



INSTITUT
POLYTECHNIQUE
DE PARIS

PSC Dots and Boxes

BENYAMINE Axel, BORDERIES Titouan, COLLOMB Louis, DELPECH de
SAINT GUILHEM Dimitri, DIBY Wilfried

Sommaire

Dots and boxes

p. 3

Interface graphique

p. 12

Présentation et études
des solveurs

p. 16

Enseignements

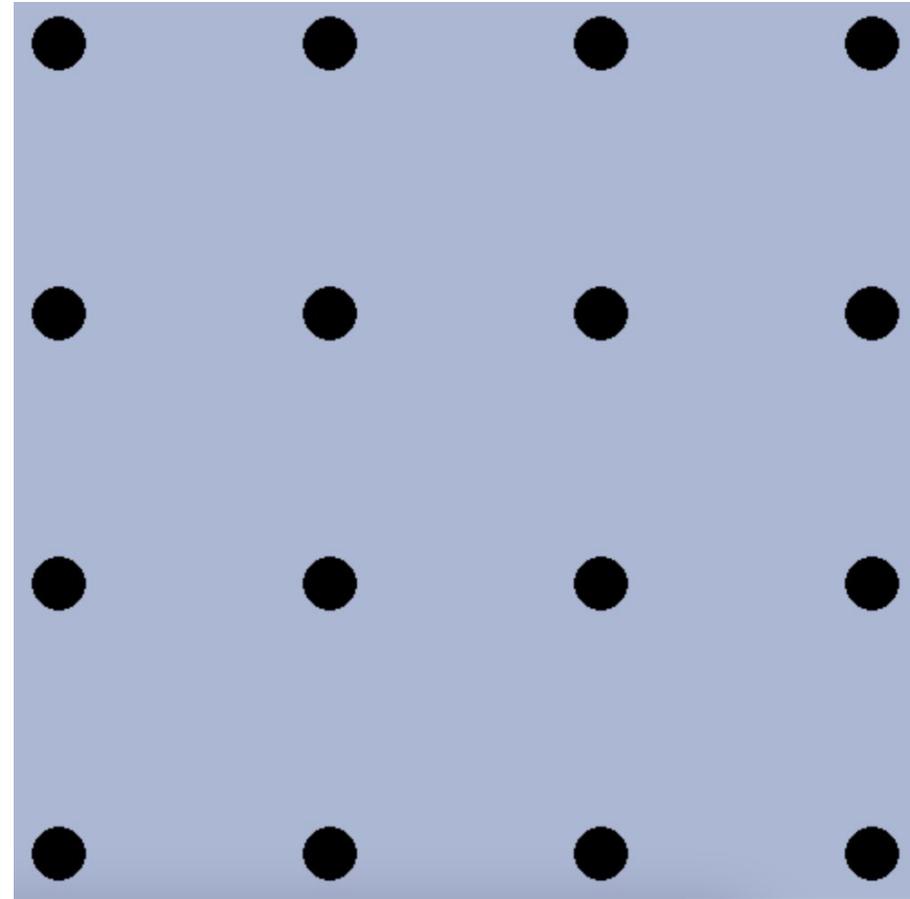
p. 52

Présentation de Dots and Boxes

Présentation de Dots and Boxes



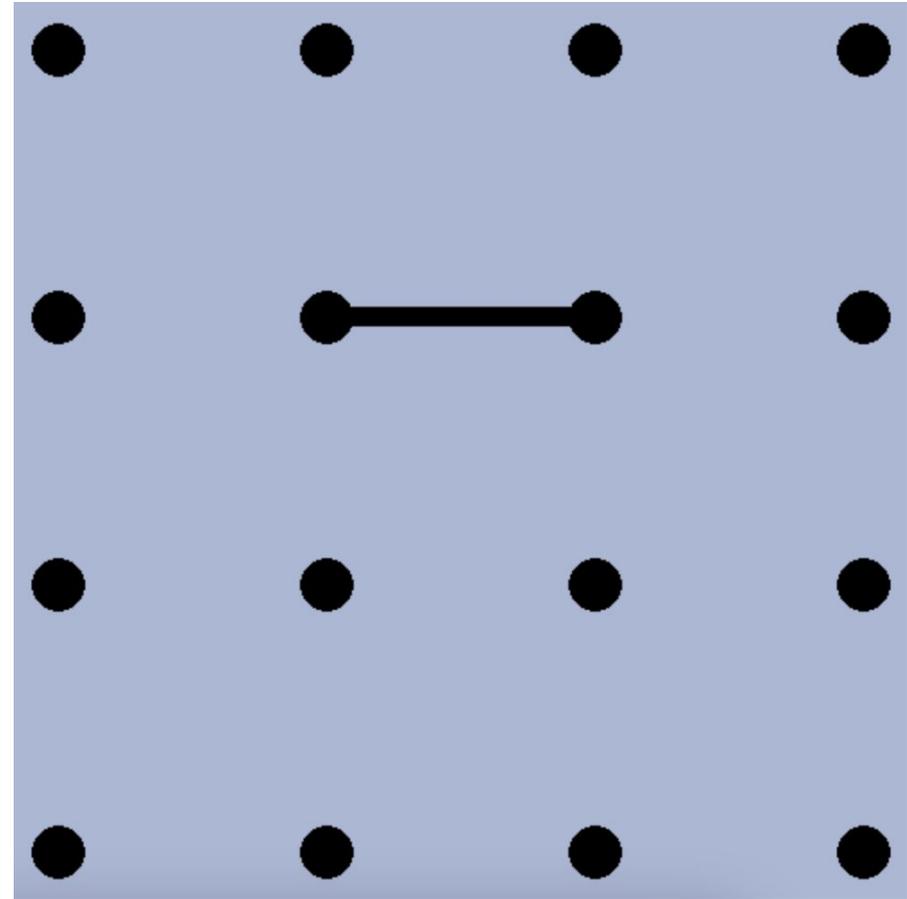
- 2 joueurs
- compléter un maximum de carrés



Présentation de Dots and Boxes



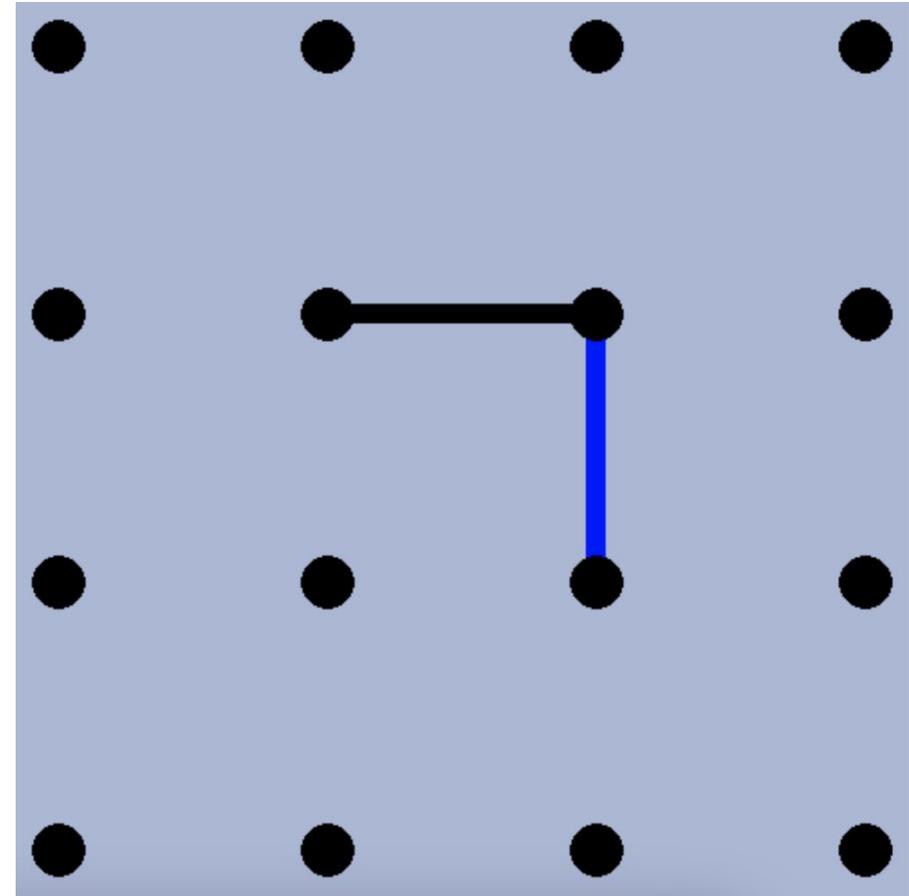
- 2 joueurs
- compléter un maximum de carrés



