

Jeu du Pingouin: documentation technique

Francesco Bariatti Adrien Gasté Mikaël Le
Romain Lebouc
Encadrant: Pascal Garcia
2015-2016

Table des matières

1	Introduction	1
2	Implémentation d'un jeu	2
2.1	Implémentation d'un nouveau jeu : l'interface game	2
2.2	Notre implémentation : le Jeu du Pingouin	3
3	L'interface graphique	4

1 Introduction

Ce document est la documentation technique du projet d'études pratiques "Jeu du Pingouin" réalisé dans le cadre de la 3^e année de formation à l'INSA de Rennes.

Le projet est composé de deux parties : une intelligence artificielle codée en C++ et une interface graphique codée en JavaFX. Dans ce document, les abréviations IA ou MCTS (pour Monte Carlo Tree Search, le nom de l'algorithme de recherche utilisé) vont être souvent utilisées pour évoquer l'intelligence artificielle.

Dans la section dédiée à l'IA, nous allons voir comment implémenter un jeu pour qu'il puisse utiliser le MCTS pour jouer contre un humain. Ensuite, nous présenterons comment le Jeu du Pingouin est implémenté.

Auteurs et license

La partie MCTS a été codée par Pascal GARCIA, notre encadrant. Notre groupe a codé la partie Jeu du Pingouin (`penguin.cpp` `penguin.hpp`) et la partie Interface Graphique (dossier `GUI`).

Toutes les parties sont sous licence MIT, avec attribution aux auteurs respectifs.

Dans l'interface graphique, une bibliothèque JSON pour Java est utilisée : `org.json`. La librairie est accessible à l'adresse <http://mvnrepository.com/artifact/org.json/json> : Copyright (c) 2002 JSON.org le copyright complet est présent dans le fichier `GUI/main/Main.java`

2 Implémentation d'un jeu

2.1 Implémentation d'un nouveau jeu : l'interface game

Tous les fichiers évoqués dans cette section se situent dans le dossier `AI/src`.

Un jeu jouable par le MCTS n'est rien d'autre qu'une classe qui implémente l'interface `game`, définie dans `game/game.hpp`. Une fois la nouvelle classe codée, il suffit de changer une ligne dans le fichier `main/main.cpp` pour lancer le jeu. Deux façons sont possibles pour lancer le jeu :

- `game::run_test_two_players_game(game::MaClasse());` pour lancer un test à deux joueur : le MCTS ne va pas se lancer.
- `mcts::run_test_mcts_two_players(game::MaClasse());` pour lancer une partie contre le MCTS.

Dans l'interface, un joueur est représenté par un nombre sur 8bits (le premier joueur est le joueur 0, le deuxième le joueur 1). Chaque fin de partie a une valeur qui est utilisée dans la construction de l'arbre de décision (classiquement 1 si le MCTS gagne, -1 s'il perd).

Pour implémenter l'interface, il faut implémenter toutes ses fonctions. Elles ne sont pas toutes indispensables ou dépendantes du nouveau jeu, c'est pourquoi une partie d'entre elles pourront être directement copiées du fichier `penguin.cpp`. Les explications des fonctions indispensables sont listées ci-dessous :

- `bool end_of_game()` Indique si le jeu est terminé. Indépendamment de son résultat.
- `int value (uint8 player)` La valeur pour le joueur `player` dans l'état actuel du jeu.
- `bool won/lost/draw(uint8 player)` Si le joueur `player` a gagné/perdu/fait égalité.
- `uint8 current_player()` Le joueur dont c'est le tour.
- `uint16 number_of_moves()` Le nombre total de coups que le joueur actuel peut jouer.
- `play(uint16 move)` Joue le coup numéro `move` pour le joueur actuel.
- `string player_to_string(uint8 player)` Représentation d'un joueur de façon compréhensible pour un humain.
- `string move_to_string(uint16 m)` Représentation d'un coup jouable de façon compréhensible pour un humain.
- `string to_string()` Représentation de l'état actuel du jeu de façon compréhensible pour un humain.

