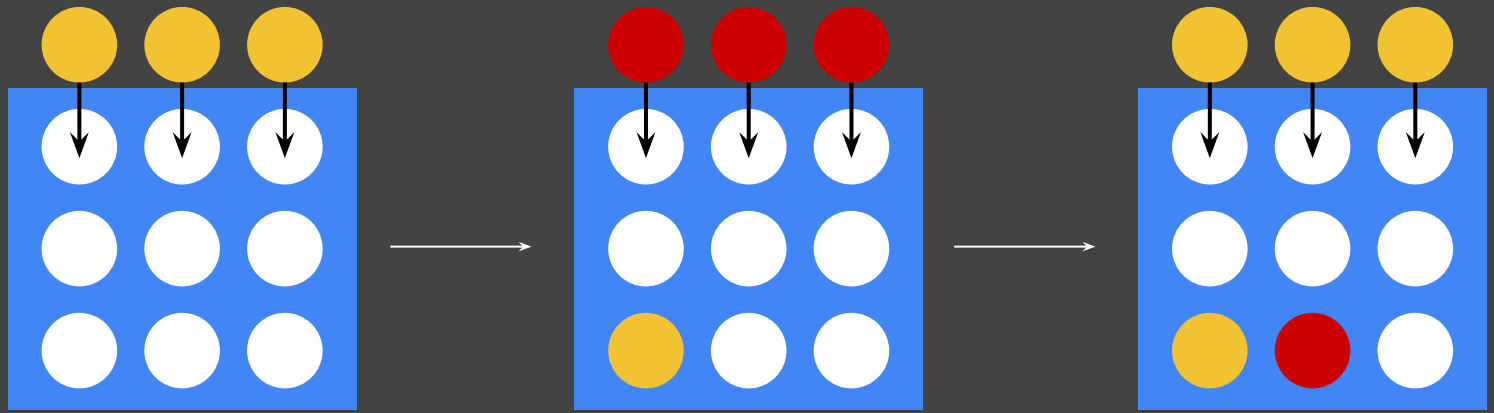
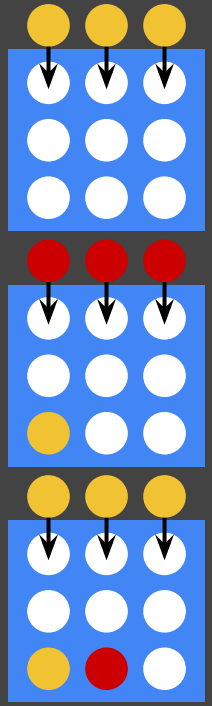
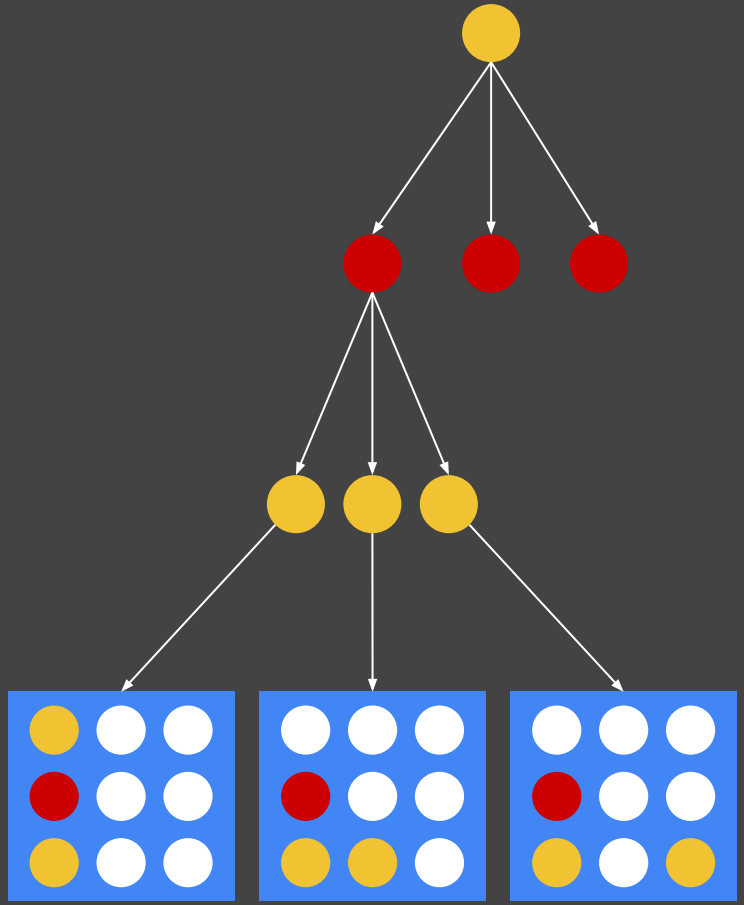
ALGORITHME MINIMAX ET ALPHA-BÊTA







