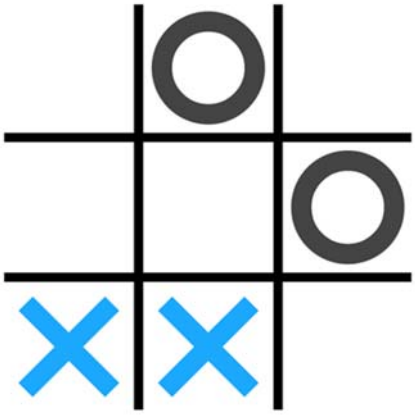
Tic Tac Toe



Le jeu Tic Tac Toe (jeu du morpion) se pratique à deux joueurs en tour par tour. Le but est de créer le premier un alignement sur une grille. Chaque joueur dispose de son marqueur : croix, trait ou cercle et d'une couleur. Le support du jeu est une grille 3x3. Les alignements possibles de trois formes peuvent s'effectuer en vertical, en horizontal ou sur les diagonales.

Ouvrez le projet et lancez la partie. L'interface du jeu réagit au clic de la souris sur la grille. Vous pouvez ainsi disposer plusieurs marqueurs.

Conventions :

- Le joueur 1 est un joueur humain (c'est vous !!!)
- Le joueur 2 est une IA
- Un tableau 2D dénommé « Cases » stocke le contenu de chaque case
- Le code 0 signifie que la case est vide
- Lorsque le joueur 1 choisit la case (x,y), la valeur Cases[x][y] passe à 1
- Lorsque le joueur 2 choisit la case (x,y), la valeur Cases[x][y] passe à 2
- La partie est nulle et terminée si toutes les cases sont remplies et qu'il n'y a aucune configuration gagnante.
- Si un joueur remporte la partie, son score augmente de 1

Consignes : choisissez des noms de variables clairs et créez une fonction par tâche / traitement / opération…

Etape 1 - Perdu Gagné Fini

Modifiez le code du jeu pour qu'après le placement d'un jeton par le joueur humain, on détecte si une configuration gagnante est apparue. Dans ce cas, modifiez le score et relancez la partie. A ce niveau, il n'y a qu'un joueur dans la partie, ce n'est donc pas très dur de gagner ☺

Si la case nouvellement occupée n'amène pas vers une configuration gagnante, vérifiez qu'il reste des emplacements disponibles. Dans le cas contraire, relancez une nouvelle partie car nous avons une partie nulle.

Ecrivez une fonction qui retourne la liste des cases disponibles. Programmez un joueur 2 artificiel qui pour l'instant ne fait que choisir une position au hasard parmi les cases disponibles.

Comme pour le joueur 1, une fois que l'IA a joué, vérifiez si le joueur 2 a gagné et s'il reste encore des places disponibles.

Une fois que le joueur 2 a placé son marqueur et que les tests sont effectués le programme n'effectue plus d'actions. En effet, c'est au joueur humain de jouer et son tour de jeu est initié par le clic de la souris. C'est donc la fonction ClicSouris appelée par le système qui va déclencher cette séquence :

- Placement du pion du joueur humain sur la grille à la position du clic souris
- Détection partie gagnée / partie finie => on réinitialise
- Placement du pion du joueur IA
- Détection partie perdue / partie finie => on réinitialise

Etape 2 - Algorithme du MiniMax

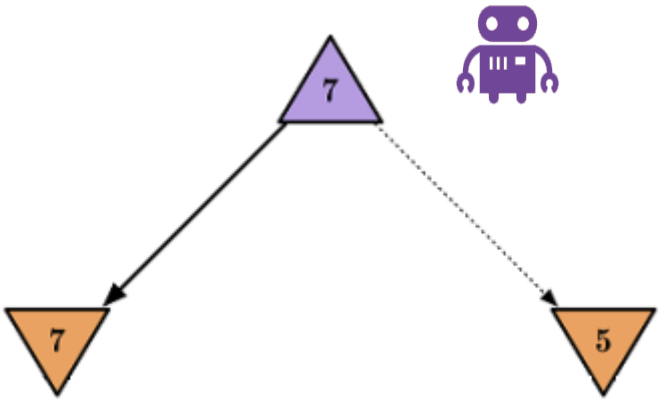
On se propose de mettre en place l'algorithme du MiniMax qui permet dans le cas du morpion de fournir un joueur IA excellent. S'il peut gagner, il le fera. Pour cela nous devons nous pencher sur la logique de jeu.

