        }
    }
    CU_ASSERT_TRUE(plateauRempli(plateau)==TRUE);
}

void test_plateauRempliFaux(void){
    Plateau plateau;
    plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);

    CU_ASSERT_TRUE(plateauRempli(plateau)==FALSE);
}

/* Tests relatifs à nbPions */

void test_nbPionsPlateauRempli(void){
    unsigned int i,j;
    int nbPionsNoirs=0,nbPionsBlancs=0;
    Position position;
    Pion pion=PI_creerPion(CL_noir());
    Plateau plateau;
    PL_creerPlateau(&plateau);
    for(i=0;i<8;i++){
        for(j=0;j<8;j++){
            POS_fixerPosition(i, j, &position);
            PL_poserPion(&plateau,position,pion);
        }
    }
    nbPions(plateau,&nbPionsNoirs, &nbPionsBlancs);

    CU_ASSERT_TRUE((nbPionsBlancs==0) && (nbPionsNoirs==64));
}

void test_nbPionsPlateauInitial(void){
    Plateau plateau;
    int nbPionsNoirs=0,nbPionsBlancs=0;
    plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);
    nbPions(plateau,&nbPionsNoirs,&nbPionsBlancs);

    CU_ASSERT_TRUE((nbPionsBlancs==2) && (nbPionsNoirs==2));
}

/* Tests relatifs à finPartie */

void test_finPartieJoueursBloques(void){
    Plateau plateau=PL_creerPlateau();
    int aPuJouerJoueur1 = FALSE, aPuJouerJoueur2 = FALSE;
    int nbPionsNoirs = 0, nbPionsBlancs = 0;
    int estFinie = FALSE;
    finPartie(plateau,aPuJouerJoueur1,aPuJouerJoueur2,&nbPionsNoirs,&nbPionsBlancs,&
        estFinie);

    CU_ASSERT_TRUE(estFinie==TRUE);
}

```

```

}

void test_finPartieUnSeulJoueurBloque(void){
    Plateau plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);
    int aPuJouerJoueur1 = TRUE, aPuJouerJoueur2 = FALSE;
    int nbPionsNoirs = 4, nbPionsBlancs = 4;
    int estFinie = FALSE;
    finPartie(plateau,aPuJouerJoueur1,aPuJouerJoueur2,&nbPionsNoirs,&nbPionsBlancs,&
        estFinie);

    CU_ASSERT_TRUE(estFinie==FALSE);
}

void test_finPartiePlateauRempli(void){
    Plateau plateau=PL_creerPlateau();
    Pion pionBlanc=PI_creerPion(CL_blanc());
    Position pos;
    int aPuJouerJoueur1 = TRUE, aPuJouerJoueur2 = TRUE;
    int estFinie = FALSE;
    int nbPionsNoirs = 0, nbPionsBlancs = 0;
    unsigned int i,j;
    for(i=0;i<8;i++){
        for(j=0;j<8;j++){
            POS_fixerPosition(i,j,&pos);
            PL_poserPion(&plateau,pos,pionBlanc);
        }
    }
    finPartie(plateau,aPuJouerJoueur1,aPuJouerJoueur2,&nbPionsNoirs,&nbPionsBlancs,&
        estFinie);

    CU_ASSERT_TRUE(estFinie==TRUE);
}

/* Tests relatifs à jouerCoup */

void test_jouerCoup(void){
    Plateau plateau;
    Coup coup;
    Position position,positionPionRetourne;
    Pion pion;
    Couleur blanc=CL_blanc();
    plateau=PL_creerPlateau();
    initialiserPlateau(&plateau);
    pion=PI_creerPion(blanc);
    POS_fixerPosition(2,4,&position);
    POS_fixerPosition(3,4,&positionPionRetourne);
    coup=CP_creerCoup(position, pion);

    jouerCoup(coup,&plateau);

    CU_ASSERT_TRUE( !(PL_estCaseVide(plateau,position))
        && PI_sontEgaux(PL_obtenirPion(plateau,position),pion)
        && CL_sontEgales(PI_obtenirCouleur(PL_obtenirPion(plateau,
            positionPionRetourne)),CL_blanc()));
}

int main(int argc, char** argv){
    CU_pSuite pSuite_initialiserPlateau;
    CU_pSuite pSuite_plateauRempli;

```

```

CU_pSuite pSuite_nbPions;
CU_pSuite pSuite_finPartie;
CU_pSuite pSuite_jouerCoup;
CU_pSuite pSuite_inverserPionsDir;
CU_pSuite pSuite_inverserPions;
CU_pSuite pSuite_pionEstPresent;

/* initialisation du registre de tests */
if (CUE_SUCCESS != CU_initialize_registry())
    return CU_get_error();

/* ajout des suites de tests */
pSuite_initialiserPlateau = CU_add_suite("Tests boîte noire : initialiserPlateau",
    , init_suite_success, clean_suite_success);
pSuite_plateauRempli = CU_add_suite("Tests boîte noire : plateauRempli",
    init_suite_success, clean_suite_success);
pSuite_nbPions = CU_add_suite("Tests boîte noire : nbPions", init_suite_success,
    clean_suite_success);
pSuite_finPartie = CU_add_suite("Tests boîte noire : finPartie",
    init_suite_success, clean_suite_success);
pSuite_jouerCoup = CU_add_suite("Tests boîte noire : jouerCoup",
    init_suite_success, clean_suite_success);
pSuite_inverserPionsDir = CU_add_suite("Tests boîte noire : inverserPionsDir",
    init_suite_success, clean_suite_success);
pSuite_inverserPions = CU_add_suite("Tests boîte noire : inverserPions",
    init_suite_success, clean_suite_success);
pSuite_pionEstPresent = CU_add_suite("Tests boîte noire : pionEstPresent",
    init_suite_success, clean_suite_success);
if ((NULL == pSuite_initialiserPlateau)
    || (NULL == pSuite_plateauRempli)
    || (NULL == pSuite_nbPions)
    || (NULL == pSuite_finPartie)
    || (NULL == pSuite_jouerCoup)
    || (NULL == pSuite_inverserPionsDir)
    || (NULL == pSuite_inverserPions)
    || (NULL == pSuite_pionEstPresent)
){
    CU_cleanup_registry();
    return CU_get_error();
}

