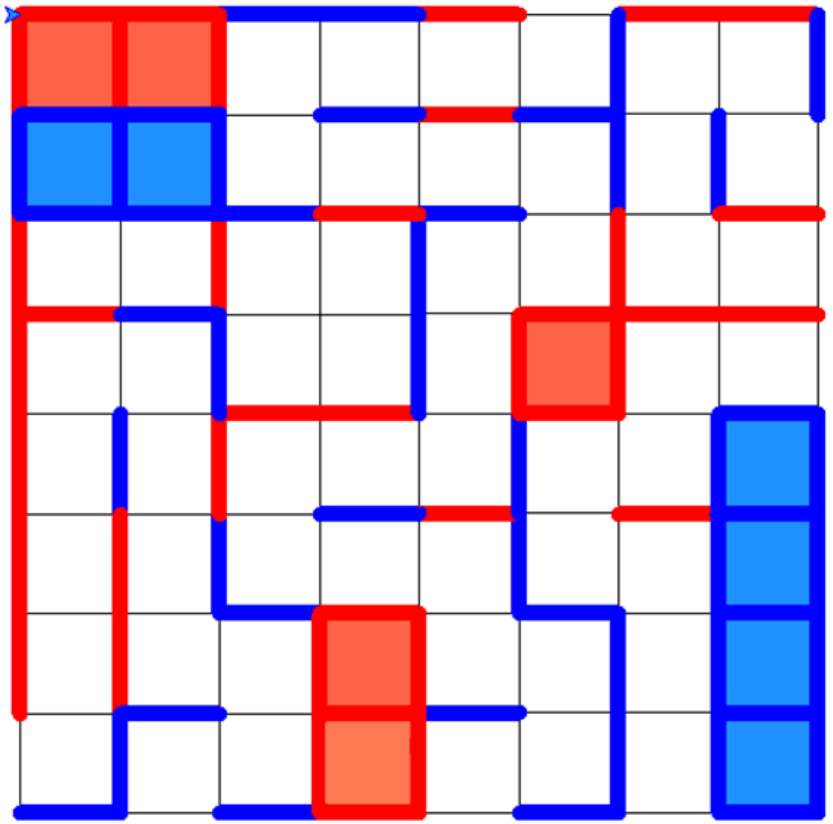
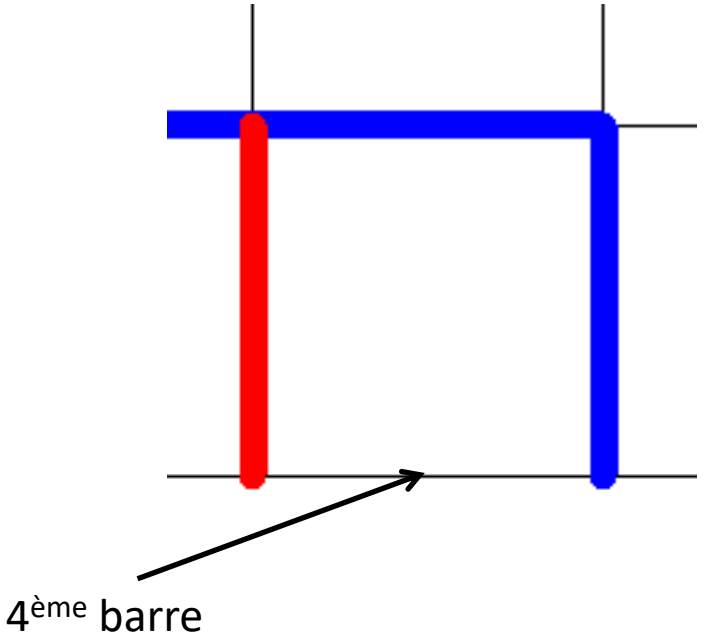
Présentation du projet informatique:

