TD08 : Jeux d'accessibilité à deux joueurs

- Référence : ce TD est inspiré de la source que l'on trouvera ici
- Code du notebook associé : <donné en classe>

1 Cadre général

1.1 Jeux d'accessibilité à deux joueurs

Dans le cadre du programme, on s'intéresse à des jeux à deux joueurs, dans lesquels chaque joueur joue à tour de rôle, avec une connaissance complète de l'état du jeu (jeux *à information complète*), avec pour but d'atteindre un état du jeu qui permet de le déclarer gagnant (jeux d'*accessibilité*).
Note : Les jeux considérés sont « sans mémoire », dans le sens où l'historique des coups précédents ne sera pas pris en compte, et se déroulent en temps « infini » (chaque joueur dispose du temps qu'il souhaite pour jouer un coup).

1.2 Modélisation du jeu

On modélise le jeu par une *arène*, qui est un graphe orienté et fini, $G = (V, E)$, dans lequel les sommets du graphe représentent des états du jeu (ensemble V), et les arêtes (ensemble E) représentent les coups possibles (faisant passer d'un état du jeu à un autre).
Cependant, un état du jeu, pour être pleinement descriptif doit indiquer quel est le joueur qui est autorisé à jouer. Typiquement, un état du jeu consistera en la description du plateau de jeu et la connaissance du joueur qui a la main.
Ainsi, si on numérote 1 et 2 les deux joueurs, l'ensemble des sommets, peut être partitionné en deux sous-ensembles, V_1 et V_2, ensembles des états du jeu où le joueur 1, respectivement le joueur 2, a la main (on dira que les états de V_i sont des états *contrôlés* par le joueur i).
On entoure usuellement d'un cercle les états contrôlés par le joueur 1 et d'un carré les états contrôlés par le joueur 2.
Dans la mesure où l'on traite de jeux d'accessibilité, on doit aussi spécifier deux sous-ensembles, Ω_1 et Ω_2, correspondant, respectivement, aux états du jeu dans lequel le joueur 1 ou le joueur 2 est gagnant.
On appelle Ω_i l'objectif du joueur i.
Éventuellement, on peut distinguer un état Ω_3 correspondant aux états du jeu donnant une partie nulle.

Lorsque les deux joueurs jouent à tour de rôle, les ensembles V_1 et V_2 sont conditionnés par la donnée du joueur qui commence la partie.
Il conviendra donc d'indiquer quel est le joueur qui jouera en premier (on conviendra ici qu'il s'agira systématiquement du joueur 1), ainsi que l'état initial du jeu.
Si les joueurs jouent à tour de rôle, les arêtes, représentant les coups jouables, *i.e.* les transitions possibles entre états du jeu, seront données par des couples (s, s') dans lesquels, nécessairement, $s \in V_1$ et $s' \in V_2$, ou bien $s \in V_2$ et $s' \in V_1$. De ce fait, l'arène sera un graphe dit *biparti*, dans lequel toute arête a son origine dans V_i et son extrémité dans V_j avec $(i, j) \in \{1,2\}$ et $i \neq j$.

1.3 Déroulement d'une partie

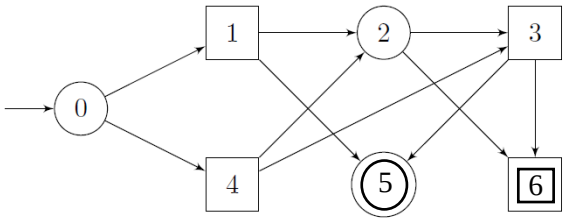
Un état de départ pour le jeu étant donné, incluant la connaissance du joueur contrôlant cet état initial (le joueur 1 ici, arbitrairement), une partie sera modélisée par une succession de coups (éventuellement infinie), retraçant les coups joués depuis le début de la partie, jusqu'à son terme s'il existe : il s'agit donc d'un chemin dans le graphe orienté qu'est l'arène.

