

Puissance 4 intelligent I3 Algorithmique

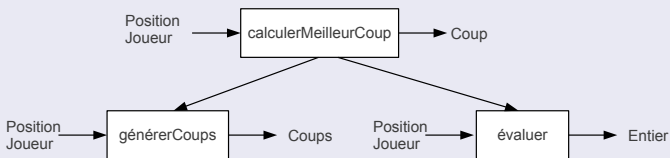
Nicolas Delestre

Plan

- 1 Force brute
- 2 Analyse
- 3 Conception préliminaire
- 4 Conception détaillée
- 5 Développement
- 6 Conclusion

Force brute

- La recherche de solution en force brute (ou recherche exhaustive) est l'une des méthodes utilisée en informatique pour faire jouer des ordinateurs
- Elle utilise trois algorithmes
 - Un générateur de coup qui pour une position et un joueur donnés est capable de lister l'ensemble des coups possibles
 - Une fonction d'évaluation qui pour une position et un joueur donnés est capable de calculer qui a l'avantage
 - Une algorithme de sélection de coup qui pour une position et un joueur donnés est capable de calculer le meilleur coup pour ce joueur. L'algorithmes le plus connu est celui du MinMax.



L'algorithme min-max 1 / 6

Principe

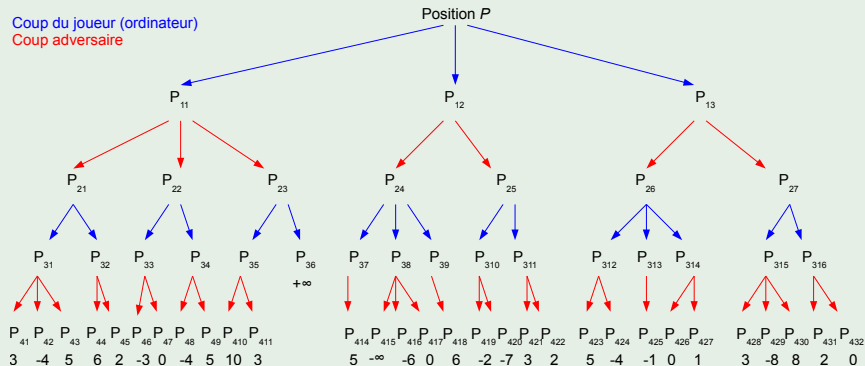
- La fonction d'évaluation retourne un entier (ou un réel) tel que :
 - le signe indique qui a l'avantage (positif pour celui pour qui on calcule le meilleur coup, négatif pour l'autre)
 - la valeur absolue indique l'importance de cet avantage
- Pour avoir l'avantage, il faut maximiser son score
- On considère que l'adversaire joue au mieux, c'est-à-dire qu'il essaye de maximiser **son** score, c'est-à-dire minimiser celui pour qui on cherche à calculer le meilleur coup

L'algorithme min-max 2 / 6

Principe : générateur de coups et fonction d'évaluation

Coup du joueur (ordinateur)

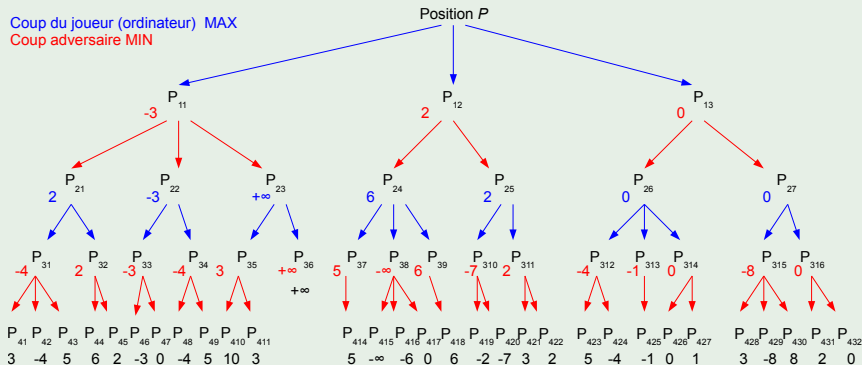
Coup adverse



L'algorithme min-max 3 / 6

Remonter des scores (min, max)