/* Ajout des tests à la suite de tests boîte noire */
if ((NULL == CU_add_test(pSuite_initialiserPlateau, "Plateau de départ",
    test_initialiserPlateau))
    || (NULL == CU_add_test(pSuite_plateauRempli, "Plateau réellement rempli",
    test_plateauRempliVrai))
    || (NULL == CU_add_test(pSuite_plateauRempli, "Plateau non rempli",
    test_plateauRempliFaux))
    || (NULL == CU_add_test(pSuite_nbPions, "Plateau rempli de pions noirs",
    test_nbPionsPlateauRempli))
    || (NULL == CU_add_test(pSuite_nbPions, "Plateau initial",
    test_nbPionsPlateauInitial))
    || (NULL == CU_add_test(pSuite_finPartie, "Joueurs bloqués",
    test_finPartieJoueursBloques))
    || (NULL == CU_add_test(pSuite_finPartie, "Plateau rempli",
    test_finPartiePlateauRempli))
    || (NULL == CU_add_test(pSuite_finPartie, "Un seul joueur bloqué",
    test_finPartieUnSeulJoueurBloque))
    || (NULL == CU_add_test(pSuite_jouerCoup, "Jouer un coup", test_jouerCoup))
    || (NULL == CU_add_test(pSuite_inverserPionsDir, "Inverser pions vers le bas",
    ,test_inverserPionsDir))

```

```

    || (NULL == CU_add_test(pSuite_inverserPions, "Inverser pions ",
        test_inverserPions))
    || (NULL == CU_add_test(pSuite_pionEstPresent, "pion trouvé ",
        test_pionEstPresent))
    /*|| (NULL == CU_add_test(pSuite_coupValide, "Coup valide, pos initiale dans
        un coin", test_coupValideCoin))
    || (NULL == CU_add_test(pSuite_coupValide, "Coup valide, pos initiale
        quelconque", test_coupValideQuelconque))
    || (NULL == CU_add_test(pSuite_listeCoupsPossibles, "Liste des coups
        possibles au début de jeu", test_listeCoupsPossibles))
    || (NULL == CU_add_test(pSuite_listeCoupsPossibles, "Liste de coups possibles
        vide", test_listeCoupsPossiblesPlateauVide))*//
    ){
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* Lancement des tests */
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    printf("\n");
    CU_basic_show_failures(CU_get_failure_list());
    printf("\n\n");

    /* Nettoyage du registre */
    CU_cleanup_registry();
    return CU_get_error();
}

```

### 3.2 Le fichier « TestListeCoupsPossibles.c »

```

#include <stdlib.h>
#include <CUnit/Basic.h>
#define TRUE 1
#define FALSE 0
#include "TAD_Couleur.h"
#include "TAD_Couleur.h"
#include "TAD_Pion.h"
#include "TAD_Coup.h"
#include "TAD_Coups.h"
#include "TAD_Plateau.h"
#include "ListeCoupsPossibles.h"
#include "ListeCoupsPossibles_Prive.h"
#include "FaireUnePartie.h"
#include "FaireUnePartie_Prive.h"
#include "ObtenirCoupIA.h"
#include "ObtenirCoupIA_Prive.h"

int init_suite_success(void) {
    return 0;
}

int clean_suite_success(void) {
    return 0;
}

/* Tests relatifs à copierPlateau */

void test_copierPlateauInterieur(void){

```

```

    Position positionInter1, positionInter2, positionInter3, positionInter4;
    Pion pionN=PI_creerPion(CL_noir());
    Pion pionB=PI_creerPion(CL_blanc());
    Plateau plateau1, plateau2;
    plateau1=PL_creerPlateau();
    plateau2=PL_creerPlateau();

    POS_fixerPosition(4,5,&positionInter1);
    POS_fixerPosition(4,4,&positionInter2);
    POS_fixerPosition(2,3,&positionInter3);
    POS_fixerPosition(5,6,&positionInter4);

    PL_poserPion(&plateau1, positionInter1, pionN);
    PL_poserPion(&plateau1, positionInter2, pionB);
    PL_poserPion(&plateau1, positionInter3, pionN);
    PL_poserPion(&plateau1, positionInter4, pionB);

    copierPlateau(plateau1, &plateau2);

    CU_ASSERT_TRUE(PI_sontEgaux(PL_obtenirPion(plateau2, positionInter1), pionN)
        && PI_sontEgaux(PL_obtenirPion(plateau2, positionInter3), pionN)
        && PI_sontEgaux(PL_obtenirPion(plateau2, positionInter2), pionB)
        && PI_sontEgaux(PL_obtenirPion(plateau2, positionInter4), pionB)
        && !PL_estCaseVide(plateau2, positionInter1)
        && !PL_estCaseVide(plateau2, positionInter2)
        && !PL_estCaseVide(plateau2, positionInter3)
        && !PL_estCaseVide(plateau2, positionInter4));
}

void test_copierPlateauBords(void){
    Position positionBord1, positionBord2, positionBord3, positionBord4;
    Pion pionN=PI_creerPion(CL_noir());
    Pion pionB=PI_creerPion(CL_blanc());
    Plateau plateau1, plateau2;
    plateau1=PL_creerPlateau();
    plateau2=PL_creerPlateau();

    POS_fixerPosition(0,5,&positionBord1);
    POS_fixerPosition(4,0,&positionBord2);
    POS_fixerPosition(7,4,&positionBord3);
    POS_fixerPosition(5,7,&positionBord4);

    PL_poserPion(&plateau1, positionBord1, pionN);
    PL_poserPion(&plateau1, positionBord2, pionB);
    PL_poserPion(&plateau1, positionBord3, pionN);
    PL_poserPion(&plateau1, positionBord4, pionB);

    copierPlateau(plateau1, &plateau2);