Présentation de Dots and Boxes

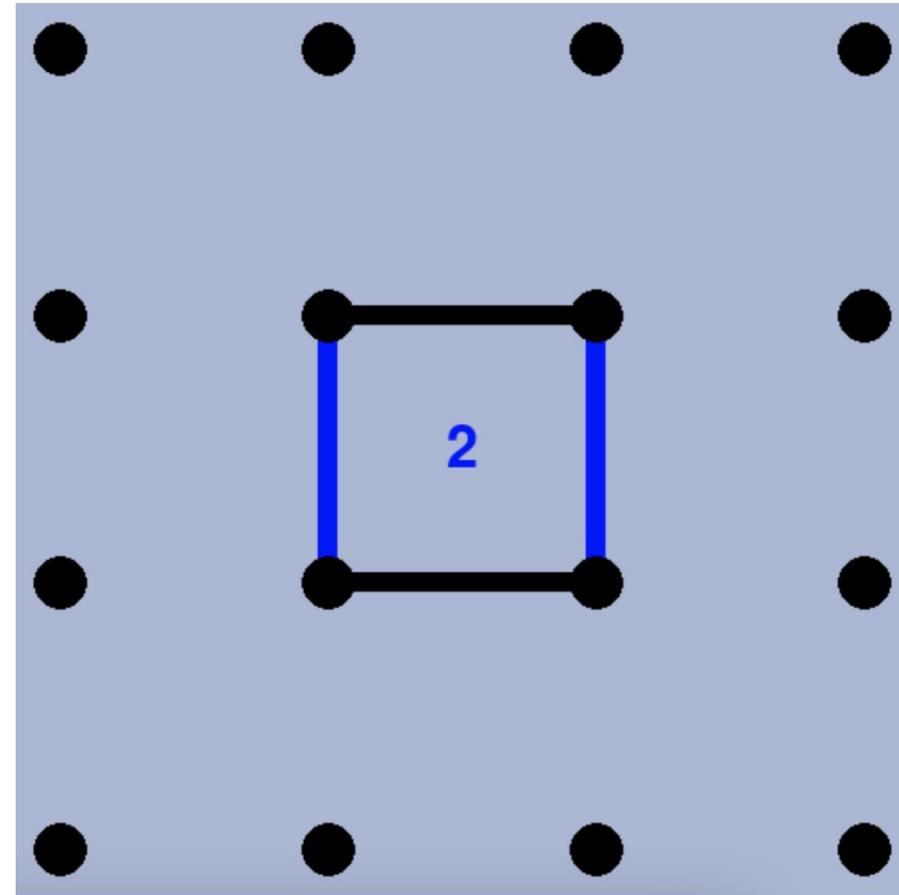


- 2 joueurs
- compléter un maximum de carrés



Présentation de Dots and Boxes

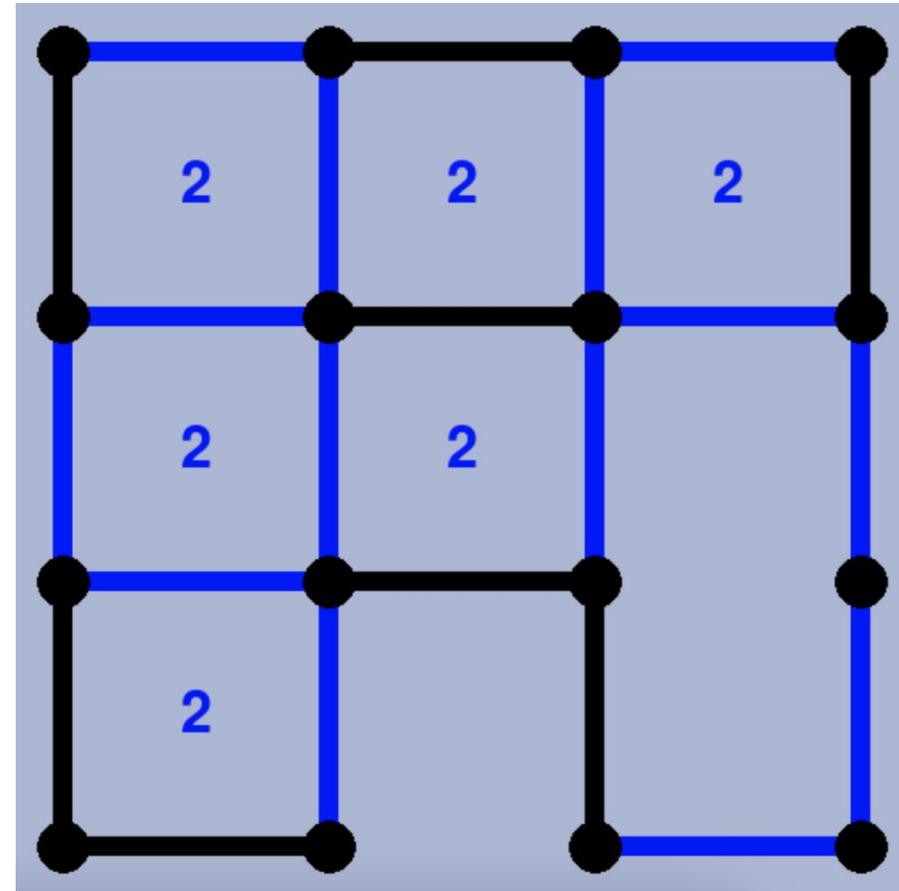
- 2 joueurs
- compléter un maximum de carrés



