Optimisation stratégique du jeu Dumble : une approche par apprentissage par renforcement.

Lagier Hadrien

June 24, 2025

Contents

1 Introduction

Le jeu Dumble est un jeu de cartes rapide et stratégique [1]. Utilisant un jeu de 52 cartes. Les règles seront expliquées un peu plus tard 2.1. Le but de cette recherche est de trouver une manière efficace de gagner le jeu. Nous utiliserons l'intelligence artificielle pour nous aider a percer les mystères du jeu. Grâce au deep learning l'entrainement d'un modèle permettant a une IA de jouer contre elle même et apprendre de ses erreurs afin qu'elle établisse la meilleure stratégie a adopter.

2 Présentation du Jeu Dumble

2.1 Règles du jeu

Les règles sont tirées de [1]. Pour de 2 à 5 joueurs avec 1 jeu de 52 cartes et une possibilité de rajouter des jokers. Le but du jeu est de faire **le moins de points possible** sachant que chaque carte possède une valeur :

- Joker : 0 points
- As : 1 points
- 2 - 9 : numéro de carte associé (ex: 2 -> 2 points)
- 10 - Valet - Dame - Roi : 10 points

Dans un jeu de 52 cartes (ou plus avec les jokers) le mélanger à chaque tour avant de distribuer 7 cartes à chaque joueur. Les cartes restantes forment la pioche, 1 carte est retournée pour démarrer la défausse.

Les joueurs jouent chacun leur tours. Le joueur actuel a deux choix :

- Dire "Dumble !" si et seulement si la somme de la main est infèrieur ou égale à 10.
- Défausser de 1 à 4 cartes suivant les règles suivantes : 1 cartes, 1 paires, 1 brelan, 1 carré ou 1 tierce (suite de la même couleur)

A chaque tour le joueur pioche 1 cartes. Soit dans la défausse, qui doit **forcément** être la main du joueur le précédent. Soit dans la pioche du jeu.

Il existe une règle qui contre un "Dumble" d'un joueur. Le "contre-dumble" peut être utilisé par tous les autres joueurs. Si le joueur qui a annoncé le contre a moins de points que celui qui a annoncé le "Dumble", le joueur qui contre gagne la partie. Sinon le joueur qui a annoncé "Dumble" gagne.

2.2 Caractéristiques du jeu

Dans un premier temps nous pouvons établir une stratégie naïve étant la suivante : pour minimiser nos points, mieux vaut avoir le moins de cartes et de se débarasser des cartes ayant une valeure la plus haute. Il faut donc essayer de faire des combinaisons de cartes (paires, brelan...) permettant de supprimer le plus de cartes possible.

3 Modélisation du Jeu pour l'apprentissage par renforcement

Nous allons donc implémenter le jeu dumble en **Python** qui semble être une bonne alternative entre l'environnement pour notre IA et la simplicité du langage permettant le codage rapide et efficace du jeu.

3.1 Représentation de l'environnement

Nous créeons une classe CardGameEnv qui utilise l'API Gymnasium (Gym) [2]. Notre classe a plusieurs méthodes permettant au bon fonctionnement de l'IA.

- La méthode init : permet l'initialisation de notre IA dans la classe.

```
def __init__(self, verbose=False):
    super().__init__()
    self.turn_cnt = 0 # compteur de tour
    self.verbose = verbose # pour le debug
    self.game = Game() # init du jeu
    self.action_space = spaces.MultiBinary(8)
        # 7 cartes + 1 pour draw choice
    self.observation_space =
        spaces.Box(low=0, high=20, shape=(34,), dtype=np.int32)
```

Nous optons pour une action space MultiBinary de 8. C'est à dire un choix binaire pour les 7 cartes de notre jeu (1 représente l'action de selection et 0 la non sélection). Et 1 pour le choix entre la pioche ou la défausse (0 pour la pioche 1 pour la défausse). Donc notre IA a chaque tour de jeu aura au maximum 8 choix a faire.

Pour ce qui est de l'espace d'observation, notre IA doit être capable de reconnaitre les différentes main possible. Par exemple si ma main contient une paire, l'IA doit être capable de l'observer. Elle doit aussi être capable d'observer notre main avec les cartes dedans. Encoder de cette façon dans un vecteur de taille 34 :

```
def _encode_card(self, card):
    valeur = self.valeur_to_int(card.valeur) if card else 0
    couleur = self.couleur_to_int(card.couleur) if card else 0
    return (valeur, couleur)
```

