Tableau et enregistrement en Pascal et Première version du jeu du puissance 4

I3 - Algorithmique et programmation

Nicolas Delestre

Plan

Architecture des Systèmes d'Information

Tableau

- Les tableaux en Pascal sont statiques
  - La taille des tableaux est fixée à la compilation
  - La taille des tableaux est déterminée en spécifiant un intervalle
- Par défaut, l'utilisation d'un indice de tableau en dehors de cet intervalle n'est pas vérifié :
  - cela *peut* provoquer un *Runtime error* si la zone mémoire n'appartient pas au processus (non compatible avec le concept de tests unitaires)
  - le mieux est d'ajouter une directive de compilation ({$RANGECHECKS ON}) qui va ajouter du code automatiquement pour vérifier que l'indice est entre les bornes de définition
- On utilise rarement directement les types tableaux :
  - Le Pascal « n'accepte pas » directement des paramètres de type tableau, mais il accepte des paramètres dont le type est basé sur le type tableau
  - On crée de nouveau type qui sont des tableaux

```
Type NouveauType = ... ;
```

Tableau à une dimension 1 / 3

Définition/Déclaration

```
array [indiceDebut..indiceFin] of TypeDesElements
```

avec :

- *indiceDebut* et *indiceFin* sont des constantes de type scalaire

Exemple

```
const
   MAX              = 100;
   PREMIERE_LETTRE  = 'a';
   DERNIERE_LETTRE  = 'z';

type
   Lettre         = PREMIERE_LETTRE..DERNIERE_LETTRE;
   Entiers        = array[1..MAX] of Integer;
   NbOccurences   = array[Lettre] of Integer;
```

Tableau à une dimension 2 / 3

Utilisation

- On utilise une case du tableau en spécifiant l'indice de cette case entre crochet

Exemple

```
procedure initialiserEntiers(var t : Entiers;
    val : Integer);
var i :  Integer;
begin
    for i:=1 to MAX do
        t[i]:=val
end;
```

Tableau à une dimension 3 / 3

Affectation

- On peut affecter le contenu d'un tableau dans un autre tableau s'ils sont de même type

Exemple

```
function compterOccurences(chaineDeLettresMinuscules : String) :
    NbOccurences;
var i          : Integer;
   resultat    : NbOccurences;
   uneLettre   : Lettre;
begin
   initialiserNbOccurrences(resultat);
   for i:=1 to length(chaineDeLettresMinuscules) do
   begin
      uneLettre:=chaineDeLettresMinuscules[i];
      resultat[uneLettre]:=resultat[uneLettre]+1;
   end;
   compterOccurences:=resultat
end;
```

Tableau à 2 dimensions

Définition/Déclaration

- Il y a deux façons de déclarer des tableaux à deux dimensions :
  1. en considérant que l'on a un tableau de tableau

```
array [indiceDebut1..indiceFin1] of array[indiceDebut2..
    indiceFin2]
of TypeDesElements
```

  2. en spécifiant directement deux intervalles séparés par une , :

```
array [indiceDebut1..indiceFin1,indiceDebut2..indiceFin2]
     of TypeDesElements
```

Utilisation

- On utilise une case du tableau en spécifiant entre crochet les deux indices de cette case séparés par un ,

Architecture des Systèmes d'Information

Tableau à *n* dimensions 1 / 2

Extension

- Par extension on peut définir et utiliser des tableaux à *n* dimensions

Exemple à 3 dimensions

```
const
   LARGEUR     = 4;
   HAUTEUR     = 4;
   PROFONDEUR  = 4;

type
   Contenu = (Vide, PionBlanc, PionNoir);
   Plateau =  array[1..LARGEUR,
                    1..PROFONDEUR,
                    1..HAUTEUR] of Contenu;
```

Architecture des Systèmes d'Information

Tableau à *n* dimensions 2 / 2

Utilisation

```
procedure intialiserPlateau(var p : Plateau);
var i,j,k :  Integer;
begin
   for i:=1 to LARGEUR do
      for j:=1 to PROFONDEUR do
         for k:=1 to HAUTEUR do
            p[i,j,k]:=Vide
end;
```

