卒 業 研 究 報 告 題 目

Q学習を用いたゲームAIの作成

Creation of Game AI using Q-learning

指 導 教 員　　出口 利憲 教授

岐 阜 工 業 高 等 専 門 学 校　電気情報工学科

2019E11 河合 雄貴

令 和 0 6 年（ 2 0 2 4 年 ）　　2 月 1 5 日 提 出

Abstract

The purpose of this study is to conduct learning for the game 'En Garde' using Q-learning. En Garde is a competitive card game characterized by randomly distributed cards and the inability to see the opponent's hand. Two methods are used for learning: one using a Q-table and the other using a Deep Q-Network. The AI was evaluated by competing against the basic policy of En Garde. As a result of the learning, the method using the Q-table was able to win against the basic policy with an 80% probability, and the Deep Q-Network method with a 70% probability.

目 次

第1章　序論

近年、機械学習は自動運転やロボット制御などで注目されている。特にボードゲームでの進歩は目覚ましく、囲碁では「AlphaGO」、将棋では「Ponanza」が人間のトッププロに勝利し大きな話題となった。

強化学習の手法の一つにQ学習がある。Q学習は試行錯誤を繰り返すことで、定義された価値を最大化する行動を取るように学習を進める。Q学習の手法の一つであるDeep Q-Networkは2015年にDeepMindによって開発された手法で、強化学習とディープニューラルネットワークを組み合わせることで様々なゲームを解くことができる。

本研究では強化学習の1手法であるQ学習を用いて、En Gardeという対戦ゲームで基本的な方策に勝つことのできるAIを作成する。

第2章　実験で使用した技法

2.1　機械学習 [1),2)]

機械学習 (Machine Learning) とは人工知能 (AI) を作成する1手法で、コンピュータにデータを与えてルールやパターンを学習させ、それを新たなデータに当てはめることで分類や予測などを可能とする手法である。機械学習は大きく「教師あり学習」「教師なし学習」「強化学習」に分けることができる。

1. 教師あり学習

   教師あり学習とは、正解の存在する「分類」「数値予測」などの問題を解くことのできる手法である。教師あり学習では、コンピュータに「入力」と「正しい出力」をセットにした学習データを与え、ある入力に対して正しい出力を返せるように学習させる。

2. 教師なし学習

   教師なし学習とは、コンピュータに入力データのみを与え、データの特徴をコンピュータに分析させる手法である。教師なし学習では正解データが存在しないため、分析結果が妥当かどうか人間が判断する必要がある。主な学習手法はクラスタリングと次元削減である。

3. 強化学習

   強化学習とは、コンピュータ自身が環境の中で試行錯誤することで学習用データを収集し、人間が決めた良い結果に結びつく行動をとるように学習する手法である。

2.2　ニューラルネットワーク (Neural Network)[3)]

ニューラルネットワークとは、人間の神経細胞の働きをプログラムで再現した、教師あり学習の1手法である。ニューラルネットワークの構造を Figure 2.1 に示す。 ニューラルネットワークは多数のパーセプトロンをネットワーク状につなぎ合わせた構造をしており、入力層 (Input)、中間層 (Hidden Layer)、出力層 (Output Layer) の3つの層で構成される。中間層には複数のレイヤを持つことができ、中間層のレイヤ数が2以上のニューラルネットワークのことを特に Deep Neural Network(DNN) という。

各パーセプトロン (ニューロン) の構造を Figure 2.2 に示す。 入力 $(x_1, x_2, \ldots, x_n)$ を与えた時の出力 y の値は以下の式で計算される。ここで、w_i は重みと呼ばれる値でパー

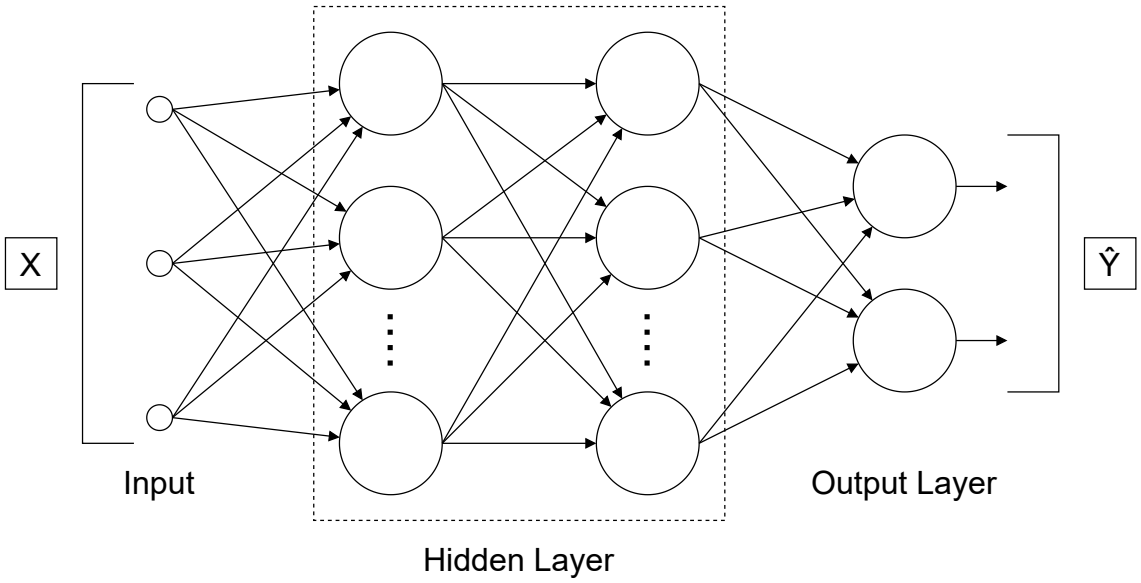

Figure 2.1　Neural network.

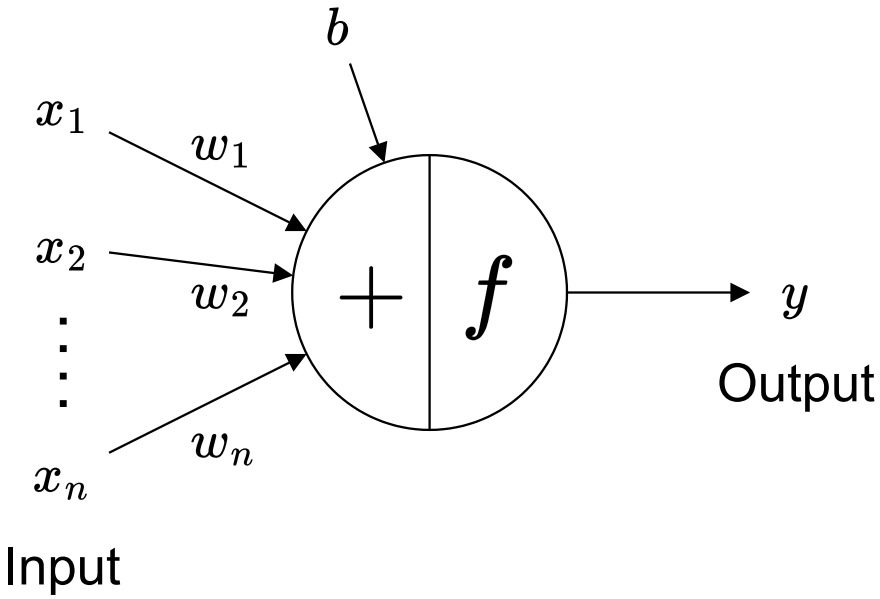

Figure 2.2　Perceptron.

セプトロンの入力と同じ数だけ存在する。b はバイアスと呼ばれる値でパーセプトロンに 1 つだけ存在する。重みとバイアスはパーセプトロンの内部変数である。f は活性化関数と呼ばれる関数で非線形関数を学習するために必要となる。

$$y = f(\sum_{i=1}^{n} x_i w_i + b) \tag{2.1}$$

ニューラルネットワークの出力層では活性化関数から出力される値が最終的なネットワークの出力値 $\hat{Y}$ となるため、活性化関数を以下のように決定する。

- 数値を予測する回帰問題：恒等関数
- 分類問題：Softmax 関数
- 二値の分類問題：Sigmoid 関数

ニューラルネットワークは入力 $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$ を受け取って出力 $\hat{\boldsymbol{Y}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m)$ を返す関数として考えることができる。出力 $\hat{\boldsymbol{Y}}$ の値は各パーセプトロンの内部変数 (重みとバイアス) によって決定されるため、内部変数の値を変えることによって任意の関数を表現することができる。この仕組みを使って、入力が与えられたときに正しい出力を返すように内部変数を調整するのがニューラルネットワークの学習である。

ニューラルネットワークの学習には、入力と正しい出力をセットにしたデータを用いる。学習の流れは以下のようになる。

1. 入力 $\boldsymbol{X}$ からネットワークの出力 $\hat{\boldsymbol{Y}}$ を計算する。
2. 損失関数を使って、ネットワークの出力 $\hat{\boldsymbol{Y}}$ と正しい出力 $\boldsymbol{Y}$ から損失を求める。
3. 最適化アルゴリズムに従い、損失を用いてネットワークの内部変数を更新する。

損失関数は、ネットワークが出力した値と正しい出力値とがどれだけ離れているか (損失) を計算する関数である。学習の目的は損失関数によって求められる損失をできるだけ小さくすることである。使われる損失関数は問題の性質によって異なり、主に以下のように決定される。

- 回帰問題：平均絶対誤差 (MAE), 平均二乗誤差 (MSE), Huber 損失関数
- 分類問題：Cross-Entropy 誤差

平均絶対誤差を用いて損失を計算する式は以下となる。$\hat{y}_k$ はネットワークが出力した値であり、y_k は出力の正解値である。N は学習に用いるデータ数であり、全データの損失の平均を取っている。

$$L = \frac{1}{N} \sum_{k=1}^{N} |y_k - \hat{y}_k| \tag{2.2}$$

最適化アルゴリズムは、損失関数の値が最小となるようにネットワークの内部変数を調整するアルゴリズムである。最も基本的な最適化アルゴリズムは確率的勾配降下法 (SGD) で、以下の式のように内部変数を更新する。w は調整する重みであり、$L(w)$ は損失関数である。η は学習率であり、内部変数の更新量を指定するためのハイパーパラメータ

である。

$$w \leftarrow w - \eta \frac{\partial L(w)}{\partial w} \tag{2.3}$$

確率的勾配降下法では内部変数の更新を繰り返し行うことによって段階的に最適な値へと近づけていく。

2.3 ReLU 関数 (Rectified Linear Unit)[4)]

ReLU 関数は以下の式で表される活性化関数である。ReLU のグラフを Figure 2.3 に示す。ReLU を使うと、層の数が多いニューラルネットワークでも勾配消失がおきにくいため、DNN に使われる最も一般的な活性化関数となっている。

$$f(x) = \max(x, 0) = \begin{cases} 0 & (x \leq 0) \\ x & (x > 0) \end{cases} \tag{2.4}$$

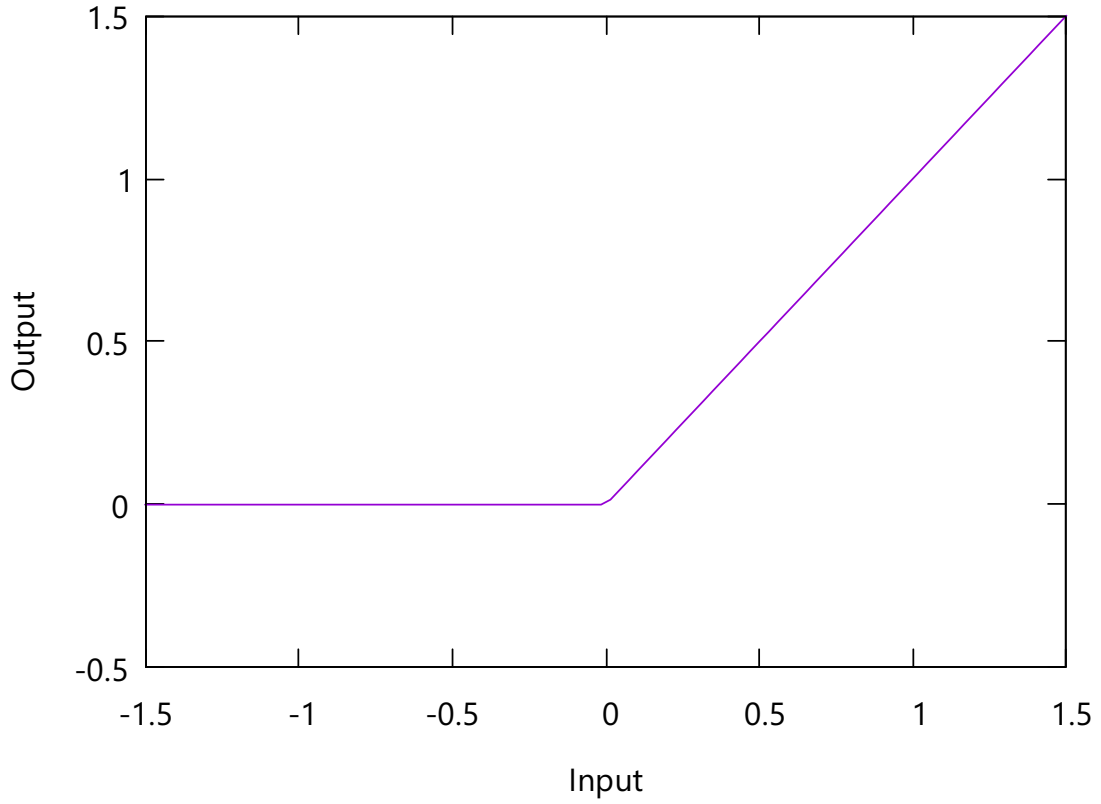

Figure 2.3 ReLU activation function.

