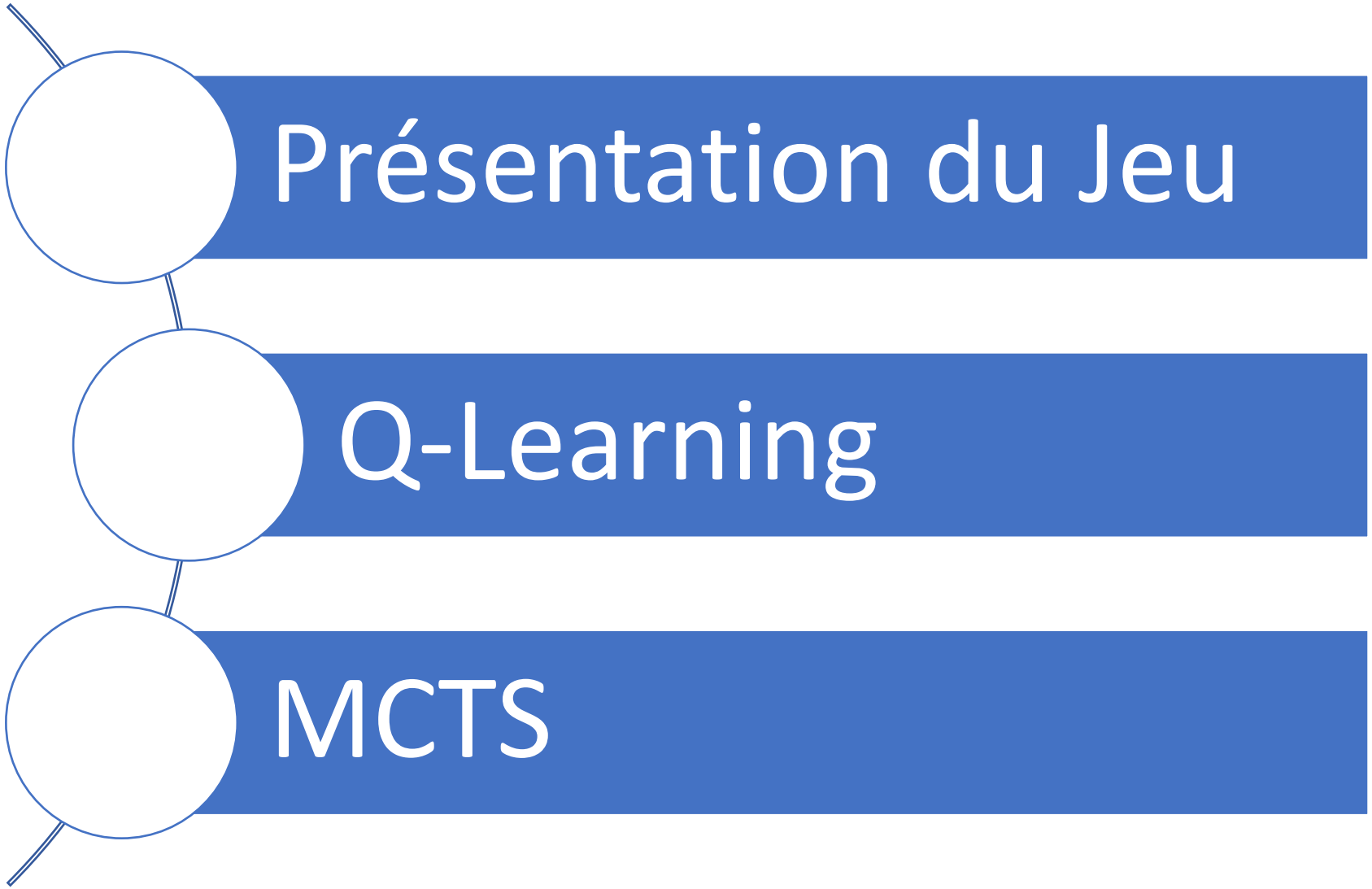
Renforcement Learning sur Super Mario Bros

