進捗報告

1 今週やったこと

- python でカードゲームの再実装
- 実験環境の構築
- 簡単な条件下の基礎実験

2 Python で実装したカードゲーム

Python だけでカードゲームの対戦を行えるように, Python で簡単なカードゲームを作成した. プレイヤーは HP を持ち, カードは攻撃力と HP をそれぞれ持つ用に設定した. またカードが敵カードに攻撃する際には, 敵カードから攻撃を受けないようにした. カードのコスト, それに関するマナコスト, カードに付随する特殊効果などは未実装である. 実装したコードは `https://github.com/1g-hub/Nishimura/tree/main/Sources`

3 実験環境の構築

自作ゲームを学習環境として利用する方法として OpenAI Gym を採用した. OpenAI Gym を採用した理由は 2 つあり, 1 つは OpenAI Gym 用のラッパーや自作環境は多くの人が作成・公開しているためインターネット上に参考になる資料が多くあるため, もう 1 つは OpenAI Gym に対応している強化学習フレームワークが多く自作環境と学習モデル構築部分を分離して作成することができるためである [1]. OpenAI Gym を用いて自作環境を作成する際, エージェントが取りうる行動空間, 状態空間, 報酬の 3 つを定義する必要がある. 基礎実験でどのように定義したかは以下の節で示す.

また, 強化学習フレームワークとして keras を採用した. 後々細かいチューニングが必要であると思われるが, 実験を回す際に簡単にモデルを作成することができるため採用した.

4 基礎実験

構築した環境がうまく機能しているかどうか確かめるため, 先週のゼミでアドバイスを頂いたように簡単な条件下で実験した.

4.1 条件

各プレイヤーについて 3 枚場に出るカードを用意し, Play (手札から場に出す動作), Draw (デッキからカードを引く) といった動作を考慮せず先攻と後攻に分かれて 1 ターンの勝負を行う. 学習する対象は先攻のプレイヤーとして, 後攻のプレイヤーは先攻プレイヤーの場における最も攻撃力が高いカードを優先的に狙うように設定した. カードの数値はカードゲームでは一般的に先手が有利であることを考慮して適当に定めた. また, 各プレイヤーの HP は 20 とした. 条件下におけるカードゲームのイメージを図 1 に示す.