2.4 Heの初期化 (He Initialization)[5),6)]

Heの初期化とはKaiming Heらが提案したネットワークの重みの初期化方法でKaimingの初期化とも呼ばれる。Heの初期化は活性化関数がReLUであるときに使う初期化方法で、従来のXavierの初期化をReLUの非線形性を考慮して改良したものである。Heの初期化では式 (2.5) のように初期値を決定する。N はパーセプトロンの入力の数である。重みの初期値は平均が0、標準偏差が $\sqrt{2/N}$ の正規分布によって決定する。バイアスは0に初期化する。

$$
\begin{aligned}
&W \sim \mathcal{N}(\mu, \sigma^2) \\
&\mu = 0,\ \sigma = \sqrt{\frac{2}{N}} \\
&b = 0
\end{aligned}
\tag{2.5}
$$

2.5 Adam[7)]

Adamとは、確率的勾配降下法をより効率的に学習できるよう改良した最適化アルゴリズムである。AdamはAdaGradとRMSPropの利点を組み合わせたアルゴリズムで、パラメータ調整を行わなくても様々な問題に適用できるため、最もよく使われる最適化アルゴリズムとなっている。

2.6 強化学習 (Reinforcement Learning)[2),8),9)]

強化学習とは試行錯誤によって環境に適応する学習方法で、ゲームAIやロボット工学などに利用されている。強化学習では、行動を行うエージェント (AI) が、行動したことに対するフィードバックとして環境から報酬を受け取ることによって、どの行動が良いのかを学習していく。強化学習は以下のプロセスの繰り返しによって行われる。

1. 環境から状態 s_t を受け取る
2. 状態 s_t からエージェントが行動 a_t を決定する
3. 環境に行動 a_t を適用し、次の状態 s_{t+1} を受け取る
4. 環境はエージェントへ報酬 r_{t+1} を与える

エージェントは直前の状態のみを元に行動を決定するため、このプロセスをマルコフ決定過程と呼ぶ。また、状態 s_t から行動 a_t を決定する関数のことをポリシー π という。

エージェントの目標は将来にわたっての累積報酬を最大化することである。タイムステップ t での累積報酬は以下の式で表される。

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{2.6}$$

ここで、γ は割引率と呼ばれる $[0, 1]$ の値で、将来の報酬ほど実際に得られる可能性は低くなるということを表す。また、T はエピソードの最終タイムステップを表す。

累積報酬を最大化する最適なポリシー π^* を見つけるために、Q 学習という手法が用いられる。

2.6.1 Q 学習 (Q-Learning)

Q 学習とは、Q 関数と呼ばれる行動価値関数をトレーニングする手法である。行動価値関数とは、状態 s と行動 a が入力されたときに、状態 s で開始されたエージェントが行動 a を実行し、その後ポリシー π に従って行動する場合に得ることが期待される累積報酬を出力する関数である。Q 関数は以下の式で定義される。

$$Q_\pi(s, a) = \boldsymbol{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a] \tag{2.7}$$

Q 関数から出力される値は Q 値と呼ばれ、状態 s において行動 a を取ることの価値を表す。

式 (2.7) の計算には、将来に得られる全ての報酬を合計する必要があるため計算コストがかかる。この計算を簡素化するためにベルマン方程式が用いられる。ベルマン方程式を以下に示す。

$$V_\pi(s) = \boldsymbol{E}_\pi[R_{t+1} + \gamma * V_\pi(S_{t+1}) | S_t = s] \tag{2.8}$$

ベルマン方程式は価値関数を再帰定義したもので、次の状態の価値 $V_\pi(S_{t+1})$ を用いて現在の状態の価値 $V_\pi(s)$ を算出することができる。

Q 学習のポリシー $\pi(s)$ は Q 関数を用いて定義され、以下の式で表される。この式は、常に Q 値が最大になる行動を選択するということを表している。

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q_\pi(s, a) \tag{2.9}$$

Q 学習の目的は最適な Q 関数 $Q^*(s, a)$ を見つけることである。最適な Q 関数が分かれば、最適なポリシー π^* を求めることができる。最適な Q 関数 $Q^*(s, a)$ はベルマン方

程式を用いて、以下の式で表される。この式をベルマン最適方程式という。

$$Q^*(s,a) = \boldsymbol{E}[R_{t+1} + \gamma * \max_a Q^*(s_{t+1}, a_{t+1})] \tag{2.10}$$

Q 関数のトレーニングは、ベルマン最適方程式を用いて Q 関数を反復更新することによって行う。

$$Q_{i+1}(s,a) \leftarrow \boldsymbol{E}[R_{t+1} + \gamma * \max_a Q_i(s_{t+1}, a_{t+1})] \tag{2.11}$$

この更新を無限回行うことによって最適な Q 関数が求まる。

$$Q_i(s,a) \to Q_*(s,a) \quad (i \to \infty) \tag{2.12}$$

Q 学習の学習方法には TD 学習が用いられる。TD 学習とは行動ステップ毎に Q 関数を更新するという学習方法である。TD 学習では式 (2.11) の右辺を TD ターゲットと呼び、TD ターゲットに現在の Q 値を徐々に近づけるように更新を行う。TD 学習では将来の累積報酬 $Q_i(s_{t+1}, a_{t+1})$ はまだ確定していないため、TD ターゲットの値はあくまで累積報酬の推定値である。TD 学習での Q 関数の更新式を以下に示す。

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.13}$$

ここで、α は学習率で $[0, 1]$ の値をとる。Q 学習ではエージェントが環境を探索し、繰り返し Q 関数を更新することで、最適な行動ができるようになる。

2.6.2 epsilon-greedy

epsilon-greedy 方策は Q 値から行動を決定する際に最適行動ばかりをとるのではなく、一定確率 ε でランダムな行動をとる手法である。これにより、エージェントにもっといい行動がないかを探索させることができる。

2.6.3 Q テーブル

古典的な Q 学習では、Q 関数を表現するのに Q テーブルという表を用いる。Q テーブルは、状態と行動を表す 2 つの軸を持ち、状態と行動に対応する Q 値を格納する。状態と行動が入力されると、Q テーブル内を検索し、入力された状態と行動に対応する Q 値を出力することによって Q 関数を表現することができる。Q 関数の学習は Q テーブルの内容を更新することによって行う。Q テーブルの値は学習前にすべて 0 に初期化され、学習時には各ステップで現在の状態と行動に対応する Q 値が更新される。

Q 関数を Q テーブルで表すことには以下の問題がある。

- 離散的な状態、行動しか扱えない

状態が連続値で表される環境では状態を離散化する必要がある。

- 状態数が膨大になると、Q テーブルのサイズが非常に大きくなる
  Q テーブルがメモリに収まりきらなくなるため、学習が困難となる。

2.6.4 Deep Q-Network(DQN)

Deep Q-Network(DQN) は 2015 年に DeepMind によって開発されたアルゴリズムで、Q 学習と深層学習を組み合わせることで幅広い Atari のゲームを解くことができる。DQN では、Q 関数をニューラルネットワークで近似する。このニューラルネットワークは Q-Network と呼ばれ、状態を入力として与えると、その状態で行える行動ごとの Q 値を出力する。ニューラルネットワークは連続値を入力できるため、離散的な状態しか扱えないという Q テーブルの問題を解決できる。

DQN の学習では、教師データを TD ターゲットとしてネットワークの重みを更新する。学習は以下の流れで行われる。

1. 行動を行い、$(s_t, a_t, r_{t+1}, s_{t+1})$ を経験として Replay Buffer に保存する
2. Replay Buffer からバッチサイズだけ経験をランダムに取得し、ニューラルネットワークをトレーニングする

DQN の学習を安定させるための工夫として、Experience Replay、Fixed Q-Target などの手法がある。

2.6.5 Experience Replay[9),10)]

経験を Replay Buffer に保存し、後からランダムに取り出して学習を行う手法を Experience Replay という。これには以下のメリットがある。

- 経験を繰り返し学習に使えるため、学習の効率が上がる
- 古い経験を忘れ、直近の経験にのみ適応するのを避けることができる
- 時間的な相関関係を取り除くことで学習が安定する

2.6.6 Fixed Q-Target[9),10)]

教師データである TD ターゲットの値を計算するときには $\max_a Q(S_{t+1}, a)$ を求めるために Q-Network が使われる。そのため、学習時に Q-Network の重みを更新すると、教師データである TD ターゲットの値も変更されてしまう。教師データが頻繁に変わると、

学習が不安定になるという問題がある。これを解決するために、TD ターゲットの計算には、Target Network と呼ばれる学習によってパラメータが変わらない別のネットワークを使用し、一定間隔で学習を行う Q-Network のパラメータを Target Network にコピーするという手法を Fixed Q-Target という。

2.6.7 Huber 損失関数[11)]

Deep Q-Network では損失関数に Huber 損失関数を利用する。Huber 損失関数の式を以下に示す。

$$L(x,y) = \begin{cases} \frac{1}{2}(x-y)^2 & (|x-y| \leq 1) \\ |x-y| - \frac{1}{2} & (|x-y| > 1) \end{cases} \tag{2.14}$$

Huber 損失関数は平均二乗誤差と平均絶対誤差の利点を組み合わせたもので、平均二乗誤差よりも外れ値の影響が小さく、平均絶対誤差よりも細かい微調整ができる。

2.6.8 Reward clipping[10)]

Deep Q-Network では、正の報酬を 1 、負の報酬を -1 に固定する Reward clipping という手法がある。報酬の範囲を $[-1, 1]$ に限定することで学習が安定するメリットがあるが、報酬の大小を区別できなくなるというデメリットがある。

2.6.9 Self-Play[9)]

対戦ゲームの学習では、エージェントの学習時の対戦相手はエージェントと同じ強さであることが望ましい。これは、対戦相手が弱すぎるとエージェントは強い相手には効果の無い行動を学習してしまい、対戦相手が強すぎるとエージェントが負け続けて何も学習できないためである。Self-Play とは対戦ゲームの学習において、エージェントと同じ強さの対戦相手を用意するための手法である。Self-Play では、対戦相手としてエージェント自身の過去のコピーを用い、学習が進むにつれて対戦相手をエージェントの最新のコピーで更新する。

2.6.10 TF-Agents[12)]

TF-Agents は TensorFlow で強化学習を行うためのライブラリである。TF-Agents を用いることで、Deep Q Network をはじめとした、Double DQN、PPO などの様々な強化学習アルゴリズムを容易に実装することができる。また、Replay Buffer や epsilon-greedy 方策などの実装も提供されている。

第3章　実験

3.1　目的

この研究の目的はEn Gardeというゲームで基本的な方策に勝つことができるAIを制作することである。

3.2　ゲームの概要[13)]

ゲームの初期画面をFigure 3.1に示す。En Gardeはゲームボードとカードを使って2人のプレイヤーが戦う対戦ゲームである。Figure 3.1の下部に表示されている23個のマスがゲームボード(フィールド)であり、ゲームボード上には2つの駒(黒と白)が置かれる。これらの駒は対戦する2人のプレイヤーに対応し、プレイヤーの1人が黒い駒を、もう1人が白い駒を動かすことによってゲームを行う。ゲームの始めには、2つの駒はゲームボードの両端のマスにそれぞれ置かれる。カードは、1〜5の数字が書かれたカードを5枚ずつ、計25枚使用する。ゲームの始めには、各プレイヤーに5枚ずつカードがランダムに配られ、配られたカードは各プレイヤーの手札となる。Figure 3.1の上部にある、四角で囲まれた5つの数字が各プレイヤーの手札である。黒い駒を動かすプレイヤーは1〜5の数字のカードを1枚ずつ持っており、白い駒を動かすプレイヤーは2の数字のカードを2枚、3の数字のカードを1枚、4の数字のカードを2枚持っている。プレイヤーは相手が持っているカードを見ることはできない。ゲームの始めに各プレイヤーにカードを配った後、残ったカードは山札(デッキ)とし、シャッフルして裏向きに置く。Figure 3.1の上部中央にある数字15は山札の枚数を表す。

ゲームは2人のプレイヤーが交互に自分の駒を動かすことによって進行する。ゲームが始まったとき、最初に駒を動かすプレイヤーは黒い駒を持つプレイヤーとする。プレ

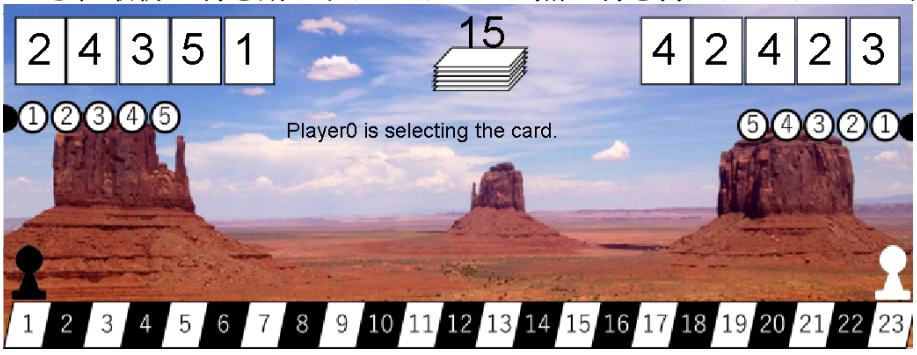

Figure 3.1　Game screen.

