Informatique

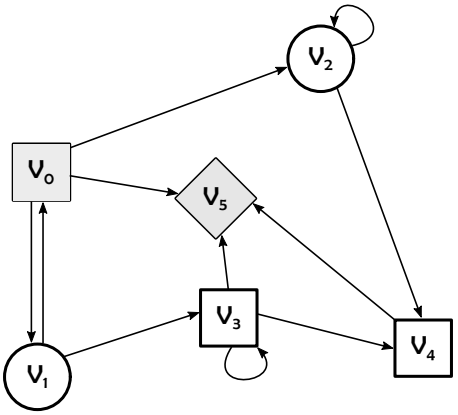
TP n°7

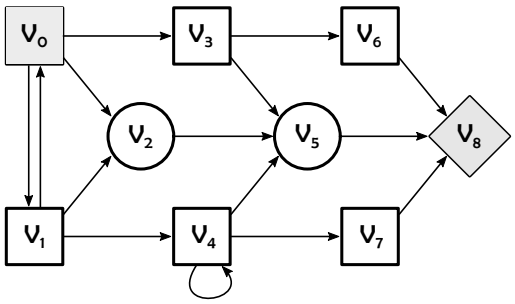
Théorie des jeux

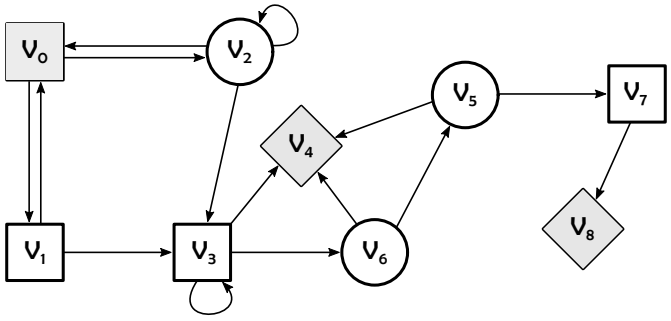
PSI : Lycée Rabelais

1 Positions gagnantes

On considère des jeux d'accessibilité pour lesquels les conditions de gain du joueur 1 sont représentées par des nœuds en forme de losange. Déterminer les positions gagnantes du joueur 1 (jouant les nœuds ronds) sur les arènes ci-dessous en détaillant les étapes (méthode des attracteurs) :





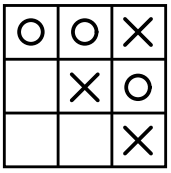


2 Heuristique *minimax* et jeu de *tic-tac-toe*

Un symbole $\times$ ou $\circ$ est attribué à chaque joueur (respectivement $J_\times$ et $J_\circ$). Les deux joueurs tracent alternativement leur symbole dans une case vide d'une grille carrée de neuf cases.

Le gagnant est le premier à obtenir une ligne horizontale, verticale ou diagonale de trois de ses symboles.

On considère que c'est au joueur $J_\circ$ de jouer et que le jeu est dans la configuration suivante :



On souhaite déterminer le meilleur coup à jouer pour $J_\circ$.

Q1. Mettre en place l'arbre de jeu (on dessinera sur chaque nœud la configuration du jeu).

On utilise une évaluation heuristique *minimax* pour chaque configuration. On rappelle que celle-ci se déduit, pour le joueur $J_\circ$, de la manière suivante :