Cet algorithme du MiniMax se lance à chaque coup de jeu pour analyser la meilleure stratégie à développer. Il est sans mémoire, c'est-à-dire qu'il ne tient pas compte du style de jeu du joueur humain et des coups précédents. Lorsqu'il est lancé, il a pour seules données : l'état de la grille et les coups pouvant être effectués. S'il sait où il peut jouer, cela sous-entend qu'il « connait » l'ensemble des règles du jeu.

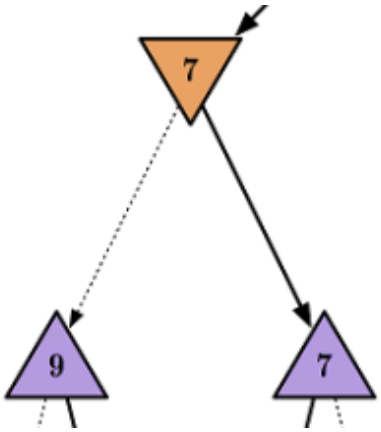
Prenons un cas pratique avec un jeu quelconque qui retourne un score en fin de partie. Le joueur Humain vient de jouer. L'IA est donc lancée pour déterminer quel coup jouer. Mais comment savoir quel coup est le meilleur ? Pas facile. Cependant, l'IA sait quels sont les coups possibles. Alors elle va « simuler » toutes les parties possibles en partant des coups possibles. Après cette « simulation », l'IA a déterminé que si elle choisit le premier coup, le meilleur score qu'elle puisse espérer est 7 en supposant que le joueur humain joue parfaitement à chaque fois. Elle a aussi déterminé que l'autre coup lui rapporterait au mieux un score de 5. Quel choix feriez-vous à sa place ? Vous allez choisir entre

les différents scénarios et prendre le plus avantageux. Ainsi, l'IA choisit de jouer le coup qui lui rapporte potentiellement un maximum de points.

Est-ce que la partie retenue comme meilleure partie possible pour l'IA va-t-elle se produire ? Non pas forcément. L'humain peut jouer maladroitement. Dans ce cas, l'IA pourra espérer plus que 7 points !



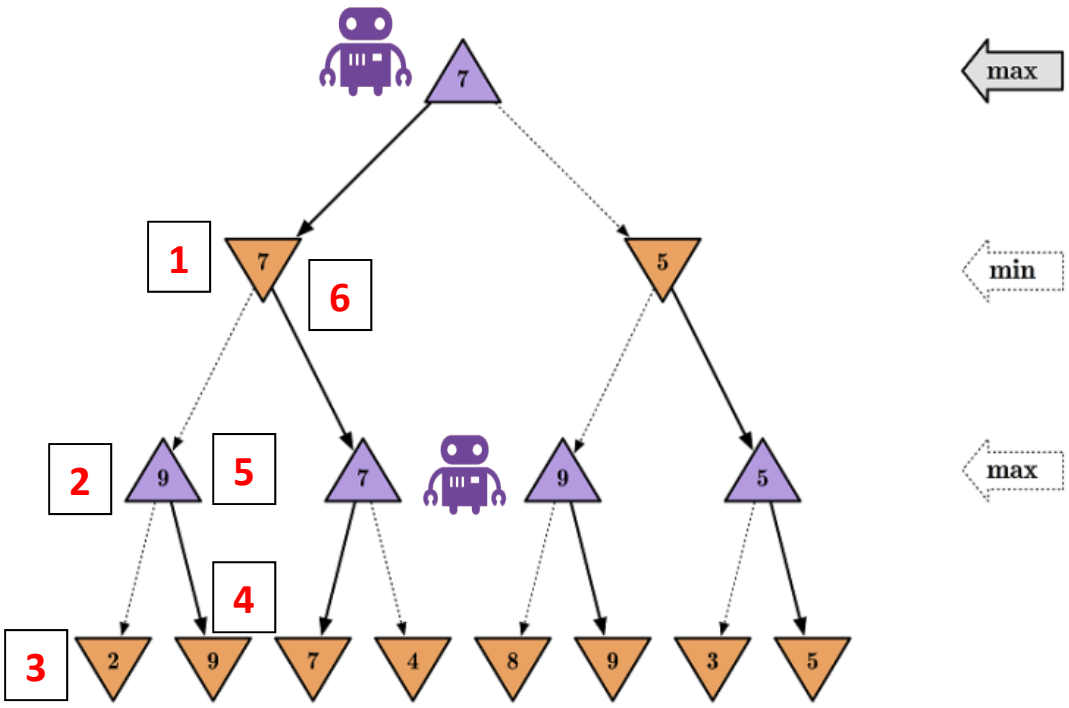
Nous avons expliqué comment l'IA faisait son choix, mais comment a-t-elle fait cette mystérieuse simulation qui amène à la connaissances des meilleurs scores de partie ? En fait, l'IA va aussi jouer pour le joueur humain dans cette simulation. L'idée est que l'on suppose que le joueur humain est parfait et que lui aussi va joueur en prenant la meilleure stratégie, c'est-à-dire celle qui lui apporte le plus de points. Durant la simulation, lorsque c'est au tour de l'IA de simuler le coup du joueur humain, elle va déterminer les différents coups possibles, puis simuler les parties et récupérer les scores associés. A cet instant, l'IA se met à la place du joeur humain. Les scores indiquées sont toujours les scores obtenus en fin de partie pour l'IA. Par exemple, nous avons deux choix possibles, un qui amène le score de 9 pour le joueur IA en fin de partie et l'autre le score de 7. Comme nous simulons le jeu du joueur humain, l'IA va donc tout simplement choisir l'option qui apporte le score le plus défavorable pour le joueur IA. La stratégie consiste donc à choisir le scénario qui rapporte un minimum de points.



