Dossier : jeu de dominos

Nicolas Poulain

27 mars 2012

Table des matières

1 Règles du jeu de dominos

Le jeu de dominos est un jeu de société d'origine chinoise, utilisant 28 pièces (dans le cas d'un jeu "double-six"), les dominos. On peut adopter une des règles suivantes :

Règle

- Distribuer au hasard 10 des 28 dominos à chaque joueur.
- Celui qui a le domino le plus fort (ordre lexicographique) commence et pose celui-ci sur la table
- Chaque joueur pose tour à tour à l'une des extrémites du jeu sur la table un domino de sorte que les parties voisines ont le même nombre de points, constituant ainsi une chaîne.
- Le joueur qui ne peut pas jouer passe son tour, et on continue à jouer jusqu'à ce qu'un des joueurs se soit débarrassé de tous ses dominos, ou que le jeu soit complètement bloqué.
- À la fin du jeu, celui qui totalise le moins de points (la somme des points de l'ensemble des dominos) est le gagnant. On a donc tout intérêt à se débarrasser en premier des dominos valant beaucoup de points.

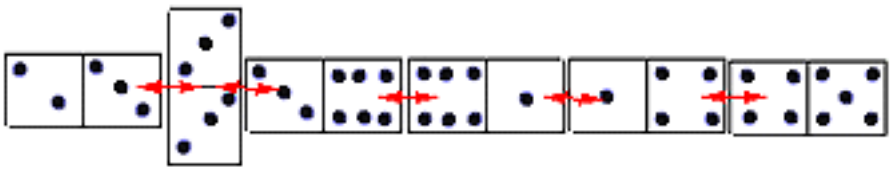

FIGURE 1 –

1.1 Le projet

Le but de ce projet est de comparer différentes stratégies de jeu en opposant deux joueurs virtuels et en leur faisant jouer un grand nombre de parties. Lesdonnées statistiques recueillies sur le nombre de victoires nous conduiront déterminer la meilleure méthode pour gagner à ce jeu.

2 Modélisation

Un domino peut être représenté par un vecteur colonne $\begin{pmatrix} x \\ y \end{pmatrix}$ ou par un nombre $10x + y$. Ainsi le jeu d'un joueur peut être représenté par la matrice (le tableau)

$J = \begin{pmatrix} 6 & 5 & 5 & 3 & 3 & 3 & 1 \\ 1 & 4 & 0 & 3 & 1 & 0 & 1 \end{pmatrix}$ ou $J = (61, 54, 50, 33, 31, 30, 11)$.

2.1 Premières fonctions

Ce sont des fonctions qui ne servent pas directement au déroulement de la partie de dominos mais qui permettent sa mise en place ou qui sont indispensables en tant qu'outils pour les fonctions avancées.

1. Écrire la fonction `creation_jeu` capable de donner l'ensemble des dominos de la boîte de jeu.

$$\begin{pmatrix} 6 & 6 & 6 & 6 & 6 & 6 & 6 & 5 & 5 & 5 & 5 & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 3 & 3 & 3 & 3 & 2 & 2 & 2 & 1 & 1 & 0 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & 5 & 4 & 3 & 2 & 1 & 0 & 4 & 3 & 2 & 1 & 0 & 3 & 2 & 1 & 0 & 2 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

2. Écrire une fonction `distribue` qui tire au hasard (et sans remise) une main contenant un nombre donné de dominos.
3. Écrire une fonction `est_avant` qui admet comme paramètres d'entrée deux dominos et qui renvoie `True` ou `False` selon que les deux dominos sont ou pas classés dans l'ordre décroissant. Par exemple `est_avant([6,2],[5,0])` renverrait `True`.
4. Écrire une fonction `tri_decr` qui trie un ensemble de dominos par ordre décroissant.

2.2 Les fonctions concernant le déroulement de la partie

Ces fonctions permettent d'initialiser la partie, d'enchaîner les tours de jeux, et de terminer la partie. Elles utilisent les fonctions citées précédemment à travers d'autres fonctions plus complexes. Les deux principales fonctions sont celle permettant de désigner le joueur qui posera le premier domino, et celle qui, grâce à une boucle, fait jouer les joueurs chacun leur tour.