Présentation de Dots and Boxes



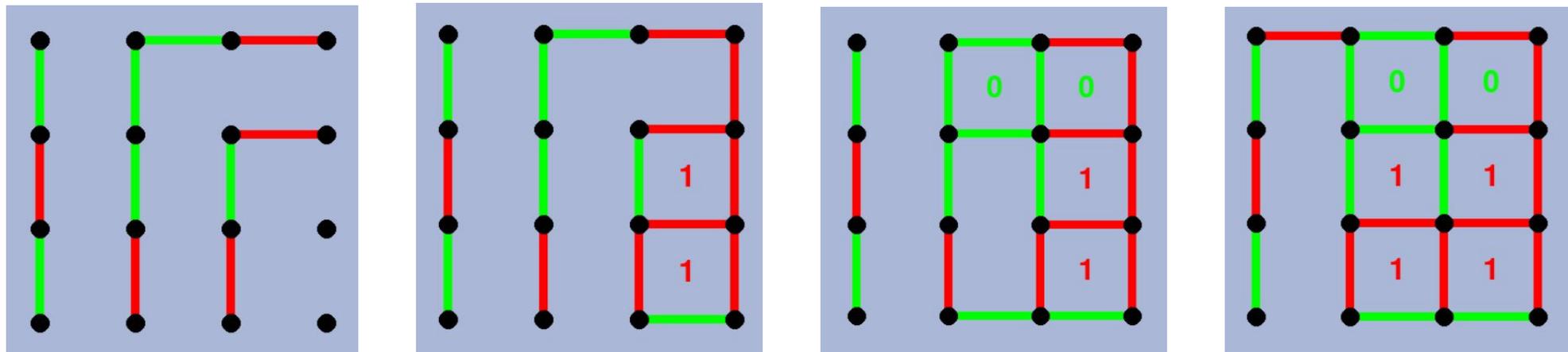
- le double dealing



Présentation de Dots and Boxes



- le double dealing



Implémentation du Jeu

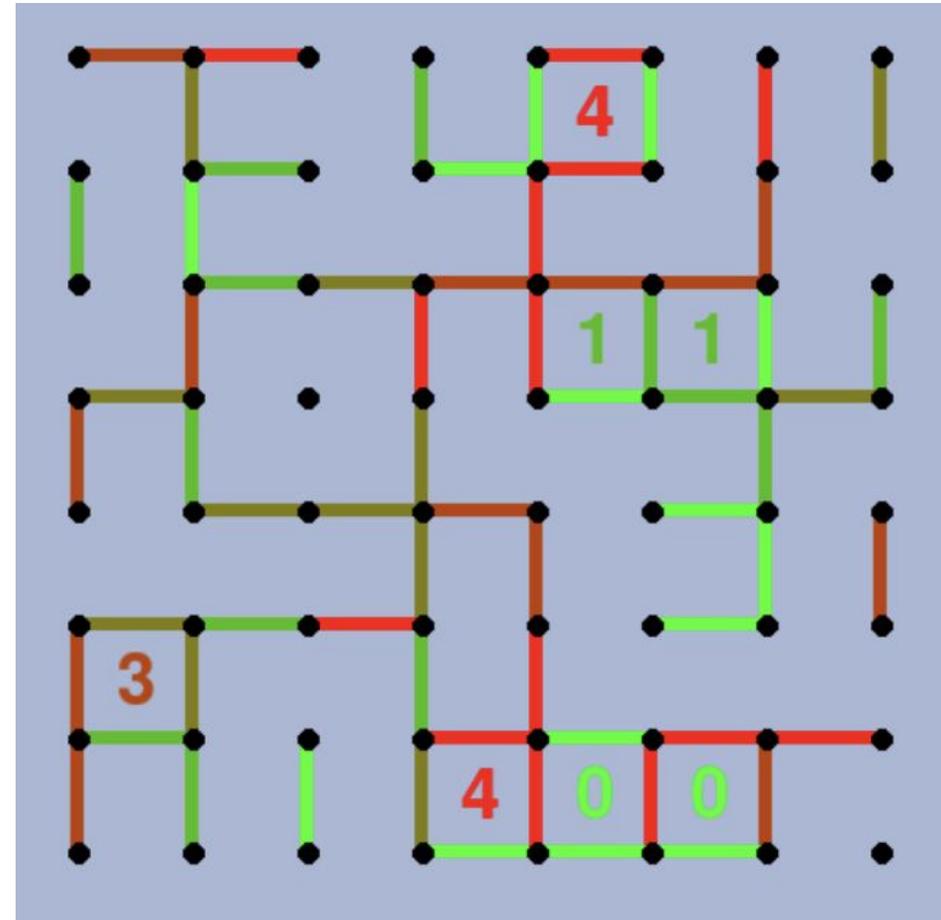
Représentation graphique



- deux matrices pour les barres: verticales et horizontales
- deux variables scores pour les joueurs
- des fonctions de la classe jeu pour avoir la valence, le score...

Variante à plusieurs joueurs

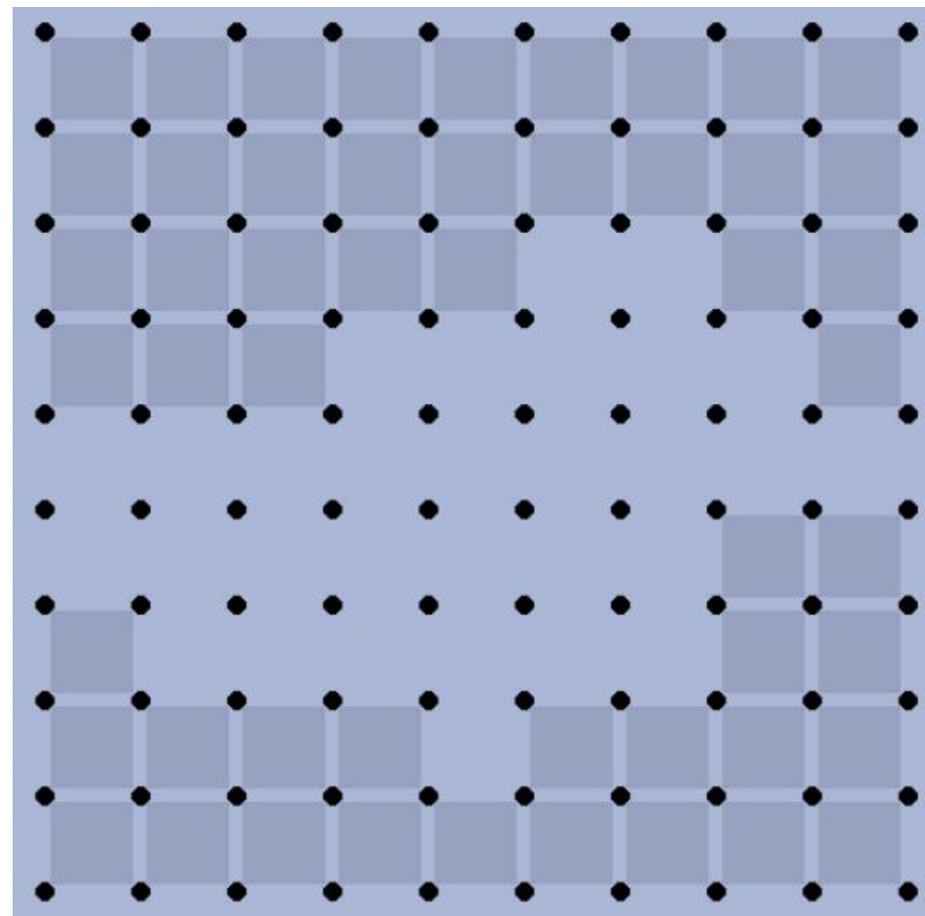
- un nombre de joueurs illimité



Variante à plusieurs joueurs



- des grilles connexes



Les algorithmes

Les algorithmes



- Algorithmes aléatoires (références)
- MiniMax
- MCTS (Monte Carlo Tree Search)
- Score

Algorithmes simplistes



Deux algorithmes élémentaires ont été implémentés :

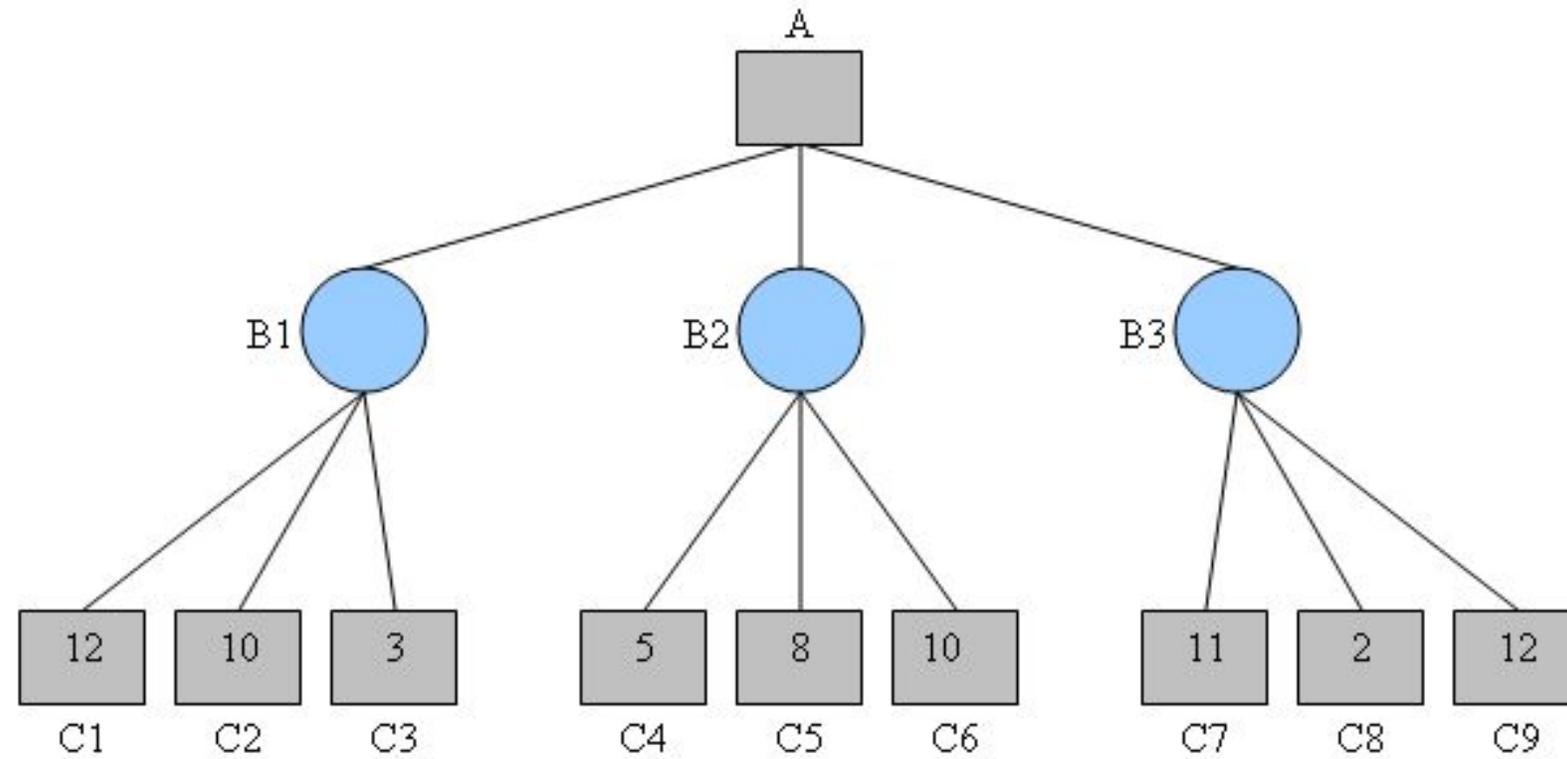
- Random1 : joue comme Random sauf qu'il forme une boîte s'il a l'opportunité
- Random2 : joue comme Random1 sauf qu'il évite les coups permettant de donner directement une boîte à l'adversaire