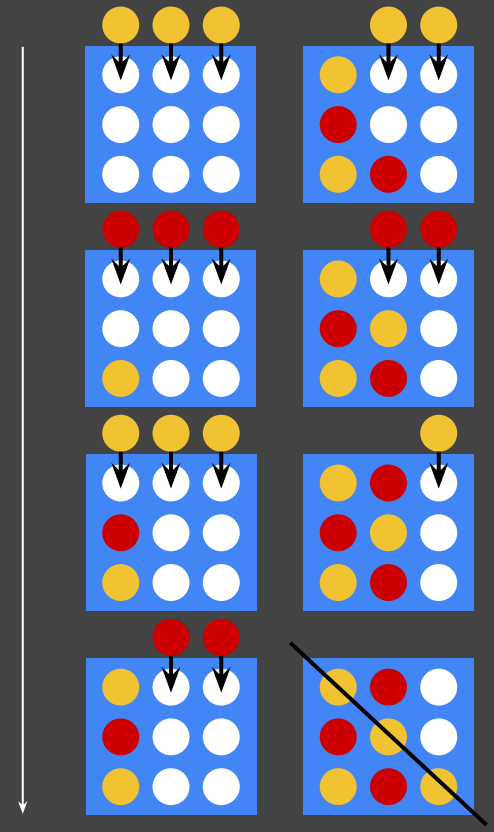
Au joueur jaune de jouer

Au joueur rouge de jouer

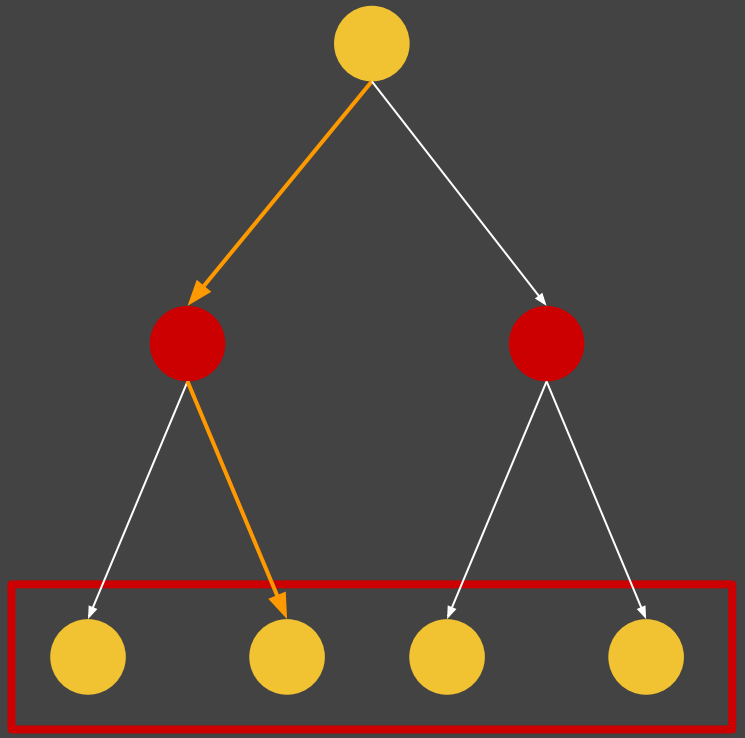
Au joueur jaune de jouer

Etat du jeu après 3 tours

Si je choisis toujours le choix le plus à gauche



Pour chaque solution (chemin dans l'arbre), continuer jusqu'à terminer le jeu ou jusqu'à une certaine profondeur au sein du graphe

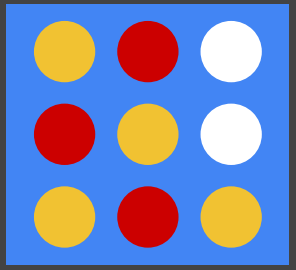


Lorsqu'on atteint la fin de l'arbre (fin du jeu ou profondeur déterminée) une évaluation de l'état du jeu est effectuée

L'évaluation est une fonction qui retourne un réel, positif ou négatif, pour signifier la domination ou non d'un joueur

Par exemple, on peut attribuer un nombre de points positifs ou négatifs (en fonction de la couleur) aux différents alignements (alignement consécutif de 2 jaunes = +3 points, de 4 rouges = -20 points, *etc.*)

La fonction effectue ensuite la somme des points et attribue une valeur à une feuille



1 alignement x3 jaune (20 points)
1 alignement x2 jaune (10 points)
2 alignement x2 rouge (-2 * 10 points)

