Projet Fondements de l'informatique et ODL

Gilles Lebrun & Luc Brun

1 Informations générales

Début du projet : Semaine du 10 décembre

Remise du projet : Semaine du 10 février

Nombre de personnes : le projet est effectué en binôme. Les deux binômes doivent appartenir au même groupe de TP.

Documents à remettre : Un rapport écrit de 10 pages (max) avec une description des méthodes algorithmiques et des structures de données utilisées ainsi qu'un jeu d'essais. Ce document doit être remis à votre encadrant de TP de C.

2 Objectif :

L'object est de réaliser un programme permettant de jouer au jeu du go. Le plateau pourra avoir les 3 tailles suivantes : 9x9, 13x13 et 19x19. Les joueurs pourront être un humain et/ou un ordinateur. Toutes combinaisons autorisées. Dans le cas d'un ordinateur, ll jouera un coup au hasard parmi les possibles. L'ordinateur servira surtout à faire des tests afin de produire des cas pas forcément attendus et pouvant révéler des bugs dans votre programme. Cependant si vous en avez la curiosité et l'envie, vous pouvez mettre en place une méthode de jeu pour l'ordinateur plus évoluée.

Votre programme devra également permettre à une personne ne connaissant pas le go de comprendre les principes en partant de configuration de grilles préétablis et exposant tel ou tel point de règle. Il sera également possible de joueur des parties avec handicap.

Pour les détails des règles je vous conseil le site suivant :

http ://jeudego.org/_php/regleGo.php ?chap=1,

mais ne vous limitez pas qu'à celui-ci.

Une partie devra pouvoir être rejouée du début à sa fin et sera alors également possible d'annuler un ou plusieurs coups afin de pouvoir tester une autre approche de la partie.

3 Aspect ODL

Pour la création et modification d'un plateau de jeu, vous exploiterez à nouveau la librairie matrice (grille) que vous avez produites pour le jeu de la vie, mais que vous adapterez si nécessaire. Elle devra être utilisé comme un librairie dynamique par votre programme. Le jeu de go devra être jouable en mode texte et en mode graphique. Pour la partie texte vous utiliserez la possibilité d'exploiter des couleurs en mode texte comme cela a été utilisé pour la réalisation de script shell pour la réalisation d'un menu lors du TP 2 d'ODL (ce lien [1] vous donne des informations complémentaires et en particulier comment le faire de façon lisible en C) Pour la partie graphique du jeu, vous exploiterez la librairie SDL en vous basant sur les exemples et productions faites lors du TP 5 d'ODL. La documentation du code devra être réalisées avec doxygen, ainsi que la partie description de votre projet. Vous réalisez ce projet en utilisant un IDE qui soit CodeBlocks ou Eclipse-CDT. Cependant votre produit final devra pouvoir être compilé et exécuté sans l'IDE sur Windows et Linux. Vous prévoirez donc un makefile spécidique pour ces deux plateformes avec les règles classiques : all, install et clean. Pour linux vous supposerez que les libraires classiques1 de SDL sont déjà installées. Pour Windows, vous placerez dans un répertoire spécifique nommé dllSDLwindows les libraires nécessaires à la compilation et à l'éxécution de votre projet. Les ressources multimédias (images, polices de caractères, sons, etc.) seront placées dans un répertoire nommé ressources et devront être communes à Linux et Windows. Tous les fichiers d'entêtes devront être dans un répetoire nommé include.

Si vous manquez d'inspiration pour la représentation l'affichage de votre plateau, aussi bien en mode texte que en mode graphique, vous pouvez vous inspirer de celle utilisée par le projet gnugo : http ://www.gnu.org/software/gnugo/gnugo.html.

Pour l'organisation de votre programme C, vous deverez obligatoirement respecter certaines contraintes qui seront énumérées dans la suite de ce document.

En premier lieu vous définirez plusieurs types (avec typedef) qui seront majoritairement des structures, mais ils pourront également correspondre à des enumérés, ainsi que des types synonymes. Chaque type ayant un rôle spécifique dans la mise en oeuvre du GO afin de permettre une organisation modulaire de votre programme. La nature des données internes à chaque type étant libre à condition de respecter les aspects algorithmiques demandés (voir section suivante). Le noms et rôles de ces types étant les suivants :

Couleur : La couleur d'un pion,

Position : Une position sur le plateau,

Positions : Un ensemble de positions

Pion : Les informations sur un point : sa position et sa couleur

Pions : Un ensemble de pions

1. http ://fr.openclassrooms.com/informatique/cours/des-couleurs-dans-la-console-linux/changer-la-couleur-de-police

Plateau : contient la position de l'ensemble des pions sur une grille de dimension n fois n à un instant donné de la partie,

Chaine : Un ensemble connexe de pions sur le plateau,

Chaines : L'ensemble des chaînes du plateau,

Libertes : Les libertés associées à une chaîne,

Territoire : Une portion de plateau encadré par une ou plusieurs chaînes de pions en tenant compte des bords du plateau.

