Cahier des Charges du Projet d'Algorithme

Groupe de projet 3
MAHAMADOU DJIBEROU Abdoul Jalil, MAISONNEUVE Justin,
MUGNERET Océane, NDIAYE Ndawa, POSE Celine

November 20, 2020

Contents

1 TAD

1.1 Pile

Nom: Pile

Paramètre: Element

Utilise: **Booleen**

Opérations: pile: $\rightarrow$ Pile

estVide: Pile $\rightarrow$ **Booleen**

empiler: Pile $\times$ Element $\rightarrow$ Pile

dépiler: Pile $\nrightarrow$ Pile

obtenirElement: Pile $\nrightarrow$ Element

Axiomes:
- estVide(pile())
- $\neg$estVide(empiler(p,e))
- depiler(empiler(p,e))=p
- obtenirElement(empiler(p,e))=e

Préconditions: depiler(p): $non(estVide(p))$

obtenirElement(p): $non(estVide(p))$

1.2 ListeChainée

Nom: ListeChainee

Paramètre: Element

Utilise: **Booleen**

Opérations: listeChainee: $\rightarrow$ ListeChainee

estVide: ListeChainee $\rightarrow$ **Booleen**

ajouter: ListeChainee $\times$ Element $\rightarrow$ ListeChainee

supprimerTete: ListeChainee $\nrightarrow$ ListeChainee

obtenirElement: ListeChainee $\nrightarrow$ Element

obtenirListeSuivante: ListeChainee $\nrightarrow$ ListeChainee

fixerListeSuivante: ListeChainee $\times$ ListeChainee $\nrightarrow$ ListeChainee

Axiomes:
- estVide(listeChainee())
- $\neg$estVide(ajouter(l,e))
- obtenirElement(ajouter(l,e))=e

- obtenirListeSuivante(ajouter(l,e))=l
- obtenirListeSuivante(fixerListeSuivante(l,l'))=l'

Préconditions: supprimerTete(l): $non(estVide(l))$

obtenirElement(l): $non(estVide(l))$

obtenirListeSuivante(l): $non(estVide(l))$

fixerListeSuivante(l,l'): $non(estVide(l))$

1.3 Direction

Nom: Direction

: **Type** Direction = haut, bas, gauche, droite, hautGauche, basGauche, hautDroite, basDroite

1.4 Repeller

Nom: Repeller

: **Type** Repeller = repN, repG, repJ

1.5 Stack

Nom: Stack

Paramètre: Repeller

Utilise: **Naturel**, Pile<Repeller>

Opérations: creerStack: $\rightarrow$ Stack

obtenirIDstack: Stack $\rightarrow$ **Naturel**

obtenirPile: Stack $\rightarrow$ Pile<Repeller>

fixerIDstack: Stack $\times$ **Naturel** $\rightarrow$ Stack

fixerPile: Stack $\times$ Pile<Repeller> $\rightarrow$ Element

Axiomes: - obtenirIDstack(obtenirStack($joueur1$))=obtenirIDJoueur($joueur1$)

1.6 Case

Nom: Case

Utilise: **Booleen**, **Naturel**, Stack, Repeller

Opérations: creerCase: **Naturel** $\times$ **Naturel** $\times$ **Naturel** $\rightarrow$ Case

obtenirCoordonnee: Case $\rightarrow$ **Naturel** $\times$ **Naturel**

obtenirValeur: Case → **Naturel**

estCaseDeDepart: Case → **Booleen**

caseLibre: Case → **Booleen**

ajouterRepeller: Case × Repeller → Case

verifierRepeller: Case → **Booleen**

supprimerRepeller: Case → Case

obtenirRepeller: Case → Repeller

ajouterStack: Case × Stack → Case

verifierStack: Case → **Booleen**

supprimerStack: case → Case

obtenirStack: case → Stack

sontCasesIdentiques: Case × Case → **Booleen**

1.7 Plateau

Nom: Plateau

Paramètre: Case

Utilise: **Booleen**, **Naturel**, Stack, Repeller, Direction

Opérations: creerPlateau: → plateau

obtenirCase: Plateau × **Naturel** × **Naturel** → case

deplacerStack : Plateau × Joueur × case → Plateau × Joueur

deplacerStackConflit : Plateau × ListeChainee<Joueur> × stack × direction → Plateau × ListeChainee<Joueur>

deplacerRepellerConflit : Plateau × case × direction → Plateau × Joueur

estDansPlateau: **Naturel** × **Naturel** × Plateau → **Booleen**

deposerRepeller : Joueur × Plateau ↛ Pateau × Joueur

Préconditions: deposerRepeller(j,p): caseLibre(obtenirCase(p,i,j)) = vrai

1.8 Score

Nom: Score

Paramètre: Repeller

Utilise: **Naturel**, ListeChainée<Repeller>

Opérations: creerScore: $\rightarrow$ Pile

obtenirListeRepellerNoir: Score $\rightarrow$ ListeChainée<Repeller>

obtenirListeRepellerGris: Score $\rightarrow$ ListeChainée<Repeller>

obtenirListeRepellerJaune: Score $\rightarrow$ ListeChainée<Repeller>

ajouterDansListeNoire: ListeChainée<Repeller> $\times$ Repeller $\rightarrow$ ListeChainée<Repeller>

ajouterDansListeGrise: ListeChainée<Repeller> $\times$ Repeller $\rightarrow$ ListeChainée<Repeller>

ajouterDansListeJaune: ListeChainée<Repeller> $\times$ Repeller $\rightarrow$ ListeChainée<Repeller>

retirerDeListerNoire: Score $\rightarrow$ Score

retirerDeListerGrise: Score $\rightarrow$ Score

retirerDeListerJaune: Score $\rightarrow$ Score

calculerScore: Score $\rightarrow$ **Naturel**

Axiomes: - retirerDeListeNoire(ajouterDansListeNoire(sc,R))=sc

1.9 Joueur

Nom: Joueur

Utilise: Stack,**Naturel**,Direction,Booleen,Plateau

Opérations: creerJoueur: Stack $\times$ Score $\times$ **Naturel** $\times$ **Naturel** $\rightarrow$ Joueur

obtenirIDJoueur: Joueur $\rightarrow$ **Naturel**

modifiernaturelIDJoueur: Joueur $\times$ **Naturel** $\rightarrow$ Joueur

obtenirScore: Joueur $\rightarrow$ Score

modifierScore: Joueur $\times$ Repeller $\rightarrow$ Joueur

obtenirPosition: Joueur $\times$ **Naturel** $\times$ **Naturel** $\rightarrow$ **Naturel** $\times$ **Naturel**

modifierPostition: Joueur $\times$ **Naturel** $\times$ **Naturel** $\rightarrow$ Joueur

obtenirStack: Joueur $\rightarrow$ Stack

modifierStack: Joueur $\times$ Stack $\rightarrow$ Joueur

1.10 ListeJoueur

Nom: ListeJoueur

Paramètre: Joueur

Utilise: **Naturel**, ListeChainee<Joueur>

Opérations: creerLienDernierSurPremier: ListeChaînée<Joueur> $\nrightarrow$ ListeChaînée<Joueur>

rechercherJoueur: ListeChaînée<Joueur> $\times$ **Naturel** $\nrightarrow$ Joueur

joueurSuivant: ListeChaînée<Joueur> $\nrightarrow$ ListeChaînée<Joueur>

Préconditions: creerLienDernierSurPremier: $non(estVide(l))$

rechercherJoueur: $non(estVide(l))$

joueurSuivant: $non(estVide(l)$

1.11 Conflit

Nom: Conflit

Paramètre: Case

Opérations: creerConflit: Case $\times$ Case $\rightarrow$ Conflit

sontConflitsIdentiques: Conflit $\times$ Conflit $\rightarrow$ **Booleen**

2 Conception préliminaire

2.1 Conception prélimiaire du TAD Pile

fonction pile () **:** Pile
fonction estVide (unePile : Pile) **: Booleen**
procédure empiller (**E/S** unePile : Pile ; **E** E : Element)
procédure dépiler (**E/S** unePile : Pile)
fonction obtenirElement (unePile : Pile) **:** Element

