Projet d'Algorithmique avancée
Othello

Groupe 1.3
KLEIBER Lise
IMBERT Gauthier
SERSOURI Mohamed
DAVID Maxence

4 janvier 2020

Table des matières

1 TAD

1.1 Couleur

Nom: Couleur

Opérations: Blanc: $\rightarrow$ Couleur
Noir: $\rightarrow$ Couleur
ChangerCouleur: Couleur $\rightarrow$ Couleur

Axiomes:
- ChangeC(Blanc)=Noir
- ChangeC(Noir)=Blanc

1.2 Pion

Nom: Pion

Utilise: Couleur, Coup

Opérations: CreerPion: Couleur $\rightarrow$ Pion
FlipPion: Pion $\rightarrow$ Pion
ObtenirCouleurPion: Pion $\rightarrow$ Couleur

Axiomes:
- ObtenirCouleurPion(Pion(a))=a
- FlipPion(FlipPion(a))=a

1.3 Position

Nom: Position

Utilise: **Caractere**, **Booleen**

Opérations: DefPosition: **Caractere** $\times$ **Caractere** $\rightarrow$ Position
Obtenirx: Position $\rightarrow$ **Caractere**
Obteniry: Position $\rightarrow$ **Caractere**
EstValide: **Caractere** $\times$ **Caractere** $\rightarrow$ Booleen

Axiomes:
- Obtenirx(DefPosition(x,y))=x
- Obteniry(DefPosition(x,y))=y

Préconditions: DefPosition(car,car): EstValide(car, car)

1.4 Coup

Nom: Coup

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

Opérations: PlacerCoup: Pion $\times$ Plateau $\times$ Position $\rightarrow$ Coup
ObtenirPosCoup: Coup $\rightarrow$ Position

ObtenirCouleurCoup: Coup → Couleur
CoupValide: Pion × Plateau × Position → **Booleen**

Axiomes: - ObtenirPosCoup(PlacerCoup(p,pl,ps))=ps
- ObtenirCouleurCoup(PlacerCoup(p,pl,ps))=p

Préconditions: PlacerCoup(p, pl, ps) : CoupValide(p, pl, ps):

1.5 Plateau

Nom: Plateau

Utilise: **Booleen**,Pion,Position, Coup

Opérations: CreerPlateau: → Plateau
ObtenirPion: Position × Plateau → Pion
RetournerPion: Position × Plateau × Coup → Plateau
EstVide: Position × Plateau → **Booleen**
ViderPlateau: Plateau → Plateau
EstGagnant: Coup × Plateau → **Booleen**

Préconditions: ObtenirPion(ps, pl) : non(EstVide(pl)):
RetournerPion(ps, pl, c) : non(EstVide(pl)):
ViderPlateau(pl) : non(estVide(pl)):
EstGagnant(pl) : non(estVide(pl)):

1.6 Coups

Nom: Coups

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

Opérations: InitCoups: → Coups
EstVide: Coups → **Booleen**
AjouterCoup: Coups × Coup → Coups
IemeCoup: Coups × **NaturelNonNul** → Coup
NbCoups: Coups → **Naturel**
EstPresent: Coups → **Naturel**

Axiomes: - estVide(InitCoups())
- nbCoups(InitCoups()) = 0
- ajouter(cs,c) = nbCoups(cs)+1

Préconditions: IemeCoup(c,i) : i ≤ nbCoups(c):

2 Signatures

2.1 affichagePlateau

procédure affichagePlateau (**E** pl : Plateau)

2.2 creerPlateau

fonction creerPlateau() **:** Plateau

2.3 majPlateau

fonction majPlateau(lePlateau : Plateau, leCoup : Coup) **:** Plateau

2.4 testLignes

fonction testLignes(laPosition : Position) **:** Position, Booleen

2.5 testColonnes

fonction testColonnes(laPosition : Position) **:** Position, Booleen

2.6 testDiagonales

fonction testDiagonales(laPosition : Position) **:** Position, Booleen

2.7 retournerPion

procédure retournerPion (p : Pion, lePlateau : Plateau, laPosition : Position)

2.8 changerCouleur

fonction changerCouleur(laCouleur : Couleur) **:** Couleur

2.9 placerCoup

fonction placerCoup(lePion : Pion,laPosition : Position, lePlateau : Plateau) **:** Coup

