Documentation de l'arbitre de jeu

Flavien Raynaud, Thomas Papillon

2011–2012

1 Règlement de la compétition

1.1 Rappel des règles du Sphinx

Au commencement d'une partie de Sphinx, chaque joueur "fabrique" une grille, et ne communique à l'adversaire que l'emplacement de l'entrée et du trésor de celle-ci. Ensuite, le jeu se rapproche plus ou moins d'une bataille navale : chacun des joueurs part avec une grille inconnue sauf les deux cases spéciales précédement définies. A chaque tour, chacun devra "jouer", c'est à dire demander à son adversaire le type (mur, chemin) d'une case qui l'intéresse. Si la case demandée est un chemin, le joueur en cours peut jouer à nouveau, sinon c'est au tour de l'adversaire. La partie se termine lorsque l'un des deux joueurs a trouvé le trésor qu'il cherche.

1.2 Règles spécifiques à la compétition

Un match se déroule entre deux IA et sans intervention humaine jusqu'à ce qu'un vainqueur ou un match nul soit détecté. Pour cela, nous utiliserons un programme arbitre qui servira de cadre au combat sans merci entre les deux IA. Ce programme invite tout d'abord les deux bots à générer une grille de jeu pour l'adversaire, vérifie qu'elle est correcte, puis lance le match.

La grille générée par votre IA devra respecter les règles suivantes, sinon le joueur adverse sera déclaré vainqueur d'office :

- Elle doit respecter la taille et le nombre de murs imposés (cf. Fonctions imposées).
- Elle doit ne comporter qu'une seule entrée, ainsi qu'un seul trésor.
- Un chemin doit exister pour aller de l'entrée au trésor (rappelons que le déplacement n'est pas autorisé en diagonale), il n'est pas nécéssairement unique.

On peut de plus remarquer les choses suivantes :

- Le choix du bot démarrant le match est soumis au hasard.
- Un coup non autorisé (en dehors de la grille ou non connexe a une case déja visitée) n'apporte rien et fait passer son tour.
- Si un joueur trouve son trésor, une dernière chance de faire un match nul est laissée à son adversaire.
- Il y a une limite au nombre de coups égale au nombre de cases de la grille, ceci étant lié à la nature *répétitive* de certaines IA.

1.3 Lancement d'un match

Pour lancer un match il suffit de récupérer l'éxécutable `judge_terminal` et de l'éxécuter dans un terminal avec la commande `./judge_terminal ./libbot1.so ./libbot2.so h w n t` sachant que :

- Les deux premiers arguments sont les IA (pour l'explication du `.so`, voir la partie suivante).
- `h`, `w`, `n`, `t`, représentent respectivement la hauteur, la largeur, le nombre du murs de la grille du match, et le temps d'arrêt après chaque coup (en ms) permettant de ralentir le jeu.

N.B. : l'arbitre tourne sous Linux, et si certains désirent une interface graphique un peu plus poussée, ils peuvent utiliser l'éxecutable `judge_graphic`. Ce dernier nécessite les paquets `libsvga1`, `libsvga1-dev` ainsi que `svgalib-bin` et devra être lancé avec les droits de superutilisateur (`sudo ./judge_graphic ./bot1.so ./bot2.so h w n t` dans la majorité des distributions).

2 Détails techniques

Pour participer à la compétition, il faudra sans doute adapter légèrement votre code pour des raisons de compatibilités avec l'arbitre. Cela passe par plusieurs étapes :

- Définir quelques types et fonctions de base qui serviront à communiquer avec votre IA ainsi que quelques types utilisés par l'arbitre.
- Modifier votre IA pour en faire une *library*, c'est à dire un fichier `.so` qui sera lisible par l'arbitre. Il faut signaler qu'un fichier nommé `monbot.pas` compilera en un fichier `libmonbot.so` (un préfixe `lib` est automatiquement ajouté).

2.1 Les types et fonctions de base

Les types sont les suivants :

```
1  type
2      Grid = array of array of integer;
3      Coord = record
4          l, c : integer;
5      end;
```

Les fonctions quand à elles doivent adopter les prototypes [1] suivants :

```
1  function get_name() : string;
2  procedure init_grid(var _grid : Grid; h, w, nb_walls : integer);
3  function play(var _grid : Grid; h, w : integer) : Coord;
4  procedure outcome(var _grid : Grid; h, w : integer; just_played : Coord; answer : integer);
```

Ces types et fonctions doivent *absolument* être définis sous peine de défaite immédiate. Nous précisons que `h` et `w` représentent respectivement la hauteur (height) et la largeur (width) de la grille et que que les tableaux sont indicés de `[0,0]` à `[h-1,w-1]`. Maintenant éclairons un peu le fonctionnement de ces derniers :

2.1.1 Les types

`Coord` est un type composé (la même chose qu'une structure, pour ceux qui ont fait du C) de deux entiers représentants respectivement un indice de ligne, et un indice de colonne. Pour une variable `monTresor` de type `Coord`, les indices seront accessibles comme ceci : `monTresor.l` et `monTresor.c`. `Grid` représente une matrice d'entiers.