L'algorithme MiniMax

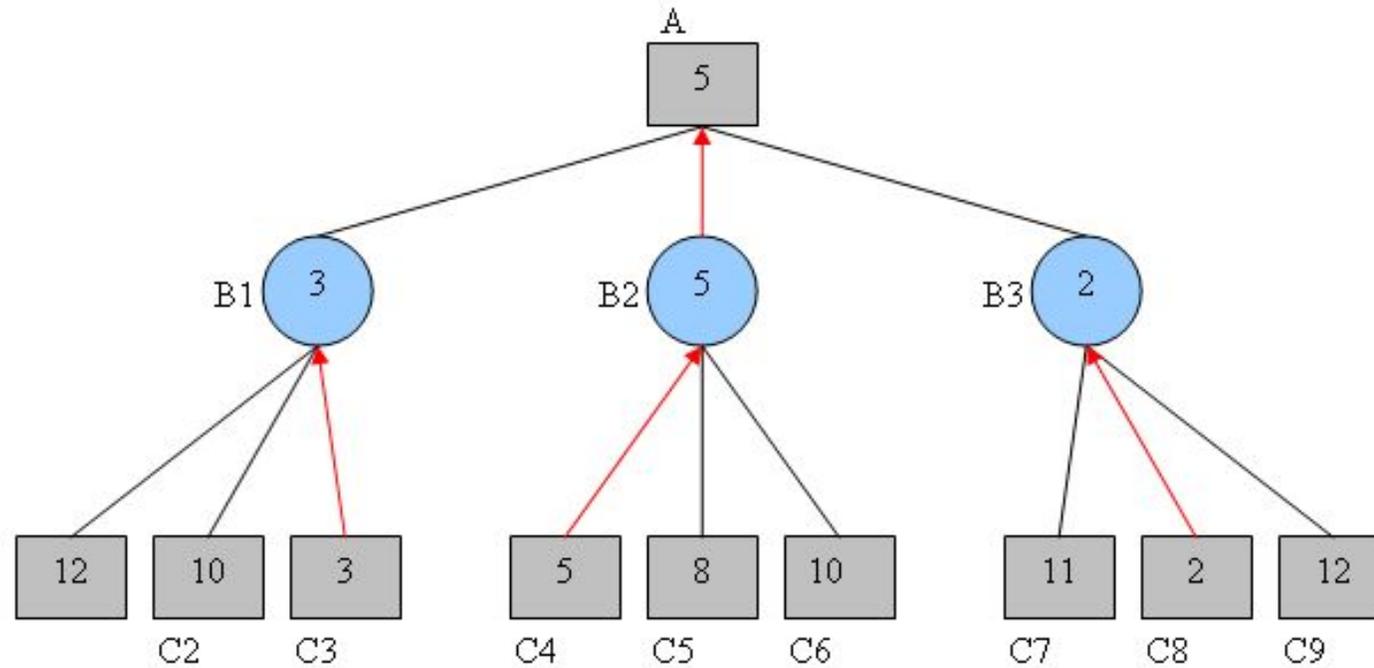


- Utilisé pour des jeux à somme nulle à deux joueurs jouant alternativement
- L'algorithme consiste à chaque état de jeu à minimiser le maximum des gains de l'adversaire
- Développer l'arbre et remonter

Exemple



Exemple



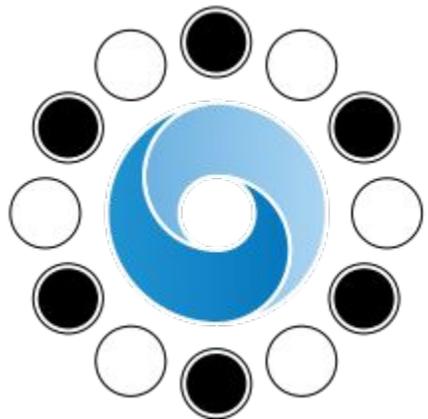
Inconvénients du Minimax



- Pas si simple computationnellement parlant...
- Dans la pratique, on explore une partie de l'arbre des possibilités :
 - profondeur limite
 - alpha-beta pruning
- Stratégie heuristique
- Impact long terme

L'algorithme Monte Carlo Tree Search

- Estime la valeur d'un état du jeu à partir de simulations
- Développe un arbre de jeu statistique de manière itérative et asymétrique



AlphaGo

Approche naïve

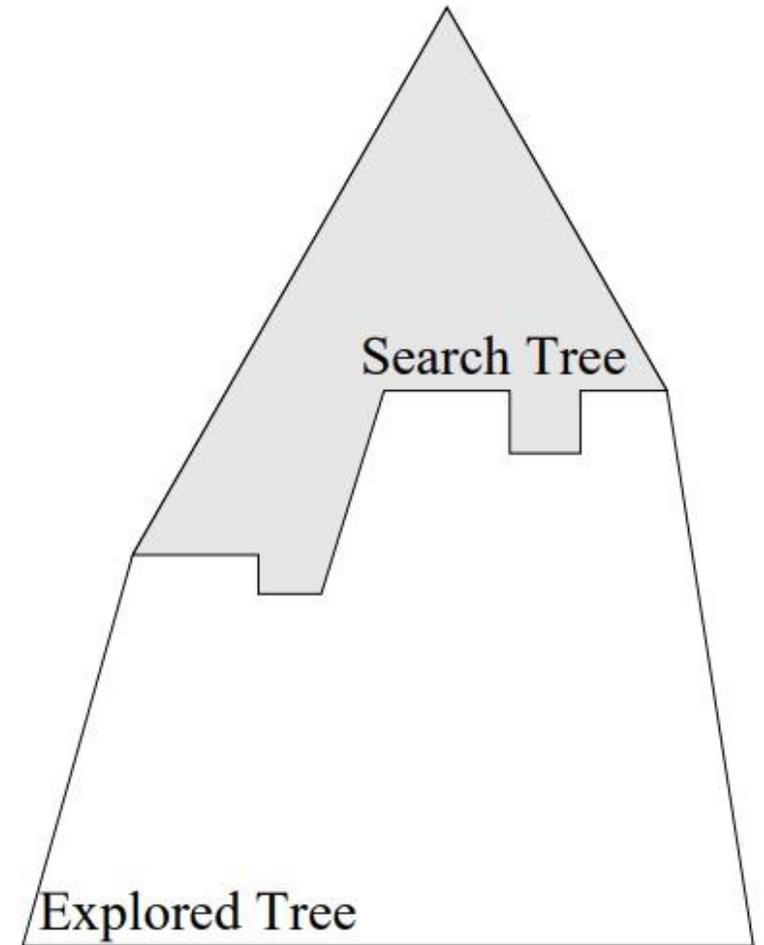


- Utiliser les simulations comme fonctions d'évaluations
- Problèmes: bruit et temps de calcul