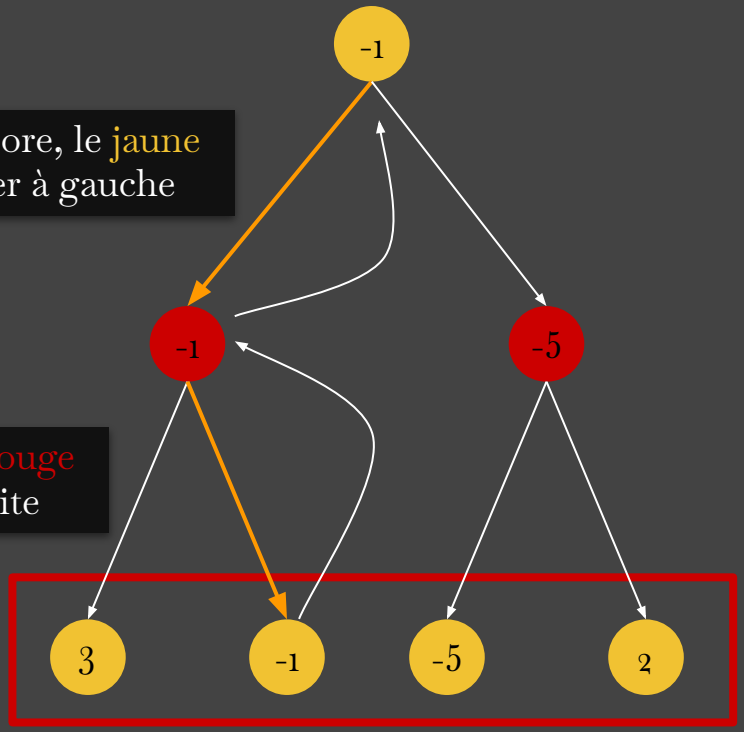
eval = +30 - 20 = +10

Minimisation pour le rouge

Maximisation pour le jaune

Maximiser le score, le jaune choisit de jouer à gauche

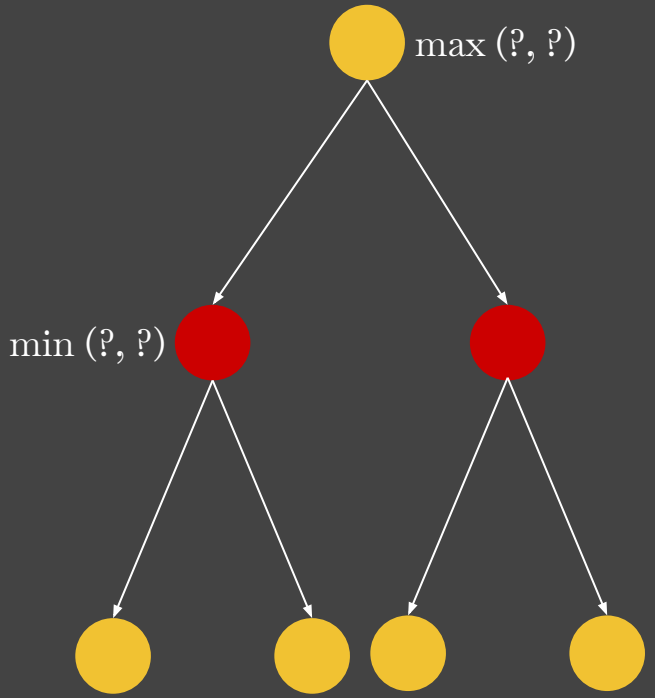
Minimiser le score, le rouge choisit de jouer à droite



Fonction d'évaluation

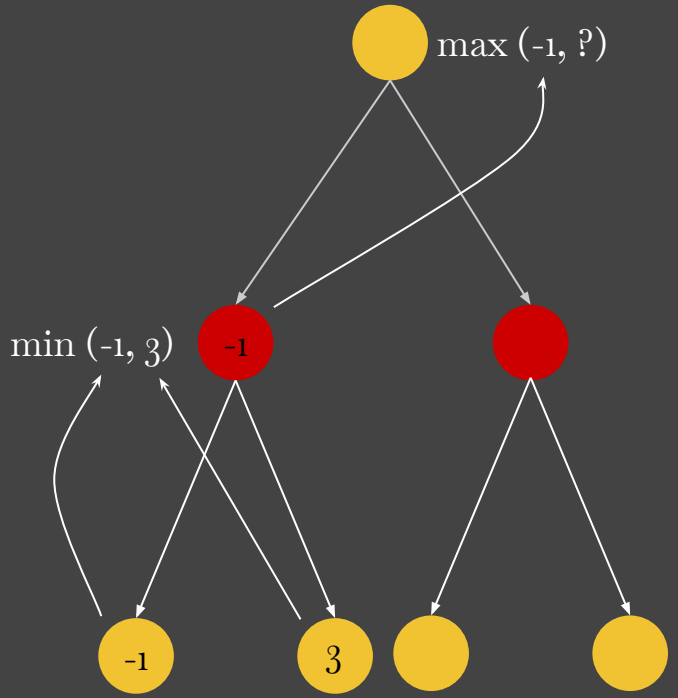
```
function minimax (position, depth, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, false)
                        maxEval = max (maxEval, eval)
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, true)
                        minEval = min (minEval, eval)
                return minEval
```

```
minimax (currentPosition, 2, true)
```



