

Définitions

Action On définit deux actions :

- Bouger l'unité a à la position p
- Attaquer l'unité b avec l'unité a
- Attendre t secondes

Le jeu est formé de 3 classes et 2 fonctions :

Unité $u = (p, hp, t_a, t_m, type)$

- $p = (x, y)$ la position de l'unité
- hp la vie de l'unité
- t_a le cooldown avant que l'unité puisse attaquer
- t_m le cooldown avant que l'unité puisse bouger
- $type$ le type de l'unité

État $s = (t, U_1, U_2, \dots, U_k)$ contenant toutes les informations du jeu nécessaires :

- t le temps
- $U_i = (u_1, \dots, u_k)$ l'ensemble des unités contrôlées par le joueur i

Mouvement $m = (a_1, \dots, a_k)$ séquence d'actions $a_i = (u, type, cible, t)$:

- u l'unité qui effectue l'action
- $type$ le type d'action

Joueur fonction $p(s, U) = m$ prenant l'état du jeu s et la liste des unités U et renvoyant les actions choisies par l'algorithme

Jeu fonction $g(s, p_1, p_2, \dots, p_k) = s'$ prenant l'état du jeu et les fonctions des joueurs et effectuant les actions

Les *IA* développées ici seront donc des fonctions player.

1 Upper Confidence bounds applied to Trees

Évolution de l'algorithme de recherche arborescente de Monte-Carlo. L'algorithme recherche dans l'arbre des possibles. La formule du UCT est :

$$UCT = \overline{X}_j + C_p \sqrt{\frac{2 \ln(n)}{n_j}}$$

avec :

- n le nombre de fois que le parent a été visité
- n_j le nombre de fois que l'enfant a été visité
- \overline{X}_j le ratio de victoire :

$$\overline{X}_j = \frac{\text{victoires} + \frac{\text{égalités}}{2}}{\text{victoires} + \text{égalités} + \text{défaites}}$$

- C une constante permettant d'ajuster le nombre d'exploration de chaque noeud

L'algorithme classique UCT n'est pas adapté aux jeux en temps réel. Ainsi nous allons considérer ici une évolution, le UCT Considering Durations (UCTCD) qui permet de gérer faire plusieurs actions en même temps. L'algorithme se déroule en quatre étapes :

1. On prends deux listes d'actions des algorithmes *NOK-AV* et *Kiter*.

2 La recherche glouton par portfolio

3 La recherche hiérarchique par portfolio

Un problème fréquent des gens de stratégie est la taille des cartes, rendant trop lourds les algorithmes classiques d'ia de jeu comme l'algorithme alpha-béta. En effet, le nombre d'actions possibles que l'algorithme UCT considère est L^U avec L le nombre d'action possible moyen et U le nombre d'unités possibles. Nous étudierons ici une évolution de la recherche glouton par portfolio : la recherche hiérarchique par portfolio (HPS) a été inventée. L'algorithme ajoute une fonction et une classe :

Joueur partiel fonction $pp(s) = m$ joueur partiel

elle est similaire à la fonction joueur p mais au lieu de calculer les actions de toutes les unités elle calcule celle d'une seule unité.

Portfolio $P = (pp_1, pp_2, \dots, pp_k)$ ensemble des joueurs partiels

L'algorithme HPS permet alors de changer un nombre de combinaison exponentiel d'action en nombre linéaire d'action.