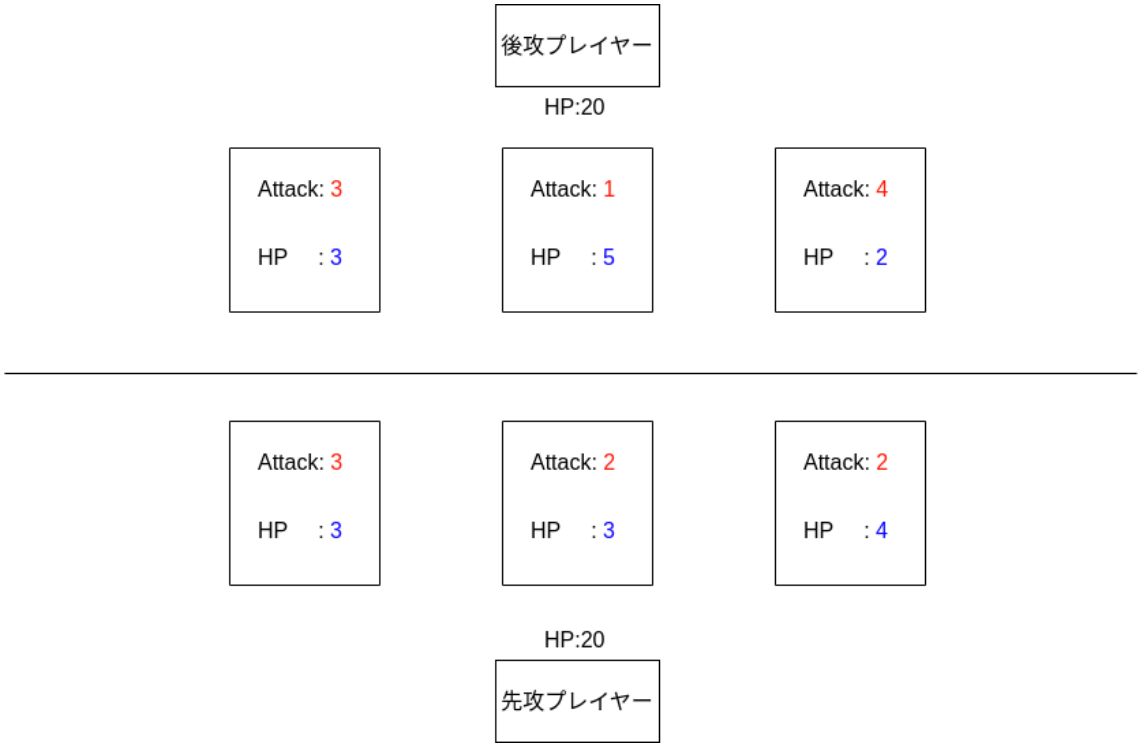

図 1: 基礎実験における条件設定のイメージ図

4.2 自作環境下におけるエージェントの行動空間の定義

OpenAI Gym では, エージェントの行動を離散値で与えることが多い. そのためにはプレイヤーの行動を数え上げ行動空間を定義する必要がある. 本実験では,

- プレイヤーのカードが相手プレイヤーを攻撃する … 3 通り
- プレイヤーのカードが敵プレイヤーのカードを攻撃する … 3*3 = 9 通り

の合計 12 通りと定義して実装した.

4.3 状態空間の定義

プレイヤーが観測できる情報は定義により様々に考えられるが本実験では,

- プレイヤー 2 名の HP
- プレイヤー双方の場に出ているカード 6 枚の攻撃力と HP
- ターン中行動したかどうか (先攻プレイヤーのカードのみ)

の合計 17 個のパラメータからなる状態空間を定義して実装した.

4.4 報酬の定義

学習における 1 ステップはエージェントの 1 行動に対応している. 報酬の定義として参考文献を参考に以下の式を立てた [2].

$$
\begin{aligned}
f(\text{盤面}) =& \alpha * (\text{味方カードの評価値の和}) + \beta * (\text{ターン中に削った敵プレイヤーの体力}) \\
& - \gamma * (\text{敵カードの評価値の和}) - \delta * (\text{ターン中に削られた自プレイヤーの体力}) \\
& where \qquad \alpha, \beta, \gamma, \delta \in \mathbb{R}^{+}
\end{aligned}
$$

$$reward = \begin{cases} 1 & (f(\text{盤面}) \geqq 0) \\ -1 & (f(\text{盤面}) < 0) \end{cases}$$

なお, 本実験では $\alpha = 2.0$, $\beta = 1.5$, $\gamma = 1.0$, $\delta = 5.0$ とした. またカードの評価値はカードの攻撃力と残り HP の和と定義している. 上式の $reward$ は 1 エピソード, すなわち両プレイヤーの行動終了後に計算される. ターン中では, 基本は $reward = 0$ とし, 同じ行動が続いた時は不適切として $reward = -1.0$ としている.

4.5 作成したモデル, エージェント

keras , keras-rl を用いて, ネット上のサンプルコード [3] を参考に Deep Q Network を実装した. ネットワークの概形を 図 2 に示す. 観測できる環境の次元を入力としてこのモデルを通して行動空間の次元の出力を得る.

エージェントの設定として, Exprience Replay のメモリの上限サイズ $limit = 1000000$, 観測の連結サイズ $window\_length = 1$ としている. また, 行動方策として ϵ-greedy 法を用いており $\epsilon = 0.1$ としている. 最適化関数として Adam を用いており, $learning\_rate = 1e - 3$ と設定した.