    CU_ASSERT_TRUE(PI_sontEgaux(PL_obtenirPion(plateau2, positionBord1), pionN)
        && PI_sontEgaux(PL_obtenirPion(plateau2, positionBord3), pionN)
        && PI_sontEgaux(PL_obtenirPion(plateau2, positionBord2), pionB)
        && PI_sontEgaux(PL_obtenirPion(plateau2, positionBord4), pionB)
        && !PL_estCaseVide(plateau2, positionBord1)
        && !PL_estCaseVide(plateau2, positionBord2)
        && !PL_estCaseVide(plateau2, positionBord3)
        && !PL_estCaseVide(plateau2, positionBord4));
}

void test_copierPlateauCoins(void){
    Position positionCoin1, positionCoin2, positionCoin3, positionCoin4;

```



```

Pion pionN=PI_creerPion(CL_noir());
Pion pionB=PI_creerPion(CL_blanc());
Plateau plateau1, plateau2;
plateau1=PL_creerPlateau();
plateau2=PL_creerPlateau();

POS_fixerPosition(0,0,&positionCoin1);
POS_fixerPosition(0,7,&positionCoin2);
POS_fixerPosition(7,0,&positionCoin3);
POS_fixerPosition(7,7,&positionCoin4);

PL_poserPion(&plateau1, positionCoin1, pionN);
PL_poserPion(&plateau1, positionCoin2, pionB);
PL_poserPion(&plateau1, positionCoin3, pionN);
PL_poserPion(&plateau1, positionCoin4, pionB);

copierPlateau(plateau1, &plateau2);

CU_ASSERT_TRUE(PI_sontEgaux(PL_obtenirPion(plateau2, positionCoin1), pionN)
    && PI_sontEgaux(PL_obtenirPion(plateau2, positionCoin3), pionN)
    && PI_sontEgaux(PL_obtenirPion(plateau2, positionCoin2), pionB)
    && PI_sontEgaux(PL_obtenirPion(plateau2, positionCoin4), pionB)
    && !PL_estCaseVide(plateau2, positionCoin1)
    && !PL_estCaseVide(plateau2, positionCoin2)
    && !PL_estCaseVide(plateau2, positionCoin3)
    && !PL_estCaseVide(plateau2, positionCoin4));
}

/* Tests relatifs à coupValide */

void test_coupValideEntoureCasesVides(void){
    Plateau plateau;
    Coup coup;
    Position positionTest;
    Pion pionTest=PI_creerPion(CL_noir());

    plateau=PL_creerPlateau();
    POS_fixerPosition(4,5,&positionTest);
    PL_poserPion(&plateau, positionTest, pionTest);
    coup=CP_creerCoup(positionTest, pionTest);
    CU_ASSERT_TRUE(!coupValide(plateau, coup));
}

void test_coupValideEntoureCasesMemeCouleur(void){
    Plateau plateau;
    Coup coup;
    Position positionTest, positionAutour1, positionAutour2, positionAutour3,
        positionAutour4, positionAutour5, positionAutour6, positionAutour7, positionAutour8
        ;
    Pion pionTest=PI_creerPion(CL_blanc()), pionAutour1=PI_creerPion(CL_blanc()),
        pionAutour2=PI_creerPion(CL_blanc()), pionAutour3=PI_creerPion(CL_blanc()),
        pionAutour4=PI_creerPion(CL_blanc()), pionAutour5=PI_creerPion(CL_blanc()),
        pionAutour6=PI_creerPion(CL_blanc()), pionAutour7=PI_creerPion(CL_blanc()),
        pionAutour8=PI_creerPion(CL_blanc());

    plateau=PL_creerPlateau();
    POS_fixerPosition(4,5,&positionTest);
    POS_fixerPosition(3,4,&positionAutour1);
    POS_fixerPosition(4,4,&positionAutour2);
    POS_fixerPosition(5,4,&positionAutour3);
    POS_fixerPosition(3,5,&positionAutour4);

```

```

    POS_fixerPosition(5,5,&positionAutour5);
    POS_fixerPosition(3,6,&positionAutour6);
    POS_fixerPosition(4,6,&positionAutour7);
    POS_fixerPosition(5,6,&positionAutour8);

    PL_poserPion(&plateau, positionTest, pionTest);
    PL_poserPion(&plateau, positionAutour1, pionAutour1);
    PL_poserPion(&plateau, positionAutour2, pionAutour2);
    PL_poserPion(&plateau, positionAutour3, pionAutour3);
    PL_poserPion(&plateau, positionAutour4, pionAutour4);
    PL_poserPion(&plateau, positionAutour5, pionAutour5);
    PL_poserPion(&plateau, positionAutour6, pionAutour6);
    PL_poserPion(&plateau, positionAutour7, pionAutour7);
    PL_poserPion(&plateau, positionAutour8, pionAutour8);

    coup=CP_creerCoup(positionTest, pionTest);
    CU_ASSERT_TRUE(! coupValide(plateau, coup));
}

void test_coupValideQueCasesAutreCouleurPuisVide(void){
    Plateau plateau;
    Coup coup;
    Position positionTest, positionAutour1, positionAutour2, positionAutour3,
        positionAutour4, positionAutour5, positionAutour6, positionAutour7, positionAutour8
        ;
    Pion pionTest=PI_creerPion(CL_noir()), pionAutour1=PI_creerPion(CL_blanc()),
        pionAutour2=PI_creerPion(CL_blanc()),
        pionAutour3=PI_creerPion(CL_blanc()), pionAutour4=PI_creerPion(CL_blanc()),
        pionAutour5=PI_creerPion(CL_blanc()),
        pionAutour6=PI_creerPion(CL_blanc()), pionAutour7=PI_creerPion(CL_blanc()),
        pionAutour8=PI_creerPion(CL_blanc());

    plateau=PL_creerPlateau();
    POS_fixerPosition(4,5,&positionTest);
    POS_fixerPosition(3,4,&positionAutour1);
    POS_fixerPosition(4,4,&positionAutour2);
    POS_fixerPosition(5,4,&positionAutour3);
    POS_fixerPosition(3,5,&positionAutour4);
    POS_fixerPosition(5,5,&positionAutour5);
    POS_fixerPosition(3,6,&positionAutour6);
    POS_fixerPosition(4,6,&positionAutour7);
    POS_fixerPosition(5,6,&positionAutour8);