2.2 Conception prélimiaire du TAD ListeChainee

fonction listeChainee () **:** ListeChainee
fonction estVide (uneListe : ListeChainee) **: Booleen**
procédure ajouter (**E/S** uneListe : ListeChainee ; **E** E : Element)
procédure supprimerTete (**E/S** uneListe : ListeChainee)
fonction obtenirElement (uneListe : ListeChainee) **:** Element
fonction obtenirListeSuivante (uneListe : ListeChainee) **:** ListeChainee
procédure fixerListeSuivante (**E/S** uneListe : ListeChainee ; **E** nouvelle-Suite : ListeChainee)

2.3 Conception prélimiaire du TAD Stack

fonction creerStack () **:** Stack
fonction obtenirIDstack (S : Stack) **: Naturel**
fonction obtenirPile (S : Stack) **:** Pile<Repeller>
procédure fixerIDstack (**E/S** S : Stack ; **E** id : **Naturel**)
procédure fixerPile (**E/S** S : Stack ; **E** P : Pile<Repeller>)

2.4 Conception prélimiaire du TAD Case

fonction creerCase (x, y, valeur : **Naturel**) **:** Case
procédure obtenirCoordonée (**E** C : Case ; **S** x, y : **Naturel**)
fonction obtenirValeur (C : Case) **: Naturel**
fonction estCaseDeDepart (C : Case) **: Booleen**
fonction caseLibre (C : Case) **: Booleen**
procédure ajouterRepeller (**E/S** C : Case ; **E** R : Repeller)
fonction verifierRepeller (C : Case) **: Booleen**
procédure supprimerRepeller (**E/S** C : Case)
fonction obtenirRepeller (C : Case) **:** Repeller
procédure ajouterStack (**E/S** C : Case ; **E** S : Stack)

fonction verifierStack (C : Case) **: Booleen**
procédure supprimerStack (**E/S** C : Case)
fonction obtenirStack (C : Case) **:** Stack
fonction sontCasesIdentiques (C1, C2 : Case) **: Booleen**

2.5 Conception prélimiaire du TAD Plateau

fonction creerPlateau () **:** P : Plateau
fonction obtenirCase (P : plateau ; x,y : **Naturel) :** case
procédure deplacerStack (**E/S** P : Plateau , **E/S** J : joueur , **E** C : Case)
procédure deplacerStackConflit (**E/S** P : Plateau, Lj : ListeChaineeDeJoueur , **E** S : stack , **E** D : direction)
procédure deplacerRepellerConflit (**E/S** P : Plateau , **E** C : case , **E** D : direction)
fonction estDansPlateau (x : **Naturel**, y : **Naturel**, P : Plateau) **: Booleen**
procédure deposerRepeller (**E/S** P : plateau , **E/S** J : joueur)

2.6 Conception prélimiaire du TAD Score

fonction creerScore () **:** Score
fonction obtenirListeRepellerNoir (sc : Score) **:** ListeChaînée<Repeller>
fonction obtenirListeRepellerGris (sc : Score) **:** ListeChaînée<Repeller>
fonction obtenirListeRepellerJaune (sc : Score) **:** ListeChaînée<Repeller>
procédure ajouterDansListeNoire (**E/S** sc : Score ; **E** R :Repeller)
procédure ajouterDansListeGrise (**E/S** sc : Score ; **E** R :Repeller)
procédure ajouterDansListeJaune (**E/S** sc : Score ; **E** R :Repeller)
procédure retirerDeListeNoire (**E/S** sc : Score)
procédure retirerDeListeGrise (**E/S** sc : Score)
procédure retirerDeListeJaune (**E/S** sc : Score)
fonction calculerScore (sc : Score) **:** Naturel

2.7 Conception prélimiaire du TAD Joueur

fonction creerJoueur (S : Stack, Sc : Score, x, y : **Naturel) :** Joueur
fonction obtenirIDJoueur (J : Joueur) **: Naturel**
procédure modifiernaturelIDJoueur (**E/S** J : Joueur ; **E** IDJ : **Naturel**)
fonction obtenirScore (J : Joueur) **:** Score
procédure modifierScore (**E/S** J : Joueur ; **E** R : Repeller)

procédure obtenirPosition **(E** J : Joueur ; **S** x, y: **Naturel)**
procédure modifierPosition **(E/S** J : Joueur ; **E** x, y : **Naturel)**
fonction obtenirStack **(**J : Joueur**) :** Stack
procédure modifierStack **(E/S** J : Joueur ; **E** S : Stack **)**

2.8 Conception prélimiaire du TAD ListeJoueur

procédure creerLienDernierSurPremier **(E/S** Lj : ListeChaînée<Joueur> **)**
fonction rechercherJoueur **(**Lj : ListeChaînée<Joueur>, id : **Naturel) :** Joueur
procédure joueurSuivant **(E/S** Lj : ListeChaînée<Joueur> **)**

2.9 Conception prélimiaire du TAD Conflit

fonction creerConflit **(**C1 : Case, C2 : Case**) :** Conflit
fonction sontConflitsIdentiques **(**co1, co2 : Conflit**) : Booleen**

3 Conception détaillée

3.1 Conception détaillée des TADs

3.1.1 Conception détaillée du TAD Pile

```
Type Pile = Structure
   nbElements : Naturel
   lesElements : ListeChaineeDElements
finstructure

fonction pile () : Pile
   Déclaration   p : Pile
debut
   p.nbElements ← 0
   p.lesElements ← listeChainee()
fin

fonction estVide (p : Pile) : Booleen
debut
   si p.nbElements=0 alors
      retourner VRAI
   sinon
      retourner FAUX
   finsi
fin

procédure empiler ( E/S p : Pile ; E E : Element )
debut
   ajouter(p.lesElements, E)
   p.nbElements ← p.nbElements+1
fin

procédure depiler ( E/S p : Pile )
debut
   supprimerTete(p.lesElements)
   p.nbElements ← p.nbElements-1
fin

fonction obtenirElement (p : Pile) : Element
debut
```

retourner listeChainee.obtenirElement(p.lesElements)
fin

3.1.2 Conception détaillée du TAD ListeChainee

Type Noeud = **Structure**
IElement : Element
listeSuivante : ListeChainee
finstructure

Type ListeChainee = Noeud^

fonction listeChainee **() :** ListeChainee
debut
retourner NULL
fin

fonction estVide (l : ListeChainee) **: Booleen**
debut
si l=NULL **alors**
retourner vrai
sinon
retourner faux
finsi
fin

procédure ajouter **(** **E/S** l : ListeChainee ; **E** e : Element **)**
Déclaration temp : ListeChainee
debut
temp ← l
l ← **allouer**(Noeud)
l->IElement ← e
fixerListeSuivante(l,temp)
fin

procédure supprimerTete **(** **E/S** l : ListeChainee **)**
Déclaration temp : ListeChainee
debut
temp ← l

```
   l ← obtenirListeSuivante(l)
   desallouer(temp)
fin

fonction obtenirElement (l:ListeChainee) : Element
debut
   retourner l->IElement
fin

fonction obtenirListeSuivante (l:ListeChainee) : ListeChainee
debut
   retourner l->listeSuivante
fin

procédure fixerListeSuivante ( E/S l : ListeChainee ; E l' : ListeChainee
)
debut
   l->listeSuivante ← l'
fin
```

3.1.3 Conception détaillée du TAD Stack

```
Type PileRep = Pile<Repeller>

Type Stack = Structure
   p : PileRep
   id : Naturel
finstructure

procédure fixerPile ( E/S S : Stack ; E p : PileRep )
debut
   S.p ← p
fin

procédure fixerIDStack ( E/S S : Stack ; E id : Naturel )
debut
   S.id ← id
fin
```

fonction obtenirPile (S : Stack) **:** PileRep
debut
  retourner S.p
fin

fonction obtenirIDStack (S : Stack) **: Naturel**
debut
  retourner S.id
fin

fonction creerStack () **:** Stack
  Déclaration resultat : Stack
debut
  fixerPile(resultat, pile())
  retourner resultat
fin