1. Écrire une fonction `is_player1_first` qui indique si le joueur numéro 1 est ou pas celui qui commence la partie selon qu'il possède le domino le plus fort.
2. Écrire une fonction `possibilites` qui à partir de la chaîne de dominos se trouvant déjà sur la table et de la main d'un joueur, renvoie les choix possibles de dominos pour ce joueur. Par exemple , pour une table constituée de la chaîne de dominos [3,4],[4,4], et pour une main constituée des dominos {[2,3],[1,5],[4,6]}, la fonction reverrait les nombres 0 et 2 qui sont les positions des deux dominos pouvant être placés sur la table.
3. Écrire une fonction `positionne` qui à partir d'une table et d'un domino qui peut être placé sur cette table (à l'un des deux bouts) renvoie la nouvelle table (avec le domino correctement placé).

2.3 Les fonctions concernant les stratégies

La stratégie d'un joueur est déterminée par le choix du domino à jouer lorsque plusieurs possibilités s'offrent.

Programmer les quatre stratégies suivantes :

1. `strategie_pire` : joue le domino placé en dernier dans l'ordre lexicographique ;
2. `strategie_hasard` : joue un domino au hasard ;
3. `strategie_plus_de_points` : joue le domino qui vaut le plus de points ;
4. `strategie_plus_present` : joue le domino dont les valeurs sont les plus présentes dans la main (ainsi il garde plus de possibilités pour les tours à venir) ;

2.4 Resultats

En lançant un très grand nombre de parties entre deux stratégies à chaque fois, on espère obtenir une bonne estimation statistique pour comparer la qualité des stratégies.

La liste suivante donne pour chaque tournoi de 100000 parties, le nombre de nuls, et le nombre de victoires de chaque côté.

Stratégie	contre Stratégie	Victoires	Nuls	Défaites
`hasard`	`pire`	55470	5826	38704
`plus_present`	`pire`	61752	6411	31837
`plus_present`	`hasard`	51217	8398	40385
`plus_de_points`	`pire`	62743	5384	31873
`plus_de_points`	`hasard`	56270	8051	35679
`plus_de_points`	`plus_present`	51666	9196	39138

Ainsi on constate que comme on s'y attendait la stratégie `pire` est la moins efficace et que la stratégie `hasard` se place deuxième. Ce qui était moins évident a priori, c'est que la stratégie élaborée `plus_present` consistant à tenter de conserver le plus de possibilités est finalement nettement moins intéressante que `plus_de_points` qui propose de se débarasser au plus vite des dominos les plus gros.

```
--------Nouvelle partie---------
La main du Joueur 1 :  [[6, 6], [6, 2], [5, 4], [5, 3], [4, 2], [4, 1], [4, 0], [3, 2], [2, 0], [1, 1]]
La main du Joueur 2 :  [[6, 4], [5, 5], [5, 0], [4, 4], [4, 3], [3, 1], [3, 0], [2, 2], [1, 0], [0, 0]]

Joueur 1 joue :  [[6, 6]]
Joueur 2 joue :  [[4, 6], [6, 6]]
Joueur 1 joue :  [[2, 4], [4, 6], [6, 6]]
Joueur 2 joue :  [[2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 1 joue :  [[0, 2], [2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 2 joue :  [[0, 0], [0, 2], [2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 1 joue :  [[4, 0], [0, 0], [0, 2], [2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 2 joue :  [[3, 4], [4, 0], [0, 0], [0, 2], [2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 1 joue :  [[2, 3], [3, 4], [4, 0], [0, 0], [0, 2], [2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 2 passe.
Joueur 1 joue :  [[6, 2], [2, 3], [3, 4], [4, 0], [0, 0], [0, 2], [2, 2], [2, 4], [4, 6], [6, 6]]
Joueur 2 passe.
Joueur 1 passe.
La main du Joueur 1 :  [[5, 4], [5, 3], [4, 1], [1, 1]]
La main du Joueur 2 :  [[5, 5], [5, 0], [4, 4], [3, 1], [3, 0], [1, 0]]
le joueur 1 gagne.
--------Nouvelle partie---------
La main du Joueur 1 :  [[6, 6], [6, 4], [6, 3], [5, 2], [5, 1], [5, 0], [4, 4], [4, 3], [4, 1], [3, 1]]
La main du Joueur 2 :  [[6, 2], [6, 1], [6, 0], [5, 5], [5, 4], [4, 0], [3, 3], [3, 0], [2, 2], [2, 1]]

Joueur 1 joue :  [[6, 6]]
Joueur 2 joue :  [[0, 6], [6, 6]]
Joueur 1 joue :  [[0, 6], [6, 6], [6, 4]]
Joueur 2 joue :  [[3, 0], [0, 6], [6, 6], [6, 4]]
Joueur 1 joue :  [[3, 0], [0, 6], [6, 6], [6, 4], [4, 4]]
Joueur 2 joue :  [[3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4]]
Joueur 1 joue :  [[1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4]]
Joueur 2 joue :  [[2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4]]
Joueur 1 joue :  [[2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1]]
Joueur 2 joue :  [[2, 2], [2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1]]
Joueur 1 joue :  [[2, 2], [2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1], [1, 5]]
Joueur 2 joue :  [[2, 2], [2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1], [1, 5], [5, 4]]
Joueur 1 joue :  [[2, 2], [2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1], [1, 5], [5, 4], [4, 3]]
Joueur 2 joue :  [[6, 2], [2, 2], [2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1], [1, 5], [5, 4], [4, 3]]
Joueur 1 joue :  [[3, 6], [6, 2], [2, 2], [2, 1], [1, 3], [3, 3], [3, 0], [0, 6], [6, 6], [6, 4], [4, 4], [4, 1], [1, 5], [5, 4], [4, 3]]
Joueur 2 passe.
Joueur 1 passe.
La main du Joueur 1 :  [[5, 2], [5, 0]]
La main du Joueur 2 :  [[6, 1], [5, 5], [4, 0]]
le joueur 1 gagne.
```