Enregistrement 1 / 4

- Les enregistrements (*record*) Pascal correspondent aux structures en algorithmique

Définition/Déclaration

- Tout comme pour les tableaux, on utilise rarement des variable de type enregistrement, on définit plutôt des types
- Syntaxe :

```
record
  champ1 : Type1
  champ2 : Type2
  ...
end;
```

Exemple

```
type
   TComplexe = record
                  re : Real;
                  im : Real;
               end;
```

Enregistrement 2 / 4

Utilisation

- On accède au champ c d'une variable v de type enregistrement en faisant suivre la variable v d'un . puis du champ c ($v.c$)
  - si le champ c est aussi de type enregistrement, on peut accéder à ses champs (par exemple $c1$) depuis v suivant le même principe ($v.c.c1$)

Exemple

```
function creerComplexe(partieReel,partieImaginaire : Real) :
    TComplexe;
var resultat :  TComplexe;
begin
   resultat.re:=partieReel;
   resultat.im:=partieImaginaire;
   creerComplexe:=resultat
end; { creerComplexe }
```

Architecture des Systèmes d'Information

Enregistrement 3 / 4

Encapsulation

- Afin d'être facilement réutilisable, les utilisateurs ne doivent pas accéder directement aux champs de l'enregistrement

Exemple

```
function partieReelle(c : TComplexe) : Real;
begin
   partieReelle:=c.re
end; { partieReelle }

function partieImaginaire(c : TComplexe) : Real;
begin
   partieImaginaire:=c.im
end; { partieImaginaire }
```

Enregistrement 4 / 4

Enregistrement vers chaîne de caractères

- Comme pour les tableaux, on ne peut pas afficher directement un enregistrement
- On utilise souvent une fonction qui créé une chaîne de caractères à partir de l'enregistrement

Exemple

```
function complexeEnChaine(c : TComplexe) : String;
var re,im :  String;
begin
   str(partieReelle(c),re);
   str(partieImaginaire(c),im);
   complexeEnChaine:=re+'+'+im+'i'
end; { complexeEnChaine }
```

Architecture des Systèmes d'Information

with . . .do

- L'instruction *with* permet d'omettre le nom de la variable de type enregistrement, et donc accéder localement directement à ses champs

Exemple

```
function creerComplexe(partieReelle,partieImaginaire : Real) :
    TComplexe;
var resultat :  TComplexe;
begin
   with resultat do
   begin
      re:=partieReelle;
      im:=partieImaginaire;
   end;
   creerComplexe:=resultat
end; { creerComplexe }
```

Cahier des charges

« Développer un programme permettant à deux joueurs humains de jouer au jeu du puissance 4. L'interface homme machine sera en mode texte. »

Exemple d'interface homme machine (IHM)

```
$ ./puissance4Txt
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
---------------
 1 2 3 4 5 6 7
Au joueur qui a
les * de jouer :4
```

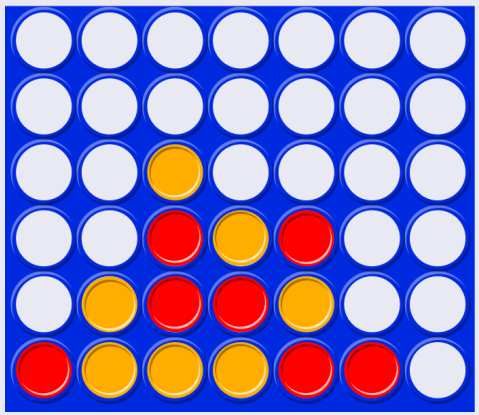
```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |*| | | |
---------------
 1 2 3 4 5 6 7
Au joueur qui a
les + de jouer :3
```

