Optimisation de l'utilisation des challenges au tennis

grâce au Processus de Décision Markovien (PDM)

YAHAYA A.K Saad

Juillet 2024

CK081M

Introduction

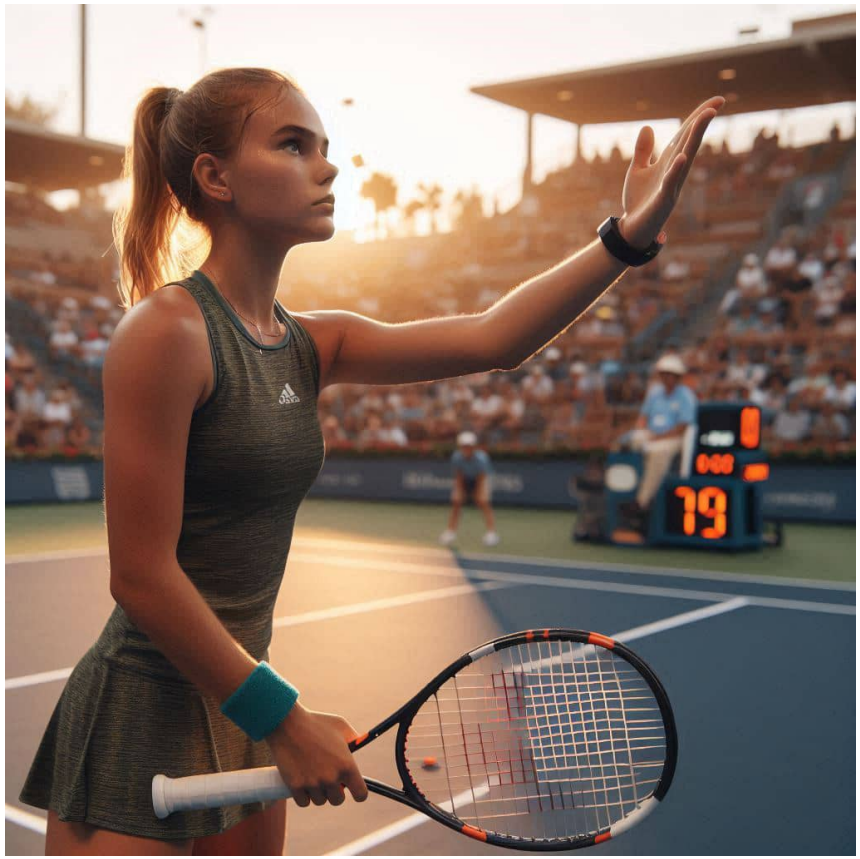


Figure – 1. Joueuse qui conteste l'arbitre

Introduction

Figure – 2. Système Hawkeye

Le dilemme

On appelle **challenge** le nombre de chances de contester à tort un arbitre.
Un joueur n'a qu'un maximum de 3 challenges.

- Un joueur trop **passif** risque de laisser beaucoup de bonnes occasions de contester correctement
- Un joueur trop **agressif** risque d'épuiser ses 3 chances trop tôt et d'être contraint de subir toute décision de l'arbitre le reste du jeu

Problématique

Comment optimiser l'utilisation des challenges au tennis ?

Théorie des PDM

C'est quoi un PDM ?

Un PDM est un quadruplet $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$ définissant :

- $\mathcal{S}$ l'ensemble des états
- $\mathcal{A}$ l'ensemble des actions
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ la fonction de transition
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ la fonction récompense

Politique π

C'est la fonction $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0; 1]$ qui détermine le comportement(choix d'actions) de l'agent dans son environnement.

Théorie des PDM

Propriété de Markov : *"Le futur ne dépend que du présent"*

Soit $(s_t)_{t\in\mathbb{N}}$ une suite d'états d'un système markovien, alors :

$$\forall t \in \mathbb{N},\ \mathbb{P}\left(s_{t+1}|s_t, ..., s_0\right) = \mathbb{P}\left(s_{t+1}|s_t\right)$$

Ainsi, $\forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$

$$\mathcal{P}\ (s, a, s') = \mathbb{P}\left(s_{t+1} = s'|s_t = s, a_t = a\right)$$