Territoires : Un ensemble de territoires

Partie : regroupe toutes les informations sur une partie, l'état du plateau (goban) actuel, à quel joueur c'est de jouer, les coups précédents, etc.,

Ces types devont permettre de répondre à plusieurs points de régle du jeu de GO. En Particulier :

- Déterminer une chaîne de pions (de même couleur) adjacent à partir d'un pion donnée sur la grille du plateau de jeu,
- Déterminer les libertés d'une chaîne de pions donnée, Déterminier un territoire comme un ensemble d'intersections adjacentes du plateau,
- Déterminer les pions éventuellement capturés par la pose d'un pion adverse,
- Vérifier que la pose du pion d'une couleur donnée ne produit pas une chaîne sans liberté de cette couleur alors qu'aucune prise ne sera possible,
- Identifier les pierres mortes, vivantes et vivantes par seki,
- Tester si deux gobans (plateaux) sont identiques,
- Déterminer le score en fin de partie en tenant compte du komi.

4 Aspect Algorithmique

Les types ensembles, chaine, plateau,positions, Territoire, Territoires seront déclarés comme des pointeurs sur structure dans les .h associés. La structure elle même sera déclaré dans le .c implémentant la structure et les fonctions associées. Pour chaque type d'objet on disposera d'une fonction `créer_type()`, `détruire_type()` et éventuellement de plusieurs fonctions d'initialisation plus de toutes les fonctions nécessaires pour accéder aux champs du type et modifier celui-ci. En résumé, il convient de manipuler ces types comme des types abstraits de donnés en masquant leur implémentation.

On commencera par implémenter les types de base (types énumérés, puis les types ensembles). On implementera d'abord les fonctionnalités de base du plateau avant de calculer les libertés, territoires ou de savoir si un territoire est un seki).

4.1 Les types ensembles

Les types ensembles seront implémentés comme des listes simplement chaînées (on peut faire plus efficace, mais certains trouveront le projet suffisament difficile comme cela).

- Implémentez le type abstrait ensemble_positions qui code un ensemble de positions. On fournira les fonctionnalités, créer, détruire, vide, ajouter, appartient, plus tête, suivant et courrant. Les types positions, libertés et Territoire ne sont que des typedef de ensemble_positions. On définira tout de même un .h pour chacun. De plus on définira dans :
  - libertes.c la fonction :

```
/** @brief Détermine l'ensemble des libertés d'une chaine donnée en fonction de la position des pions sur le plateau*/
Libertes determineLiberte(Plateau plateau,Chaine chaine);
```

- On suppose qu'une chaîne est que d'une seule couleur. Un territoire est également un ensemble de cases vides entourés par une même couleur. Une chaine ou un territoire sera donc composé d'une couleur et d'un ensemble de positions. Définissez un type abstrait ensemble_coloré. Définissez chaîne et territoire comme des typedef de ensemble_coloré. On définira un .h pour chaque type. De plus on définira :
  - Dans territoire.c les fonctions :

```
/** @brief retourne un ensemble d'intersections inoccupées voisines de proche en proche délimitées par des pierres de même couleur en commençant par l'intersection vide à la position pos. Important : Si la case ne fait pas partie d'un territoire de même couleur, retourne quand même l'ensemble des intersections voisines mais en spécifiant que ce "Territoire" n'a aucune couleur. Ce cas est exploité par la fonction estUnSeki */
Territoire determineTerritoire(Plateau plateau,Position pos);
```

    Une pile pourra être utilisée pour determineTerritoire.

```
/** @brief determine si un territoire forme un seki pour les chaines de différentes couleurs concernées. La valeur retournée est de 1 pour vrai et de 0 pour faux */
int estUnSeki(Territoire leTerritoire, Chaines lesChaines,
Plateau plateau);
```

  - Dans chaine.c la fonction

```
/** @brief Determine la position des yeux relatifs à une chaine Si la chaine n'a aucun oeil alors la valeur retournée est NULL */
Positions lesYeuxDeLaChaine(Chaine chaine, Plateau plateau);
```

- Adaptez l'implémentation de ensemble_positions pour créer Pions qui code un ensemble de pionts. On définira les mêmes fonctionnalités.
- Adaptez une dernière fois ensemble_positions pour stocker non plus des coordonnées mais des pointeurs void*. Les types Territoires et Chaînes ne sont que des typedef de ce dernier type.

