Étude de quelques algorithmes de joueurs artificiels participants à des jeux de stratégie en temps réel

Dimitri Cocheril-Crèvecœur
13960

2 juin 2024

Plan

1. Présentation du problème
2. Moteur
3. Stratégies testées

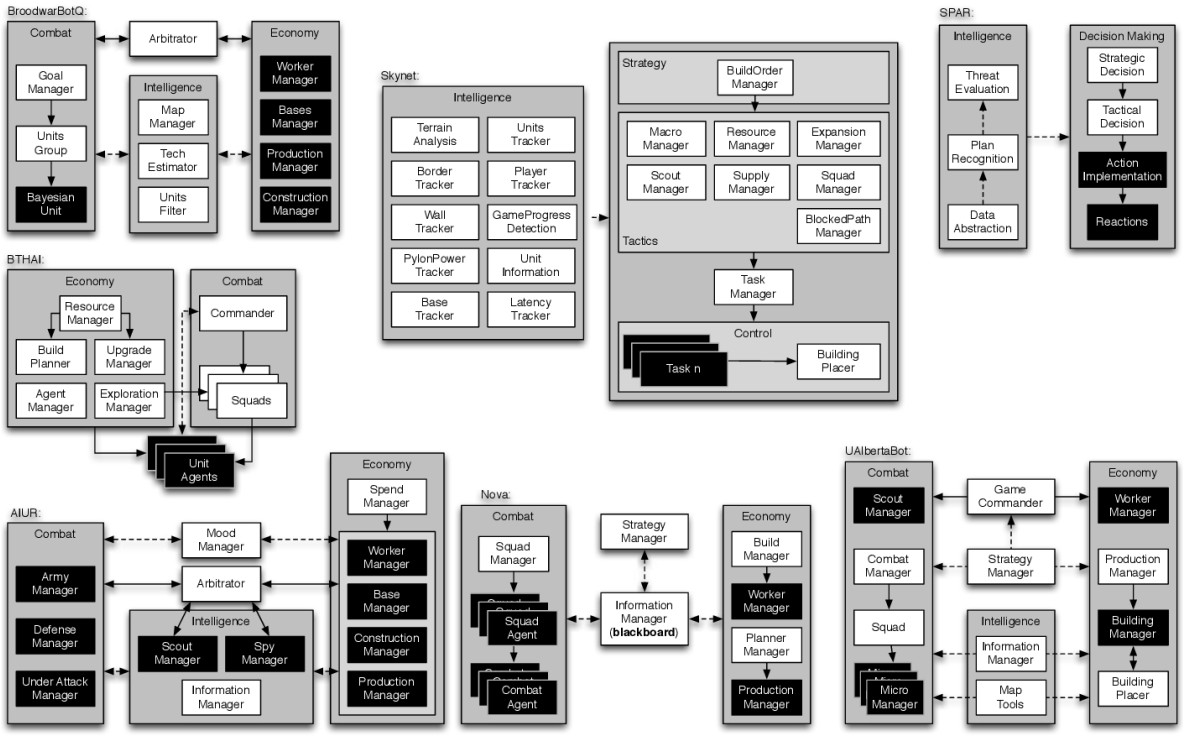
StarCraft

Jeu de stratégie en temps réel :

- Jeux simultané
- Deux joueurs s'affrontent pour le contrôle d'une carte
- Gestion de ressources : minage, création d'unitées, de bases (stratégie)
- Combat entre unitées (tactique)