1.4 Stratégies

On appelle *stratégie* pour le joueur i, une fonction qui, à chaque état contrôlé par le joueur i, associe une transition possible depuis cet état (*i.e.* le coup que jouera le joueur i).
Une stratégie sera dite *gagnante* pour le joueur i, si l'application de cette fonction assure la victoire au joueur i, quelle que soit la stratégie du joueur adverse.

2 Étude d'un exemple

On considère le jeu décrit par l'arène suivante



où $G = (V, E)$ avec $V = \{(0,1)\,;(1,2);(2,1);(3,2);(4,2);(5,1);(6,2)\}$, un sommet de V étant décrit par un couple (i, J) donnant son numéro, i, et le numéro du joueur qui contrôle cet état, J, et avec $E = \{((0,1),(1,2)),((0,1),(4,2)),((1,2),(2,1)),\ldots,((4,2),(3,2))\}$.
L'état initial (signalé par une flèche) est l'état 0, contrôlé par le joueur 1, qui commence donc toute partie.
Les états 5 et 6 correspondent, respectivement aux objectifs des joueurs 1 et 2, qui gagnent s'ils les atteignent au cours d'une partie. On les a mis en évidence en les entourant avec un trait double.
Avec les notations précédentes, les objectifs des deux joueurs sont donc $\Omega_1 = \{(5,1)\}$ et $\Omega_2 = \{(6,2)\}$.

Q1. Afin d'implémenter le jeu décrit ci-dessus, définir une liste des numéros de sommets, `LS`, et un dictionnaire, `dC`, dont les clés sont les numéros de sommets, avec pour valeur associée, le numéro du joueur contrôlant cet état, ainsi qu'un dictionnaire des listes d'adjacences de chaque sommet, `dAdj`.
Définir ensuite deux listes, `LObj1` et `LObj2`, donnant les états-objectifs des deux joueurs.
On définira enfin le jeu avec toutes ses caractéristiques, de la façon suivante :

```
jeu = {'etats': LS, 'init': 0, 'controles': dC, 'transitions': dAdj,
       'objectifs1': LObj1, 'objectifs2': LObj2}
```

où la clé `'init'` est associée à l'état initial du jeu.

Une stratégie pour le joueur 1, correspond à la donnée d'une fonction associant à chaque état contrôlé par le joueur 1, un état suivant possible.
On peut modéliser une telle fonction par un dictionnaire `strategie1`, dont les clés sont les états contrôlés par le joueur 1 et les valeurs associées l'état suivant que choisit d'atteindre le joueur 1, à partir de chacun d'eux.
Par exemple, le dictionnaire suivant est une stratégie possible pour le joueur 1 :

```
strategie1 = {0: 1, 2: 3}
```

Q2. Combien existe-t-il de stratégies pour le joueur 1 ?

Q3. Il existe une stratégie gagnante pour le joueur 2. Laquelle ? La décrire à l'aide d'un dictionnaire, `strategie2`.

Q4. Écrire une fonction `execute` qui prend en paramètre une description du jeu sous la forme donnée en Q2, et deux stratégies, une pour chaque joueur, et fait avancer la partie en suivant les instructions données par les deux stratégies, et s'arrête quand un objectif a été atteint et renvoie la liste des états du jeu. Dans le cas où une stratégie propose une transition non autorisée, on lèvera une exception.

Q5. Tester la fonction `execute` avec le jeu décrit en Q1 et la stratégie gagnante pour le joueur 2 et diverses stratégies pour le joueur 1, afin de vérifier que l'on arrive toujours dans l'état gagnant pour le joueur 2.

Attracteurs

Afin de déterminer si une stratégie gagnante existe pour l'un des deux joueurs, on peut calculer des ensembles d'*attracteurs* pour chacun des joueurs.
Pour déterminer les stratégies gagnantes, s'il en existe, pour le joueur *j*, on définit une suite $\left(Attr_i^j\right)_{i\geq 0}$, où $Attr_i^j$ est l'ensemble des états du jeu à partir desquels le joueur *j* dispose d'une stratégie qui le fait gagner en *i* étapes ou moins (chacune des *i* étapes correspondant à un coup joué par l'un ou l'autre joueur).
On construit cette suite par récurrence.