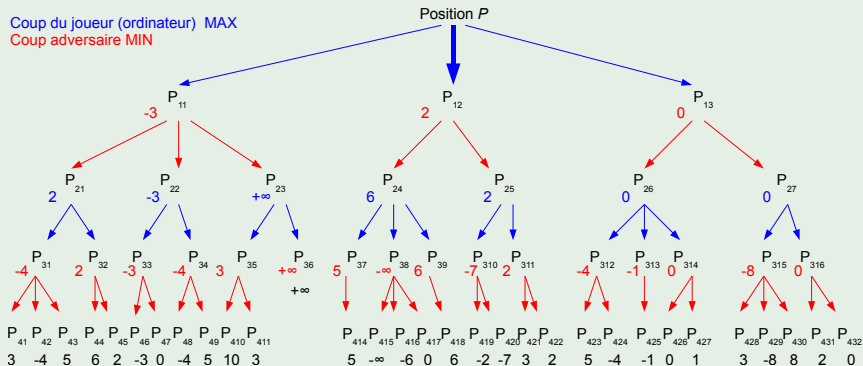
Coup du joueur (ordinateur) MAX
 Coup adverse MIN



L'algorithme min-max 4 / 6

Choix du coup

Coup du joueur (ordinateur) MAX
Coup adverse MIN



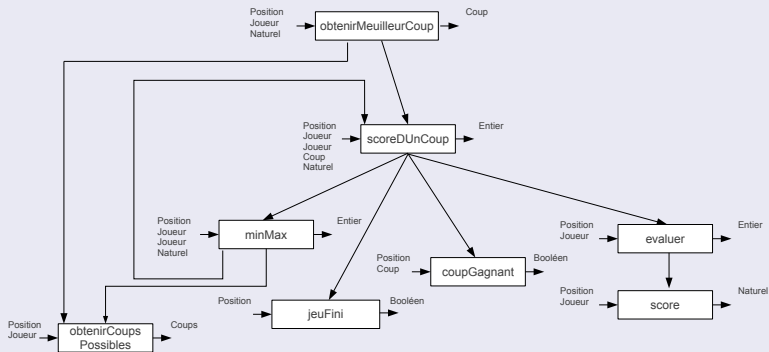
L'algorithme min-max 5 / 6

Synthèse

- Le sous-programme de *choix d'un coup* a besoin :
 - d'un *générateur de coups*
 - d'un *calcul de score* d'un coup
- Le sous-programme de *calcul de score* a besoin :
 - d'une *fonction d'évaluation*
 - de l'algorithme *min-max*
- Le sous-programme *min-max* a besoin :
 - d'une *générateur de coups*
 - d'un *calcul de score* d'un coup

L'algorithme min-max 6 / 6

Analyse descendante



Force brute pour le puissance 4

Générateur de coups

- Il suffit de répertorier toutes les colonnes du plateau qui ne sont pas totalement remplies

Fonction d'évaluation (version très simple)

- Faire la somme du nombre de pions alignés multipliée par un coefficient pour chaque joueur :
 - 1 pour un alignement d'un pion
 - 5 pour un alignement de deux pions
 - 50 pour un alignements de trois pions
 - 1000 pour un alignements de quatre pions
- Faire la différence des scores des deux joueurs

Les types de données et leurs opérations 1 / 2

- Nous avons les types :
 - *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 et lignes
 - *EtatPartie* l'état finale de la partie (partie gagnée ou partie nulle)
- Auxquels on ajoute
 - **Coups** qui un ensemble de coups, avec un coup qui est le numéro d'une colonne du plateau

