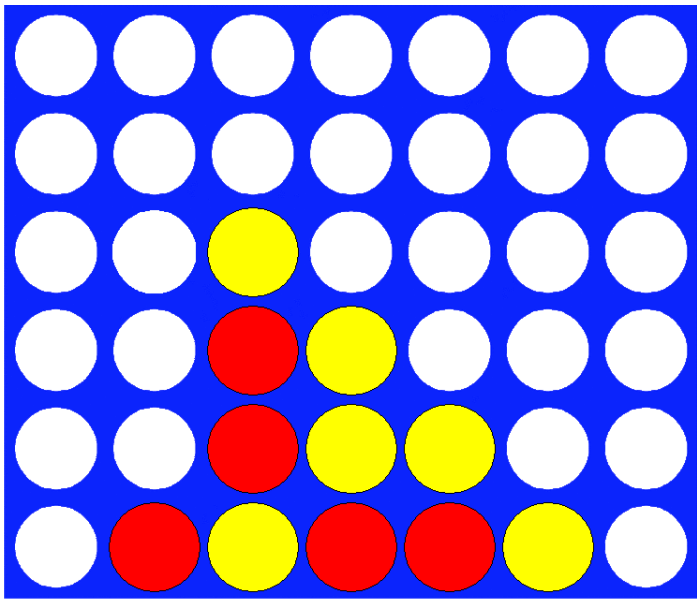
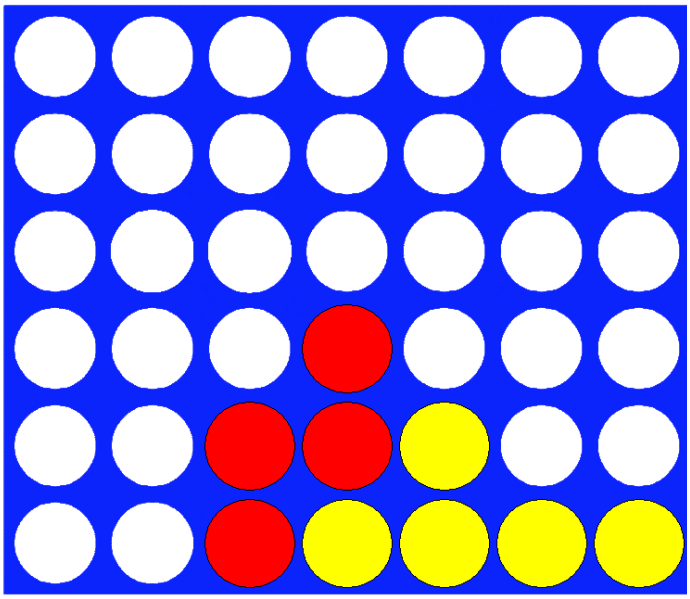
Jaillet Adrien
binôme : Vachey Jérémy

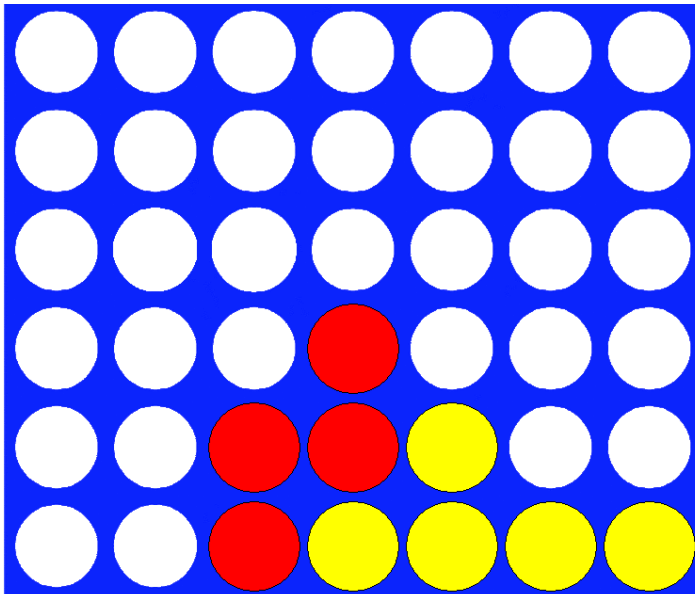
Dossier ISN :

Le principe de ce jeu est d'aligner 4 jetons de même couleur afin de gagner. Les joueurs peuvent aligner les pions horizontalement, verticalement ou diagonalement.

