Chapitre 1

Fonctions d'évaluation

1.1 Influence de la représentation du problème

La façon dont on représente un jeu et les coups de ce jeu peut avoir une grande influence sur la difficulté d'un jeu. De même, elle peut avoir une grande influence sur les niveaux des programmes de jeux. On peut l'illustrer à l'aide des trois jeux suivants [62] :

- Jeu numéro 1 : Les joueurs jouent chacun leur tour. Au début du jeu 9 cartes numérotées de 1 à 9 sont posées et prêtes à être prises. A son tour, un joueur prend une des 9 cartes. Le premier joueur qui, parmi toutes ses cartes, a trois cartes dont la somme fait 15 a gagné.
- Jeu numéro 2, TicTacToe : Les joueurs jouent chacun leur tour. Un joueur fait des ronds, l'autre joueur fait des croix. A son tour, un joueur remplit avec un rond ou une croix une case vide du quadrillage. Le premier joueur qui aligne 3 ronds (resp. croix) a gagné.

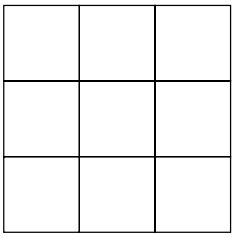

FIGURE 1.1 – Un damier de TicTacToe

– Jeu numéro 3 : Jouer au TicTacToe sur un carré magique :

4	9	2
3	5	7
8	1	6

FIGURE 1.2 – Un damier de TicTacToe Magique

Les trois jeux sont équivalents, la représentation n'est pas la même. Il nous est plus facile de raisonner avec la représentation du deuxième jeu qu'avec la représentation du premier. Le troisième jeu nous permet de jouer au jeu numéro 1 avec nos connaissances du TicTacToe.

De manière plus générale, des études psychologiques ont montré que la représentation d'un problème a une grande influence sur sa résolution. Par exemple on peut donner un problème sous plusieurs formes [5] :

Problème numéro 1 : On dispose d'autant de dominos 2×1 que l'on souhaite. Peut on complètement remplir un carré 4×4 dont on a ôté deux coins opposés avec des dominos 2×1 ?

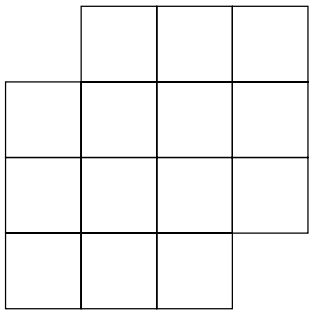

FIGURE 1.3 – Un damier tronqué

Problème numéro 2 : On dispose d'autant de dominos 2×1 que l'on souhaite. Chaque domino comporte une case noire et une case blanche. Peut on complètement remplir un damier tronqué coloré avec ces dominos ?

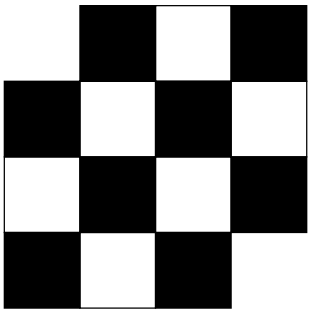

FIGURE 1.4 – Un damier tronqué coloré

La réponse est non, elle se trouve plus facilement que dans le cas du problème numéro 1. On utilise une représentation qui guide naturellement vers la solution. En effet, on peut observer que pour remplir le damier complet avec des dominos, cela ne pose pas de problème. On observe aussi que chaque case noire est voisine de cases blanches et vice-versa. On est donc obligé de remplir une case blanche à chaque fois qu'on remplit une case noire. Or dans le cas du damier tronqué, il manque deux cases blanches. On ne pourra donc pas le remplir puisqu'il y a deux cases noires de plus que de cases blanches. Il arrive souvent qu'un problème soit difficile avec une représentation et facile avec une autre représentation. Les chercheurs sur la reformulation automatique de problèmes essaient d'écrire des programmes qui se déplacent dans l'espace des représentations pour trouver celle qui est la plus appropriée à la résolution d'un problème.

1.2 Existence de stratégies toujours gagnantes

"- Combien de coups d'avance un grand maître calcule-t-il habituellement ?"

"- Un seul."

Richard Réti.

Un exemple de jeu résolu pour lequel il existe une stratégie simple pour gagner est le jeu de Nim. Le jeu de Nim se joue à deux. On dispose de plusieurs tas d'allumettes, chaque joueur prend à son tour autant d'allumettes qu'il le veut dans un tas. Le gagnant est le joueur qui prend la dernière allumette. On joue habituellement le jeu de Nim en commençant avec quatre tas de 1, 3, 5 et 7 allumettes. La stratégie gagnante consiste à transformer le nombre d'allumettes de chaque tas en sa représentation binaire. On fait alors la somme binaire sur chaque bit de la représentation binaire des nombres sans utiliser la retenue (ce qui est équivalent à faire un XOR des nombres). Si la somme binaire (ou le XOR) vaut 0, on est dans une position sûre, ce sont les positions qu'on cherche à atteindre lorsqu'on joue un coup. Si par contre on se retrouve après un coup dans une position non sûre, on a perdu.