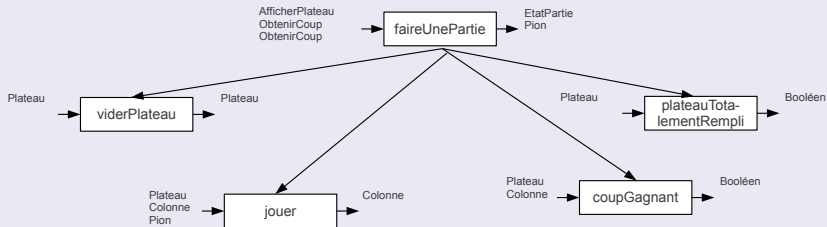
Les types de données et leurs opérations 2 / 2

Opérations de Coups

- obtenir un ensemble de coups vide (*coups*)
 - Sortie : Coups
- obtenir le nb de coups (*nb*)
 - Entrée : Coups
 - Sortie : Naturel
- ajouter un coups (*ajouter*)
 - Entrée : Coups, Colonne
 - Sortie : Coups
- obtenir le *i*ème coups (*ieme*)
 - Entrée : Coups, Naturel
 - Sortie : Colonne

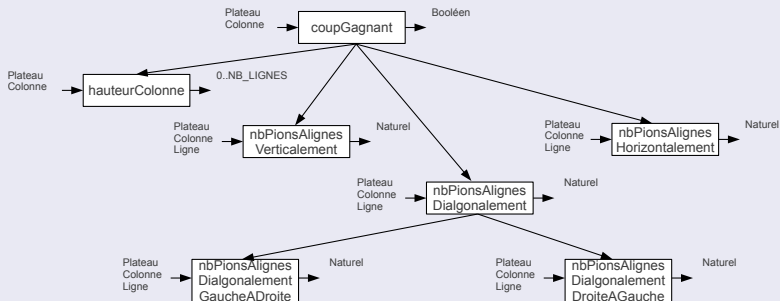
Analyse descendante (rappel) 1 / 3

faire une partie



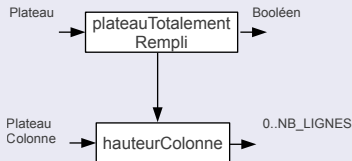
Analyse descendante (rappel) 2 / 3

coup gagnant



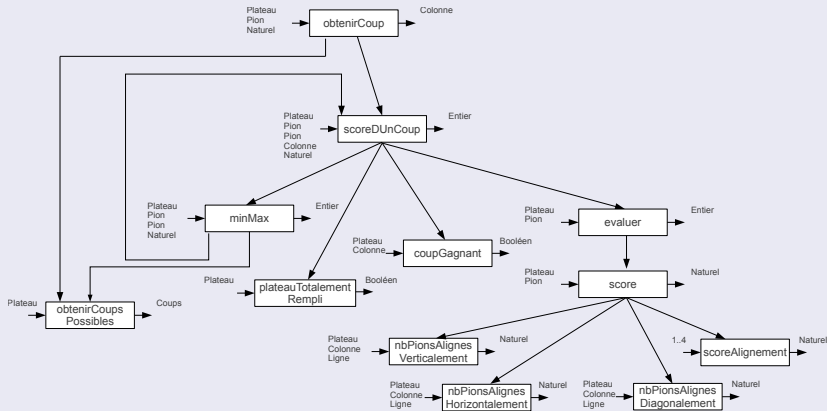
Analyse descendante (rappel) 3 / 3

totalementRempli



Nouvelle analyse descendante pour l'IA

obtenirCoup



Conception préliminaire, nouvelles fonctions / procédures

1 / 2

Opérations de Coups

fonction coups () : Coups

fonction nb (cps : Coups) : **Naturel**

procédure ajouter (**E/S** cps : Coups, **E** col : Colonne)

fonction ieme (cps : Coups, ieme : **Naturel**) : Colonne

|**précondition(s)** $0 < ieme$ et $ieme \leq nb(cps)$

Conception préliminaire, nouvelles fonctions / procédures

2 / 2

Opérations pour faire jouer l'ordinateur au puissance 4 (IA)