イヤーは自分の手番となった時、以下の行動を行う。

1. 手札の中からカードを 1 枚選ぶ
2. 以下のいずれかの動作を行う
    - 選んだカードに書かれている数だけ駒を前進させる。ただし相手の駒と同じマスに入ったり、相手の駒を飛び越えることはできない。
    - 選んだカードに書かれている数だけ駒を後退させる。ただしフィールド外に出ることはできない。
    - 選んだカードに書かれている数と 2 つの駒の距離が同じ時、相手を攻撃することができる。攻撃時には同じ番号のカードを複数枚使うことができ、より多くの枚数を使うほど攻撃の威力が上がる。
3. 使ったカードを場に出す
4. 使ったカードの枚数だけ山札からカードを引く

相手から攻撃を受けたとき、攻撃に使われたカードと同じ番号のカードを攻撃に使われた枚数以上持っていれば攻撃を受け流す (パリーする) ことができる。攻撃をパリーできない時は攻撃したプレイヤーの勝ちとなる。攻撃をパリーした時はパリーに使用したカードを場に出し、カードを補充せずに次の行動に移る。以下の条件の時にゲームが終了する。

- 攻撃をパリーできなかった → 攻撃したプレイヤーが勝ち
- 手番プレイヤーが前進、後退、攻撃のいずれもできない → 手番プレイヤーが負け
- 山札がなくなる → 相手を攻撃できるカードが多いほうが勝ち。同じ場合は駒がより前に進んでいるプレイヤーが勝ち。すべて同じ場合は引き分け。

3.3 状態

片方のプレイヤーから見たゲームの状態を以下の 4 つの要素で表す。

- プレイヤーの位置

  駒を初期位置からどれだけ前に動かしたのかを 0〜21 の整数で表す。Figure 3.1 において黒い駒を動かすプレイヤーから見た場合、駒が 1 のマスにある時のプレイヤーの位置の値は 0、駒が 22 のマスにある時のプレイヤーの位置の値は 21 と

なる。

- 相手との距離

  相手の駒とどれだけ離れているかを 0〜21 の整数で表す。Figure 3.1 では、2 つの駒は 22 マス離れているため、相手との距離の値は 21 となる。黒い駒が 1 のマス、白い駒が 2 のマスにある時、相手との距離の値は 0 となる。

- プレイヤーの手札

  1〜5 のカードのそれぞれの枚数を 0〜5 の 5 つの整数で表す。Figure 3.1 において黒い駒を動かすプレイヤーから見た場合、プレイヤーの手札は $[1, 1, 1, 1, 1]$ となる。

- 場に公開されているカード

  自分や相手が行動した後に場に出したカードのことである。1〜5 のカードのそれぞれの枚数を 0〜5 の 5 つの整数で表す。Figure 3.1 においてはまだ行動が行われていないため、場に公開されているカードは $[0, 0, 0, 0, 0]$ となる。黒い駒を動かすプレイヤーが 5 のカードで駒を前進させたとき、場に公開されているカードは $[0, 0, 0, 0, 1]$ となる。

3.4 行動

エージェントを学習する上で、プレイヤーの行動を数値で表す方法を考える。プレイヤーが行う行動は、前進、後退、攻撃である。前進は 1〜5 のいずれかのカードで行うため、使うカードごとに数値を割り当て、1〜5 の整数で表す。後退も同様に使うカードごとに数値を割り当て、6〜10 の整数で表す。攻撃は常に最大枚数で行うこととし、数値 0 で表す。これは、手札にある枚数より少ない枚数での攻撃はあまり効果的ではないため、考慮する必要がないと判断したためである。これらのことからプレイヤーの行動は 0〜10 の整数で表される。

3.5 評価方法

AI の評価は、ある一定のアルゴリズムに従って動く相手プレイヤーとの対戦によって行う。このアルゴリズムは En Garde における最も基本的な戦略と考えられるものである。相手プレイヤーのアルゴリズムを以下に示す。

1. 基本的には前進を行う (カードはランダム)

2. 攻撃できるときには攻撃を行う (多い枚数での攻撃を優先)

3. 前進も攻撃もできない時には後退を行う (カードはランダム)

この対戦相手と N 回戦ったときに勝った回数をスコアとし、スコアを N で割った値を勝率とする。作成した AI の評価には勝率を利用する。

3.6 Q テーブルを用いた学習

Q テーブルを用いて En Garde の学習を行う。

Q テーブルは、double 型の値を持つ 2 次元配列として表現する。状態と行動は、Q テーブルのインデックスとして Q テーブル内の検索に使われるため、それぞれ整数値で表す必要がある。行動については第 3.4 節のように数値化することとする。状態については、プレイヤーの位置や相手との距離など複数の数値の組み合わせとして表されるため、単一の整数値に変換する必要がある。状態が N 個の数値列 $s_1, s_2, \ldots, s_N$ で表され、各数値の最大値が $m_1, m_2, \ldots, m_N$ で表されるとき、以下の式を用いて変換を行う。

$$I = \sum_{i=1}^{N} (s_i \times \prod_{j=1}^{i-1} (m_j + 1)) \tag{3.1}$$

状態として場に公開されたカードまで考慮すると、状態数が膨大となり Q テーブルでは扱うことができないため、ここではプレイヤーの手札までを状態として扱う。学習前の Q テーブルの値は、範囲が $[0, 1]$ のランダムな値で初期化することとする。

Q テーブルの学習は、評価に用いる対戦相手と対戦することによって行う。学習時には行動を行う各ステップごとに、式 (2.13) を用いて Q 値の更新を行う。次の状態 S_{t+1} は、相手が行動した後に自分の手番になった時の状態とする。ゲーム開始時の初期ステップでは $\max_a Q(S_{t+1}, a)$ の値が計算できないため更新を行わない。また、ゲーム終了時の最終ステップでは次の状態が存在しないため、 $\max_a Q(S_{t+1}, a)$ の値は 0 とする。エージェントに与える報酬 R_{t+1} は、ゲームに勝利した際に $+100$ 、ゲームに負けた際に -100 とする。1 ステップごとの報酬は与えない。学習パラメータ α, γ の値は、グリッドサーチによって最適な値を決定する。

行動の選択には epsilon-greedy 方策を使用する。ε の確率でランダムな行動を取ることにより環境を探索させる。それ以外の場合は、現在の状態で行える行動の中から Q 値が

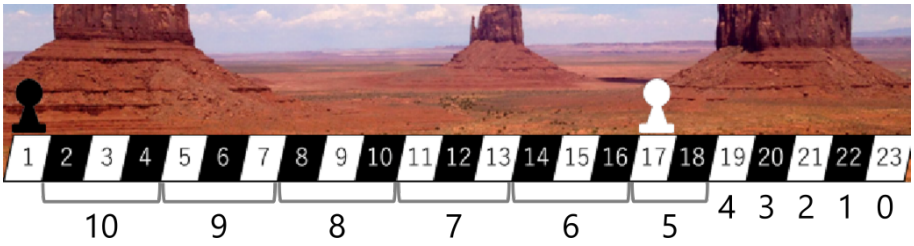

Figure 3.2　State reduction of player position.

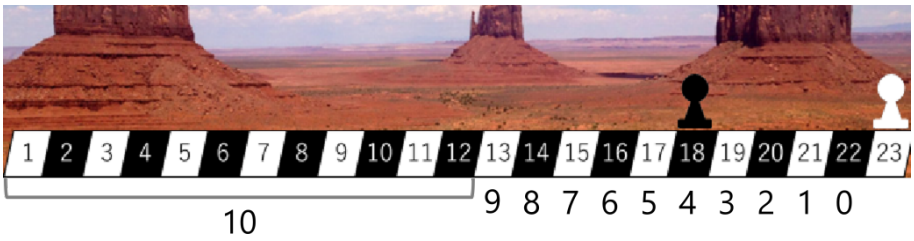

Figure 3.3　State reduction of distance to opponent.

最大の行動を選択する。ε の減衰は行わないものとする。

ゲーム開始から勝敗が決まるまでを 1 エピソードとする。先手と後手で勝ちやすさが異なる可能性があるため、1 エピソードごとに先手と後手を入れ替える。評価時に勝率を求めるための対戦回数は 10000 回とする。

3.6.1　状態数の削減

Q テーブルのサイズを削減するために状態数の削減を行う。

プレイヤーの位置は 0〜21 の整数で表されるため状態数は 22 であるが、隣接する 3 マスを 1 つの状態にまとめることで状態数を削減する。また、フィールドの端に近い 0〜4 のマスは後退できるかの判定に使うため、個別に状態を割り当てる。これらの操作により、プレイヤーの位置の状態数は 11 となる (Figure 3.2)。

相手との距離は 0〜21 の整数で表されるため状態数は 22 であるが、距離が遠い場合を 1 つの状態で表すことで状態数を削減する。距離が 9 の時に 5 のカードで前進すると相手に攻撃される可能性があるため、距離が 0〜9 のときに個別に状態を割り当てる。この操作により、相手との距離の状態数は 11 となる (Figure 3.3)。

プレイヤーの手札は状態数の削減を行わず、すべての状態を考慮する。プレイヤーの手札の状態数を求めるために、重複組合せの問題として考える。重複組合せは n 種類のものから重複を許して r 個選ぶ組合せの総数のことである [14]。プレイヤーの手札の場

Table 3.1　Parameters explored in grid search.

Parameter	Value
Learning Rate α	0.01, 0.05, 0.1, 0.3, 0.5
Discount Factor γ	0.7, 0.8, 0.9, 0.95, 0.99
Exploration Probability ε	0.01, 0.05, 0.1, 0.2, 0.3

合、1 から 5 の 5 種類のカードがあるため $n = 5$ となる。また手札の枚数は、攻撃をパリーした時も考慮すると 3 から 5 枚となるため $r = 3, 4, 5$ の組合せ数をすべて合計した値が状態数となる。状態数を求める式を以下に示す。

$$\sum_{5}^{r=3} {}_{n+r-1}C_r = 35 + 70 + 126 = 231 \tag{3.2}$$

最終的な状態数は $11 \times 11 \times 231 = 27951$ となる。状態数が 27951、行動数が 11 なので、Q テーブルのサイズは $27951 \times 11 = 307461$ となる。

3.6.2　パラメータ調整

学習に使用する最適なパラメータを調べるため、グリッドサーチを用いて探索を行う。調べるパラメータの値を Table 3.1 に示す。 各パラメータの組み合わせで 200000 エピソード学習させた結果、最も勝率が高かった 10 組を Table 3.2 に、最も勝率が低かった 10 組を Table 3.3 に示す。

パラメータの傾向として、α は大きすぎると勝率が下がり、0.05 から 0.1 の間あたりで最適な値となることが分かった。また、γ は小さいほど勝率が上がり、ε は大きいほど勝率が上がることが分かった。γ と ε について、調べた範囲外に最適な値があると考えられるため、α の値を 0.07 に固定して再度グリッドサーチを行う。調べる値はそれぞれのパラメータで、0.1 から 0.9 まで 0.1 刻みの値とする。各パラメータの組み合わせで 200000 エピソード学習させた時の、グリッドサーチの結果を Figure 3.4 に示す。 Figure 3.4 より、γ の値は 0.4 から 0.6、ε の値は 0.3 から 0.7 が最適であることが分かった。一般的に使われるパラメータ $\gamma \geq 0.9, \varepsilon \leq 0.1$ とは大きく異なる値だが、これは次の状態 S_{t+1} が相手の行動によって変わるため、状態遷移のランダム性が高いからだと考えられる。状態遷移のランダム性が高いと、将来の報酬を得られる可能性が低くなるため割引率の値

Table 3.2 Best 10 parameters in grid search.

α	γ	ε	Win Rate
0.1	0.7	0.3	77.76
0.1	0.7	0.2	77.36
0.1	0.8	0.3	77.33
0.05	0.7	0.3	76.63
0.05	0.7	0.2	75.94
0.1	0.7	0.1	75.27
0.05	0.8	0.3	75.05
0.1	0.8	0.2	74.91
0.1	0.9	0.3	74.57
0.01	0.7	0.3	74.49

Table 3.3 Worst 10 parameters in grid search.