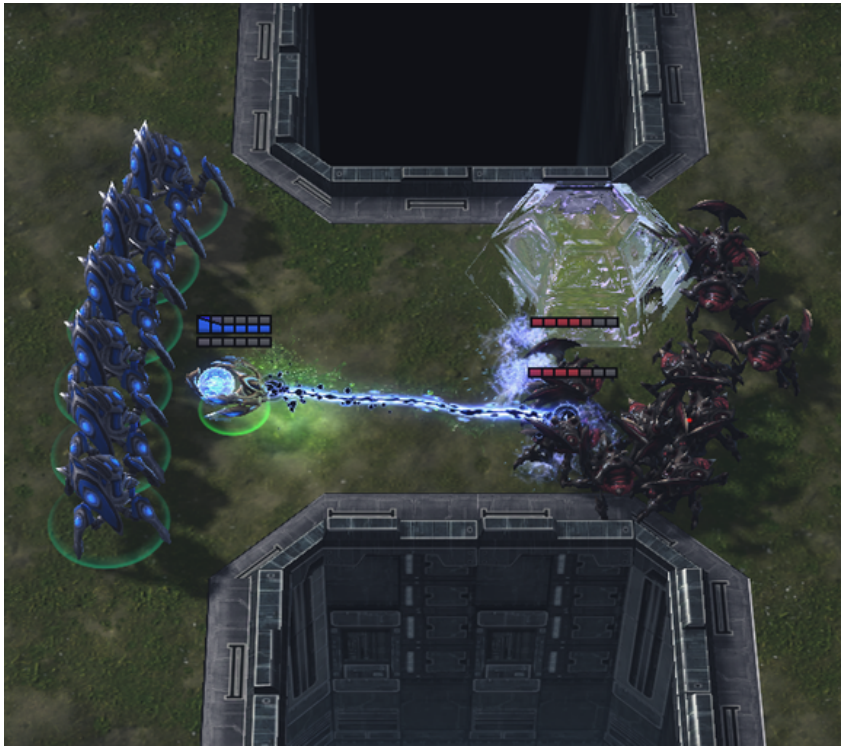
Modèles existants

- IA de Google : *AlphaStar*
  Apprentissage supervisé puis par renforcement
- Robots ("bots") :



StarCraft

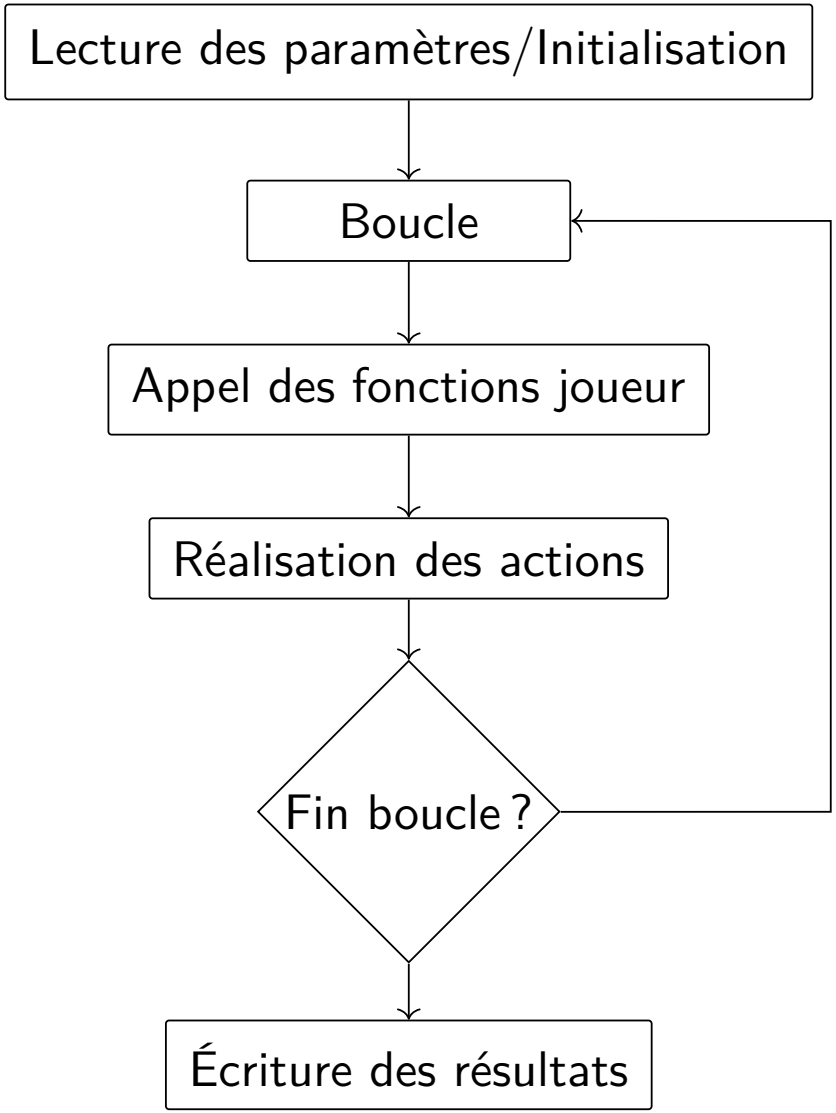
On se concentre ici sur la partie "combat" :
Deux joueurs disposent d'unitées pouvant bouger et attaquer celles adverses
Chaque joueur doit tuer toute les unitées ennemies