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |+|*| | | |
---------------
 1 2 3 4 5 6 7
Au joueur qui a
les * de jouer :5
```

Puissance 4

≪ Puissance 4 est un jeu de société combinatoire abstrait au tour par tour, édité pour la première fois en 1974 par MB (détenu de nos jours par Hasbro) et qui se joue à 2.

Le but est de faire une ligne de 4 pions sur une grille comptant 6 rangées et 7 colonnes, les parties durant en général une dizaine de minutes.



Chaque joueur dispose de 21 pions d'une couleur (jaune ou rouge). Tour à tour les joueurs posent un pion dans une colonne, le pion coulisse jusqu'à sa position la plus basse dans la colonne, et c'est à l'autre joueur de jouer.

Le vainqueur est le premier joueur qui aligne quatre pions de sa couleur verticalement, horizontale ou diagonalement. ≫(Wikipedia)

Architecture des Systèmes d'Information

Méthodologie

Étapes du projet

1. Analyse :
   - Identifier les types de données propres au projet ainsi que les opérations pour les utiliser
   - Faire une analyse descendante de la résolution du problème (`faireUnePartie`)
2. Conception préliminaire : donner les signatures des fonctions et procédures :
   - des opérations des types du projet
   - des opérations de l'analyse descendante
3. Conception détaillée :
   - Expliciter la conception des types de données
   - Expliciter les algorithmes des fonctions et procédures de la CP
4. Développement en Pascal

Les types de données et leurs opérations 1 / 2

- L'analyse du sujet fait apparaître les types de données suivants :
  - *Pion* qui est jaune ou rouge
  - *Contenu* d'une case d'un plateau qui est soit vide soit remplie par un pion
  - *Plateau* qui est un ensemble de case organisé en colonnes (type *Colonne*) et lignes (type *Ligne*)