Théorie des PDM

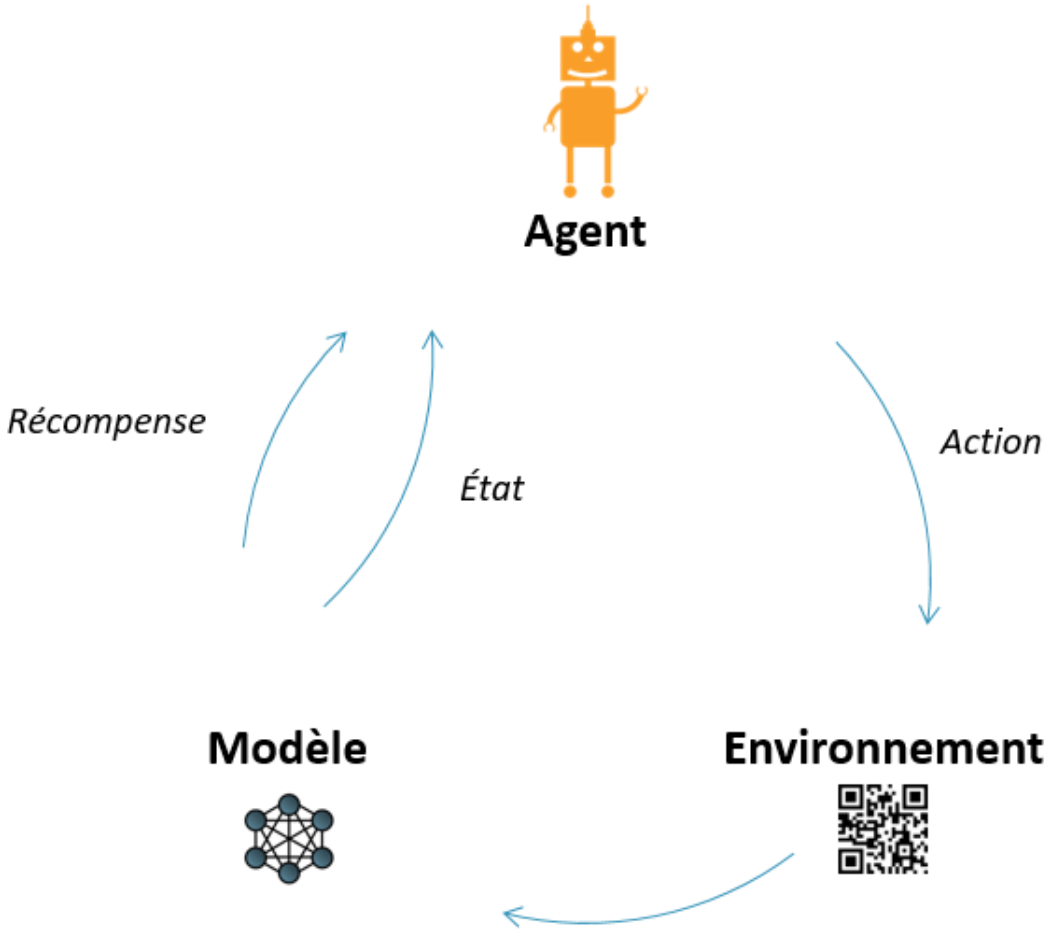


Figure – 3. Modèle d'apprentissage par renforcement

Théorie des PDM

Le Gain futur

Pour une politique $\pi, \forall t \in \mathbb{N}$, on note $\mathcal{R}_t$ le total de récompenses **amorties** que peut espérer l'agent à partir du temps t :

$$\mathcal{R}_t = \sum_{k=0}^{+\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \quad (1)$$

$\gamma \in [0,1]$: le facteur d'amortissement
r_t : la récompense obtenue au temps t suivant une politique π

Théorie des PDM

fonction valeur d'états v_π

Pour une politique π, on définie :

$$v_\pi : \mathcal{S} \rightarrow \mathbb{R} \tag{2}$$

$$s \rightarrow \mathbb{E}(\mathcal{R}_t | s_t = s)$$

Résoudre un PDM revient dès lors à déterminer la politique qui maximise la fonction valeur d'états

politique optimale π^*

C'est la politique qui vérifie :

$$\forall s \in \mathcal{S}, v_{\pi^*}(s) = v^*(s) = \max_\pi v_\pi(s) \tag{3}$$

Théorie de PDM

Equation de Bellman

La fonction valeur d'états peut s'écrire récursivement :

$$\forall s \in \mathcal{S}, v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')(\mathcal{R}(s,a,s') + \gamma v_\pi(s')) \quad (4)$$

Equation d'optimalité de Bellman

Dans ce cas, elle s'écrit :

$$\forall s \in \mathcal{S}, v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')(\mathcal{R}(s,a,s') + \gamma v^*(s')) \quad (5)$$

Théorie des PDM

Algorithme 1 : Policy iteration

Entrées : π *quelconque*
Sorties : π^*

1 V(s) $= 0\ \forall s \in \mathcal{S}$;
2 **repeat**
3 | **pour** $s \in S$ **faire**
4 | | $v_\pi(s) \leftarrow \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')(\ \mathcal{R}(s,a,s') + \gamma v_\pi(s')\ )$
5 | **fin**
6 **until** *V ne change presque plus*;

7 **pour** $s \in \mathcal{S}$ **faire**
8 | $\pi(s) \leftarrow \arg\max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')(\ \mathcal{R}(s,a,s') + \gamma v_\pi(s')\ )$
9 **fin**

10 **si** π *ne change plus* **alors**
11 | **return** $v \approx v^*, \pi \approx \pi^*$
12 **sinon**
13 | revenir à la ligne 2
14 **fin**

Hypothèses

1. le jeu oppose le joueur A et B
2. le score d'un joueur varie de 0 a 4
3. le premier joueur à avoir le score 4 gagne le jeu
4. le joueur A peut contester tant qu'il ne compte pas encore 3 échecs
5. Si A conteste avec succès, il gagne le point joué, sinon B gagne le point
6. On ne prend pas en compte les challenges de B
7. Perception=Probabilité : Si le joueur pense qu'il a une chance ϵ de contester correctement, alors la probabilité qu'il conteste correctement est ϵ