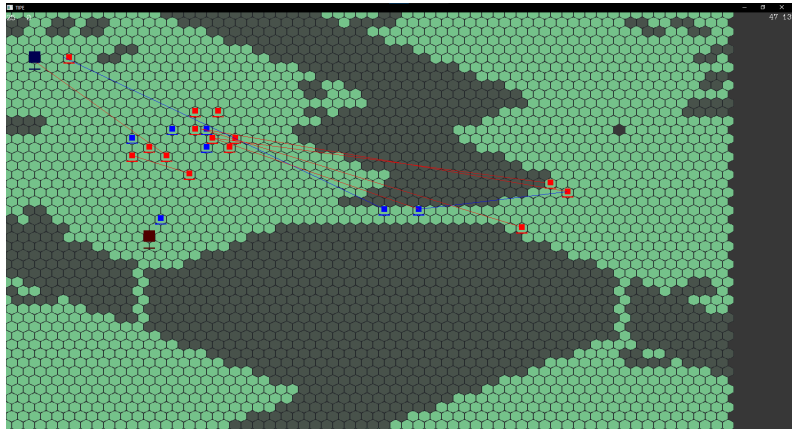


Problème

1. Facteur de branchement entre 10^{50} et 10^{200}
2. Durée typique d'une partie : 25 minutes donc $25 \times 60 \times 24 = 36000$ états

Développement moteur du jeu





```
void game_class::play() const
{
    for (const auto p : players_)
    /*Pour chaque joueur...*/
    {
        p->moves_get(this, state_);
        /*..on choisit les actions..*/
    }
    state_->moves_make(map_);
    /*..et on effectue les actions.*/
}
```

Problème de la recherche de chemin

- Optimisation compliquée :
  On utilise un A* pondérée à 5 (trouvé de manière expérimentale)
- Parrallélisation du programme
  On alloue les espaces dynamiquements au lieu d'utiliser le tas pour éviter la conccurence

Parrallélisation :

Joueur aléatoire

Joueur témoin : choisis une action aléatoire, et l'effectue avant d'en choisir une autre aléatoirement

On effectue 10000 combats sur une carte vide entre deux joueurs aléatoires avec 50 unités chacun :