4.2 Le type plateau

Le type énuméré couleur devra contenir vide, noir et blanc.

Le type plateau sera implémenté comme une structure contenant un tableau de couleurs et les deux dimensions du plateau. A la construction la couleur de chaque case est vide.

En plus des fonctions de construction et destruction, on implémentera les fonctions suivantes :

- Accesseurs :

```
Couleur plateau_get(Plateau plateau,int i, int j);
void plateau_set(Plateau plateau,int i, int j,Couleur couleur);
```

- Détermination de chaîne

```
/** * @brief Produit la chaîne à laquelle appartient le pion à la position pos sur le plateau. S'il n'y a pas de pion sur cette case, alors le résultat retournée est NULL */
Chaine plateau_determiner_chaine(Plateau plateau,Position pos);
```

- Réalisation de capture

```
/** @brief Réalise la capture des pions correspondant à la chaine en les enlevant du plateau. */
void plateau_realiser_capture(Plateau plateau,Chaine chaine);
```

- Egalité de plateau

```
/** @brief indique si l'organisation du plateau est identique à une précédente organisation de plateau. La valeur retournée est de 1 pour vrai et de 0 pour faux */
int plateau_est_identique(Plateau plateau, Plateau ancienPlateau);
```

- Copie de plateau

```
/** @brief Copie un plateau. Les deux tableaux sont supposés déjà alloués*/
int plateau_copie(Plateau from, Plateau to);
```

- Calcul des chaînes entourant un territoire

```
/** @brief Détermine la ou les chaines entourant un territoire */
Chaines plateau_entoure_un_territoire(Territoire leTerritoire,
Plateau plateau);
```

  On pourra pour cela utiliser une pile.

- Entrées/Sorties du plateau :

```
/** Sauvegarde le plateau dans sa position actuelle Si la sauvegarde se passe
sans problème, la fonction retourne 1, sinon 0 */
int plateau_sauvegarde(Plateau plateau,FILE* fichier);
```

```
/** Charge un plateau précédement sauvegardé. Si le chargement n'est pas pos-
sible, retourne un Pointeur NULL */
Plateau plateau_chargement(File* fichier);
```

– Capture de chaînes

```
/** @brief en fonction de la position du pion et de sa couleur retourne les chaines
capturées. Si aucune chaîne n'est capturé par la pose du pion, alors la valeur
NULL est retournée. L'entier (référencé par) valide est égal à zéro si le fait de
placer le poin en cette position conduit à contruire une chaîne sans liberté de
la couleur du pion (sauf si ce coup produit la capture d'au moins une chaine
adverse). Dans le cas contraire l'entier référencé par valide est égale à 1 */
Chaines captureChaines(Plateau plateau,Pion pion, int* valide);
```

4.3 Le type Partie

Ce type correspond à un type abstrait contenant le plateau courant, le précédent et diverses informations précisées dans la section précédente. On implementera nottament les fonctionnalités suivantes :

– Initialisation

```
/** Initialise la partie en fonction des réponses aux différentes questions : *
noms et natures des joueurs, taille du plateau parmi 9x9, 13x13 et 19x19, etc.
Pour rendre cette partie indépendante du mode texte ou mode SDL, une fonc-
tion spécique permettra de prendre en compte ces deux mode. La déclaration
de cette fonction étant la suivante : typedef void (*FonctionQuestions)(int nu-
meroQuestion, Partie* partie) ;
*/
Partie initialisationPartie(FonctionQuestions fonctionQuestions);
```

– Sauvegarde

```
/** Sauvegarde la partie dans sa position actuelle Si la sauvegarde se passe sans
problème, la fonction retourne 1, sinon 0 */
int partie_sauvegarde(Partie partie,FILE* fichier);
```

– Chargement

```
/** Charge une partie précédement sauvegardée. Si le chargement n'est pas
possible, retourne un Pointeur NULL *
Partie partie_charge(FILE* fichier);
```

– Calcul des scores

/** Calcul la valeur des deux scores des deux joueurs en fin de partie * en tenant compte du komi. */
```
void partie_score_joueurs(Partie p, int *scores, float valKomi);
```