Aabu

Geste de savoir

# Combien y a-t-il de positions différentes au morpion ?

12 décembre 2020

# Table des matières



Dans mon précédent billet , je montrais combien de positions il existait au morpion. Cependant, on peut voir que certaines sont équivalentes du point de vue du jeu, car elles présentent des pions qui jouent des rôles similaires les uns par rapport aux autres.

Par exemple, le premier joueur n'a vraiment que trois options: jouer au centre, jouer dans un coin, jouer sur une arête, vu que tous les coins et arêtes sont identiques à ce point-là du jeu. Ensuite, le deuxième joueur a similairement des choix réels restreints: si le premier joueur a joué au centre, le deuxième ne peut jouer que dans un coin ou une arête, encore une fois indistinguables à ce stade du jeu.

Combien y a-t-il alors de positions non-équivalentes?

## 1. Réduction du nombre de positions par symétrie

Intuitivement, on se rend compte qu'on peut regarder une grille de morpion en tournant autour sans changer la partie en cours, de même qu'en la regardant dans un miroir.

Tourner ou regarder une réflexion permet de voir 8 parties symétriques. Il n'y en a pas plus, car une grille de morpion est un carré et à ce titre, on peut faire au plus 8 symétriques différent, en procédant par rotations et réflexions.

Pour le jeu de morpion cela nous permet d'obtenir un minorant du nombre de position en divisant par huit le nombre de positions obtenu dans mon précédent billet. C'est bien un minorant, car certaines positions sont invariantes selon certaines symétries et il n'y a donc pas 8 symétriques différents, mais moins.

| Coups | Minorant |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 9 |
| 3 | 31 |
| 4 | 94 |

| | |
|---|---|
| 5 | 157 |
| 6 | 190 |
| 7 | 142 |
| 8 | 48 |
| 9 | 9 |
| Total | 681 |

## 2. Nombre exact de positions non-équivalentes

Je n'ai malheureusement pas trouvé de méthode astucieuse pour compter les positions différentes sans les énumérer. Heureusement, j'ai pu modifier légèrement le programme de vérification de mon précédent billet pour éliminer du compte les positions équivalentes!

Le programme est ci-dessous.

Contenu masqué n°1

On obtient le tableau suivant.

| Coups | Nombre exact de positions |
|---|---|
| 0 | 1 |
| 1 | 3 |
| 2 | 12 |
| 3 | 38 |
| 4 | 108 |
| 5 | 174 |
| 6 | 204 |
| 7 | 153 |
| 8 | 57 |
| 9 | 15 |
| Total | 765 |

Une réalisation indépendante donne le même résultat. Youpi! En prime, le programme du billet précédent étant correct, on peut avoir une confiance assez grande dans le résultat.

## 3. Relation d'équivalence mathématique

Tel que défini auparavant, «être stratégiquement équivalent» est une relation d'équivalence au sens mathématique, autrement dit un genre d'égalité.

Les propriétés (vérifiées ici) sont les suivantes:

— symétrie: si une partie est équivalente à une autre, alors on peut inverser les rôles et c'est toujours vrai;
— transitivité: on peut faire des chaînes d'équivalence, et les extrémités seront équivalentes;
— réflexivité: toute position est équivalente à elle-même.

C'est d'ailleurs en redéfinissant l'égalité dans le programme de mon précédent billet qu'on obtient le résultat du présent billet.

Dans un sens, ce la revient à compresser l'ensemble des positions. C'est pratique notamment pour écrire des programmes cherchant à jouer stratégiquement au jeu, car cela fait moins de cas à analyser. Il suffit de savoir reconnaître la situation symétrique connue, trouver le coup à jouer, puis le transposer dans la position originale.

---

Et voilà! Il y a encore et toujours beaucoup à dire sur les mathématiques du morpion, mais… ce sera pour une prochaine fois.