Comme vous le voyez à chaque étape, le jeu change de main, une fois c'est l'humain qui est supposé jouer, une fois c'est le joueur IA. Ainsi, à chaque étage , on alterne recherche du score min et recherche du score max. Oui, il s'agit ici d'un algorithme récursif, c'est-à-dire qui a la capacité de s'appeler lui-même pour résoudre le problème.

Comment connaissons nous les scores ? En fait, l'algorithme consiste d'abord à explorer les parties en analysant l'ensemble des coups possibles et en simulant les parties associées. A force de placer des pions, on arrive à un état final : perdu / gagné / nul. Cet état final est associé à un score et nous retournons ce score à la fonction appelante. Par convention, nous rappelons que nous retournons le score associé au joueur IA. Une fois que toutes les parties possibles à partie d'un état du jeu ont été testées, alors, leur scores sont connus et il suffit suivant que l'on simule le joueur humain ou le joueur IA de retourner le min ou le max des scores.

Prenons un exemple où toutes les parties s'achèvent au bout de trois coups (c'est uniquement car cela simplifie le schéma ci-dessous). L'IA démarre son cheminement en haut du schéma. Elle détermine alors que deux coups sont possibles. Elle commence par joueur le premier. Nous sommes ici au niveau du marqueur 1. L'IA va ensuite simuler les parties possibles après avoir joué ce coup. Pour cela, elle détermine les coups possibles et en trouve 2. Elle choisit le premier coup pour continuer son exploration et nous passons au marqueur 2. L'IA simule alors les deux coups possibles (marqueurs 3 et 4). Comme ces coups sont terminaux, ils sont associés à un score : 2 pour le premier et 9 pour le second. Au niveau de la position 5, c'était au tour du joueur IA de jouer. L'IA va donc retenir le score maximal c'est-à-dire 9 et le retourner comme résultat à la fonction appelante (marqueur 6).



Voici la logique :

JoueurIA :
L = liste des coups possibles
Si la partie est finie alors retourner le score

```
Scores = [ ]
Pour chaque coup K dans L
        Joueur le coup K
        Score = JoueurHumainSimule()
        Stocker Score dans Scores
        Annuler le coup K (je retire mon pion)

Retourner Max(Scores) avec le coup associé
```

```
JoueurHumainSimule :
        L = liste des coups possibles
        Si la partie est finie alors retourner le score
        Scores = [ ]
        Pour chaque coup K dans L
                Joueur le coup K
                Score = JoueurIA()
                Stocker Score dans Scores
                Annuler le coup K (je retire mon pion)

        Retourner Min(Scores) avec le coup associé
```

Ces deux fonctions peuvent n'en faire qu'une, mais ce sera un peu plus difficile à mettre en œuvre, à fusionner dans un deuxième temps.

CalculScore :

- Le joueur IA a gagné => retourner 1
- Le joueur IA a perdu => retourner -1
- Match nul : retourner 0

Et pour jouer :

```
CoupAJouer = JoueurIA()
Cases[CoupAJouer] = 1
```

Puissance 4

Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la dite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

Nous vous conseillons à partir de cette étape de dupliquer le source du jeu du morpion car nous allons nous en servir pour créer le jeu du puissance 4.

Etape 3 - Jeu minimal

Modifier le jeu du morpion pour gérer une grille de 6x7.