Pipopipette ou jeu des petits carrés



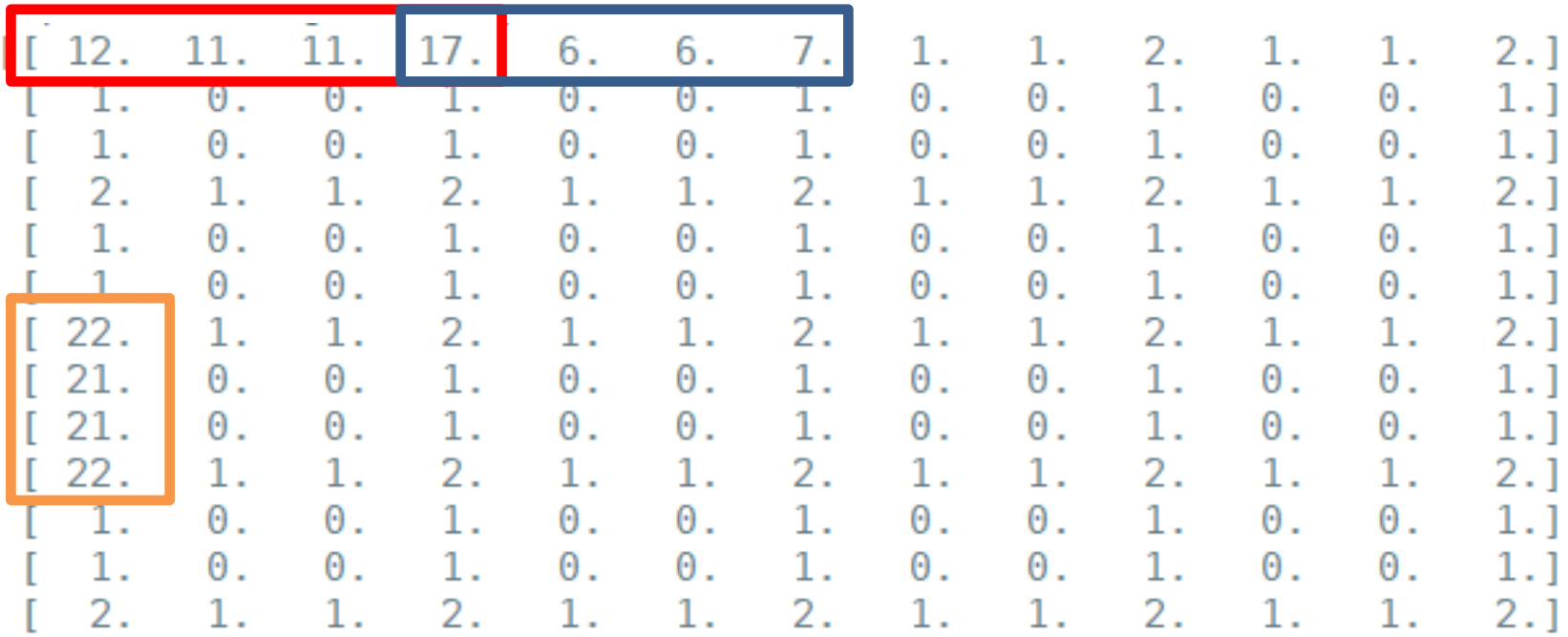
Principe du jeu

1. Chaque joueur pose une barre sur un quadrillage
2. Poser une quatrième barre permet de remporter un point et de rejouer
3. Le jeu est fini lorsque le plateau est rempli
4. Celui qui a le plus de points gagne



La combinaison de nombreuses stratégies ainsi que l'anticipation des coups à venir permettent de construire un jeu à son avantage

Interagir avec le plateau de jeu



Barre posée du joueur 1

Barre posée du joueur 2

Curseur (barre en cours de jeu, non fixée)

Codage pour le jeu du joueur

```
def deplacement_droite(curseur,cur_c,C):
    if cur_c+len(curseur[0])<C:
        cur_c+=3
    return cur_c
```

```
def deplacement_haut(curseur,cur_l,L):
    if cur_l>0:
        cur_l-=3
    return cur_l
```

```
def rotation(curseur,c,cur_l,cur_c,L,C):
    if len(curseur)==1:
        if cur_l<L-2:
            curseur=[[c],[c],[c],[c]]
    else:
        if cur_c<C-2:
            curseur=[[c,c,c,c]]
    return curseur
```

```
while not fixe:
    choix=input('deplacement: gauche:(q)  droite:(d)  haut:(z)
bas:(s)  rotation:(a)  fixation:(f) =')
    if choix=='q':
        cur_c=deplacement_gauche(curseur,cur_c,C)
        conversion_graphique_curseur(cur_l,cur_c,curseur,Lt,Ct)
    elif choix=='d':
        cur_c=deplacement_droite(curseur,cur_c,C)
        conversion_graphique_curseur(cur_l,cur_c,curseur,Lt,Ct)
    elif choix=='z':
        cur_l=deplacement_haut(curseur,cur_l,L)
        conversion_graphique_curseur(cur_l,cur_c,curseur,Lt,Ct)
    elif choix=='s':
        cur_l=deplacement_bas(curseur,cur_l,L)
        conversion_graphique_curseur(cur_l,cur_c,curseur,Lt,Ct)
    elif choix=='a':
        curseur=rotation(curseur,c,cur_l,cur_c,L,C)
        conversion_graphique_curseur(cur_l,cur_c,curseur,Lt,Ct)
    elif choix=='f':
        fixe=controle_pose_barre(plateau,curseur,cur_l,cur_c)
plateau=pose_barre(plateau,curseur,cur_l,cur_c,tour)
trace_barre(curseur,Lt,Ct)
plateau,nb,e,liste_carre=remplissage(plateau,L,C,tour)
trace_remplissage(liste_carre,c,Lt,Ct)
if nb==0:
    tour+=1
else:
    case+=nb
```

