# Projet informatique: Puissance 4

Intelligence artificielle et jeu en ligne

Valentin Meunier
Thomas Prévost

# SOMMAIRE

# Présentation générale du projet

But du projet : Réalisation d'un jeu de puissance 4 avec une intelligence artificielle et un jeu en ligne intégrés

Partage des tâches :

-Valentin : Intelligence Artificielle

-Thomas : Jeu en ligne

# Intelligence Artificielle

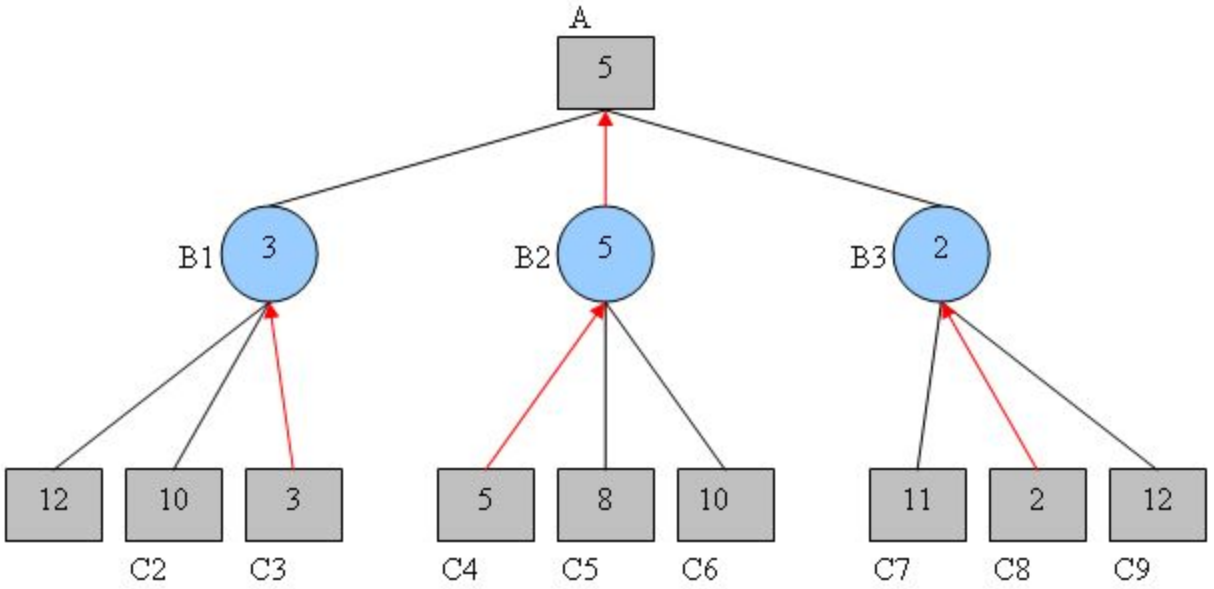
**Puissance 4 :** Jeu de stratégie combinatoire abstrait -> 2 joueurs, tour par tour, jeu à information complète, jeu à somme nulle, pas de hasard

**Algorithme MinMax :** Algorithme de la théorie des jeux pouvant être utilisé dans les jeux à 2 joueurs, à somme nulle et à information complète