Victoires de joueur 0 : 4689
Victoires de joueur 1 : 5306
Égalités : 5

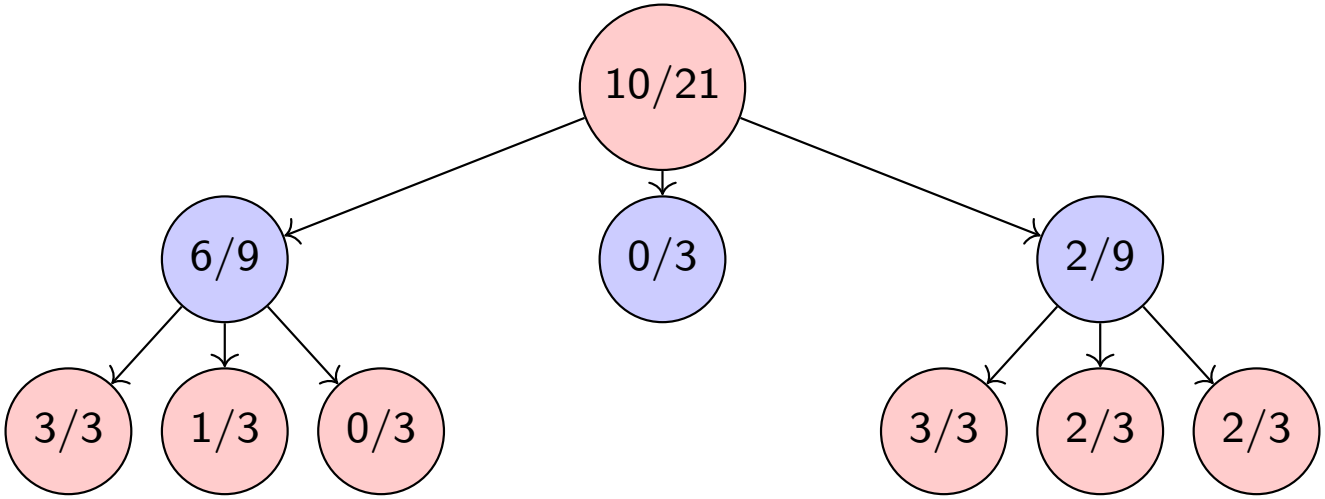
Stratégie naïve : attaque par puissance

Toutes nos unités attaquent l'unitée ennemie avec la plus grande puissance
Résultats contre joueur aléatoire avec 50 unités :
100% de victoire pour joueur aléatoire

Joueur MCTS

Trop grand nombre de possibilités à chaque tick : plus de 10^50
On utilise la recherche arborescente Monte-Carlo