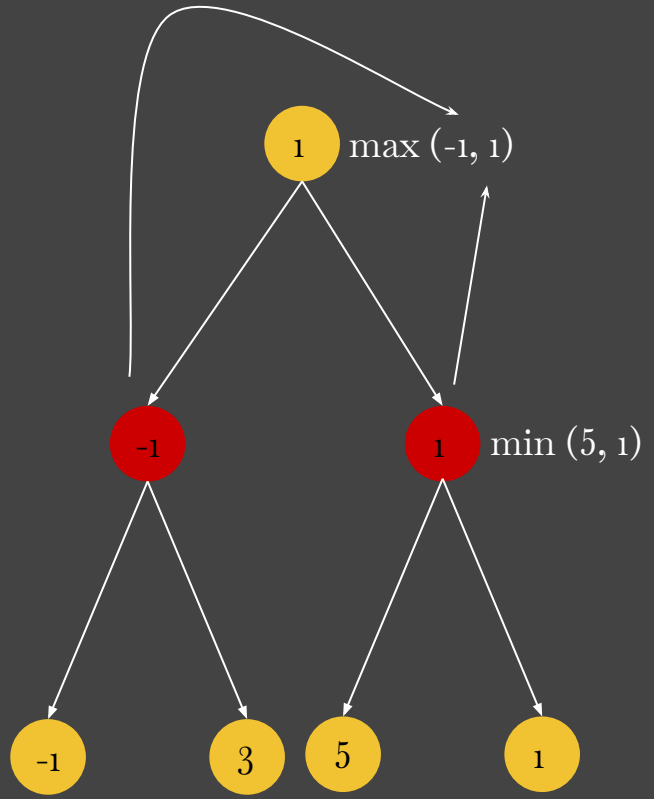
```
function minimax (position, depth, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, false)
                        maxEval = max (maxEval, eval)
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, true)
                        minEval = min (minEval, eval)
                return minEval
```

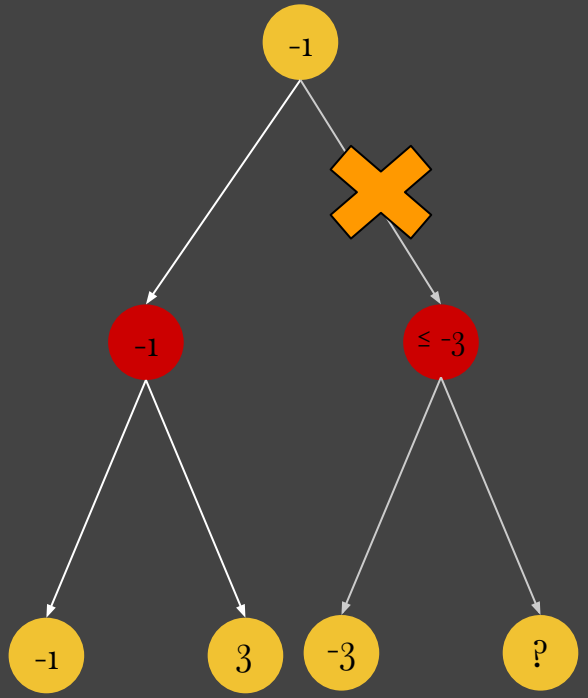
```
minimax (currentPosition, 2, true)
```



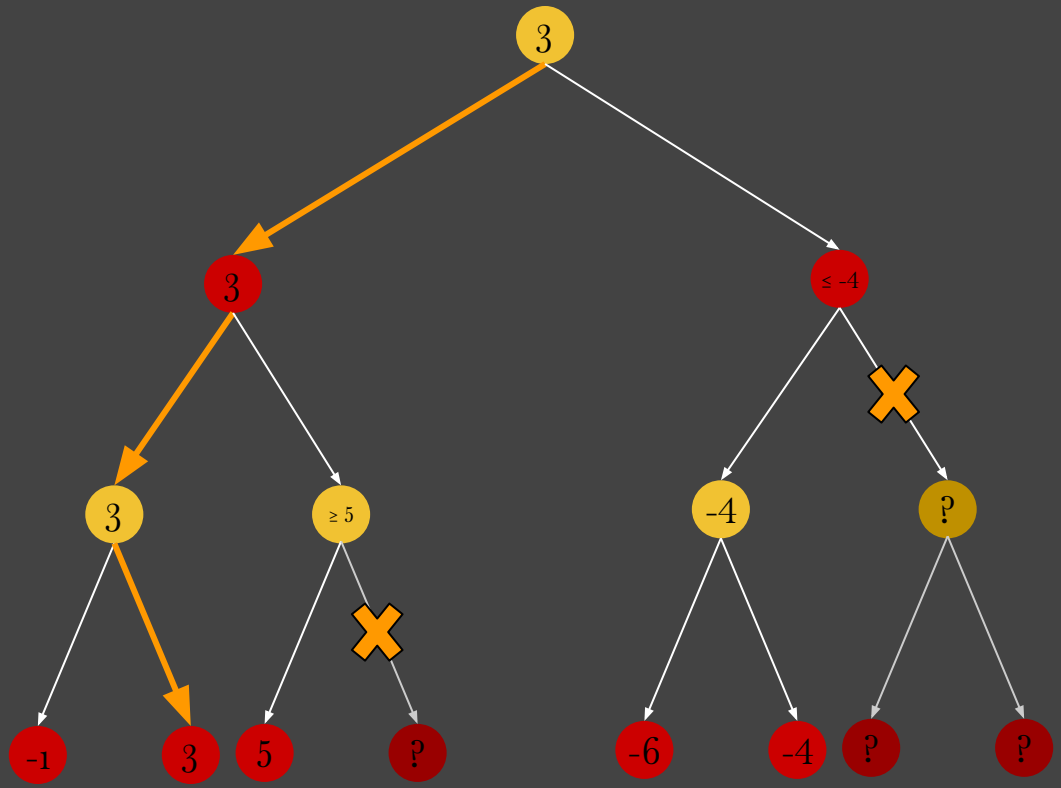
```
function minimax (position, depth, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, false)
                        maxEval = max (maxEval, eval)
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, true)
                        minEval = min (minEval, eval)
                return minEval
```

