Théorème 1 (Zermelo) :

Soit T un ensemble d'issues possibles d'un jeu fini à deux joueurs à information parfaite. Alors de deux choses l'une :

- Soit le joueur I possède une stratégie qui lui permet d'assurer une issue dans T.
- Soit le joueur II possède une stratégie qui lui permet d'assurer une issue dans $\sim$T, le complémentaire de T dans l'ensemble des issues possibles.

Preuve :

Algorithme de Zermelo → Preuve par **récurrence forte sur la hauteur de l'arbre**.

On considérera sans perte de généralité que le joueur I commence toujours une partie.

Considérons l'arbre représentant le jeu. Un sommet de cet arbre sera étiqueté :

- « I, T » si et seulement si le joueur le joueur I peut, à partir de cette position, assurer une issue dans l'ensemble T
- « II, $\sim$T » si et seulement si le joueur II peut, à partir de cette position, assurer une issue dans l'ensemble $\sim$T.

Montrons que tout sommet peut être étiqueté de la sorte, et en particulier que la racine peut l'être :

$\mathcal{H}_n$: « Tout sommet d'un arbre de hauteur n peut être étiqueté selon les règles établies ci-dessus. »

Initialisation : $\mathcal{H}_0$ est bien vérifiée.

Hérédité : Supposons $\mathcal{H}_i$ vérifiée pour tout $i \in [\![0\,;\,n]\!]$.

Soit un arbre de hauteur $n+1$; considérons tous les sous-arbres issus des sommets contigus à sa racine → Sous-arbres de hauteur au plus n.

↳ Par hypothèse de récurrence, tous leurs sommets (et en particulier leurs racines) peuvent être étiquetés.

Considérons à présent la racine de notre arbre de hauteur $n+1$.

↳ Si un des sommets contigus à la racine est étiqueté « I, T », alors la racine peut être étiquetée de même.

↳ Si aucun des sommets contigus à la racine n'est étiqueté « I, T », alors la racine peut être étiquetée « II, $\sim$T ».

Dans les deux cas, la racine peut être étiquetée : $\mathcal{H}_{n+1}$ est donc vérifiée.

Le principe de récurrence forte permet donc de conclure. ■

Corollaire 1 :

Dans le jeu de Morpionception, un des deux joueurs possède une stratégie permettant d'obtenir, dans le pire des cas selon ses critères, un nul.

Preuve :

Considérons :

$$T = \{ \text{"Victoire du joueur jouant avec les croix"} \}$$

D'après le théorème 1, soit le joueur jouant avec les croix peut assurer une issue dans T, soit le joueur jouant avec les ronds peut assurer une issue dans :

$$\sim T = \{ \text{"Victoire du joueur jouant avec les ronds"} , \text{"Match nul"} \}$$

Ainsi, l'un des deux joueurs peut s'assurer au moins le nul, sinon la victoire. ■

Théorème 2 (Zermelo, appliqué au Morpionception) :
Si chacun des deux joueurs joue de manière parfaitement rationnelle, une partie de Morpionception aura toujours sur la même issue.

Preuve : Considérons :

$$T_1 = \{ \text{"Victoire du joueur jouant avec les croix"} \}$$

$$T_2 = \{ \text{"Victoire du joueur jouant avec les ronds"} \}$$

Utilisons le théorème 1 sur l'ensemble d'issues T_1.

Cas 1 : Le joueur jouant avec les croix peut assurer une issue dans T_1. Puisqu'il joue de manière parfaitement rationnelle, toute partie de Morpionception s'achèvera donc sur la victoire de ce joueur.

Cas 2 : Le joueur jouant avec les ronds peut assurer une issue dans :

$$\sim T_1 = \{ \text{"Victoire du joueur jouant avec les ronds"} , \text{"Match nul"} \}$$

Utilisons à nouveau le théorème 1, cette fois-ci sur l'ensemble d'issues T_2.

- Cas 2.1 : Le joueur jouant avec les ronds peut assurer une issue dans T_2. Puisqu'il joue de manière parfaitement rationnelle, toute partie de Morpionception s'achèvera donc sur la victoire de ce joueur.