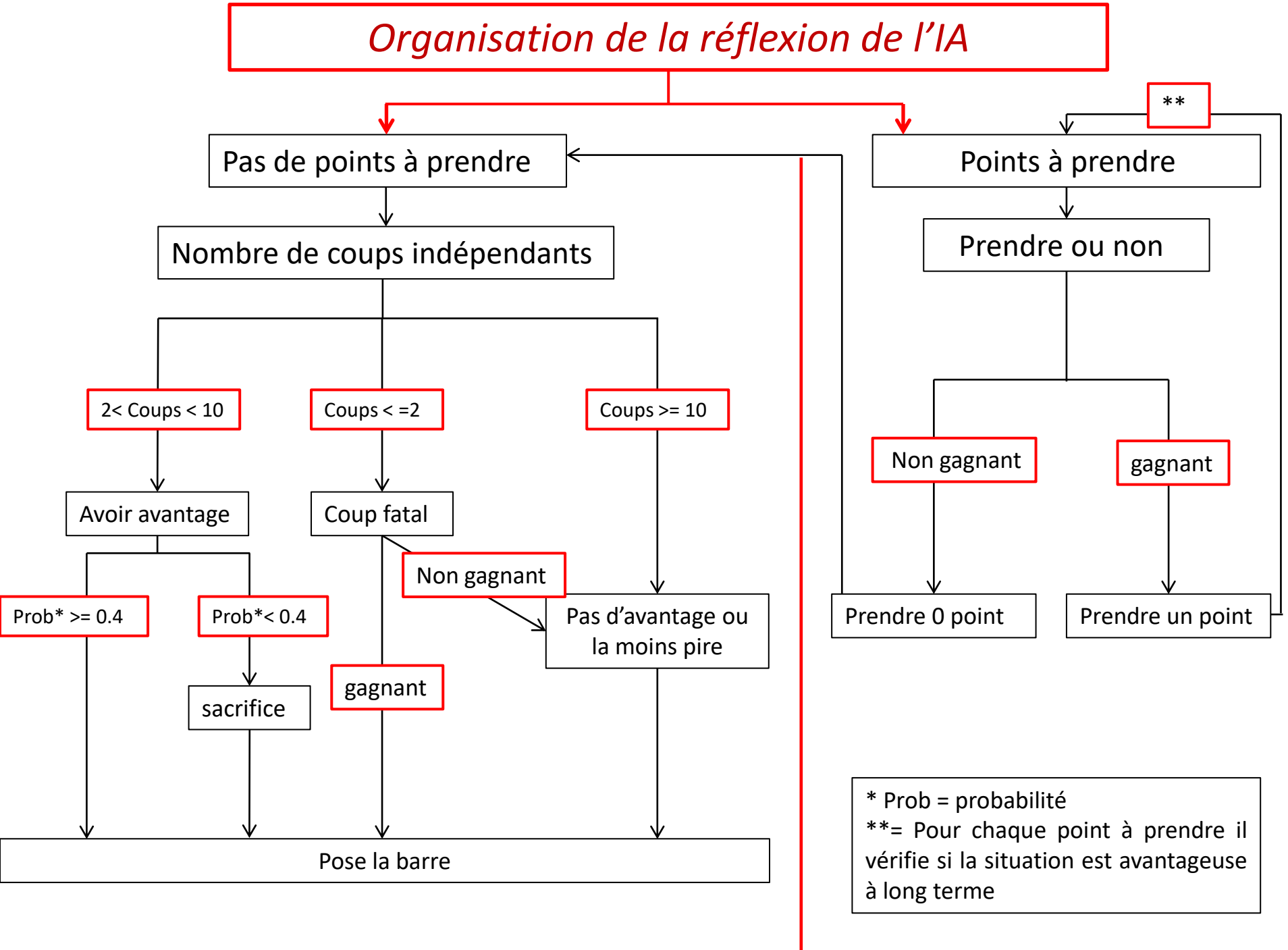
```python
def controle_pose_barre(plateau,curseur,cur_l,cur_c):
    if len(curseur)==1:
        if plateau[cur_l,cur_c+1]!=1:
            return False
        else:
            return True
    else:
        if plateau[cur_l+1,cur_c]!=1:
            return False
        else:
            return True
```

```python
def remplissage(plateau,L,C,tour):
    nombre_de_points=0# nombre de points gagnés
    liste_carre=[]#cette liste est renvoyée pour la convertion en graphique
    if tour%2==0: #les carrés du joueur 1 sont codés par des 10
        R=[[10,10,10,10],[10,10,10,10],[10,10,10,10],[10,10,10,10]]
    else:         #les carrés du joueur 2 sont codés par des 20
        R=[[20,20,20,20],[20,20,20,20],[20,20,20,20],[20,20,20,20]]
    for l in range (0,L-3,3):
        for c in range (0,C-3,3):
            cote=0
            if plateau[l+1,c+1]==0:
                if plateau[l,c+2]!=1:
                    cote+=1
                if plateau[l+2,c]!=1:
                    cote+=1
                if plateau[l+2,c+3]!=1:
                    cote+=1
                if plateau[l+3,c+1]!=1:
                    cote+=1
            if cote==4:
                nombre_de_points+=1
                plateau[l:l+4,c:c+4]=R
                liste_carre.append([l,c])
    return plateau,nombre_de_points,cote,liste_carre
```

```
[[ 12.  11.  11.  12.   1.   1.   2.]
 [  1.   0.   0.   1.   0.   0.   1.]
 [  1.   0.   0.   1.   0.   0.   1.]
 [  2.   1.   1.   2.   1.   1.   2.]
 [  1.   0.   0.   1.   0.   0.   1.]
 [  1.   0.   0.   1.   0.   0.   1.]
 [  2.   1.   1.  22.  21.  21.  22.]]
```