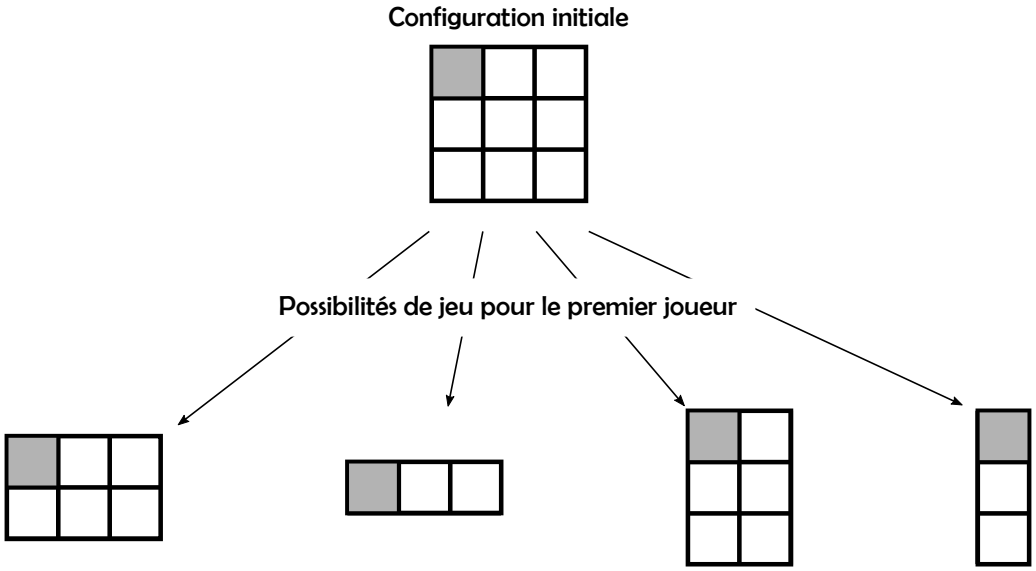
- Une feuille de l'arbre sera associée au gain +1 si elle correspond à la victoire de $J_\circ$, -1 à une défaite et 0 à une partie nulle.
- On définit les autres valeurs de manière récursive :
    - Si le nœud est contrôlé par $J_\circ$, sa valeur est le maximum des valeurs de ses sous-arbres.
    - Si le nœud est contrôlé par $J_\times$, sa valeur est le minimum des valeurs de ses sous-arbres.

Q2. Associer le coût de chacune des configurations.

Q3. Déterminer le meilleur coup à jouer.

3 Programmation d'un jeu de *Chomp* simplifié

On considère un jeu de *Chomp* simplifié. Dans ce jeu, on dispose d'une tablette de chocolat (ici de taille 3×3). Le carré en gris ne doit pas être mangé, il est empoisonné. Le joueur qui le mange a perdu. Successivement chaque joueur (noté J_1 ou J_2) peut prendre une ou plusieurs ligne(s), ou une ou plusieurs colonne(s).



On considère arbitrairement que c'est le joueur J_1 qui commence.