2.10 definirCouleurNouveauJoueur

fonction definirCouleurNouveauJoueur(dernierPionPlace : Pion) **:** Couleur

2.11 definirPosition

fonction definirPosition(abcisse, ordonnee : **Caractere**) **:** Postition

2.12 coupValide

fonction coupValide(lePion : Pion, positionCoup : Position, lePlateau : Plateau) **: Booleen**

2.13 estVide

fonction estVide(pl : Plateau, positionCoup : Position) **: Booleen**

2.14 retournerAuMoinsUnPion

fonction retournerAuMoinsUnPion(pl : Plateau, leCoup : Coup) **: Booleen**

2.15 partieTerminee

fonction partieTerminee(pl : Plateau) **: Booleen**

2.16 plateauPlein

fonction plateauPlein(lePlateau : Plateau) **: Booleen**

2.17 estGagnant

fonction estGagnant(pl : Plateau, lePion : Pion) **: Booleen**, Couleur

2.18 obtenirCouleur

fonction obtenirCouleur(p : Pion) **:** Couleur

2.19 plusDeCoups

fonction plusDeCoups(pl : Plateau) **: Booleen**

2.20 faireUnePartie

fonction faireUnePartie() **:** Couleur

2.21 moduleIA

fonction moduleIA(pl : Plateau) **:** Coup

2.22 scoreDUnCoup

fonction scoreDUnCoup(lesCoups : Coups) **: Naturel**

2.23 minMax

fonction minMax(lesCoups : Coups) **:** Coup

2.24 evaluer

fonction evaluer(leCoupAEvaluer : Coup) **: Naturel**

2.25 coupsPossibles

fonction coupsPossibles(lePlateau : plateau) **:** Coups

2.26 nbCoups

fonction nbCoups(lesCoups : Coups) **: Naturel**

2.27 iemeCoup

fonction iemeCoup(lesCoups : Coups, indice : **NaturelNonNul) :** Coup

3 Conception détaillée

3.1 estVide

fonction estVide(pl : Plateau, positionCoup : Position) **: Booleen**
début
  retourner etatPion(obtenirPion(positionCoup,pl))
fin

3.2 obtenirCouleurGagnant

fonction obtenirCouleurGagnant(pl : Plateau) **: Booleen**, Couleur

déclaration nbPionsNoirs, nbPionsBlancs, i, j : **Naturel**

début
```
nbPionsNoirs ← 0
nbPionsBlancs ← 0
pour i ←1 à 8 faire
    pour j ←1 à 8 faire
        si obtenirCouleur(obtenirPion(defPosition(i,j),pl))=Noir alors
            nbPionsNoirs ← nbPionsNoirs+1
        sinon
            si obtenirCouleur(obtenirPion(defPosition(i,j),pl))=Blanc alors
                nbPionsBlancs ← nbPionsBlancs+1
            finsi
        finsi
    finpour
finpour
si nbPionsNoirs > nbPionsBlancs alors
    retourner FAUX , Noir
sinon
    si nbPionsNoirs < nbPionsBlancs alors
        retourner FAUX , Blanc
    sinon
        retourner VRAI , Blanc  onretourneVRAIsiegaliteetunecouleurpardefaut
    finsi
finsi
```
fin

3.3 plusDeCoups

fonction plusDeCoups(pl : Plateau, couleurJoueurCourant : Couleur) **: Booleen**

déclaration i, j : **NaturelNonNul**
coupOK : **Booleen**

début
```
coupOK ← FAUX
i ← 1
tant que non(coupOK) et i ⩽ 8 faire
    j ← 1
```

```
    tant que non(coupOK) et j ⩽ 8 faire
      si coupValide(creerPion(couleurJoueurCourant),defPosition(i,j),pl) alors

        coupOK ← VRAI
      finsi
      j ← j+1
    fintantque
    i ← i+1
  fintantque
  retourner non(coupOK)
fin
```