Contrôle de « remplissage »

Contrôle de « contrôle_pose_barre »

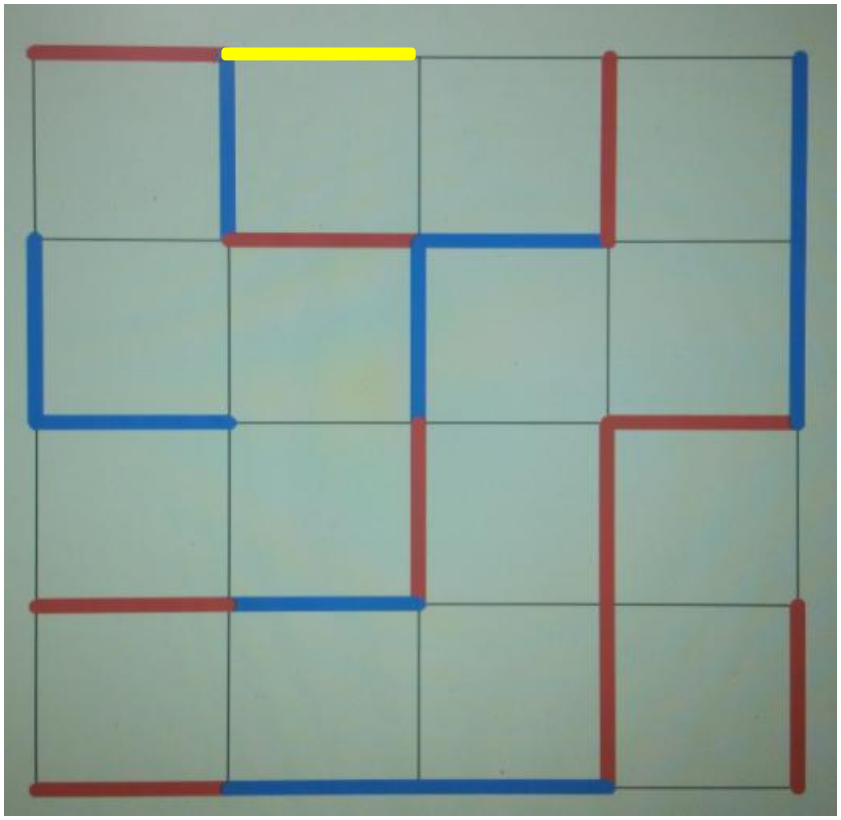

Organisation de la réflexion de l'IA
Pas de points à prendre
Nombre de coups indépendants
2< Coups < 10
Coups < =2
Coups >= 10
Avoir avantage
Coup fatal
Non gagnant
Prob* >= 0.4
Prob*< 0.4
Pas d'avantage ou la moins pire
sacrifice
gagnant
Pose la barre
**
Points à prendre
Prendre ou non
Non gagnant
gagnant
Prendre 0 point
Prendre un point
* Prob = probabilité
**= Pour chaque point à prendre il vérifie si la situation est avantageuse à long terme

Présentation de la fonction « Prendre ou non »

Choix de présentation: apporte une dimension stratégique des plus importantes dans le jeu. Le choix de sacrifier et de prendre ou non un sacrifice est cruciale dans ce jeu de plateau.