Chacun des deux joueurs a donc des pions de couleurs différents et joue chacun leur tour afin d'aligner 4 de ses pions.





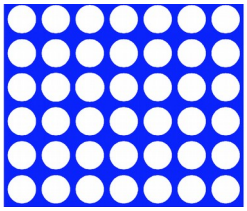


Chacune des 3 combinaisons ci-dessus et ci-contre sont donc gagnantes pour le joueur possédant les pions jaunes.

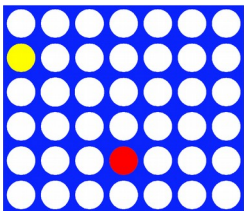
Le fonctionnement du jeu :

Le plateau est simulé par des listes imbriquées dans une grande liste. La grande liste est donc le plateau et les sous-listes correspondent à chacune des colonnes.
Un 0 dans cette liste correspond a une case vide, un 1 à un jeton jaune, et un 2 à un jeton rouge.
Chaque sous liste à un nom, de c1 pour la colonne 1 à c7 pour la septième colonne.

Plateau = [[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],
[0,0,0,0,0,0],[0,0,0,0,0,0]]
Ce plateau se caractérise par la grille vide :



Plateau = [[0,0,0,0,1,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,2,0,0,0,0],[0,0,0,0,0,0],
[0,0,0,0,0,0],[0,0,0,0,0,0]]
Ce plateau a un jeton jaune dans la première colonne à la cinquième position et un jeton rouge dans la quatrième colonne à la deuxième position.



Il y a deux fonctions servant à remplir la liste : Une pour mettre un jeton rouge, et une pour mettre un jeton jaune.

```
10
11  def jeton_jaune(liste):
12      """L'entrée est une liste.
13      La sortie est une liste.
14      La fonction retourne la liste entrée avec un 1 ajouter a la fin, chiffre simulant un jeton jaune(si la longueur de cette liste est plus petite que
    6).
15      Ex dans une sous liste : jetonjaune([1,2,2,1]) donne [1,2,2,1,1]
16      """
17      for s in range (0,6):
18          if liste[s] ==0:
19              liste[s] = 1
20              break
21      return liste
22
23
24
25  def jeton_rouge(liste):
26      """L'entrée est une liste.
27      La sortie est une liste.
28      La fonction retourne la liste entrée avec un 2 ajouter a la fin, chiffre simulant un jeton rouge(si la longueur de cette liste est plus petite que
    6).
29      Ex dans une sous liste : jetonjaune([1,2,2,1]) donne [1,2,2,1,2]
30      """
31      for s in range (0,6):
32          if liste[s] ==0:
33              liste[s] = 2
34              break
35      return liste
```

Dans une vraie grille, les pions tombent un à un en bas de la grille, puis si il y a déjà un pion tout en bas, ils tombent sur le pions déjà existant.
Lorsque le joueur choisit une colonne, la fonction s'effectue pour la colonne choisit, donc la sous liste correspondante. La fonction vérifie si le premier terme de la sous liste vaut 0 (cas où il n'y a pas de pion), si c'est le cas, il met un 1 où un 2 suivant le joueur, si ce n'est pas le cas, la fonction effectue la même opération avec le deuxième terme de la liste, et ainsi de suite.