3.4 retournerAuMoinsUnPion

3.4.1 retournerAuMoinsUnPion

fonction retournerAuMoinsUnPion(pl : Plateau, leCoup : Coup) **: Booleen**

déclaration modifHG, modifH, modifD, modifBD, modifB, modifBG, modifG : **Booleen**

début

```
  modifHG ← testModifDirection(leCoup,HG)
  modifH ← testModifDirection(leCoup,H)
  modifHD ← testModifDirection(leCoup,HD)
  modifD ← testModifDirection(leCoup,D)
  modifBD ← testModifDirection(leCoup,BD)
  modifB ← testModifDirection(leCoup,B)
  modifBG ← testModifDirection(leCoup,BG)
  modifG ← testModifDirection(leCoup,G)
  retourner modifHG ou modifH ou modifHD ou modifD ou modifBD ou
  modifB ou modifBG ou modfigG
```

fin

3.4.2 testModifDirection

fonction testModifDirection(pl : Plateau, leCoup : Coup, direction : {HG,H,HD,D,BD,B,BG,G}) **: Booleen**

déclaration incrementX, incrementY : **Entier**
positionAtester : Position

début

```
  cas où direction vaut
```

HG:
incrementX ← -1
incrementY ← 1
H:
incrementX ← 0
incrementY ← 1
HD:
incrementX ← 1
incrementY ← 1
D:
incrementX ← 1
incrementY ← 0
BD:
incrementX ← 1
incrementY ← -1
B:
incrementX ← 0
incrementY ← -1
BG:
incrementX ← -1
incrementY ← -1
G:
incrementX ← -1
incrementY ← 0

fincas
positionAtester ← defPosition(obtenirX(obtenirPosCoup(leCoup)+incrementX, obtenirY(obtenirPosCoup(leCoup)+incrementY)))
si obtenirCouleurCoup(leCoup)=Noir **alors**
couleurAdverse ← Blanc
sinon
couleurAdverse ← Noir
finsi
si obtenirCouleurCoup(leCoup) = obtenirCouleur(obtenirPion(positionAtester,pl))
alors
retourner FAUX
sinon
tant que (obtenirEtatPion(obtenirPion(positionAtester,pl)) $\neq$ 0) et (obtenirCouleur((obtenirPion(positionAtester,pl)) = couleurAdverse)) et (0 $\leqslant$ obtenirX(positionAtester) $\leqslant$ 8) et (0 $\leqslant$ obtenirY(positionAtester) $\leqslant$ 8) **faire**
positionAtester ← defPosition(obtenirX(positionAtester) + incre-

```
        mentX, obtenirY(positionAtester) + incrementY)
    fintantque
    si (obtenirEtatPion(obtenirPion(positionAtester,pl)) ≠ 0) et (obtenir-
    Couleur(obtenirPion(positionAtester,pl)) = obtenirCouleurCoup(leCoup))
    alors
      retourner VRAI
    sinon
      retourner FAUX
    finsi
  finsi
fin
```

3.5 partieTerminee

```
fonction partieTerminee( pl : Plateau, couleurJoueurCourant : Couleur) :
Booleen
début
  retourner (plateauPlein(pl) ou plusDeCoups(pl, couleurJoueurCourant))

fin
```

3.6 plateauPlein

```
fonction plateauPlein( pl : Plateau) : Booleen
déclaration  i,j : NaturelNonNul
             caseVide : Booleen
             pos : Position
début
  caseVide ← FAUX
  i ← 1
  tant que non(caseVide) et i ⩽ 8 faire
    j ← 1
    tant que non(caseVide) et j ⩽ 8 faire
      pos ← defPosition(i,j)
      si estVide(pl,pos) alors
        caseVide ← VRAI
      finsi
      j ← j + 1
    fintantque
    i ← i + 1
```

fintantque
retourner non(caseVide)
fin

3.7 changerCouleur

fonction changerCouleur(laCouleur : Couleur) **:** Couleur

déclaration blanc, noir : Couleur

début
si laCouleur = blanc **alors**
retourner noir
sinon
retourner blanc
finsi
fin

3.8 definirCouleurNouveauJoueur

fonction definirCouleurNouveauJoueur(dernierPionPlace : Pion) **:** Couleur

déclaration blanc, noir : Couleur

début
retourner changerCouleur(obtenirCouleurPion(dernierPionPlace))
fin

3.9 coupValide

fonction coupValide(leCoup : Coup, lePlateau : Plateau) **: Booleen**

⌊**précondition(s)** $(1 \leqslant \text{obtenirx}(\text{obtenirPosition}(\text{leCoup})) \leqslant \text{lePlateau.largeur})$ et $(1 \leqslant \text{obteniry}(\text{obtenirPosition}(\text{leCoup})) \leqslant \text{lePlateau.hauteur})$

début
si (estVide(obtenirPosition(leCoup), lePlateau)) et (retournerAuMoinsUnPion(lePlateau, leCoup)) **alors**
retourner VRAI
sinon
retourner FAUX
finsi
fin