Objectif: identifie si la prise d'un point est avantageuse :

- en simulant les points que pourra remporter l'adversaire aux tours suivants
- en identifiant les liens entre les différentes cellules de points possibles



Etudions cette situation type d'une fin de partie ou les choix de jeu sont les plus importants

Dernière barre posée

```
def prendre_ou_non(plateau,L,C,coups,tour):
    """ vérifie si la prise de points ne va pas lui être fatal"""
    k=0
    gagnant=True
    points=[]
    copie1_plateau=deepcopy(plateau)
    copie1_plateau,case1,remplie,choix=poser_les_quatrieme_barre(copie1_plateau,L,C,tour)
    if coups<7 and remplie==True:
        # verifie que s'il reste un nombre de coups<7 car sinon il y a aucun risque que ca lui
soit fatal
        place_vide_possible1=place_vide(copie1_plateau,L,C)
        for elt in place_vide_possible1:
            copie2_plateau=deepcopy(copie1_plateau)
            copie2_plateau=strategie_pas_poser_troisieme_barre([elt],copie2_plateau,L,C,tour)
            place_vide_possible2=place_vide(copie2_plateau,L,C)
            copie2_plateau,case,remplie,choix=poser_les_quatrieme_barre(copie2_plateau,L,C,tour)
            points.append(case)
        while k!=len(points) and gagnant==True:
            if points[k]>case1:
                # si il existe une possibilité pour que l'adversaire remporte un plus grand
nombre de points au prochain tour il ne prend pas les points pour le piéger
                gagnant=False
            k+=1
        if not gagnant:
            # cependant il regarde le nombre de cellules de points indépendantes, et juge si la
prise de points à long terme ne lui sera pas profitable
            copie1_plateau=deepcopy(plateau)
            nombre_cellule=nombre_de_cellule(copie1_plateau,L,C,tour)
            print(nombre_cellule, "nombre cellule")
            if nombre_cellule%2==0:
                gagnant=True
                print(gagnant)
    if gagnant:
        return quatrieme_barre(plateau,L,C)
    else:
        return []
```

Etape 1

Case1=2

Etape 2

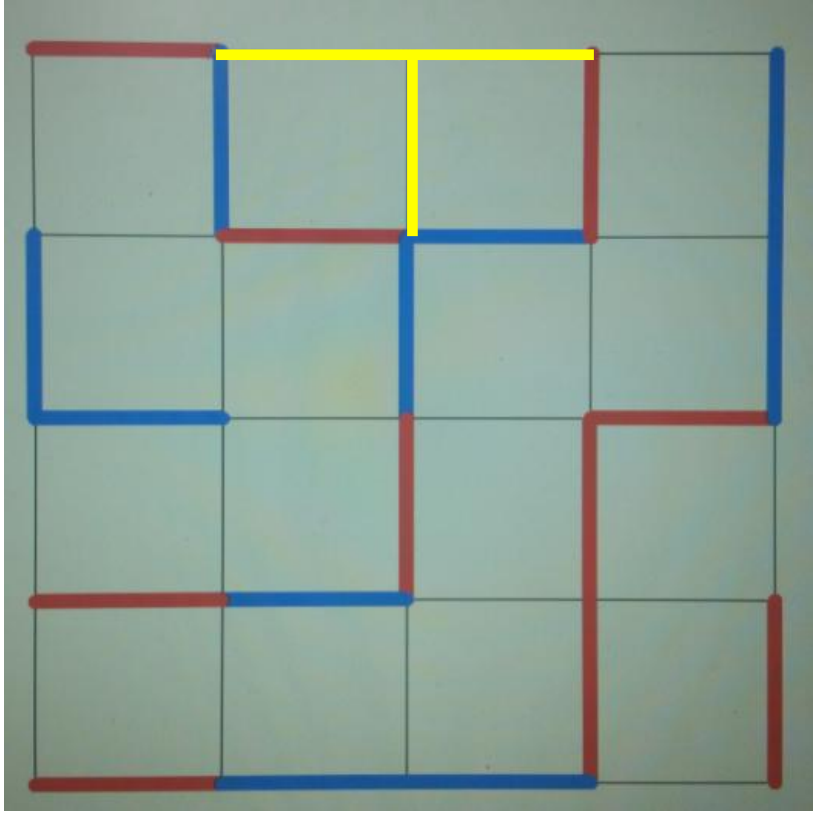
Points=[5]

choix=
[[positions des barres des cellules ¡],[…]]