3.1.4 Conception détaillée du TAD Case

Type Case = **Structure**
  x : **Naturel**
  y : **Naturel**
  valeur : **Naturel**
  B : **Booleen**
  R : Repeller
  S : Stack
finstructure

fonction creerCase (x : **Naturel**,y : **Naturel**, valeur : **Naturel) :** Case
  Déclaration resultat : Case
debut
  resultat.x ← x
  resultat.y ← y
  resultat.valeur ← valeur
  si C.y<=10 et C.y=>4 et C.x=4 ou C.y<=10 et C.y=>4 et C.x=10
  ou C.x<=10 et C.x=>4 et C.y=4 ou C.x<=10 et C.x=>4 et C.y=10
  alors
    resultat.B ← VRAI
  sinon

```
      resultat.B ← FAUX
   finsi
   retourner resultat
fin

procédure obtenirCoordonnée ( E C : Case ; S abscisse : Naturel, ordonnée : Naturel )
debut
   abscisse ← C.x
   ordonnée ← C.y
fin

fonction obtenirValeur (C : Case ) : Naturel
   Déclaration
debut
   retourner C.valeur
fin

fonction estCaseDeDepart (C : Case ) : Booleen
   Déclaration
debut
   retourner C.B
fin

fonction caseLibre (C : Case) : Booleen
   Déclaration
debut
   si C.R = NULL et C.S = NULL alors
      retourner VRAI
   sinon
      retourner FAUX
   finsi
fin

procédure ajouterRepeller ( E/S C : Case ; E R : Repeller )
debut
   C.R ← R
fin

fonction verifierRepeller (C : Case) : Booleen
```

```
   Déclaration
debut
   si C.R = NULL alors
      retourner FAUX
   sinon
      retourner VRAI
   finsi
fin

procédure supprimerRepeller ( E/S C : Case )
debut
   si C.R != NULL alors
      C.R ← NULL
   finsi
fin

fonction obtenirRepeller (C : Case) : Repeller
   Déclaration
debut
   retourner C.R
fin

procédure ajouterStack ( E/S C : Case ; E S : Stack )
debut
   C.S ← S
fin

fonction verifierStack (C : Case) : Booleen
   Déclaration
debut
   si C.S = NULL alors
      retourner FAUX
   sinon
      retourner VRAI
   finsi
fin

procédure supprimerStack ( E/S C : Case )
debut
   si C.S != NULL alors
```

```
      C.S ← NULL
   finsi
fin

fonction obtenirStack (C : Case) : Stack
   Déclaration
debut
   retourner C.S
fin

fonction sontCasesIdentiques (C1, C2 : Case) : Booleen
debut
   retourner C1.x=C2.x et C1.y=C2.y
fin
```

3.1.5 Conception détaillée du TAD Plateau

```
Type Plateau =  Tableau[1...15][1...15] de Case

Type ListeChaineeDeConflit = listeChainee<Conflit>

fonction creerPlateau () : Plateau
   Déclaration  P : Plateau
debut
   P(1)(1) ← creerCase(1,1,0)
   P(1)(2) ← creerCase(1,2,0)
   P(1)(3) ← creerCase(1,3,0)
   P(1)(4) ← creerCase(1,4,0)
   P(1)(5) ← creerCase(1,5,0)
   P(1)(6) ← creerCase(1,6,0)
   P(1)(7) ← creerCase(1,7,0)
   P(1)(8) ← creerCase(1,8,0)
   P(1)(9) ← creerCase(1,9,0)
   P(1)(10) ← creerCase(1,10,0)
   P(1)(11) ← creerCase(1,11,0)
   P(1)(12) ← creerCase(1,12,0)
   P(1)(13) ← creerCase(1,13,0)
   P(1)(14) ← creerCase(1,14,0)
   P(1)(15) ← creerCase(1,15,0)
```

P(2)(1) ← creerCase(2,1,0)
P(2)(2) ← creerCase(2,2,3)
P(2)(3) ← creerCase(2,3,4)
P(2)(4) ← creerCase(2,4,5)
P(2)(5) ← creerCase(2,5,6)
P(2)(6) ← creerCase(2,6,1)
P(2)(7) ← creerCase(2,7,2)
P(2)(8) ← creerCase(2,8,3)
P(2)(9) ← creerCase(2,9,4)
P(2)(10) ← creerCase(2,10,5)
P(2)(11) ← creerCase(2,11,6)
P(2)(12) ← creerCase(2,12,1)
P(2)(13) ← creerCase(2,13,2)
P(2)(14) ← creerCase(2,14,3)
P(2)(15) ← creerCase(2,15,0)
P(3)(1) ← creerCase(3,1,0)
P(3)(2) ← creerCase(3,2,5)
P(3)(3) ← creerCase(3,3,6)
P(3)(4) ← creerCase(3,4,1)
P(3)(5) ← creerCase(3,5,2)
P(3)(6) ← creerCase(3,6,3)
P(3)(7) ← creerCase(3,7,4)
P(3)(8) ← creerCase(3,8,5)
P(3)(9) ← creerCase(3,9,6)
P(3)(10) ← creerCase(3,10,1)
P(3)(11) ← creerCase(3,11,2)
P(3)(12) ← creerCase(3,12,3)
P(3)(13) ← creerCase(3,13,4)
P(3)(14) ← creerCase(3,14,5)
P(3)(15) ← creerCase(3,15,0)
P(4)(1) ← creerCase(4,1,0)
P(4)(2) ← creerCase(4,2,1)
P(4)(3) ← creerCase(4,3,2)
P(4)(4) ← creerCase(4,4,3)
P(4)(5) ← creerCase(4,5,4)
P(4)(6) ← creerCase(4,6,5)
P(4)(7) ← creerCase(4,7,6)
P(4)(8) ← creerCase(4,8,3)
P(4)(9) ← creerCase(4,9,2)
P(4)(10) ← creerCase(4,10,3)

P(4)(11) ← creerCase(4,11,4)
P(4)(12) ← creerCase(4,12,5)
P(4)(13) ← creerCase(4,13,6)
P(4)(14) ← creerCase(4,14,1)
P(4)(15) ← creerCase(4,15,0)
P(5)(1) ← creerCase(5,1,0)
P(5)(2) ← creerCase(5,2,4)
P(5)(3) ← creerCase(5,3,5)
P(5)(4) ← creerCase(5,4,6)
P(5)(5) ← creerCase(5,5,1)
P(5)(6) ← creerCase(5,6,2)
P(5)(7) ← creerCase(5,7,3)
P(5)(8) ← creerCase(5,8,4)
P(5)(9) ← creerCase(5,9,5)
P(5)(10) ← creerCase(5,10,6)
P(5)(11) ← creerCase(5,11,1)
P(5)(12) ← creerCase(5,12,2)
P(5)(13) ← creerCase(5,13,3)
P(5)(14) ← creerCase(5,14,4)
P(5)(15) ← creerCase(5,15,0)
P(6)(1) ← creerCase(6,1,0)
P(6)(2) ← creerCase(6,2,6)
P(6)(3) ← creerCase(6,3,1)
P(6)(4) ← creerCase(6,4,2)
P(6)(5) ← creerCase(6,5,3)
P(6)(6) ← creerCase(6,6,4)
P(6)(7) ← creerCase(6,7,5)
P(6)(8) ← creerCase(6,8,6)
P(6)(9) ← creerCase(6,9,1)
P(6)(10) ← creerCase(6,10,2)
P(6)(11) ← creerCase(6,11,3)
P(6)(12) ← creerCase(6,12,4)
P(6)(13) ← creerCase(6,13,5)
P(6)(14) ← creerCase(6,14,6)
P(6)(15) ← creerCase(6,15,0)
P(7)(1) ← creerCase(7,1,0)
P(7)(2) ← creerCase(7,2,2)
P(7)(3) ← creerCase(7,3,3)
P(7)(4) ← creerCase(7,4,4)
P(7)(5) ← creerCase(7,5,5)

