Algorithmique avancé, TP1

Nicolas Gast, David Glesser, Clément Pernet, Jean-Louis Roch, Frédéric Wagner

Année 2014-2015

1 Familiarisation

Téléchargez, décompressez et compilez blobwar.

De façon très brève, la structure du programme s'organise en:

- un fichier `main.cc` qui crée un nouvel objet **blobwar** et lance une boucle qui affiche l'interface et attend les divers événements.
- La création de l'objet `blobwar` initialise les données du jeu ainsi que la partie graphique et lie les différents événements graphiques au jeu.
- Le fichier `bidiarray` décrit un tableau à deux dimensions.
- Le fichier `move` décrit les mouvements des pions.
- Le fichier `rules` contient une description des joueurs ainsi que les méthodes du jeu.
- Les fichiers `board,image,mouse,widget,sound,font,label, rollover,blob,button` qui s'occupent de la partie graphique.
- Le fichier `network` s'occupe de la partie réseau.

Par exemple:

- Quand on clique sur "Start 1P game", il lance la fonction "board_selection" qui change les éléments qui apparaissent à l'écran.
- Quand on clique ensuite sur "Start!", il crée un objet de type `rules` construit le jeu puis appelle la méthode "next_turn".
- Quand on appelle la méthode `next_turn`, celle-ci décide quel est le prochain joueur à pouvoir jouer. Si celui-ci est un joueur humain, elle ne fait rien (on attend un clic de sa part). Sinon elle lance dans un nouveau processus `./launchStrategy`. Ce processus calcul un coup dans la classe `Strategy` et le transmet au processus blobwar grâce à de la mémoire partagée.

2 Une première stratégie

Dans le fichier `strategy.cc` se trouve la méthode `computeBestMove()` qui pour l'instant se contente de rechercher les mouvements possibles et s'arrête au premier trouvé. Cette fonction utilise la classe `move` dont les attributs ox, oy, nx, ny représentent le mouvement d'un blob depuis la position (ox, oy) vers la position (nx, ny).

La classe `Strategy` est définie dans le fichier `strategy.h`. C'est principalement à la classe `Strategy` que vous allez vous intéresser dans tous les TP suivants. Cette classe contient:

- des champs qui vont indiquer quelle est la positions courante du jeu à laquelle nous nous intéressons:
  - `bidiarray<Sint16> _blobs;` – le tableau de "blobs",
  - `bidiarray<bool>& _holes;` – les trous du plateau de jeu,
  - `Uint16 _current_player;` – le numéro du joueur courant;
  - `void (*_saveBestMove)(move&)` – la fonction permettant de transmettre à blobwar un coup à jouer;
- des constructeurs pour manipuler la classe:
  - `Strategy(bidiarray<Sint16>& blobs, const bidiarray<bool>& holes,const Uint16 current_player)` – constructeur qu'il faut appeler au début,
  - `Strategy (const Strategy& St)` – constructeur de copie,
  - `~Strategy()` – destructeur;
- des méthodes à implémenter:
  - `void applyMove (move& mv);` – modifie le tableau `_blobs` pour lui appliquer le mouvement décrit par `mv`.
  - `vector<move>& computeValidMoves (vector<move>& _valid_moves) const;` – remplit un vecteur avec l'ensemble des mouvements possibles pour le joueur courant.
  - `Sint32 estimateCurrentScore () const;` – estime la "valeur" du jeu pour le joueur courant (le nombre de blobs à soi moins le nombre de blobs à l'adversaire).
  - `void computeBestMove ();` – calcul le meilleur mouvement calculé par votre stratégie.

Implémentez dans le fichier `strategy.cc` les quatre méthodes de la classe stratégie. Pour les trois premières, vous pouvez vous inspirer de bouts de codes déjà présent dans la classe `rules.cc`. Pour la dernière, on commencera par une stratégie dite "gloutonne":

1. Calculer la liste des coups possibles.
2. Pour chaque coup mv possible, créer un nouvel objet de type Strategy auquel on appliquera mv et calculer la valeur.
3. Rendre le meilleur coup.

N'oubliez pas de tester vos fonctions quand vous programmez.