Projet informatique : jeu du Breakthrough

Sébastien Rodriguez
Christian Ingouff

Année 2011/2012
Semestre 2

Sommaire

1 Problématique, règles du jeu **2**

2 Description du programme **3**
2.1 Menus 3
2.2 Jeu 4
2.3 Fonctionnement de l'intelligence artificielle 4
2.3.1 Algorithme Minmax 4
2.3.2 Fonction d'évaluation 8

3 Déroulement du projet **9**
3.1 Décision personnelle : l'interface graphique 9
3.2 Implication et rôles des élèves 9
3.3 Bilans personnels 10
3.3.1 Sébastien 10
3.3.2 Christian 10

Chapitre 1

Problématique, règles du jeu

Le but du projet de fin d'année était de coder le jeu Breakthrough en langage Pascal.

Il s'agit d'un jeu de plateau proche du jeu des dames. Chaque joueur dispose de seize pions répartis sur deux lignes de chaque coté d'une grille de huit cases par huit cases. Les joueurs jouent chacun leur tour en commençant par le joueur blanc.
Un pion ne peut se déplacer qu'en avant : en diagonale ou tout droit ; un pion peut prendre un pion adverse uniquement lors d'un mouvement diagonal. Le premier joueur faisant atteindre l'extrémité opposé du plateau avec l'un de ses pions l'emporte. Si un joueur n'a plus de pion il perd la partie.

Chapitre 2

Description du programme

Le programme possède une interface graphique, entièrement maniée à l'aide de la souris. Il accueille l'utilisateur dans son meu principal.

2.1 Menus

Le menu principal donne l'accès à plusieurs sous-menus.

Le menu "nouvelle partie" permet de lancer une partie en monojoueur ou en multijoueur en suivant les paramètres choisis.

Le menu "charger" permet de récupérer l'une des trois parties mises en pause et sauvegardées : lorsque la sauvegarde est notée en vert, cela signifie qu'elle est non vide et ainsi chargeable.

Le menu "paramètres" permet de changer la couleur du fond d'écran du jeu, la taille de la grille et la difficulté de l'IA : maintenez le clic sur la barre verticale et glissez pour modifier l'un de ces paramètres.

Le menu "Comment Jouer" donne accès à un récapitulatif des règles.

2.2 Jeu

Une fois en jeu le joueur humain commence au bas de l'écran contre l'IA ou un autre joueur, selon le mode de jeu sélectionné. Le joueur blanc commence systématiquement, puis le joueur qui se voit attribuer le tour voit s'afficher un pion noir en dehors de l'écran au niveau de sa ligne de départ.

Sur le côté droit de l'écran se trouvent le bouton sauvegarde, afin de sauvegarder la partie actuelle sur l'emplacement de l'une des trois sauvegardes, et le bouton quitter qui permet de revenir au menu principal.

2.3 Fonctionnement de l'intelligence artificielle

2.3.1 Algorithme Minmax

Pour déterminer le meilleur coup à jouer, l'algorithme va tester tous les coups possibles pour comparer les différentes conséquences de ces coups joués, ceci grâce à une fonction d'évaluation de la grille. L'algorithme retient la situation la plus avantageuse, et retourne le coup correspondant.

En testant un coup possible, l'algorithme pourra aussi vérifier tous les coups possibles à partir de la situation fille générée par le coup possible déjà effectué. Il fera cela n fois, n étant la profondeur définie de l'algorithme. L'intérêt réside dans le fait de pouvoir prédire une bonne ou une mauvaise situation.

Avantages L'algorithme Minmax évalue très rigoureusement toutes les situations possibles à partir d'une situation de départ, ce qui permet à l'ordinateur de tomber à coup sûr sur le meilleur coup possible.

Limites L'algorithme comporte cependant des limites : plus la profondeur définie sera grande, plus le nombre de vérifications de grilles effectuées sera grand, et plus le temps alloué à la "réflexion" de l'intelligence artificielle sera grand. Elle est donc limitée par la capacité de l'ordinateur à effectuer des actions rapidement.

```
fonction minmax(situation : matrice ; profondeur,
                          joueur_actuel : entier):entier;
// Retournera la conséquence de la suite de coups effectués
début
  si profondeur = 0 ou quelqu'un gagne alors
    retourner evaluation(situation)
  sinon pour tous les coups possibles
    si joueur_actuel = bot alors // maximisation
      retourner maximum(minmax(situation_fille, profondeur-1, autre_joueur)
    sinon // minimisation
      retourner minimam(minmax(situation_fille, profondeur-1, autre_joueur)
    finSi
  finSi
fin
```

Compte tenu du grand nombre de vérifications de grilles à effectuer selon la profondeur de l'algorithme, il paraîtra judicieux de vouloir réduire ce nombre en excluant les situations qui vont forcément être désavantageuses : c'est de la que vient l'amélioration de l'algorithme Minmax : l'algorithme alpha-bêta.