  - *EtatPartie* l'état finale de la partie (partie gagnée ou partie nulle)

Opération de Pion

- obtenir l'autre joueur (*autreJoueur*)
  - Entrée : Pion
  - Sortie : Pion

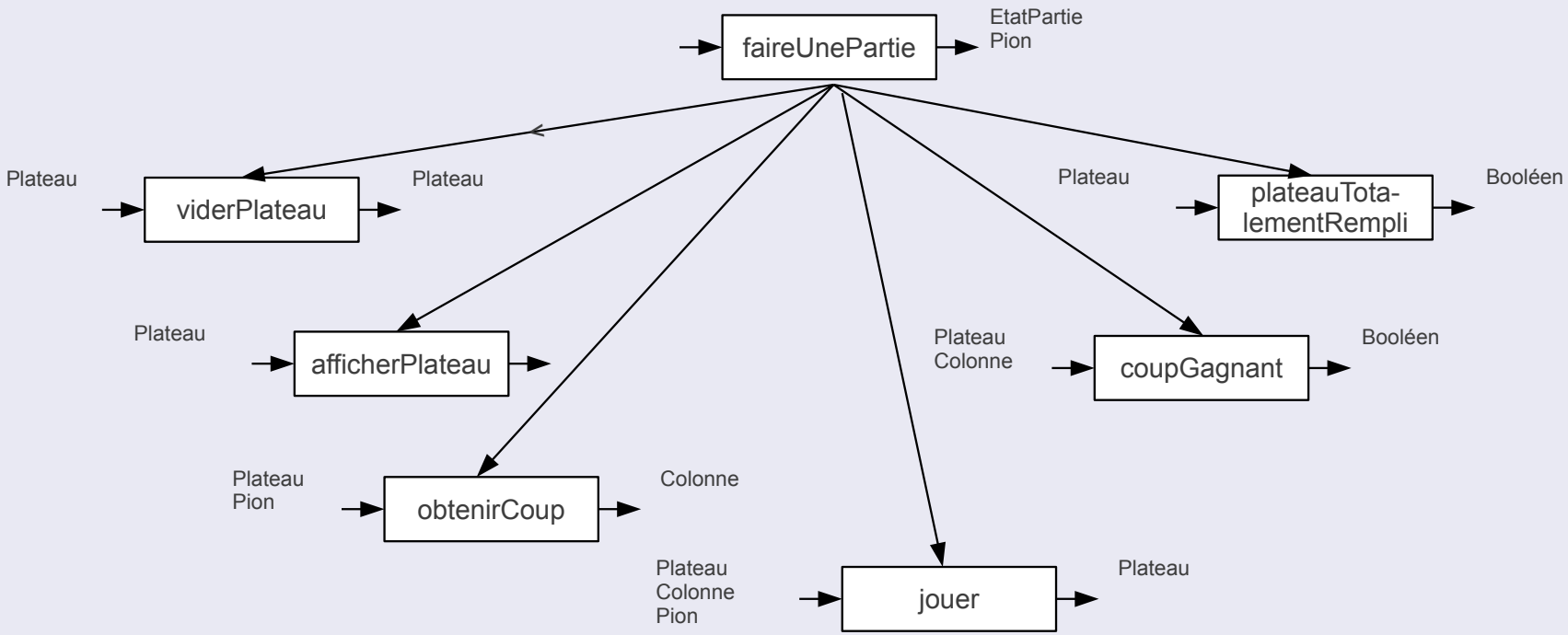
Les types de données et leurs opérations 2 / 2

Opérations de Plateau

- vider le plateau (*vider*)
  - Entrée : Plateau
  - Sortie : Plateau
- obtenir le contenu d'une case (*contenuCase*)
  - Entrée : Plateau, Colonne, Ligne
  - Sortie : Contenu
- obtenir la hauteur d'une colonne (*hauteurColonne*)
  - Entrée : Plateau, Colonne
  - Sortie : Naturel
- jouer un pion dans une colonne (*jouer*)
  - Entrée : Plateau, Colonne, Pion
  - Sortie : Plateau
- savoir si on peut jouer dans une colonne : colonne totalement rempli ? (*peutJouer*)
  - Entrée : Plateau, Colonne
  - Sortie : Booléen

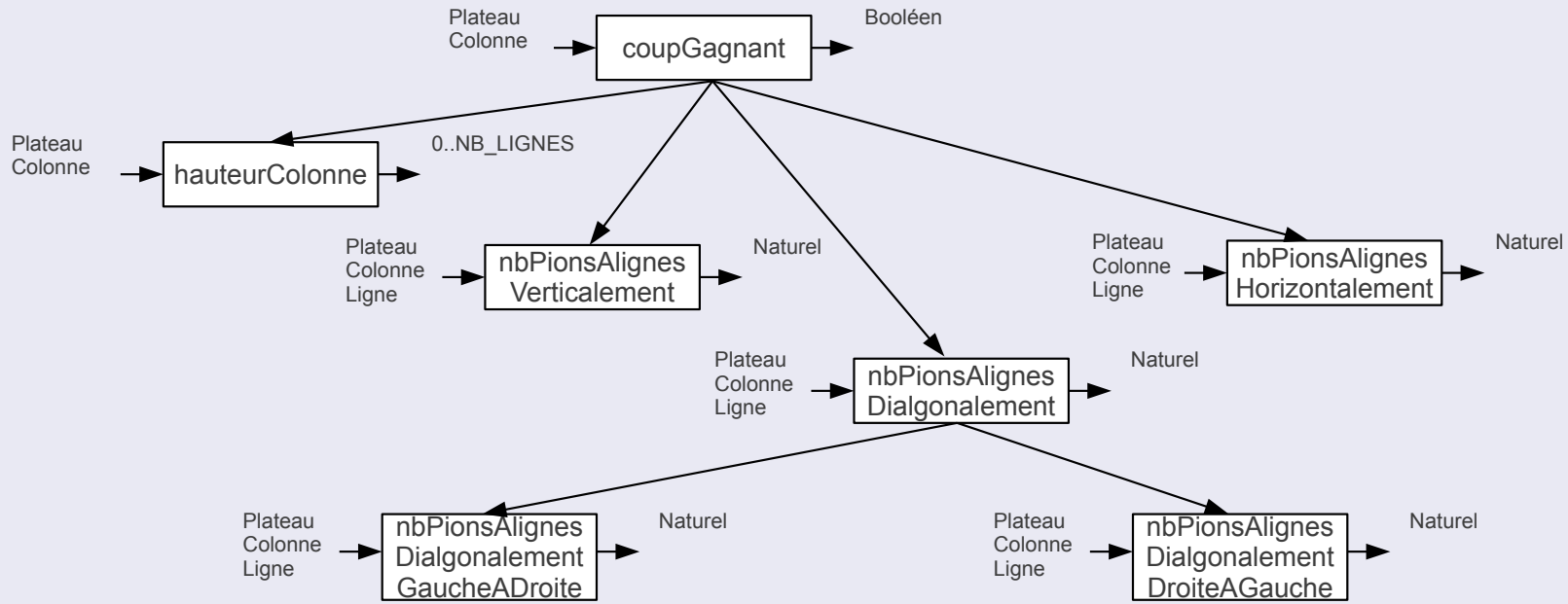
Analyse descendante 1 / 3

faire une partie



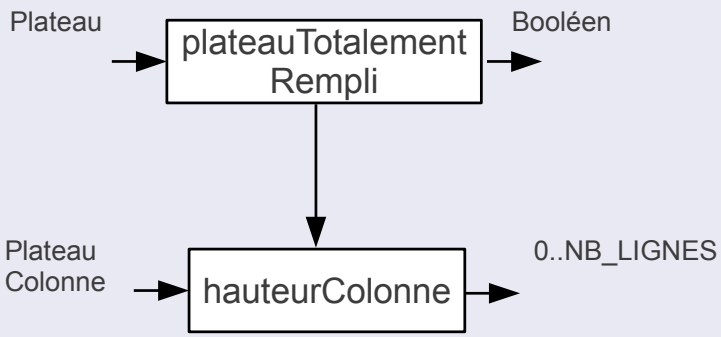
Analyse descendante 2 / 3

coup gagnant



Analyse descendante 3 / 3

totalementRempli



Conception préliminaire 1 / 2

Opérations de Pion

fonction autreJoueur **(**unJoueur : Pion**) :** Pion

Opérations de Plateau

procédure vider **(S** unPlateau : Plateau**)**
fonction contenuCase **(**unPlateau : Plateau, col : Colonne, lig : Ligne**) :** Contenu
fonction hauteurColonne **(**unPlateau : Plateau, col : Colonne**) :** 0..NB_LIGNES
procédure jouer **(E/S** unPlateau : Plateau,**E** col : Colonne, unPion : Pion**)**
fonction peutJouer **(**unPlateau : Plateau, col : Colonne**) : Booleen**

Conception préliminaire 2 / 2

Opérations pour jouer au puissance 4

procédure faireUnePartie **(E** afficher : AfficherPlateau , obtenirCoupJaune, obtenirCoupRouge : ObtenirCoup,**S** etatPartie : EtatPartie, quiAGagne : Pion**)**