α	γ	ε	Win Rate
0.5	0.95	0.3	61.89
0.5	0.9	0.1	61.84
0.5	0.9	0.01	61.8
0.5	0.95	0.01	61.75
0.5	0.95	0.1	61.72
0.5	0.95	0.05	61.33
0.5	0.95	0.2	61.31
0.5	0.9	0.05	61.29
0.5	0.99	0.1	61.23
0.5	0.99	0.3	61.08

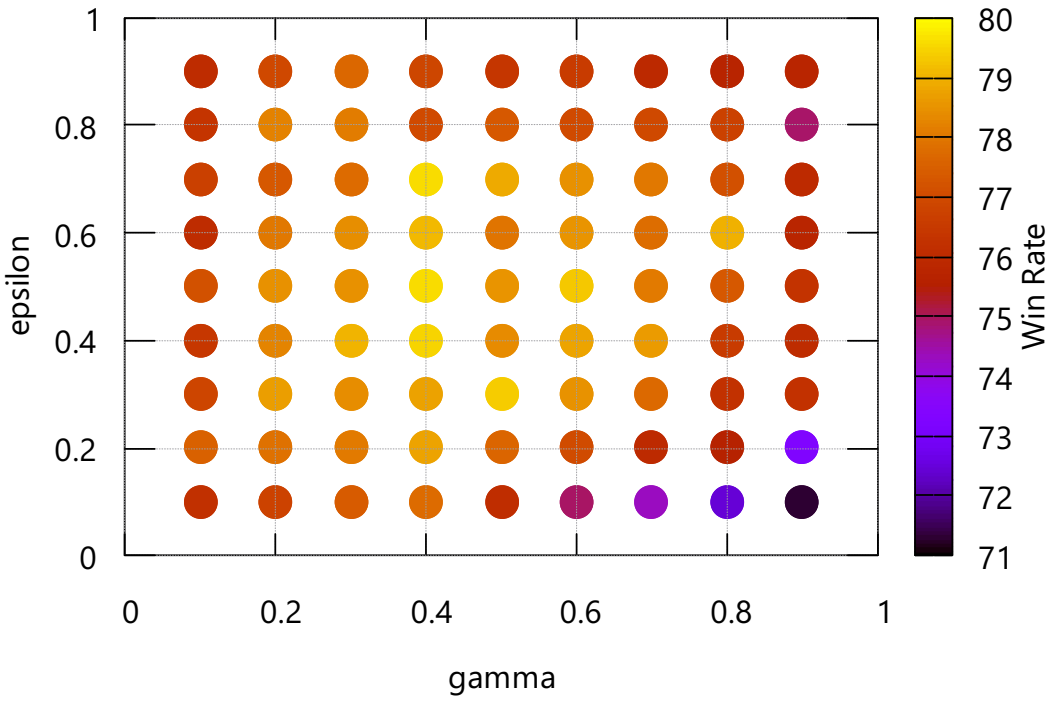

Figure 3.4　Grid search result of γ and ε.

Table 3.4　Parameters of Q-Learning.

Parameter	Value
Learning Rate α	0.07
Discount Factor γ	0.6
Exploration Probability ε	0.5

が小さくなり、より多くの探索を必要とするため ε の値が大きくなったと考えられる。また、公開されているカードを状態に含めていれば相手の手札がある程度限定されるため遷移後の状態も限られるが、今回は公開されているカードを状態に含めていないため、より状態遷移のランダム性が高くなっていると考えられる。

グリッドサーチの結果より、学習に使用するパラメータを Table 3.4 のように決定する。

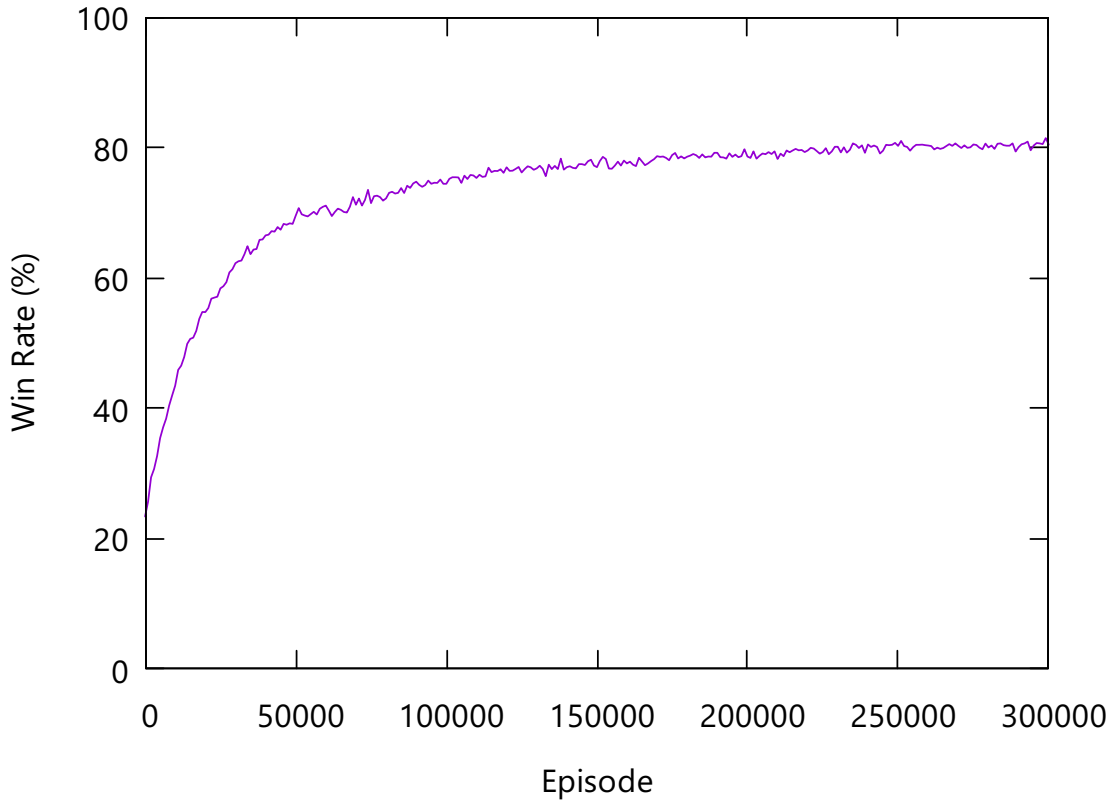

Figure 3.5　Learning results of the Q-Table.

3.6.3　結果

Table 3.4 のパラメータを用いて、300000 エピソード学習させた結果を Figure 3.5 に示す。勝率の計算は 1000 エピソードごとに行っている。 50000 エピソードあたりで勝率の上昇が緩やかになった。学習後の勝率は 81 % まで上がった。

第 3.5 節の対戦相手と 10000 回対戦を行うと、AI が勝った回数は 8078 回であった。勝った回数をゲームの終了条件ごとに求めると Table 3.5 のようになった。勝った要因の 9 割近くが攻撃の成功によるものだと分かる。また、負けた回数 1922 回をゲームの終了条件ごとに分けると Table 3.6 のようになった。40.01 % は相手からの攻撃によって負けているが、38.71 % は山札が無くなったときに相手よりも前に進んでいなかったことによって負けている。10000 回の対戦において相手プレイヤーに攻撃した回数は 9261 回であった。攻撃回数を攻撃の結果ごとに分けたものを Table 3.7 に示す。攻撃が成功した回数は、相手からの攻撃をパリーした後に反撃した場合とそうでない場合で分けて表示している。8 割以上の攻撃が成功しており、相手から反撃された攻撃は 5 % 未満であることが分かる。相手プレイヤーから攻撃された回数は 4718 回であった。攻撃回数を攻撃の結果ごとに分けたものを Table 3.8 に示す。8 割以上の攻撃をパリーできており、62.87 %

Table 3.5　Wins per game end condition in Q-Table.

Game End Condition	Number of Wins	Ratio
Attack Successful	7336	90.81 %
Unable to Act	9	0.11 %
Deck Depleted (had more cards)	578	7.16 %
Deck Depleted (ahead of opponent)	155	1.92 %

Table 3.6　Losses per game end condition in Q-Table.

Game End Condition	Number of Losses	Ratio
Attack Successful	769	40.01 %
Unable to Act	83	4.32 %
Deck Depleted (had more cards)	326	16.96 %
Deck Depleted (ahead of opponent)	744	38.71 %

Table 3.7　Number of AI attacks per outcome in Q-Table.

Attack Outcome	Number of Attacks	Ratio
Attack Successful	4851	52.38 %
Attack Successful (Counterattack)	2778	30.0 %
Parried with Equal Number of Cards	1182	12.76 %
Parried with More Cards	450	4.86 %

の攻撃は反撃できていることが分かる。

学習後の AI と対戦することにより、AI の挙動を確認する。AI の挙動は以下のようになった。

- 1 手目は数字が大きなカードで前進
- 相手が近づいてきたら、相手の攻撃範囲 (5 マス) の直前まで前進
- 相手の攻撃をパリーできるカードを 2 枚以上持っていたら、相手の攻

Table 3.8　Number of opponent attacks per outcome in Q-Table.

Attack Outcome	Number of Attacks	Ratio
Attack Successful	393	8.33 %
Attack Successful (Counterattack)	419	8.88 %
Parried with Equal Number of Cards	940	19.92 %
Parried with More Cards	2966	62.87 %

撃範囲に入る

- 攻撃できるカードを 2 枚以上持っていたら攻撃
- 攻撃できるカードが 1 枚の時は、
  プレイヤーの位置が前の方なら移動
  プレイヤーの位置が後ろの方なら攻撃
- 相手の攻撃をパリーしたら反撃
- 相手の攻撃範囲で自分が攻撃できないときは後退

AI の対戦相手の位置が相手側のフィールドの端付近にあり、AI の位置がフィールドの中間より相手側にあるとき、AI はパリーできるカードを持っていないのに相手の攻撃範囲に入ったり、相手との距離が離れているのに後退するなど、ランダムに動くような挙動を見せた。

3.6.4　考察

学習の結果、AI は評価用の対戦相手に対して 81 % の確率で勝つことができた。場に公開されているカードの情報を使わなくても、評価用の対戦相手に対しては高い確率で勝てることが分かった。

評価用の対戦相手との 10000 回の対戦において、成功した攻撃のうち 36.4 % が攻撃をパリーした後の反撃によるものであり、相手から受けた攻撃のうち 62.87 % は反撃できるものであった。よって、AI が学習によって習得した戦略は、相手の攻撃をパリーできる位置に移動し、攻撃をパリーした後に反撃するというものだと考えられる。この戦略では、対戦相手が攻撃カードを持っていないときには相手を後退させることができ、攻撃を受けた後に反撃できる場合は必ず勝つことができるため、有効な戦略であると考え

られる。

AI は、AI の位置がフィールドの中間より相手側にあるときにランダムに動く挙動を見せた。これは、評価用の対戦相手は前進を優先して行動するため、学習時の AI の位置は AI から見て後方にあることが多く、AI の位置が前方にあるときの経験が少ないからだと考えられる。

3.7 Deep Q-Network

Deep Q-Network は TensorFlow の TF-Agents ライブラリを用いて実装する。

Q-Network は状態を入力として受け取り、行動ごとの Q 値を出力する。行動は離散化する必要があるため、Q テーブルの学習と同様に第 3.4 節のように表すこととする。状態については Q テーブルの学習と異なり、場に公開されているカードの情報も含めて状態として扱うこととする。

Deep Q-Network の学習は、評価に用いる対戦相手と対戦することによって行う。ネットワークの学習には Experience Replay の手法を用いる。学習前には Replay Buffer の初期値として、ランダムに行動したときの経験を 10 エピソード分 Replay Buffer に追加する。学習の流れとしては、以下の処理を 1 ステップとし n_i 回繰り返すことによって学習を行う。

1. n_e エピソードの間行動を行い、収集した経験を Replay Buffer に追加する
2. Replay Buffer からバッチサイズの個数分経験をランダム抽出し、ネットワークの重みを更新する

経験の収集には、Q テーブルの学習と同様に epsilon-greedy 方策を用いる。ネットワークを更新する時には、Fixed Q-Target の手法を用いる。ネットワークの重みを Target Network にコピーする間隔は 10 ステップとする。報酬は Reward clipping の手法を用い、ゲームに勝利した際に $+1$ 、ゲームに負けた際に -1 とする。1 ステップごとの報酬は与えない。

評価時に勝率を求めるための対戦回数は、ニューラルネットワークを使うことによる計算コストがかかるため、100 回とする。

3.7.1 状態の正規化

Deep Q-Network の状態は、ニューラルネットワークに入力するために $[0, 1]$ の範囲に正規化する必要がある。

プレイヤーの位置、相手との距離については、最大値の 21 で割ることによって範囲を $[0, 1]$ にする。状態数はそれぞれ 1 である。プレイヤーの手札については、1 から 5 のカードのそれぞれの枚数を、最大枚数 5 で割ることによって範囲を $[0, 1]$ にする。状態数は 5 である。

