INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

asi Architecture des Systèmes d'Information

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

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** $\nrightarrow$ Position

Axiomes:
- *obtenirLigne(fixerPosition(ligne,colonne))=ligne*
- *obtenirColonne(fixerPosition(ligne,colonne))=colonne*

Préconditions: fixerPosition(ligne,colonne): $1 \leqslant \text{ligne} \leqslant 8$ & $1 \leqslant \text{colonne} \leqslant 8$

1.4 Le TAD « `Plateau` »

Nom: Plateau

Utilise: **Booleen**, Position, Pion

Opérations: creerPlateau: $\rightarrow$ Plateau
estCaseVide: Plateau $\times$ Position $\rightarrow$ **Booleen**
viderCase: Plateau $\times$ Position $\nrightarrow$ Plateau
poserPion: Plateau $\times$ Position $\times$ Pion $\nrightarrow$ Plateau
obtenirPion: Plateau $\times$ Position $\nrightarrow$ Pion
inverserPion: Plateau $\times$ Position $\nrightarrow$ Plateau

Axiomes:
- *estCaseVide(creerPlateau(),position)=VRAI*
- *estCaseVide(viderCase(plateau,position),position)=VRAI*
- *estCaseVide(poserPion(plateau,position,pion),position)=FAUX*
- *obtenirPion(poserPion(plateau,position,pion),position)=pion*
- *inverserPion(inverserPion(plateau,position),position)=plateau*

Préconditions: viderCase(plateau,position): non(estCaseVide(plateau,position))
poserPion(plateau,position): estCaseVide(plateau,position)
obtenirPion(plateau,position): non(estCaseVide(plateau,position))
inverserPion(plateau,position): non(estCaseVide(plateau,position))

1.5 Le TAD « Coup »

Nom: Coup

Utilise: Position, Pion

Opérations: creerCoup: Position $\times$ Pion $\rightarrow$ Coup
obtenirPositionCoup: Coup $\rightarrow$ Position
obtenirPionCoup: Coup $\rightarrow$ Pion

Axiomes:
- *obtenirPositionCoup(creerCoup(pos,pion))=pos*
- *obtenirPionCoup(creerCoup(pos,pion))=pion*

1.6 Le TAD « Coups »

Nom: Coups

Utilise: **Naturel**, **NaturelNonNul**, Coup

Opérations: creerCoups: $\rightarrow$ Coups
ajouterCoups: Coups $\times$ Coup $\rightarrow$ Coups
nbCoups: Coups $\rightarrow$ **Naturel**
iemeCoup: Coups $\times$ **NaturelNonNul** $\nrightarrow$ Coup

Axiomes:
- *iemeCoup(ajouterCoups(cps,cp),nbCoups(cps))=cp*
- *nbCoups(creerCoups())=0*
- *nbCoups(ajouterCoups(cps,cp))=nbCoups(cps)+1*

Préconditions: iemeCoup(cps,i): i$\leqslant$nbCoups(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)** $1 \leqslant$ ligne $\leqslant 8$ & $1 \leqslant$ colonne $\leqslant 8$

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)** $i \leqslant$ nbCoups(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** pionEstPresentRecursif (**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**) **: Entier**
— **fonction** scoreDUnCoup (plateau : Plateau, coup : Coup, couleurRef, couleurCourante : Couleur, profondeurCourante : **Naturel**) **: Entier**
— **fonction** score (plateau : Plateau, couleur : Couleur) **: Entier**
— **fonction** evaluerPlateau (plateau : Plateau, couleur : Couleur) **: 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] **de**Coup

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_inver

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_estCase
alors
pos ← DIR_positionSelonDirection(pos,dirATester)
pionEstPresentRecursif(pionJoueur, dirATester, pos, plateau, pionPresent)
sinon

```
        pionPresent ← FAUX
      finsi
   finsi
fin
```

2.7 La procédure « pionEstPresentRecursif »

procédure pionEstPresentRecursif **(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)
            pionEstPresentRecursif(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, pronfondeurMinMax : **Naturel**
coupsPossibles : Coups
scoreCourant, meilleurScore : **Entier**
coupCourant, meilleurCoup : Coup

debut
profondeurMinMax ← profondeur()
coupsPossibles ← listeCoupsPossibles(plateau, couleur)
si nbCoups(coupsPossibles) > 0 **alors**
meilleurCoup ← iemeCoup(coupsPossibles, 1)
meilleurScore ← scoreDUnCoup(plateau, meilleurCoup, couleur, couleur, profondeurMinMax)
pour i ←2 **à** nbCoups(coupsPossibles) **faire**
coupCourant ← iemeCoup(coupsPossibles, i)
scoreCourant ← scoreDUnCoup(plateau, coupCourant, couleur, couleur, profondeurMinMax)
si scoreCourant > meilleurScore **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) : Entier**