Etape 3

nombre_cellule=3

Etape 2



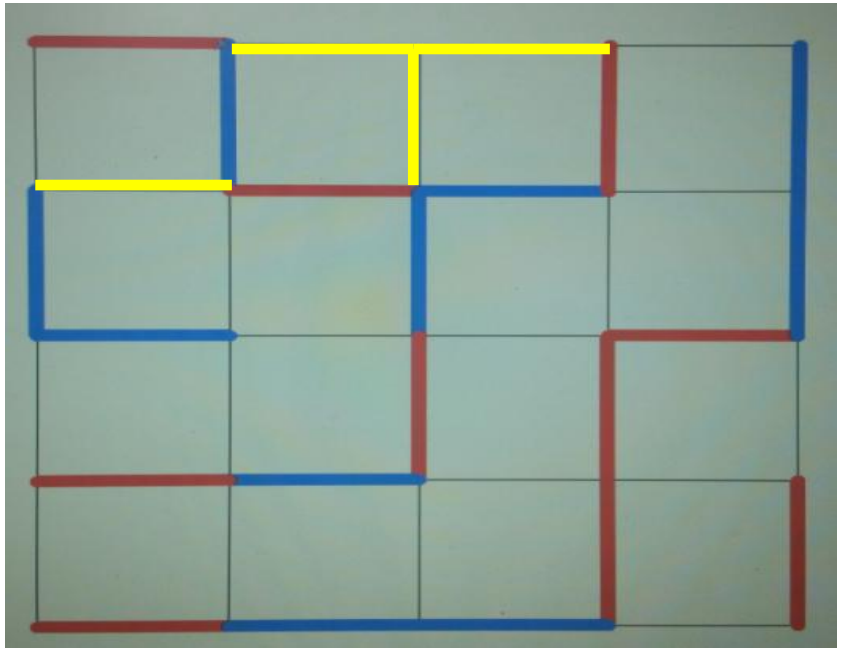
```
def poser_les_quatrieme_barre(plateau,L,C,tour):
    case=0
    remplie=False
    choix=[]
    matrice_coordonnees=quatrieme_barre(plateau,L,C)
    for elt in matrice_coordonnees:
        choix.append(elt)
    if matrice_coordonnees!=[]:
        while len(matrice_coordonnees)!=0 :
            plateau=pose_quatrieme_barre(matrice_coordonnees,tour,plateau,L,C)
            for elt in matrice_coordonnees:
                plateau,nb,e,liste_carre=remplissage(plateau,L,C,tour)
                case+=nb
            matrice_coordonnees=quatrieme_barre(plateau,L,C)
            for elt in matrice_coordonnees:
                choix.append(elt)
        remplie=True
        return plateau,case,remplie,choix
    else:
        return plateau,case,remplie,choix
```

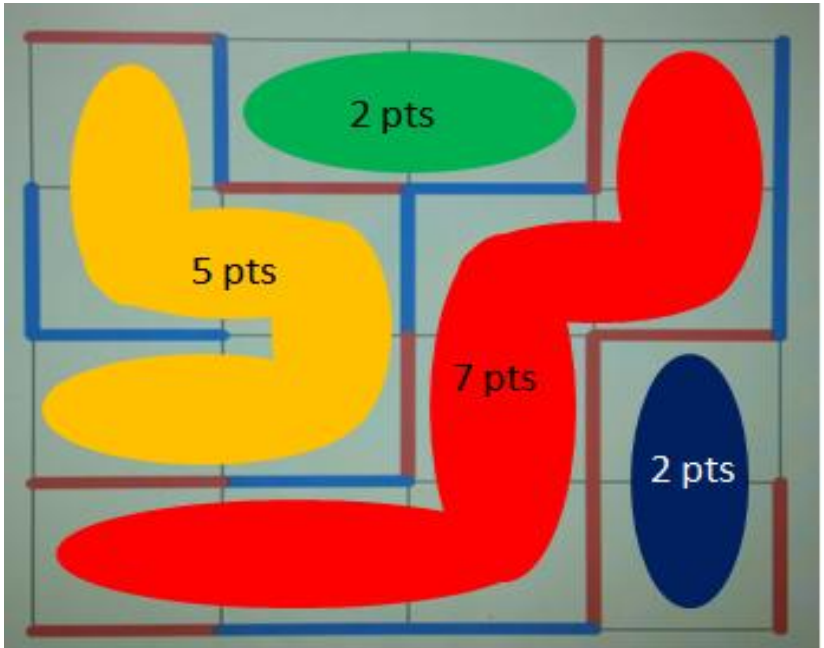
choix=[[0,3,[[10,10,10,10]]],
[0,6,[[10], [10], [10], [10]]], [0,6 ,[[10,10,10,10]]]]