Modélisation

Actions

- Contester
- Ne pas contester

Récompense

On récompense l'agent à la fin du jeu :

- Si le joueur gagne le jeu, sa récompense est $+100$
- S'il perd le jeu, sa récompense est -100

Modélisation

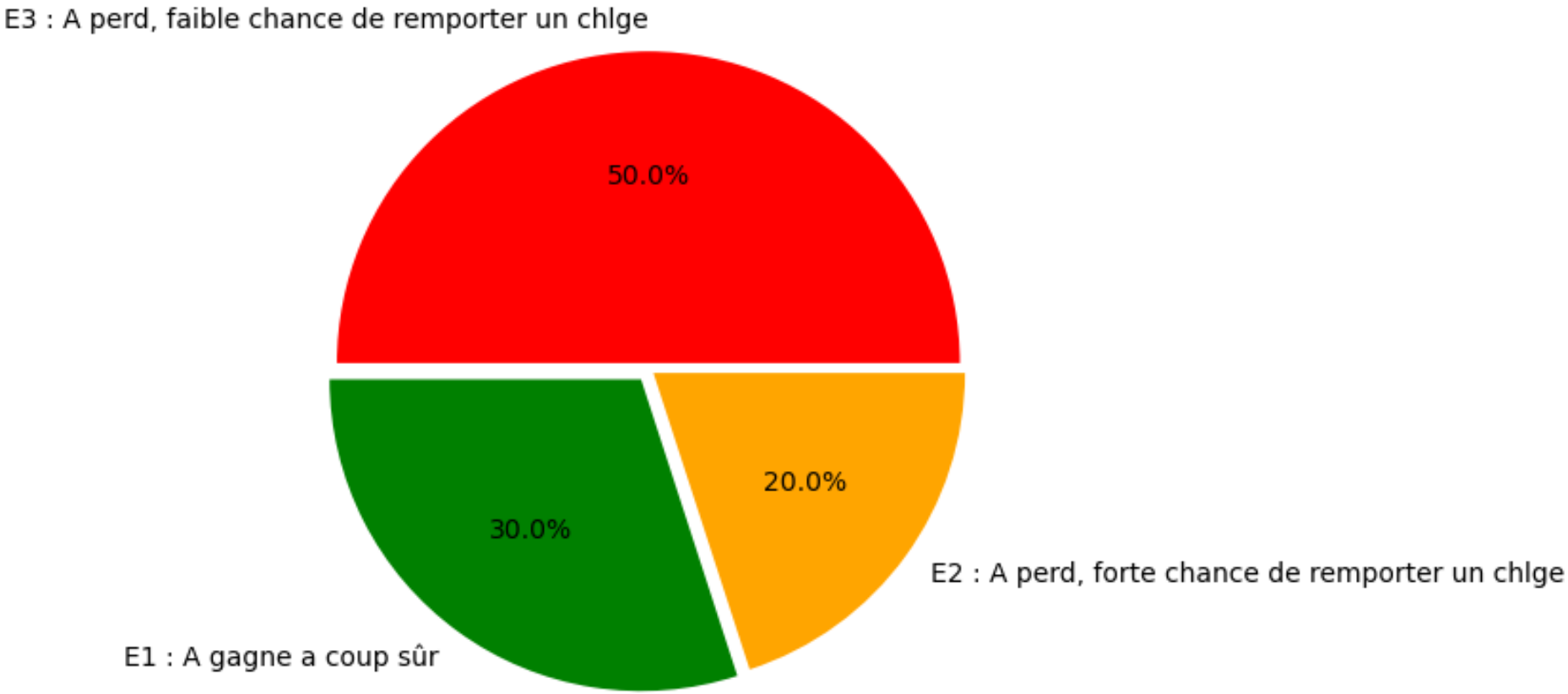


Figure – 4. Les types d'états de A

Modélisation

Algorithme 2 : Génération de $\mathcal{S}$

Sorties : $\mathcal{S} = [\,]$

```
1  S = [ ]
2  Types = [ E1, E2, E3 ]
3  pour scoreA de 0 a 4 faire
4      pour scoreB de 0 a 4 faire
5          pour nbreChlg de 0 a 3 faire
6              pour Ei dans Types faire
7                                          ▷ vérifier si les scores verifient les hypothèses
8                  si scoreB ≥ nbre_Chlge_ratés et scoreA + scoreB < 8 alors
9                      s = ( (scoreA, scoreB), nbreChlge, Ei )
10                     Ajouter s a S
11                 fin
12             fin
13         fin
14     fin
15 fin
```