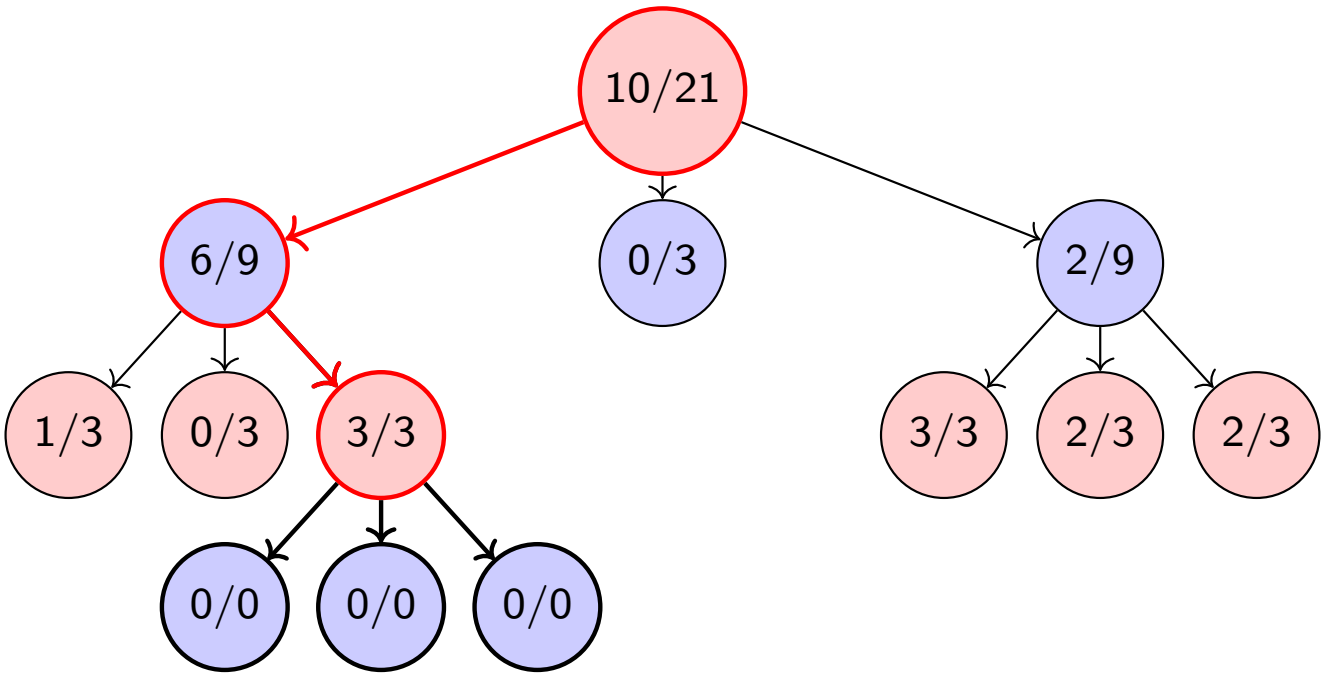
Utilisation MCTS



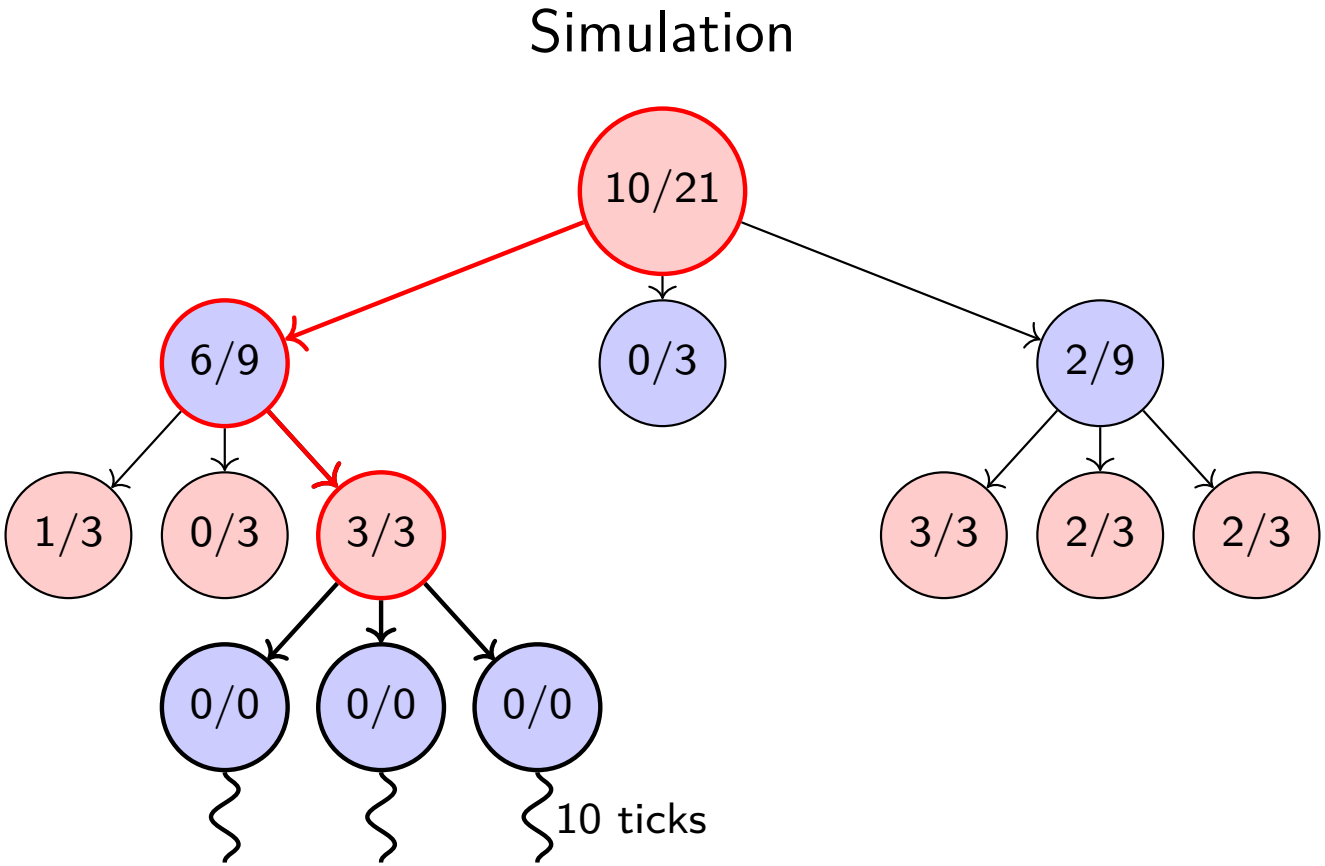
Utilisation MCTS

Utilisation de UCT pour la selection : $\frac{w}{n} + c\sqrt{\frac{\ln N}{n}}$

