Exemple de conception

les jeux à 2 joueurs

Conception Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université Lille1

Algorithme min-MAX

- un exemple de conception sur un algorithme “générique”

Contexte :

- jeux à deux joueurs à information complète et à somme nulle
  ↪ dames, puissance 4, échecs, othello, etc.
- les joueurs jouent un coup chacun leur tour,
- on veut faire jouer le programme,

Présentation du problème

Le déroulement des parties d'un jeu peut être représenté par un arbre de jeu :

- racine = situation initiale,
- nœuds = une situation légale du jeu
- fils d'un nœud = situations filles, c-à-d que l'on peut atteindre à partir de ce nœud en respectant les règles du jeu
- une branche = une séquence de coups (légaux) dans une partie

Faire jouer le programme :

- choisir parmi les fils de la racine le "meilleur" coup à jouer
- cela se fait en anticipant sur le devenir des situations (pas de calcul à court terme) : "on calcule des coups en avant"

Le min-MAX

- Il faut disposer d'un fonction numérique ϕ qui attribue une valeur numérique à une situation de jeu

  $$\phi : \textit{Situation} \mapsto \textit{Nombre}$$

  cette fonction est **croissante** avec la qualité de la situation.
- Le but du programme est de maximiser la situation atteignable.
  Le but de son adversaire est inverse : il cherche à minimiser la situation atteignable
- Il faut tenir compte du jeu de l'adversaire dans le calcul de l'anticipation des coups joués
  - ↪ le programme suppose que l'adversaire joue au mieux
  - ↪ et donc qu'il joue **comme le programme** (avec un objectif inverse)

L'algorithme

`minmax(situation, profondeur, joueur)`

situation la situation courante (racine de l'arbre de jeu)

profondeur le nombre de coups d'anticipation

joueur `min` ou `MAX`

appel : `minmax(situation_initiale,profondeur,MAX)`

```
minmax(situation, prof, joueur) =
   si prof = 0 ou situation est terminale
      retourner φ(situation)
   sinon
      Fils = les situations filles de situation
      si joueur = MAX
         retourner max_{f∈Fils} {minmax(f, prof − 1, min)}
      sinon                                    // joueur = min
         retourner min_{f∈Fils} {minmax(f, prof − 1, MAX)}
      fin si
   fin si
```

Remarques

- effet d'horizon (si arbre développé partiellement)
- algorithme exponentiel
  $\hookrightarrow$ peut être amélioré en pratique par l'algorithme $\alpha\beta$

- tel quel retourne la valeur espérée de la meilleure situation fille, doit être adapté pour retourner la situation
- construction de l'arbre de jeu implicite

- algorithme générique :
  $\hookrightarrow$ pour tout jeu à deux joueurs à information complète
- difficulté (la seule) = fournir une fonction d'évaluation ϕ pour le jeu concerné
  $\hookrightarrow$ il existe plusieurs fonctions pour un même jeu ($\Rightarrow$ différentes "stratégies" de jeu)

Décomposition...

Les notions mises en œuvre :

- Jeu
- Situations (de jeu)
- Joueurs
- Fonctions d'évaluation
- L'algorithme minMax (et des joueurs qui l'utilisent)
  $\hookrightarrow$ indépendant d'un jeu particulier

Analyse

La mise en place de l'algorithme fait naturellement apparaître des fonctionnalités.
$\Longrightarrow$ méthodes (abstraites ?), attributs

Extraits

Pour jouer à un jeu, il faut...

- connaître la situation initiale et les 2 joueurs
- déterminer qui commence

TantQue le jeu n'est pas terminé

- le joueur dont c'est le tour joue
- on obtient ainsi une nouvelle situation de jeu
- c'est alors à l'autre joueur de jouer

finTantQue

- déterminer qui est le vainqueur.

ce qui pourrait donner qque chose comme... (...dans une classe `TwoPlayerGames`)

```
public void play(Situation starting, Player player1, Player player2) {
  this.setCurrentSituation(starting);
  this.player1 = player1;
  this.player2 = player2;
  this.currentPlayer = player1;

  while (!this.isFinalSituation(this.currentSituation,this.currentPlayer))
    this.display();
    Situation playerChoice = this.currentPlayer.play(this);
    this.setCurrentSituation(playerChoice);
    this.changePlayer();
  }

  this.display();
  Player winner = this.getWinner(currentPlayer);
  System.out.println("**************** Winner is "+winner);
}
```

le minMAX ressemble à ....