fonction obtenirCoup (unPlateau : plateau, joueur : Pion, profondeur : **Naturel**) : Colonne

 |précondition(s) *non plateauTotalementRempli(unPlateau)*

fonction obtenirCoupsPossibles (unPlateau : plateau) : Coups

fonction scoreDUnCoup (unPlateau : plateau, joueurRef,joueurCourant : Pion, unCoup : Colonne, profondeur : **Naturel**) : **Entier**

fonction minmax (unPlateau : plateau, joueurRef,joueurCourant : Pion, profondeur : **Naturel**) : **Entier**

fonction evaluer (unPlateau : plateau, joueurRef : Pion) : **Entier**

fonction score (unPlateau : plateau, joueur : Pion) : **Entier**

Conception détaillée - Coups 1 / 2

Type Coups

Type Coups = Structure

lesCoups : **Tableau**[1..NB_COLONNES] de Colonne

nbCoups : **Naturel**

finstructure

Opérations du type Coups

fonction coups () : Coups

Déclaration resultat : Coups

debut

resultat.nbCoups ← 0

retourner resultat

fin

fonction nb (cps : Coups) : Naturel

debut

retourner cps.nbCoups

fin

Conception détaillée - Coups 2 / 2

Opérations du type Coups (suite)

procédure ajouterCoup (**E/S** cps : Coups, **E** cp : Colonne)

debut

 cps.nbCoups \leftarrow cps.nbCoups+1

 cps.lesCoups[cps.nbCoups] \leftarrow cp

fin

fonction ieme (cps : Coups, i : **Naturel**) : Colonne

 | **précondition(s)** $0 < ieme$ et $ieme \leq nb(cps)$

debut

retourner cps.lesCoups[i]

fin

Conception détaillée - IA 1 / 6

obtenirCoup

fonction obtenirCoup (unPlateau : Plateau, joueur : Pion, profondeur : **Naturel**) : Colonne

Déclaration resultat : Colonne, cps : Coups, score, meilleurScore : **Entier**, i : **Naturel**

debut

cps ← obtenirCoupsPossibles(unPlateau)

resultat ← ieme(cps,1)

meilleurScore ← scoreDUnCoup(unPlateau,resultat,joueur,joueur,profondeur)

pour i ← 2 à nb(cps) **faire**

 score ← scoreDUnCoup(unPlateau,ieme(cps,i),joueur,joueur,profondeur)

si score > meilleurScore **alors**

 resultat ← ieme(cps,i)

 meilleurScore ← score

finsi

finpour

retourner resultat

fin

Conception détaillée - IA 2 / 6

obtenirCoupsPossibles

fonction obtenirCoupsPossibles (unPlateau : Plateau) : Coups

| **précondition(s)** *non plateauTotalementRempli(unPlateau)*

Déclaration i : Naturel

resultat : Coups

debut

resultat ← coups()

pour i ← 1 à NB_COLONNES **faire**

si hauteurColonne(unPlateau,i) < NB_LIGNES **alors**

ajouter(resultat,i)

finsi

finpour

retourner resultat

fin

Conception détaillée - IA 3 / 6

scoreDUnCoup

fonction scoreDUnCoup (unPlateau : Plateau, unCoup : Colonne, joueurRef,joueurCourant : Pion, profondeur : Naturel) : Entier

debut

 jouer(unPlateau,unCoup,joueurCourant)

si plateauTotaletementRempli(unPlateau) ou coupGagnant(unPlateau,unCoup) ou
 profondeur=0 **alors**

retourner evaluer(unPlateau,joueurRef)

sinon

retourner minMax(unPlateau,joueurRef,autreJoueur(joueurCourant),profondeur-1)

finsi

fin

Conception détaillée - IA 4 / 6

minMax

fonction minMax (unPlateau : Plateau, joueurRef,joueurCourant : Pion, profondeur : **Naturel**)
: **Entier**

