TD IA et Jeux

8 Janvier 2018

1. Commentaires sur le code de création des jeux de données

Le code suivant vous a été donné à la séance précédente :

```
def RencontreJeu_et_SauveDatas(j, N, j1, j2):
    Resus = [0,0,0]
    States=np.zeros((0,6))
    Scores=np.zeros((0,2))
    Liste_Parties=[]
    for i in range(N): #tqdm_notebook(xrange(N)): #
        if (i%100==0):
            print (i*100/N, " completed")
        j = AwaleGame()
        a = play_game(j, [j1, j2] , todisplay=False)
        print(a)
        if (a ==0) or (a==1):
            Resus[a]+=1
        if (a==-3):
            Resus[2] +=1
        if (a == 0)  or (a==1):
            if (a==0):
                 a=-1
            States_tmp, Scores_tmp, Moves_tmp = j.rejoue3()
            indices_pairs=[x for x in range(len(States_tmp)) if (x%2 ==0) ]
            player = np.ones((len(States_tmp),1))
            res_tmp = np.ones((len(States_tmp),1)) * a * (-1)
            res_tmp[indices_pairs] = a
            player[indices_pairs] = 0
            Tomemorize_tmp= np.hstack((States_tmp,Scores_tmp, Moves_tmp, res_tmp, player))
            Tomemorize_tmp = Tomemorize_tmp[10:,:]
            Liste_Parties.append(Tomemorize_tmp)
    return (Resus, Liste_Parties)
```

qui utilise la méthode suivante de AwaleGame

```
def rejoue3(self):
    self.Moves.append(-1)
    l = self.Partie
    l2 = self.Moves
    States = np.zeros((len(l),12))
    Moves = np.zeros((len(l),1))
    Scores = np.zeros((len(l),2))
    i=0
    while (len(l)>0):
        e = l.pop()
        m = l2.pop()
        #self.display(e )
        #print (e.board)
        States[i,:]= e.board
        Scores[i,:]= e.score
        Moves[i]= m
        i+=1
    return States, Scores, Moves
```

Exemples de lignes créées

```
print (parties[:10,:])
```

```
[[ 1.  9.  7.  1.  7.  2.  8.  1.  0.  7.  0.  1.  2.  2.  4. -1.  0.]
 [ 1.  9.  7.  1.  0.  3.  9.  2.  1.  8.  1.  0.  4.  2.  8.  1.  1.]
 [ 1.  9.  7.  1.  0.  3.  9.  2.  0.  9.  1.  0.  4.  2.  0. -1.  0.]
 [ 0. 10.  7.  1.  0.  3.  9.  2.  0.  9.  1.  0.  4.  2.  6.  1.  1.]
 [ 1. 11.  8.  0.  0.  3.  0.  3.  1. 10.  2.  1.  4.  4.  0. -1.  0.]
 [ 0. 12.  8.  0.  0.  3.  0.  3.  1. 10.  2.  1.  4.  4.  9.  1.  1.]
 [ 1. 13.  9.  1.  1.  4.  1.  4.  1.  0.  3.  2.  4.  4.  2. -1.  0.]
 [ 1. 13.  0.  2.  2.  5.  2.  5.  2.  1.  4.  0.  7.  4.  7.  1.  1.]
 [ 0. 13.  0.  2.  2.  5.  2.  0.  3.  2.  5.  1.  7.  6.  3. -1.  0.]
 [ 0. 13.  0.  0.  3.  6.  2.  0.  3.  2.  5.  1.  7.  6. 11.  1.  1.]]
```

2. Joueur Alpha beta

On vous redonne ci dessous le code d'un joueur Alpha Beta standard :

```
class ABplayer(Player):

    def __init__(self, depth=2, cutoff_test=None, eval_fn=None):
        self.depth = depth
        self.cutoff_test = cutoff_test
        self.evaluation_function = eval_fn

    def feval(self, game, state):
        return (self.evaluation_function(game, state, self.depth))

    def best_move(self, game, state):
        "Return the player whose move it is in this state."

        def max_value(game, state, alpha, beta, depth):
            if self.cutoff_test(state, depth):
                return self.eval_fn(game, state)
            v = -infinity
            for a in game.actions(state):
                v = max(v, min_value(game, game.result(state,a),
                                     alpha, beta, depth+1))
                if v >= beta:
                    return v
                alpha = max(alpha, v)
            return v

        def min_value(game, state, alpha, beta, depth):
            if self.cutoff_test(state, depth):
                return self.eval_fn(game, state)
            v = infinity
            for a in game.actions(state):
                v = min(v, max_value(game, game.result(state,a),
                                     alpha, beta, depth+1))
                if v <= alpha:
                    return v
                beta = min(beta, v)
            return v

        player = game.to_move(state)
        self.cutoff_test = (lambda state,depth: depth>=self.depth or game.terminal_test(state))
        self.eval_fn =  (lambda game, state: self.feval(game, state))
        v = -infinity
        alpha = -infinity
        beta = infinity
        if (game.actions(state)==[]):
            print ("bizarre")
        for a in game.actions(state):
            vtemp = min_value(game, game.result(state,a), alpha, beta, 1)
            if (vtemp >=v):
                v = vtemp
                best_move_temp = a
        return best_move_temp
```

1.1 Joueur AB à profondeur variable

Ecrivez une variante de ce joueur qui va a une profondeur variable suivant le nombre de graines restantes en jeu.
En notant *depth* la profondeur passée en paramètre à la création du joueur AB on veut que le joueur explore jusqu'à *min(depth, 3)* si le nombre de graines est supérieur à 30, *min(depth, 5)* si le nombre de graines est compris entre 20 et 30, et *min(depth, 5)* si le nombre de graines est inférieur à 20.

1.2 Joueur avec predicteur du coup à jouer

Ecrivez une variante de ce joueur qui utilise un classifieur qui prédit le coup à jouer en fonction de l'état, ainsi que la commande permettant de créer un tel joueur

1.3 Joueur AB avec prédicteur de victoire ou défaite

Ecrivez une variante de ce joueur qui utilise un classifieur qui prédit si l'état du jeu est gagnant ou perdant, ainsi que la commande permettant de créer un tel joueur.