Pour $i = 0$, $Attr_0^j = \Omega_j$. En effet, le joueur j gagne en zéros étapes s'il se trouve déjà dans l'un des états réalisant son objectif.
Pour tout $i \geq 1$, si l'ensemble $Attr_{i-1}^j$ est connu, alors le joueur j est assuré de gagner en i étapes ou moins, si et seulement si :

- le jeu est dans un état élément de $Attr_{i-1}^j$;
- le jeu est dans un état s contrôlé par j et à partir duquel il existe une transition (un coup à jouer par j) vers un état s' élément de $Attr_{i-1}^j$;
- le jeu est dans un état contrôlé par l'adversaire, mais à partir duquel toutes les transitions possibles (tous les coups que l'adversaire peut jouer) amène à un état s' élément de $Attr_{i-1}^j$.

Ce qui peut se traduire, en notant $\bar{j}$ l'opposant du joueur j, par la relation de récurrence suivante :

$$Attr_i^j = Attr_{i-1}^j \cup \{s \in V_j, \exists\, (s,s') \in E, s' \in Attr_i^j\} \cup \{s \in V_{\bar{j}}, \forall\, (s,s') \in E, s' \in Attr_i^j\}.$$

On définit ainsi une suite d'ensembles d'états, croissante pour l'inclusion, et nécessairement stationnaire, car l'ensemble des états est fini.
On pourra noter $Attr_\infty^j$ l'ensemble des états atteint au moment où la suite devient stationnaire.

Si l'on construit cette suite jusqu'à ce qu'elle soit stationnaire, **on peut conclure à l'existence d'une stratégie gagnante pour le joueur j, si l'état initial est élément de $Attr_\infty^j$.**

Q6. Définir deux fonctions prenant toutes deux en argument la description d'un jeu, `jeu`, un état, `e`, et une fonction booléenne prenant en argument un état, `f` :

- la première de ces fonctions, `exist_succ`, renverra `True` s'il existe au moins un successeur de l'état `e` évalués à `True` par la fonction `f` ;
- la seconde, `forall_succ`, renverra `True` si l'état `e` a au moins un successeur et que tous les successeurs possibles de l'état `e` sont évalués à `True` par la fonction `f`.

Q7. En supposant que les attracteurs sont définis sous la forme de listes Python d'états, définir une fonction, `appartient`, prenant en argument un état `e` et un attracteur, `A`, et renvoyant un booléen indiquant si l'état `e` appartient à `A` ou non. Évaluer la complexité d'un appel à `appartient` en fonction de `len(A)`.

Q8. Expliquer en quoi une représentation des attracteurs sous la forme de dictionnaires dont les clés sont les états appartenant à l'attracteur, avec pour valeur associée None (ou toute autre valeur arbitrairement choisie), permet d'améliorer la complexité de la fonction `appartient`.
On adoptera cette représentation pour la suite.

Q9. Modifier si nécessaire le code de la fonction `appartient` afin de tenir compte de ce choix d'implémentation des attracteurs.

Q10. Définir une fonction `step`, prenant en argument la description d'un jeu, `jeu`, un numéro de joueur, `j`, et un attracteur, `A`, représentant un état de l'attracteur après i itérations, $Attr_i^j$, et ajoutant à l'attracteur `A`, les états qu'il convient pour obtenir l'attracteur $Attr_{i+1}^j$. La fonction `step` renverra `True` si des états ont été ajoutés et `False` sinon.

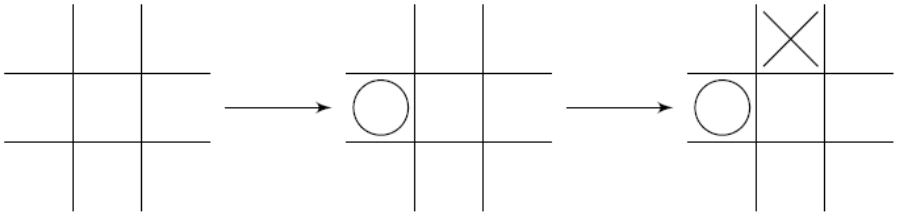
Q11. Écrire une fonction `attracteur_infty`, prenant en argument la description d'un jeu, `jeu` et un numéro de joueur, `j`, et renvoyant le dictionnaire représentant l'attracteur $Attr_\infty^j$.