    PL_poserPion(&plateau, positionTest, pionTest);
    PL_poserPion(&plateau, positionAutour1, pionAutour1);
    PL_poserPion(&plateau, positionAutour2, pionAutour2);
    PL_poserPion(&plateau, positionAutour3, pionAutour3);
    PL_poserPion(&plateau, positionAutour4, pionAutour4);
    PL_poserPion(&plateau, positionAutour5, pionAutour5);
    PL_poserPion(&plateau, positionAutour6, pionAutour6);
    PL_poserPion(&plateau, positionAutour7, pionAutour7);
    PL_poserPion(&plateau, positionAutour8, pionAutour8);

    coup=CP_creerCoup(positionTest, pionTest);
    CU_ASSERT_TRUE(! coupValide(plateau, coup));
}

void test_coupValideCoin(void){
    Plateau plateau;
    Coup coup;

```

```

    Position positionTest, positionAutreCouleur1, positionAutreCouleur2,
        positionMemeCouleur;
    Pion pionTest=PI_creerPion(CL_noir()), pionAutreCouleur1=PI_creerPion(CL_blanc()),
        pionAutreCouleur2=PI_creerPion(CL_blanc()),
        pionMemeCouleur=PI_creerPion(CL_noir());

    plateau=PL_creerPlateau();
    POS_fixerPosition(0,0,&positionTest);
    POS_fixerPosition(1,0,&positionAutreCouleur1);
    POS_fixerPosition(2,0,&positionAutreCouleur2);
    POS_fixerPosition(3,0,&positionMemeCouleur);

    PL_poserPion(&plateau, positionAutreCouleur1, pionAutreCouleur1);
    PL_poserPion(&plateau, positionAutreCouleur2, pionAutreCouleur2);
    PL_poserPion(&plateau, positionMemeCouleur, pionMemeCouleur);

    coup=CP_creerCoup(positionTest, pionTest);
    CU_ASSERT_TRUE(coupValide(plateau, coup));
}

void test_coupValideQuelconque(void){
    Plateau plateau;
    Coup coup;
    Position positionTest, positionAutreCouleur1, positionAutreCouleur2,
        positionMemeCouleur;
    Pion pionTest=PI_creerPion(CL_noir()), pionAutreCouleur1=PI_creerPion(CL_blanc()),
        pionAutreCouleur2=PI_creerPion(CL_blanc()),
        pionMemeCouleur=PI_creerPion(CL_noir());

    plateau=PL_creerPlateau();
    POS_fixerPosition(4,3,&positionTest);
    POS_fixerPosition(4,4,&positionAutreCouleur1);
    POS_fixerPosition(4,5,&positionAutreCouleur2);
    POS_fixerPosition(4,6,&positionMemeCouleur);

    PL_poserPion(&plateau, positionAutreCouleur1, pionAutreCouleur1);
    PL_poserPion(&plateau, positionAutreCouleur2, pionAutreCouleur2);
    PL_poserPion(&plateau, positionMemeCouleur, pionMemeCouleur);

    coup=CP_creerCoup(positionTest, pionTest);
    CU_ASSERT_TRUE(coupValide(plateau, coup));
}

/* Tests relatifs à listeCoupsPossibles */

void test_listeCoupsPossibles(void){
    Plateau plateau;
    Coup coup1, coup2, coup3, coup4;
    Coups listeCoups;
    Couleur couleurJoueur=CL_blanc();
    Position position1, position2, position3, position4, positionCoup1, positionCoup2,
        positionCoup3, positionCoup4;
    Pion pion1=PI_creerPion(CL_blanc()), pion2=PI_creerPion(CL_noir()), pion3=
        PI_creerPion(CL_noir()), pion4=PI_creerPion(CL_blanc()); /* Configuration
        initiale du plateau */
    Pion pionCoup1=PI_creerPion(CL_blanc()), pionCoup2=PI_creerPion(CL_blanc()),
        pionCoup3=PI_creerPion(CL_blanc()), pionCoup4=PI_creerPion(CL_blanc());

    plateau=PL_creerPlateau();
    CPS_creerCoups(&listeCoups);

```

```

    POS_fixerPosition(3,3,&position1);
    POS_fixerPosition(3,4,&position2);
    POS_fixerPosition(4,3,&position3);
    POS_fixerPosition(4,4,&position4);
    POS_fixerPosition(2,4,&positionCoup1);
    POS_fixerPosition(3,5,&positionCoup2);
    POS_fixerPosition(4,2,&positionCoup3);
    POS_fixerPosition(5,3,&positionCoup4);

    PL_poserPion(&plateau, position1, pion1);
    PL_poserPion(&plateau, position2, pion2);
    PL_poserPion(&plateau, position3, pion3);
    PL_poserPion(&plateau, position4, pion4);

    coup1=CP_creerCoup(positionCoup1, pionCoup1);
    coup2=CP_creerCoup(positionCoup2, pionCoup2);
    coup3=CP_creerCoup(positionCoup3, pionCoup3);
    coup4=CP_creerCoup(positionCoup4, pionCoup4);

    listeCoups=listeCoupsPossibles(plateau, couleurJoueur);

    CU_ASSERT_TRUE(CP_sontEgaux(CPS_iemeCoup(listeCoups, 0), coup1)
        && CP_sontEgaux(CPS_iemeCoup(listeCoups, 1), coup2)
        && CP_sontEgaux(CPS_iemeCoup(listeCoups, 2), coup3)
        && CP_sontEgaux(CPS_iemeCoup(listeCoups, 3), coup4));
}

void test_listeCoupsPossiblesPlateauVide(void){
    Plateau plateau;
    Coups listeCoups;
    Couleur couleurJoueur=CL_blanc();

    plateau=PL_creerPlateau();
    CPS_creerCoups(&listeCoups);

    listeCoups=listeCoupsPossibles(plateau, couleurJoueur);
    CU_ASSERT_TRUE(CPS_nbCoups(listeCoups)==0);
}

/* Tests relatifs à ObtenirCoupIA */

void test_ObttenirCoupIA(void)
{
    Plateau plateau=PL_creerPlateau();
    Couleur couleurJoueur=CL_noir();
    Coup Meilleurecoup, meilleurCoupIA;
    Position positionPionBlanc, positionMeilleurecoup;
    Pion pionBlanc=PI_creerPion(CL_blanc()), pionMeilleurecoup=PI_creerPion(CL_noir());

    initialiserPlateau(&plateau);
    POS_fixerPosition(2,4,&positionPionBlanc);
    PL_poserPion(&plateau, positionPionBlanc, pionBlanc);
    POS_fixerPosition(2,3,&positionMeilleurecoup);