Modifier l'affichage pour que le joueur 1 dispose des jetons jaunes et le joueur 2 des jetons rouges.

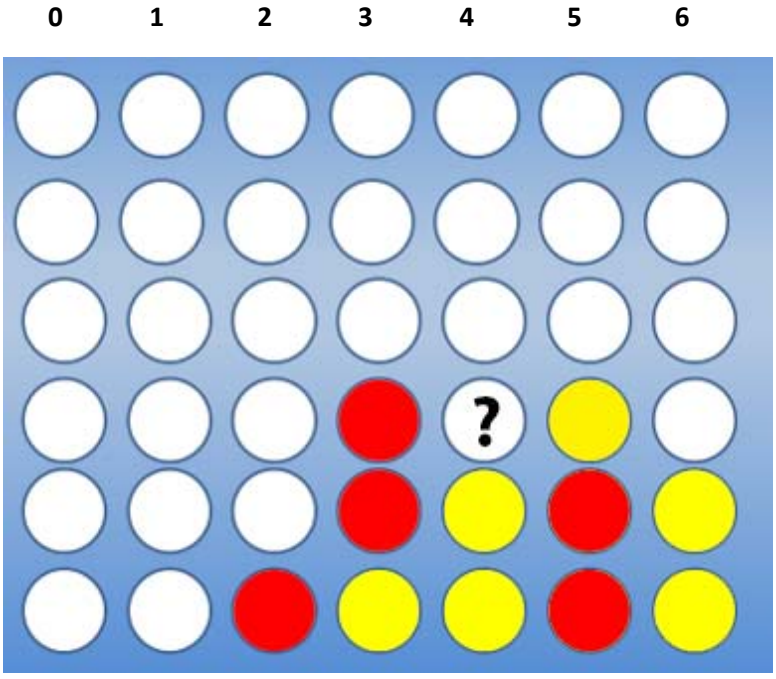
Mettre en place une fonction qui détecte si un alignement de 4 pions du même joueur existe. Le numéro du joueur sera un paramètre de la fonction. Les alignements peuvent comme dans le cas du morpion être horizontaux, verticaux ou diagonaux. Cette fonction n'est pas forcément simple à mettre en œuvre.

Mettre à jour la fonction qui détermine les coups possibles. Nous vous proposons de coder un coup possible par son numéro de colonne. Ainsi en début de partie la liste des coups possibles sera égale à [0, 1, 2, 3, 4, 5, 6]. Dès qu'une colonne est pleine, son numéro va disparaître de la liste des coups possibles.

Nous sommes donc à cette étape capable de mettre en place un jeu contre une IA. Nous pouvons tout simplement reprendre l'IA qui consiste à choisir un coup au hasard parmi les coups possibles. Cela vous permet de tester le bon fonctionnement du jeu et surtout de valider la détection des coups gagnants.

Etape 4 - Placements judicieux

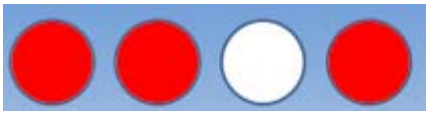
Nous allons essayer de fournir une IA plus perspicace qu'un simple choix aléatoire. Pour cela, nous vous proposons d'associer un score suivant les opportunités fournies par un placement. Par exemple, lorsque nous avons la grille suivante et que le joueur rouge doit jouer, nous aurions plus tendance à placer notre pion sur la colonne 4 car il permet de construire une ligne diagonale de trois pions rouges. Mais finalement, jouer sur la colonne 5 semble tout aussi intéressant car on place trois des quatre pions rouges nécessaires à un alignement. Joueur sur la colonne 6 ne semble pas fournir d'intérêt pour former un alignement rouge, par contre, il permet de bloquer un début d'alignement jaune composées de 2 pions pour l'instant.



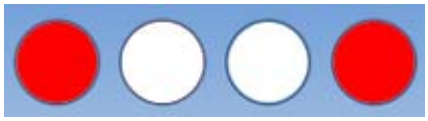
Jouer sur la colonne 6 s'avère cependant un mauvais choix, car cela permet au joueur jaune de créer un alignement en jouant sur la colonne 6. Nous ne prendrons pas en compte cela pour l'instant. Nous allons juste considérer les avantages/désavantages produits immédiatement par le placement d'un pion. Nous vous proposons les scores suivants, vous pouvez les faire évoluer à votre guise.