3.10 majPlateau

procédure majPlateau (**E/S** lePlateau : Plateau, **E** leCoup : Coup, lesDirections : Ensemble<Direction>)

déclaration element : Direction, onRetourne : **Booleen**, pos : Position

début
  pour chaque element **de** lesDirections
    positionsARetourner, onRetourne ← testModifDirection(lePlateau, leCoup, element)
    si onRetourne = VRAI **alors**
      pour chaque pos **de** positionsARetourner
        retournerPion(lePlateau, pos)
      finpour
    finsi
  finpour
fin

3.11 retournerPion

procédure retournerPion (**E/S** lePlateau : Plateau, **E** positionDuPion : Position)

déclaration lePionModifie : Pion

début
  lePionModifie ← obtenirPion(lePlateau, positionDuPion)
  lePionModifie.couleurPion ← changerCouleur(obtenirCouleurPion(lePionModifie))

  poserPion(lePionModifie, positionDuPion, lePlateau)
fin

3.12 placerCoup

fonction placerCoup(partieFinie : booleen, lePlateau : Plateau) **:** Coup

  ⌊**précondition(s)** non(partieFinie)

déclaration pionAPlacer : Pion, abscisse, ordonnee : **Naturel**, nouveauCoup : Coup

début
  pionAPlacer.couleur ← definirCouleurNouveauJoueur(pionAPlacer)
  nouveauCoup.pion ← pionAPlacer
  nouveauCoup.position ← defPosition(caractereEnEntier(lire(abscisse)), caractereEnEntier(lire(ordonnee)))

si coupValide(nouveauCoup, pionAPlacer, obtenirPosition(nouveauCoup), lePlateau) **alors**
changerEtat(pionAPlacer)
retourner nouveauCoup
finsi
fin

3.13 defPosition

fonction defPosition(abscisse, ordonnee : entier) **:** Position

⌊**précondition(s)** (1 ⩽ abscisse ⩽ lePlateau.largeur) et (1 ⩽ ordonnee ⩽ lePlateau.hauteur)

déclaration laPosition : Position

début
laPosition.positionx ← abscisse
laPosition.positiony ← ordonnee
retourner laPosition
fin

3.14 obtenirPion

fonction obtenirPion(laPosition : Position, lePlateau : Plateau) **:** Pion

⌊**précondition(s)** (1 ⩽ obtenirx(laPosition) ⩽ lePlateau.largeur) et (1 ⩽ obteniry(laPosition) ⩽ lePlateau.hauteur)

début
retourner lePlateau[obtenirx(laPosition)][obteniry(laPosition)]
fin

3.15 CoupIA

fonction CoupIA(pl : plateau, CouleurReference : Couleur) **:** Coup

déclaration estPossible **Booleen**
CoupsATester : coups
CoupTest,BestCoup : Coup
Alpha,Beta :**Reel**
BestScoreCoup, ScoreTemp : **Naturel**

début
CoupsATester ← ObtenirCoupPossible(pl,CouleurReference)
profondeur ← 6

```
   estPossible ← non(estVide(CoupsATester))
   Alpha ← -infini
   Beta ← +infini
   si estPossible alors
      BestScoreCoup ← 0
      tant que non(estVide(CoupsATester)) faire
         ScoreTemp ← ScoreDUnCoup(pl,CouleurReference,CoupTest,profondeur,Alpha,Beta)

         si BestScoreCoup<ScoreTemp alors
            BestScoreCoup ← ScoreTemp
            CoupIA ← CoupTest
         finsi
         CoupsATester ← CoupsATester-CoupTest
      fintantque
   finsi
   retourner CoupIA
fin
```