    Meilleurecoup=CP_creerCoup(positionMeilleurecoup, pionMeilleurecoup);

    meilleurCoupIA=obtenirCoupIA(plateau, couleurJoueur);

```

```

    CU_ASSERT_TRUE(CP_sontEgaux(Meilleurcoup,meilleurCoupIA));
}
/* Tests relatifs à scoreDUnCoup */

void test_scoreDUnCoup(void)
{
    Plateau plateau=PL_creerPlateau();
    Couleur couleurJoueur=CL_noir();
    unsigned int profondeurMinMax=profondeur();
    Coup coup1, Meilleurcoup;
    int scoreCourant, meilleurScore;
    Position positionPionBlanc, positionCoup1, positionMeilleurcoup;
    Pion pionCoup1=PI_creerPion(CL_noir()), pionMeilleurcoup=PI_creerPion(CL_noir()),
        pionBlanc=PI_creerPion(CL_blanc());
    int** grilleScore=initialiserGrilleScore();

    /* Configuration initiale du plateau */

    initialiserPlateau(&plateau);
    POS_fixerPosition(2,4,&positionPionBlanc);
    PL_poserPion(&plateau, positionPionBlanc, pionBlanc);
    POS_fixerPosition(1,4,&positionCoup1);
    POS_fixerPosition(3,2,&positionMeilleurcoup);

    coup1=CP_creerCoup(positionCoup1,pionCoup1);
    Meilleurcoup=CP_creerCoup(positionMeilleurcoup,pionMeilleurcoup);

    meilleurScore = scoreDUnCoup(plateau,Meilleurcoup,couleurJoueur,couleurJoueur,
        profondeurMinMax,grilleScore);
    scoreCourant = scoreDUnCoup(plateau,coup1,couleurJoueur,couleurJoueur,
        profondeurMinMax,grilleScore);

    CU_ASSERT_TRUE    ((coupValide(plateau, coup1))
                        && (coupValide(plateau, Meilleurcoup))
                        && (meilleurScore>scoreCourant));
}

/* Tests relatifs à evaluerPlateau */

void test_evaluerNbCoupsPossiblesAdversaire(void){
    Plateau plateau=PL_creerPlateau();
    Couleur couleurJoueur=CL_blanc(), couleurAdversaire;
    int NbCoupsAdversaire,resultat;
    Position position1,position2,position3,position4;
    Pion pion1=PI_creerPion(CL_blanc()), pion2=PI_creerPion(CL_noir()), pion3=
        PI_creerPion(CL_noir()), pion4=PI_creerPion(CL_blanc());

    POS_fixerPosition(3,3,&position1);
    POS_fixerPosition(3,4,&position2);
    POS_fixerPosition(4,3,&position3);
    POS_fixerPosition(4,4,&position4);

    PL_poserPion(&plateau,position1,pion1);
    PL_poserPion(&plateau,position2,pion2);
    PL_poserPion(&plateau,position3,pion3);
    PL_poserPion(&plateau,position4,pion4);

    couleurAdversaire=CL_changerCouleur(couleurJoueur);
    NbCoupsAdversaire=4;
}

```

```

    resultat=(60-10*NbCoupsAdversaire);

    CU_ASSERT_TRUE(evaluerNbCoupsPossiblesAdversaire(plateau,couleurAdversaire) ==
        resultat);
}

void test_evaluerNbPionsCouleur(void){
    Plateau plateau=PL_creerPlateau();
    Couleur couleurJoueur=CL_blanc();
    int nbPionsNoirs,nbPionsBlancs;
    Position position1,position2,position3,position4,positionCoup1,positionCoup2;
    Pion pion1=PI_creerPion(CL_blanc()), pion2=PI_creerPion(CL_noir()), pion3=
        PI_creerPion(CL_noir()),
        pion4=PI_creerPion(CL_blanc()),pionCoup1=PI_creerPion(CL_blanc()),pionCoup2
            =PI_creerPion(CL_blanc());

    POS_fixerPosition(3,3,&position1);
    POS_fixerPosition(3,4,&position2);
    POS_fixerPosition(4,3,&position3);
    POS_fixerPosition(4,4,&position4);
    POS_fixerPosition(3,2,&positionCoup1);
    POS_fixerPosition(4,2,&positionCoup2);

    PL_poserPion(&plateau,position1,pion1);
    PL_poserPion(&plateau,position2,pion2);
    PL_poserPion(&plateau,position3,pion3);
    PL_poserPion(&plateau,position4,pion4);
    PL_poserPion(&plateau,positionCoup1,pionCoup1);
    PL_poserPion(&plateau,positionCoup2,pionCoup2);

    nbPions(plateau,&nbPionsNoirs,&nbPionsBlancs);

    CU_ASSERT_TRUE (evaluerNbPionsCouleur(plateau,couleurJoueur)==2);
}

void test_evaluerPositionsPionsPlateau(void){
    int** grilleScore=initialiserGrilleScore();
    Plateau plateau=PL_creerPlateau();
    Couleur couleurJoueur=CL_blanc();
    Position position1,position2,position3,position4,positionCoup1,positionCoup2;
    Pion pion1=PI_creerPion(CL_blanc()), pion2=PI_creerPion(CL_noir()), pion3=
        PI_creerPion(CL_noir()),
        pion4=PI_creerPion(CL_blanc()),pionCoup1=PI_creerPion(CL_blanc()),pionCoup2
            =PI_creerPion(CL_blanc());
    int resJoueurTEST,resAdversaireTEST;

    resJoueurTEST=0;
    resAdversaireTEST=0;

    POS_fixerPosition(3,3,&position1);
    POS_fixerPosition(3,4,&position2);
    POS_fixerPosition(4,3,&position3);
    POS_fixerPosition(4,4,&position4);
    POS_fixerPosition(3,2,&positionCoup1);
    POS_fixerPosition(4,2,&positionCoup2);