3.1 Arbre de jeu et heuristique *minimax*

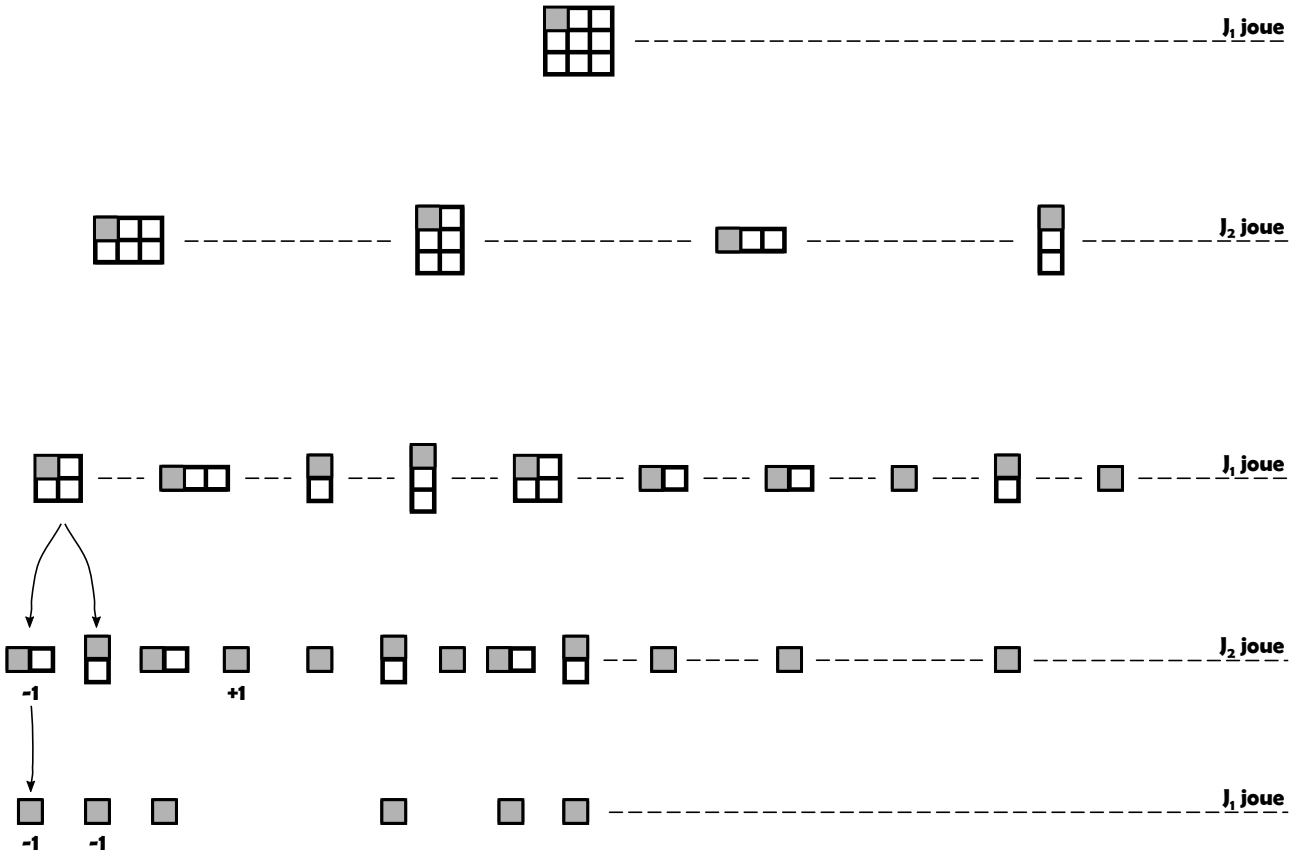
Question 1. Compléter le graphe ci-dessous en ajoutant les liens entre les différentes configurations.

On utilise une évaluation heuristique *minimax* pour chaque configuration. On rappelle que celle-ci se déduit, pour le joueur J_1, de la manière suivante :

- Une feuille de l'arbre sera associée au gain +1 si elle correspond à la victoire de J_1, -1 à une défaite .
- On définit les autres valeurs de manière récursive :
  - Si le nœud est contrôlé par J_1, sa valeur est le maximum des valeurs de ses sous-arbres.
  - Si le nœud est contrôlé par J_2, sa valeur est le minimum des valeurs de ses sous-arbres.

Question 2. Associer le coût de chacune des configurations.

Question 3. Faut-il être le premier à jouer ?