場に公開されているカードも状態として考慮するため、数値化の方法を考える。まず、場に公開されているカードにはゲームがどれだけ進んだかという情報が含まれているため、この情報を切り分けて独立した状態として扱う。カードの総枚数 25 枚から、場に公開されているカードの枚数、プレイヤーの手札の枚数、相手の手札の枚数 (手番時には常に 5 枚) をそれぞれ引くと、山札の枚数を求めることができる。山札の枚数は、あと何枚カードを引けばゲームが終了するのかというゲームの進行状態を表すため、これを状態として扱う。山札の枚数の最大値は 15 枚であるため、山札の枚数を 15 で割ることによって $[0, 1]$ の範囲にする。状態数は 1 である。また、場に公開されているカードを使うことによって、相手の手札やプレイヤーがこれから引くカードなどを予想することができる。例として、5 のカードが場に 4 枚公開されておりプレイヤーの手札に 1 枚あるとすると、5 のカードの総枚数は 5 枚であるため、相手は 1 枚も 5 のカードを持っていないということが分かり、山札の中にも 5 のカードは存在しないことが分かる。このようにして使う場合、公開されているカードから考えるよりも、まだプレイヤーから見えないカード (相手の手札 + 山札) を考えた方が分かりやすい。そのため、プレイヤーから見えないカードを計算によって求め、これを状態として扱う。プレイヤーから見えないカードは、プレイヤーから見えるカード (場に公開されているカード + プレイヤーの手札) を、すべてのカード (1〜5 のカード各 5 枚) から引くことによって求められる。プレイヤーから見えないカードの状態は 1〜5 のカードのそれぞれの枚数で表し、それぞれの数値を最大枚数 5 で割ることによって $[0, 1]$ の範囲にする。状態数は 5 である。

最終的に状態数は $1 + 1 + 5 + 1 + 5 = 13$ となる。

以上の操作により、状態は数値の範囲が $[0, 1]$ となる 13 個の数値列で表されることとなり、それぞれの値がネットワークの入力層の対応するニューロンに入力される。

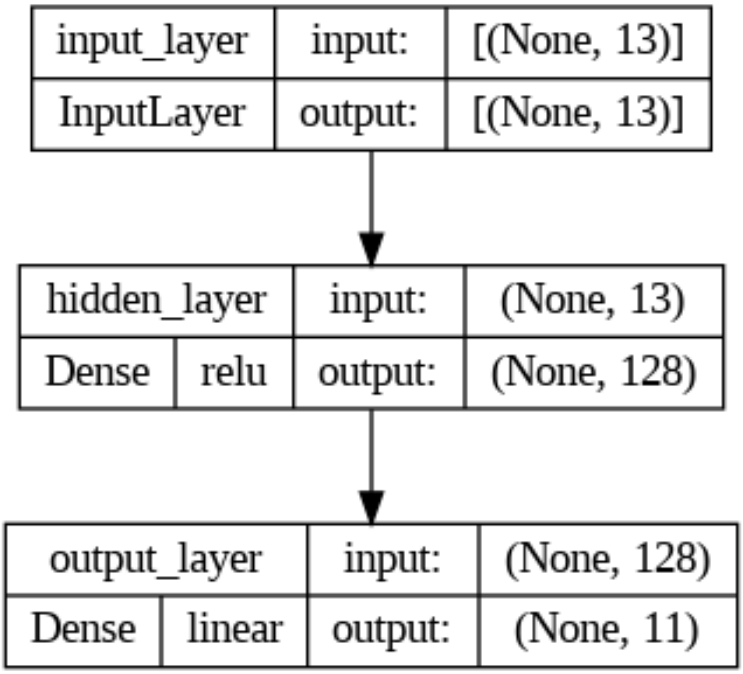

Figure 3.6　Structure of the learning network.

3.7.2　ネットワーク

今回学習するモデルの構造を Figure 3.6 に示す。 入力層には $[0, 1]$ の範囲に正規化された状態を入力する。入力層のニューロンの数は状態の要素数となる。出力層からは各行動の Q 値を出力する。出力層のニューロンの数は行動の数となる。出力された Q 値を用いて、現在の状態で行える行動の中から Q 値が最大になる行動を選択する。今回の場合、状態の要素数は 13 なので入力層のニューロン数は 13、行動数は 11 なので出力層のニューロン数は 11 となる。出力層からは Q 値を出力するため、活性化関数は恒等関数とする。

中間層のニューロンの数は 128 とする。中間層の活性化関数は ReLU とする。ReLU を用いているため、ニューロンの重みの初期化には He の初期化を用いる。

ネットワークの最適化アルゴリズムには Adam を用い、損失関数には Huber 損失関数を用いることとする。

3.7.3　パラメータ

学習に使用するパラメータを Table 3.9 に示す。割引率 γ 、探索確率 ε の値は Q テーブルの学習に使用した値を用いる。

Table 3.9　Parameters of Deep Q-Network.

Parameter	Value
Learning Rate lr	10^{-5}
Discount Factor γ	0.6
Exploration Probability ε	0.5
Batch Size	32
ReplayBuffer Size	10^6

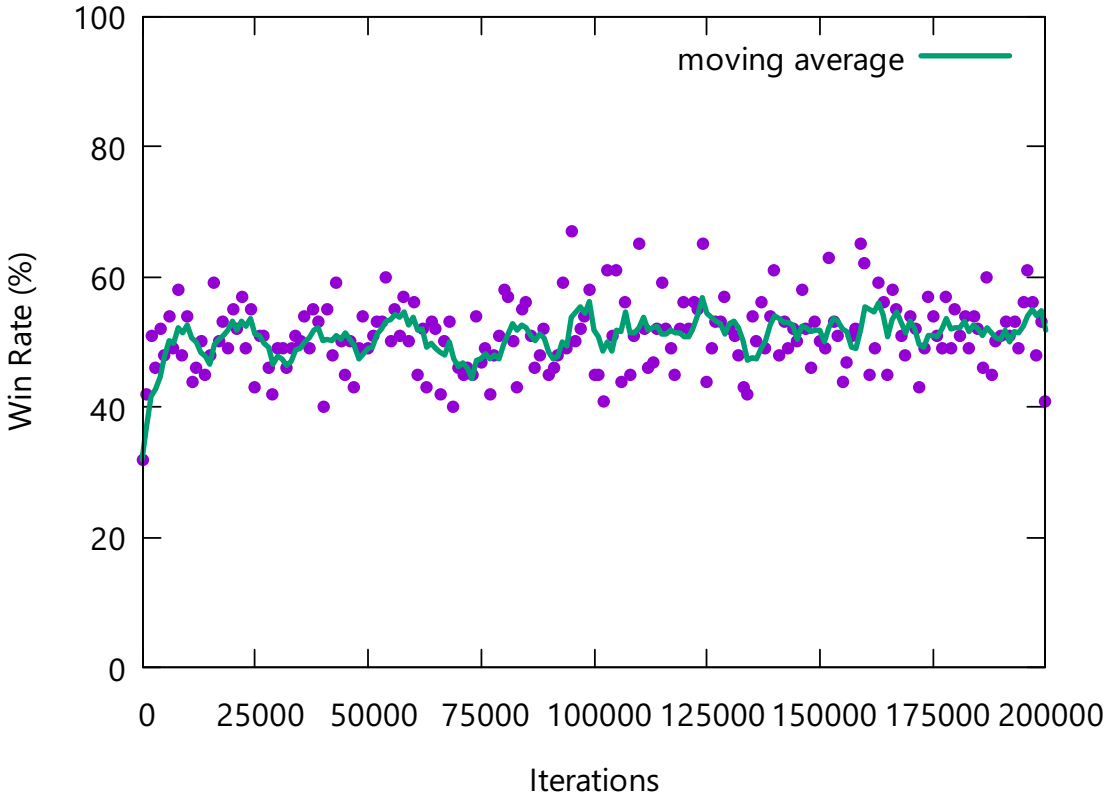

Figure 3.7　Learning results of the Deep Q-Network (13 states).

3.7.4　結果

学習ステップ毎のエピソード数 $n_e = 1$ として、200000 ステップ学習させた結果を Figure 3.7 に示す。勝率の計算は 1000 ステップごとに行っている。データのばらつきが大きいため、移動平均を実線で表示している。移動平均は区間 5 の後方移動平均であり、Iterations が 0 付近で 5 個のデータを取れない場合は取れたデータのみで平均を計算している。 5000 ステップあたりで勝率が 50 % に達した後、ほとんど勝率が上がらなかっ

た。学習後の勝率を 10000 回の対戦によって求めると、51.60 % であった。データにばらつきが生じているのは、勝率を求めるための対戦を 100 回しか行っていないからであり、En Garde がそれだけランダム性の高いゲームだということが分かる。

勝率が 50 % 程度までしか上がらなかった原因として様々なことが考えられるが、ここではまず状態の表し方について考える。Q テーブルとの状態の表し方の違いとして、Q テーブルでは相手との距離を 11 個の離散値として表していたのに対し、Deep Q-Network では 0〜1 の連続値で表している。相手との距離が近い時には、距離が 1 マス異なるだけで行える行動が全く違ってくる。これを連続値で表してしまうと、1 マスの違いにより状態が全く異なるにもかかわらず、相手との距離の数値はネットワークへの入力時に 21 で割られるため、数値的違いは 1/21 しか存在しないことになる。この違いをネットワークが判別するのは難しいため、これが勝率を下げる要因の一つになっていると考えられる。この問題を解決するため、特定の状態を離散値で表すように状態を改良する。

3.7.5 一部の状態の離散化

プレイヤーの位置と相手との距離に対して離散化を行う。状態の離散化には One-Hot エンコーディングを利用する。全ての箇所を One-Hot ベクトルで表すのではなく、Q テーブルの学習の際に個別に状態を割り当てた箇所だけ One-Hot ベクトルで表す。

プレイヤーの位置はこれまでの連続値に加えて、フィールドの端に近い 0〜4 のマスを One-Hot ベクトルによって表したものを状態として扱う。One-Hot ベクトルはプレイヤーが 0〜4 のマスにいるときは対応する要素が 1 となり、その他の要素が 0 となる。プレイヤーが 0〜4 のマスにいないときは、全ての要素が 0 となる。状態数は $1 + 5 = 6$ となる。

相手との距離はこれまでの連続値に加えて、距離が近い 0〜9 の時を One-Hot ベクトルで表したものを状態として扱う。状態数は $1 + 10 = 11$ となる。

その他の状態については、第 3.7.1 項と同様とする。最終的に、状態数は 15 だけ増え、28 となる。

ネットワークの入力が増えたため、中間層のニューロンの数を 256 まで増やすこととする。

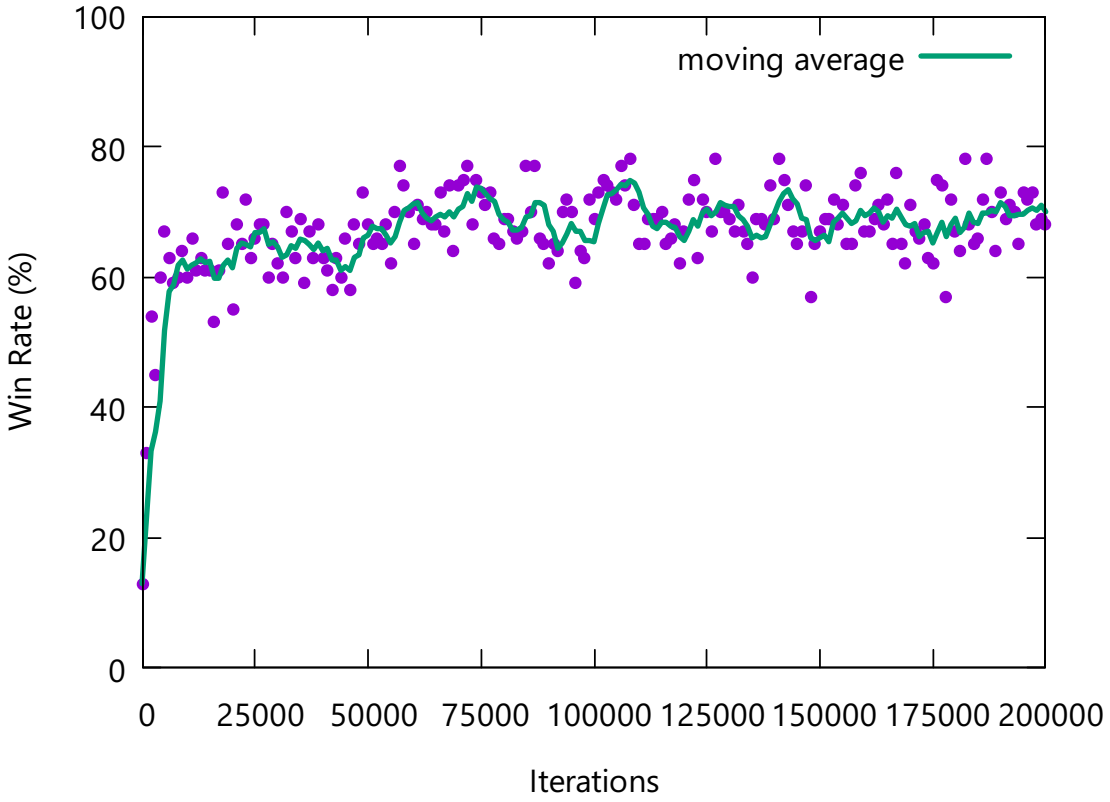

Figure 3.8　Learning results of the Deep Q-Network (28 states).