$card(\mathcal{S}) = 146$

Modélisation

Algorithme 3 : Définition de $\mathcal{P}(s, a, s\_prime)$

Entrées : s, a, s_prime
Sorties : $\mathcal{P}(s, a, s\_prime)$

pE1 = 0.3, pE2 = 0.2, b = 0.8, pE3 = 0.5, c = 0.2
Calculer ΔScore et ΔnbreChalge
si *A a perdu le point joué* **alors**
  si *la transition n'est pas possible (Ex : s est du type E1)* **alors**
    return p = 0
  sinon
    p' = f(a, type d'état Ei de s, b, c)
  fin
fin

si *A a perdu le point joué* **alors**
  ... Se déduit par symétrie
fin

si *s_prime est du type Ei* **alors**
  $p = p' * pEi$
fin
return p

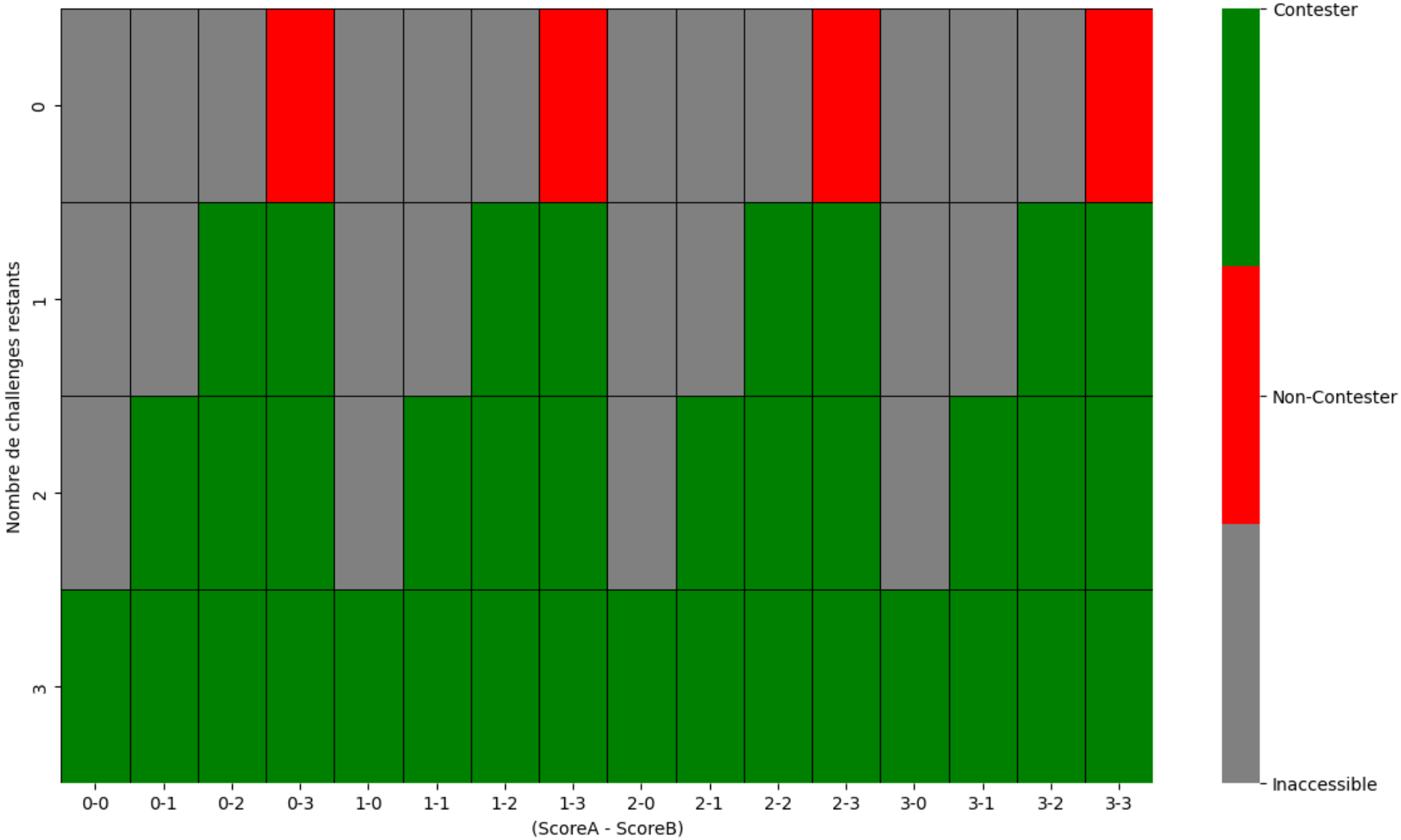


Figure – 5. Actions optimales dans un état de forte perception ; E2

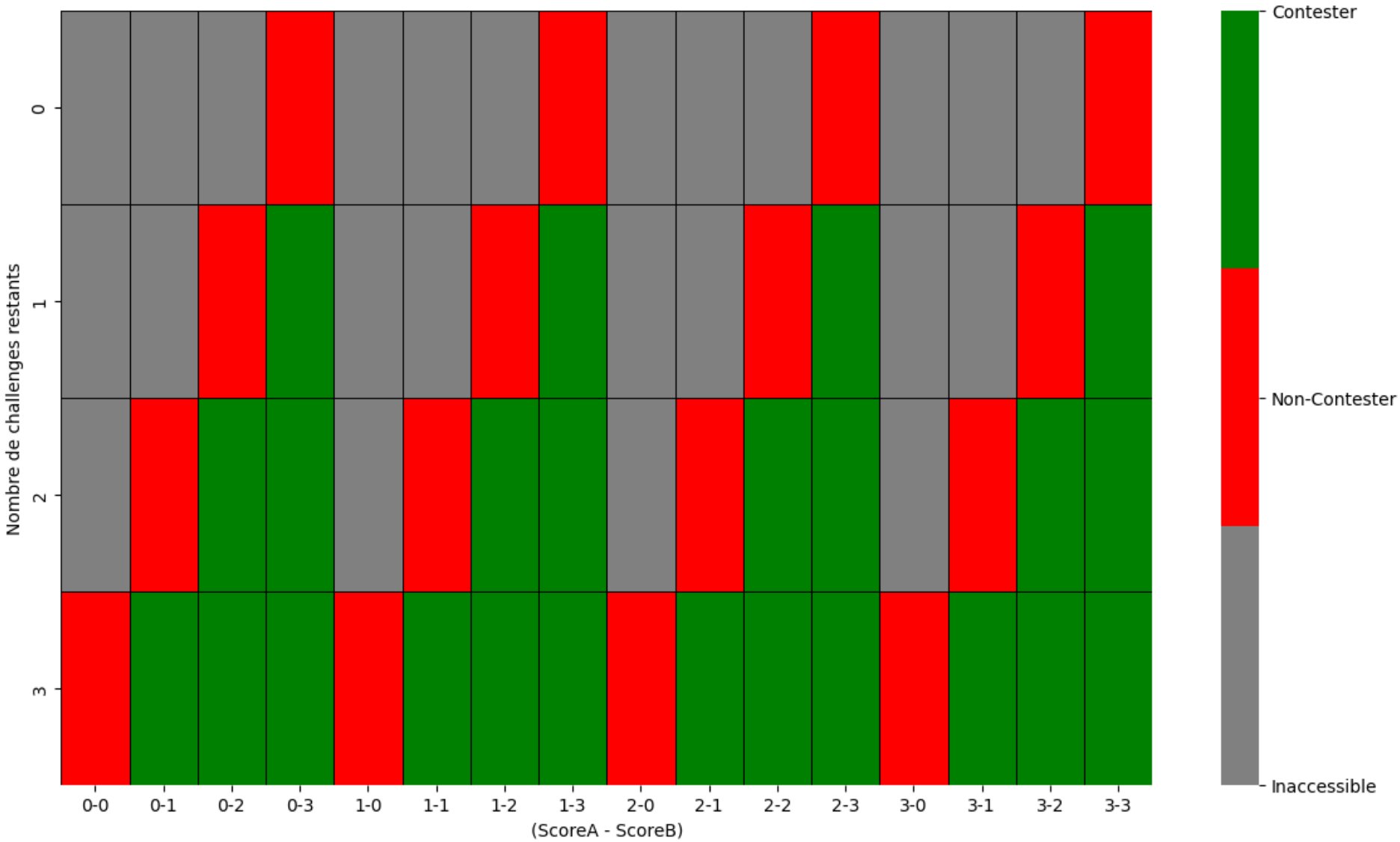


Figure – 6. Actions optimales dans un état de faible perception ; E3

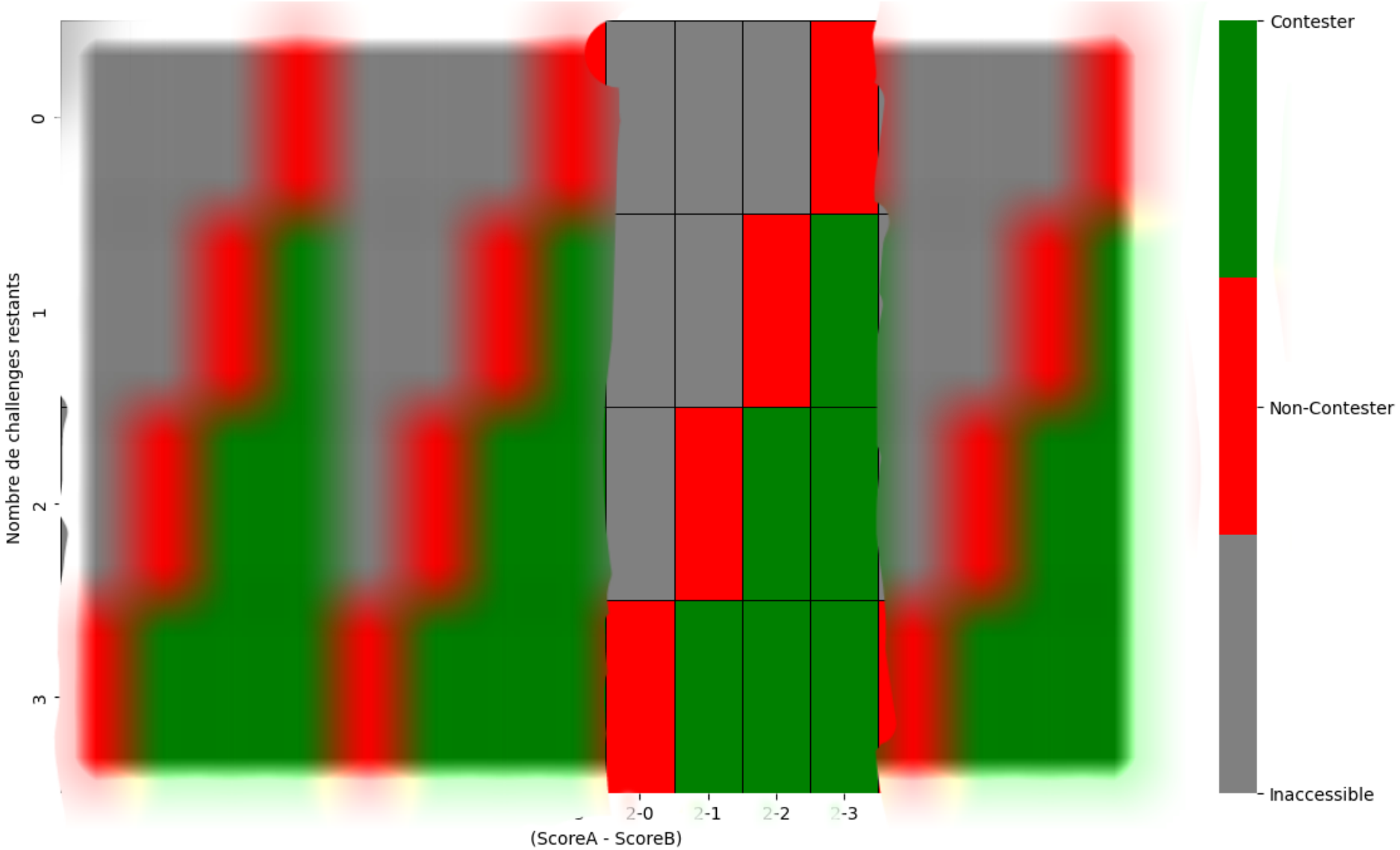


Figure – 7. Actions optimales dans un état de faible perception ; E3

Conclusion

- Le joueur doit toujours contester s'il pense qu'il a une très grande chance d'avoir raison