Fonctionnement intuitif



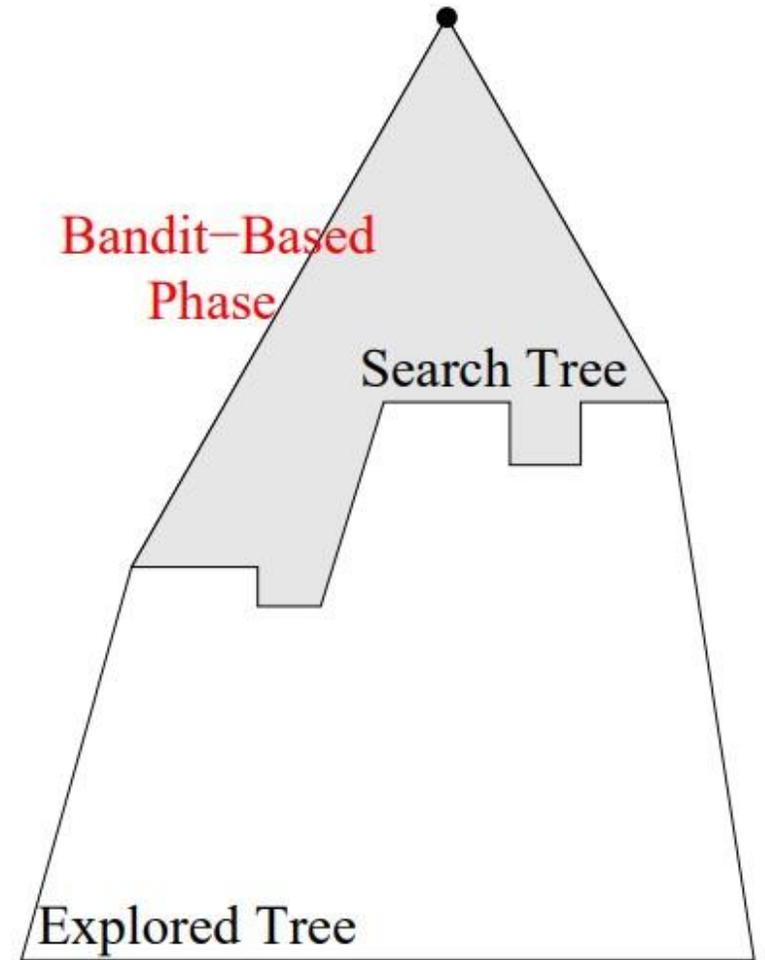
- Deux arbres



Fonctionnement intuitif

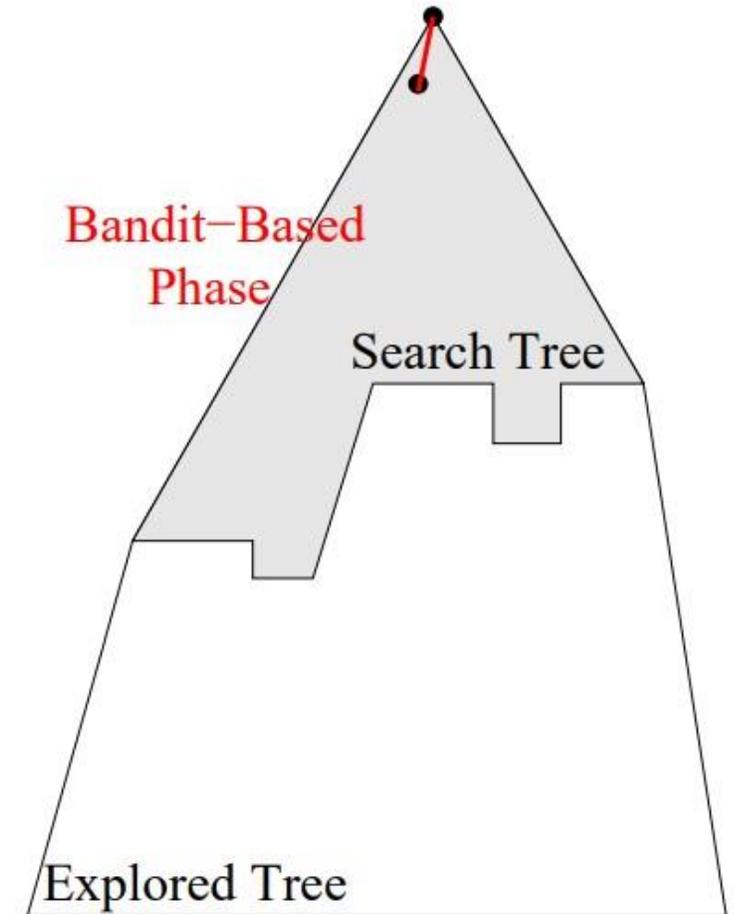


- Deux arbres



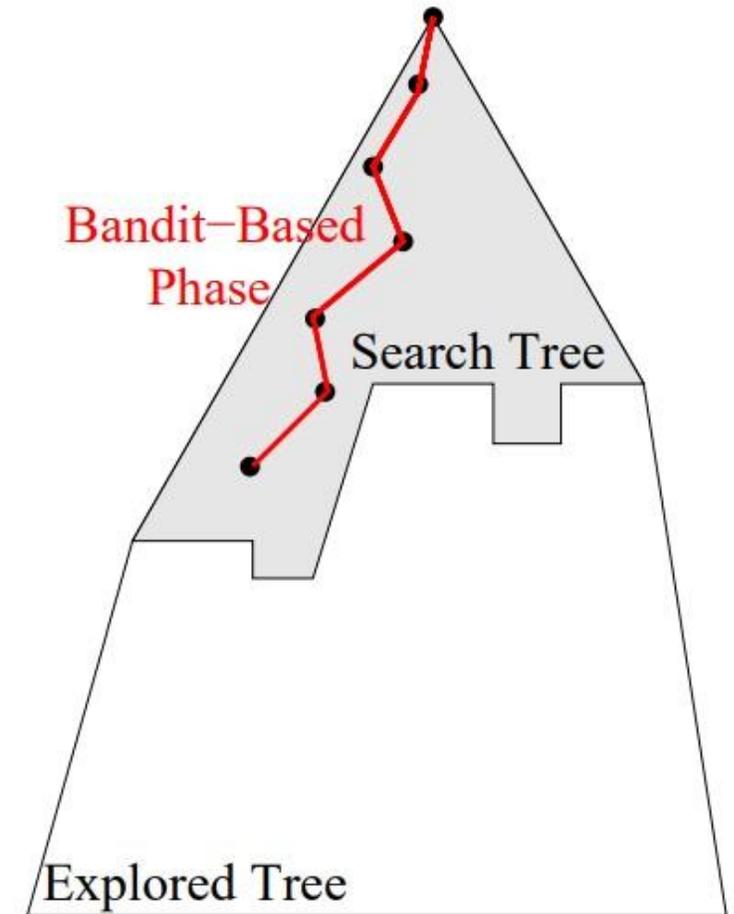
Fonctionnement intuitif

- Deux arbres
- Phase de sélection



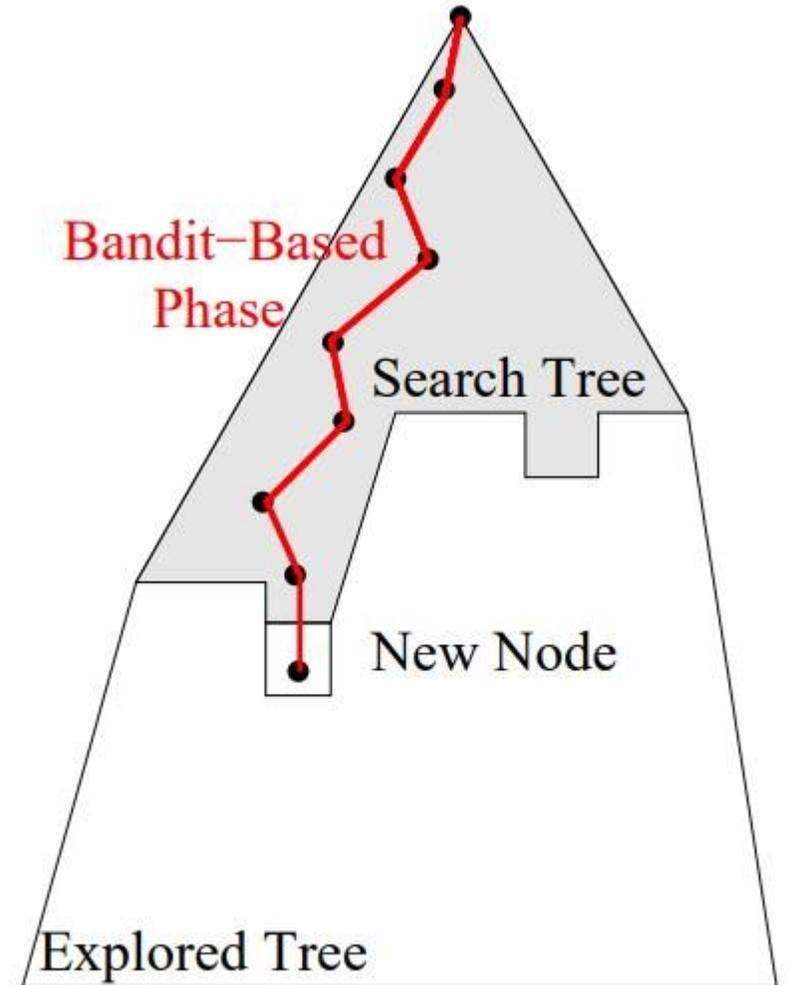
Fonctionnement intuitif

- Deux arbres
- Phase de sélection



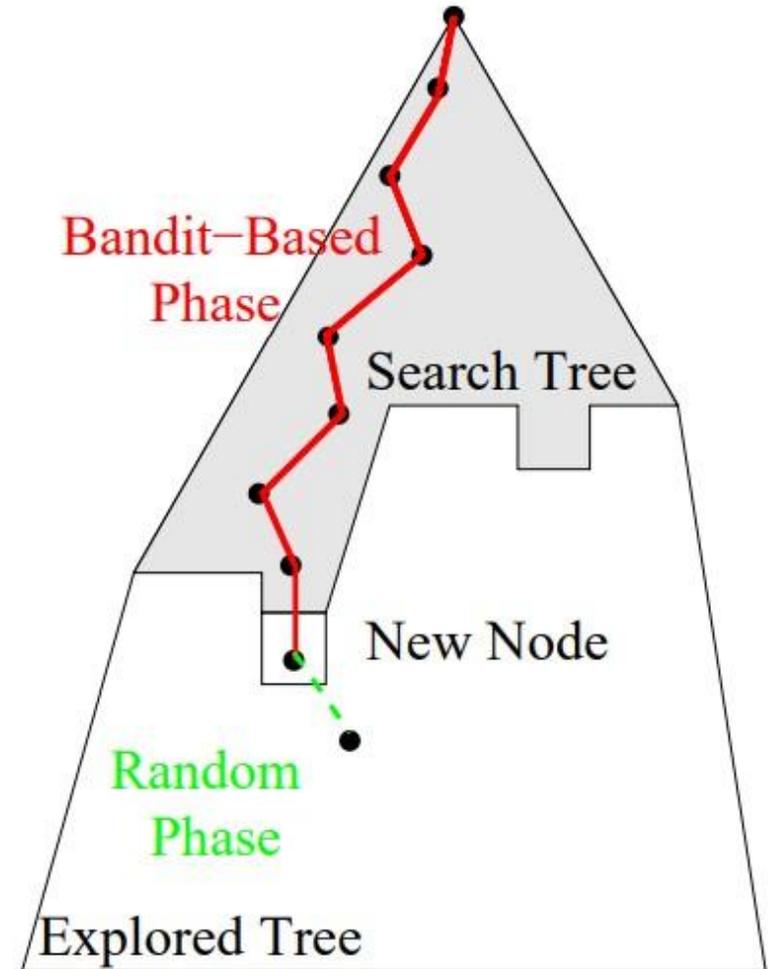
Fonctionnement intuitif

- Deux arbres
- Phase de sélection
- Phase d'expansion



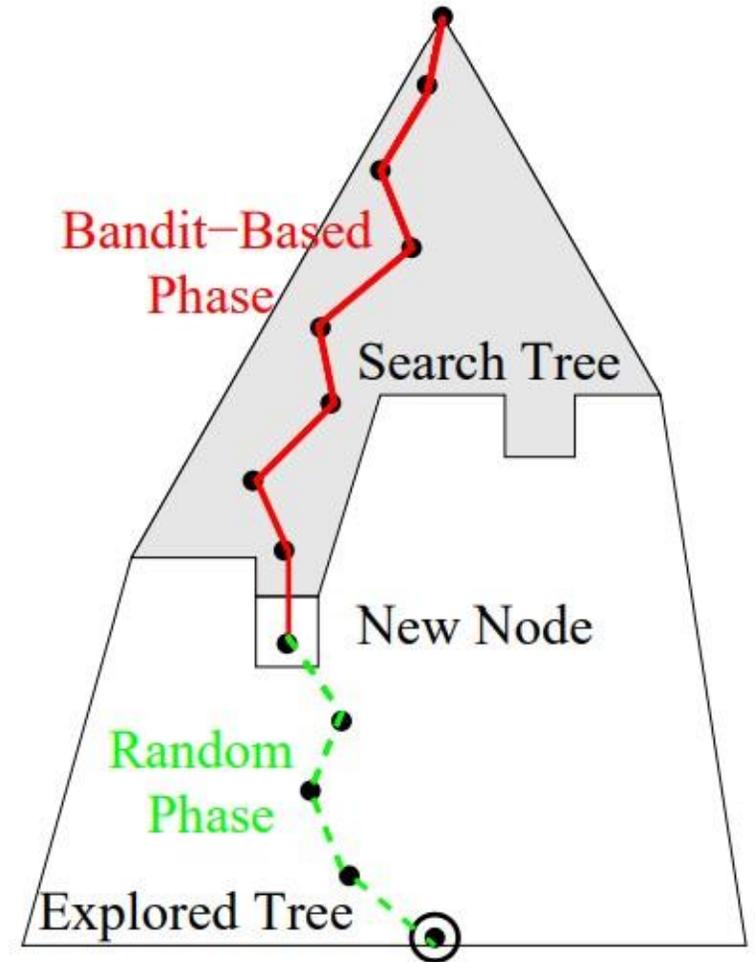
Fonctionnement intuitif

- Deux arbres
- Phase de sélection
- Phase d'expansion
- Phase d'expansion



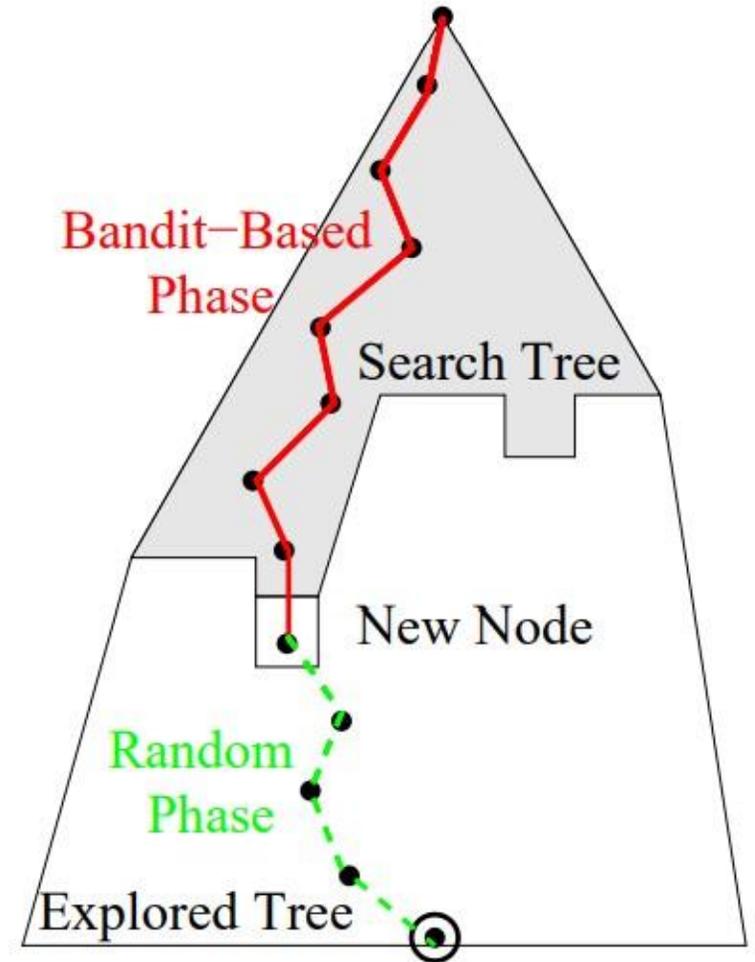
Fonctionnement intuitif

- Deux arbres
- Phase de sélection
- Phase d'expansion
- Phase d'expansion



Fonctionnement intuitif

- Deux arbres
- Phase de sélection
- Phase d'expansion
- Phase d'expansion



Un noeud

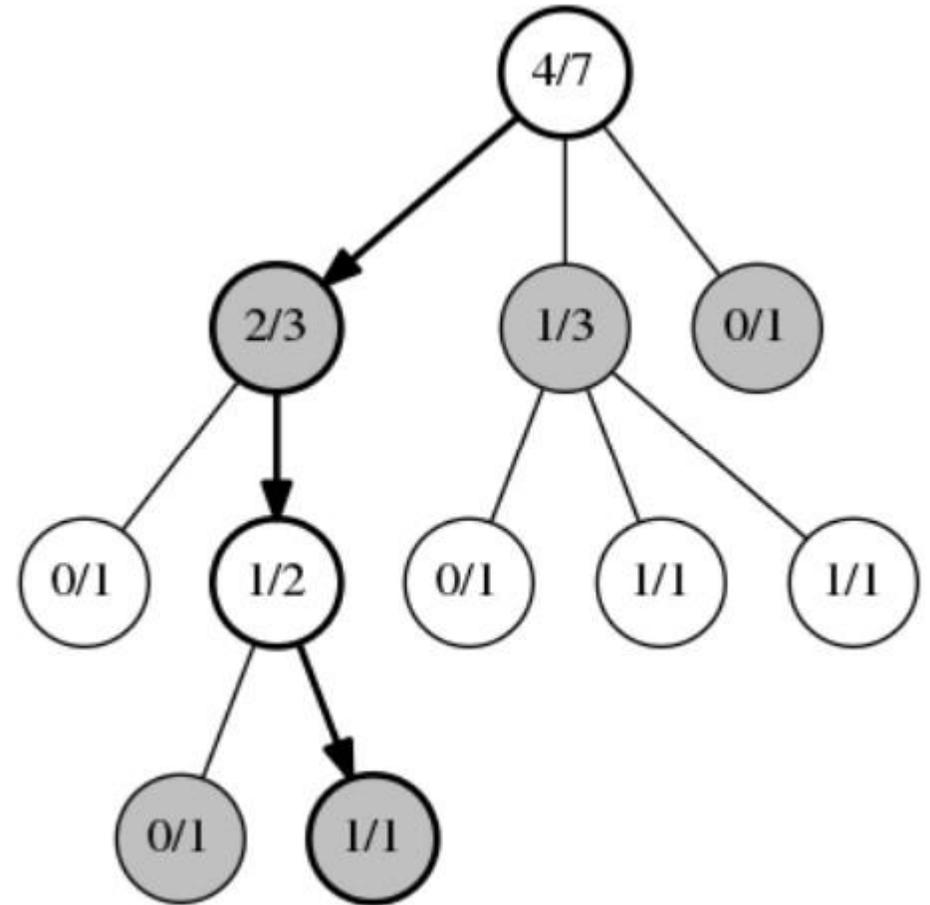


- $s \in S$ un état de jeu
- $a \in A$ une action de jeu

$$W(s) = \frac{Q(s)}{N(s)}$$

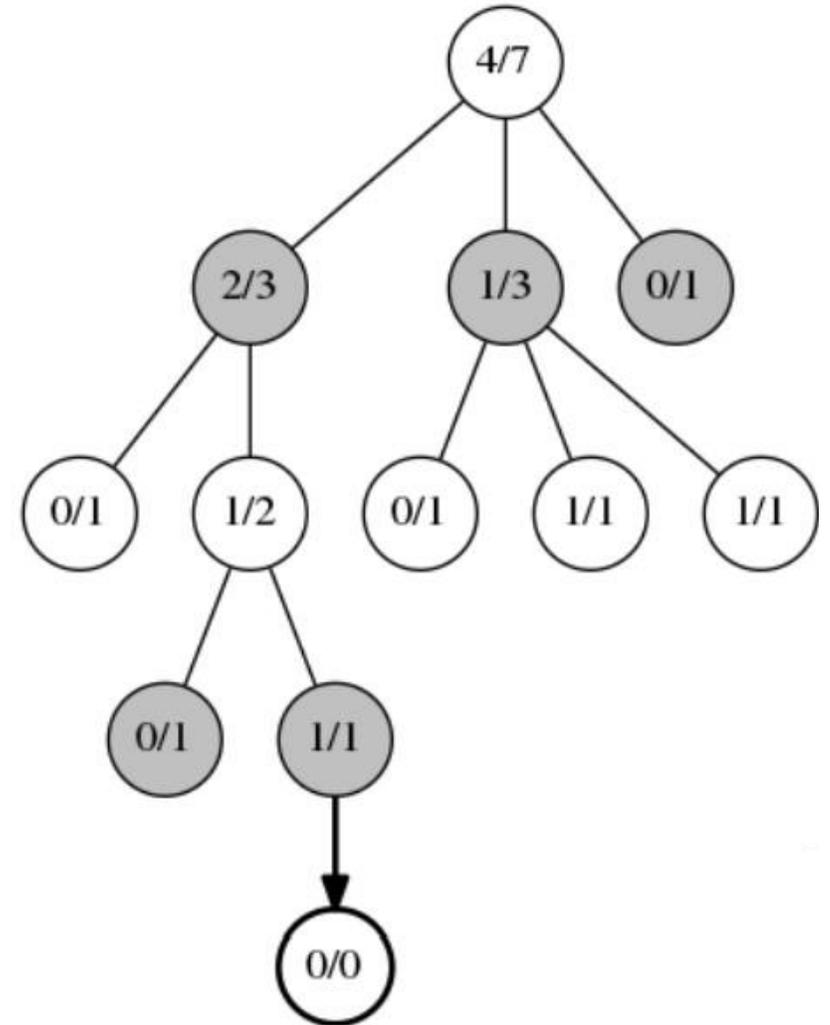
La sélection

- Utilisation de la sélection policy
- Maximise la borne UCB



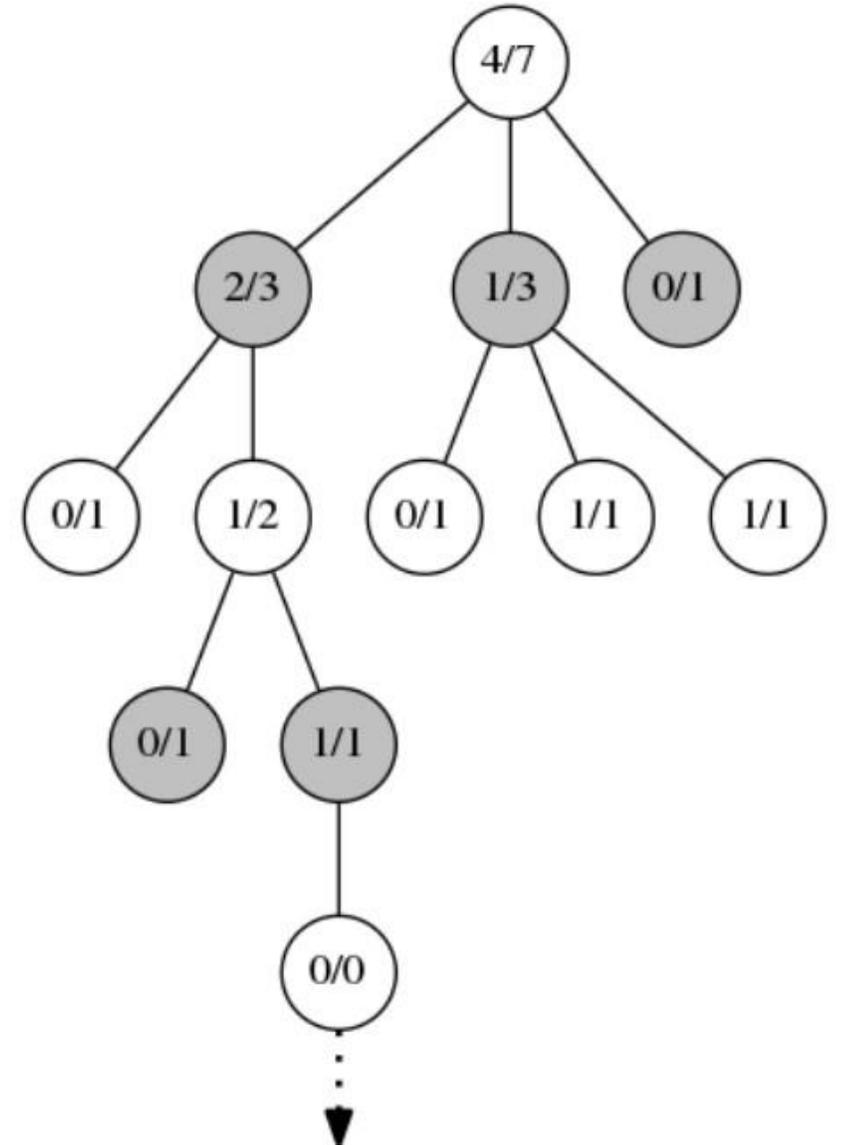
L'expansion

- Création d'un nouveau noeud



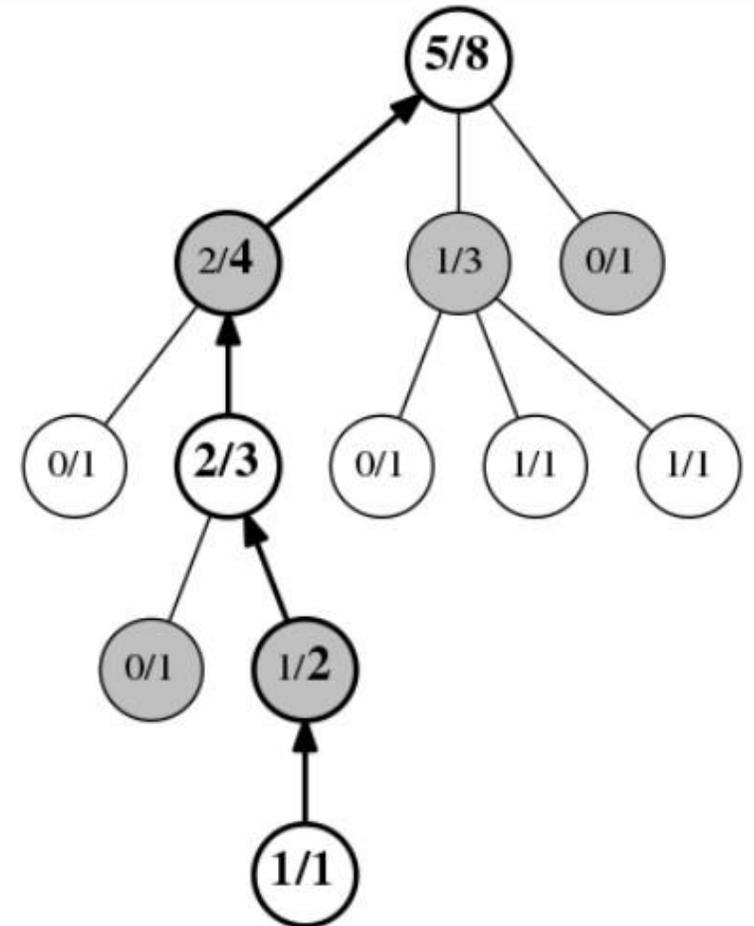
La simulation

- Utilisation d'une autre policy (souvent random policy)