3.7.6　結果

第 3.7.5 項の状態で、200000 ステップ学習させた結果を Figure 3.8 に示す。グラフの作成方法は第 3.7.4 項と同様とする。5000 ステップあたりで勝率が 60 % に達し、50000 ステップまでは 65 % あたりを推移し、60000 ステップからは 70 % あたりで推移した。学習後の勝率を 10000 回の対戦によって求めると、70.21 % であった。状態を離散化することによって、勝率が 18.61 % 上昇することが確認された。

3.7.7　全ての状態の離散化

状態の離散化によって勝率が上がることが分かったため、全ての状態を One-Hot ベクトルで表してみる。

プレイヤーの位置、相手との距離は 0〜21 の整数で表されるため、これを大きさ 22 の One-Hot ベクトルで表す。状態数はそれぞれ 22 である。山札の枚数は 1〜15 の整数で表されるため、大きさ 15 の One-Hot ベクトルで表す。状態数は 15 である。

プレイヤーの手札、プレイヤーから見えないカードについては、One-Hot エンコーディングではなく Table 3.10 の手法でベクトル化を行う。プレイヤーの手札は 1〜5 の番号の

Table 3.10　Vectorization of the number of cards.

Number of cards	Vector
0	[0, 0, 0, 0, 0]
1	[1, 0, 0, 0, 0]
2	[1, 1, 0, 0, 0]
3	[1, 1, 1, 0, 0]
4	[1, 1, 1, 1, 0]
5	[1, 1, 1, 1, 1]

カードそれぞれが大きさ 5 のベクトルで表される。状態数は 25 である。プレイヤーから見えないカードも同様に表され、状態数は 25 になる。

最終的に状態数は $22 + 22 + 15 + 25 + 25 = 109$ となる。

3.7.8　結果

第 3.7.7 項の状態で、200000 ステップ学習させた結果を Figure 3.9 に示す。グラフの作成方法は第 3.7.4 項と同様とする。5000 ステップあたりで勝率が 60 % に達し、10000 ステップからは 65〜70 % あたりで推移し、60000 ステップからは 70 % あたりで推移した。Figure 3.8 との違いとしては、学習初期の 0〜20000 ステップで勝率が 65 % を超える回数が以前よりも多くなり、学習後期の 150000〜200000 ステップで勝率が 70 % を超える回数も以前より多くなった。これらのことから、全ての状態を One-Hot ベクトルで表すことにより、学習の安定性が上がったと考えられる。学習後の勝率を 10000 回の対戦によって求めると、70.67 % であった。これは第 3.7.6 項とほぼ同じ結果となった。

3.7.9　ネットワークのパラメータ調整

エージェントの勝率を上げるためにネットワークのパラメータ調整を行う。ここで調整するパラメータは、学習率と中間層のニューロンの数とする。

まずは学習率の調整から行う。ニューロンの数を 256 に固定し、学習率を変えて学習を行う。調べる学習率の値は $[10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$ とする。それぞれの学習率で 50000 ステップ学習したときの勝率の値を Figure 3.10 に示す。勝率は 10000 回の対戦によって求めている。学習率が 10^{-4} の時に最も勝率が高くなった。学習率が

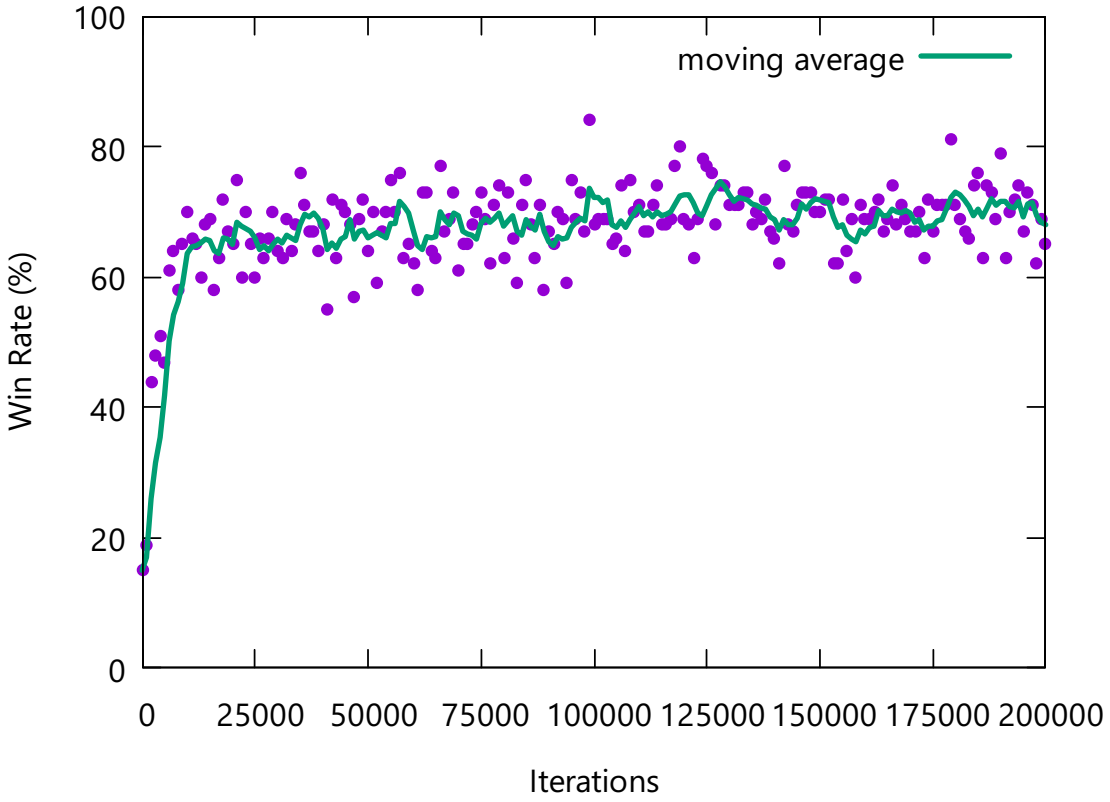

Figure 3.9　Learning results of the Deep Q-Network (109 states).

10^{-4} から小さくなるにつれて勝率が徐々に下がっていき、10^{-7} ではほとんど学習が行われなかった。これは学習率が小さいことで重みの更新が進まなかったからだと考えられる。また、学習率が 10^{-4} から大きくなるにつれて勝率が徐々に下がっていった。これは更新時に重みの値を動かしすぎて最適な値を通り過ぎてしまっているからだと考えられる。最適な学習率が 10^{-5} と 10^{-4} の間にある場合、勝率が最も高い 10^{-4} の時には既に重みの更新量が大きすぎる可能性があるため、より細かい調整ができる 10^{-5} の値を学習率として使用する。

次に中間層のニューロンの数を調整する。学習率を 10^{-5} に固定し、ニューロンの数を変えて学習を行う。調べるニューロン数は $[32, 64, 128, 256, 512, 1024, 2048, 4096]$ とする。それぞれのニューロン数で 50000 ステップ学習したときの勝率の値を Figure 3.11 に示す。勝率は 10000 回の対戦によって求めている。 中間層のニューロン数によって、勝率はほとんど変わらなかった。中間層のニューロンの数が 256 の時に最も勝率が高くなった。

パラメータ調整の結果として、学習率は 10^{-5} 、中間層のニューロンの数は 256 が最適なパラメータであることが分かった。これは第 3.7.7 項のパラメータと同じである。

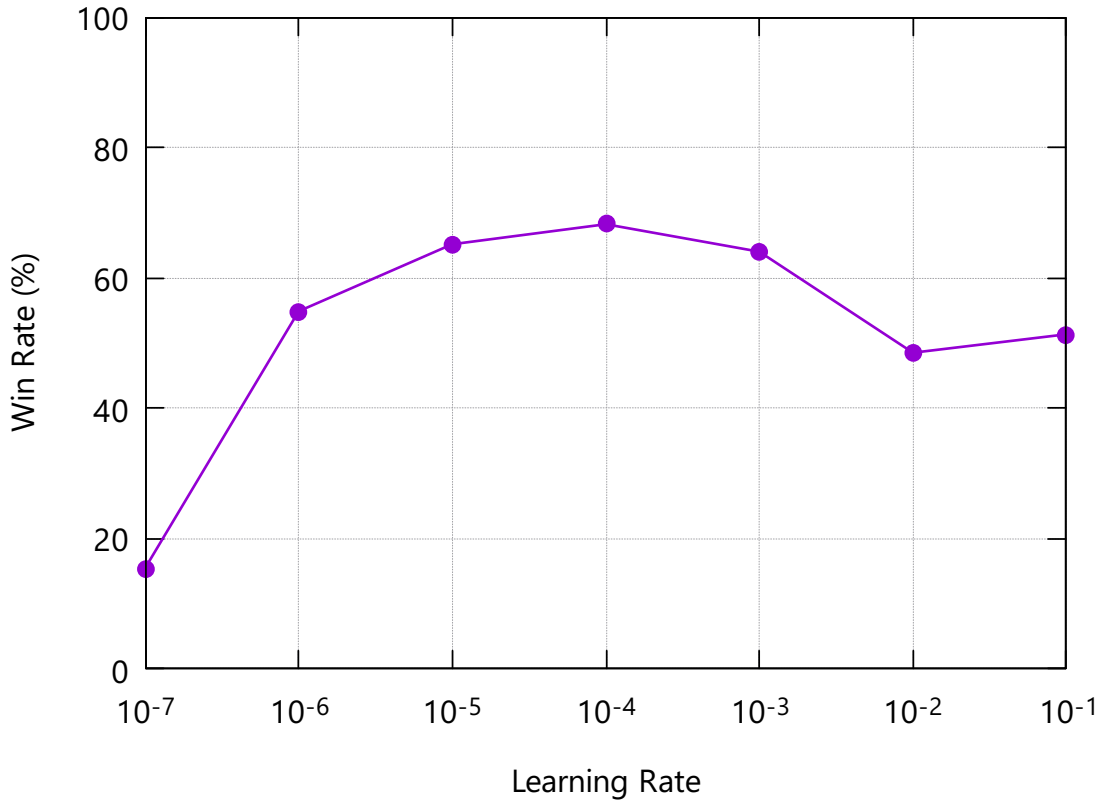

Figure 3.10　Learning rate tuning for Deep Q-Network.

3.7.10　Deep Q-Network の結果

これまでの実験によって、Deep Q-Network での最適な状態の表し方、学習パラメータが分かった。これらの結果を用いて、第 3.6.3 項と同様に 300000 エピソード分の学習を行う。ゲームの状態は第 3.7.7 項の方法で表現し、学習パラメータは Table 3.9 の値を使用する。中間層のニューロンの数は 256 とする。Q-Network 内の学習する重みとバイアスの数は、30987 個となった。第 3.7.8 項で、同様のパラメータで 200000 エピソード学習を行っているため、同項で学習済みのモデルに対して 100000 ステップ追加学習を行う。

学習時の勝率の推移を Figure 3.12 に示す。グラフの作成方法は第 3.7.4 項と同様とする。200000 ステップ後で勝率の上昇は見られなかった。学習後の勝率を 10000 回の対戦によって求めると、69.77 % であった。これは、第 3.7.8 項の勝率 70.67 % とほとんど同じ値であるため、学習が進んでいないことが分かる。

勝率の算定における勝った回数 6977 回の、ゲームの終了条件ごとの内訳を Table 3.11 に示す。Q テーブルを使った AI よりも、攻撃の成功によって勝つ回数が 1000 回程度少なくなった。また、負けた回数 3023 回の内訳を Table 3.12 に示す。Q テーブルを使った AI よりも、相手の攻撃を受けて負ける回数が 1000 回程度多くなった。

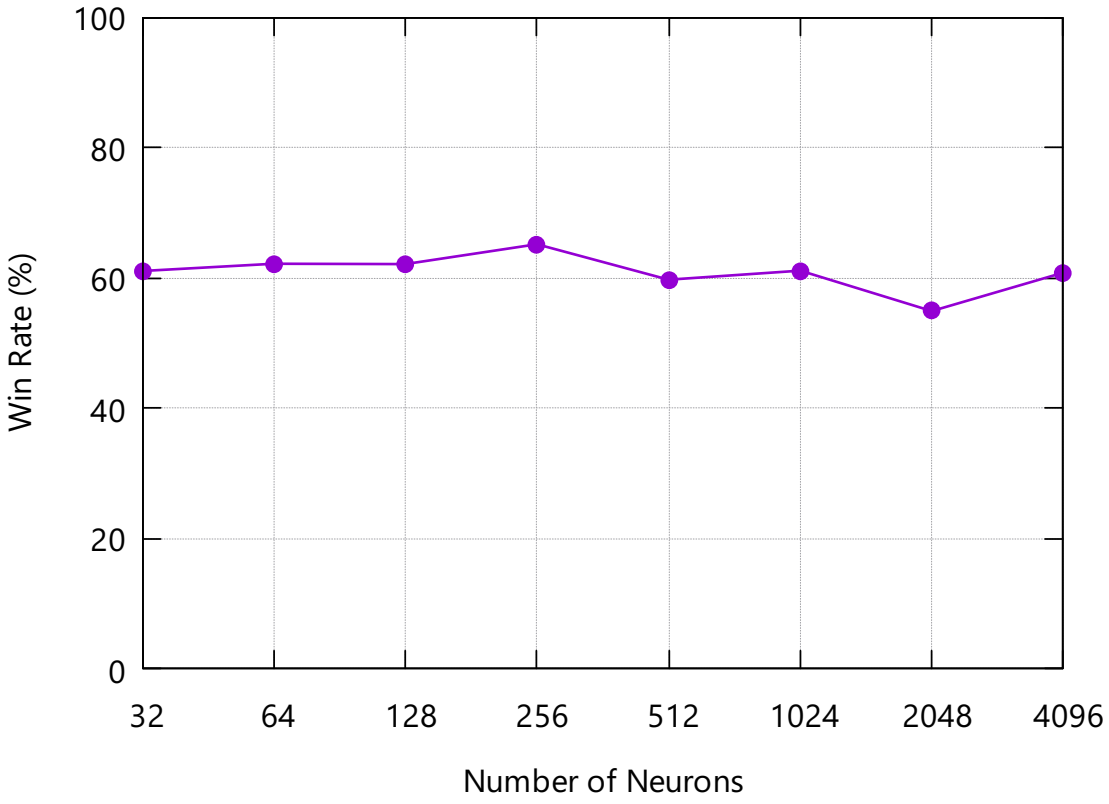

Figure 3.11　Number of neurons tuning for Deep Q-Network.