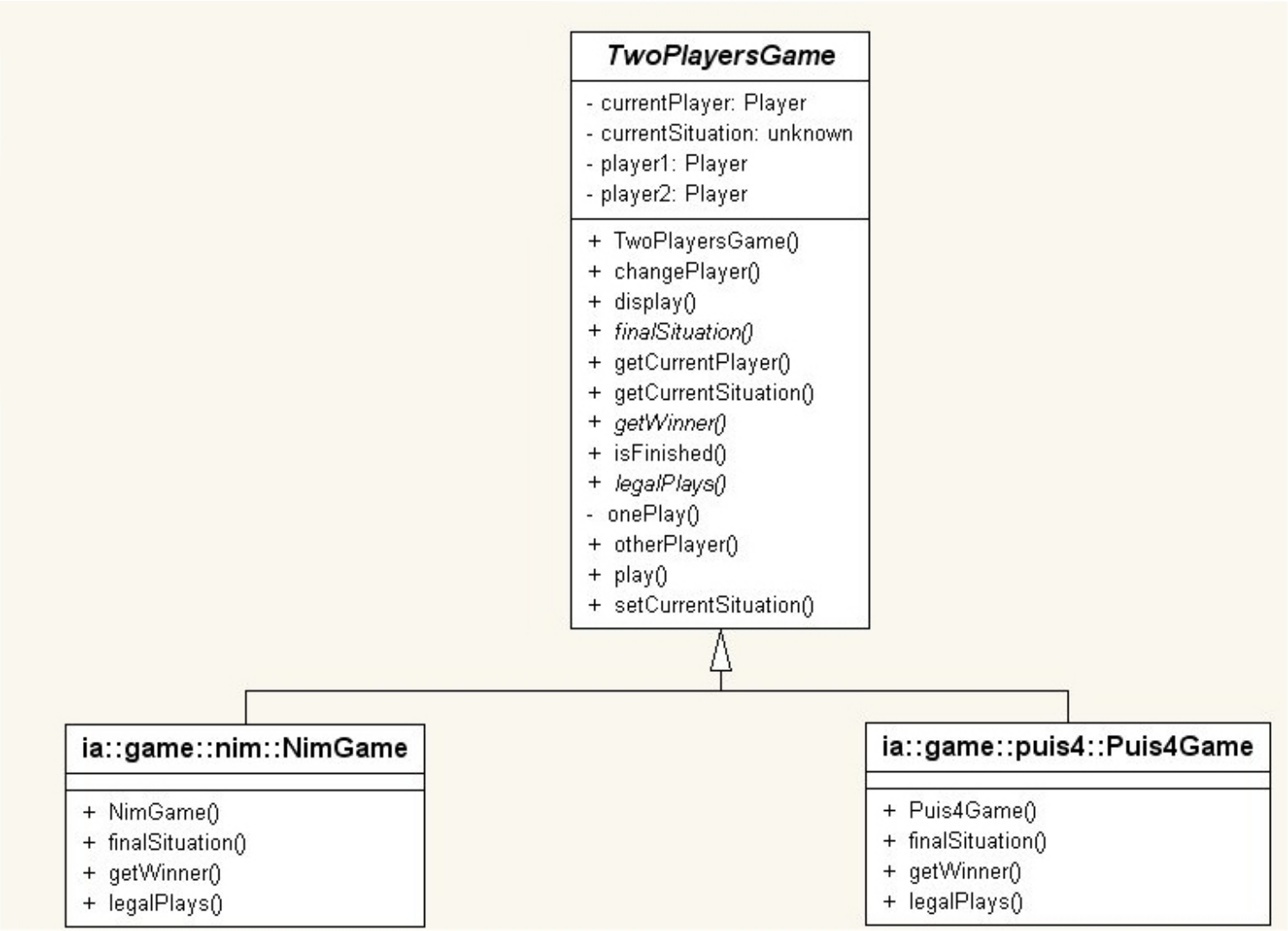
dans la classe MinMax

```
public int minmax(Situation situation, int depth, Player player) {
  if (depth == 0 || this.game.finalSituation(situation, player)) {
    return MAX.evaluation(situation, player) * sign(player);
  }
  else {
    Iterator<Situation> itPlays = this.game.legalPlays(situation, player).iterator();
    Collection<Integer> values = new ArrayList<Integer>();
    if (player == MAX) {
      while (itPlays.hasNext()) {
        Situation nouvelleSituation = itPlays.next();
        values.add(minmax(nouvelleSituation, depth - 1, MIN));
      }
      return Collections.max(values);
    }
    else { //player == MIN
      while (itPlays.hasNext()) {
        values.add(minmax(itPlays.next(), depth - 1, MAX));
      }
      return minValue = Collections.min(values);
    }   } }
```

- `MAX` est une instance de type `MinMaxPlayer` qui est un cas particulier de `Player`, qui possède une fonction d'évaluation.
- `MIN` est de type `Player`.