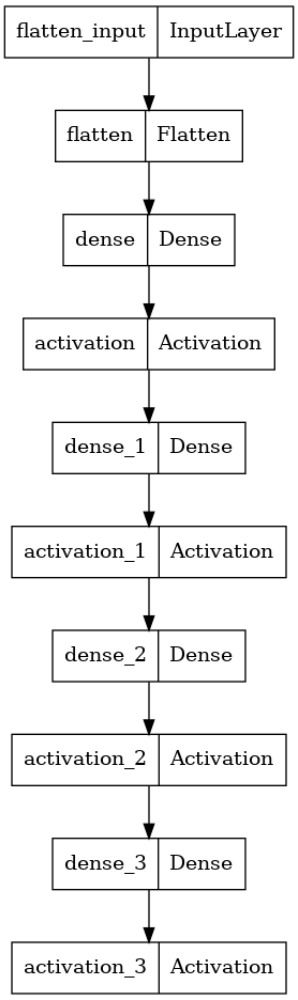

図 2: モデルの構造

4.6 実験結果

10000 ステップ実験を回した.

図 3 は実験において学習過程で 1 エピソードごとのステップ数, 報酬を記録したものである.

また, 図 4 には学習の後検証を行った際の最後のエピソードにおけるログを示している. この時, 図 1 において action = [11, 5, 1] となっているが, これは学習によって得られた先攻プレイヤーの行動を意味しており, (Attack , HP) = (2 , 4) のカードで敵の (4 , 2) のカードを攻撃し,(2 , 3) のカードで敵の (3 , 3) カードを攻撃, (3 , 3) のカードで敵の (3 , 3) のカードを攻撃するという行動を意味する.

4.7 考察

図 3 において, 今回実装した環境が学習環境として正常に動作していることがわかる. 今回の実験設定の場合, 先攻プレイヤーの理想的な動作は 3 枚のカードを最大の報酬を得られるように使うことであり, steps は 3 , その時の reward は 1 となる. 実際図 **??** ではエピソードを重ねていくにつれ理想的な値に近づいていっている. 初期のエピソードなどの値の絶対値が多い時は, エージェントが探索を優先しいろいろな action を試しており, そのため action の重複が発生し steps の値が大きくなり reward の値が小さくなっていったと考えられる.

一方, 実験結果に関しては人間視点では理想的なカードの動かし方とは言えない. (2 , 3) のカードで (3 , 3) のカードを攻撃するのは無意味であり, 相手プレイヤーに直接攻撃したほうがより高い報酬を得られた. これはステップの試行回数の少なさ, 報酬の定義の仕方に問題があると考えられる.

