# Définitions

**Action** On définit deux actions :
- Bouger l'unitée $a$ à la position $p$
- Attaquer l'unitée $b$ avec l'unité $a$
- Attendre $t$ secondes

Le jeu est formé de 3 classes et 2 fonctions :

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

**État** $s = (t, U_1, U_2, ..., U_k)$ contenant toutes les informations du jeu nécessaires :
- $t$ le temps
- $U_i = (u_1, ..., u_k)$ l'ensemble des unitées controlées par le joueur i

**Mouvement** $m = (a_1, ..., a_k)$ séquence d'actions $a_i = (u, type, cible, t)$ :
- $u$ l'unitée 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 unitées $U$ et renvoyant les actions choisies par l'alghorithme

**Jeu** fonction $g(s, p_1, p_2, ..., p_k) = s'$ 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 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 noed

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

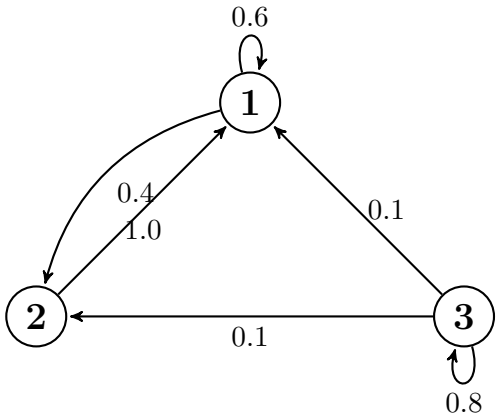
Un problème fréquent des gens de stratégie est la taille des cartes, rendant trop lourds les algorithmes classiques d'ia de jeu comme l'algorithme alpha-béta. En effet, le nombre d'actions possibles que l'algorithme UCT considère est $L^U$ avec $L$ le nombre d'action possible moyen et $U$ le nombre d'unités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

**Joueur partiel** fonction $pp(s) = m$ joueur pariel
elle est similaire à la fonction joueur $p$ mais au lieu de calculer les actions de toutes les unitées elle calcule celle d'une seule unitée.

**Portfolio** $P = (pp_1, pp_2, ..., pp_k)$ ensemble des joueurs partiels

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

## 4 Test




La boucle du jeu est :

On effectue des tests avec un joueur choisissant toutes ses actions de manière aléatoire.