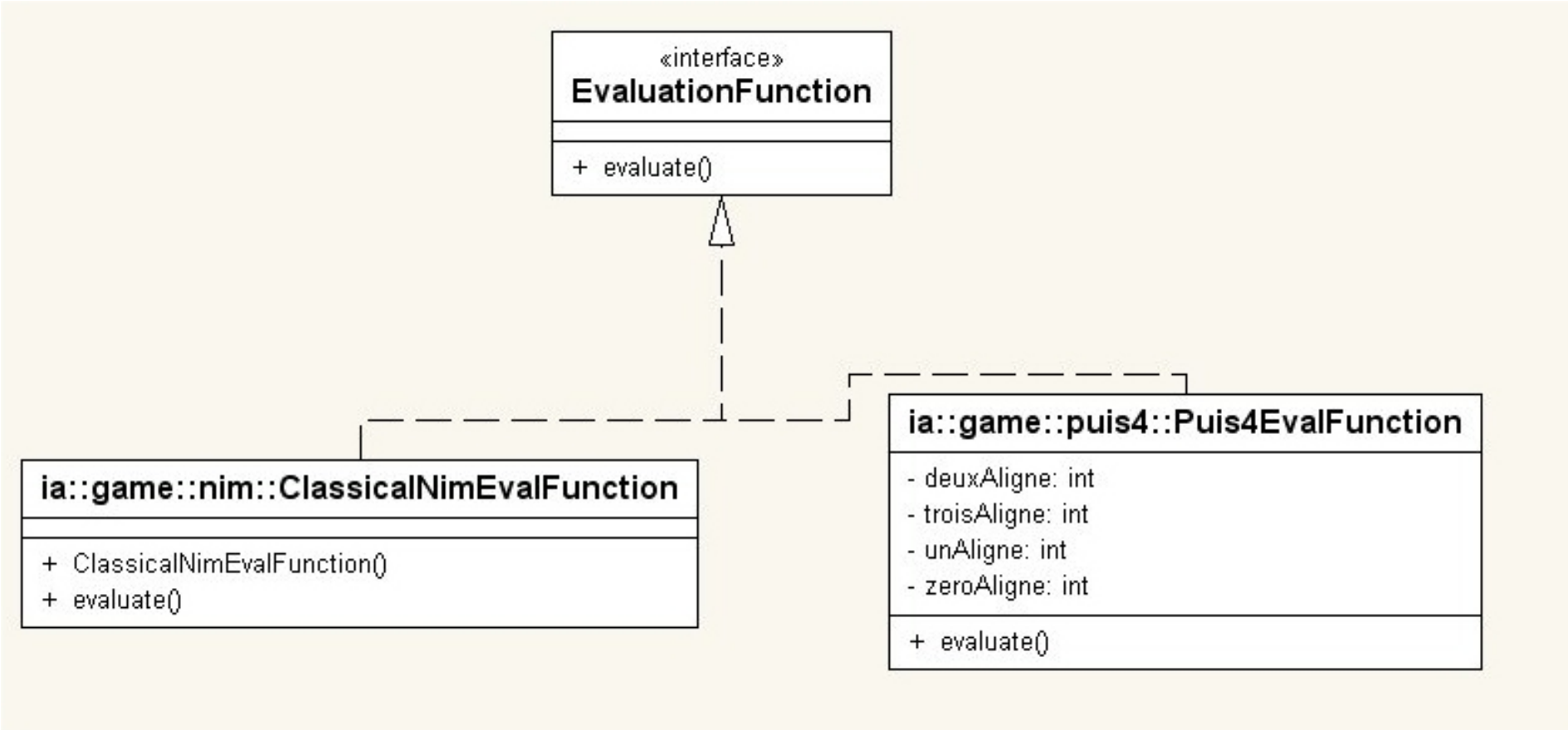
TwoPlayerGames

Classe abstraite...

