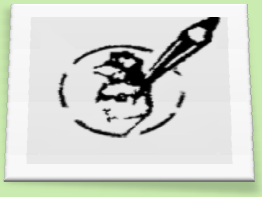


One Man Army

Push'em all

IA

Algorithmes

Avant-propos

Objectif du présent document

Ce document présente les différents algorithmes utilisés pour l'intelligence artificielle. Plus particulièrement, seront traités les différents comportements face à l'utilisation des cartes spéciales par l'IA.

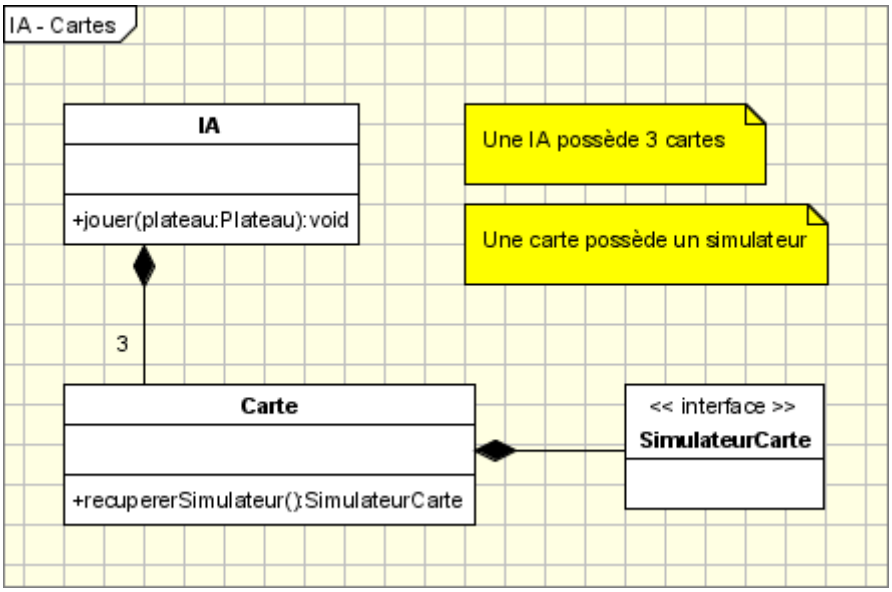
Historique des versions

Le tableau suivant retrace l'historique du document :

Auteur	Date	Version	Changements
Houssem Achouri	16/11/2008	1.0	Version initiale : copie du wiki : http://onemanarmy.free.fr/wiki