La fonction `move_to_string` est importante car c'est la représentation que l'humain devra utiliser pour spécifier le coup qu'il veut jouer.

Il ne faut pas oublier d'ajouter les nouveaux fichiers dans le `makefile` !

2.2 Notre implémentation : le Jeu du Pingouin

Pour plus d'informations sur les structures et méthodes utilisées pour modéliser le Jeu du Pingouin, regarder le rapport du projet, qui se trouve dans le dossier `Doc/fr/Rapport`.

Le fichier `penguin.hpp` définit la structure `penguin_state`, qui décrit un état du jeu. Cette structure définit les bitboards des poissons, les bitboards des pingouins des 2 joueurs, les scores, le joueur courant et le nombre de coups de chaque joueur.

Les fichiers `penguin.hpp` et `penguin.cpp` définissent la classe `penguin`, implémentant l'interface `game`, qui décrit un jeu. En plus des méthodes de l'interface, cette classe implémente les méthodes suivantes :

- `penguin_state get_state()` Retourne l'état du jeu
- `void move_penguin(uint32_t* p, uint16_t rel_move)` Déplace le pingouin `p` lui faisant faire son coup numéro `rel_move`. À la fin de la fonction, le pingouin n'est plus composé que de sa nouvelle position (tous les autres bits sont à 0).
- `uint64_t create_obstacles_bitboard()` Crée le bitboard des obstacles : pour chaque case, le bit correspondant est à 1 s'il y a un obstacle, 0 sinon.
- `int update_penguin_moves(uint32_t* p, uint64_t obstacles)` Met à jour tous les coups du pingouin `p` en fonction de sa position et du bitboard des obstacles. Retourne le nombre total de coups du pingouin.

Par ailleurs, le constructeur de la classe `penguin` gère le chargement d'un état via l'entrée standard du programme.

3 L'interface graphique

Tous les fichiers évoqués dans cette section se situent dans le dossier `GUI/src`. L'interface graphique est développée en Java 1.8 (elle utilise notamment des lambda-expressions) et JavaFX.

L'interface graphique est codée suivant un modèle MVC (Modèle - Vue - Contrôleur).

Le modèle : il sert à stocker toutes les données du jeu. Les classes sont utilisées comme conteneurs pour stocker les valeurs nécessaires au jeu.

La classe principale est la classe `GameState` qui contient la plupart des informations relatives au jeu. Les valeurs sont stockés en forme de bitboards et utilise des opérations binaires sont utilisées pour les extraire quand nécessaire.

Une méthode permet de mettre à jour un objet `GameState` à partir du JSON renvoyé par le MCTS.

La vue : Le visuel de l'application est décrit dans le fichier `view.fxml`, les différentes cases sont définies avec un positionnement absolu et ont un id qui vont de `tile0` à `tile59`. Les autres éléments sont aussi positionnés de façon absolue.

Cette façon de faire n'est pas celle préconisée (elle n'est pas "propre", nous nous en excusons). L'interface graphique n'étant pas l'objectif initial du projet, nous y avons alloué moins de temps.

Le contrôleur : il s'occupe de faire le lien entre le modèle, la vue et le programme en C++. Pour ce faire, à l'initialisation, l'IA sera exécutée comme sous-processus. Un `UpdateThread` s'occupera de modifier le modèle à chaque fois que le sous-processus va écrire quelque chose dans sa sortie standard, et ensuite notifier la vue de se mettre à jour aussi.

La classe `Controller` est la classe "principale" du programme : elle s'occupe de créer le plateau, lancer la partie, écouter les clics du joueur au travers du `EventHandler`, et de gérer la fin de partie (à la demande de l'`UpdateThread`).

Lorsque l'interface graphique est fermée (peu importe le moment) le sous-processus est tué. Ceci se fait grâce à la fonction Java :

```
Runtime.getRuntime().addShutdownHook(new Thread()  
{  
    public void run()  
    {  
        gameProcess.destroy();  
    }  
});
```