*Miniature du billet: d'après une illustration d'une partie de morpion par Symode09, domaine public (source ).*

## Contenu masqué

### Contenu masqué n°1

```
P1 = 1
P2 = -1
EMPTY = 0

class Position:
    def __init__(self, pos_vec):
        self.pos_vec = pos_vec

    def allowed_moves(self):
        if self.is_endgame():
            return []
        else:
            return [i for i in range(len(self.pos_vec)) if
                self.pos_vec[i] == EMPTY]

    def current_player(self):
```

```
        if len([p for p in self.pos_vec if p == EMPTY]) % 2 == 0:
            return P2
        else:
            return P1

    def next_player(self):
        if self.current_player() == P1:
            return P2
        else:
            return P1

    def play(self, player, move):
        new_pos_vec = [p for p in self.pos_vec]
        new_pos_vec[move] = player
        return Position(new_pos_vec)

    def is_endgame(self):
        p = self.current_player()
        if ((self.pos_vec[0] == p and self.pos_vec[1] == p and
            self.pos_vec[2] == p)
           or (self.pos_vec[3] == p and self.pos_vec[4] == p and
               self.pos_vec[5] == p)
           or (self.pos_vec[6] == p and self.pos_vec[7] == p and
               self.pos_vec[8] == p)
           or (self.pos_vec[0] == p and self.pos_vec[3] == p and
               self.pos_vec[6] == p)
           or (self.pos_vec[1] == p and self.pos_vec[4] == p and
               self.pos_vec[7] == p)
           or (self.pos_vec[2] == p and self.pos_vec[5] == p and
               self.pos_vec[8] == p)
           or (self.pos_vec[0] == p and self.pos_vec[4] == p and
               self.pos_vec[8] == p)
           or (self.pos_vec[2] == p and self.pos_vec[4] == p and
               self.pos_vec[6] == p)
          ):
           return True
        elif [p for p in self.pos_vec if p == EMPTY] == []:
           return True
        else:
            return False

    def __hash__(self):
        return self.pos_vec[4]

    def __eq__(self, other):
        def reorder(order):
            return [self.pos_vec[i] for i in order]
        d1 = reorder([0, 1, 2, 3, 4, 5, 6, 7, 8])  # OK
        d2 = reorder([6, 3, 0, 7, 4, 1, 8, 5, 2])  # OK
        d3 = reorder([8, 7, 6, 5, 4, 3, 2, 1, 0])  # OK
```

*Contenu masqué*

```
        d4 = reorder([2, 5, 8, 1, 4, 7, 0, 3, 6])  # OK
        i1 = reorder([2, 1, 0, 5, 4, 3, 8, 7, 6])  # OK
        i2 = reorder([6, 7, 8, 3, 4, 5, 0, 1, 2])  # OK
        i3 = reorder([8, 5, 2, 7, 4, 1, 6, 3, 0])  # OK
        i4 = reorder([0, 3, 6, 1, 4, 7, 2, 5, 8])  # OK
        return (d1 == other.pos_vec or d2 == other.pos_vec or d3 ==
            other.pos_vec or d4 == other.pos_vec
                or i1 == other.pos_vec or i2 == other.pos_vec or i3
                    == other.pos_vec or i4 == other.pos_vec)

    def size(self):
        return len([c for c in self.pos_vec if c == P1 or c == P2])


def generate_next_positions(current_position):
    allowed_moves = current_position.allowed_moves()
    player = current_position.next_player()
    next_positions = []
    for move in allowed_moves:
        np = current_position.play(player, move)
        next_positions.append(np)
    return next_positions


def generate_positions():
    empty_board = Position([EMPTY, EMPTY, EMPTY, EMPTY, EMPTY,
        EMPTY, EMPTY, EMPTY, EMPTY])
    all_positions = {empty_board}
    def gp(current_position):
        nps = generate_next_positions(current_position)
        for p in nps:
            if p not in all_positions:
                all_positions.add(p)
                gp(p)
    gp(empty_board)
    return all_positions

def count_positions():
    all_positions = generate_positions()
    return len(all_positions)


def count_positions(n=None):
    all_positions = generate_positions()
    if n is None:
        return len(all_positions)
    else:
        return len([p for p in all_positions if p.size() == n])

p = count_positions()
```

*Contenu masqué*

```
print(p)
print("---")
for i in range(10):
    print(count_positions(i))
```

Retourner au texte.