3.16 ScoreDUnCoup

```
fonction ScoreDUnCoup( pl :plateau,CouleurReference :Couleur,coup :Coup,profondeur :Nature
Beta :Reel) : Entier
déclaration  TestFin : Booleen
             GrilleTemp : plateau
             AutreCouleur : Couleur
             score :Entier
début
   GrilleTemp ← CopierGrille(pl)
   AutreCouleur ← ChangerCouleur(CouleurReference)
   TestFin ← (ObtenirCoupPossible(pl,CouleurReference)=0) et (Obtenir-
   CoupPossible(pl,AutreCouleur)=0)
   MiseAJourPateau(GrilleTemp,coup)
   si (profondeur=0) et (TestFin) alors
      score ← evalue(GrilleTemp,CouleurReference)
      retourner score
   sinon
      retourner AlphaBeta(GrilleTemp,CouleurReference,AutreCouleur,Alpha,Beta,profondeur-
      1)
   finsi
fin
```

3.17 AlphaBeta

fonction AlphaBeta(pl :plateau,CouleurReference :Couleur,CouleurActuel :Couleur,Alpha : **Reel**,Beta : **Reel**,profondeur :**Naturel) : Entier**

déclaration CoupsPossible : Coups
res : **Entier**
score : **Entier**
i : **Entier**

début
CoupsPossible ← ObtenirCoupPossible(pl,CouleurActuel)
si non(estVide(CoupsPossible)) **alors**
res ← ScoreDUnCoup(pl,CouleurReference,iemeCoup(CoupsPossible,1)profondeur,Alpha,B

finsi
pour i ←2 **à** nbCoups(CoupsPossible) **faire**
score ← ScoreDUnCoup(pl,CouleurReference,iemeCoup(CoupsPossible,i),profondeur,Alpha

si CouleurReference<>CouleurActuel **alors**
res ← max(score,res)
si res>Alpha **alors**
Alpha ← res
si ALpha>Beta **alors**
retourner res
finsi
finsi
sinon
res ← min(score,res)
si res<Beta **alors**
Beta ← res
si Beta<ALpha **alors**
retourner res
finsi
finsi
finsi
finpour
retourner res
fin

3.18 affichagePlateau

procédure majPlateau (**E** lePlateau : Plateau)

déclaration i, j : **Naturel**

début
  pour i ←1 **à** lePlateau.hauteur **faire**
    pour j ←1 **à** lePlateau.largeur **faire**
      ecrire(lePlateau.cases[i][j])
    finpour
  finpour
fin

3.19 faireUnePartie

fonction faireUnePartie( ) **:** Couleur

déclaration couleurDuGagnant, joueurCourant : Couleur, joueurOuIA : Booleen, unPlateau : Plateau, coupJoueur1, coupJoueur2, coupIA : Coup, lesDirections : Ensemble<Direction>

début
  lire(joueurOuIA)
  joueurCourant ← noir
  unPlateau ← creerPlateau()
  si non(JoueurContreIA) **alors**
    tant que non(partieTerminee(unPlateau, joueurCourant)) **faire**
      coupJoueur1 ← placerCoup(unPlateau, non(partieTerminee(unPlateau)))

      majPlateau(unPlateau, coupJoueur1, lesDirections)
      affichagePlateau(unPlateau)
      coupJoueur2 ← placerCoup(unPlateau, non(partieTerminee(unPlateau)))

      majPlateau(unPlateau, coupJoueur2, lesDirections)
      affichagePlateau(unPlateau)
    fintantque
  sinon
    tant que non(partieTerminee(unPlateau, joueurCourant)) **faire**
      coupJoueur ← placerCoup(unPlateau, non(partieTerminee(unPlateau)))

      majPlateau(unPlateau, coupJoueur, lesDirections)
      affichagePlateau(unPlateau)
      coupIA ← moduleIA(unPlateau)
      majPlateau(unPlateau, coupIA, lesDirections)
      affichagePlateau(unPlateau)
    fintantque

finsi
retourner(obtenirCouleurGagnant(unPlateau))
fin

3.20 creerPlateau

fonction creerPlateau( ) **:** Plateau

déclaration blanc, noir : Couleur, unPlateau : Plateau

début
unPlateau.hauteur ← 8
unPlateau.largeur ← 8
poserPion(creerPion(Blanc), defPosition(4,4), unPlateau)
poserPion(creerPion(Noir), defPosition(4,5), unPlateau)
poserPion(creerPion(Blanc), defPosition(5,5), unPlateau)
poserPion(creerPion(Noir), defPosition(5,4), unPlateau)
fin