- Si le placement permet de faire un alignement de 4 => 100 points
- Si le placement bloque un alignement de 3 pions du joueur adverse => 50 points
- Si le placement permet de faire un alignement de 3 sur 4 => 30 points
- Si le placement bloque un alignement de 2 sur 4 pions du joueur adverse => 15 points
- Si le placement permet de faire un alignement de 2 sur 4 => 10 points

Attention, on considère uniquement le score max, on ne cumule pas les différents scores. Car on ne veut pas qu'un placement qui permette de bloquer 2 alignements de 3 pions du joueur adversaire passe devant le score d'un alignement possible de 4 pions pour le joueur courant.

 Le placement disponible obtient un score de 100 points

Mais dans la même logique !

 Les deux placements disponibles obtiennent un score de 30

Car ce qui est important c'est le nombre de pions présent sur les 4 emplacements. Le fait qu'ils ne soient pas contigus n'est pas une raison pour ne pas comptabiliser de score.

Travail à effectuer : réutiliser la fonction qui détermine les colonnes où il est possible de joueur. Pour chaque colonne disponible, trouver l'emplacement correspondant et calculez ses points associés. Pour l'ensemble des colonnes, trouver celle associée au placement de meilleur score. Faîtes évoluer votre IA pour qu'elle place son pion sur la colonne retenue.

A ce niveau, si vous voulez vous amuser, vous pouvez lancer un challenge entre l'IA qui joue au hasard et l'IA qui cherche un emplacement judicieux. Sur 10 parties, quel est le ratio de réussite ?

Etape 5 - Minimax

Nous avons vu tout à l'heure que l'algorithme du minimax permet de trouver la meilleure solution. En effet, il simule TOUTES les parties pouvant être jouées à partir d'une grille de jeu et suivant une séquence de min/max alternée détermine la meilleure option. Ainsi, au début de la partie de morpion, la simulation a 9 choix pour poser son premier jeton. Une fois posée, elle en a 8 pour le joueur suivant, puis 7 encore pour le joueur d'après. Cela nous amène à un total de 9! simulations soit 9x8x7x6x5x4x3x2 = 362 880 parties à analyser. Sur un ordinateur récent avec une vitesse de l'ordre du GigaHertz, cette exploration est possible dans un temps acceptable.

Cependant, dans le cas du puissance 4, le ton change. En effet, il y a 42 tours de jeu possibles et un nombre de parties différentes évaluées à environ 4 531 milliards. Nous avons donc un jeu 10 millions de fois plus complexes que le morpion ! L'algorithme développé pour le morpion ne pourrait pas trouver la solution dans un temps acceptable.

Nous allons cependant appliquer l'approche du minimax mais en restreignant la profondeur d'exploration. Pour cela, nous allons simuler toutes les parties à partir de la grille actuelle en jouant au maximum 2 coups. Certes, la partie ne sera pas forcément finie et nous n'aurons donc pas le score associé à perdu/gagné/nul. Mais si l'on examine l'algorithme minimax, tout ce qu'il requiert pour prendre une décision est finalement un score ou une sorte de « note ». On peut donc comme d'habitude si la partie se termine retourner le score associé mais si la partie n'est pas finie on peut se contenter de retourner une note qui caractérise la grille de jeu. Plus cette note sera haute, plus elle sera à l'avantage du joueur IA et plus elle sera basse, plus elle sera à l'avantage du joueur humain. L'algorithme Minimax va donc prendre une décision de jeu qu'il sait nous amener vers une configuration de jeu de « bonne » qualité.

Nous devons donc créer une fonction retournant une évaluation de la grille actuelle. Si la partie n'est pas gagnante/perdante/nulle, c'est qu'il n'y a aucun alignement de 4 pions. Nous vous proposons de de réutiliser la fonction qui note les placements pour construire cette nouvelle fonction. Ainsi nous aurons :

```
Note(Grille)

    Si IA est gagnante retourner 500
    SI Humain est gagnant retourner -500
    LIA = []   # Liste des notes des emplacements pour le joueur IA
    LH  = []   # Liste des notes des emplacements pour le joueur Humain
    Pour chaque colonne où l'on peut joueur
            Calculer la position de l'emplacement libre
            Calculer le score de l'emplacement pour le joueur IA => LIA
            Calculer le score de l'emplacement pour le joueur Humain => LH
    Retourner Max(LIA) – MAX(LH)
```

Déterminez le maximum de coups que l'on peut simuler pour avoir une réponse de l'IA qui prend moins d'une seconde ?

Mettez en place un challenge entre l'IA qui examine uniquement les placements judicieux (question précédente) et l'IA avec un minimax à profondeur limitée. Y-a-t-il un réel gain sur la stratégie ?