P(7)(6) ← creerCase(7,6,6)
P(7)(7) ← creerCase(7,7,1)
P(7)(8) ← creerCase(7,8,2)
P(7)(9) ← creerCase(7,9,3)
P(7)(10) ← creerCase(7,10,4)
P(7)(11) ← creerCase(7,11,5)
P(7)(12) ← creerCase(7,12,6)
P(7)(13) ← creerCase(7,13,1)
P(7)(14) ← creerCase(7,14,2)
P(7)(15) ← creerCase(7,15,0)
P(8)(1) ← creerCase(8,1,0)
P(8)(2) ← creerCase(8,2,4)
P(8)(3) ← creerCase(8,3,5)
P(8)(4) ← creerCase(8,4,6)
P(8)(5) ← creerCase(8,5,1)
P(8)(6) ← creerCase(8,6,2)
P(8)(7) ← creerCase(8,7,3)
P(8)(8) ← creerCase(8,8,4)
P(8)(9) ← creerCase(8,9,5)
P(8)(10) ← creerCase(8,10,6)
P(8)(11) ← creerCase(8,11,1)
P(8)(12) ← creerCase(8,12,2)
P(8)(13) ← creerCase(8,13,3)
P(8)(14) ← creerCase(8,14,4)
P(8)(15) ← creerCase(8,15,0)
P(9)(1) ← creerCase(9,1,0)
P(9)(2) ← creerCase(9,2,6)
P(9)(3) ← creerCase(9,3,1)
P(9)(4) ← creerCase(9,4,2)
P(9)(5) ← creerCase(9,5,3)
P(9)(6) ← creerCase(9,6,4)
P(9)(7) ← creerCase(9,7,5)
P(9)(8) ← creerCase(9,8,6)
P(9)(9) ← creerCase(9,9,1)
P(9)(10) ← creerCase(9,10,2)
P(9)(11) ← creerCase(9,11,3)
P(9)(12) ← creerCase(9,12,4)
P(9)(13) ← creerCase(9,13,5)
P(9)(14) ← creerCase(9,14,6)
P(9)(15) ← creerCase(9,15,0)

P(10)(1) ← creerCase(10,1,0)
P(10)(2) ← creerCase(10,2,2)
P(10)(3) ← creerCase(10,3,3)
P(10)(4) ← creerCase(10,4,4)
P(10)(5) ← creerCase(10,5,5)
P(10)(6) ← creerCase(10,6,6)
P(10)(7) ← creerCase(10,7,1)
P(10)(8) ← creerCase(10,8,2)
P(10)(9) ← creerCase(10,9,3)
P(10)(10) ← creerCase(10,10,4)
P(10)(11) ← creerCase(10,11,5)
P(10)(12) ← creerCase(10,12,6)
P(10)(13) ← creerCase(10,13,1)
P(10)(14) ← creerCase(10,14,2)
P(10)(15) ← creerCase(10,15,0)
P(11)(1) ← creerCase(11,1,0)
P(11)(2) ← creerCase(11,2,5)
P(11)(3) ← creerCase(11,3,6)
P(11)(4) ← creerCase(11,4,5)
P(11)(5) ← creerCase(11,5,2)
P(11)(6) ← creerCase(11,6,3)
P(11)(7) ← creerCase(11,7,4)
P(11)(8) ← creerCase(11,8,5)
P(11)(9) ← creerCase(11,9,6)
P(11)(10) ← creerCase(11,10,1)
P(11)(11) ← creerCase(11,11,2)
P(11)(12) ← creerCase(11,12,3)
P(11)(13) ← creerCase(11,13,4)
P(11)(14) ← creerCase(11,14,5)
P(11)(15) ← creerCase(11,15,0)
P(12)(1) ← creerCase(12,1,0)
P(12)(2) ← creerCase(12,2,1)
P(12)(3) ← creerCase(12,3,2)
P(12)(4) ← creerCase(12,4,3)
P(12)(5) ← creerCase(12,5,4)
P(12)(6) ← creerCase(12,6,5)
P(12)(7) ← creerCase(12,7,6)
P(12)(8) ← creerCase(12,8,3)
P(12)(9) ← creerCase(12,9,2)
P(12)(10) ← creerCase(12,10,3)

P(12)(11) ← creerCase(12,11,4)
P(12)(12) ← creerCase(12,12,5)
P(12)(13) ← creerCase(12,13,6)
P(12)(14) ← creerCase(12,14,1)
P(12)(15) ← creerCase(12,15,0)
P(13)(1) ← creerCase(13,1,0)
P(13)(2) ← creerCase(13,2,3)
P(13)(3) ← creerCase(13,3,4)
P(13)(4) ← creerCase(13,4,5)
P(13)(5) ← creerCase(13,5,6)
P(13)(6) ← creerCase(13,6,1)
P(13)(7) ← creerCase(13,7,2)
P(13)(8) ← creerCase(13,8,1)
P(13)(9) ← creerCase(13,9,4)
P(13)(10) ← creerCase(13,10,5)
P(13)(11) ← creerCase(13,11,6)
P(13)(12) ← creerCase(13,12,1)
P(13)(13) ← creerCase(13,13,2)
P(13)(14) ← creerCase(13,14,3)
P(13)(15) ← creerCase(13,15,0)
P(14)(1) ← creerCase(14,1,0)
P(14)(2) ← creerCase(14,2,5)
P(14)(3) ← creerCase(14,3,6)
P(14)(4) ← creerCase(14,4,1)
P(14)(5) ← creerCase(14,5,2)
P(14)(6) ← creerCase(14,6,3)
P(14)(7) ← creerCase(14,7,4)
P(14)(8) ← creerCase(14,8,5)
P(14)(9) ← creerCase(14,9,6)
P(14)(10) ← creerCase(14,10,1)
P(14)(11) ← creerCase(14,11,2)
P(14)(12) ← creerCase(14,12,3)
P(14)(13) ← creerCase(14,13,4)
P(14)(14) ← creerCase(14,14,5)
P(14)(15) ← creerCase(14,15,0)
P(15)(1) ← creerCase(15,1,0)
P(15)(2) ← creerCase(15,2,0)
P(15)(3) ← creerCase(15,3,0)
P(15)(4) ← creerCase(15,4,0)
P(15)(5) ← creerCase(15,5,0)

P(15)(6) ← creerCase(15,6,0)
P(15)(7) ← creerCase(15,7,0)
P(15)(8) ← creerCase(15,8,0)
P(15)(9) ← creerCase(15,9,0)
P(15)(10) ← creerCase(15,10,0)
P(15)(11) ← creerCase(15,11,0)
P(15)(12) ← creerCase(15,12,0)
P(15)(13) ← creerCase(15,13,0)
P(15)(14) ← creerCase(15,14,0)
P(15)(15) ← creerCase(15,15,0)

(P(3)(3)).R ← repN
(P(3)(13)).R ← repN
(P(13)(3)).R ← repN
(P(13)(13)).R ← repN
(P(3)(8)).R ← repG
(P(13)(8)).R ← repG
(P(8)(3)).R ← repG
(P(8)(13)).R ← repG
(P(8)(8)).R ← repJ
fin

fonction obtenirCase (P : Plateau, x : **Naturel**, y : **Naturel) :** Case
Déclaration
debut
retourner P(x)(y)
fin

procédure deplacerStack (**E/S** P : plateau , **E/S** J : joueur , **E** C : case)
debut
P(C.x)(C.y) ← J.S
(P(J.x)(J.y)).S ← NULL
J.x ← C.x
J.y ← C.y
fin

procédure deplacerStackConflit (**E/S** P : plateau, Lj : ListeChaineeDeJoueur , **E** S : stack , **E** D : direction)
Déclaration J : Joueur

debut
J ← rechercherJoueur(Lj, obtenirIDStack(S))
cas où D **vaut**
Haut
P(J.x-1)(J.y).S ← S
P(j.x)(j.y).S ← NULL
J.x ← x-1