Maintenant que nous savons remplir le plateau, il faut trouver les conditions de victoire afin d'arrêter le jeu lorsqu'un des deux joueurs a gagné.
Il y a trois fonctions dans le code servant à trouver les conditions de victoire :

-La première pour trouver un alignement de 4 jetons dans une même colonne.

```
def verif_colonne(plateau):
    """L'entrée est une liste.
    La sortie est une variable booléenne.
    La fonction vérifie dansdans toutes les sous listes indépendament si 4 jetons (le dernier compris) a coté sont les mêmes.
    Ex dans une sous liste : verif_colonne([1,2,1,1,1,1]) retourne true, verif_colonne([1,2,2,2,1,1]) rertourne false.
    """
    for i in range (0,7):
        for k in range (0,3):
            if plateau[i][k]==plateau[i][k+1]==plateau[i][k+2]==plateau[i][k+3]!=0:
                return True
    else:
        return False
```

-La deuxième pour trouver un alignement de 4 jetons dans une même ligne.

```
def verif_ligne(plateau):
    """L'entrée est une liste.
    La sortie est une variable booléenne.
    La fonction vérifie dans la liste totale, 4 élément de sous liste du même rang (le dernier compris) sont les mêmes.
    Ex verif_ligne(2, [[1,2,1,1,2],[2,1,1,0,0,0],[2,2,1,1,1,0],[1,2,1,1,2,0],[2,2,2,0,0,0],[2,2,1,1,1,0],[0,0,0,0,0,0]]) donne true (les terme de rang 2 des 4 premières listes sont des 1, il y a donc victoire.
    """
    for i in range (0,6):
        for k in range (0,4):
            if plateau[k][i]==plateau[k+1][i]==plateau[k+2][i]==plateau[k+3][i]!=0:
                return True
    else:
        return False
```

-La troisième pour trouver un alignement de 4 jetons en diagonale.

```
def verif_diag(plateau):
    """L'entrée est une liste.
    La sortie est une variable booléenne.élé
    La fonction vérifie dans la liste totale, si 4 éléments de la liste ayantpour sous liste 1 liste d'écarts entre eux et que leur terme dans la sous liste diffère d'1 terme entre eux.
    Ex verif_ligne(1, [[1,0,0,0,0,0],[2,1,1,0,0,0],[2,2,1,1,1,0],[1,1,1,1,2,0],[2,2,2,0,0,0],[2,2,1,1,1,0],[0,0,0,0,0,0]]) donne true.
    """
    for i in range (0,3):
        for k in range (0,4):
            if plateau[k][i]==plateau[k+1][i+1]==plateau[k+2][i+2]==plateau[k+3][i+3]!=0:
                return True
        for k in range (3,7):
            if plateau[k][i]==plateau[k-1][i+1]==plateau[k-2][i+2]==plateau[k-3][i+3]!=0:
                return True

    else:
        return False
```

Ces trois fonctions fonctionnent de la même manière, elles testent toutes le positions où il pourrait avoir 4 jetons alignés (horizontalement, verticalement ou diagonalement selon les fonctions). Ainsi après chaque tour un appel à ces fonctions permet de savoir si quelqu'un à gagné et si il faut arrêter le jeu.

Ainsi l'agencement de ces fonctions permet simplement de jouer au puissance 4, un compteur s'incrémentant à chaque tour permet de savoir quelle joueur doit jouer (si le compteur est pair, c'est à un des deux joueurs, si il est impair, c'est à l'autre). On peut ainsi remplir la liste, et les 3 fonctions de vérifications permettent de savoir si l'un des deux joueur a gagné.

Nous nous sommes rapidement rendu compte que ce programme était assez basique, on s'est donc demandé si l'on ne pouvait pas coder une sorte « d'intelligence artificielle » afin de pouvoir jouer contre l'ordinateur.

L'intelligence artificielle :

Afin d'avoir une partie correct avec un minimum de difficulté, il fallait trouver des techniques afin que l'ordinateur ne joue pas n'importe comment.
Cette ordinateur joue donc en plusieurs temps:

-Premièrement: Il cherche à gagner sur le coup, où à empêcher de l'adversaire de gagner.

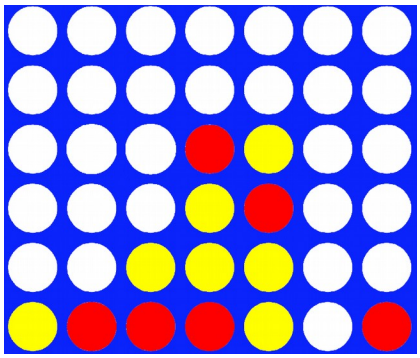
-Secondement: Si il n'a pas besoin d'accomplir la première action, il cherche à anticiper un minimum.