**Élagage Alpha-Beta :** Technique permettant de réduire le nombre de noeuds évalué par l'algorithme MinMax

# Intelligence Artificielle

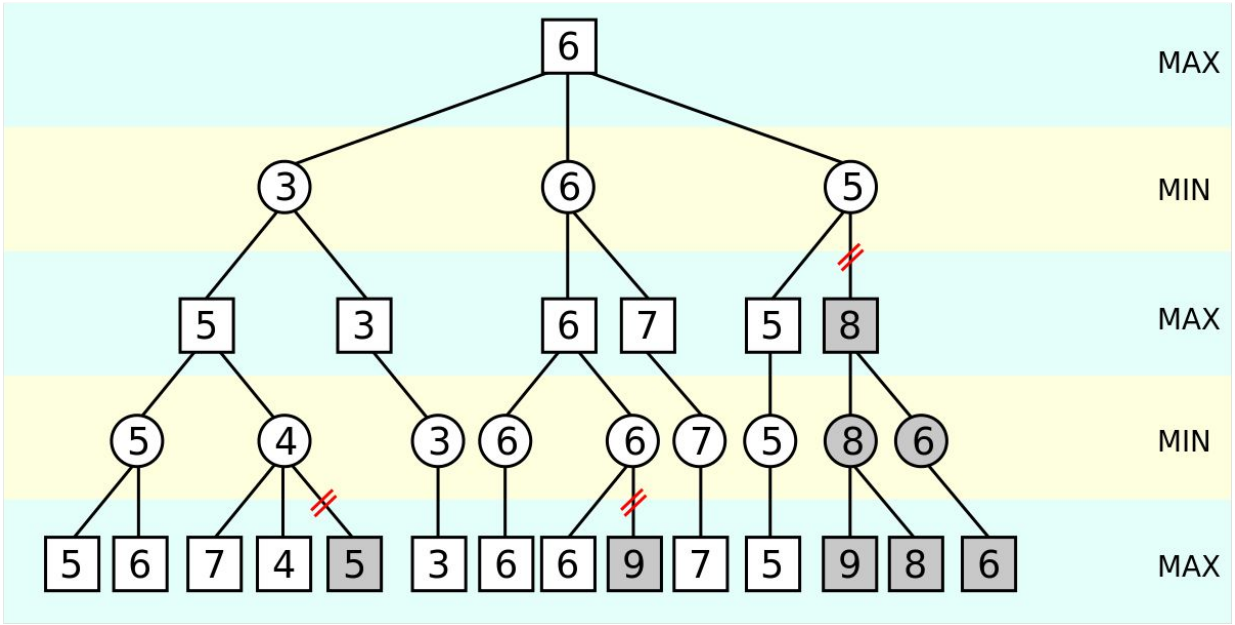
**Algorithme MinMax :**




Source : Wikipedia

# Intelligence Artificielle

**Élagage Alpha-Beta :**




**Puissance 4:**

Sans Alpha-Beta : ≈20000 tests par coups

Avec Alpha-Beta : ≈500 tests par coups

(début de partie, profondeur 5)

Principe : Si la valeur d'un fils d'un noeud min est inférieure à la valeur courante d'un noeud max parent, alors ses frères n'ont pas besoin d'être explorés et inversement

Source : Wikipedia

# Structure du programme

```
class Grille(object):...

class Coup(object):...

class Jeu(object):...

class JoueurHumain(object):...

class IaAleatoire(object):...

class PartieConsole(object):...

class IAMinMax(object):...
```

# Algorithme MinMax

```python
def _choix_minmax(self):
    """
    Détermine le meilleur coup possible pour l'IA
    Cette méthode n'est appelée qui s'il y a au moins un coup jouable pour l'IA
    :return: colonne jouée
    """
    # Il faut retenir le coup qui donnera le score maximal pour l'IA.
    # On initialise donc "score_max" à la valeur la moins élevée...
    score_max = -self._score_max - 1  # -1 pour être certain que "score > score_max" sera vérifié au moins une fois

    # ...puis pour chaque premier coup possible pour l'IA...
    for colonne in self._jeu.colonnes_jouables:
        # ...on simule le coup...
        coup = self._jeu.jouer(colonne, self.pion)

        # ...puis on simule le meilleur coup possible de l'adversaire
        # et on récupère le score obtenu en évaluant la situation du point de vue de l'IA
        score = self.simuler_coup_adversaire(self._profondeur - 1)
        self._jeu.annuler(coup)  # annulation du coup simulé

        # Il faut donc retenir le score le plus avantageux pour l'IA...
        if score > score_max:
            score_max = score
            coup_retenu = coup

        # Puis on recommence pour rechercher un éventuel meilleur premier coup
        # (si tous les coups possibles ont été testés la boucle s'arrête)

    # Parmi tous les coups possibles de l'IA on retourne le meilleur
    return coup_retenu.colonne
```