- Plus la perception de contester correctement est grande, mieux il vaut de contester
- Moins le joueur a des challenges, plus il doit être prudent et passif
- Plus l'adversaire se rapproche de la victoire, plus il devient necessaire de contester

Thank you !

YAHAYA A.K Saad

Annexe

FAQ :

- D'où viennent les 3 types d'états : E1, E2, E3 ?
- Démontrer les équations de Bellman.
- Comment les valeurs des probabilités pE1 $= 0.3$, pE2 $= 0.2$, b $= 0.8$, pE3 $= 0.5$, c $= 0.2$ affectent-elles le résultat final ?
- Quelles sont les fonctions du facteur d'amortissement γ ?

Le scoring

- **Point** : C'est l'unité de base au tennis.
  - 0 ← Love
  - 1 ← 15
  - 2 ← 30
  - 3 ← 40
  - 4 ← jeu .
- **Jeu** : Pour gagner un jeu, il faut inscrire au moins **4** points.
  A **4-4**, c'est le **Deuce**. Pour remporter le jeu, le joueur doit gagner **4** points consécutifs.
- **Set** : Le premier joueur a gagner **6** jeux avec une marge d'au moins **2** gagne le set. A **6-6**, un jeu décisif (**tiebreak**) est joué : le premeir a gagner **7** points avec marge d'au moins **2** remporte le set.