HADDOU Amine

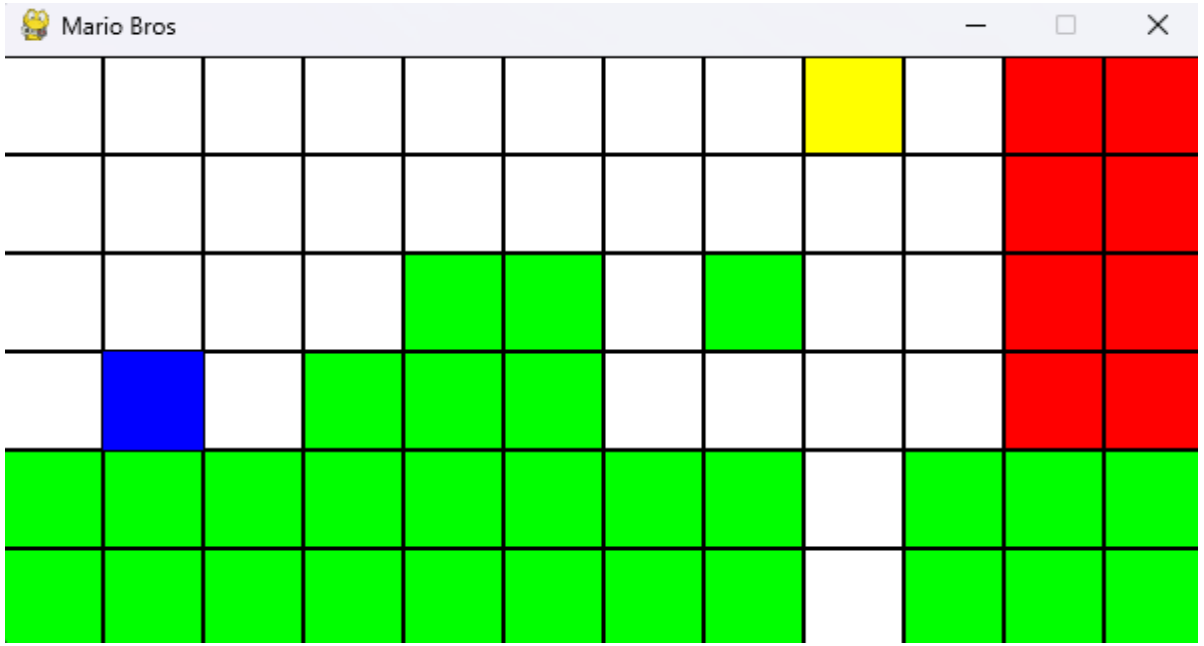
DE SEROUX Colin

Plan



Super Mario Bross

- Un jeu de parcours de type “Grid worlds” (tabulaire)
- Les types d’**obstacles** :
  - Trou
  - Bloc barrant le passage
- Deux **actions** sont possibles :
  - Avancer d’un bloc
  - Sauter de deux blocs en avant
- Score :
  - Les pièces (blocs jaunes) : +50 pnts
  - Terminer le parcours : +100 pnts

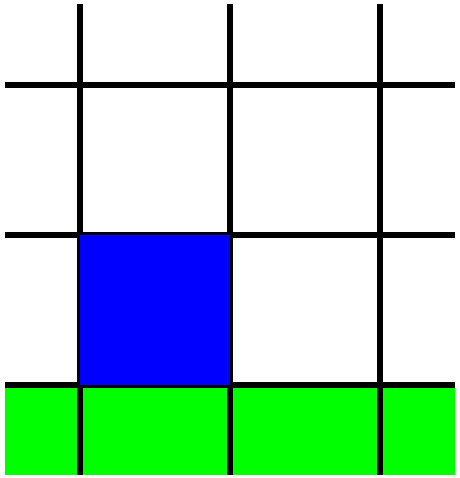


Exemple d’un parcours. Le joueur est modélisé grâce à un bloc bleu.