Sans Alpha-Beta

```python
def _choix_alpha_beta(self):
    """
    Détermine le meilleur coup possible pour l'IA
    Cette méthode n'est appelée qui s'il y a au moins un coup jouable pour l'IA
    :return: colonne jouée
    """
    alpha = -self._score_max
    beta = self._score_max
    # Il faut retenir le coup qui donnera le score maximal pour l'IA.
    # On initialise donc "score_max" à la valeur la moins élevée...
    score_max = -self._score_max - 1  # -1 pour être certain que "score > score_max" sera vérifié au moins une fois
    # ...puis pour chaque premier coup possible pour l'IA...
    for colonne in self._jeu.colonnes_jouables:
        # ...on simule le coup...
        coup = self._jeu.jouer(colonne, self.pion)
        # ...puis on simule le meilleur coup possible de l'adversaire
        # et on récupère le score obtenu en évaluant la situation du point de vue de l'IA
        score = self.simuler_coup_adversaire_alpha_beta(self._profondeur - 1, alpha, beta)
        self._jeu.annuler(coup)  # annulation du coup simulé
        # Il faut donc retenir le score le plus avantageux pour l'IA...
        if score > score_max:
            score_max = score
            coup_retenu = coup
        if score_max >= beta:
            break
        alpha = max(alpha, score_max)
        # Puis on recommence pour rechercher un éventuel meilleur premier coup
        # (si tous les coups possibles ont été testés la boucle s'arrête)
    # Parmi tous les coups possibles de l'IA on retourne le meilleur
    return coup_retenu.colonne
```

Avec Alpha-Beta

# Fonction de simulation de coup

```python
def simuler_coup_adversaire(self, profondeur):
    """
    Simulation des coups possibles pour l'adversaire
    If faut retenir le score minimal du point de vue de l'IA
    C'est la fonction "min" de l'algorithme minmax
    :param profondeur: profondeur maximale de recherche des coups
    :return: score obtenu le plus défavorable du point de vue de l'IA
    """
    # Si la profondeur maximale est atteinte (-1 si départ avec profondeur=0, 0 autrement) ou si le jeu est fini,
    # il faut évaluer la situation suite au dernier coup joué. Ce coup a été joué par l'IA
    if profondeur <= 0 or self._jeu.fini:
        return self._evaluer_score(self.pion)

    # On va simuler les coups possibles de l'adversaire en réponse au dernier coup joué par l'IA
    # On va retenir le score minimal obtenu en évaluant la situation du point de vue de l'IA
    # On va donc récupérer la pire situation dans laquelle va se trouver l'IA suite au coup de l'adversaire
    score_min = self._score_max

    for colonne in self._jeu.colonnes_jouables:
        coup = self._jeu.jouer(colonne, self._pion_adversaire)
        score = self.simuler_coup_ia(profondeur - 1)
        self.nbre_coups_simules += 1
        self._jeu.annuler(coup)

        score_min = min(score, score_min)

    return score_min
```

```python
def simuler_coup_ia_alpha_beta(self, profondeur, alpha, beta):
    """
    Simulation des coups possibles pour l'IA
    If faut retenir le score maximal du point de vue de l'IA
    C'est la fonction "max" de l'algorithme minmax
    :param profondeur: profondeur maximale de recherche des coups
    :param alpha: valeur minimale que l'IA peut espérer obtenir en jouant le coup
    :param beta: valeur maximale que l'IA peut espérer obtenir en jouant le coup
    :return: score obtenu le plus favorable du point de vue de l'IA
    """
    # Si la profondeur maximale est atteinte (-1 si départ avec profondeur=0, 0 autrement) ou si le jeu est fini,
    # il faut évaluer la situation suite au dernier coup joué. Ce coup a été joué par l'adversaire
    if profondeur <= 0 or self._jeu.fini:
        return self._evaluer_score(self._pion_adversaire)
    # On va simuler les coups possible de l'IA en réponse au dernier coup joué par l'adversaire
    # On va retenir le score maximal obtenu en évaluant la situation du point de vue de l'IA
    # On va donc récupérer la meilleur situation dans laquelle va se trouver l'IA après qu'elle ait jouée
    score_max = -self._score_max
    for colonne in self._jeu.colonnes_jouables:
        coup = self._jeu.jouer(colonne, self.pion)
        score = self.simuler_coup_adversaire_alpha_beta(profondeur - 1, alpha, beta)
        self.nbre_coups_simules += 1
        self._jeu.annuler(coup)

        score_max = max(score, score_max)
        # coupure beta
        if score_max >= beta:
            break
        alpha = max(alpha, score_max)

    return score_max
```

