Puissance 4

PAN Liuyan , CHABALIER Andy

Projet Intelligence Artificielle ENSISA 2A IR

Repository git du projet: https://git.chabalier.com/snowert/puissance4

Objectifs

- Implementer les algorithmes de jeu Min-Max et Alpha-Beta pour un Puissance 4
- Ecrire une fonction d'évaluation permettant de donner un score a une grille non terminale
- paralléliser les calculs pour accélerer le temps de reflexion des IA et les rendres plus efficaces
- Réaliser une interface graphique pour le jeu tout en proposant la possibilité de jouer en console

Algorithmes

Les IA disposent de deux algorithmes pour prévoir et jouer. La fonction d'évaluation des grilles est unique pour tout les algorithmes Il est possible d'ajouter d'autres algorithmes sans toucher aux autres et au fonctionnement de l'application.

Algorithme Min-Max

```
function MINIMAx-DECISION(state) returns an action
        inputs: state, current state in game

        return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
        if TERMINAL-TEST(state) then return UTILITY(8tate)
```

```
            v <— —00
            for a, s in SUCCESSORS(state) do w— MAx(v. MIN-VALUE(3))
            return v

    function MIN-VALUE(state) returns a utility value
            if TERMINAL-TEST(state) then return UTILITY(state)
            v <— 00
            for a, s in SUCCESSORS(state) do w— MIN(v. MAX-VALUE(3))
            return v
```

Algorithme Alpha-Beta

```
    function ALPHA—BETA-DECISION(state) returns an action
            return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state»

    function MAX-VALUE(state, alpha ,beta) returns a utility value
            inputs: state. current state in game

            alpha, the value of the best alternative for MAX along the path to
state
            beta. the value of the best alternative for MIN along the path to state

            if TERMINAL-TEST(state) then return UTILITY(state)
            v i— -00
            for a, s in SUCCESSORS(state) do
                v<— MAX(v. MIN-VALUE(s,alpha,beta)
                if v >= beta then return 2)
                alpha *— Max(alpha,v))
            return v

    function MIN-VALUE(state, alpha, beta) returns a utility value
            same as MAX-VALUE but with roles of alpha,beta reversed
```

Fonctionnement

Le projet est construit autour d'une architecture Model - View - Controller.

Le modele est commun a toutes les IHM et comporte les packages "data" et "jeu".

Le package "data" contient les différentes classes permettant de définir les acteurs du jeu. C'est à dire la grille, les joueurs et la partie qui les englobent. Il existe deux types de joueurs. Les joueurs Humains et les Intelligences Artificielles. La classe partie est celle qui fait le lien entre les joueurs et la grille. Elle s'occupe de la gestion des tours.

Le package "jeu" contient les différents algorithmes des Intelligences Artificielles. La classe jeu est seulement utilisée par l'application en mode console. Héritant de "Thread" elle permet de lancer la partie et d'afficher sur la sortie standard.

le package "UI" contient les controlleurs, les différentes vues et quelques ressources.

Parmis les controlleurs, le controlleur de la console s'occupe de faire le pont entre la partie et les

vues qui sont les entrées et sorties standards.

Le controlleur "GUI" s'occupe de lancer l'affichage principal et y inserer la vue de choix. "GUI" est couplé avec "RootView.fxml" et "MenuController" qui gère les évènements de la barre de menu.

Le controlleur "ChoiceController" s'occupe de la gestion de la vue "ChoiceView.fxml" qui permet de récuperer les choix des joueurs et lancer la partie.

Le controller "GameController" s'occupe de la gestion de la vue "GameView.fxml" qui est composé de la grille et d'une zone d'historique. "GameController" s'occupe de lancer la partie et les différents tour et d'afficher la grille et les etapes dans la zone d'historique. Le choix du coup a joueur pour les joueurs humains est demandé par le biais d'une boite de dialogue.

Divers

Pour les deux algorithmes Min-Max et Alpha-Beta, le choix du coup à est légèrement parralélisé. le programme lance un thread par coup initiaux, soit septs. C'est un pool de threads qui s'occupe de leur création et génération.

Extrait de l'algorithme Alpha-Beta: ExecutorService threadPoolExecutor = new ThreadPoolExecutor(corePoolSize, maxPoolSize, keepAliveTime, TimeUnit.MILLISECONDS, new LinkedBlockingQueue());

```
    for (int i = 0; i < Constantes.NB_COLONNES; i++) {
        final int coup = i;

        threadPoolExecutor.submit(new Runnable() { //we submit new task to explore
a branch of the decision tree

            @Override
            public void run() {
                int profondeur = levelIA;
                Grille virtualGrille = new Grille(grilleDepart.getGrille());
                virtualGrille.ajouterCoup(coup, symboleMax);
                double val = minValue(virtualGrille, profondeur - 1, tourSimule);

                result.put(val, coup);
            }
        });
    }

    try {
        threadPoolExecutor.shutdown(); //Lock the Pool, execute all previous
submited task.
        threadPoolExecutor.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS);
//Wait for task's execution
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```