On a donc un vecteur d'observation :

```
def _get_obs(self):
    player = self.game.current_player()
    ordered_hand =
        sorted(
            player.hand.hand,
            key=lambda c: self.valeur_to_int(c.valeur)
        )
```

```
        hand = ordered_hand[:14]

        flat_obs = []
        for card in hand:
            flat_obs.extend(self._encode_card(card))

        while len(flat_obs) < 28:
            flat_obs.extend((0, 0))  # padding

        valid_combos = self.find_valid_combinations(hand)

        has_pair = any(self.is_pair(combo)
            for combo in valid_combos)
        has_brelan = any(self.is_three_of_a_kind(combo)
            for combo in valid_combos)
        has_straight = any(self.game.is_straight(combo)
            for combo in valid_combos)
        has_carre = any(self.is_four_of_a_kind(combo)
            for combo in valid_combos)
        has_combo_10_plus =
            any(self.game.get_sum_card_select(combo) >= 10
                for combo in valid_combos)

        flat_obs.extend([
            int(has_pair),
            int(has_brelan),
            int(has_carre),
            int(has_straight),
            int(has_combo_10_plus),
            len(hand)
        ])

        return np.array(flat_obs, dtype=np.int32)
```

- La méthode step : permet la récompense a notre IA (cf. 3.2.2). C'est cette fonction qui permet a l'IA de jouer au jeu.
- La méthode reset : remet la classe a l'état initial, pour recommencer le jeu par exemple.

3.2 Choix du cadre RL

3.2.1 Adversaire

Pour que notre IA progresse vite et apprenne bien, il est essentiel d'avoir un adversaire "qui joue bien". On utilise donc la stratégie établie initialement 2.2 qui semble être une façon simple et efficace de finir la partie.

```
    def simulate_adversary_turn(self):
```

```
adversary = self.game.current_player()
hand = adversary.hand.hand

valid_combos = []
for r in range(1, min(5, len(hand) + 1)):
    for combo in combinations(hand, r):
        if self.game.can_play(list(combo)):
            valid_combos.append(list(combo))

if valid_combos:
    def combo_score(combo):
        return (len(combo), self.game.get_sum_card_select(combo))

    best_combo = max(valid_combos, key=combo_score)

    adversary.hand.delete(best_combo)
    self.game.bin.insert(0, best_combo)

self.game.current_player_pioche()
```

Ce que fais la fonction c'est qu'elle classe les mains qui peuvent être jouée en fonction de la main du joueur. Elle prend ensuite la combinaison qui a la plus grosse somme. Elle preferera prendre une main qui possède le plus de carte.

3.2.2 Récompense

Les récompenses données a l'IA fonctionne de la manière suivante. Pour une bonne action on récompense et pour une mauvaise on punit.

Si l'IA gagne on récompense fortement :

```
if self.game.is_finished():
    if self.verbose:
        print("L'IA a dumble")
    reward += 200
    reward += max(0, 30 - self.turn_cnt) * 2
    return self._get_obs(), reward, True, {}
```

On récompense évidemment la victoire mais aussi le nombre de tours. Plus l'IA finit vite la partie plus elle est récompensée. Au contraire si l'IA joue trop longtemps, on force l'arrêt et on punit :

```
if self.turn_cnt > 30:
    reward -= 100
    return self._get_obs(), reward, True, {}
```

Si l'IA na pas gagné elle peut donc jouer des cartes. On va récompenser l'IA si elle joue une selection de cartes avantageuses. Surtout les combinaisons a plusieurs cartes.

```
if self.game.is_straight(card_select):
    reward += 80
```

```
elif self.is_pair(card_select):
    reward += 50
elif self.is_three_of_a_kind(card_select):
    reward += 60
elif self.is_four_of_a_kind(card_select):
    reward += 90

# si une carte simple
# récompense la valeur des cartes selectionnées
sum_card_select = self.game.get_sum_card_select(card_select)
if sum_card_select <= 3:
    reward -= 5
else:
    reward += sum_card_select
```

Au contraire si l'IA a joué une selection de cartes impossible, comme par exemple : 7 coeur, 8 trêfle. Qui n'est pas une combinaison possible. On punit moins fortment qu'une defaite ; pour faire comprendre que ce n'est pas ce qu'on attend d'elle.

```
if not self._is_valid_action(action):
    reward -= 30
    return self._get_obs(), reward, False, {}
```

4 Apprentissage et Analyse des Résultats

4.1 Méthodologie expérimentale

L'élaboration de la fonction de récompense a été menée de manière itérative et empirique. Nous avons adopté une approche expérimentale, consistant à tester différentes configurations de récompenses et de pénalités, puis à analyser leur impact sur le comportement de l'agent et sur sa performance globale. Lorsque les signaux de récompense s'avéraient insuffisamment informatifs, trop faibles ou menaient à des comportements sous-optimaux, ceux-ci étaient ajustés en conséquence. Cette démarche nous a permis d'affiner progressivement la fonction de récompense pour mieux guider l'apprentissage de l'agent dans des situations complexes.

4.2 Stratégies émergentes

4.3 Optimisation des stratégies

5 Discussion

5.1 Limites de la modélisation

5.2 Apport de l'approche RL

5.3 Perspectives

6 Conclusion

References

[1] Yann Cochard. *Règles du Dumble : Petit jeu de carte rapide*. 2019. URL: https://yanncochard.com/458-regles-du-dumble-petit-jeu-de-carte-rapide/.

[2] Gymnasium. *Gymnasium Documentation: Python API for Reinforcement Learning*. 2023. URL: https://www.gymlibrary.dev/.