2.5 Programme Python

```
#!/usr/bin/python
import random

def creation_jeu(max=6):
  """Cree la boite de jeu avec l'ensemble des dominos
  qui seront distribues
  """
  jeu=[]
  for i in range(max,-1,-1):
    for j in range(i,-1,-1):
      jeu = jeu + [[i,j]]
  return jeu

def distribue(jeu,n=10):
  """Tire dans jeu (sans remise) une main de n dominos"""
  main = []
  for i in range(n):
    r = random.randint(0,len(jeu)-1)
    main = main + [ jeu[r] ]
    jeu.pop(r)
  return main

def est_avant(d,e):
  """Verifie si deux dominos sont dans l'ordre lexicographique
  - Exemples :
  >>> est_avant( [2,3], [1,2] ); est_avant( [2,3], [2,4] )
  True
  False
  """
  if d[0]>e[0] or ( d[0]==e[0] and d[1]>e[1] ):
    return True
  return False

def tri_decr(player):
  """Trie les dominos du joueur dans l'odre
  decroissant lexocographique
  - Exemple :
  >>> tri_decr([ [1,2], [3,4], [1,3], [2,0] ])
  [[3, 4], [2, 0], [1, 3], [1, 2]]
  """
  player = sorted(player)
  player.reverse()
  return player

def is_player1_first(pl1,pl2):
  """Le joueur 1 a-t-il une meilleure main que le joueur 2 ?"""
  if est_avant(pl1[0],pl2[0]):
   return True
  return False

def possibilites(table,pl):
  """Donne, la liste des dominos de pl qui peuvent
  etre places sur la table
  >>> possibilites([[3,4],[4,4]], [[2,3],[1,5],[4,6]])
  [0, 2]
  """
  possbl = []
  x=table[0][0]
```

```
  y=table[-1][1]
  for i in range(len(pl)):
    if pl[i][0]==x or pl[i][1]==x or pl[i][0]==y or pl[i][1]==y:
      possbl = possbl + [ i ]
  return possbl

def positionne(domino,table):
  """Positionne le domino correctement sur la table"""
  if domino[0]==table[0][0]:
    table = [ [ domino[1],domino[0] ] ] + table
  elif domino[1]==table[0][0]:
    table = [ [ domino[0],domino[1] ] ] + table
  elif domino[0]==table[-1][1]:
    table = table + [ [ domino[0],domino[1] ] ]
  else:
    table = table + [ [ domino[1],domino[0] ] ]
  return table

def strategie_pire(table,player,possbl):
  """Place sur la table le dernier domino dans l'ordre lexicographique"""
  table = positionne(player[ possbl[len(possbl)-1] ], table)
  player.pop(possbl[len(possbl)-1])
  return table,player

def strategie_hasard(table,player,possbl):
  """Place sur la table un domino au hasard"""
  ind = random.randint(0,len(possbl)-1)
  table = positionne(player[ possbl[ind] ], table)
  player.pop(possbl[ind])
  return table,player

def strategie_plus_de_points(table,player,possbl):
  """Place sur la table le domino qui vaut le plus de points"""
  # Repere l'indice du meillleur domino
  max = -1
  for i in range(len(possbl)):
    val = player[possbl[i]][0]+player[possbl[i]][1]
    if val > max:
      ind = i
      max = val
  # place le domino sur la table
  table = positionne(player[ possbl[ind] ], table)
  player.pop(possbl[ind])
  return table,player

def strategie_plus_present(table,player,possbl):
  """Place sur la table le domino dont la moyenne de presence des deux valeurs
  est la plus elevee"""
  # Construit le tableau des effectifs des valeurs presentes dans la main
  effectifs = [0, 0, 0, 0, 0, 0, 0]