勝率の算定において相手プレイヤーに攻撃した回数は 11132 回であった。これは、Q テーブルを使った AI よりも 2000 回程度多くなっている。攻撃回数の攻撃の結果ごとの内訳を Table 3.13 に示す。攻撃が成功した回数は、相手からの攻撃をパリーした後に反撃した場合とそうでない場合で分けて表示している。Q テーブルを使った AI と比較すると、先に攻撃を仕掛けて勝つ回数は 1000 回程度多くなったが、相手の攻撃をパリーした後に反撃する回数が 2500 回程度少なくなったため、合計の攻撃が成功した回数は少なくなっている。また、相手に攻撃をパリーされる回数が 3000 回程度多くなり、その中でも相手が反撃できる回数が 1000 回程度多くなった。

相手プレイヤーから攻撃された回数は 2749 回であった。これは、Q テーブルを使った AI よりも 2000 回程度少なくなっている。攻撃回数の攻撃の結果ごとの内訳を Table 3.14 に示す。Q テーブルを使った AI と比較すると、相手の反撃によって負ける回数が 1000 回程度多くなり、相手の攻撃をパリーした回数が 3000 回程度少なくなった。

学習後の AI と対戦することにより、AI の挙動を確認する。AI と対戦した時の、AI の特徴的な挙動を Table 3.15 に示す。表には、AI から見た状態とその時の相手の手札、その時に AI が取った行動、行動後の相手との距離を表示している。状態に書いてある数値

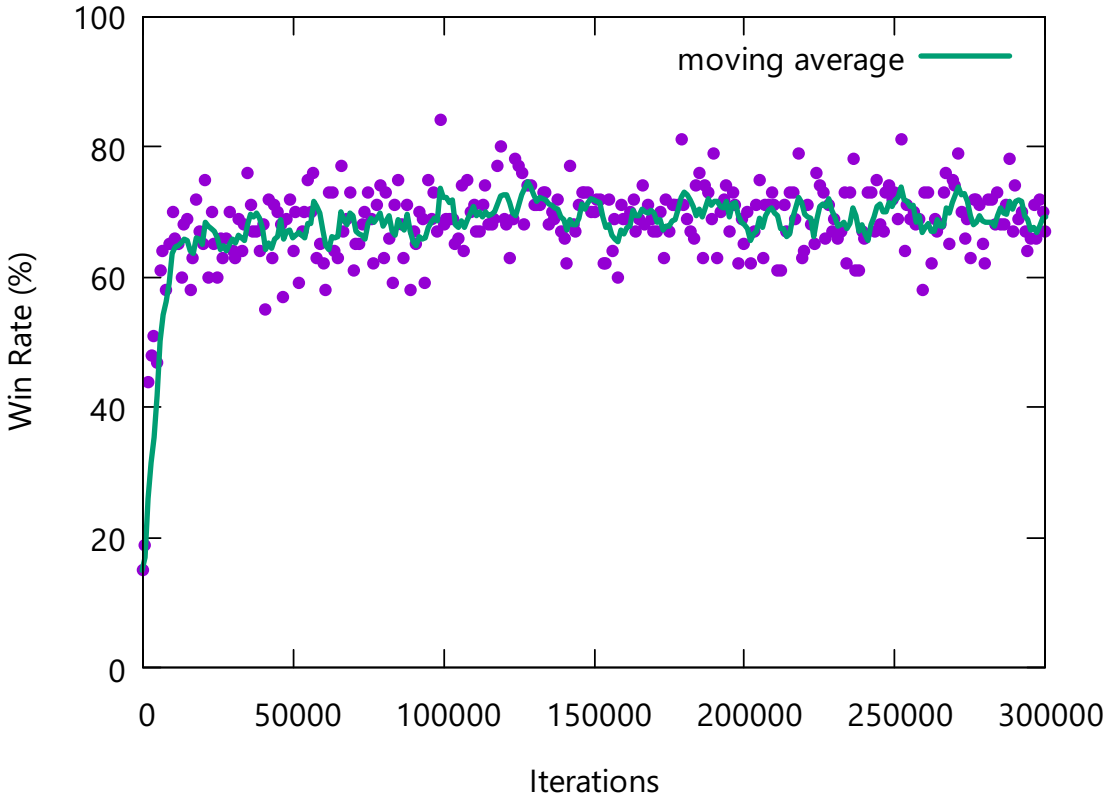

Figure 3.12　Learning results of the Deep Q-Network (300000 episodes).

Table 3.11　Wins per game end condition in Deep Q-Network.

Game End Condition	Number of Wins	Ratio
Attack Successful	6448	92.42 %
Unable to Act	0	0.0 %
Deck Depleted (had more cards)	503	7.21 %
Deck Depleted (ahead of opponent)	26	0.37 %

は左から、プレイヤーの位置、相手との距離、プレイヤーの手札、プレイヤーから見えないカード、山札の枚数である。プレイヤーの位置、相手との距離、プレイヤーの手札については、第 3.3 節の数値を表示している。プレイヤーから見えないカードについては 1〜5 のカードのそれぞれの枚数を表示している。相手の手札はプレイヤーの手札と同じ方法で表示している。AI が取った行動には行動の種類と使ったカードの番号を表示している。

AI から見た状態の、プレイヤーの位置が 4、相手との距離が 8、プレイヤーの手札が

Table 3.12　Losses per game end condition in Deep Q-Network.

Game End Condition	Number of Losses	Ratio
Attack Successful	1856	61.4 %
Unable to Act	100	3.31 %
Deck Depleted (had more cards)	398	13.17 %
Deck Depleted (ahead of opponent)	669	22.13 %

Table 3.13　Number of AI attacks per outcome in Deep Q-Network.

Attack Outcome	Number of Attacks	Ratio
Attack Successful	6130	55.07 %
Attack Successful (Counterattack)	348	3.13 %
Parried with Equal Number of Cards	2953	26.53 %
Parried with More Cards	1701	15.28 %

Table 3.14　Number of opponent attacks per outcome in Deep Q-Network.

Attack Outcome	Number of Attacks	Ratio
Attack Successful	390	14.19 %
Attack Successful (Counterattack)	1630	59.29 %
Parried with Equal Number of Cards	333	12.11 %
Parried with More Cards	396	14.41 %

$[1, 0, 1, 2, 1]$ の時、AI は 3 のカードで前進した。これによって相手との距離は 5 となった。他のカードで前進する選択肢もある中、相手との距離が 5 となる 3 のカードを選んだことから、AI は相手の攻撃範囲 (相手との距離が 4 以下) に入らない位置まで前進するという行動をしていると考えられる。このような、相手との距離を 5 にするような行動は他の状態でもよく見られた。

プレイヤーの位置が 10、相手との距離が 5、プレイヤーの手札が $[1, 2, 1, 1, 0]$ の時、AI

Table 3.15　Behavior of AI in Deep Q-Network.

State	Opponent's hand	Action	Next Distance
4 8 [1, 0, 1, 2, 1] [4, 5, 4, 1, 3] 12	[1, 2, 2, 0, 0]	Forward 3	5
1 6 [1, 1, 0, 2, 1] [0, 1, 2, 3, 2] 3	[0, 0, 1, 3, 1]	Forward 1	5
10 7 [0, 1, 1, 2, 1] [0, 1, 4, 3, 1] 4	[0, 0, 2, 2, 1]	Forward 2	5
10 5 [1, 2, 1, 1, 0] [3, 2, 3, 4, 3] 10	[0, 1, 1, 1, 2]	Backward 3	8
7 5 [1, 2, 0, 1, 1] [3, 2, 2, 4, 2] 8	[0, 1, 1, 1, 2]	Backward 5	10
7 5 [1, 1, 1, 2, 0] [3, 2, 3, 1, 3] 7	[2, 0, 1, 1, 1]	Backward 2	7
5 0 [1, 0, 2, 1, 0] [0, 5, 3, 3, 0] 6	[0, 2, 1, 2, 0]	Attack 1	0
4 3 [0, 1, 2, 1, 1] [4, 2, 2, 3, 2] 8	[1, 2, 0, 1, 1]	Attack 4	3
6 3 [0, 3, 0, 1, 1] [1, 1, 0, 2, 3] 2	[1, 0, 0, 2, 2]	Attack 4	3
11 6 [0, 0, 2, 3, 0] [0, 0, 2, 2, 2] 1	[0, 0, 2, 2, 1]	Forward 3	3
5 5 [2, 0, 1, 0, 2] [1, 2, 1, 2, 0] 1	[1, 2, 0, 2, 0]	Forward 5	0

は 3 のカードで後退した。これ以外の状態でも、相手との距離が 5 の時には、AI は後退することが多かった。このことから、AI は前進すると相手の攻撃範囲に入るとき、後退を選ぶと考えられる。

プレイヤーの位置が 6、相手との距離が 3、プレイヤーの手札が $[0, 3, 0, 1, 1]$ の時、AI は 4 のカード 1 枚で攻撃した。このときの相手の手札は $[1, 0, 0, 2, 2]$ であり、AI の攻撃の結果は相手に反撃を受けて負けてしまった。このことから、AI は攻撃できるカードが 1 枚でもあれば攻撃を行っていると考えられる。

プレイヤーの位置が 11、相手との距離が 6、プレイヤーの手札が $[0, 0, 2, 3, 0]$、山札の枚数が 1 枚の時、AI は 3 のカードで前進した。これによって相手との距離は 3 となり、山札が無くなったときに 4 のカードを多く持っていたことにより AI は勝利した。それまで相手の攻撃範囲に入らないように行動していた AI が、相手の攻撃範囲に入って勝ったことから、山札が残り 1 枚の時には AI は攻撃できる位置に移動するという行動を取ると考えられる。

AI の位置がフィールドの中間より相手側にあるとき、Q テーブルを用いた AI がランダムに行動したのに対して、Deep Q-Network を用いた AI はランダムに行動することな

く、相手の攻撃範囲に入らない位置をとり続けた。

3.7.11 考察

学習したQ-Network内の、学習によって更新される重みとバイアスの数は30987個となった。Qテーブルの学習では、扱う状態を絞り状態数を削減した後でも学習する値は307461個と多かったため、Q-Networkを使うことによって学習するパラメータ数をかなり抑えられていると言える。学習するパラメータ数が多いと学習が収束しなくなるため、複雑な状態を扱うときにはDeep Q-Networkを使う方が良いと考えられる。

300000エピソード学習したときの勝率の値は、200000エピソード学習したときの勝率とほぼ変わらなかった。よって、これ以上学習を行っても勝率は上昇せず、200000エピソードの時点で学習が収束していたと考えられる。

学習後のAIの挙動は、相手の攻撃範囲に入らない位置まで前進し、前進すると相手の攻撃範囲に入るときには後退するというものであった。この戦略をとることによって、AIは相手に攻撃を受けることなく、自身の攻撃範囲に入ってきた相手に攻撃することができると考えられる。また、AIは攻撃できるカードが1枚でもあれば攻撃を行ったが、相手がより多くのカードを持っていた場合に反撃を受けて負けてしまうことも多かった。相手に反撃されるかどうかの判断ができていないと考えられる。また、山札が残り1枚の時に、攻撃できる位置に移動するという行動が見られた。これはQテーブルを用いたAIには見られない行動であり、山札の枚数の情報を利用した行動だと言える。山札が後1枚の時には相手の手札がほぼ確定し、相手を攻撃できる位置に移動すれば高い確率で勝つことができるため、このような行動を学習したと考えられる。

