

# 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

29 décembre 2015



# Introduction

# Table des matières

<b>Introduction</b>	<b>2</b>
<b>I Analyse</b>	<b>5</b>
<b>1 Analyse des TAD</b>	<b>6</b>
1.1 Le TAD « Couleur » . . . . .	6
1.2 Le TAD « Pion » . . . . .	6
1.3 Le TAD « Position » . . . . .	6
1.4 Le TAD « Plateau » . . . . .	6
1.5 Le TAD « Coup » . . . . .	7
1.6 Le TAD « Coups » . . . . .	7
<b>2 Analyse descendante</b>	<b>8</b>
<b>II Conception préliminaire</b>	<b>9</b>
<b>1 Conception préliminaire des TAD</b>	<b>10</b>
1.1 Conception préliminaire du TAD « Couleur » . . . . .	10
1.2 Conception préliminaire du TAD « Pion » . . . . .	10
1.3 Conception préliminaire du TAD « Position » . . . . .	10
1.4 Conception préliminaire du TAD « Plateau » . . . . .	11
1.5 Conception préliminaire du TAD « Coup » . . . . .	11
1.6 Conception préliminaire du TAD « Coups » . . . . .	11
<b>2 Conception préliminaire des fonctions et procédures issues des analyses descendantes</b>	<b>12</b>
2.1 Conception préliminaire de l'analyse descendante de « Faire une partie » . . . . .	12
2.1.1 Types . . . . .	12
2.1.2 Type Direction . . . . .	12
2.1.3 Sous-programmes . . . . .	12
2.2 Conception préliminaire de l'analyse descendante de « obtenirCoupIA » . . . . .	13
<b>III Conception détaillée</b>	<b>14</b>
<b>1 Conception détaillée des TAD</b>	<b>15</b>
1.1 CD du type « Couleur » . . . . .	15
1.2 CD du type « Pion » . . . . .	15
1.3 CD du type « Position » . . . . .	15
1.4 CD du type « Plateau » . . . . .	15

1.5	CD du type « Coup » . . . . .	15
1.6	CD du type « Coups » . . . . .	15
<b>2</b>	<b>Conception détaillée des algorithmes compliqués de l'analyse « faireUnePartie »</b>	<b>17</b>
2.1	La procédure « faireUnePartie » . . . . .	17
2.2	La procédure « jouer » . . . . .	18
2.3	La procédure « jouerCoup » . . . . .	18
2.4	La procédure « inverserPions » . . . . .	18
2.5	La procédure « inverserPionsDir » . . . . .	19
2.6	La procédure « pionEstPresent » . . . . .	19
2.7	La procédure « pionEstPresentRecuratif » . . . . .	20
<b>3</b>	<b>Conception détaillée des algorithmes compliqués de l'analyse « obtenirCoupIA »</b>	<b>21</b>
3.1	La fonction « obtenirCoupIA » . . . . .	21
3.2	La fonction « scoreDUnCoup » . . . . .	22
3.3	La fonction « listeCoupsPossibles » . . . . .	22
3.4	La fonction « coupValide » . . . . .	23
3.5	La fonction « minMax » . . . . .	23
3.6	La fonction « evaluerPlateau » . . . . .	24
3.7	La fonction « evaluerNbCoupsPossiblesAdversaire » . . . . .	24
3.8	La fonction « evaluerNbPionsCouleur » . . . . .	24
3.9	La fonction « evaluerPositionsPionsPlateau » . . . . .	25
<b>IV</b>	<b>Développement</b>	<b>26</b>
<b>V</b>	<b>Répartition du travail</b>	<b>27</b>
1	Analyse descendante	28
2	Conception préliminaire	29
3	Conception détaillée	30
4	Développement	31
	<b>Conclusion</b>	<b>31</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:**  $\text{creerPlateau}: \rightarrow \text{Plateau}$   
 $\text{estCaseVide}: \text{Plateau} \times \text{Position} \rightarrow \text{Booleen}$   
 $\text{viderCase}: \text{Plateau} \times \text{Position} \rightarrow \text{Plateau}$   
 $\text{poserPion}: \text{Plateau} \times \text{Position} \times \text{Pion} \rightarrow \text{Plateau}$   
 $\text{obtenirPion}: \text{Plateau} \times \text{Position} \rightarrow \text{Pion}$   
 $\text{inverserPion}: \text{Plateau} \times \text{Position} \rightarrow \text{Plateau}$

**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:**  $\text{creerCoups}: \rightarrow \text{Coups}$   
 $\text{ajouterCoups}: \text{Coups} \times \text{Coup} \rightarrow \text{Coups}$   
 $\text{nbCoups}: \text{Coups} \rightarrow \text{Naturel}$   
 $\text{iemeCoup}: \text{Coups} \times \text{NaturelNonNul} \rightarrow \text{Coup}$

**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 ← VRAI
    finsi
fin
    
```

```

        pionPresent ← FAUX
    finssi
finssi
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**

k,nbPionsBlancs,nbPionsNoirs,nbPionsAParcourir : **Naturel**

**debut**

k ← 0

CPS\_creerCoups(coupsPossibles)

pionJoueur ← PI\_creerPion(couleur)

nbPions(plateau,nbPionsNoirs,nbPionsBlancs)

nbPionsAParcourir ← 64 - (nbPionsBlancs + nbPionsNoirs)

**pour** i ← 1 à 8 **faire**

**pour** j ← 1 à 8 **faire**

**si** k < nbPionsAParcourir **alors**

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)

k ← k+1

**finsi**

**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 ← evaluerNbCoupsPossiblesAdversaire(plateau,couleur)
    evaluer2 ← evaluerNbPionsCouleur(plateau,couleur)
    evaluer3 ← evaluerPositionsPionsPlateau(plateau,couleur, grilleScore)
    res ← 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 ← CL_changerCouleur(couleur)
    coupsAdversaire ← listeCoupsPossibles(plateau, couleurAdversaire)
    nbCoupsAdversaire ← nbCoups(coupsAdversaire)
    res ← 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 ← nbPionsNoirs−nbPionsBlancs
    sinon
        res ← 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

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
Sous-programme					
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

## Chapitre 4

# Développement

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

TABLE 4.1 – Répartition des tâches dans la phase de développement



# Conclusion