Déclaration resultat : **Entier**, cps : Coups, score : **Entier**, i : **Naturel**

debut

cps ← obtenirCoupsPossibles(unPlateau)

resultat ← scoreDUnCoup(unPlateau,ieme(cps,1),joueurRef,joueurCourant,profondeur)

pour i ← 2 à nb(cps) **faire**

score ← scoreDUnCoup(unPlateau,ieme(cps,i),joueurRef,joueurCourant,profondeur)

si joueurCourant=joueurRef **alors**

resultat ← max(resultat,score)

sinon

resultat ← min(resultat,score)

finsi

finpour

retourner resultat

fin

Conception détaillée - IA 5 / 6

evaluer

```
fonction evaluer (unPlateau : Plateau, joueurRef : Pion) : Entier
debut
    retourner score(unPlateau,joueurRef)-score(unPlateau,autreJoueur(joueurRef))
fin
```

score

```
fonction scoreAlignement (nbPionsAlignes : 1..4) : Entier
    Déclaration resultat : Entier
debut
    cas où nbPionsAlignes vaut
        1:
            resultat ← 1
        2:
            resultat ← 5
        3:
            resultat ← 50
        4:
            resultat ← 1000
    fincas
    retourner resultat
fin
```

Conception détaillée - IA 6 / 6

score

fonction score (unPlateau : Plateau, joueur : Pion) : Entier

Déclaration resultat : Entier, i : Colonne, j : Ligne

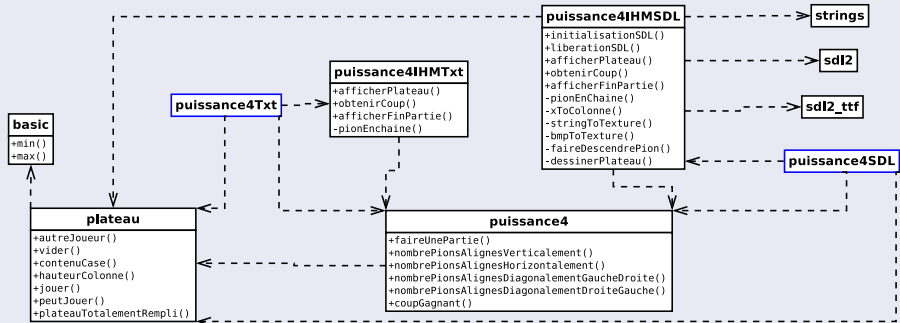
```

debut
  resultat ← 0
  pour i ← 1 à NB_COLONNES faire
    pour j ← 1 à NB_LIGNES faire
      si contenuCase(unPlateau,i,j)=joueurRef alors
        resultat ← resultat+
          scoreAlignement(nbPionsAlignesVerticalement(unPlateau,i,j))
        resultat ← resultat+
          scoreAlignement(nbPionsAlignesHorizontalement(unPlateau,i,j))
        resultat ← resultat+
          scoreAlignement(nbPionsAlignesDiagonalementGaucheADroite(unPlateau,i,j))
        resultat ← resultat+
          scoreAlignement(nbPionsAlignesDiagonalementDroiteAGauche(unPlateau,i,j))
      finsi
    finpour
  retourner resultat
fin

```

Développement en Pascal 1 / 2

Ancien diagramme d'unités - Texte et graphique



Conclusion

Conclusion

- Sans la méthodologie du cycle en V :
 - Il aurait été difficile d'atteindre le résultat
 - Il aurait été impossible de séparer le travail
- Ce qu'il reste à faire pour avoir un bon programme
 - Compléter la documentation du code
 - Finir les tests unitaires
 - Améliorer :
 - la fonction d'évaluation (prendre en compte qu'une suite de pions non entourés compte plus qu'une suite de pions entourés)
 - les performances : élagage de l'arbre de récursion (algorithme $\alpha - \beta$)