Adversaire, Sans Alpha-Beta

IA, Avec Alpha-Beta

- Possibilité de réunir ces deux fonctions en une seule : Négamax

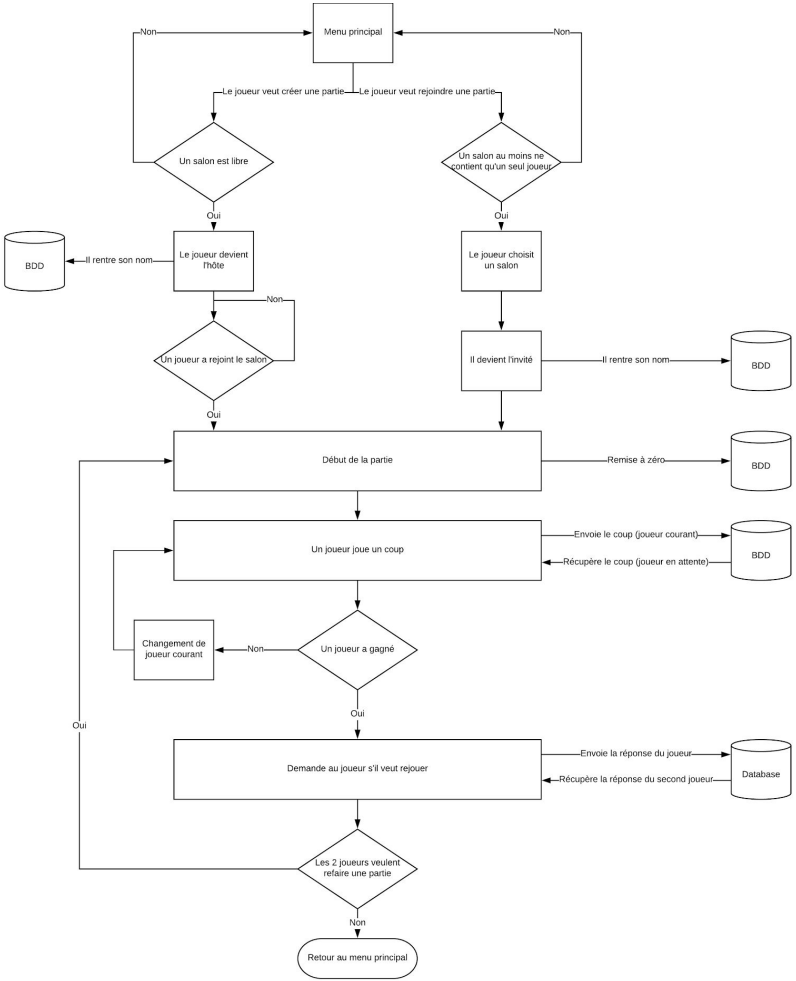
# Fonction d'évaluation

```python
def _evaluer_score(self, pion):
    """
    Calcul une évaluation de la situation du jeu du point de vue de l'IA
    :param pion: c'est le dernier pion qui a été joué
    :return: un entier compris entre -score_max et +score_max
    """
    if self._jeu.gagne:
        if pion == self.pion:
            # la partie est gagnée par l'IA
            return self._score_max
        else:
            # la partie est gagnée par l'adversaire
            return -self._score_max

    elif self._jeu.nul:
        # égalité
        return 0

    # évaluation de la situation du point de vue de l'IA

    # on calcule le nombre maximal de cases alignées pour l'Ia et l'adversaire
    nbre_pions_alignes_ia = self._jeu.nbre_cases_alignees_pion(self.pion)
    nbre_pions_alignes_adversaire = self._jeu.nbre_cases_alignees_pion(self._pion_adversaire)

    # on calcule le score pour chaque joueur (arrondi à l'entier inférieur) tel que la différence reste comprise
    # entre -score_max et score_max
    score_ia = nbre_pions_alignes_ia * self._score_max // (2 * self._jeu.nbre_cases_gain)
    score_adversaire = nbre_pions_alignes_adversaire * self._score_max // (2 * self._jeu.nbre_cases_gain)

    # le score relatif pour l'IA est la différence
    return score_ia - score_adversaire
```

# Jeu en ligne

**Base de données en temps réel :** Système de base de données basée sur un système temps réel, permettant de prendre en charge des données constamment variables et de les restituer immédiatement