2.1.2 La fonction `get_name`

Elle doit renvoyer le nom de votre IA, sous forme d'une variable de type `string`, celui-ci ne doit pas dépasser 20 caractères.

2.1.3 La procédure `init_grid`

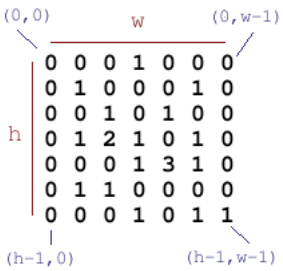
Cette procédure est appelée une unique fois au début d'un match, l'arbitre lui passe la grille que l'IA doit fournir par référence (avec le mot clé `var`), ce qui signifie que vous pouvez modifier directement la variable `_grid`. La grille de base ainsi fournie contient uniquement des chemins libres, ensuite à vous de la modifier pour placer des murs, ainsi qu'une entrée et un trésor. Faites attention de ne pas la redimensionner, cela entrainerait une défaite. L'arbitre passe aussi les paramètres numériques (hauteur, largeur, nombre de murs) de la grille, pour simplifier la fonction.

Dans une `Grid` la représentation des différentes choses composant le labyrinthe suit la convention suivante :

- Chemin libre : 0
- Mur : 1
- Entrée du labyrinthe : 2
- Trésor : 3
- Case non connue : -1 (ne pas mettre dans votre grille, sert juste lors du match pour vous indiquer les case que vous n'avez pas encore explorées)

L'existence d'un chemin reliant l'entrée au trésor sera vérifiée, veillez-y donc. La grille d'exemple du polycopié d'explication du projet se représentera donc ainsi :

1. Le prototype d'une fonction (ou une procédure) est une ligne contenant le nom, les arguments et, le cas échéant, le type de retour de la fonction. Elle se présente par exemple comme ceci : `function mafonction(a, b : integer) : boolean`.



2.1.4 La fonction `play`

A cette fonction est envoyée la grille *privée* de l'IA — c'est à dire la grille comportant des cases non explorées — ainsi que les paramètres numériques. Cette grille est en quelque sorte le *papier de brouillon* de l'IA, et cette dernière la modifie à volonté, la seule restriction est encore une fois de ne pas modifier sa taille. La fonction doit renvoyer sous forme de `Coord` la case à laquelle elle souhaite accéder. Attention : tout les coups, qu'ils soient réussis ou non, rentrent dans le décompte, c'est pourquoi vous pouvez jouer n'importe quelle case de la grille du moment qu'elle est connexe à une case déja visitée.

2.1.5 La procédure `outcome`

Cette procédure sera appelée à la fin d'un tour pour que l'IA puisse faire les modifications nécéssaires sur sa grille *privée*, notemment changer le type de la case concernée.

2.2 Adapter la forme de votre code

Pour le concours, votre IA devra être rendue sous forme de *library*, c'est à dire que vous devrez respecter le format de fichier suivant :

```
1  library monbot; (* Nom de la library *)
2
3  type
4      Grid = array of array of integer;
5      Coord = record
6          l, c : integer;
7      end;
8
9  function get_name : string; cdecl;
10 begin
11     (* La vous mettez votre code *)
12 end;
13
14 procedure init_grid(var _grid : Grid; h, w, nb_walls : integer); cdecl;
15 begin
16     (* La vous mettez votre code *)
17 end;
18
19 function play(var _grid : Grid; h, w : integer) : Coord; cdecl;
20 begin
21     (* La vous mettez votre code *)
22 end;
23
24 procedure outcome(var _grid : Grid; h, w : integer; just_played : Coord; answer : integer); cdecl;
25 begin
26     If (_grid[just_played.l, just_played.c] = -1) Then
27         Case answer Of
28                 1, 0, 2 :
29                         _grid[just_played.l, just_played.c] := answer;
30         End;  (* Exemple de code *)
31 end;
32
33 (* Ici vous pouvez mettre vos fonctions internes *)
34
35 (* A partir d'ici, vous ne touchez plus *)
36 exports
37     get_name, init_grid, play, outcome;
38 end.
```

Avec `fpc`, cela compile automatiquement en un fichier `.so`, que vous pourrez donner à l'arbitre. Si le code de votre IA est trop imposant et nécéssite d'être cloisonné, vous pouvez vous renseigner sur la directive `$include`. [2]

2. cf. la doc de FreePascal : `http://www.freepascal.org/docs-html/prog/prog.html`.

3 Le mot de la fin

Pour toute question sur l'arbitre[3], ainsi que pour tout rapport de bug ou suggestion quelconque, vous pouvez venir voir les deux auteurs de l'arbitre : Flavien Raynaud (CPI 1B, `raynaudfla@eisti.eu`) et Thomas Papillon (CPI 1B, `papillonth@eisti.eu`). Pour les questions sur le langage Pascal en lui même nous vous invitons à regarder en priorité la documentation.

Bonne chance à tous et que le meilleur gagne !

3. dont le code sera soit dit en passant libéré et diffusé après le tournoi pour ceux que ca intéresse.