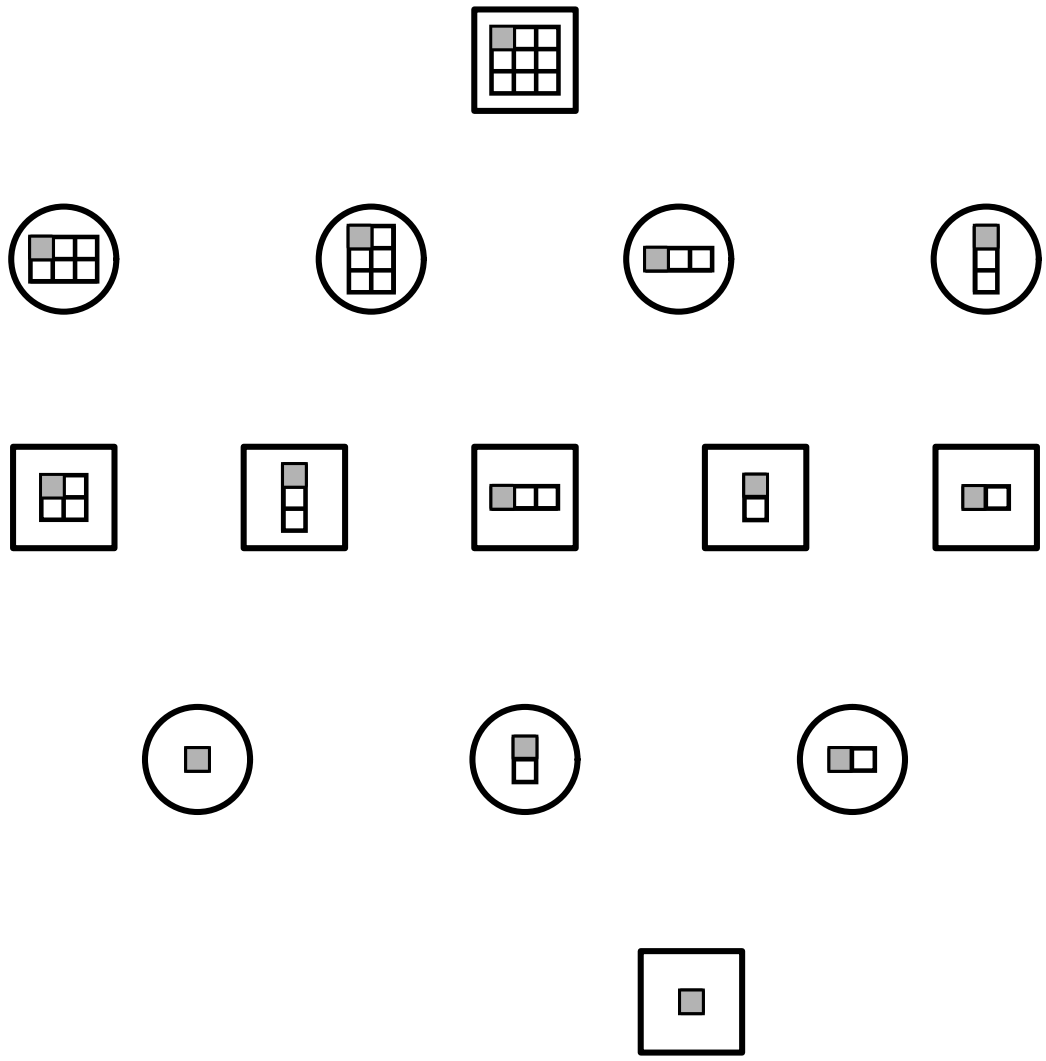


3.2 Modélisation du jeu

On veut modéliser le jeu par un graphe. Dans celui-ci, on tracera dans des nœuds carrés les configurations de la tablette de chocolat si c'est au joueur 1 de jouer. On notera dans des cercles les configurations de la tablette de chocolat si c'est au joueur 2 de jouer.

On rappelle que c'est le joueur J_1 qui commence.

Question 4. Compléter le graphe ci-dessous (en rajoutant les arêtes) .



Question 5. Colorier en bleu les positions gagnantes pour le joueur 1 et en rouge les positions gagnantes pour le joueur 2.

À la place de numéroter chaque nœud, on préfère choisir un `tuple` qui respecte le formalisme suivant : `(joueur , nombre de lignes , nombre de colonnes)`. Le nœud de départ sera, par exemple, associé au `tuple (1,3,3)`.

Question 6. Annoter le schéma précédent avec le `tuple` de chacun des nœuds.

On donne sur votre *cahier de prépa*, un fichier `chomp.py`. Dans ce fichier, une fonction `tablette(nblignes,nbcolonnes)` permet d'afficher une tablette ayant `nbcolonnes` colonnes et `nblignes` lignes.

Question 7. Afficher à l'écran une tablette de taille 2 lignes et 3 colonnes.

On donne également un dictionnaire `G` qui modélise partiellement le graphe établit précédemment (rappelé ci-dessous). Le formalisme est le suivant :

- Les clés du dictionnaire sont les nœuds du graphe ;
- La valeur associée à une clé est la liste de tous les nœuds pouvant être atteint.