Q12. Tester la fonction `attracteur_infty`, sur le jeu donné en exemple et vérifier qu'il est confirmé qu'il existe une stratégie gagnante pour le joueur 2, mais pas pour le joueur 1.

Dans le cas où il existe une stratégie gagnante pour un joueur, on voudrait pouvoir déterminer explicitement l'une de ces stratégies.

Q13. Modifier les fonctions `step` et `attracteur_infty` afin de construire un dictionnaire des coups à jouer pour tous les états présents dans l'attracteur $Attr_\infty^j$ dans le cas où il existe une stratégie gagnante pour le joueur j.

3 Étude du jeu de Morpion



On représente un état du plateau de jeu par un tableau à deux dimensions de taille 3×3, dont les éléments sont les caractères, espace, X ou O, figurant, respectivement une case vide, une case occupée par les joueurs 1 ou 2, et dont un exemplaire est stocké dans une liste `symb = [' ', 'O ', 'X']`.

Q14. Écrire une fonction `plateau_vide()`, renvoyant le plateau dans son état initial, vide.

Q15. Écrire une fonction `put(plateau, i, j, k)`, qui renvoie une copie du plateau, dans laquelle on a ajouté le symbole du joueur k dans la cellule de coordonnées (i, j).

Q16. Écrire une fonction `get(plateau, i, j, k)`, qui le symbole contenu dans la cellule de coordonnées (i, j).

On cherche à se donner une représentation du jeu ayant la même forme que précédemment, soit

```
jeu = {'etats': LS, 'init': 0, 'controles': dC, 'transitions': dAdj,
       'objectifs1': LObj1, 'objectifs2': LObj2}
```

Afin de ne pas occuper inutilement de la place en mémoire, on stockera les 3^9 configurations possibles du plateau sous la forme d'une liste `confs`, et ces configurations seront représentées dans jeu, par leur position dans la liste `confs`.

Q17. Créer la liste, `confs`, des 3^9 configurations possibles. Construire la liste `LS` des numéros de configurations correspondante.

On rappelle qu'il est convenu que le joueur 1 est le joueur commençant la partie.

Q18. Écrire une fonction `owner`, prenant en argument le numéro d'une configuration (un état du plateau de jeu), et renvoyant le numéro du joueur qui doit jouer le coup suivant, et qui lève une exception si la configuration n'est pas correcte. Utiliser la fonction `owner` pour construire le dictionnaire `dC` donnant pour chaque numéro de configuration, le numéro du joueur la contrôlant.

On rappelle qu'une configuration est gagnante pour le joueur k si le joueur k a réussi à aligner trois symboles, horizontalement, verticalement ou le long de l'une des diagonales du carré.

Q19. Écrire une fonction `winning(confnum, k)`, renvoyant `True` si la configuration du plateau de numéro `confnum` est gagnante pour le joueur k. Utiliser la fonction `winning` pour construire les listes `LObj1` et `LObj2` des objectifs des deux joueurs, les configurations gagnantes étant désignées par leurs numéros.

Q20. Écrire une fonction `successeurs(confnum)`, renvoyant une liste des successeurs possibles de la configuration du plateau donnée en entrée. Utiliser la fonction successeurs pour construire le dictionnaire `dAdj` donnant les transitions possibles depuis chaque configuration du jeu.

Q21. Définir le dictionnaire `jeu` décrivant le jeu.

Q22. Calculer les attracteurs, `attr1`, `attr2`, des deux joueurs.

Q23. Déterminer s'il existe une stratégie gagnante pour l'un des deux joueurs.

Q24. Si une telle stratégie existe, la définir sous la forme d'un dictionnaire et la tester face à diverses stratégies pour l'adversaire.