Bas
P(J.x+1)(J.y).S ← S
P(j.x)(j.y).S ← NULL
J.x ← x+1

Gauche
P(j.x)(j.y-1).S ← S
P(j.x)(j.y).S ← NULL
J.y ← y-1

Droite
P(j.x)(j.y+1).S ← S
P(j.x)(j.y).S ← NULL
J.y ← y+1

HautGauche
P(J.x-1)(J.y-1).S ← S
P(J.x)(J.y).S ← NULL
J.x ← x-1
J.y ← y-1

HautDroite
P(J.x-1)(J.y+1).S ← S
P(J.x)(J.y).S ← NULL
J.x ← x-1
J.y ← y+1

BasGauche
P(J.x+1)(J.y-1).S ← S
P(J.x)(J.y).S ← NULL
J.x ← x+1
J.y ← y-1

```
        BasDroite
        P(J.x+1)(J.y+1).S ← S
        P(J.x)(J.y).S ← NULL
        J.x ← x+1
        J.y ← y+1

    fincas
    J.S ← S
fin

procédure deplacerRepellerConflit ( E/S P : plateau , E C: case , E D
: direction )
    Déclaration  i : Naturel
                 j : Naturel
debut
    i ← C.x
    j ← C.y
    si  non estVide(C) alors
        cas où D vaut
            Haut
            P(i-1)(j).R ← C.R
            P(i)(j).R ← NULL

            Bas
            P(i+1)(j).R ← C.R
            P(i)(j).R ← NULL

            Gauche
            P(i)(j-1).R ← C.R
            P(i)(j).R ← NULL

            Droite
            P(i)(j+1).R ← C.R
            P(i)(j).R ← NULL

            HautGauche
            P(i-1)(j-1).R ← C.R
            P(i)(j).R ← NULL
```

```
            HautDroite
            P(i-1)(j+1).R ← C.R
            P(i)(j).R ← NULL

            BasGauche
            P(i+1)(j-1).R ← C.R
            P(i)(j).R ← NULL

            BasDroite
            P(i+1)(j+1).R ← C.R
            P(i)(j).R ← NULL
        fincas
    finsi
fin

fonction estDansPlateau ( P : plateau, x : Naturel, y : Naturel) :
Booleen
debut
    si x ≥ 2 et x ≤ 14 et y ≥ 2 et y ≤ 14 alors
        retourner VRAI
    sinon
        retourner FAUX
    finsi
fin

procédure deposerRepeller ( E/S P : plateau , E/S J : joueur )
    Déclaration  R : Repeller
debut
    R ← repN
    (P(J.x)(J.y)).R ← R
    depiler((J.S).p)
fin
```

3.1.6 Conception détaillée du TAD Score

Type ListeChaineeDeRepeller = ListeChainee<Repeller>

Type Score = **Structure**
  Ln : ListeChaineeDeRepeller

```
   Lg :  ListeChaineeDeRepeller
   Lj :  ListeChaineeDeRepeller
finstructure

fonction creerScore () : Score
   Déclaration  sc : Score ;
debut
   sc.Ln ← listeChainee()
   sc.Lg ← listeChainee()
   sc.Lj ← listeChainee()
   retourner sc
fin

fonction obtenirListeRepellerNoir (sc : Score) : ListeChaineeDeRepeller
debut
   retourner sc.Ln
fin

fonction obtenirListeRepellerGris (sc : Score) : ListeChaineeDeRepeller
debut
   retourner sc.Lg
fin

fonction obtenirRepellerJaune (sc : Score) : ListeChaineeDeRepeller
debut
   retourner sc.Lj
fin

procédure ajouterDansListeNoire ( E/S sc : Score ; E R : Repeller )
debut
   ajouter(obtenirListeRepellerNoir(sc),R)
fin

procédure ajouterDansListeGrise ( E/S sc : Score ; E R : Repeller )
debut
   ajouter(obtenirListeRepellerGris(sc),R)
fin

procédure ajouterDansListeJaune ( E/S sc : Score ; E R : Repeller )
debut
```

```
    ajouter(obtenirListeRepellerJaune(sc),R)
fin

procédure retirerDeListeNoire ( E/S sc : Score ;)
debut
    sc.Ln ← obtenirListeSuivante(sc.Ln)
fin

procédure retirerDeListeGrise ( E/S sc : Score ;)
debut
    sc.Lg ← obtenirListeSuivante(sc.Lg)
fin

procédure retirerDeListeJaune ( E/S sc : Score ;)
debut
    sc.Lj ← obtenirListeSuivante(sc.Lj)
fin

fonction calculerScore (sc : Score) : Naturel
    Déclaration  L1, L2, L3 : ListeChaineeDeRepeller; somme : Naturel
debut
    somme ← 0
    L1 ← obtenirListeRepellerNoir(sc)
    L2 ← obtenirListeRepellerGris(sc)
    L3 ← obtenirListeRepellerJaune(sc)
    tant que non(estVide(L1)) faire
        somme ← somme+1
        L1 ← obtenirListeSuivante(L1)
    fintantque
    tant que non(estVide(L2)) faire
        somme ← somme+3
        L2 ← obtenirListeSuivante(L2)
    fintantque
    tant que non(estVide(L3)) faire
        somme ← somme+5
        L3 ← obtenirListeSuivante(L3)
    fintantque
    retourner somme
fin
```

3.1.7 Conception détaillée du TAD Joueur

Type Joueur = **Structure**
S : Stack
id : **Naturel**
Sc : Score
x : **Naturel**
y : **Naturel**
finstructure

fonction creerJoueur (S : stack, Sc : Score , x : **Naturel**, y : **Naturel**) **:** Joueur
Déclaration J : Joueur
debut
J.S ← S
J.id ← S.id
J.Sc ← Sc
J.x ← x
J.y ← y
retourner J
fin

fonction obtenirIDJoueur (J : Joueur) **: Naturel**
debut
retourner J.id
fin

procédure modifiernaturelIDJoueur (**E/S** J : Joueur ; **E** IDJ : **Naturel**)
debut
J.id ← IDJ
fin

fonction obtenirScore (J : Joueur) **:** Score
debut
retourner J.Sc
fin

procédure modifierScore (**E/S** J : Joueur ; **E** R : Repeller)
debut

```
    si R=repN alors
        ajouterDansListeNoire(J.Sc,R)
    sinon
        si R = repG alors
            ajouterDansListeGrise(J.Sc,R)
        sinon
            ajouterDansListeJaune(J.Sc,R)
        finsi
    finsi
fin

procédure obtenirPosition ( E J : Joueur ;  S  x,y : Naturel  )
debut
    x ← J.x
    y ← J.y
fin

procédure modifierPosition (  E/S J : Joueur ; E  x,y : Naturel )
debut
    J.x ← x
    J.y ← y
fin

fonction obtenirStack (J : Joueur) : Stack
debut
    retourner J.S
fin

procédure modifierStack ( E S : Stack ;  E/S J : Joueur )
debut
    J.S ← S
fin
```

3.1.8 Conception détaillée du TAD ListeJoueur

Type ListeChaineeDeJoueur = ListeChainee<Joueur>

procédure creerLienDernierSurPremier (**E/S** Lj : ListeChaineeDeJoueur)

```
   Déclaration   temp : ListeChaineeDeJoueur
debut
   temp ← Lj
   tant que obtenirListeSuivante(temp) != NULL faire
      temp ← obtenirListeSuivante(temp)
   fintantque
   fixerListeSuivante(temp, Lj)
fin

fonction rechercherJoueur (Lj : listeChaineeDeJoueur, idatrouver : Naturel
) : Joueur
debut
   tant que obtenirElement(Lj).id != idatrouver faire
      Lj ← obtenirListeSuivante(Lj)
   fintantque
   retourner obtenirElement(Lj)
fin

procédure joueurSuivant ( E/S Lj : ListeChaineeDeJoueur )
debut
   Lj ← obtenirListeSuivante(Lj)
fin
```