**NoSQL (Not only SQL) :** Famille de systèmes de gestion de base de données s'écartant de la structure classique d'une base de données relationnelle


Menu principal
Non
Non
Le joueur veut créer une partie
Le joueur veut rejoindre une partie
Un salon est libre
Un salon au moins ne contient qu'un seul joueur
Oui
Oui
BDD
Il rentre son nom
Le joueur devient l'hôte
Le joueur choisit un salon
Non
Un joueur a rejoint le salon
Il devient l'invité
Il rentre son nom
BDD
Oui
Début de la partie
Remise à zéro
BDD
Un joueur joue un coup
Envoie le coup (joueur courant)
Récupère le coup (joueur en attente)
BDD
Changement de joueur courant
Non
Un joueur a gagné
Oui
Oui
Demande au joueur s'il veut rejouer
Envoie la réponse du joueur
Récupère la réponse du second joueur
Database
Les 2 joueurs veulent refaire une partie
Non
Retour au menu principal

# Connexion à la base de données

```
def init():

    oppName = "none"
    selfName = "none"
    # On récupère les informations de connexion
    path = os.path.join(sys.path[ 0], 'service-account-credentials.json' )

    cred = credentials.Certificate( 'serviceAccountCredentials.json' )
    # On initialise la connexion au serveur
    firebase_admin.initialize_app(cred, {
        'databaseURL' : 'https://puissance-4-cyka.firebaseio.com/' ,
        'databaseAuthVariableOverride' : None
    })

    global ref
    ref = db.reference("/1v1-online") #On prend la partie "online" de la BDD
```

# Manipulation des salons

```
def newlobby():
    if (status=='h'):
        lobbylist = ref.get() #On récupère la liste des salons sur la BDD
        for lobby in lobbylist:
            current = ref.child(lobby)
            global myref
            myref = ref.child(lobby)
            if(current.child('gameOn').get()=="False"):
                current.update({"gameOn" : "True"})
                myref.update({"playercount" : 1})
                resetlobby() #On remet le salon à zéro
                getName() #On demande au joueur son nom
                wait() #On attend un adversaire
                return True
        print('Malheureusement, il n\'y a pas de salon disponible...')
        input('Presser une touche pour continuer...')
        subprocess.call("launch.bat" if os.name=='nt' else 'welcome.py', shell=True)
                return False
           else:
                joinlobby()
```

Création d'un salon

```
def joinlobby():
    screenClear() #On efface la console
    print("Salons disponibles :")
    showlobbies()
    i = input("Lequel voulez-vous rejoindre ?)

    [...] #Gestion de la saisie

        lobby = 'lobby'+i #On fabrique une référence à la BDD à partir de l'entrée
utilisateur
        global myref
        myref = ref.child(lobby) #On établit la connexion au salon choisi
        getName() #On récupère le nom du joueur
        myref.update ({'playercount' : 2}) #On met à jour la BDD
        global savedLp
        savedLp = myref.child('h_last').get() #
        return True
    return False
```

Connexion à un salon

# Actions des joueurs

```
def hisTurn():
    global savedLp
    i = 0
    while(myref.child(o_last).get() == savedLp): #On affiche un message d'attente
tant que l'adversaire n'a pas joué
        time.sleep(0.5)
        sys.stdout.write("\r" + 'L\'adversaire joue.' + (i % 3) * '.' + (16 - (i %
4)) * ' ')
        sys.stdout.flush()
        i+=1
    n = myref.child(o_play).get() #On récupère le coup de l'adversaire
    savedLp = myref.child(o_last).get()
    return n #On renvoie le coup de l'adversaire
```

L’adversaire joue

```
def myturn(n):
    myref.update({s_play : n}) #On enregistre le coup
    myref.update({s_last : time.time()}) #On enregistre le timecode
    return True
```

L’utilisateur joue

# Sources


http://pauillac.inria.fr/~fpottier/X/INF441/projets/alpha-beta/sujet.pdf

http://leiber.free.fr/?http://leiber.free.fr/programmation/ia/

https://ljk.imag.fr/membres/Clement.Pernet/Enseignements/L3METI_ProgOO/MinMaxAlphaBeta.pdf

https://firebase.google.com/docs/

https://stackoverflow.com/