Projet semestre 1 Jeu de Go

BOËDA Gautier LE ROUZIC Steven

ENSICAEN 2013-2014, 1A Info, Groupe TP 2

Contents

1 Environnement de développement

Le programme a été réalisé principalement sous Windows, avec l'IDE Eclipse et l'environnement MingW. Nous avons utilisé la librairie SDL 1.2 ainsi que

ses addons SDL_image et SDL_ttf. La librairie matrice écrite en TP a été réécrite pour correspondre aux conventions établies pour ce projet. Nous avons utilisé Git comme système de versionnage afin de faciliter le travail depuis de multiples stations de travail, et GitHub comme dépôt central. (http://github.com/stevenlr/ProjetGO)

Chaque sous projet était organisé comme ceci : un dossier `assets` contenant les images, les fontes et autres ressources, un dossier `lib` contenant les librairies nécessaires à la compilation, un dossier `include` contenant les fichiers d'en-tête des librairies et un dossier `src` contenant le code source et un dossier `include` contenant les en-têtes.

Des conventions d'écriture ont été adoptées : nous avons utilisé le CamelCase, le code était écrit le plus possible en français (à l'exception des mots `get` et `set`) et les prototypes de fonctions suivaient le style `Type_nomFonction` où `Type` est le type sur lequel opère la fonction.

Les fichiers sources étant organisés en sous-dossiers, un fichier `Makefile` a été placé dans chaque dossier en appelant un autre commun à tous ceux-ci contenant les instructions de compilation pour chaque sous dossiers. Tous les fichiers objets générés étaient alors placés dans un dossiers `obj` d'où ils étaient assemblés en l'exécutable final. Les instructions de compilation sont spécifiées dans le fichier README.md.

Ce rapport a été écrit en Markdown puis compilé en LaTeX puis en PDF grâce à l'utilitaire Pandoc.

2 La libmatrice

La librairie matrice écrite en TP a été entièrement réécrite afin de suivre les conventions établies, en particulier l'encapsulation des données. En interne, une représentation linéaire des données a été utilisée au lieu d'un tableau bidimensionnel.

3 Le jeu de Go

3.1 Structures de données

3.1.1 Liste

3.1.2 Pile

3.1.3 Position

3.1.4 Couleur

3.1.5 Pion

3.1.6 Chaîne

3.1.7 Territoire

3.1.8 Plateau

3.1.9 Partie

3.1.10 Tutoriel

3.2 Algorithmes

3.2.1 Déterminer les libertés

3.2.2 Déterminer territoire

3.2.3 Déterminer séki

3.2.4 Déterminer les yeux d'une chaîne

3.2.5 Déterminer une chaîne

3.2.6 Réaliser une capture

3.2.7 Déterminer les chaînes entourant un territoire

3.2.8 Déterminer les chaînes capturées

3.2.9 Calcul du score

3.3 Intelligence Artificielle

Une intelligence artificielle relativement simple (mais non naïve) a été implémentée. Son exécution se déroule en plusieurs phase : la phase de passage de tour,

de défense, d'attaque et de jeu naïf.

3.3.1 Passage de tour

Si l'adversaire vient de passer son tour, alors l'ordinateur a une chance sur quatre de faire de même. Celà permet, lorsqu'un humain joue contre l'ordinateur, de terminer le jeu dans des temps raisonnables si le joueur décide qu'il ne peut plus rien faire qui pourrait l'avantager. En effet, cette IA de réfléchira pas directement à la possibilité de passer son tour si elle n'a plus l'avantage nul part sur le plateau.

Si l'ordinateur n'a pas décidé de passer, alors la phase de défense commence.

3.3.2 Phase de défense

L'ordinateur va commencer par vérifier s'il est en danger quelque part sur le goban : si une de ses chaînes ne possède plus qu'une libertée. Si c'est le cas, il va placer un pion sur cette liberté afin d'essayer de se libérer. Le système de placement de pion permet de s'assurer que si le coup joué résulte en une chaîne n'ayant plus de liberté pour l'ordinateur lui-même, alors le coup est annulé, car illégal.

Si l'ordinateur n'a pas joué défensivement, il va passer à l'offensive.

3.3.3 Phase d'attaque

L'ordinateur va alors chercher sur le goban toutes les chaînes ennemies et sélectionner celle possédant le moins de libertés. S'il en trouve une, il va placer un pion sur une de ses libertés.

Si l'ordinateur n'a pas pu jouer de coup offensif, alors il va passer à la phase naïve.

3.3.4 Phase naïve

Le dernier recours de l'ordinateur est alors de placer un pion au hasard sur les cases restantes du goban. S'il n'a pas réussi à en placer, alors il passe son tour.

4 Architecture du programme

4.1 Les écrans et leurs états

4.2 Les contextes

4.3 Les interfaces

4.4 Les événements

5 Autres éléments du programme

5.1 Eléments d'interface graphique

5.1.1 Les boutons

5.1.2 Les boutons à choix multiple

5.1.3 Les champ de texte

5.2 Les ramasses miettes

Afin de limiter le nombre d'allocations et de désallocations par unité de temps pour les structures qui en demandent beaucoup telles que `ElementListe` et `Position`, nous avons implémenté un ramasse miettes très simple nous permettant de recycler les précédents instances de ces types dont voici le fonctionnement. Chaque type à recycler possède son ramasse miettes.

Une ramasse miettes est configuré avec le nombre maximal d'instances recyclées stockées et la taille d'une structure à allouée. Au début, aucun pointeur n'est recyclé. Lors de l'allocation d'un élément, le ramasse miettes regarde si on a des pointeurs disponibles à recycler. Si c'est le cas, il retourne celui au dessus de la pile des pointeurs recyclés, dans le cas contraire il alloue un élément de la taille spécifiée à la création du ramasse miettes. Lors de la désallocation, si la pile d'éléments recyclés n'est pas pleine, il place le pointeur en haut de la pile et dans le cas contraire désalloue l'élément. À la fin de l'exécution du programme, tous les pointeurs recyclés restants dans la pile sont désalloués.