3.1.9 Conception détaillée du TAD Conflit

```
Type Conflit = Structure
   c1 : Case
   c2 : Case
finstructure

fonction creerConflit (C1 : Case, C2 : Case ) : Conflit
   Déclaration   co : Conflit
debut
   co.c1 ← C1
   co.c2 ← C2
   retourner co
fin

fonction sontConflitsIdentiques (co1, co2 : Conflit ) : Booleen
```

debut
retourner (sontCasesIdentiques(co1.c1,co2.c1) et sontCasesIdentiques(co1.c2,co2.c2))
ou (sontCasesIdentiques(co1.c1,co2.c2) et sontCasesIdentiques(co1.c2,co2.c1))

fin

3.2 Conception détaillée des fonctions de l'analyse descendante

3.2.1 Conception détaillée de la fonction jeuRepello

procédure jeuRepello **()**
Déclaration nbJ : **Naturel**, P : Plateau, Lj : ListeChaineeDeJoueur, J : Joueur
debut
nbJ ← DUnbJoueur()
initialiserPartie(nbJ,P,Lj)
jouerPartie(P, Lj,nbJ)
J ← gagnant(Lj, nbJ)
ecrire("le gagnant est le joueur ",J.id)
fin

fonction DUnbJoueur **() :** Naturel
Déclaration nbJ : **Naturel**
debut
ecrire(Combien de joueurs jouent?)
lire(nbJ)
retourner nbJ
fin

fonction gagnant **(**Lj : ListeChaineeDeJoueur, nbJ : **Naturel) :** Joueur
Déclaration J, JGagnant : Joueur, Sc : Score, Score, ScoreMax : **Naturel**, i : **Naturel**
debut
ScoreMax ← -1
pour i ←1 **à** nbJ **pas de** 1 **faire**
J ← obtenirElement(Lj)
Sc ← obtenirScore(J)
Score ← calculerScore(Sc)

```
        si Score ≥ ScoreMax alors
            ScoreMax ← Score
            JGagnant ← J
        finsi
        J ← joueurSuivant(Lj)
    finpour
    retourner JGagnant
fin
```

3.2.2 Conception détaillée de la fonction initialiserPartie

```
fonction creerListeJoueur () : ListeChaineeDeJoueur
    Déclaration  Lj : ListeChaineeDeJoueur
debut
    Lj ← listeChainee()
    retourner Lj
fin

procédure initialiserPartie ( E nbj : Naturel ;  S P : Plateau, Lj : ListeChaineeDeJoueur )
debut
    P ← creerPlateau()
    Lj ← creerListeJoueur()
    initialiserJoueur(P,Lj,nbj)
fin
```

3.2.3 Conception détaillée de la fonction initialiserJoueur

```
procédure ajouterJoueurListe ( E/S Lj : ListeChaineeDeJoueur ;  E J : Joueur )
debut
    ajouter(Lj,J)
fin

procédure initialiserJoueur ( E/S P : Plateau, Lj : ListeChaineeDeJoueur ; E N : Naturel )
    Déclaration  Sc : Score, S : Stack, x,y,nb : Naturel, J : Joueur
debut
    nb ← N
```

```
    tant que nb != 0 faire
        S ← initialiserStack(N,nb)
        Sc ← creerScore()
        x ← 1
        y ← 1
        tant que non(estCaseDeDepart(obtenirCase(P,x,y)) et conflitPresent(P,
        P(x)(y))  faire
            ecrire(("Saisir des coordonées de départ valides"))
            lire(x)
            lire(y)
        fintantque
        J ← creerJoueur(S,Sc,x,y)
        P(x)(y).S ← S
        nb ← nb-1
    fintantque
    creerLienDernierSurPremier(Lj)
fin
```

3.2.4 Conception détaillée de la fonction initialiserStack

```
fonction creerPileRepeller () : PileRep
    Déclaration   P : PileRep
debut
    P ← pile()
    retourner P
fin

fonction initialiserStack (N : Naturel, id : Naturel) : Stack
    Déclaration   P : PileRep, R : Repeller, S : Stack, nbRep : Naturel
debut
    S ← creerStack()
    fixerIDStack(S,id)
    cas où N vaut
        2
        nbRep ← 15
        3
        nbRep ← 12
        4
        nbRep ← 10
```

```
   fincas
   P ← creerPileRepeller()
   R ← repN
   pour i ←1 à nbRep faire
      empiler(P,R)
   finpour
   fixerPile(S,P)
   retourner S
fin
```

3.2.5 Conception détaillée de la fonction jouerPartie

```
fonction partieFinie (Lj : ListeChaineeDeJoueurs, nbj : Naturel) : Booleen

   Déclaration  B : Booleen, J : Joueur, i : Naturel
debut
   B ← VRAI
   pour i ←1 à nbj pas de 1 faire
      J ← obtenirElement(Lj)
      si (((J.S).p).nbElement = 0) alors
         B ← B et VRAI
      sinon
         B ← B et FAUX
      finsi
      Lj ← obtenirListeSuivante(Lj)
   finpour
   retourner B
fin

procédure jouerPartie ( E/S Lj :ListeChaineeDeJoueur ;  E/S P : Plateau
; E nbj : Naturel )
debut
   tant que non(partieFinie(Lj,nbj)) faire
      jouerCoup(Lj,P)
   fintantque
fin
```

3.2.6 Conception détaillée de la fonction jouerCoup

fonction choisirCaseArriveeValide **(**Lc : ListeChaineeDeCases**) :** Case
  Déclaration C : Case, temp : ListeChaineeDeCases, i,choix : **Naturel**
debut
  i ← 1
  temp ← Lc
  tant que obtenirListeSuivante(temp) ≠ null **faire**
    ecrire("Choix numéro ", i ,":", (obtenirElement(temp).x ,obtenirElement(temp).y))
    temp ← obtenirListeSuivante(temp)
    i ← i+1
  fintantque
  ecrire("Faites votre choix")
  lire(choix)
  i ← 1
  temp ← Lco
  repeter
    si i = choix **alors**
      C ← obtenirElement(temp)
    finsi
    temp ← obtenirListeSuivante(temp)
    i ← i+1
  jusqu'a ce que (i = choix) ou (obtenirListeSuivante(temp) = null)
  retourner C
fin

procédure jouerCoup **(E/S** Lj : ListeChaineeDeJoueur ; **E/S** P :Plateau **)**
  Déclaration J : Joueur, Lc : ListeChaineeDeCase, C : Case
debut
  J ← obtenirElement(Lj)
  deposerRepeller(J,P)
  Lc ← listerCasesArrivéesValides(J,P)
  si ¬(estVide(Lc)) **alors**
    C ← choisirCaseArrivéeValide(Lc)
    deplacerStack(J,P,C)
    Conflit(Lj,P)
    joueurSuivant(Lj)
  sinon

```
        repeter
            ecrire("Pas de déplacement possible : Saisir nouvelle position pour
            le joueur ")
            lire(x)
            lire(y)
        jusqu'a ce que estCaseDeDepart(obtenirCase(P,x,y) et ¬(conflitPresent(P,P(x)(y))
        et caseLibre(P(x)(y))
        supprimerStack(P(J.x)(J.y))
        J.x ← x
        J.y ← y
        (P(x)(y)).S ← J.S
    finsi
fin
```