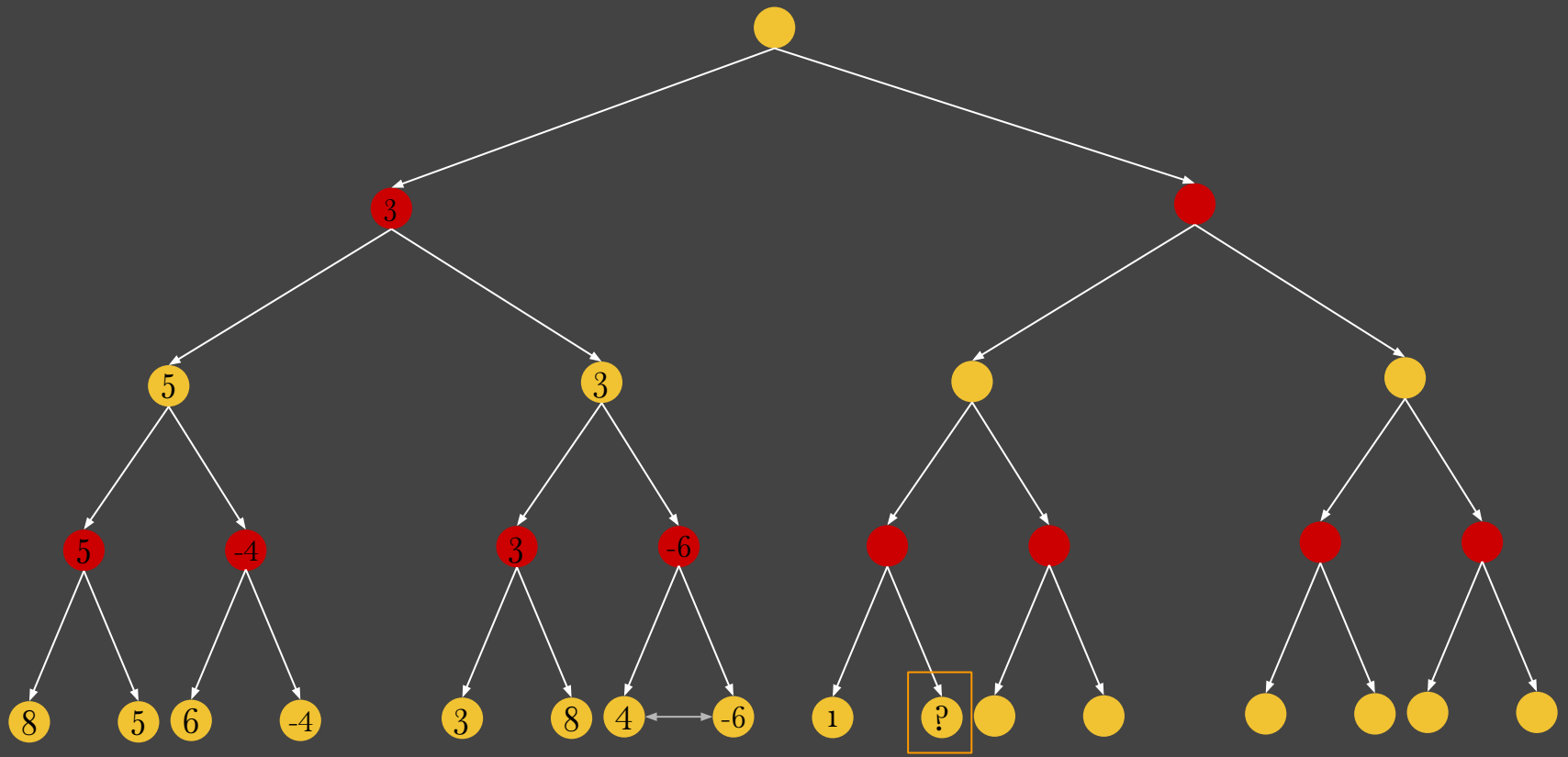
```
minimax (currentPosition, 2, true)
```