Premier temps :

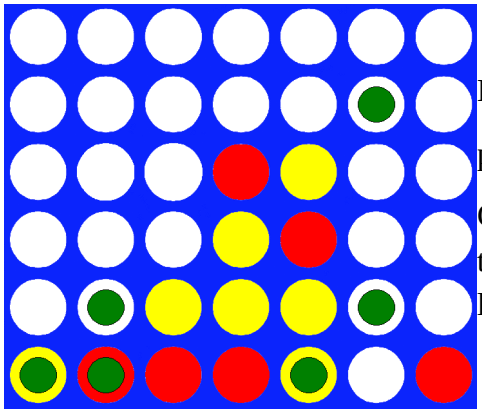
Cette partit est composé de deux fonctions que vous pourrez retrouver dans le document annexe, ce sont verif_bot_3 et possible3.

La première fonction fonctionne a peu près de la même manière que les fonctions de vérifications de conditions de victoire. Au lieu de chercher 4 jetons alignés sur le plateau, elle n'en cherche que trois et nous renvoie, sous forme d'une liste, les position où il faudrait mettre le quatrième jeton pour gagner en plus de sa couleur (rouge ou jaune représenter par un 1 ou un 2), même si il est impossible de jouer à cette endroit.

Prenons un exemple : avec notre liste plateau=[[1,0,0,0,0,0], [2,0,0,0,0,0],[2,1,0,0,0,0],[2,1,1,2,0,0],[1,1,2,1,0,0],[0,0,0,0,0,0], [2,0,0,0,0,0]] Nous avons la grille suivante :



Les positions qui peuvent faire gagner sont celles qui apparaissent avec un rond vert :

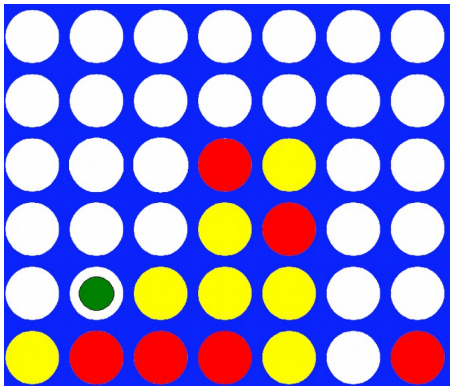


La fonction nous ressort donc la liste suivante :

position_3=[[0,0,2],[1,0,1],[4,0,2],[1,1,1],[5,1,1],[5,4,1]]

Chaque sous liste correspond à une position, les deux premiers termes de la sous liste indique respectivement l'abscisse et l'ordonnée du jeton, le troisième terme indique sa couleur.

Le deuxième fonction, possible3, enlève de la liste position_3 les endroits impossible à jouer, après le passage dans cette fonction, la liste position_3 contient donc seulement [[1,1,1]] car c'est le seul endroit jouable dans ce cas.



Ainsi le bot sait donc qu'il doit jouer dans la deuxième colonne.

Dans chaque partie, l'ordinateur a les pions rouges, dans le cas ci dessus, il aurait donc contrer son adversaire. Si dans certain cas, il peut gagner et contrer son adversaire dans le même tour, l'ordinateur jouerai en sorte de gagner.

Si à la sortie de ces deux fonctions, position_3 est une liste vide, c'est à dire que peut importe où l'on joue, on ne peut ni gagner, ni empêcher de gagner dans le tour, alors le bot va essayer

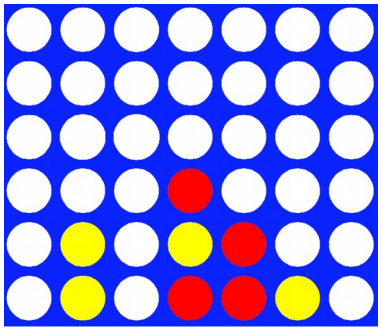
d’anticiper, de prendre de l’avance dans ses coups…