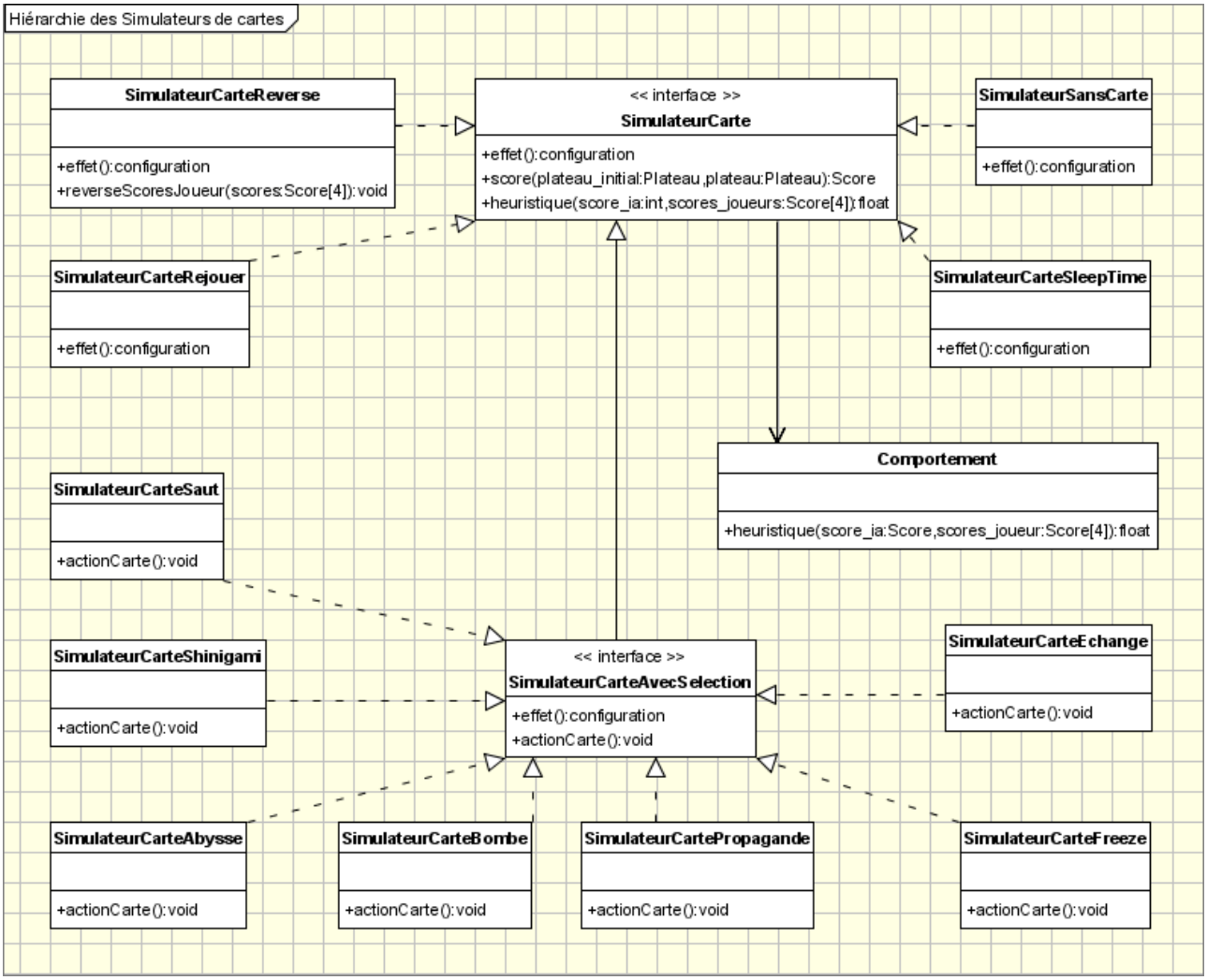
Licence

Principe général




```
fonction IA::jouer(plateau_initial) : configuration
Pour Chaque carte disponible Faire
   // structure représentant une carte + une sélection + deux directions + un
   // score
   configuration;

   // Récupérer l'objet simulant l'effet d'une carte sur un plateau donnée
   simulateur = carte.recupererSimulateur();
   // Simuler l'effet de la carte
   configuration = simulateur.effet(plateau_initial);

   // Sauvegarder la configuration carte
   tableau_des_scores.ajouter(configuration);
Fin Pour

// Récupérer la meilleure configuration sans carte
configuration = SimulateurSansCarte.effet(plateau_initial);
tableau_des_scores.ajouter(configuration);

retourner meilleureConfiguration(tableau_des_scores);
```

Les simulateurs de cartes



Rejouer

```
fonction SilmulateurCarteRejouer::effet(plateau) : configuration
// Initialisation de la configuration
configuration.carte = recupererTypeCarte();
configuration.selection = null;
configuration.score = 0;

Pour Chaque direction Faire
    // On copie le plateau afin de pouvoir le modifier
    plateau_intermedaire = plateau.copie();

    // On déplace l'IA selon la direction courante puis on récupère le score
    plateau_intermediaire.deplacerEquipe(ia, direction);
    score_ia = score(plateau, plateau_intermediaire);

    // Rejouer
    Pour Chaque direction2 Faire
        plateau_intermediaire2 = plateau_intermediaire.copie();

        plateau_intermediaire2.deplacerEquipe(ia, direction2);
        score_ia2 = score(plateau_intermediaire, plateau_intermediaire2);

        // Simuler les actions du joueur
        Pour Chaque direction_joueur Faire
            // Déplacer l'équipe du joueur selon la direction courante
            plateau_intermediaire2.deplacerEquipe(joueur, direction_joueur);
            // Sauvegarder le score pour la direction courante
            scores_joueur.direction = score( plateau_intermediaire,
                                             plateau_intermediaire2 );
        Fin Pour

        // On calcule le score final grâce à l'heuristique
        score_final = heuristique(score_ia + score_ia2, scores_joueur);

        // Traitement du score
        Si configuration.score < score_final Alors
            configuration.score = score_final;
            configuration.direction = direction;
            configuration.direction2 = direction2;
        Fin Si
    Fin Pour
Fin Pour

retourner configuration;
```

Sleep Time

fonction SimulateurCarteSleepTime::effet(plateau) : configuration

```
// Initialisation de la configuration
configuration.carte = recupererTypeCarte();
configuration.selection = null;
configuration.direction = null;
configuration.score = 0;

// On déplace l'équipe du joueur et on récupère le score pour chaque direction
Pour chaque direction_joueur Faire
    plateau_intermediaire = plateau.copie();
    plateau_intermediaire.deplacerEquipe(joueur, direction_joueur);
    scores_joueur.direction_joueur = score(plateau, plateau_intermediaire);
Fin Pour

// L'IA n'ayant pas joué, son score ne bouge pas
score_ia = score(plateau, plateau);
// Calcul du score final via l'heuristique
configuration.score = heuristique(score_ia, scores_joueur);

retourner configuration;
```