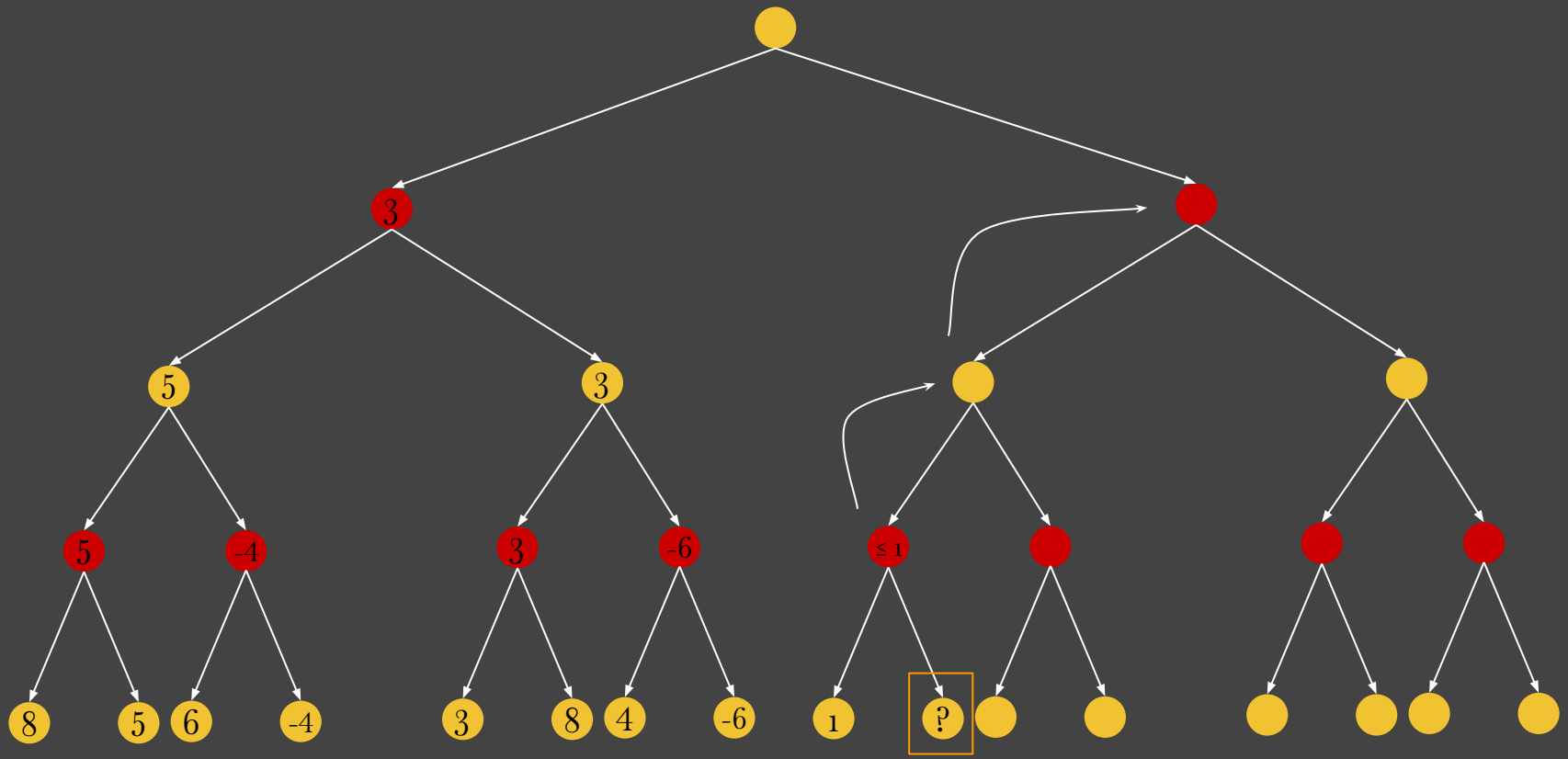


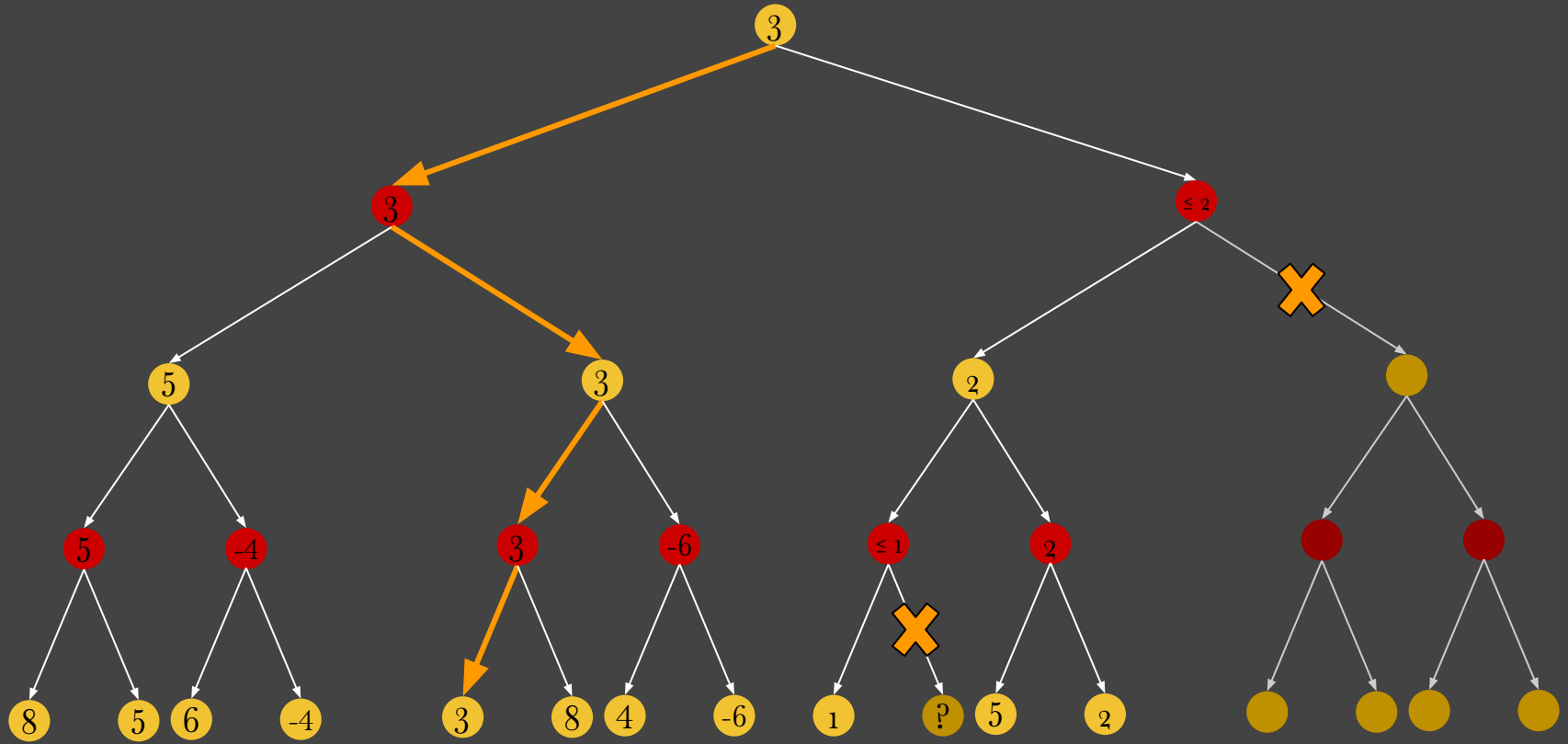
Alpha-bêta permet d'éviter de calculer des solutions sous-optimales


3
3
≤ -4
3
≥ 5
-4
?
-1
3
5
?
-6
-4
?
?



(L'ordre a une importance)

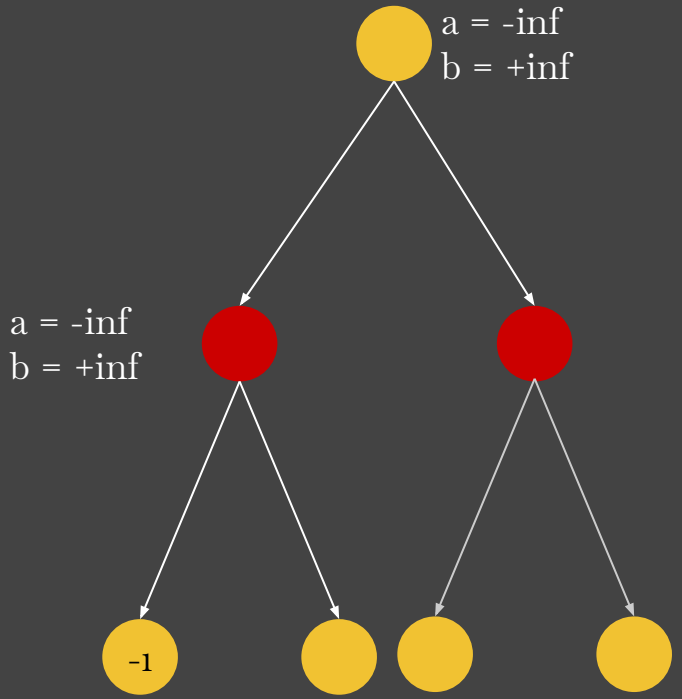

3
5
3
5
-4
3
-6
≤ 1
8
5
6
-4
3
8
4
-6
1
?


3
3
≤ 2
5
3
2
5
-4
3
-6
≤ 1
2
8
5
6
-4
3
8
4
-6
1
?
5
2

```
function minimax (position, depth, alpha, beta, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, false)
                        maxEval = max (maxEval, eval)
                        alpha = max (alpha, eval)
                        if beta <= alpha
                                break
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, true)
                        minEval = min (minEval, eval)
                        beta = min (beta, eval)
                        if beta <= alpha
                                break
                return minEval
```

```
minimax (currentPosition, 2, -inf, +inf, true)
```