Exemple :

```
I                    1              0001
III                  3              0011
IIIII                5              0101
IIIIIII              7              0111
                                    0000
```

La position de départ est une position sûre. Tous les coups qui peuvent être joués à partir de cette position sont des coups qui amènent à une position non sûre.

Exercice : Donner les coups gagnants dans cette position :

```
I                    1              0001
III                  3              0011
IIII                 4              0100
IIIIIII              7              0111
                                    0001
```

Exercice : Programmer un joueur parfait de Nim

1.3 Une fonction d'évaluation aux Échecs

"- Préférez-vous une dame de plus aux échecs ou dans la vie ?"

"- Ça dépend de la position."

Boris Spassky.

Une fonction d'évaluation prend en entrée une position dans un jeu et donne en sortie une évaluation numérique pour cette position. L'évaluation est d'autant plus élevée que la position est meilleure. Dans les programmes d'Échecs, la fonction d'évaluation est composée d'une partie matérielle et d'une partie positionnelle.

Exemple de partie matérielle :

$$\begin{aligned}\text{Eval}(P) = {} & 200 \times (\text{Roi} - \text{RoiAdverse}) + 10 \times (\text{Dames} - \text{DamesAdverses}) + \\ & 5 \times (\text{Tours} - \text{ToursAdverses}) + \\ & 3 \times (\text{Cavaliers} - \text{CavaliersAdverses}) + \\ & 3 \times (\text{Fous} - \text{FousAdverses}) + \text{Pions} - \text{PionsAdverses}.\end{aligned}$$

L'évaluation de la position comprend aussi des facteurs positionnels :

- structure de pions
- mobilité
- contrôle du centre
- contrôle des cases importantes de la position
- placement des pièces
- sécurité du roi

La façon de calculer les facteurs positionnels peut être adaptée à la position avec un oracle. Par exemple HITECH, de H. Berliner et C. Ebeling comporte un algorithme d'oracle qui analyse en détail la position de départ et en déduit les points importants à considérer. Par exemple si on n'est pas en fin de partie, il est inutile de perdre du temps à examiner si un pion peut être promu s'il est loin de la 8ème ligne. Un autre exemple : avancer un pion couvrant un roque peut être catastrophique dans certaines situations, on peut décider de donner un handicap à la fonction d'évaluation aux positions pour lesquelles ce pion est avancé.

1.4 Le jeu du virus

Le jeu du virus est une simplification du jeu vidéo *Attaxx*. Le plateau de jeu est une grille carrée qui contient au début deux pions noirs et deux pions blancs placés sur des coins opposés comme sur le plateau de la figure 1.5. Un coup consiste à placer un pion de sa couleur sur une des huit cases voisines d'un pion déjà posé. La case doit être vide pour qu'on puisse poser un pion dessus. De plus après avoir posé le pion, tous les pions adverses sur les cases voisines de la case du pion posé changent de couleur et deviennent de la couleur du pion posé. Un exemple de coup est donné dans la figure 1.6. Le jeu se termine lorsque le plateau de jeu est rempli. Le gagnant est celui qui a le plus de pions sur le plateau. Le jeu du virus se joue habituellement sur un damier 7×7.

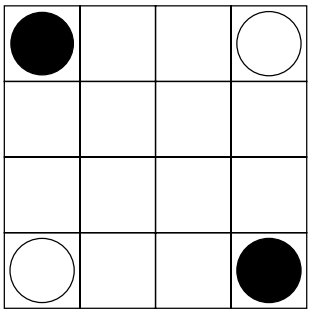

FIGURE 1.5 – La position de départ au jeu du virus 4x4.

Exercice : Imaginer une fonction d'évaluation pour le jeu du virus.

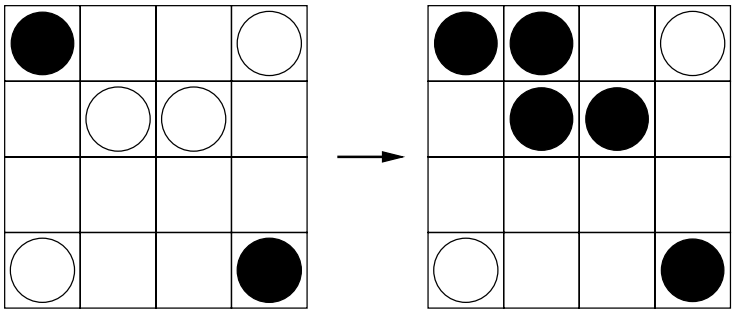

FIGURE 1.6 – Un coup noir au jeu du virus 4x4.