- **Match** : Il est généralement composé de **3** sets. Le premier à gagner **2** sets remporte le match.

Stats

	Australian Open		Wimbledon	
	Men's	Women's	Men's	Women's
Total number of challenges	436	194	428	191
Number of correct challenges	137	68	120	49
Number of incorrect challenges	299	126	308	142
Percentage overturned	31.42%	35.05%	28.04%	25.65%
Total points played	13726	8317	14906	8811

Figure – 8. Singles challenge summary : Australian Open and Wimbledon 2012.

	Men's	Women's
Total Number of Challenges	260	140
Number of Correct Challenges	92	44
Number of Incorrect Challenges	168	96
Percentage Overturned	35.38%	31.43%
Avg. Challenges per Match	5.65	4.24

Figure – 9. Singles challenge summary Wimbledon 2010

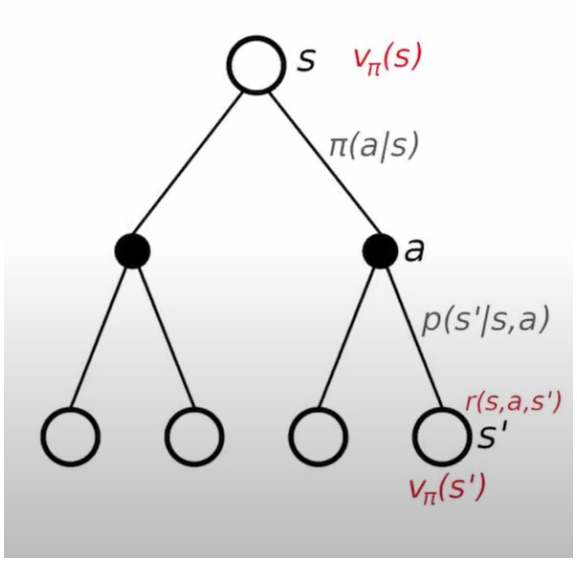
Equation de Belman

$$v_\pi(s) = \mathbb{E}(\mathcal{R}_t|s_t = s) = \mathbb{E}(\sum_{k=0}^{+\infty} \gamma^k r_{t+k+1}|s_t = s) = \mathbb{E}(r_{t+1} + \gamma\mathcal{R}_{t+1}|s_t = s)$$