La rétropropagation

- Actualisation, mise à jour
- Apprentissage (similaire au DL)



Le dilemme exploration-exploitation



- Exploitation: concentration sur les actions prometteuses
- Exploration: concentration sur les actions dont l'incertitude de précision est grande

$$\hat{Q}(s, a) + C \sqrt{\frac{\ln(n_s)}{n_{s,a}}}$$

Approche de l'état de l'art : DRL



- AlphaGo et AlphaZero: MCTS couplé à RL couplés à des réseaux de neurones
- Principe du RL : Action-Réaction
- Progression de l'agent en le faisant jouer comme lui même

Approche de l'état de l'art : DRL



- Du deep learning pour définir une meilleure fonction score
- En couplant cela à du reinforcement learning on entre dans le domaine du deep reinforcement learning.

Un algorithme basé sur une
fonction de score

Un solveur basé sur une fonction score



Objectif : implémenter un solveur qui prend en considération des résultats particuliers de Dots and Boxes.

Choix du théorème mathématique :

Ouvrage de référence : Elwyn Berlekamp, *The dots and Boxes game*

Règle des *long chains* :

Le joueur A (respectivement joueur B) a intérêt à ce que la somme du nombre de *long chains* et du nombre de points soit paire (respectivement impaire).

Un algorithme de comptage de long chains



Entrée : une grille donnée

Sortie : nombre de *long chains* et longueur de chacune d'elle