Exercice : Écrire un programme qui représente un jeu du virus et qui comprend une fonction d'évaluation pour ce jeu. Écrire une intelligence artificielle pour le jeu du virus qui choisit le coup qui amène à la position ayant la meilleure évaluation.

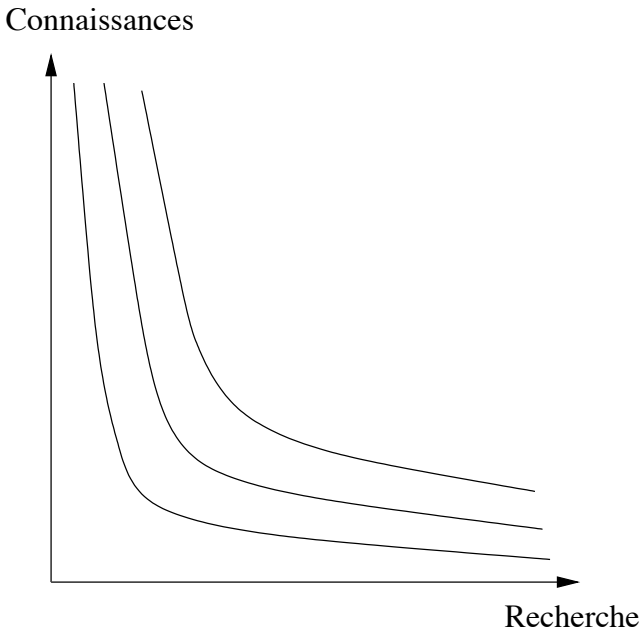

FIGURE 1.7 – Les courbes de niveau en fonction de l'effort de recherche et des connaissances utilisées dans la fonction d'évaluation.

1.5 Recherche contre connaissances

Lorsqu'on écrit un programme de jeu il est tentant d'écrire une fonction d'évaluation élaborée comportant de nombreuses connaissances de façon à bien diriger la recherche du meilleur coup. Toutefois la complexification de la fonction d'évaluation rend souvent le programme plus lent pour faire des évaluations et ralentit donc la recherche. Il peut donc arriver qu'un programme avec une fonction d'évaluation plus élaborée soit moins bon qu'un programme avec une fonction d'évaluation simple qui fait plus de recherche. Il devient donc naturel de se poser la question de la pertinence d'ajouter des connaissances dans la fonction d'évaluation en fonction du ralentissement de la recherche qu'elles induisent.

La figure 1.7 donne les courbes de niveau auxquelles on peut s'attendre en fonction des connaissances et de l'effort de recherche qu'on donne à un programme de jeu [61, 8]. Une courbe représente un niveau du programme constant. On peut remarquer qu'ajouter des connaissances lorsqu'il y a peu de recherche ou ajouter de la recherche lorsqu'il y a peu de connaissances améliore peu le programme. La conclusion est qu'il faut garder un bon équilibre entre recherche et connaissances.

Un étude empirique des courbes recherche versus connaissances a été faite aux Échecs, à Othello et au Checkers [47] mais aussi sur des finales d'Échecs [72] et sur le jeu Lines of Action [10]. Les courbes sont assez proches des courbes théoriques de la figure 1.7. Toutefois, pour les abscisses de l'effort de recherche proches de zéro, les courbes associées ne montent pas aussi haut que sur la courbe théorique.

1.6 Corrigés des exercices

1.6.1 Coups gagnants à Nim

Dans cette position :

```
I          1          0001
III        3          0011
IIII       4          0100
IIIIIII    7          0111
                      0001
```

Les coups gagnants sont de retirer une allumette soit dans le tas à une allumette, soit dans le tas à trois allumettes soit dans le tas à sept allumettes.

1.6.2 Joueur parfait pour Nim

```
#include <iostream>

using namespace std;

int tas [4] = {7, 5, 3, 1};

int main () {
  int t, nombre;
  while (true) {
    cout << tas [0] << " " << tas [1] << " " <<
            tas [2] << " " << tas [3] << endl;
    if (tas [0] + tas [1] + tas [2] + tas [3] == 0) {
      cout << "J'ai gagne !" << endl;
      break;
    }
    do {
      cout << "Donnez le tas (entre 0 et 3): ";
      cin >> t;
      cout << "Donnez le nombre : ";
      cin >> nombre;
    } while ((t < 0) || (t > 3)  ||
             (tas [t] < nombre) || (nombre < 0));
    tas [t] -= nombre;
    cout << tas [0] << " " << tas [1] << " " <<
            tas [2] << " " << tas [3] << endl;
    for (int i = 0; i < 4; i++)
      for (int j = 1; j <= tas [i]; j++) {
        tas [i] -= j;
        if ((tas [0] ^ tas [1] ^ tas [2] ^ tas [3]) == 0) {
          t = i;
          nombre = j;
        }
        tas [i] += j;
      }
    cout << "Je prend " << nombre <<
            " dans le tas " << t << endl;
    tas [t] -= nombre;
  }
  return 0;
}
```