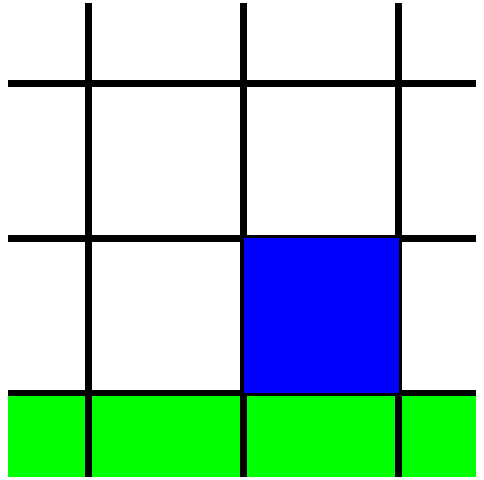
Actions

1) Avancer

Sans Bloc



avant

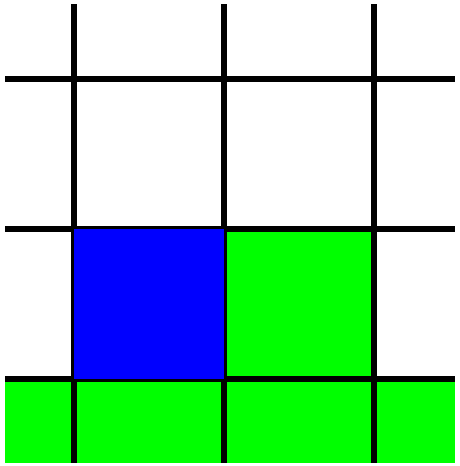


après

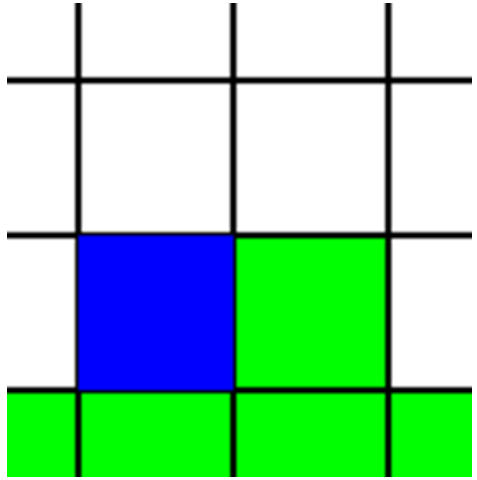
Actions

1) Avancer

Avec Bloc



avant

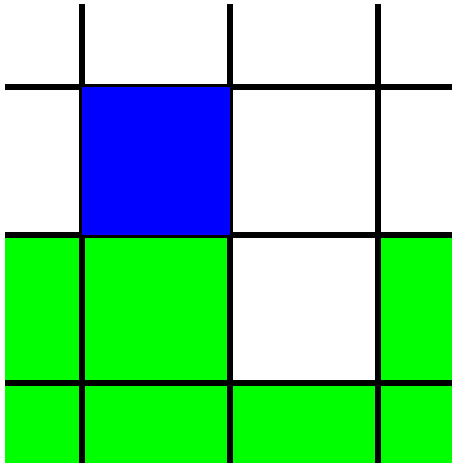


après

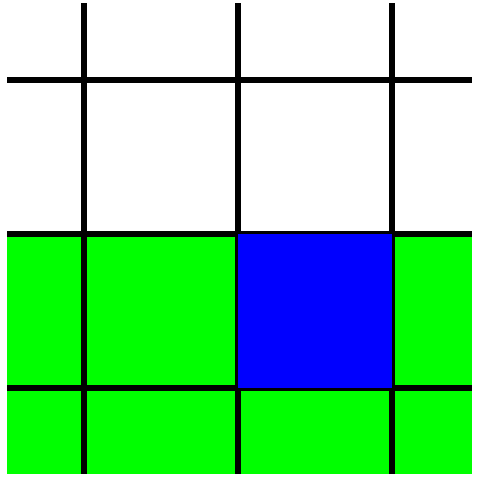
Actions

1) Avancer

Avec Trou



avant

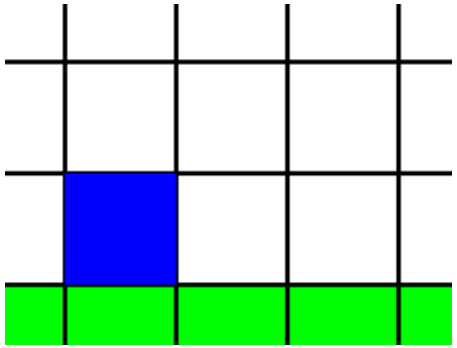


après

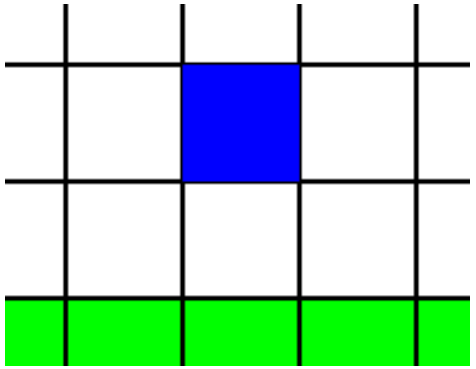
Actions

2) Sauter

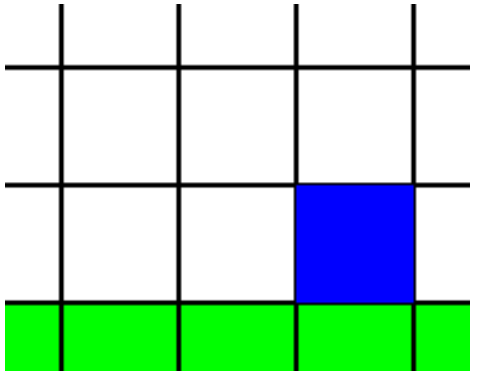
Sans Bloc



avant



pendant

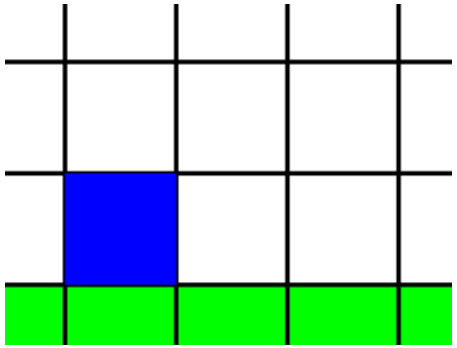


après

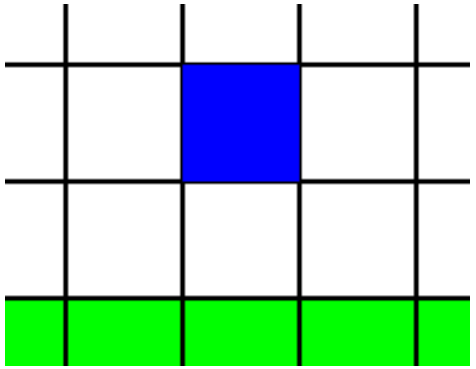
Actions

2) Sauter

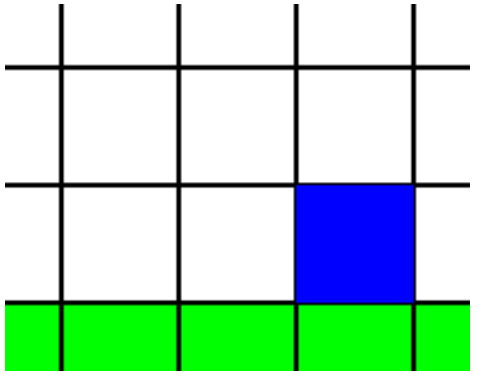
Avec Trou



avant



pendant

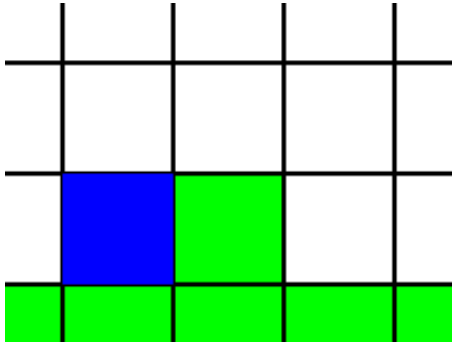