fonction coupGagnant **(**unPlateau : Plateau, col : Colonne**) : Booleen**
fonction plateauTotalementRempli **(**unPlateau : Plateau**) : Booleen**
fonction nbPionsAlignesVerticalement **(**unPlateau : Plateau, col : Colonne, lig : Ligne**) : Naturel**

⌊**précondition(s)** $0 < lig$ et $lig \leq hauteurColonne(unPlateau, col)$

fonction nbPionsAlignesHorizontalement **(**unPlateau : Plateau, col : Colonne, lig : Ligne**) : Naturel**

⌊**précondition(s)** $0 < lig$ et $lig \leq hauteurColonne(unPlateau, col)$

fonction nbPionsAlignesDiagionalement **(**unPlateau : Plateau, col : Colonne, lig : Ligne**) : Naturel**

⌊**précondition(s)** $0 < lig$ et $lig \leq hauteurColonne(unPlateau, col)$

fonction nbPionsAlignesDiagionalementDroiteAGauche **(**unPlateau : Plateau, col : Colonne, lig : Ligne**) : Naturel**

⌊**précondition(s)** $0 < lig$ et $lig \leq hauteurColonne(unPlateau, col)$

fonction nbPionsAlignesDiagionalementGaucheADroite **(**unPlateau : Plateau, col : Colonne, lig : Ligne**) : Naturel**

⌊**précondition(s)** $0 < lig$ et $lig \leq hauteurColonne(unPlateau, col)$

Conception détaillée - Constantes et Types

Constantes

Constante NB_COLONNES = 7
Constante NB_LIGNES = 6
Constante NB_PIONS_ALIGNES_POUR_GAGNER = 4

Types

Type EtatPartie = {partieGagnee, partieNulle}
Type Contenu = {vide,pionJaune,pionRouge}
Type Pion = pionJaune..pionRouge
Type Colonne = 1..NB_COLONNES
Type Ligne = 1..NB_LIGNES
Type Plateau = **Structure**
    contenu : **Tableau[**1..NB_COLONNES**][**1..NB_LIGNES**] de** Contenu
    hauteurColonne : **Tableau[**1..NB_COLONNES**] de Naturel**
finstructure

Conception détaillée - Opérations sur les types

Opérations du type Plateau

procédure vider **(S** lePlateau : Plateau**)**