- Cas 2.2 : Le joueur jouant avec les croix peut assurer une issue dans :

  $$\sim T_2 = \{ \text{"Victoire du joueur jouant avec les croix"} , \text{"Match nul"} \}$$

  et le joueur jouant avec les ronds peut assurer une issue dans :

  $$\sim T_1 = \{ \text{"Victoire du joueur jouant avec les ronds"} , \text{"Match nul"} \}$$

  Chacun des deux joueurs a donc la possibilité d'éviter la victoire de l'autre. Comme ils jouent tous deux de manière parfaitement rationnelle, et que le jeu est **strictement compétitif**, alors toute partie de Morpionception s'achèvera sur un nul.

Le jeu de Morpionception vérifiant les hypothèses d'un (et d'un seul) de ces trois cas, une partie de Morpionception aura toujours la même issue. ■

Algorithme du MinMax « naïf » (première étude informatique) :

```
from copy import deepcopy


def MinMax (Morpion, morpion_intermediaire, Joueur, a, b) :

    if victoire_morpion(morpion_intermediaire) == "Victoire croix":
        return 1

    if victoire_morpion(morpion_intermediaire) == "Victoire ronds":
        return -1

    if victoire_morpion(morpion_intermediaire) == "Match nul":
        return 0


    Liste_coups_contrainte = coups_disponibles_morpion(Morpion[a,b])
    Liste = []

    if Joueur == "Croix" :
        if Liste_coups_contrainte != [] :
            for coup in Liste_coups_contrainte:
                morpion_jeu = deepcopy(Morpion)
                morpion_intermediaire_jeu = deepcopy(morpion_intermediaire)
                avance_configuration_morpionception(morpion_jeu,
                morpion_intermediaire_jeu, Joueur, (a,b,coup[0], coup[1]))
                Liste+= [MinMax(morpion_jeu, morpion_intermediaire_jeu,
                "Ronds", coup[0], coup[1])]


        else :
            for coup in coups_disponibles_morpionception(Morpion):
                morpion_jeu = deepcopy(Morpion)
                morpion_intermediaire_jeu = deepcopy(morpion_intermediaire)
                avance_configuration_morpionception(morpion_jeu,
    morpion_intermediaire_jeu, Joueur, coup)
                Liste+= [MinMax(Morpion, morpion_intermediaire, "Ronds",
    coup[2], coup[3])]

        return max(Liste)


    if Joueur == "Ronds" :
        (…)
```

Algorithme du MinMax élagué par méthode Alpha-Beta :

```
@memo_morpionception_morpion_joueur_a_b_AlphaBeta
def MinMax_AlphaBeta (Morpion, morpion_intermediaire, Joueur, a, b,
AlphaBeta):

    if victoire_morpion(morpion_intermediaire) == "Victoire croix":
        return 1
    if victoire_morpion(morpion_intermediaire) == "Victoire ronds":
        return -1
    if victoire_morpion(morpion_intermediaire) == "Match nul":
        return 0

    Liste_coups_contrainte = coups_disponibles_morpion(Morpion[a,b])

    if Joueur == "Croix":
        max = -1

        if Liste_coups_contrainte != []:
            for coup in Liste_coups_contrainte :
                morpion_jeu = deepcopy(Morpion)
                morpion_intermediaire_jeu = deepcopy(morpion_intermediaire)
                avance_configuration_morpionception(morpion_jeu,
                morpion_intermediaire_jeu, Joueur, (a, b, coup[0], coup[1]))

                v = MinMax_AlphaBeta(morpion_jeu, morpion_intermediaire_jeu,
                "Ronds", coup[0], coup[1], max)

                if v >= max:
                        max = v
                        if max >= AlphaBeta:
                            return AlphaBeta
            return max


        else :
            for coup in coups_disponibles_morpionception(Morpion) :
                morpion_jeu = deepcopy(Morpion)
                morpion_intermediaire_jeu = deepcopy(morpion_intermediaire)
                avance_configuration_morpionception(morpion_jeu,
                morpion_intermediaire_jeu, Joueur, coup)

                v = MinMax_AlphaBeta(morpion_jeu, morpion_intermediaire_jeu,
                "Ronds", coup[2], coup[3], max)

                if v >= max:
                        max = v
                        if max >= AlphaBeta:
                            return AlphaBeta
            return max



    if Joueur == "Ronds":
        min = 1
        (…)
```