après

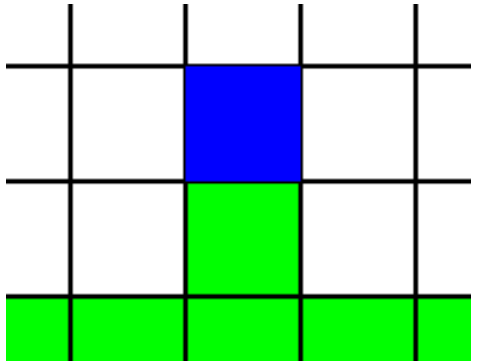
Actions

2) Sauter

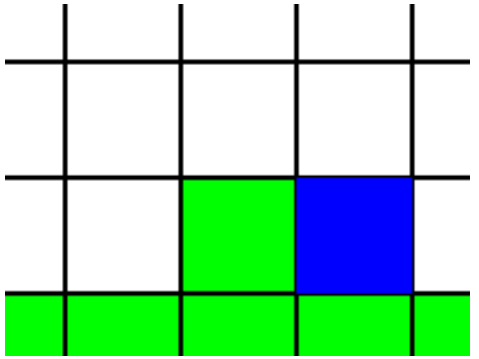
Avec Bloc Bas



avant



pendant

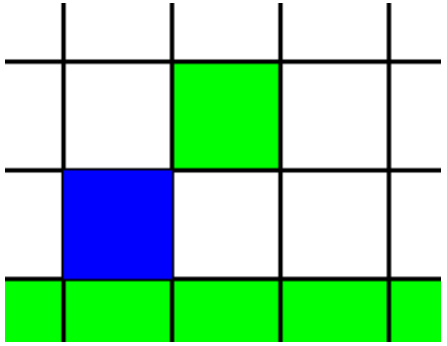


après

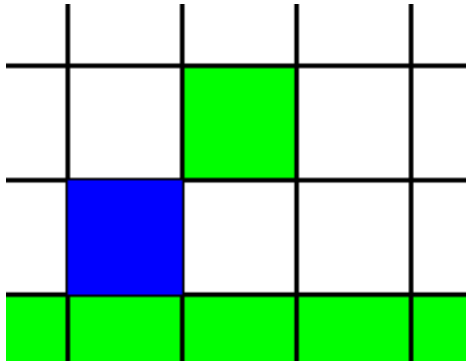
Actions

2) Sauter

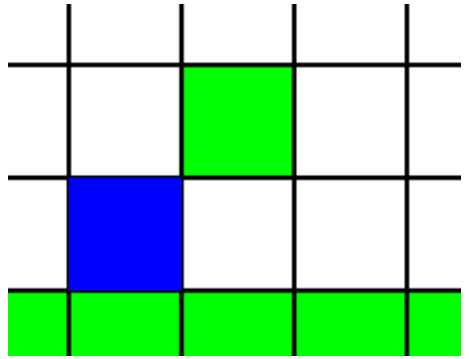
Avec Bloc Haut



avant



pendant

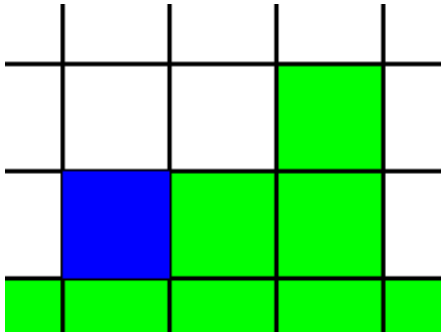


après

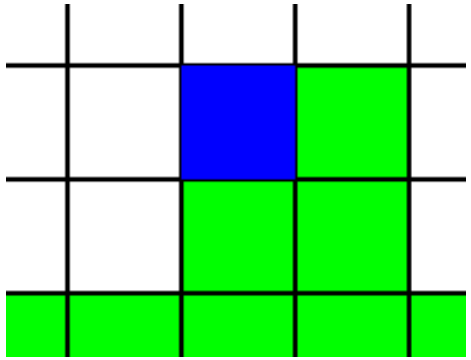
Actions

2) Sauter

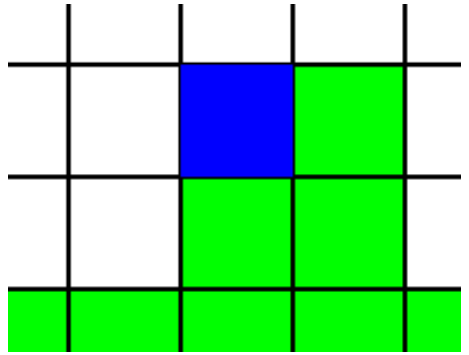
Avec Escaliers



avant



pendant



après

Q-Learning Algorithm

- Un algorithme de type « Temporal Difference »
- Un algorithme dit « Off Policy »
  - Une politique pour décider du prochain mouvement
  - Une politique pour l'apprentissage.
- Qtable pour le jeu :
  - 2 actions
  - 53 états (nb colonnes)

	Action 1	Action 2
Etat 1	$a_{1,1}$	$a_{1,2}$
…	…	…
Etat 53	$a_{53,1}$	$a_{53,2}$

Exemple Q-Table

E-Greedy Exploitation

```
def choose_action(self):
    if random.uniform(0, 1) < self.exploration_rate:
        return self.game.get_random_action()

    state = self.state

    return np.argmax(self.q_table[state])
```

La politique de mouvement

Fonction d'apprentissage

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{\{a'\}} Q(s',a') - Q(s,a))$$

Q(s, a) : la valeur Q actuelle pour l'état *s* et l'action *a.*

α : le taux d'apprentissage.

r : la récompense obtenue après avoir exécuté l'action *a* dans l'état *s*.

γ : le facteur discount (importance des récompenses).

