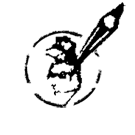
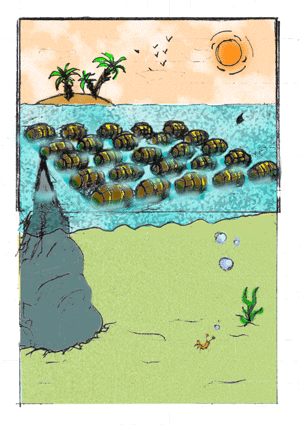
**  
One Man Army**



Push’Em All

IA

Algorithmes

[Avant-propos 3](#_Toc214617333)

[Objectif du présent document 3](#_Toc214617334)

[Historique des versions 3](#_Toc214617335)

[Licence 3](#_Toc214617336)

[Principe général 4](#_Toc214617337)

[Les simulateurs de cartes 5](#_Toc214617338)

[Rejouer 6](#_Toc214617339)

[Sleep Time 7](#_Toc214617340)

[Reverse 7](#_Toc214617341)

[Cartes avec sélection 8](#_Toc214617342)

[Sélection d'un pion adverse 8](#_Toc214617343)

[Shinigami 8](#_Toc214617344)

[Propagande 8](#_Toc214617345)

[Freeze 9](#_Toc214617346)

[Saut 9](#_Toc214617347)

[Bombe 9](#_Toc214617348)

[Echange 9](#_Toc214617349)

[Abysse 10](#_Toc214617350)

[Sans carte 10](#_Toc214617351)

[Score 10](#_Toc214617352)

[Heuristique 11](#_Toc214617353)

[Comportement agressif 11](#_Toc214617354)

[Comportement défensif 11](#_Toc214617355)

# 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

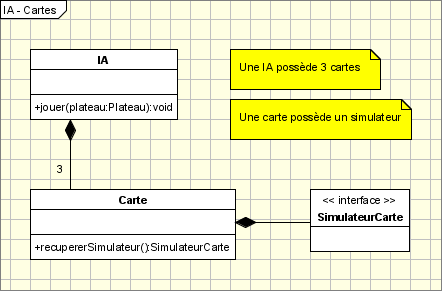
Le tableau suivant retrace l’historique du document :

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

## Licence

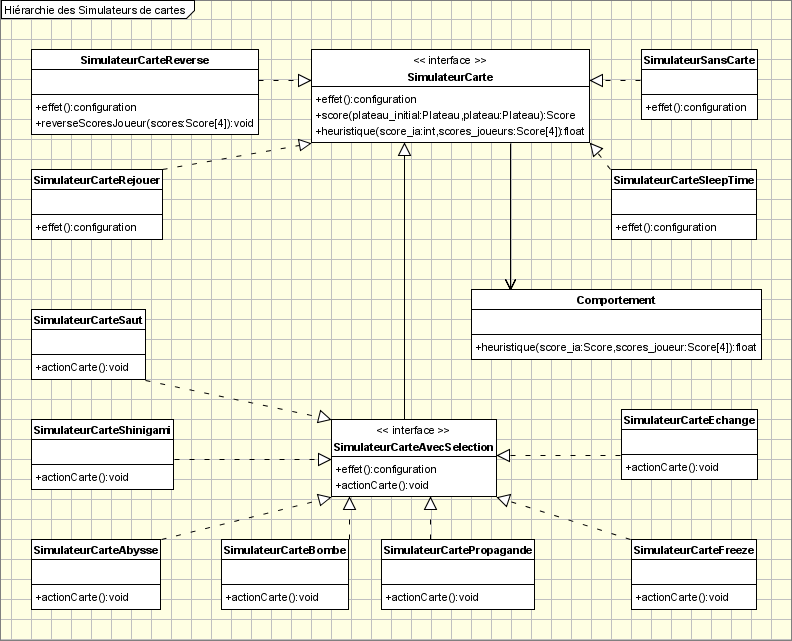
Cette création est mise à disposition selon le contrat **Creative Commons Paternité-Pas d'Utilisation Commerciale-Partage des Conditions Initiales à l'Identique** 2.0 France License, disponible en ligne à l’adresse suivante : http://creativecommons.org/licenses/by-nc-sa/2.0/fr/

# 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

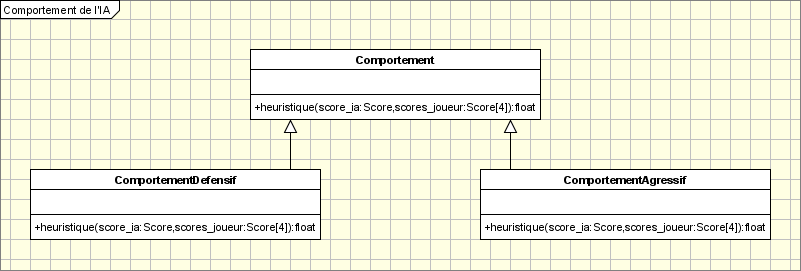
|  |
| --- |
| fonction SimulateurSansCarte::effet(plateau) : configuration |
| configuration.carte = null;  configuration.selection = null;  configuration.score = 0;  **Pour chaque** direction **faire**  plateau\_temporaire = plateau.copie();  plateau\_temporaire.deplacerEquipe(ia, direction);  score\_ia = 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; |