    PL_poserPion(&plateau,position1,pion1);
    PL_poserPion(&plateau,position2,pion2);
    PL_poserPion(&plateau,position3,pion3);

```

```

    PL_poserPion(&plateau, position4, pion4);
    PL_poserPion(&plateau, positionCoup1, pionCoup1);
    PL_poserPion(&plateau, positionCoup2, pionCoup2);

    resJoueurTEST=grilleScore[3][3]+grilleScore[4][4]+grilleScore[3][2]+grilleScore
        [4][2];
    resAdversaireTEST=grilleScore[3][4]+grilleScore[4][3];

    CU_ASSERT_TRUE ( evaluerPositionsPionsPlateau(plateau, couleurJoueur, grilleScore)
        ==(resJoueurTEST-resAdversaireTEST));
}

int main(int argc, char** argv){
    CU_pSuite pSuite_copierPlateau;
    CU_pSuite pSuite_coupValide;
    CU_pSuite pSuite_listeCoupsPossibles;
    CU_pSuite pSuite_ObtenirCoupIA;
    CU_pSuite pSuite_scoreDUnCoup;
    CU_pSuite pSuite_evaluerPlateau;

    /* initialisation du registre de tests */
    if (CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();

    /* ajout des suites de tests */
    pSuite_copierPlateau = CU_add_suite("Tests boite noire : copierPlateau",
        init_suite_success, clean_suite_success);
    pSuite_coupValide = CU_add_suite("Tests boite noire : coupValide",
        init_suite_success, clean_suite_success);
    pSuite_listeCoupsPossibles = CU_add_suite("Tests boite noire :
        listeCoupsPossibles", init_suite_success, clean_suite_success);
    pSuite_scoreDUnCoup = CU_add_suite("Tests boite noire : scoreDUnCoup",
        init_suite_success, clean_suite_success);
    pSuite_ObtenirCoupIA = CU_add_suite("Tests boite noire : ObtenirCoupIA",
        init_suite_success, clean_suite_success);
    pSuite_evaluerPlateau = CU_add_suite("Tests boite noire : evaluerPlateau",
        init_suite_success, clean_suite_success);
    if ((NULL == pSuite_copierPlateau)
        || (NULL == pSuite_coupValide)
        || (NULL == pSuite_listeCoupsPossibles)
        || (NULL == pSuite_scoreDUnCoup)
        || (NULL == pSuite_ObtenirCoupIA)
        || (NULL == pSuite_evaluerPlateau)
    ){
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* Ajout des tests à la suite de tests boite noire */
    if ((NULL == CU_add_test(pSuite_copierPlateau, "Pions à l'intérieur",
        test_copierPlateauInterieur))
        || (NULL == CU_add_test(pSuite_copierPlateau, "Pions sur les bords",
        test_copierPlateauBords))
        || (NULL == CU_add_test(pSuite_copierPlateau, "Pions dans les coins",
        test_copierPlateauCoins))
        || (NULL == CU_add_test(pSuite_coupValide, "Pion entouré de cases vides",
        test_coupValideEntoureCasesVides))
    )

```

```

    || (NULL == CU_add_test(pSuite_coupValide, "Pion entouré de cases de même
        couleur", test_coupValideEntoureCasesMemeCouleur))
    || (NULL == CU_add_test(pSuite_coupValide, "Pion entouré de cases de l'autre
        couleur mais vides après", test_coupValideQueCasesAutreCouleurPuisVide))
    || (NULL == CU_add_test(pSuite_coupValide, "Coup valide, pos initiale dans un
        coin", test_coupValideCoin))
    || (NULL == CU_add_test(pSuite_coupValide, "Coup valide, pos initiale
        quelconque", test_coupValideQuelconque))
    || (NULL == CU_add_test(pSuite_listeCoupsPossibles, "Liste des coups
        possibles au début de jeu", test_listeCoupsPossibles))
    || (NULL == CU_add_test(pSuite_listeCoupsPossibles, "Liste de coups possibles
        vide", test_listeCoupsPossiblesPlateauVide))
    || (NULL == CU_add_test(pSuite_scoreDUnCoup, "calcul le bon score du coup",
        test_scoreDUnCoup))
    || (NULL == CU_add_test(pSuite_ObtenirCoupIA, "Renvoie le bon coup",
        test_ObtenirCoupIA))
    || (NULL == CU_add_test(pSuite_evaluerPlateau, "evaluer le nombre de coups
        possibles de l'adversaire", test_evaluerNbCoupsPossiblesAdversaire))
    || (NULL == CU_add_test(pSuite_evaluerPlateau, "evaluer le nombre de pions de
        la meme couleur", test_evaluerNbPionsCouleur))
    || (NULL == CU_add_test(pSuite_evaluerPlateau, "evaluer la position des pions
        sur le plateau", test_evaluerPositionsPionsPlateau))
    ){
    CU_cleanup_registry();
    return CU_get_error();
}

/* Lancement des tests */
CU_basic_set_mode(CU_BRM_VERBOSE);
CU_basic_run_tests();
printf("\n");
CU_basic_show_failures(CU_get_failure_list());
printf("\n\n");

/* Nettoyage du registre */
CU_cleanup_registry();
return CU_get_error();
}

```

### 3.3 Le fichier « TestTADs.c »

```

#include <stdlib.h>
#include <CUnit/Basic.h>
#define TRUE 1
#define FALSE 0
#include "TAD_Couleur.h"
#include "TAD_Couleur.h"
#include "TAD_Pion.h"
#include "TAD_Coup.h"
#include "TAD_Coups.h"
#include "TAD_Plateau.h"

int init_suite_success(void) {
    return 0;
}

int clean_suite_success(void) {
    return 0;
}

```



```

}

//////////* On vérifie les axiomes des TADs *//////////

/* Tests relatifs au TAD Couleur */
void test_changerCouleur(void){
    CU_ASSERT_TRUE((CL_sontEgales(CL_changerCouleur(CL_noir()),CL_blanc()))
        && (CL_sontEgales(CL_changerCouleur(CL_blanc()),CL_noir())));
}