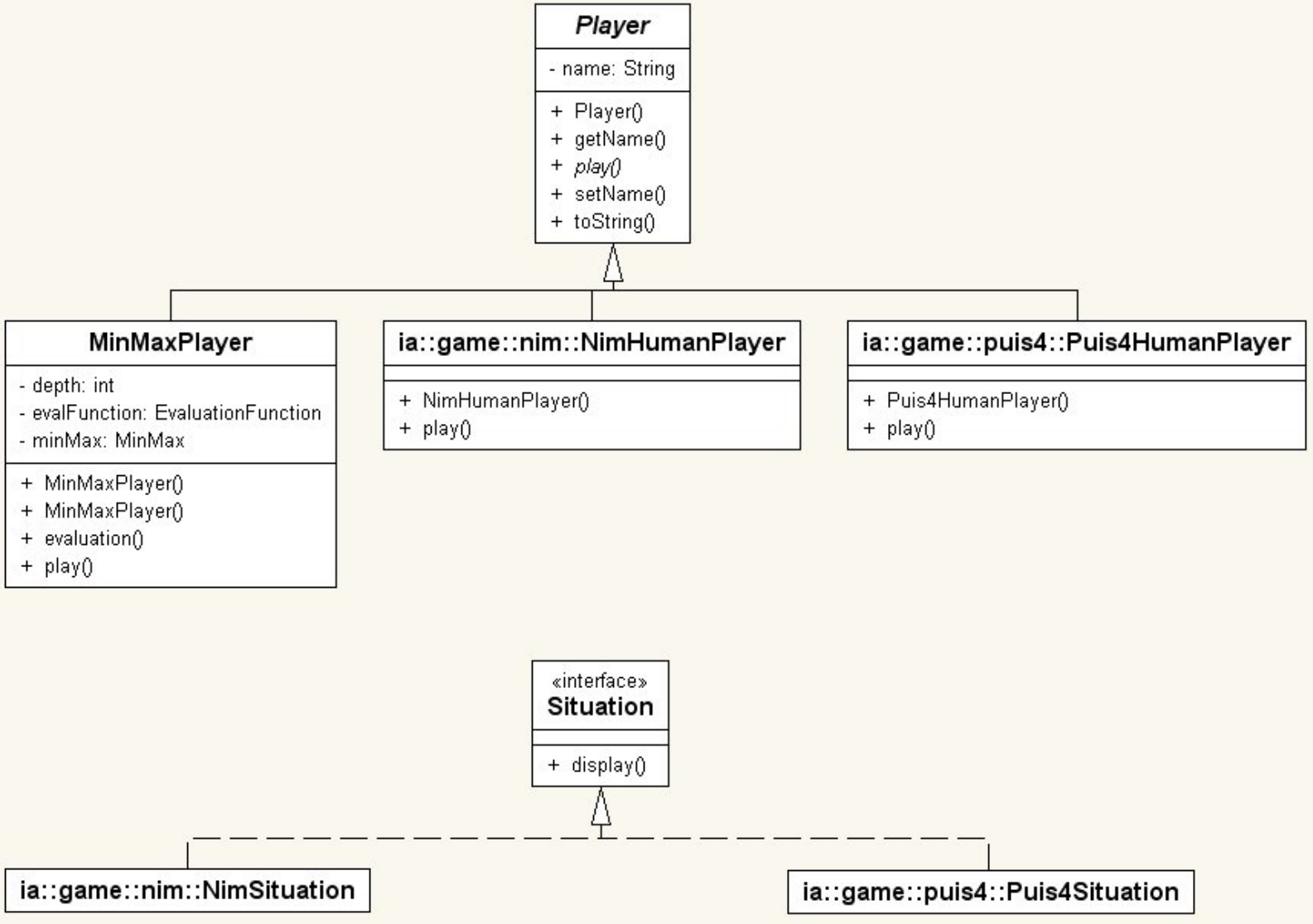


EvalFunction

Interface...



Player et Situation



Conclusion

- 4 types abstraits
  ↪ `TwoPlayerGames`, `EvalFunction`, `Player`, `Situation`
- 1 algorithme générique (le minMAX) qui s'appuie sur ces types abstraits

- $\Longrightarrow$ framework dédié aux jeux à 2 joueurs
  ↪ il "ne reste qu'à" l'instancier pour un jeu donné, en concrétisant les types abstraits proposés