```
    Déclaration   i : Colonne
                  j : Ligne
debut
    pour i ←1 à NB_COLONNES faire
        lePlateau.hauteurColonne[i] ← 0
        pour j ←1 à NB_LIGNES faire
            lePlateau.contenu[i][j] ← vide
        finpour
    finpour
fin
```

Exercices

Donnez les algorithmes des fonctions et procédures : `contenuCase`, `hauteurColonne`, `jouer`, `peutJouer`

Conception détaillée - Puissance 4 1 / 3

faireUnePartie

```
procédure faireUnePartie (S etatPartie : EtatPartie, quiAGagne : Pion)
    Déclaration  lePlateau : Plateau, joueurCourant : Pion, coup : Colonne
debut
    vider(lePlateau)
    joueurCourant ← pionJaune
    afficher(lePlateau)
    repeter
        coup ← obtenirCoup(lePlateau,joueurCourant)
        jouer(lePlateau,coup,joueurCourant)
        afficher(lePlateau)
        joueurCourant ← autreJoueur(joueurCourant)
    jusqu'a ce que coupGagnant(lePlateau,coup) ou totalementRempli(lePlateau)
    si coupGagnant(lePlateau,coup) alors
        etatPartie ← partieGagnee
        quiAGagne ← autreJoueur(joueurCourant)
    sinon
        etatPartie ← partieNulle
    finsi
fin
```

Conception détaillée - Puissance 4 2 / 3

totalementRempli

fonction plateauTotalementRempli **(**unPlateau : Plateau**) : Booleen**

Déclaration resultat : **Booleen**, i : Colonne

debut

i ← 1

resultat ← Vrai

tant que resultat et i≤NB_COLONNES **faire**

resultat ← hauteurColonne(unPlateau,i)=NB_LIGNES

i ← i+1

fintantque

retourner resultat

fin

Conception détaillée - Puissance 4 3 / 3

coupGagnant

fonction coupGagnant **(**unPlateau : Plateau, col : Colonne**) : Booleen**
debut
    retourner nbPionsAlignesVerticalement(unPlateau,col,hauteurColonne(unPlateau,col))
        ≥NB_PIONS_ALIGNES_POUR_GAGNER
    ou nbPionsAlignesHorizontalement(unPlateau,col,hauteurColonne(unPlateau,col))
        ≥NB_PIONS_ALIGNES_POUR_GAGNER
    ou nbPionsAlignesDiagionalement(unPlateau,col,hauteurColonne(unPlateau,col))
        ≥NB_PIONS_ALIGNES_POUR_GAGNER
fin

Développement en Pascal 1 / 9

Renomage des types

- Préfixer chaque type par un T

Organisation du programme

- Déclarer les constantes et types
- Déclarer les fonctions et procédures (utilisation du mot clé forward) pour éviter les utilisations avant les déclarations
- Définir les fonctions et procédures

Méthodologie de développement

- Commencer par coder les opérations de bases des types, puis les opérations de l'analyse descendante (du bas vers le haut)
- Après chaque codage d'une fonction ou procédure, on compile

Développement en Pascal 2 / 9

Déclaration des types et constantes