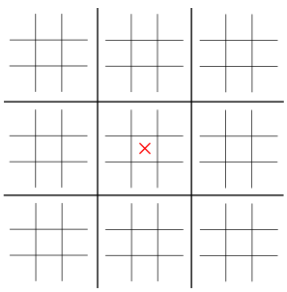
Algorithme du MinMax élagué par une heuristique :

```
@memo_morpionception_morpion_joueur_a_b_profondeur_JeuDeCoeff_delta_
ValeurStop
def MinMax_heuristique (Morpion, morpion_intermediaire, Joueur, a, b,
profondeur, jeu_de_coeffs, Delta, valeur_stop_exploration):

    if victoire_morpion(morpion_intermediaire) == "Victoire croix":
        return 1

    if victoire_morpion(morpion_intermediaire) == "Victoire ronds":
        return -1

    if victoire_morpion(morpion_intermediaire) == "Match nul":
        return 0


    Liste = []

    check_list = exploration_profondeur_limitee (Morpion,
    morpion_intermediaire, Joueur, a, b, 0, profondeur)
    Liste_valeurs = []

    for elem in check_list :
        Liste_valeurs+=[eval(elem[0], elem[1], elem[2], elem[3], elem[4],
        jeu_de_coeffs)]
    valeur_max = max(Liste_valeurs)
    valeur_min = min(Liste_valeurs)


    if Joueur == "Croix" :
        for i in range (len(Liste_valeurs)) :
            if valeur_max - Liste_valeur[i] < Delta and Liste_valeur[i] >
            -valeur_stop_exploration :
                Liste+=[MinMax_heuristique(check_list[i][0], check_list[i][1],
                check_list[i][2],check_list[i][3], check_list[i][4], profondeur,
                jeu_de_coeffs, Delta, valeur_stop_exploration)]

        if Liste !=[] :
            return max(Liste)

        else :
            return -1


    if Joueur == "Ronds":
        for i in range (len(Liste_valeurs)) :
            if Liste_valeur[i] - valeur_min < Delta and Liste_valeur[i] <
            valeur_stop_exploration :
                Liste+=[MinMax_heuristique(check_list[i][0], check_list[i][1],
                check_list[i][2],check_list[i][3], check_list[i][4], profondeur,
                jeu_de_coeffs, Delta, valeur_stop_exploration)]

        if Liste !=[] :
            return min(Liste)

        else :
            return 1
```

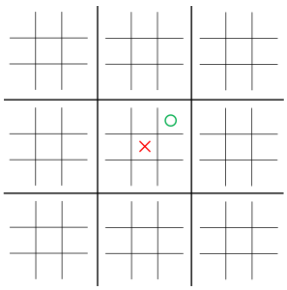
Algorithme du MinMax optimisé (seconde étude informatique) :

```
@memo_morpionception_morpion_joueur_a_b_profondeur_JeuDeCoeff_delta_
ValeurStop_
AlphaBeta
def MinMax_heuristique_AlphaBeta(Morpion, morpion_intermediaire, Joueur,
a, b, profondeur, jeu_de_coeffs, Delta, valeur_stop_exploration, AlphaBeta):

    if victoire_morpion(morpion_intermediaire) == "Victoire croix":
        return 1
    if victoire_morpion(morpion_intermediaire) == "Victoire ronds":
        return -1
    if victoire_morpion(morpion_intermediaire) == "Match nul":
        return 0


    check_list = exploration_profondeur_limitee (Morpion,
    morpion_intermediaire, Joueur, a, b, 0, profondeur)
    Liste_valeurs = []

    for elem in check_list :
        Liste_valeurs+=[eval(elem[0], elem[1], elem[2], elem[3], elem[4],
        jeu_de_coeffs)]

    valeur_max = max(Liste_valeurs)
    valeur_min = min(Liste_valeurs)

    if Joueur == "Croix" :
        max = -1
        for i in range (len(Liste_valeurs)) :
            if valeur_max - Liste_valeur[i] < Delta and Liste_valeur[i] >
            -valeur_stop_exploration :

                v = MinMax_heuristique_AlphaBeta(check_list[i][0], check_list[i][1],
                check_list[i][2],check_list[i][3], check_list[i][4], profondeur,
                jeu_de_coeffs, Delta, valeur_stop_exploration, max)

                if v >= max:
                        max = v
                        if max >= AlphaBeta:
                            return AlphaBeta
        return max

    if Joueur == "Ronds":
        min = 1
        for i in range (len(Liste_valeurs)) :
            if Liste_valeur[i] - valeur_min < Delta and Liste_valeur[i] <
            valeur_stop_exploration :

                v = MinMax_heuristique_AlphaBeta(check_list[i][0], check_list[i][1],
                check_list[i][2],check_list[i][3], check_list[i][4], profondeur,
                jeu_de_coeffs Delta, valeur_stop_exploration, min)

                if v <= min:
                        min = v
                        if min <= AlphaBeta:
                            return AlphaBeta
        return min
```