Or

$$\mathbb{E}(r_{t+1}) = \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\mathcal{R}(s,a,s')$$

$$\mathbb{E}(\mathcal{R}_{t+1}|s_t = s) = \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')v_\pi(s')$$



fonction de valeur d'état

Les fonctions de valeur d'états peuvent s'ecrire plus s :

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}, q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')(\mathcal{R}(s,a,s') + \gamma v_\pi)$$

$$\forall s \in \mathcal{S}, v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} q_\pi(s,a)$$

Nouvelles expressions des equations optimales

Les equations de Bellman optimales deviennent :

$$\forall s \in \mathcal{S}, v^*(s) = \max_{a \in \mathcal{A}} q^*(s,a)$$

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}, q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')(\mathcal{R}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} q^*(s,a'))$$

fonction de valeur d'état d'action

```python
def compute_q_value_for_s_a(env, V, s, a, gamma):
    q = 0
    for (p_sPrime, sPrime, r_ss_a, done) in env.P[s][a]:
        q += p_sPrime * (r_ss_a + gamma * V[sPrime])
    return q
```

Policy iteration

Initialisation

```python
env=Environnement()
pi = {state: [0.5,0.5] for state in States}
V = {state: 0 for state in States}

gamma = 0.99 #facteur de remise du return, j'aurai pu prendre gamma = 1 car States est fini
epsilon = 0.00001 #seuil de similitude requis pour stopper les updates
```

Policy iteration

```python
def evaluate_policy(env, pi, V, gamma, epsilon):
    V_updated = copy.deepcopy(V)
    improved = True

    while True:
        delta = 0
        for s in States:
            V_new = 0
            nA = len(env.P[s])

            for a in range(nA):
                prob_a = pi[s][a]
                q_s_a = compute_q_value_for_s_a(env, V_updated, s, a, gamma)

                V_new += prob_a * q_s_a

            delta = max(delta, np.abs(V_new - V_updated[s]))
            V_updated[s] = V_new

        if(delta < epsilon):
            break

    if(np.array_equal(V, V_updated)):
        improved = False

    return V_updated, improved

def improve_policy(env, pi, V, gamma):
    for s in States:
        nA = len(env.P[s])
        q_s = np.zeros([nA, 1])
        for a in range(nA):
            q_s[a] = compute_q_value_for_s_a(env, V, s, a, gamma)

        best_a = np.argmax(q_s)
        pi[s] = np.eye(nA)[best_a]

    return pi
```

```python
i = 0
while True:
    i+=1

    V, improved = evaluate_policy(env, pi, V, gamma, epsilon)
    pi = improve_policy(env, pi, V, gamma)

    if(improved == False):
        print("Terminé après " + str(i) + " itérations.")
        break
```

```
Terminé après 5 itérations.
```

Création des états

Les états

```python
States=[]
Categories=['E1', 'E2', 'E3'] # les diff situation du joueur, tout comme leur probabilité d'occurence pE1, pE2, pE3

for i in range(5):
    for j in range(5):
        for q in range(4):
            for Eperception in Categories:

                if j >= 3-q and i+j < 8: # le joueur adverse a moins le nbre de challenge raté comme score, et le premier joueur a avoir 4 pts rempor
                    # done=False

                    if i==4 or j==4 : # critere de fin de jeu
                        # done = True
                        Eperception=None # pas besoin
                    s= ((i, j), q, Eperception)
                    if s not in States:  # enelever les repetitions
                        States.append(s)