Reverse

fonction SimulateurCarteReverse::effet(plateau) : configuration

```
configuration.carte = recupererTypeCarte();
configuration.selection = null;
configuration.score = 0;

Pour chaque direction faire
   plateau_temporaire = plateau.copie();
   plateau_temporaire.deplacerEquipe(ia, direction);
   score_ia = score(plateau, plateau_temporaire);

   Pour chaque direction_joueur faire
       plateau_temporaire.deplacerEquipe(joueur, direction_joueur);
       scores_joueur.direction_joueur = score(plateau, plateau_temporaire);
   Fin Pour

   reverseScoresJoueur(scores_joueur);

   score_final = heuristique(score_ia, scores_joueur);
   Si configuration.score < score_final Alors
       configuration.score = score_final;
       configuration.direction = direction;
   Fin Si
Fin Pour

retourner configuration;
```

fonction SimulateurCarteReverse::reverseScoresJoueur(scores[4])

```
echanger(score.haut, score.bas);
echanger(score.gauche, score.droite);
```

Cartes avec sélection

```
fonction SimulateurCarteAvecSelection::effet(plateau) : configuration
// On récupère le type de la carte
configuration.carte = recupererTypeCarte();

// On récupère également les sélections possibles pour ce simulateur
selections = recupererSelections(plateau);

Pour Chaque selection dans selections Faire
   Pour Chaque direction Faire
       plateau_temporaire = plateau.copie();

       // On déplace l'équipe de l'IA
       plateau_temporaire.deplacerEquipe(ia, direction);
       score_ia = score(plateau, plateau_temporaire);

       // On exécute l'action de la carte
       actionCarte(selection, direction, plateau_temporaire);

       // On simuler les déplacements du joueur
       Pour chaque direction_joueur faire
           plateau_temporaire2 = plateau_temporaire.copie();
           plateau_temporaire.deplacerEquipe(joueur, direction_joueur);
           scores_joueur.direction_joueur = score( plateau_temporaire,
                                                   plateau_temporaire2 );
       Fin Pour

       // Calcul du score final
       score_final = heuristique(score_ia, scores_joueur);
       Si configuration.score < score_final Alors
           configuration.score = score_final;
           configuration.direction = direction;
           configuration.selection = selection;
       Fin Si
   Fin Pour
Fin Pour

retourner configuration;
```

Sélection d'un pion adverse

```
fonction SimulateurCarte[...]::recupererSelections(plateau) : selection[]
Pour Chaque pion_joueur de plateau Faire
   selection.pion = pion_joueur;
   selections.ajouter(selection);
Fin Pour

retourner selections
```

Shinigami

```
fonction SimulateurCarteShinigami::actionCarte(selection, direction, plateau)
plateau.eliminerPion(selection.pion);
```

Propagande

```
fonction SimulateurCartePropagande::actionCarte(selection, direction, plateau)
plateau.changerEquipePion(selection.pion);
```

Freeze

```
fonction SimulateurCarteFreeze::actionCarte(selection, direction, plateau)
plateau.gelerPion(selection.pion);
```

Saut

```
fonction SimulateurCarteSaut::recupererSelection(plateau) : selection[]
Pour Chaque pion_ia qui n'est pas sur le bord de plateau Faire
   selection.pion = pion_ia;
   selections.ajouter(selection);
Fin Pour

retourner selections;
```

```
fonction SimulateurCarteSaut::actionCarte(selection, direction, plateau)
plateau.deplacerPion(selection.pion, direction);
```

Bombe

```
fonction SimulateurCarteBombe::recupererSelections(plateau) : selection[]
Pour Chaque ligne de plateau Faire
   selection.rangee = ligne;
   selections.ajouter(selection);
Fin Pour

Pour Chaque colonne de plateau Faire
   selection.rangee = colonne;
   selections.ajouter(selection);
Fin Pour

retourner selections;
```

```
fonction SimulateurCarteBombe::actionCarte(selection, direction, plateau)
plateau.eliminierRangee(selection.rangee);
```

Echange

```
fonction SimulateurCarteEchange::recupererSelection(plateau) : selection[]
Pour Chaque pion_ia de plateau Faire
   Pour Chaque pion_joueur de plateau Faire
       selection.pion1 = pion_ia;
       selection.pion2 = pion_joueur;
       selections.ajouter(selection);
   Fin Pour
Fin Pour

retourner selections;
```

```
fonction SimulateurCarteEchange::actionCarte(selection, direction, plateau)
plateau.echangerPions(selection.pion1, selection.pion2);
```