3.2.7 Conception détaillée de la fonction listerCasesArrivéesValides

```
Type ListeChaineeDeCase = ListeChainee<Case>

fonction creerListeCase () : ListeChaineeDeCases
    Déclaration   Lc : ListeChaineeDeCases
debut
    Lc ← listeChainee()
    retourner Lc
fin

procédure ajouterCaseListe ( E/S Lc : ListeChaineeDeCase ;  E C : Case
)
debut
    ajouter(Lc,C)
fin
```

```
fonction listerCasesArrivéesValides (J : Joueur,P : Plateau) : ListeChaineeDeCase
   Déclaration  Lc : ListeChaineeDeCase; x,y,i,j,k,m,val : Naturel; c : Case
debut
   Lc ← creerListeCase()
   obtenirPosition(jo,x,y)
   pour k ←1 à 8 pas de 1 faire
      i ← x
      j ← y
      cas où k vaut
         1
         c ← obtenirCase(p,x,y+1)
         2
         c ← obtenirCase(p,x-1,y+1)
         3
         c ← obtenirCase(p,x-1,y)
         4
         c ← obtenirCase(p,x-1,y-1)
         5
         c ← obtenirCase(p,x,y-1)
         6
         c ← obtenirCase(p,x+1,y-1)
         7
         c ← obtenirCase(p,x+1,y)
         8
         c ← obtenirCase(p,x+1,y+1)
      fincas
      val ← obtenirValeur(c)-1
      m ← k
      tant que caseLibre(c) et val != 0 faire
         cas où m vaut
            1
            j ← j+1
            si j = 15 alors
               j ← j-2
               m ← 5
            finsi
            2
            i ← i-1
```

```
j ← j+1
si i=1 et j=15 alors
    i ← i+2
    j ← j-2
    m ← 6
sinon
    si i=1 alors
        i ← i+2
        m ← 8
    sinon
        si j=15 alors
            j ← j-2
            m ← 4
        finsi
    finsi
finsi
3
i ← i-1
si i=1 alors
    i ← i+2
    m ← 7
finsi
4
i ← i-1
j ← j-1
si i=1 et j=1 alors
    i ← i+2
    j ← j+2
    m ← 8
sinon
    si i=1 alors
        i ← i+2
        m ← 6
    sinon
        si j=1 alors
            j ← j+2
            m ← 2
        finsi
    finsi
finsi
```

```
5
j ← j-1
si j=1 alors
    j ← j+2
    m ← 1
finsi
6
i ← i+1
j ← j-1
si i=15 et j=1 alors
    i ← i-2
    j ← j+2
    m ← 2
sinon
    si j=1 alors
        j ← j+2
        m ← 8
    sinon
        si i=15 alors
            i ← i-2
            m ← 4
        finsi
    finsi
finsi
7
i ← i+1
si i=15 alors
    i ← i-2
    m ← 3
finsi
8
i ← i+1
j ← j+1
si i=15 et j=15 alors
    i ← i-2
    j ← j-2
    m ← 4
sinon
    si j=15 alors
        j ← j-2
```

```
                m ← 6
            sinon
                si i=15 alors
                    i ← i-2
                    m ← 2
                finsi
            finsi
        finsi
        fincas
        c ← obtenirCase(p,i,j)
        val ← val-1
    fintantque
    si val = 0 alors
        ajouterCaseListe(Lc,c)
    finsi
  finpour
  retourner Lc
fin
```

3.2.8 Conception détaillée de la fonction conflit

procédure recupererPionsSortis **(E/S** Lj : ListeChaineeDeJoueur, P : Plateau **)**

Déclaration J,J2 : Joueur, i,x,y : naturel, R : Repeller, S : Stack

```
debut
  J ← obtenirElement(Lj)
  pour i ←1 à 15 pas de 1 faire
    si verifierRepeller(P(1)(i)) alors
      R ← (P(1)(i)).R
      modifierScore(J,R)
      supprimerRepeller(P(1)(i))
    sinon
      si verfierStack(P(1)(i)) alors
        S ← (P(1)(i)).S
        modifierScore(J,repN)
        J2 ← rechercherJoueur(Lj,S.id)
        depiler(S.p)
        si ¬(estVide(obtenirListeRepellerJaune(J2.Sc))) alors
          retirerDeListeJaune(J2.Sc)
          modifierScore(J,repJ)
```

```
            sinon
                si ¬(estVide(obtenirListeRepellerGris(J2.Sc))) alors
                    retirerDeListeGrise(J2.Sc)
                    modifierScore(J,repG)
                sinon
                    si ¬(estVide(obtenirListeRepellerNoir(J2.Sc))) alors
                        retirerDeListeNoire(J2.Sc)
                        modifierScore(J,repN)
                    finsi
                finsi
            finsi
            repeter
                ecrire("Saisir nouvelle position pour le joueur sortie")
                lire x
                lire y
            jusqu'a ce que estCaseDeDepart(obtenirCase(P,x,y) et ¬(conflitPresent(P,P(x)(y))
            et caseLibre(P(x)(y))
            supprimerStack(P(J2.x)(J2.y))
            J2.x ← x
            J2.y ← y
            (P(x)(y)).S ← S
        finsi
    finsi
finpour
pour i ←2 à 15 pas de 1 faire
    si verifierRepeller(P(i)(1)) alors
        R ← (P(i)(1)).R
        modifierScore(J,R)
        supprimerRepeller(P(i)(1))
    sinon
        si verfierStack(P(i)(1)) alors
            S ← (P(i)(1)).S
            modifierScore(J,repN)
            J2 ← rechercherJoueur(Lj,S.id)
            depiler(S.p)
            si ¬(estVide(obtenirListeRepellerJaune(J2.Sc))) alors
                retirerDeListeJaune(J2.Sc)
                modifierScore(J,repJ)
            sinon
                si ¬(estVide(obtenirListeRepellerGris(J2.Sc))) alors
```

```
                retirerDeListeGrise(J2.Sc)
                modifierScore(J,repG)
            sinon
                si ¬(estVide(obtenirListeRepellerNoir(J2.Sc))) alors
                    retirerDeListeNoire(J2.Sc)
                    modifierScore(J,repN)
                finsi
            finsi
        finsi
        repeter
            ecrire("Saisir nouvelle position pour le joueur sortie")
            lire x
            lire y
        jusqu'a ce que estCaseDeDepart(obtenirCase(P,x,y) et ¬(conflitPresent(P,P(x)(y))
        et caseLibre(P(x)(y))
        supprimerStack(P(J2.x)(J2.y))
        J2.x ← x
        J2.y ← y
        (P(x)(y)).S ← S
    finsi
finsi
finpour
pour i ←2 à 15 pas de 1 faire
    si verifierRepeller(P(15)(i)) alors
        R ← (P(15)(i)).R
        modifierScore(J,R)
        supprimerRepeller(P(15)(i))
    sinon
        si verfierStack(P(15)(i)) alors
            S ← (P(15)(i)).S
            modifierScore(J,repN)
            J2 ← rechercherJoueur(Lj,S.id)
            depiler(S.p)
            si ¬(estVide(obtenirListeRepellerJaune(J2.Sc))) alors
                retirerDeListeJaune(J2.Sc)
                modifierScore(J,repJ)
            sinon
                si ¬(estVide(obtenirListeRepellerGris(J2.Sc))) alors
                    retirerDeListeGrise(J2.Sc)
                    modifierScore(J,repG)
```

```
                sinon
                    si ¬(estVide(obtenirListeRepellerNoir(J2.Sc))) alors
                        retirerDeListeNoire(J2.Sc)
                        modifierScore(J,repN)
                    finsi
                finsi
            finsi
            repeter
                ecrire("Saisir nouvelle position pour le joueur sortie")
                lire x
                lire y
            jusqu'a ce que estCaseDeDepart(obtenirCase(P,x,y) et ¬(conflitPresent(P,P(x)(y))
            et caseLibre(P(x)(y))
            supprimerStack(P(J2.x)(J2.y))
            J2.x ← x
            J2.y ← y
            (P(x)(y)).S ← S
        finsi
    finsi
finpour
pour i ←2 à 14 pas de 1 faire
    si verifierRepeller(P(i)(15)) alors
        R ← (P(i)(15)).R
        modifierScore(J,R)
        supprimerRepeller(P(i)(15))
    sinon
        si verfierStack(P(i)(15)) alors
            S ← (P(i)(15)).S
            modifierScore(J,repN)
            J2 ← rechercherJoueur(Lj,S.id)
            depiler(S.p)
            si ¬(estVide(obtenirListeRepellerJaune(J2.Sc))) alors
                retirerDeListeJaune(J2.Sc)
                modifierScore(J,repJ)
            sinon
                si ¬(estVide(obtenirListeRepellerGris(J2.Sc))) alors
                    retirerDeListeGrise(J2.Sc)
                    modifierScore(J,repG)
                sinon
                    si ¬(estVide(obtenirListeRepellerNoir(J2.Sc))) alors
```

```
                retirerDeListeNoire(J2.Sc)
                modifierScore(J,repN)
            finsi
        finsi
    finsi
    repeter
        ecrire("Saisir nouvelle position pour le joueur sortie")
        lire x
        lire y
    jusqu'a ce que estCaseDeDepart(obtenirCase(P,x,y) et ¬(conflitPresent(P,P(x)(y))
    et caseLibre(P(x)(y))
    supprimerStack(P(J2.x)(J2.y))
    J2.x ← x
    J2.y ← y
    (P(x)(y)).S ← S
  finsi
finsi
finpour

fonction choisirConflit (Lco : ListeChaineeDeConflit) : Conflit
  Déclaration  i , choix : Naturel, temp : ListeChaineeDeconlfit, co
               : Conflit
debut
  i ← 1
  temp ← Lco
  tant que obtenirListeSuivante(temp) ≠ null faire
    ecrire("Choix numéro ",i,":", (obtenirElement(temp).c1.x ,obtenirEle-
    ment(temp).c1.y), (obtenirElement(temp).c2.x , obtenirElement(temp).c1.y))

    temp ← obtenirListeSuivante(temp)
    i ← i+1
  fintantque
  ecrire("Faites votre choix")
  lire(choix)
  i ← 1
  temp ← Lco
  repeter
    si i = choix alors
      co.c1 ← obtenirElement(temp).c1
      co.c2 ← obtenirElement(temp).c2
```

```
            finsi
            temp ← obtenirListeSuivante(temp)
            i ← i+1
        jusqu'a ce que (i = choix) ou (obtenirListeSuivante(temp) = null)
        retourner co
    fin

    procédure conflit ( E/S Lj : ListeChaineeDeJoueur, P : Plateau )
        Déclaration   Lco :ListeChaineeDeConflit, Co : Conflit
    debut
        si ¬(estVide(listerConflit(P))) alors
            Lco ← listerConflit(P)
            Co ← choisirConflit(Lco)
            resoudreConflit(Lj, P, Co)
            recupererPionsSortis(Lj,P)
            conflit(Lj,P)
        finsi
    fin

fin
```