Deuxième temps :

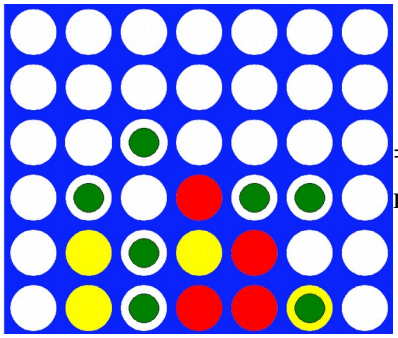
Afin de prendre de l’avance, il y a aussi deux fonctions, qui correspondent aux deux fonctions vus juste au dessus, sauf qu’elle s’applique lorsqu’il y a déjà 2 jetons d’alignés et qu ‘un joueur est susceptible d’en aligner un troisième. Ces fonctions sont verif_bot_2 et possible2.

Prenons un exemple :

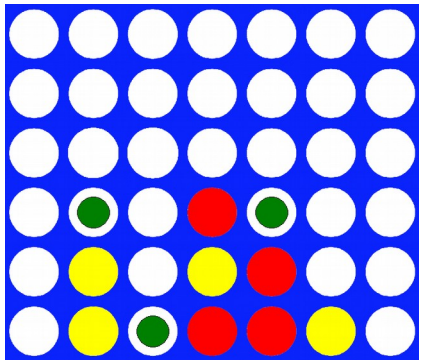
Avec plateau=[[0,0,0,0,0,0],[1,1,0,0,0,0],[0,0,0,0,0,0],[2,1,2,0,0,0], [2,2,0,0,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0]] On a la grille suivante :



Les position permettant d’aligné 3 jetons de la même couleur apparaissent en vert :



La fonction verif_bot_2 nous donne donc la liste : position_2 = [[1,2,1],[2,0,2],[2,1,1],[2,3,2],[4,2,2],[5,0,2],[5,2,2]] de la même manière que la fonction verif_bot_3 nous avais donné position_3.



La fonction possible2 enlève donc de position_2 toutes les positions où l’on ne peut pas jouer, elle nous ressort donc : position_2=[[1,2,1],[2,0,2], [4,2,2]]

Ici l’ordinateur a donc le choix entre jouer à la deuxième, troisième ou quatrième colonne. Ce choix sera fait aléatoirement.

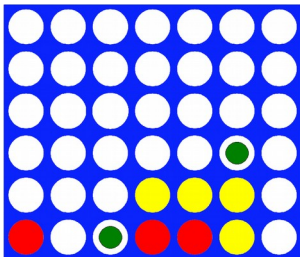
Si la liste position_ 2 est elle aussi vide, l’ordinateur jouera aléatoirement dans une des sept colonne de la grille.

En créant cette sorte d’anticipation, il y a quelque fois des réactions de l’ordinateur que l’on aimerais pas, c’est ce que je vais expliquer dans cette dernière partie :

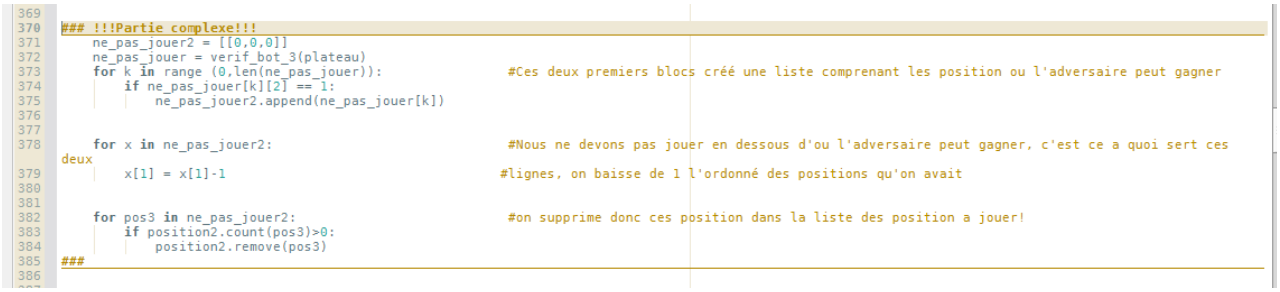
Nous allons prendre le cas le plus basique possible pour essayer d’expliquer ce qu’il se passe.