```
1  program Puissance4Txt;
2
3  const
4     NB_LIGNES                    = 6;
5     NB_COLONNES                  = 7;
6     NB_PIONS_ALIGNES_POUR_GAGNER = 4;
7
8  type
9     TContenu = (vide,pionJaune,pionRouge);
10    TPion    = pionJaune..pionRouge;
11    TColonne = 1..NB_COLONNES;
12    TLigne   = 1..NB_LIGNES;
13    TPlateau = record
14                  contenu        : array[1..NB_COLONNES,1..NB_LIGNES] of TContenu;
15                  hauteurColonne : array[1..NB_COLONNES] of Integer;
16               end;
17    TEtatPartie  =  (partie_gagnee,partie_nulle);
18
19
20 function obtenirCoupHumain(lePlateau : TPlateau; joueur : TPion) : TColonne;forward;
21 procedure afficher(lePlateau :  TPlateau);forward;
```

Développement en Pascal 3 / 9

Déclaration des fonctions et procédures

```
25 function autreJoueur(joueur :  TPion) : TPion;
26 begin
27    case joueur of
28      pionJaune : autreJoueur:=pionRouge;
29      pionRouge : autreJoueur:=pionJaune;
30    end { case }
31 end;
32
33 procedure vider(var lePlateau : TPlateau);
34 var
35    i : TColonne;
36    j : TLigne;
37 begin
38    for i:=1 to NB_COLONNES do
39    begin
40       lePlateau.hauteurColonne[i]:=0;
41       for j:=1 to NB_LIGNES do
42          lePlateau.contenu[i,j]:=vide;
43    end
44 end; { vider }
45
46 function contenuCase(lePlateau : TPlateau; colonne : TColonne; ligne : TColonne) :
      TContenu;
47 begin
48    contenuCase:=lePlateau.contenu[colonne,ligne]
49 end; { contenuCase }
```

Développement en Pascal 4 / 9

Fonctions et procédures de la logique métier

```
51 function hauteurColonne(lePlateau : TPlateau; colonne : TColonne) : Integer;
52 begin
53    hauteurColonne:=lePlateau.hauteurColonne[colonne]
54 end; { hauteurColonne }
55
56 procedure jouer(var lePlateau : TPlateau; colonne : TColonne; pion : TPion);
57 begin
58    inc(lePlateau.hauteurColonne[colonne]);
59    lePlateau.contenu[colonne,lePlateau.hauteurColonne[colonne]]:=pion
60 end; { jouer }
61
62 function peutJouer(lePlateau : TPlateau; colonne : TColonne) : Boolean;
63 begin
64    peutJouer:=hauteurColonne(lePlateau,colonne)<NB_LIGNES
65 end; { peutJouer }
66
67 function plateauTotalementRempli(unPlateau : TPlateau ) : Boolean;
68 var i : TColonne;
69 begin
70    i:=1;
71    plateauTotalementRempli:=True;
72    while plateauTotalementRempli and (i<=NB_COLONNES) do
73    begin
74       plateauTotalementRempli:=hauteurColonne(unPlateau,i)=NB_LIGNES;
75       inc(i)
76    end
77 end; { plateauTotalementRempli }
```

Développement en Pascal 5 / 9

Fonctions et procédures de la logique métier

```
79 function nombrePionsAlignesVerticalement(unPlateau : TPlateau; colonne : TColonne;
     ligne : TLigne) : Integer;
80   function nbPionsAlignesVerticalement(unPlateau : TPlateau; pionRef : TPion; colonne,
       ligne,direction,borne : Integer) : Integer;
81   var
82     resultat,i : Integer;
83   begin
84     i:=ligne;
85     resultat:=0;
86     while (i<>borne) and (contenuCase(unPlateau,colonne,i)=pionRef) do
87     begin
88       i:=i+direction;
89       inc(resultat);
90     end;
91     nbPionsAlignesVerticalement:=resultat-1;
92   end; { nbPionsAlignesVerticalement }
93 begin
94   nombrePionsAlignesVerticalement:=nbPionsAlignesVerticalement(unPlateau,contenuCase(
       unPlateau,colonne,ligne),colonne,ligne,-1,0)
95   +nbPionsAlignesVerticalement(unPlateau,contenuCase(unPlateau,colonne,ligne),colonne,
       ligne,1,NB_LIGNES+1)+1;
96 end; { nombrePionsAlignesVerticalement }
```

Architecture des Systèmes d'Information

Développement en Pascal 6 / 9

Fonctions et procédures de la logique métier