/* Tests relatifs au TAD Position */
void test_obtenirLigne(void){
    Position positionTest;
    unsigned int l,c;
    l = 3;
    c = 5;
    POS_fixerPosition(l,c,&positionTest);
    // la position (l,c) respectera les dimensions d'un plateau d'Othello
    CU_ASSERT_TRUE((POS_obtenirLigne(positionTest)==l));
}

void test_obtenirColonne(void){
    Position positionTest;
    unsigned int l,c;
    l = 3;
    c = 5;
    POS_fixerPosition(l,c,&positionTest);
    // la position (l,c) respectera les dimensions d'un plateau d'Othello
    CU_ASSERT_TRUE((POS_obtenirColonne(positionTest)==c));
}

/* Tests relatifs au TAD Pion */
void test_obtenirCouleur(void){
    CU_ASSERT_TRUE((CL_sontEgales(PI_obtenirCouleur(PI_creerPion(CL_blanc()))),
        CL_blanc())
        && (CL_sontEgales(PI_obtenirCouleur(PI_creerPion(CL_noir()))),CL_noir
            ()))));
}

void test_retournerPion(void){ // Axiome 2 reformulé
    Couleur couleurAvant, couleurApres;
    Pion pionTest = PI_creerPion(CL_blanc());
    couleurAvant = PI_obtenirCouleur(pionTest);
    PI_retournerPion(&pionTest);
    couleurApres = PI_obtenirCouleur(pionTest);
    CU_ASSERT_TRUE(!CL_sontEgales(couleurAvant,couleurApres));
}

/* Tests relatifs au TAD Coup */
void test_obtenirPositionCoup(void){
    Position positionTest;
    POS_fixerPosition(3,5,&positionTest);
    Pion pionTest = PI_creerPion(CL_blanc());
    CU_ASSERT_TRUE((POS_sontEgales(CP_obtenirPositionCoup(CP_creerCoup(positionTest,
        pionTest)),positionTest));
}

```

```

void test_obtenirPionCoup(void){
    Position positionTest;
    Pion pionTest;
    pionTest = PI_creerPion(CL_blanc());
    CU_ASSERT_TRUE(PI_sontEgaux(CP_obtenirPionCoup(CP_creerCoup(positionTest, pionTest)), pionTest));
}

//

/* Tests relatifs au TAD Coups */
void test_iemeCoup(void){
    Position positionTest;
    Coup cp;
    Coups coupsTest;
    POS_fixerPosition(0,0,&positionTest);
    cp = CP_creerCoup(positionTest, PI_creerPion(CL_blanc()));
    CPS_creerCoups(&coupsTest);
    CPS_ajouterCoups(&coupsTest, cp);
    CU_ASSERT_TRUE(CP_sontEgaux(CPS_iemeCoup(coupsTest, CPS_nbCoups(coupsTest)-1), cp));
}

void test_nbCoups(void){    // les 2 derniers axiomes du TAD Coups
    Coup coupTest;
    Coups coupsVide, coupsAjout;
    CPS_creerCoups(&coupsVide);
    CPS_creerCoups(&coupsAjout);
    CPS_ajouterCoups(&coupsAjout, coupTest);
    CU_ASSERT_TRUE((CPS_nbCoups(coupsVide)==0)
        && (CPS_nbCoups(coupsAjout)==1));
}

/* Tests relatifs au TAD Plateau */
void test_estCaseVide(void){
    Plateau plateauTest;
    Position positionCaseVide, positionCaseAVider, positionCaseNonVide;
    plateauTest=PL_creerPlateau();
    Pion pionTest=PI_creerPion(CL_blanc());
    POS_fixerPosition(0,0,&positionCaseVide);
    POS_fixerPosition(0,1,&positionCaseAVider);
    POS_fixerPosition(1,0,&positionCaseNonVide);
    PL_poserPion(&plateauTest, positionCaseAVider, pionTest);
    PL_viderCase(&plateauTest, positionCaseAVider);
    PL_poserPion(&plateauTest, positionCaseNonVide, pionTest);
    CU_ASSERT_TRUE( (PL_estCaseVide(plateauTest, positionCaseVide))    // Axiome 1
        && (PL_estCaseVide(plateauTest, positionCaseAVider)) // Axiome 2
        && !(PL_estCaseVide(plateauTest, positionCaseNonVide))); // Axiome 3
}

void test_obtenirPion(void){
    Plateau plateauTest;
    Pion pionTest;
    Position positionTest;
    POS_fixerPosition(0,0,&positionTest);
    pionTest = PI_creerPion(CL_noir());
    PL_poserPion(&plateauTest, positionTest, pionTest);
    CU_ASSERT_TRUE(PI_sontEgaux(PL_obtenirPion(plateauTest, positionTest), pionTest));
}

void test_inverserPion(void){

```

```

Plateau plateauTest;
Pion pionTest;
Position positionTest;
POS_fixerPosition(0,0,&positionTest);
pionTest = PI_creerPion(CL_blanc());
PL_poserPion(&plateauTest,positionTest,pionTest);
PL_inverserPion(&plateauTest,positionTest);
PI_retournerPion(&pionTest);
CU_ASSERT_TRUE(PI_sontEgaux(PL_obtenirPion(plateauTest,positionTest),pionTest));
}

int main(int argc, char** argv){
    CU_pSuite pSuite_Couleur;
    CU_pSuite pSuite_Position;
    CU_pSuite pSuite_Pion;
    CU_pSuite pSuite_Coup;
    CU_pSuite pSuite_Coups;
    CU_pSuite pSuite_Plateau;

    /* initialisation du registre de tests */
    if (CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();