3.2.9 Conception détaillée de la fonction listerConflit)

```
Type ListeChaineeDeConflit = ListeChainee<Conflit>

fonction estConflitPresent (C1 : case, C2 : case) : Booleen
debut
    si ¬(caseLibre(C1)) et ¬(caseLibre(C2)) alors
        retourner VRAI
    sinon
        retourner FAUX
    finsi
fin

fonction listerConflit (P : plateau) : ListeChaineeDeConlfit
    Déclaration   i , j : Naturel
                  Lco : ListeChaineeDeConflit
                  co : Conflit
debut
```

```
    Lco ← creerListeConflit()
    pour i ←2 à 14 pas de 1 faire
        pour j ←2 à 14 pas de 1 faire
            si estConflitPresent( P(i)(j) , P(i-1)(j) )  alors
                co ← creerConflit( P(i)(j) , P(i-1)(j) )
                ajouterConflitListe(Lco , co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i-1)(j-1) )  alors
                co ← creerConflit( P(i)(j) , P(i-1)(j-1) )
                ajouterConflitListe(Lco , co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i)(j+1) )  alors
                co ← creerConflit( P(i)(j) , P(i)(j+1) )
                ajouterConflitListe(Lco, co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i+1)(j) )  alors
                co ← creerConflit( P(i)(j) , P(i+1)(j) )
                ajouterConflitListe(Lco , co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i+1)(j+1) )  alors
                co ← creerConflit( P(i)(j) , P(i+1)(j+1))
                ajouterConflitListe(Lco , co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i+1)(j-1) )  alors
                co ← creerConflit( P(i)(j) , P(i+1)(j-1))
                ajouterConflitListe(Lco , co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i)(j-1) )  alors
                co ← creerConflit( P(i)(j) , P(i)(j-1))
                ajouterConflitListe(Lco , co)
            finsi
            si estConlfitPresent( P(i)(j) , P(i-1)(j+1) )  alors
                co ← creerConflit( P(i)(j) , P(i-1)(j+1))
                ajouterConflitListe(Lco , co)
            finsi
        finpour
    finpour
    retourner Lco
fin
```

3.2.10 Conception détaillée de la fonction ResoudreConflit

fonction CalculerDirection **(** co : Conflit, k : **Naturel) :** Direction
  Déclaration C1 : case
  C2 : case
  i : **Naturel**
  j : **Naturel**
debut
  C1 ← co.c1
  C2 ← co.c2
  i ← C1.x
  j ← C1.y
  si k=1 **alors**
    si (C2.x = i-1) et (C2.y = j) **alors**
      retourner Haut
    finsi
    si (C2.x = i+1) et (C2.y = j) **alors**
      retourner Bas
    finsi
    si (C2.x = i) et (C2.y = j-1) **alors**
      retourner Gauche
    finsi
    si (C2.x = i) et (C2.y = j+1) **alors**
      retourner Droite
    finsi
    si (C2.x = i-1) et (C2.y = j+1) **alors**
      retourner HautDroite
    finsi
    si (C2.x = i-1) et (C2.y = j-1) **alors**
      retourner HautGauche
    finsi
    si (C2.x = i+1) et (C2.y = j-1) **alors**
      retourner BasGauche
    finsi
    si (C2.x = i+1) et (C2.y = j+1) **alors**
      retourner BasDroite
    finsi
  sinon
    C1 ← co.c1
    C2 ← co.c2

```
        i ← C2.x
        j ← C2.y
        si ( C1.x = i-1 ) et ( C1.y = j) alors
            retourner Haut
        finsi
        si ( C1.x = i+1 ) et ( C1.y = j) alors
            retourner Bas
        finsi
        si ( C1.x = i ) et ( C1.y = j-1) alors
            retourner Gauche
        finsi
        si ( C1.x = i ) et ( C1.y = j+1) alors
            retourner Droite
        finsi
        si ( C1.x = i-1 ) et ( C1.y = j+1) alors
            retourner HautDroite
        finsi
        si ( C1.x = i-1 ) et ( C1.y = j-1) alors
            retourner HautGauche
        finsi
        si ( C1.x = i+1 ) et ( C1.y = j-1) alors
            retourner BasGauche
        finsi
        si ( C1.x = i+1 ) et ( C1.y = j+1) alors
            retourner BasDroite
        finsi
    finsi
fin

procédure ChoixDeplacer ( E/S Co : Conflit ; S D : Direction, C : case
)
    Déclaration  choix : Naturel
                 i : Naturel
                 j : Naturel
                 C1 , C2 : case
debut
    ecrire( " Choisir case à déplacer 1 : case 1 ou 2 : case 2 " )
    lire(choix)
    si choix = 1 alors
        C ← co.c1
```

```
        D ← CalculerDirection(co,1)
    sinon
        C ← co.c2
        D ← CalculerDirection(co,2)
    finsi
fin

procédure resoudreConflit ( E/S Lj : ListeChaineeDeJoueur, P : Plateau
;  E co : Conflit )
    Déclaration   c : Case, d : Direction, S : Stack
debut
    choixDeplacer(co,d,c)
    si verifierStack(c) alors
        S ← obtenirStack(c)
        deplacerStackConflit(P,Lj,S,d)
    sinon
        si verifierRepeller(c) alors
            R ← obtenirRepeller(c)
            deplacerRepellerConflit(P,c,d)
        finsi
    finsi
fin
```

3.3 Analyse descendante