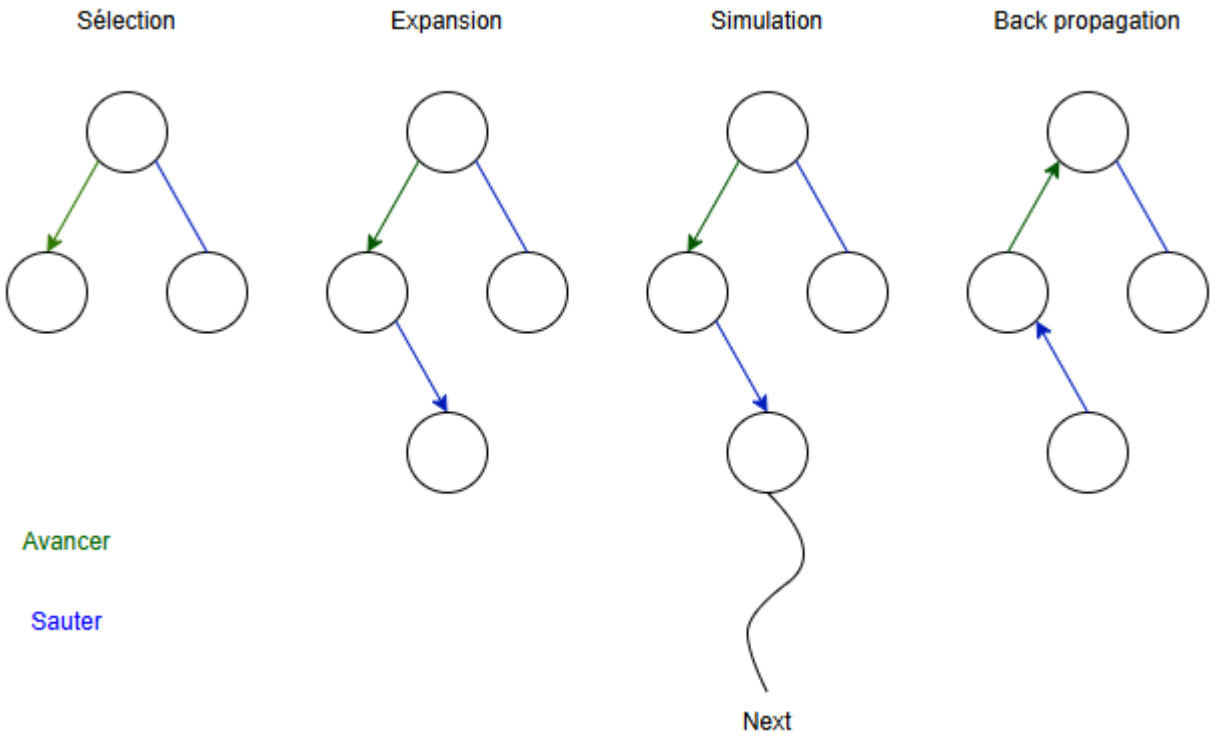
$\max_{\{a'\}} Q(s',a')$: la valeur **maximale** de l'état suivant *s'* en considérant toutes les actions *a'.*

MCTS

- Algorithme d’exploration et d’optimisation utilisé pour les jeux et problèmes de décision complexe
- Combine la recherche aléatoire et l’optimisation pour explorer efficacement les espaces d’actions vastes
- Construit un arbre de décision basé sur des simulations pour déterminer les meilleurs actions en fonction des résultats passés
- Utilisation : Sélection, Expansion, Simulation, Back propagation

Application de MCTS dans le Jeu

- Objectif : faire de la prise de décision pour le joueur, qui peut avancer ou sauter sur une grille.
- Processus
- Arbre de décisions
- Bénéfices



Upper Confidence bounds for Trees

$$\mathrm{UTC}(s_i) = \frac{w_i}{n_i} + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

s_i : un nœud ou coup donné dans l'arbre MCTS.

w_i: le score total de victoires pour le nœud s_i.

n_i : le nombre de fois que le nœud s_i a été visité.

N: le nombre total de visites du nœud parent de s_i.

C : le paramètre d'exploration qui équilibre l'exploration et l'exploitation.

Terme d'exploitation : représente le taux de victoires moyen pour un nœud.

Terme d'exploration : encourage l'algorithme à explorer les nœuds moins visités en augmentant le score UTC pour les coups peu explorés.

Optimisation

- Amélioration de l'exploration

$$reward = score + \frac{(player\_pos[0] - star\_pos[0])}{100}$$

CONCLUSION