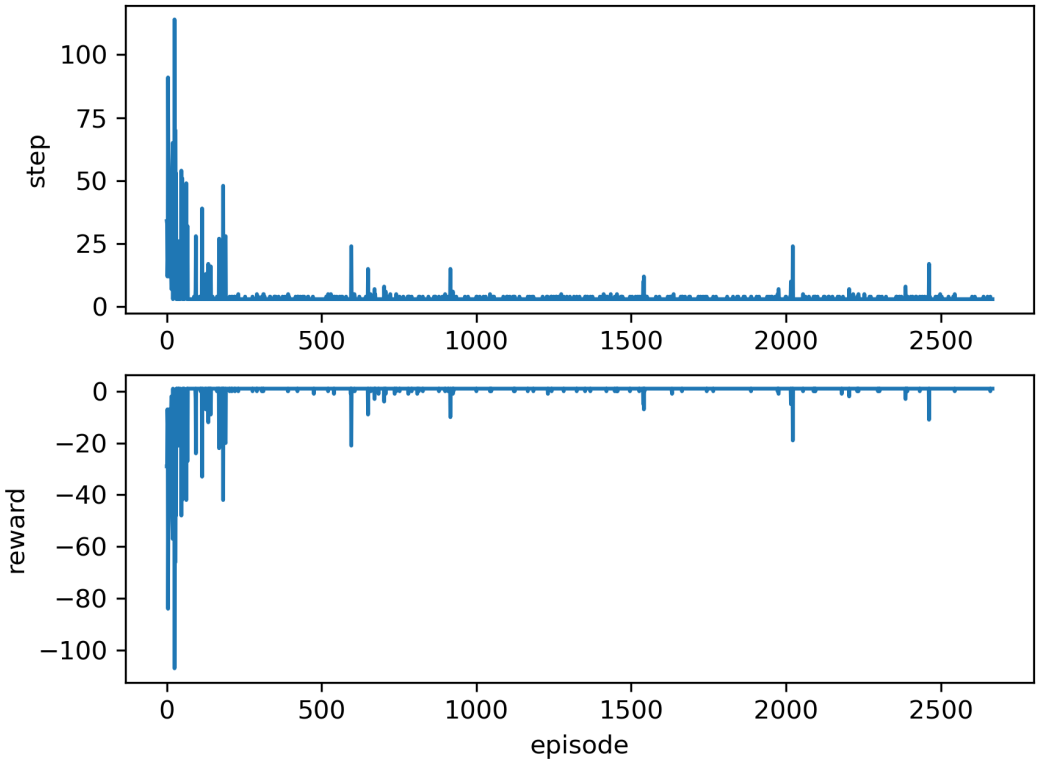

図 3: 1 エピソードごとのステップ数, 報酬

```
RESET STATE
[20, 20, 3, 3, 2, 3, 2, 4, 3, 3, 1, 5, 4, 2, 1, 1, 1]
player1の{Unit3: 4,0}は破壊された
action
[11]
state
[20, 20, 3, 3, 2, 3, 2, 4, 3, 3, 1, 5, 0, 0, 1, 1, 0]
action
[11, 5]
state
[20, 20, 3, 3, 2, 3, 2, 4, 3, 1, 1, 5, 0, 0, 1, 0, 0]
player1の{Unit1: 3,0}は破壊された

player1の攻撃
{Unit1: 0,0}attacks{Unit1: 3,3}

player1の攻撃
{Unit2: 1,5}attacks{Unit1: 3,3}

player1の攻撃
{Unit3: 0,0}attacks{Unit1: 3,2}
reward
18.0
reward
1
action
[11, 5, 1]
state
[20, 20, 3, 2, 2, 3, 2, 4, 0, 0, 1, 5, 0, 0, 0, 0, 0]
Episode 10: reward: 1.000, steps: 3
学習おわ
```

図 4: 検証過程における最後のエピソードのログ

5　今後の課題

- 自作の環境の改良

  今回の実験では, Draw や Play , TurnChange といった動作を考慮せず事前に盤面を設定した. このことで 4.2 節で述べた行動空間の定義を簡単に求めることができた. しかし, 実際のカードゲームではターンが変わるたびに盤面の状況が変わるため逐一行動空間の次元を求めて環境に適用させなければならない. また, 手札などの要素が加わると行動空間の次元が途端に増えるため対処法を考慮, 実装する必要がある.

  また, 実験結果がログとして出てきているが分かりづらいので分かりやすい実験結果の表示を行えるようにしたい.

- カードゲーム自体の改良

  2 節で述べたようにシャドウバースやハースストーンに見られるコスト, 特殊効果といった要素をまだ実装できていない. 必要に応じてやる.

- 自動難易度調整の方法検討

  カードのパラメータを何かしらの結果に基づいて変更すれば良い？ としか思っていないため類似研究などから知識を深めたい.

参考文献

[1] AGIRobots. Openai gym 用ラッパーの自作方法, 2022.10.11. https://agirobots.com/openai-gym-custom-wrapper/.

[2] Cygames Engineers' Blog. ゲーム ai 実践編 － shadowverse に見る tcg ai 開発の事例 (1) －, 2016.07.27. https://tech.cygames.co.jp/archives/2853/.

[3] 田中遥輝. Keras-rl でゲームをプレイする強化学習 ai を作る！, 2022.03.21. https://qiita.com/TanakaSU/items/3221d7fe641aa7fd9006.