fonction chaine(grille):

Initialisation : chaine une matrice de zéros de même taille que la grille

Pour tous les carrés :

 prochain_chaine(carré)

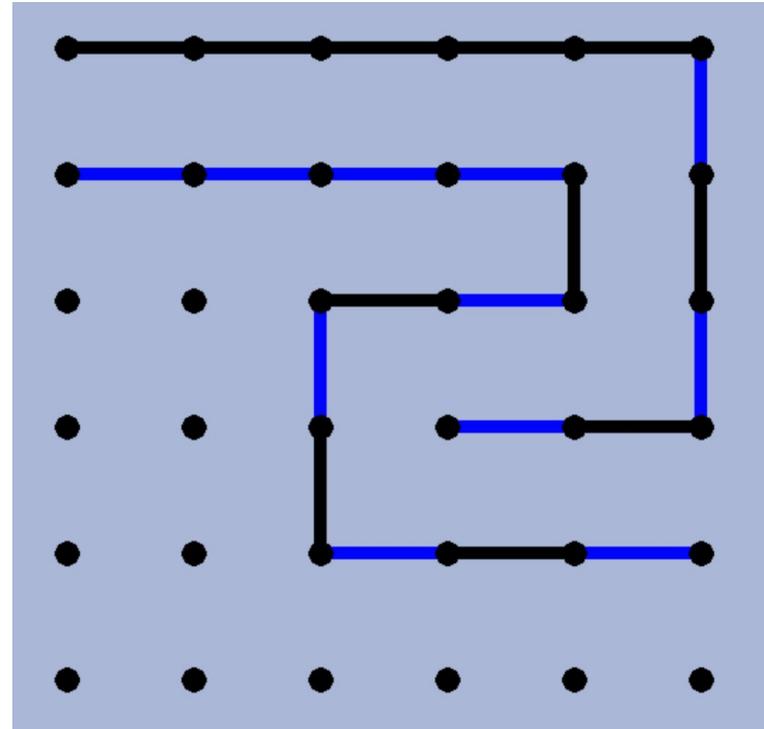
On compte les long chains en "suivant" les directions de chaine

Un algorithme de comptage de long chains

Idée de `prochain_carré(carré, direction_origine=null)`:

1. Prendre un voisin non étudié jusqu'alors et noter la direction empruntée.
2. Appeler `prochain_chaine(voisin, direction_opposée)`

```
[[4., 4., 4., 4., 2.],  
 [0., 0., 4., 3., 2.],  
 [0., 0., 2., 3., 3.],  
 [0., 0., 4., 4., 3.],  
 [0., 0., 0., 0., 0.]], [(0, 0)], [12])
```



directions : 1, 2, 3, 4 = haut, bas, gauche, droite

Une fonction de score basée sur les long chains



Entrée : grille, score courant de chaque joueur

Sortie : $\text{score_joueurA} \in [-1, 1]$ ($\text{score_joueurB} = - \text{score_joueurA}$)

$\text{score} = ax + b$ où:

- b est le nombre de carré fermés par A - nombre de carrés fermés par B
- a est positif ou négatif selon la règle des long chains
- $|a| \Leftrightarrow$ importance relative *long chains* versus carrés déjà fermés.
- x est la longueur moyenne des *long chains*.

La policy adaptée (comment choisir l'arête suivante)



Score_policy :

Pour toutes les arêtes non prises :

Prendre temporairement l'arête (maj des scores courants si besoin)