```

L'environnement

```python
class Environnement():
    def __init__(self, pE1=0.3 , pE2=0.2, b=0.8, pE3=0.5, c=0.2):

        def proba( s, a, s_prime):
            ### 1e etape: calcul de proba pour passer au "score" de s_prime ###
            if s[2] == None : # l Eperception None est celle des etats terminaux
                return 0

            deltaS = [ s_prime[0][0] - s[0][0], s_prime[0][1] - s[0][1] ] # difference entre le score d'arrivé et départ
            deltaNbreChlge = s_prime[1] - s[1] # différence entre le nbr de challenge d'arrivé et départ

            if deltaS == [0,1]: # A perd le jeu

                if s[2] == 'E1' : # E1 est un etat gagant à coup sûr
                    p = 0
                elif a == 0 : # A n'a pas fait un challenge

                    if deltaNbreChlge != 0 : # nbreChlge diminue sans que A ait contesté, ABSURDE
                        p = 0
                    else:
                        p = 1 # A perd a coup sûr

                else: # A a fait un challenge

                    if deltaNbreChlge == 0 : # A a raté un challenge et nbreChlge = cst, ABSURDE
                        p = 0
                    elif s[2]  == 'E2' :
                        p = 1-b
                    elif s[2]  == 'E3' :
                        p = 1-c

            elif deltaS == [1,0] :# A gagne le jeu

                if s[2] == 'E1' : # E1 est un etat gagant à coup sûr
                    p = 1
                elif a == 0 : # A n'a pas fait un challenge
                    p = 0
                else:
                    if s[2]  == 'E2' :
                        p = b
                    elif s[2]  == 'E3':
                        p = c
```

```
                    p = c

            else: # transition impossible ( d'apres nos hypotheses, sinon, possible dans la réalité) # pas sur qu il y ait d autre, juste précaution
                p = 0

                ### 2e etape: multiplication par la proba de tomber dans l Eperception de s_prime ###

            if  s_prime[2]  == 'E1':# on met a jour p pour prendre en compte 'la Eperception' du prochain etat
                p*=pE1
            elif s_prime[2]  == 'E2':
                p*=pE2
            elif s_prime[2]  == 'E3':
                p*=pE3
            # sinon, le Eperception est None, donc etat terminal

            return p

            ### FIN de la def de Proba ###

        ### Remplissage du dictionaire de transition ###

        P = {}
        for s in States:
            i, j = s[0][0], s[0][1] # le score
            q = s[1] # challenges restants
            Eperception = s[2]
            nA = 2 # nombres d'actions disponibles

            dynamics_s = {} # le dico qui contiendra les infos de transition de chaque etat s

            ### Remplissage de dynamics_s ###

            if q == 0 or Eperception == 'E1' : # A ne peut plus challenger ou bien il est dans un etat gagnant
                nA = 1

            for a in range(nA):
                margeQ1 = 0 # intret dans le boucle ci apres
                margeQ2 = 2
                if a == 0:
                    margeQ2=1
                else:
                    margeQ1=1

                s_prime_list = [] # succeseurs de s
```

```
                s_prime_list = [] # succeseurs de s

                if i < 4 and j < 4: # les etats terminaux n'ont pas de successeurs

                    for t in range(margeQ1,margeQ2): # le petit delta q, il doit etre nul si a==0
                        for Eperception in Categories:

                            if i+1 < 4:
                                s_prime1 = ((i+1, j), q, Eperception) # on ne change pas q car A gagne dans ici
                                done1 = False
                                reward1 = 0
                            elif i+1 == 4:
                                s_prime1 = ((i+1, j), q, None)
                                reward1 = 100 # etat terminal gagnant
                                done1 = True
                            p1 = proba(s, a, s_prime1)

                            if j+1 < 4:
                                s_prime2 = ((i, j+1), q-t, Eperception)
                                done2 = False
                                reward2 = 0
                            elif j+1 == 4:
                                s_prime2 = ((i, j+1), q-t, None)
                                done2 = True
                                reward2 = -100
                            p2 = proba(s, a, s_prime2)

                            if (p1, s_prime1, reward1, done1) not in s_prime_list:   # il y a repetition avec les Eperception None
                                s_prime_list.append((p1, s_prime1, reward1, done1))
                            if(p2, s_prime2, reward2, done2) not in s_prime_list:
                                s_prime_list.append((p2, s_prime2, reward2, done2))

                dynamics_s.update({a:s_prime_list}) # attention, les etats terminaux n ont pas de succeseur

            ### FIN remplissage de dynamics_s ###

            P.update({s: dynamics_s})

        ### FIN Remplissage du dictionaire de transition ###

        self.P = P
```

Les graphes

Les types d'états

```python
import matplotlib.pyplot as plt

name = ['E1 : A gagne a coup sûr', 'E2 : A perd, forte Eperception de remporter un chlge', 'E3 : A perd, faible Eperception de remporter un chlge']
data = [0.3, 0.2, 0.5]

explode=(0.04, 0.02, 0.02)
plt.pie(data, explode=explode, labels=name, autopct='%1.1f%%', startangle=180, shadow=False, colors=['g','orange','red'])
plt.axis('equal')
plt.show()
```

le résultat

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Les données fournies
data = E3

# Préparer les combinaisons de score et le nombre de challenges restants
scores = sorted(set(key[0] for key in data.keys()))
challenges = sorted(set(key[1] for key in data.keys()))

# Créer une matrice pour stocker les décisions
decision_matrix = np.full((len(challenges), len(scores)), -1)  # Initialiser avec -1 pour les valeurs manquantes

# Remplir la matrice de décision
for key, value in data.items():
    score, challenges_left, _ = key
    decision = np.argmax(value)  # Obtenir l'indice de la décision (0 ou 1)
    score_idx = scores.index(score)
    challenges_idx = challenges.index(challenges_left)
    decision_matrix[challenges_idx, score_idx] = decision

# Créer les étiquettes pour les axes
score_labels = [f"{s[0]}-{s[1]}" for s in scores]

# Définir la palette de couleurs : gris pour les manquants, vert pour challenger (1)
cmap = sns.color_palette(["grey","red", "green"])

# Création de la heatmap
plt.figure(figsize=(14, 10))
ax = sns.heatmap(decision_matrix, annot=False, cbar=True, cmap=cmap, xticklabels=score_labels, yticklabels=challenges, linewidths=.5, linecolor='blac

# Personnalisation de la colorbar
colorbar = ax.collections[0].colorbar
colorbar.set_ticks([-1, 0, 1])
colorbar.set_ticklabels(['Manquant', 'Non-challenger', 'Challenger'])

plt.xlabel('ScoreA, ScoreB')
plt.ylabel('Nombre de challenges restants')
plt.title('Décision de challenger en fonction du score et du nombre de challenges restants')
plt.show()
```