    /* ajout des suites de tests */
    pSuite_Couleur = CU_add_suite("Tests boîte noire : TAD Couleur",
        init_suite_success, clean_suite_success);
    pSuite_Position = CU_add_suite("Tests boîte noire : TAD pSuite_Position",
        init_suite_success, clean_suite_success);
    pSuite_Pion = CU_add_suite("Tests boîte noire : TAD Pion", init_suite_success,
        clean_suite_success);
    pSuite_Coup = CU_add_suite("Tests boîte noire : TAD Coup", init_suite_success,
        clean_suite_success);
    pSuite_Coups = CU_add_suite("Tests boîte noire : TAD Coups", init_suite_success,
        clean_suite_success);
    pSuite_Plateau = CU_add_suite("Tests boîte noire : TAD Plateau",
        init_suite_success, clean_suite_success);
    if ((NULL == pSuite_Couleur)
        || (NULL == pSuite_Position)
        || (NULL == pSuite_Coup)
        || (NULL == pSuite_Coups)
        || (NULL == pSuite_Pion)
        || (NULL == pSuite_Plateau)
    ){
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* Ajout des tests à la suite de tests boîte noire */
    if ((NULL == CU_add_test(pSuite_Couleur, "CL_changerCouleur", test_changerCouleur
    ))
        || (NULL == CU_add_test(pSuite_Position, "POS_obtenirLigne",
            test_obtenirLigne))
        || (NULL == CU_add_test(pSuite_Position, "POS_obtenirColonne",
            test_obtenirColonne))
        || (NULL == CU_add_test(pSuite_Pion, "PI_obtenirCouleur", test_obtenirCouleur
        ))
        || (NULL == CU_add_test(pSuite_Pion, "PI_retournerPion", test_retournerPion))
        || (NULL == CU_add_test(pSuite_Coup, "CP_obtenirPositionCoup",
            test_obtenirPositionCoup))
    )

```

```
    || (NULL == CU_add_test(pSuite_Coup, "CP_obtenirPionCoup",
        test_obtenirPionCoup))
    || (NULL == CU_add_test(pSuite_Coups, "CPS_iemeCoup", test_iemeCoup))
    || (NULL == CU_add_test(pSuite_Coups, "CPS_nbCoups", test_nbCoups))
    || (NULL == CU_add_test(pSuite_Plateau, "PL_estCaseVide", test_estCaseVide))
    || (NULL == CU_add_test(pSuite_Plateau, "PL_obtenirPion", test_obtenirPion))
    || (NULL == CU_add_test(pSuite_Plateau, "PL_inverserPion", test_inverserPion)
    )
){
    CU_cleanup_registry();
    return CU_get_error();
}

/* Lancement des tests */
CU_basic_set_mode(CU_BRM_VERBOSE);
CU_basic_run_tests();
printf("\n");
CU_basic_show_failures(CU_get_failure_list());
printf("\n\n");

/* Nettoyage du registre */
CU_cleanup_registry();
return CU_get_error();
}
```

Cinquième partie

Répartition du travail

# Chapitre 1

## Analyse descendante

Responsables		Claire	Riadh	Sandratra	Gautier	Romain
Sous-programme						
faireUnePartie						
initialiserPlateau						
jouer						
finPartie						
plateauRempli						
nbPions						
jouerCoup						
inverserPions						
inverserPionsDir						
pionEstPresent						
pionEstPresentRecuratif						
obtenirCoupIA						
profondeur						
listeCoupsPossibles						
coupValide						
copierPlateau						
minMax						
scoreDUnCoup						
score						
evaluerPlateau						

TABLE 1.1 – Répartition des tâches dans la phase d'analyse descendante

## Chapitre 2

# Conception préliminaire

Responsables	Claire	Riadh	Sandratra	Gautier	Romain
<b>Sous-programme</b>					
faireUnePartie					
initialiserPlateau					
jouer					
finPartie					
plateauRempli					
nbPions					
jouerCoup					
inverserPions					
inverserPionsDir					
pionEstPresent					
pionEstPresentRecuratif					
obtenirCoupIA					
profondeur					
listeCoupsPossibles					
coupValide					
copierPlateau					
minMax					
scoreDUnCoup					
score					
evaluerPlateau					
Type afficherPlateau					
Type getCoup					

TABLE 2.1 – Répartition des tâches dans la phase de conception préliminaire

## Chapitre 3

# Conception détaillée

		Gautier	Riadh	Romain	Claire	Sandratra
TAD	TAD « Couleur »					
	TAD « Pion »					
	TAD « Position »					
	TAD « Plateau »					
	TAD « Coup »					
	TAD « Coups »					
faireUnePartie	« faireUnePartie »					
	« jouer »					
	« jouerCoup »					
	« inverserPions »					
	« inverserPionsDir »					
	« pionEstPresent »					
	« pionEstPresentRecuratif »					
obtenirCoupIA	« obtenirCoupIA »					
	« scoreDUnCoup »					
	« coupValide »					
	« minMax »					
	« evaluerPlateau »					
	« evaluerNbCoupsPossiblesAdversaire »					
	« evaluerNbPionsCouleur »					
	« evaluerPositionsPionsPlateau »					

TABLE 3.1 – Répartition des tâches dans la phase de conception détaillée



## Chapitre 4

# Développement

	Fonction en C	Test unitaire associé
<b>afficher</b>	Gautier	
<b>TAD Couleur, Coups, Coup</b>	Gautier	Romain
<b>TAD Pion, Position, Plateau</b>	Claire	Romain
<b>faireUnePartie</b>	Riadh	
<b>initialiserPlateau</b>	Riadh, Gautier	Claire
<b>jouer</b>	Riadh	
<b>finPartie</b>	Riadh	Gautier
<b>plateauRempli</b>	Gautier	Claire
<b>nbPions</b>	Riadh, Gautier	Claire, Gautier
<b>jouerCoup</b>	Riadh	Claire
<b>inverserPion</b>	Riadh, Romain	Sandratra
<b>inverserPionDir</b>	Riadh, Romain	Sandratra
<b>pionEstPresent</b>	Riadh, Romain	Sandratra
<b>pionEstPresentRecuratif</b>	Riadh, Romain	
<b>obtenirCoupHumain</b>	Claire	Sandratra
<b>obtenirCoupIA</b>	Romain	Riadh
<b>profondeur</b>	Romain	
<b>listeCoupsPossibles</b>	Sandratra	Claire
<b>coupValide</b>	Sandratra	Claire
<b>copierPlateau</b>	Gautier	Claire
<b>minMax</b>	Gautier	
<b>scoreDUnCoup</b>	Gautier	Claire
<b>score</b>	Romain	
<b>evaluerPlateau</b>	Claire	Riadh
<b>main</b>	Gautier	

TABLE 4.1 – Répartition des tâches dans la phase de développement

# Conclusion