Déclaration plateauTest : Plateau

```
debut
   plateauTest ← copierPlateau(plateau)
   jouerCoup(coup, plateauTest)
   si plateauRempli(plateauTest) ou profondeurCourante = 0 alors
      retourner score(plateauTest, couleurRef)
   sinon
      retourner minMax(plateauTest, couleurRef, changerCouleur(couleurCourante), profondeurCourante − 1)
   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) 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) : Entier
   Déclaration  coupsPossibles : Coups
                resultat, score : Entier
                i : Naturel
debut
   coupsPossibles ← listeCoupsPossibles(plateau, couleurCourante)
   si nbCoups(coupsPossibles) > 0 alors
      resultat ← scoreDUnCoup(plateau, iemeCoup(coupsPossibles, 1), couleurRef, couleurCourante, profondeurCourante)
      pour i ←2 à nbCoups(coupsPossibles) faire
         score ← scoreDUnCoup(plateau, iemeCoup(coupsPossibles, i), couleurRef, couleurCourante, profondeurCourante)
         si couleurCourante = couleurRef alors
            resultat ← max(resultat, score)
         sinon
            resultat ← min(resultat, score)
         finsi
      finpour
   sinon
      si 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) **: Entier**

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

debut
evaluer1 ← evaluerNbCoupsPossiblesAdversaire(plateau,couleur)
evaluer2 ← evaluerNbPionsCouleur(plateau,couleur)
evaluer3 ← evaluerPositionsPionsPlateau(plateau,couleur)
res ← evaluer1 + evaluer2 + evaluer3
retourner res
fin

3.7 La fonction « evaluerNbCoupsPossiblesAdversaire »

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

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

debut
couleurAdversaire ← 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 sontEgales(couleur,noir()) **alors**
res ← nbPionsNoirs−nbPionsBlancs
sinon
res ← nbPionsBlancs−nbPionsNoirs
finsi
retourner res
fin

3.9 La fonction « evaluerPositionsPionsPlateau »

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

Déclaration res, resJoueur, resAdversaire : **Entier**
i, j, x, y : **Naturel**
pos : Position
grilleScore : **Tableau**[1..8][1..8] **de Entier**

```
debut
    grilleScore ← initialiserGrilleScore()
    resJoueur ← 0
    resAdversaire ← 0
    pour i ←1 à 8 faire
        pour j ←1 à 8 faire
            fixerPosition(i−1,j−1,pos)
            si non estCaseVide(plateau, pos) et sontEgales(obtenirCouleur(obtenirPion(pos)), couleur) alors
                resJoueur ← resJoueur+grilleScore[i−1][j−1]
            sinon
                si non estCaseVide(plateau,pos) alors
                    resAdversaire ← 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 / Sous-programme	Claire	Riadh	Sandratra	Gautier	Romain
`faireUnePartie`					
`initialiserPlateau`					
`jouer`					
`finPartie`					
`plateauRempli`					
`nbPions`					
`jouerCoup`					
`inverserPions`					
`inverserPionsDir`					
`pionEstPresent`					
`pionEstPresentRecursif`					
`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 / Sous-programme	Claire	Riadh	Sandratra	Gautier	Romain
`faireUnePartie`					
`initialiserPlateau`					
`jouer`					
`finPartie`					
`plateauRempli`					
`nbPions`					
`jouerCoup`					
`inverserPions`					
`inverserPionsDir`					
`pionEstPresent`					
`pionEstPresentRecursif`					
`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
`afficher`	Gautier	
TAD `Couleur, Coups, Coup`	Gautier	Romain
TAD `Pion, Position, Plateau`	Claire	Romain
`faireUnePartie`	Riadh	
`initialiserPlateau`	Riadh, Gautier	Claire
`jouer`	Riadh	
`finPartie`	Riadh	Gautier
`plateauRempli`	Gautier	Claire
`nbPions`	Riadh, Gautier	Claire, Gautier
`jouerCoup`	Riadh	Claire
`inverserPion`	Riadh, Romain	Sandratra
`inverserPionDir`	Riadh, Romain	Sandratra
`pionEstPresent`	Riadh, Romain	Sandratra
`pionEstPresentRecursif`	Riadh, Romain	
`obtenirCoupHumain`	Claire	Sandratra
`obtenirCoupIA`	Romain	Riadh
`profondeur`	Romain	
`listeCoupsPossibles`	Sandratra	Claire
`coupValide`	Sandratra	Claire
`copierPlateau`	Gautier	Claire
`minMax`	Gautier	
`scoreDUnCoup`	Gautier	?
`score`	Romain	?
`evaluerPlateau`	Claire	?
`main`	Gautier	

Table 4.1 – Répartition des tâches dans la phase de développement

Conclusion