Abysse

```
fonction SimulateurCarteAbysse::recupererSelection(plateau) : selection[]
Pour Chaque case_libre de plateau Faire
    selection.case = case_libre;
    selections.ajouter(selection);
Fin Pour

retourner selections;
```

```
fonction SimulateurCarteAbysse::actionCarte(selection, direction, plateau)
plateau.detruireCase(selection.case);
```

Sans carte

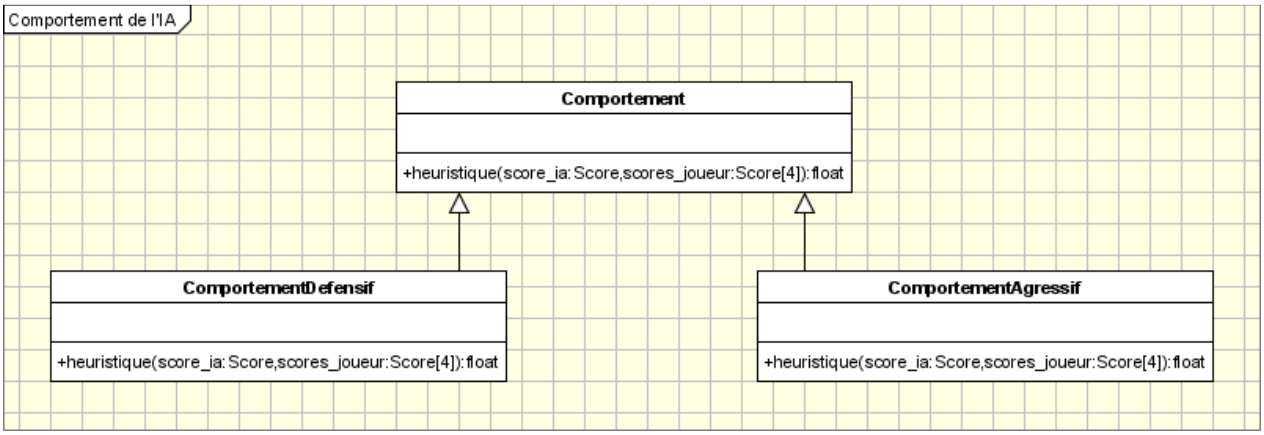
```
fonction SimulateurSansCarte::effet(plateau) : configuration
configuration.carte = null;
configuration.selection = null;
configuration.score = 0;

Pour chaque direction faire
   plateau_temporaire = plateau.copie();
   plateau_temporaire.deplacerEquipe(ia, direction);
   score_ia = score(plateau, plateau_temporaire);

   Pour chaque direction_joueur faire
       plateau_temporaire.deplacerEquipe(joueur, direction_joueur);
       scores_joueur.direction_joueur = score(plateau, plateau_temporaire);
   Fin Pour

   score_final = heuristique(score_ia, scores_joueur);
   Si configuration.score < score_final Alors
       configuration.score = score_final;
       configuration.direction = direction;
   Fin Si
Fin Pour

retourner configuration;
```

Score

```
fonction SimulateurCarte::score(plateau_initial, plateau) : score
// Le score est une structure de 4 entiers représentant le nombre de pions
// restant pour chaque équipe, ainsi que le nombre de pions éliminés d'un
// plateau à l'autre
score.pions_restants_ia = plateau.pionsRestants(ia);
score.pions_elimines_ia = plateau_initial.pionsRestants(ia)
                        -  score.pions_restants_ia;

score.pions_restants_joueur = plateau.pionsRestants(joueur);
score.pions_elimines_joueur = plateau_initial.pionsRestants(joueur)
                            - score.pions_restants_joueur;

retourner score;
```

Heuristique

```
fonction SimulateurCarte::heuristique(score_ia, scores_joueur[4]) : float
// La classe SimulateurCarte devra avoir accès a une classe Comportement
// qui elle implémente réellement la méthode heuristique.
// On pourra avoir ainsi plusieurs comportements pour l'IA
retourner comportement.heuristique(score_ia, score_joueur);
```



Comportement agressif

```
fonction ComportementAgressif::heuristique(score_ia, scores_joueur[4]) : float

// Privilégier le nombre de pions adverses éliminés sur les pions alliés
// restants

retourner score_final;
```

Comportement défensif

```
fonction ComportementDefensif::heuristique(score_ia, scores_joueur[4]) : float

// Privilégier le nombre de pions alliés restants sur le nombre de pions
// adverses éliminés

retourner score_final;
```