case=2
rempli= True

Etape 3

```
def nombre_de_cellule(plateau,L,C,tour):
    type=[]
    comparaison=[]
    copie1_plateau=deepcopy(plateau)
    copie1_plateau,case1,remplie,choix1=poser_les_quatrieme_barre(copie1_plateau,L,C,tour)
    place_vide_possible1=place_vide(copie1_plateau,L,C)
    for elt in place_vide_possible1:
        copie2_plateau=deepcopy(copie1_plateau)
        copie2_plateau=strategie_pas_poser_troisieme_barre([elt],copie2_plateau,L,C,tour)
        place_vide_possible2=place_vide(copie2_plateau,L,C)
        copie2_plateau,case,remplie,choix=poser_les_quatrieme_barre(copie2_plateau,L,C,tour)
        choix.append(curseur_forme(elt,tour))
        type.append(choix)
    for k in range(len(type)):
        if not comparaison_liste(comparaison,type[k]):
            comparaison.append(type[k])
    return len(comparaison)
```

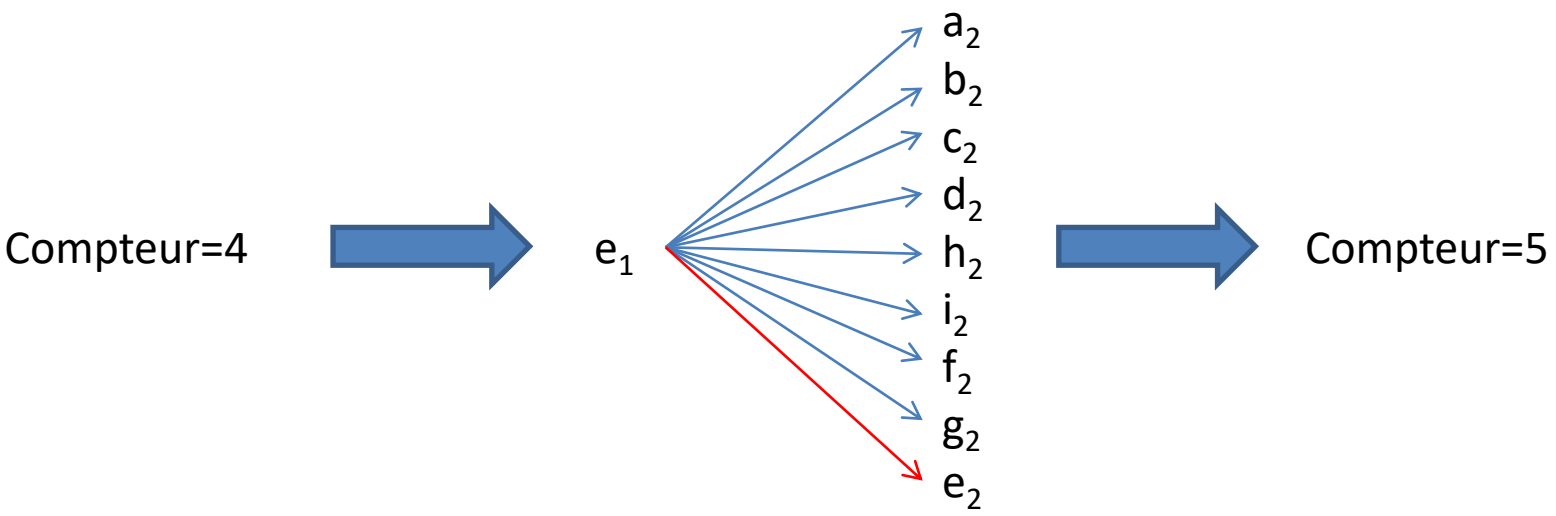




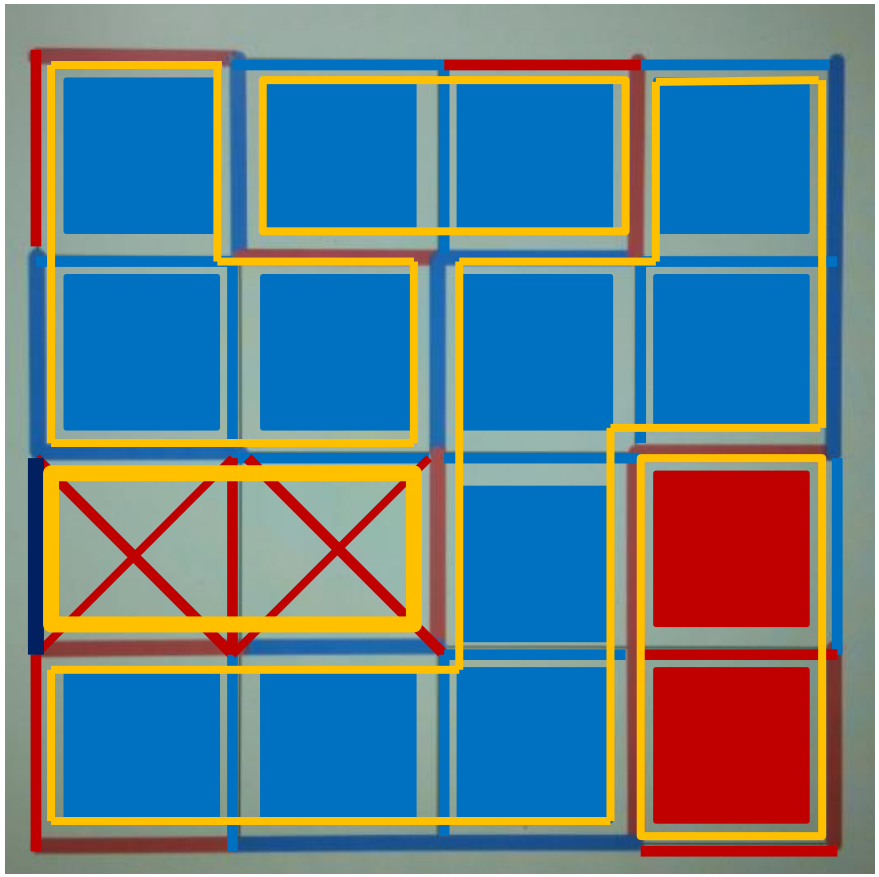
```python
def comparaison_liste(comparaison,type):
    if len(comparaison)!=len(type):
        return False
    else:
        for elt in comparaison:
            compteur=0
            for elt1 in elt:
                if elt1 in type:
                    compteur+=1
            if compteur==len(type):
                return True
        return False
```

comparaison=[a,b,c,d,e,f,g,h,i]$_1$

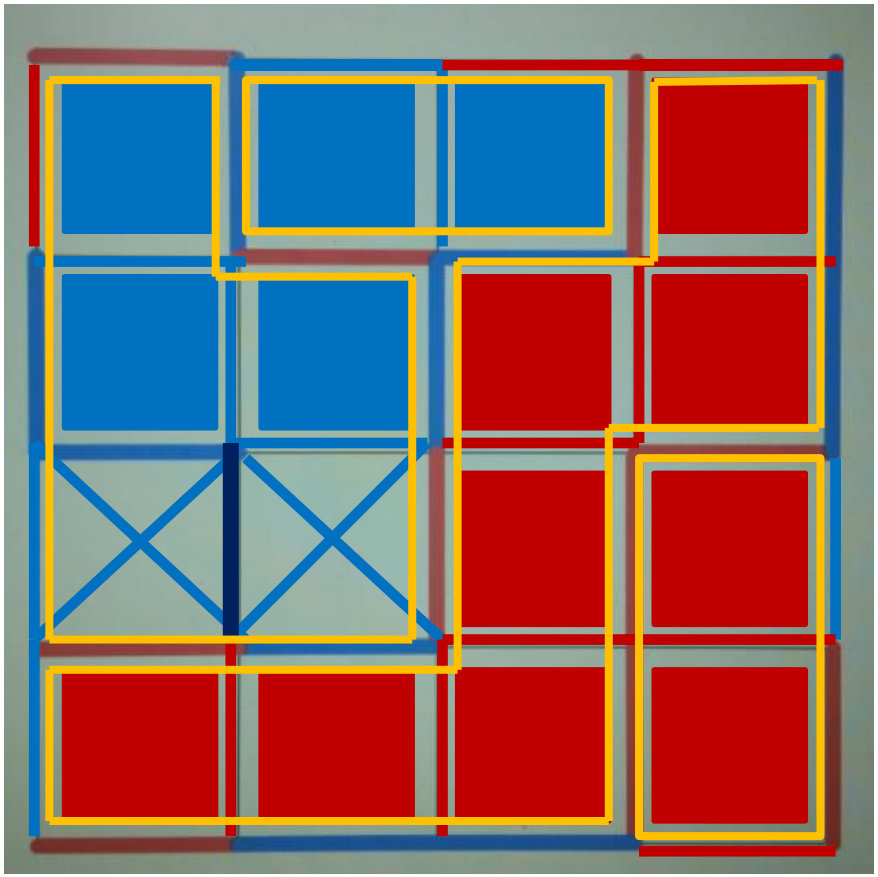
type=[a,b,c,d,h,i,f,g,e]$_2$



Résultat avec « prendre ou non »



Résultat sans « prendre ou non »



IA	**12**	**7**
Joueur	**4**	**9**