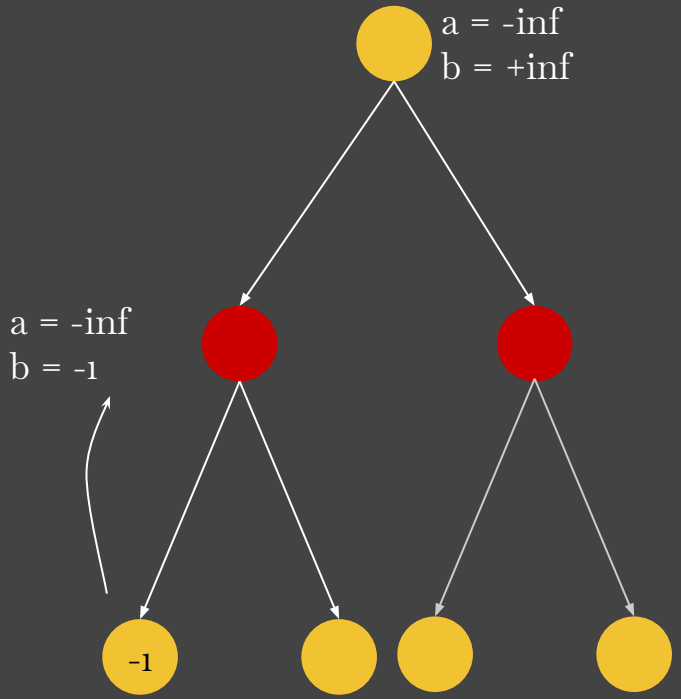
(-inf = pire score pour alpha et +inf = pire score pour beta)



```
function minimax (position, depth, alpha, beta, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, false)
                        maxEval = max (maxEval, eval)
                        alpha = max (alpha, eval)
                        if beta <= alpha
                                break
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, true)
                        minEval = min (minEval, eval)
                        beta = min (beta, eval)
                        if beta <= alpha
                                break
                return minEval
```

```
minimax (currentPosition, 2, -inf, +inf, true)
```

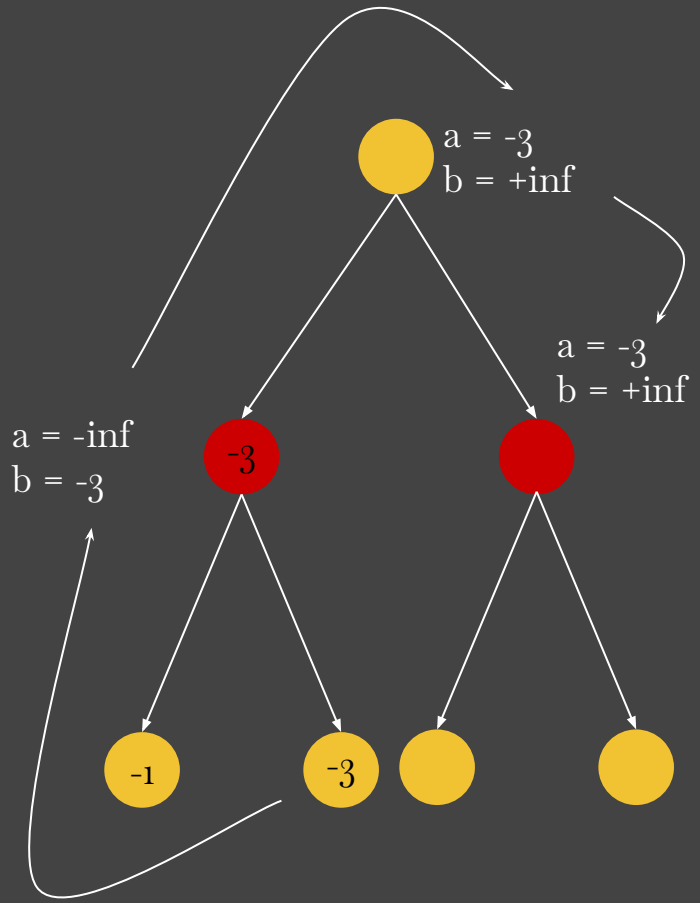
(-inf = pire score pour alpha et +inf = pire score pour beta)



```
function minimax (position, depth, alpha, beta, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, false)
                        maxEval = max (maxEval, eval)
                        alpha = max (alpha, eval)
                        if beta <= alpha
                                break
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, true)
                        minEval = min (minEval, eval)
                        beta = min (beta, eval)
                        if beta <= alpha
                                break
                return minEval
```

```
minimax (currentPosition, 2, -inf, +inf, true)
```

(-inf = pire score pour alpha et +inf = pire score pour beta)



```
function minimax (position, depth, alpha, beta, maximizingPlayer)
        if depth == 0 or game_over in position
                return evaluation of position

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, false)
                        maxEval = max (maxEval, eval)
                        alpha = max (alpha, eval)
                        if beta <= alpha
                                break
                return maxEval
        else
                minEval = +infinity
                for each child of position
                        eval = minimax (child, depth - 1, alpha, beta, true)
                        minEval = min (minEval, eval)
                        beta = min (beta, eval)
                        if beta <= alpha
                                break
                return minEval
```

```
minimax (currentPosition, 2, -inf, +inf, true)
```

(-inf = pire score pour alpha et +inf = pire score pour beta)