```
159 function coupGagnant(unPlateau : TPlateau; colonne : TColonne) : Boolean;
160 begin
161     coupGagnant:=(nombrePionsAlignesVerticalement(unPlateau,colonne,hauteurColonne(
            unPlateau,colonne))=NB_PIONS_ALIGNES_POUR_GAGNER)
162     or (nombrePionsAlignesHorizontalement(unPlateau,colonne,hauteurColonne(unPlateau,
            colonne))=NB_PIONS_ALIGNES_POUR_GAGNER)
163     or (nombrePionsAlignesDiagonalementGaucheADroite(unPlateau,colonne,hauteurColonne(
            unPlateau,colonne))=NB_PIONS_ALIGNES_POUR_GAGNER)
164     or (nombrePionsAlignesDiagonalementDroiteAGauche(unPlateau,colonne,hauteurColonne(
            unPlateau,colonne))=NB_PIONS_ALIGNES_POUR_GAGNER);
165 end; { coupGagnant }
```

Architecture des Systèmes d'Information

Développement en Pascal 7 / 9

Fonctions et procédures de la logique métier

```
167 procedure faireUnePartie(var etatPartie : TEtatPartie; quiAGagne : TPion);
168 var
169    lePlateau     : TPlateau;
170    joueurCourant : TPion;
171    coup          : TColonne;
172 begin
173    vider(lePlateau);
174    joueurCourant:=pionJaune;
175    afficher(lePlateau);
176    repeat
177       coup:=obtenirCoupHumain(lePlateau,joueurCourant);
178       jouer(lePlateau,coup,joueurCourant);
179       afficher(lePlateau);
180       joueurCourant:=autreJoueur(joueurCourant);
181    until coupGagnant(lePlateau,coup) or plateauTotalementRempli(lePlateau);
182    if coupGagnant(lePlateau,coup) then
183    begin
184       etatPartie:=partie_gagnee;
185       quiAGagne:=autreJoueur(joueurCourant);
186    end else
187       etatPartie:=partie_nulle
188 end; { jouer }
```

Architecture des Systèmes d'Information

Développement en Pascal 8 / 9

Fonctions et procédures de l'IHM

```
199 procedure afficher(lePlateau :  TPlateau);
200 var
201    i : TColonne;
202    j : TLigne;
203    c : String;
204 begin
205    for j:=NB_LIGNES downto 1 do
206    begin
207       for i:=1 to NB_COLONNES do
208       begin
209          write('|');
210          if contenuCase(lePlateau,i,j)=vide then
211             write('␣')
212          else
213             write(pionEnChaine(contenuCase(lePlateau,i,j)));
214       end;
215       writeln('|');
216    end;
217    for i:=1 to  NB_COLONNES do
218       write('--');
219    writeln('-');
220    for i:=1 to  NB_COLONNES do
221    begin
222       str(i,c);
223       write('␣'+c);
224    end;
225    writeln()
226 end; { afficher }
```

Développement en Pascal 9 / 9

Programme principal

```
246 var
247    etatFinale : TEtatPartie;
248    quiAGagne  : TPion;
249 begin
250    faireUnePartie(etatFinale,quiAGagne);
251    if etatFinale=partie_gagnee then
252    begin
253       if quiAGagne=pionJaune then
254          writeln('Vous␣avez␣gagne␣!!')
255       else
256          writeln('J''ai␣gagne␣!!')
257    end else
258       writeln('Partie␣nulle')
259 end.
```

Conclusion

Critères d'un bon programme

lisible	✗ ✓
fiable	✗ ✓
maintenable	✓
réutilisable	✗
portable	✗
correct (preuve)	✗
efficace (complexité)	✓
faire face à des contraintes "économiques"	✗

Architecture des Systèmes d'Information