  for i in range(len(player)):
    effectifs[ player[i][0] ] = effectifs[ player[i][0] ] + 1
    effectifs[ player[i][1] ] = effectifs[ player[i][1] ] + 1
  # Repere l'indice du meilleur domino
  max_moyenne = -1
  for i in range(len(possbl)):
    moyenne = effectifs [ player[possbl[i]][0] ] + effectifs [ player[possbl[i]][1] ]
    if moyenne > max_moyenne:
      max_moyenne = moyenne
      ind = i
  # place le domino sur la table
```

```
  table = positionne(player[ possbl[ind] ], table)
  player.pop(possbl[ind])
  return table,player

def who_wins(player1,player2):
  somme_points_p1 = 0
  for i in range(len(player1)):
    somme_points_p1 = somme_points_p1 + player1[i][0] + player1[i][0]
  somme_points_p2 = 0
  for i in range(len(player2)):
    somme_points_p2 = somme_points_p2 + player2[i][0] + player2[i][0]
  if somme_points_p1 < somme_points_p2:
    return 1
  elif somme_points_p1 > somme_points_p2:
    return 2
  else:
    return 0

if __name__ == "__main__":

  resultats_tournoi = [0, 0, 0]
  for i in range(10000):
    jeu = creation_jeu()
    player1 = tri_decr(distribue(jeu))
    player2 = tri_decr(distribue(jeu))
    print "--------Nouvelle partie---------"
    print "La main du Joueur 1 : ",player1
    print "La main du Joueur 2 : ",player2
    print ""
    # Initialisation de la partie
    if is_player1_first(player1,player2):
      table = [ player1[0] ]
      print "Joueur 1 joue : ", table
      player1.pop(0)
      a_qui_le_tour = 2
    else:
      table = [ player2[0] ]
      print "Joueur 2 joue : ", table
      player2.pop(0)
      a_qui_le_tour = 1

    # Les tours de jeu :
    # ils vont durer tant que les deux joueurs possedent des dominos
    # et qu'ils n'ont pas consecutivement passe leur tour.
    cpt_passe = 0
    while cpt_passe<2 and len(player1)>0 and len(player2)>0:
      if a_qui_le_tour==1:
        possbl = possibilites(table,player1)
        if len(possbl)==0:
          print "Joueur 1 passe."
          cpt_passe = cpt_passe + 1
        else:
          table,player1 = strategie_plus_present(table,player1,possbl)
          cpt_passe = 0
          print "Joueur 1 joue : ", table
        a_qui_le_tour = 2
      else:
        possbl = possibilites(table,player2)
        if len(possbl)==0:
          print "Joueur 2 passe."
          cpt_passe = cpt_passe + 1
```

```
            else:
              table,player2 = strategie_hasard(table,player2,possbl)
              cpt_passe = 0
              print "Joueur 2 joue : ", table
            a_qui_le_tour = 1

        # Fin de partie : on affiche ce qu'il reste a chacun.
        print "La main du Joueur 1 : ",player1
        print "La main du Joueur 2 : ",player2
        winner = who_wins(player1,player2)
        if winner == 0:
          print "Partie nulle."
        else:
          print "le joueur",winner,"gagne."
        resultats_tournoi[winner] = resultats_tournoi[winner] + 1

    print resultats_tournoi
```

Le code est disponible sur http://code.google.com/p/npoulain.