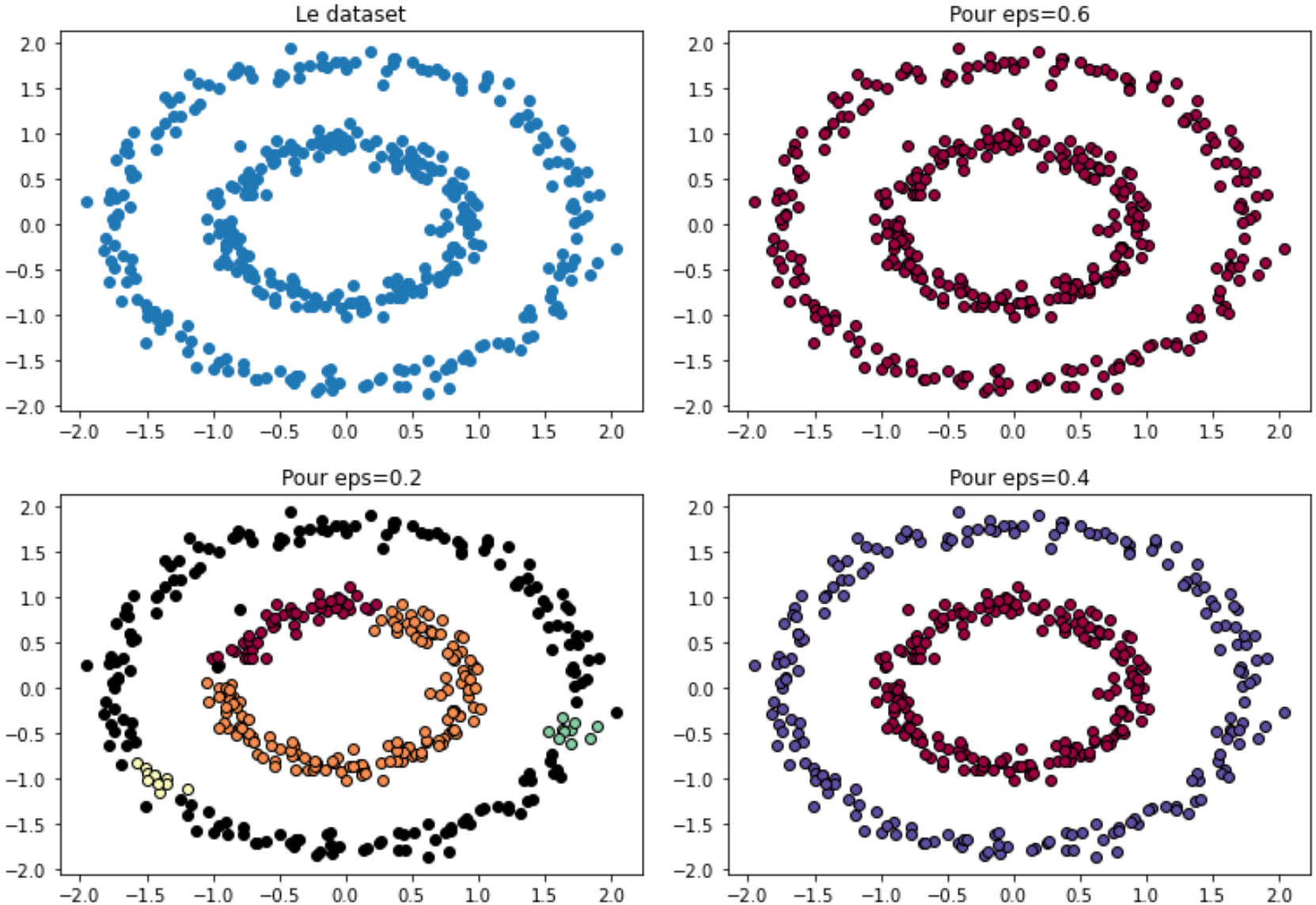
Selection et expansion



Utilisation MCTS

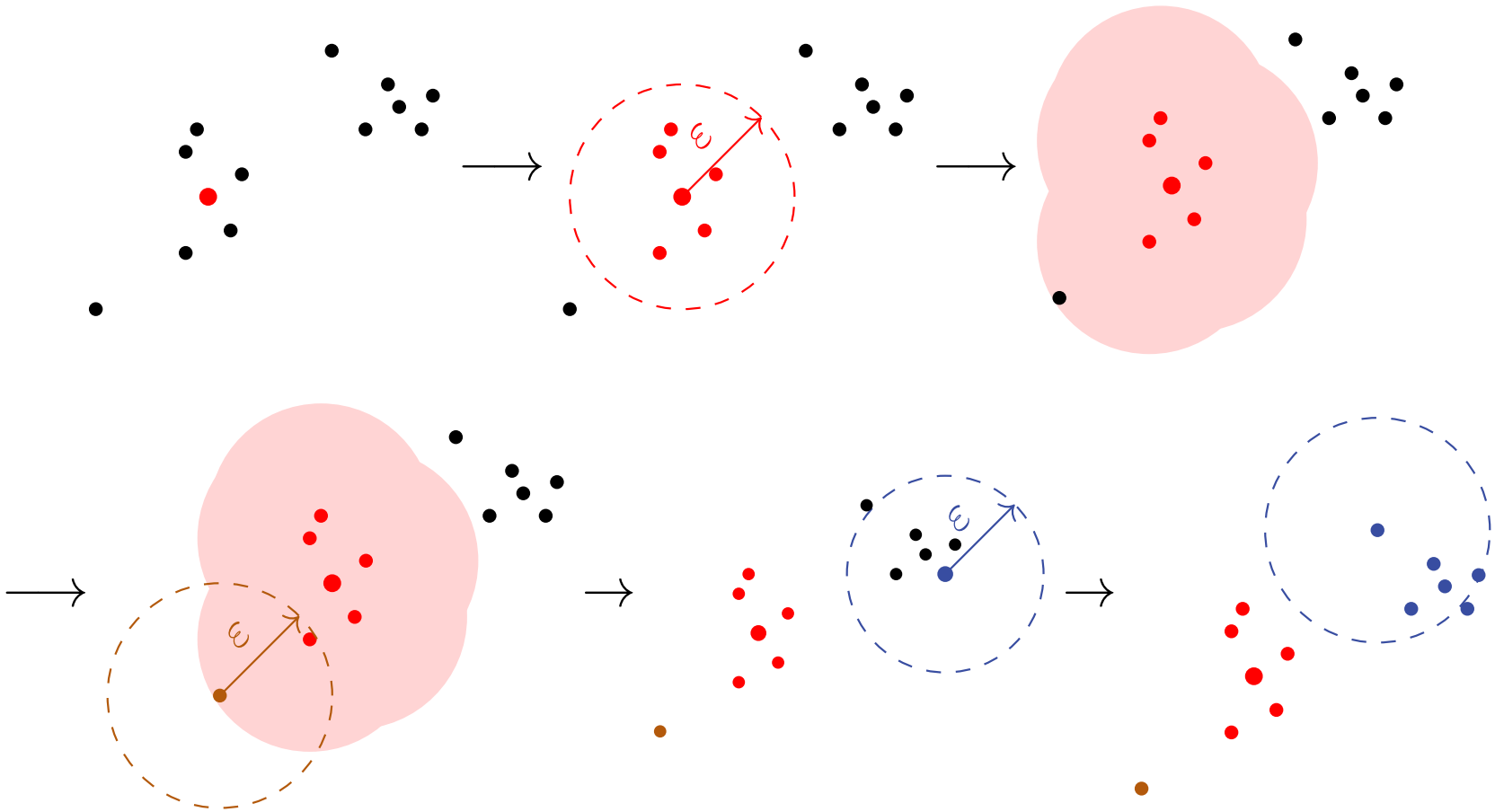


Rassemblement des unitées : DBSCAN



Rassemblement des unitées : DBSCAN

Deux paramètres : ε et *MinPts*
Ici avec $MinPts = 3$:



Points frontière et centraux du cluster 1 et du cluster 2

Application du DBSCAN