Stockage du score de la nouvelle grille pour le joueur courant

Enlever l'arête

Arête retenue : $\text{argmin}(\text{max des scores})$

Variante inspirée de Random 2

Si une arête permet de prendre un carré :

Prendre l'arête

Sinon :

Stocker toutes les arêtes qui ne cèdent aucun carré à l'adversaire

Parmi ces arêtes :

Prendre l'arête qui mène au meilleur score

Résultats contre Minimax :

- $n,p = 5 \rightarrow 70\%$ de victoires
- $n,p = 7 \rightarrow 75\%$ de victoires
- $n,p = 10 \rightarrow 40\%$ de victoires

Etude des algorithmes

Caractéristiques



Randoms : naïveté et erreurs fréquentes.

Score v1 : choix contre-intuitifs source d'erreurs.

Score v2 : contrôle mais erreurs coûteuses.

Minimax : efficacité

AlphaZero : reconnaissance des situations.

Enseignements

Notre organisation



Un travail assez divisé :

- interface de test et algorithmes
- compatibilité du code

Un manque de structure dans le code :

- manque d'anticipation des ajouts.
- premier projet informatique de telle ampleur

Démarche scientifique



Une démarche initialement théorique :

- études des algorithmes
- la règle de la long chain

Le passage à l'implémentation donne la priorité à l'expérimentation :

- obtention des résultats
- interprétation et explication
- affinement des paramètres
- nouvelles idées



Merci