Par exemple, pour la première ligne, le nœud de départ `(1,3,3)` permet bien d'accéder aux nœuds de la liste `[(2,2,3) , (2,3,2) , (2,1,3) , (2,3,1)]`.

```
G = {(1,3,3):[(2,2,3),(2,3,2),(2,1,3),(2,3,1)],
     (2,1,3):[(1,1,2),(1,1,1)],
     (2,3,1):[(1,2,1),(1,1,1)],
     (2,1,1):[],
```

```
    (1,1,1):[]}
```

Question 8. Compléter le dictionnaire `G` pour représenter entièrement le graphe.

On considèrera dans toute la suite que `G` est une variable globale. Il n'est donc pas nécessaire de la mettre en argument des fonctions.

Question 9. Écrire puis tester une fonction `predecesseurs(s)` qui prend en argument un sommet `s` et renvoie une liste contenant les sommets prédécesseurs de ce sommet. Par exemple, `predecesseurs((1,2,2))` devra renvoyer `[(2, 2, 3), (2, 3, 2)]`.

Question 10. Écrire puis tester une fonction `conditionsGain(joueur)` qui renvoie la liste des sommets gagnants (conditions de gain) pour le joueur `joueur` (où `joueur` prendra les valeurs 1 ou 2). Par exemple, `conditionsGain(1)` devra renvoyer `[(2, 1, 1)]`.

Question 11.

Version très difficile : Écrire puis tester une fonction `attracteur(joueur)` qui renvoie la liste des sommets qui correspondent à des positions gagnantes pour le joueur `joueur`.

Version difficile : Compléter le code `attracteur(joueur)` ci-dessous qui renvoie la liste des sommets qui correspondent à des positions gagnantes pour le joueur `joueur`.

```
def attracteur(joueur):

    testG = {} ## Dictionnaire qui contiendra :
               ##    clés : sommet
               ##    valeur : True si le sommet est un attracteur de joueur
               ##             False sinon
    for si in G:
        testG[si] = False ## Ce n'est pas un attrateur.
                          ## On changera par True si c'est un attracteur.

    ## initialisation
    A = conditionsGain(joueur)
    for ai in A:
        testG[ai] = ... à compléter ...

    for j in range(0,len(G)): ## certains passages sont superflus...
        for si in G:
            if testG[si]==True:
                pred = ... à compléter ...
                for pi in pred:
                    if joueur == pi[0]: ## joueur joue le coup précédent
                        testG[pi] = ... à compléter ...

                    else: ## autreJoueur joue le coup précédent
                          ## c'est un attracteur si l'autre joueur est
                          ## obligé de faire gagner joueur

                          ## on teste tous les sucesseur de pi
                          ## si ça ne convient pas, ce n'est pas un attracteur
```

```
                        convient = True
                        for succ in G[pi]:
                            if testG[succ] == False:
                                convient = False

                        if convient: ## == True
                            ... à compléter ...
    listAttracteur = []
    for si in testG:
        if testG[si] == True:
            listAttracteur.append(si)
    return listAttracteur
```

Version facile : Utiliser le code fourni sur votre *cahier de prépa* dans le fichier `attract.py` (faire un copier-coller) contenant la fonction `attracteur(joueur)` et vérifier qu'elle renvoie la liste des sommets qui correspondent à des positions gagnantes pour le joueur `joueur`.

Question 12. Lancer la fonction `jouerEnCommencant()` (commentée dans le fichier `champ.py`) et faites une partie. Analyser le code et expliquer comment l'ordinateur choisit le coup à jouer ? Est-il possible de gagner ?

Question 13. On propose également le code `jouerEnCommencantFacile()`. Expliquez pourquoi, cette fois-ci, ils vous est possible de gagner.

Question 14. Écrire une fonction `jouerSansCommencer()` où c'est à l'ordinateur de commencer à jouer.

Question 15. Lire la "vraie règle" du Chomp sur internet et déterminer à nouveau l'arbre de jeu pour une tablette de taille 3×3.