Son fonctionnement est le même que le Minmax : on lui ajoute uniquement le fait de pouvoir avorter prématurément la vérification de grilles si cela s'avère inutile car la situation sera forcément moins avantageuse que ce que l'on a déjà analysé.

```
fonction alphaBeta(situation : matrice ; profondeur,
                 joueur_actuel, alpha, beta : entier):entier;
// Retournera la conséquence de la suite de coups effectués.
// Initialisation : alpha = -inf, beta = +inf
var
  temp : entier
début
  si profondeur = 0 ou quelqu'un gagne alors
    retourner evaluation(situation)
  sinon pour tous les coups possibles
    si joueur_actuel = bot alors // maximisation
      temp <- maximum(minmax(situation_fille, profondeur-1,
                                  autre_joueur, alpha, beta)
      si temp > alpha alors
        alpha <- temp
      finSi
    sinon // minimisation
      temp <- minimum(minmax(situation_fille, profondeur-1,
                                 autre_joueur, alpha, beta)
      si temp < beta alors
        beta <- temp
      finSi
    finSi
    si alpha > beta alors
      retourner temp // temp sera forcément la valeur évaluation
                        désignant la plus avantageuse situation
  finSi
fin
```

La version utilisée diffère légèrement de l'alpha-bêta classique, pour des raisons de facilité d'implémentation :

```
fonction alphaBeta(situation : matrice ; profondeur,
              joueur_actuel, minmax_des_Freres : entier):entier;
// Retournera la conséquence de la suite de coups effectués.
// Initialisation : minmaxFrere = +inf (en considérant que le bot maximisera)
var
  fils, minmax_des_Fils : entier
début
  si profondeur = 0 ou quelqu'un gagne alors
    retourner evaluation(situation)
  sinon pour tous les coups possibles
    minmax_des_fils <- -inf si maximisation, +inf si minimisation
    si joueur_actuel = bot alors // maximisation
      fils <- maximum(minmax(situation_fille, profondeur-1,
                                autre_joueur, minmax_des_fils)
      si fils > minmax_des_fils alors
        minmax_des_fils <- fils
      finSi
      si minmax_des_fils > minmax_des_freres alors // coupure
        retourner minmax_des_fils
      finSi
    sinon // minimisation
      fils <- minimum(minmax(situation_fille, profondeur-1,
                                autre_joueur, minmax_des_fils)
      si fils < minmax_des_fils alors
        minmax_des_fils <- fils
      finSi
      si minmax_des_fils < minmax_des_freres alors // coupure
        retourner minmax_des_fils
      finSi
    finSi
  finSi
fin
```

2.3.2 Fonction d'évaluation

Son utilité est de déterminer si la situation qui est fournie à l'ordinateur (une grille à analyser) est avantageuse pour lui : sa valeur sera d'autant plus grande que la situation sera avantageuse.

La valeur retournée est la somme des valeurs de l'évaluation de chaque pion présent sur le plateau, ayant différents rôles selon leur position, non seulement sur le plateau mais aussi par rapport aux autres pions. On assigne alors à chaque pion une valeur avec diverses vérifications : on lui attribuera une très grande valeur si elle elle joue un grand rôle dans la victoire du joueur que le pion dessert. Par exemple, si le pion a atteint l'opposé de son point de départ, on lui attribuera beaucoup de valeur.

La valeur pour la situation du bot sera calculée ainsi : valeur = somme des valeurs des pions du bot - somme des valeurs des pions de l'autre joueur.

Chapitre 3

Déroulement du projet

3.1 Décision personnelle : l'interface graphique

Nous avions d'abord réalisé le projet, IA exclue, dans le terminal à la manière du Sphinx de début d'année. Puis ayant fini cela rapidement grâce à un "recyclage" du projet Sphinx, nous avions envie d'aller plus loin et d'apprendre l'interface graphique. Nous avons donc adapté notre projet afin d'intégrer une interface graphique. Une fois maîtrisée, l'interface graphique s'est même avérée être plus simple d'utilisation que le code en terminal. Cela nous permet en plus d'utiliser la souris, ce qui est bien plus simple d'accès pour l'utilisateur.

3.2 Implication et rôles des élèves

Ce projet a suivi le même schéma que pour le premier projet. Sébastien s'est particulièrement occupé de tout ce qui est menus et aspect graphique pour le jeu alors que Christian s'est chargé de l'IA ainsi que du jeu lui même.

3.3 Bilans personnels

3.3.1 Sébastien

Ce projet est encore une fois une expérience intéressante au niveau du travail de groupe mais surtout au niveau de l'interface graphique. En effet on se remémore le début d'année où nous voulions tous faire du graphique tout de suite et le chemin parcouru jusqu'à aujourd'hui. La découverte d'une "vraie" IA et de la complexité d'en réaliser une est réellement intéressante et nous aide à nous projeter dans de la programmation future plus professionnelle. Cela apporte également une nouvelle dimension à la programmation informatique.

3.3.2 Christian

L'abordage de nouvelles notions comme la manipulation de fichiers, l'interface graphique et l'algorithme Minmax m'ont beaucoup séduit et encouragé au travail approfondi. Je remarque aussi les progrès que l'ensemble de la classe a fait par rapport au 1er semestre, et ne peut espérer que mieux pour la seconde année, et d'autant plus pour la vie professionnelle. Ce projet a aussi été l'occasion de pouvoir aider des élèves davantage en difficulté, ce qui est intéressant pour le métier d'ingénieur, dans lequel la capacité de guider son équipe est très importante.