

Exemple de conception

les jeux à 2 joueurs

Conception Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université Lille 1



UFR IEAA
Formations en
Informatique de
Lille 1



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"

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,

Le min-MAX

- ▶ Il faut disposer d'un fonction numérique ϕ qui attribue une valeur numérique à une situation de jeu
 $\phi : \text{Situation} \mapsto \text{Nombre}$
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  $\phi(situation)$ 
  sinon
    situFilles = les situations filles légales de situation
    si joueur = MAX
      retourner  $\max_{fille \in situFilles} \{minmax(fille, prof - 1, min)\}$ 
    sinon // joueur = min
      retourner  $\min_{fille \in situFilles} \{minmax(fille, prof - 1, MAX)\}$ 
    fin si
  fin si
```

Conception...

Les notions mises en œuvre :

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

Analyse

La mise en place de l'algorithme fait naturellement apparaître des fonctionnalités.

⇒ méthodes (abstraites ?), attributs

Remarques

- ▶ effet d'horizon (si arbre développé partiellement)
- ▶ algorithme exponentiel
 - ↪ 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 :
 - ↪ pour tout jeu à deux joueurs à information complète
- ▶ difficulté (la seule) = fournir une fonction d'évaluation ϕ pour le jeu concerné
 - ↪ il existe plusieurs fonctions pour un même jeu (⇒ différentes "stratégies" de jeu)

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
 - afficher le plateau de jeu
 - 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
- fin TantQue*
- déterminer qui est le vainqueur.

De cet algorithme on peut déjà déduire beaucoup de choses...

...dans une classe TwoPlayerGames :

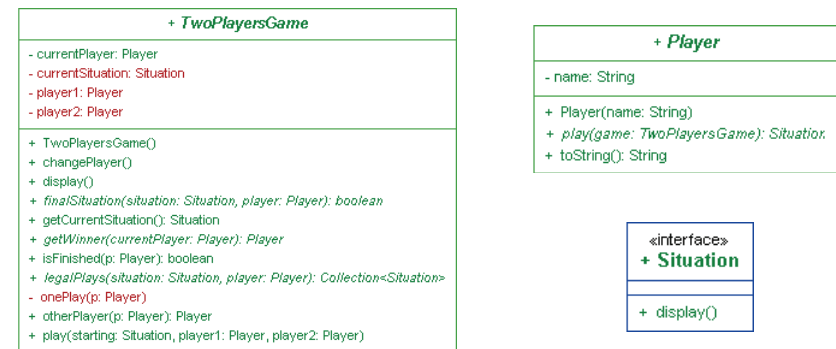
```
public void play(Situation starting, Player player1, Player player2) {
    this.setCurrentSituation(starting);
    this.player1 = player1;
    this.player2 = player2;
    this.currentPlayer = player1;
    while (!this.isFinalSituation(this.currentSituation, this.currentPlayer)) {
        this.display();
        Situation playerChoice = this.currentPlayer.play(this);
        this.setCurrentSituation(playerChoice);
        this.changePlayer();
    }
    Player winner = this.getWinner(currentPlayer);
}
```

on produit :

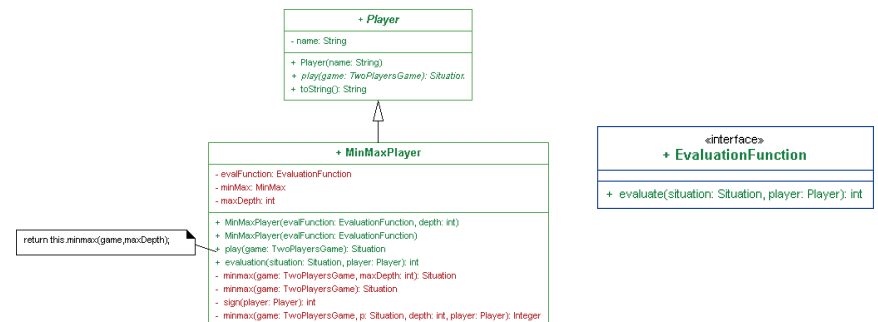
```
public int minmax(TwoPlayerGame game, Situation situation, int depth, Player player) {
    if (depth == 0 || game.finalSituation(situation, player)) {
        return MAX.evaluation(situation, player) * sign(player);
    }
    else {
        Collection<Integer> values = new ArrayList<Integer>();
        if (player == MAX) {
            for(Situation fille : situFilles) {
                values.add(minmax(game, fille, depth - 1, MIN));
            }
            return Collections.max(values);
        }
        else { //player == MIN
            for(Situation fille : situFilles) {
                values.add(minmax(game, fille, depth - 1, MAX));
            }
            return minValue = Collections.min(values);
        }
    }
}
```

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

Il en résulte pour TwoPlayerGame et Player et Situation :



MinMaxPlayer est un type de joueur particulier avec une méthode play instanciée qui utilise l'algo minMax et utilise des EvaluationFunction



Le framework est terminé

Si je veux un jeu particulier ? \Rightarrow types abstraits à étendre :

- ▶ classe abstraite TwoPlayerGame,
- ▶ classe abstraite Player,
- ▶ interface Situation,
- ▶ interface EvaluationFunction (éventuellement plusieurs fois)

- ▶ 4 types abstraits
 \hookrightarrow TwoPlayerGames, EvalFunction, Player, Situation
- ▶ 1 algorithme générique (le minMAX) qui s'appuie sur ces types abstraits
- ▶ \Rightarrow framework dédié aux jeux à 2 joueurs
 \hookrightarrow il "ne reste qu'à" l'instancier pour un jeu donné, en concrétisant les types abstraits proposés