Avec plateau = [[2,0,0,0,0,0], [0,0,0,0,0,0], [0,0,0,0,0,0] ,[2,1,0,0,0,0],[2,1,0,0,0,0],[1,1,0,0,0,0], [0,0,0,0,0,0]]

Ici l’ordinateur joue avec les pions rouges, c’est donc à son tour de jouer. Après passage dans la première partie du bot, il n’y a nul part où le bot peut gagner où empêcher de gagner. C’est donc la deuxième partie du bot qui se met en route, et il y a 2 endroits où il pourrait jouer : position_ 2=[[2,0,2],[5,2,1]]



Mais si l'ordinateur joue dans la troisième colonne, il ouvre la voie à son adversaire pour gagner, il y a donc une partie de la fonction possible2 qui s'occupe de ces situations, c'est la partie que j'ai nommée « partie complexe ».



```
369
370 ### !!!Partie complexe!!!
371     ne_pas_jouer2 = [[0,0,0]]
372     ne_pas_jouer = verif_bot_3(plateau)
373     for k in range (0,len(ne_pas_jouer)):                #Ces deux premiers blocs créé une liste comprenant les position ou l'adversaire peut gagner
374         if ne_pas_jouer[k][2] == 1:
375             ne_pas_jouer2.append(ne_pas_jouer[k])
376
377
378     for x in ne_pas_jouer2:                              #Nous ne devons pas jouer en dessous d'ou l'adversaire peut gagner, c'est ce a quoi sert ces
    deux
379         x[1] = x[1]-1                                   #lignes, on baisse de 1 l'ordonné des positions qu'on avait
380
381
382     for pos3 in ne_pas_jouer2:                           #on supprime donc ces position dans la liste des position a jouer!
383         if position2.count(pos3)>0:
384             position2.remove(pos3)
385 ###
386
```

Cette partie utilise dans un premier temps la fonction de vérification pour savoir où l'on doit jouer pour aligner un quatrième jeton, et ne garde que les position de l'adversaire.

Comme nous l'avons dit ci dessus, position_ 2=[[2,0,2],[5,2,1]]

Ici la liste ne_pas_jouer2=[[2,1,1],[6,1,1]]

Pour empêcher l'adversaire de gagner, il ne faut pas que l'on joue en dessous d'où il peut gagner on baisse donc d'une case les positions de la liste ne_pas_jouer2, elle vaut donc : [[2,0,1],[6,1,1]].

Ces deux positions correspondent aux endroits qui feraient gagner l'adversaire si l'on y joue. La troisième boucle de cette partie sert donc à supprimer dans la liste position_ 2 qui est la liste des position à jouer, les position qui ferait gagner l'adversaire. Nous avons donc à la fin position_ 2 = [[5,2,1]]. Le bot ne fera donc pas gagner l'adversaire.

Pour conclure :

Comme vous avez pu le constater tout au long de ce compte rendu, ma partie était dédiée à la conception de « l'intelligence artificielle » afin de pouvoir jouer contre l'ordinateur. Cela m'a pris pas mal de temps, en effet à chaque fois que je rajoutais des fonctionnalités à l'ordinateur, de nouveaux problèmes se soulevaient. Cet ordinateur n'est pas imbattable, et je n'ai pas réussit à le rendre imbattable car le programme entier commence à devenir complexe, et lorsque que je teste l'ordinateur je ne comprends pas tous le temps pourquoi il joue d'une certaine manière. Mais au final je trouve quand même qu'il n'est pas si évident que ça à battre, c'est pourquoi je suis plutôt satisfait de mon travail. De plus la partie graphique gérer par mon binôme est quand même assez complexe, c'est très agréable de jouer. C'est pourquoi nous avons passé pas mal de temps consacré à ce projet, mais étant donné que cela nous plaisait, c'était loin d'être du temps perdu !