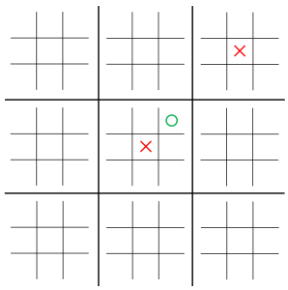
Illustration d'un début de partie joué en imposant au joueur jouant avec les croix la stratégie de la conjecture :



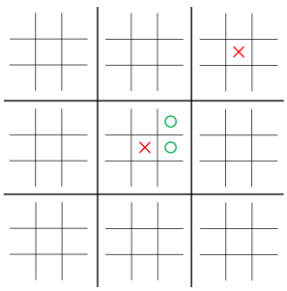
Position 1



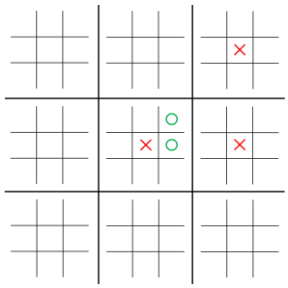
Position 2



Position 3

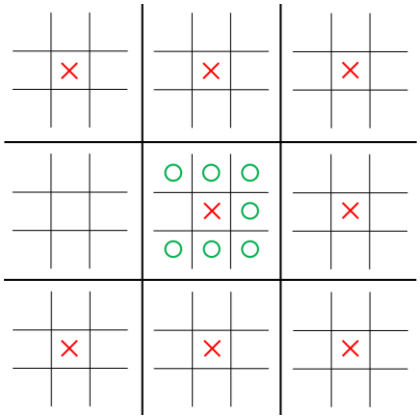


Position 4



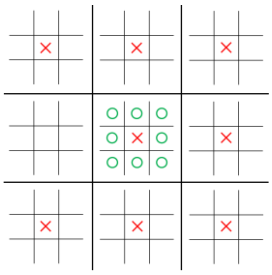
Position 5

Au bout de 15 coups, on arrive par exemple à la position suivante :

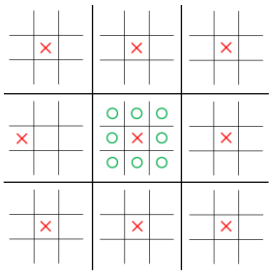


Position 15

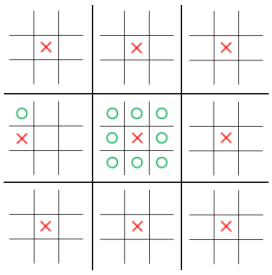
La partie peut alors se poursuivre de la manière suivante :



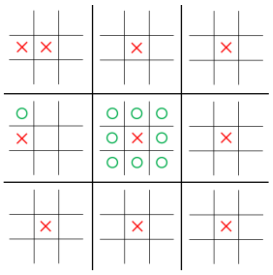
Position 16



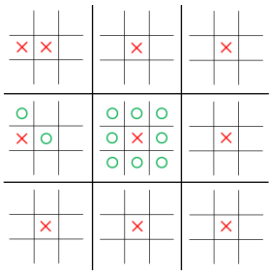
Position 17



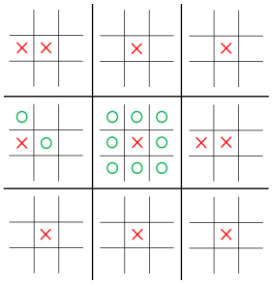
Position 18



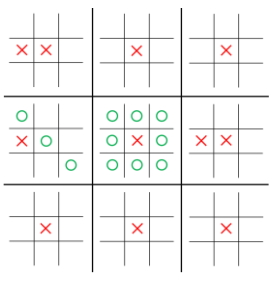
Position 19



Position 20



Position 21



Position 22