Qテーブルを用いたAIとの違いとして、AIの位置がフィールドの中間より相手側にあるときでも、ランダムな行動を行わなかった。これは、Qテーブルでは状態が複数の要素で表されるときに各要素の値の組み合わせごとにQ値を学習するため、プレイヤーの位置の値が大きいという経験したことのない要素を含む組み合わせではQ値が全く学習されていないが、Q-Networkは各要素とQ値の関係性を学習するため、プレイヤーの位置の経験が活用できなかったとしてもそれ以外の要素の経験を活用して行動できたからだと考えられる。

評価用の対戦相手と10000回対戦を行ったときの結果を用いて、Qテーブルを用いたAIとの比較を行う。Deep Q-Networkを用いたAIが相手プレイヤーに攻撃した回数は、

Q テーブルを用いた AI よりも 2000 回程度多くなった。これは Deep Q-Network を用いた AI が、攻撃できるカードが 1 枚でもあれば攻撃を行っているからだと考えられる。攻撃の内訳において相手に反撃される回数が 1000 回程度多くなっているため、Deep Q-Network を用いた AI は、相手から反撃されうる状態においても攻撃を行ってしまっていることが分かる。また、Deep Q-Network を用いた AI が攻撃によって勝った回数は、Q テーブルを用いた AI よりも 1000 回程度少なくなった。攻撃の内訳を見ると、先に攻撃を仕掛けて勝つ回数は 1000 回程度多くなったが、相手の攻撃をパリーした後に反撃する回数が 2500 回程度少なくなったため、結果として攻撃が成功した回数は少なくなっている。これは、Q テーブルを用いた AI は相手の攻撃範囲に積極的に入り相手に先に攻撃させてから反撃するという戦略をとっていたが、Deep Q-Network を用いた AI は相手の攻撃範囲に入らないという戦略をとっているため、相手の攻撃を受けそれに反撃する回数が減ったからだと考えられる。Deep Q-Network を用いた AI が相手の攻撃範囲に入らないことにより、相手プレイヤーから攻撃された回数が 2000 回程度少なくなっていることが分かる。

3.7.12 Self-Play

第 3.7.10 項のエージェントに対して、Self-Play を用いて追加学習を行う。Self-Play ではエージェント自身の過去のコピーを対戦相手として学習を行う。エージェントのコピーは学習前に 1 つ作成し、その後は 10000 ステップごとに作成することとする。エージェントの対戦相手は 1 エピソードごとに以下のように決定する。

- 50 % の確率でエージェントの最新のコピー
- 30 % の確率でエージェントの以前のコピーから 1 つ選択
- 20 % の確率で第 3.5 節の対戦相手

学習中のエージェントから得られる経験だけでなく、対戦相手から得られる経験も Replay Buffer に追加することによって学習効率を向上させる。300000 ステップ学習が進んだことで探索の必要性が下がったと考えられるため、epsilon-greedy 方策の ε の値は 0.1 とする。

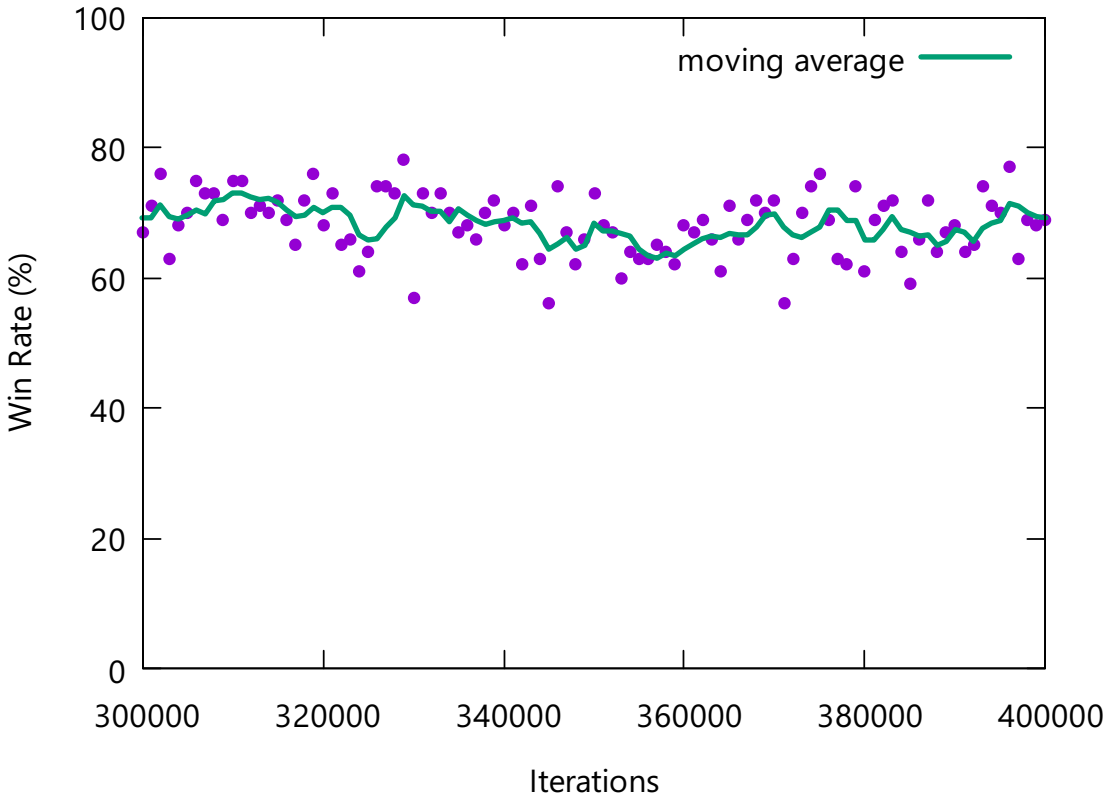

Figure 3.13　Learning results of Self-Play.

3.7.13　Self-Play の結果

第 3.7.10 項のエージェントに対して Self-Play によって 100000 ステップ追加学習させた結果を Figure 3.13 に示す。グラフには追加学習を開始した 300000 ステップ以降の勝率を表示している。勝率は Self-Play の学習開始から少しずつ低下していった。

学習後の勝率を 10000 回の対戦によって求めると、66.83 % であった。Self-Play の学習前のモデルと学習後のモデルとで 10000 回対戦を行うと、学習前のモデルが勝った回数は 4654 回、学習後のモデルが勝った回数は 5346 回であった。学習前のモデルに対しては、学習後のモデルの方が強くなっていることが分かる。

勝率の算定における勝った回数 6683 回の、ゲームの終了条件ごとの内訳を Table 3.16 に示す。また、負けた回数 3317 回の内訳を Table 3.17 に示す。Self-Play を行う前と比較すると、攻撃の成功によって勝つ回数が 500 回程度少なくなった。また、相手の攻撃によって負ける回数はほとんど変わっていないが、山札がなくなったときに負ける回数が 200 回程度増加していた。

勝率の算定において相手プレイヤーに攻撃した回数は 9611 回であった。これは、Self-Play を行う前よりも 1500 回程度少なくなった。攻撃回数の攻撃の結果ごとの内訳を Ta-

Table 3.16　Wins per game end condition in Self-Play.

Game End Condition	Number of Wins	Ratio
Attack Successful	5917	88.54 %
Unable to Act	1	0.01 %
Deck Depleted (had more cards)	721	10.79 %
Deck Depleted (ahead of opponent)	44	0.66 %

Table 3.17　Losses per game end condition in Self-Play.

Game End Condition	Number of Losses	Ratio
Attack Successful	1885	56.83 %
Unable to Act	185	5.58 %
Deck Depleted (had more cards)	486	14.65 %
Deck Depleted (ahead of opponent)	761	22.94 %

ble 3.18 に示す。Self-Play を行う前と比較すると、先に攻撃を仕掛けて勝つ回数は 1000 回程度少なくなったが、相手の攻撃をパリーした後に反撃する回数が 500 回程度多くなった。また、相手に攻撃をパリーされる回数が 1000 回程度少なくなり、その中でも相手が反撃できる回数が 400 回程度少なくなった。

相手プレイヤーから攻撃された回数は 3578 回であった。これは、Self-Play を行う前よりも 800 回程度多くなった。攻撃回数の攻撃の結果ごとの内訳を Table 3.14 に示す。Self-Play を行う前と比較すると、相手の反撃によって負ける回数が 400 回程度少なくなり、相手の攻撃をパリーした回数が 900 回程度多くなった。

Table 3.15 と同じ状態の時に、Self-Play で学習後の AI が行う行動を Table 3.20 に示す。表の作成方法は Table 3.15 と同じとする。

Self-Play での学習前との違いとして、プレイヤーの位置が 4、相手との距離が 8、プレイヤーの手札が $[1, 0, 1, 2, 1]$ の時、学習前の AI は 3 のカードを使って相手の攻撃範囲に入らない位置まで前進したが、学習後の AI は 5 のカードで前進し相手の攻撃範囲に入っている。学習後の AI が行動した後、相手との距離は 3 となった。このとき AI は、攻撃

Table 3.18　Number of AI attacks per outcome in Self-Play.

Attack Outcome	Number of Attacks	Ratio
Attack Successful	5181	53.91 %
Attack Successful (Counterattack)	792	8.24 %
Parried with Equal Number of Cards	2382	24.78 %
Parried with More Cards	1256	13.07 %

Table 3.19　Number of opponent attacks per outcome in Self-Play.

Attack Outcome	Number of Attacks	Ratio
Attack Successful	806	22.53 %
Attack Successful (Counterattack)	1188	33.2 %
Parried with Equal Number of Cards	716	20.01 %
Parried with More Cards	868	24.26 %

される可能性のある 4 のカードを 2 枚持っており、相手の攻撃をパリーできる状態であることが分かる。ここから、学習後の AI は相手の攻撃をパリーできる可能性があるとき、相手の攻撃範囲に入るという行動をしていると考えられる。

プレイヤーの位置が 4、相手との距離が 3、プレイヤーの手札が $[0, 1, 2, 1, 1]$ の時、学習前の AI は 4 のカード 1 枚で攻撃していたが、学習後の AI は 2 のカードで後退し相手の攻撃範囲外に移動している。このとき、プレイヤーから見えない 4 のカードは 3 枚あり、相手に攻撃した場合反撃を受ける可能性が高いため、学習後の AI は後退を選択したと考えられる。

3.7.14　考察

Self-Play の手法を用いて学習した結果、評価用の対戦相手に対しての勝率は 67 % に下がったが、Self-Play の学習前の AI に対しての勝率は 53 % となった。自分自身との対戦によって学習しているため、評価用の対戦相手に適した行動はしなくなったが、以前の自分自身に対しては強くなったと考えられる。

Table 3.20　Behavior of AI in Self-Play.

State	Opponent's hand	Action	Next Distance
4 8 [1, 0, 1, 2, 1] [4, 5, 4, 1, 3] 12	[1, 2, 2, 0, 0]	Forward 5	3
1 6 [1, 1, 0, 2, 1] [0, 1, 2, 3, 2] 3	[0, 0, 1, 3, 1]	Forward 5	1
10 7 [0, 1, 1, 2, 1] [0, 1, 4, 3, 1] 4	[0, 0, 2, 2, 1]	Forward 2	5
10 5 [1, 2, 1, 1, 0] [3, 2, 3, 4, 3] 10	[0, 1, 1, 1, 2]	Backward 4	9
7 5 [1, 2, 0, 1, 1] [3, 2, 2, 4, 2] 8	[0, 1, 1, 1, 2]	Forward 5	0
7 5 [1, 1, 1, 2, 0] [3, 2, 3, 1, 3] 7	[2, 0, 1, 1, 1]	Forward 4	1
5 0 [1, 0, 2, 1, 0] [0, 5, 3, 3, 0] 6	[0, 2, 1, 2, 0]	Attack 1	0
4 3 [0, 1, 2, 1, 1] [4, 2, 2, 3, 2] 8	[1, 2, 0, 1, 1]	Backward 2	5
6 3 [0, 3, 0, 1, 1] [1, 1, 0, 2, 3] 2	[1, 0, 0, 2, 2]	Attack 4	3
11 6 [0, 0, 2, 3, 0] [0, 0, 2, 2, 2] 1	[0, 0, 2, 2, 1]	Forward 3	3
5 5 [2, 0, 1, 0, 2] [1, 2, 1, 2, 0] 1	[1, 2, 0, 2, 0]	Forward 5	0

Self-Play での学習後の AI は、相手の攻撃をパリーできる可能性があるときに相手の攻撃範囲に入るようになった。また、相手に反撃を受ける可能性が高いときに後退を行うようになった。これらのことから、学習前よりも Q テーブルを用いた AI に近い戦略を行うようになったと考えられる。

評価用の対戦相手と 10000 回対戦を行ったときの結果を、Self-Play を行う前と比較すると、相手に攻撃する回数が 1500 回程度少なくなった。これは、相手に反撃を受ける可能性が高いときに後退を行うようになったためだと考えられる。相手に攻撃をパリーされる回数が 1000 回程度少なくなり、相手に反撃される回数が 400 回程度少なくなったことからも、Self-Play での学習前より相手に反撃されないように行動していることが分かる。また、相手プレイヤーから攻撃された回数が 800 回程度多くなり、相手の攻撃をパリーした後に反撃する回数が 500 回程度多くなった。このことから、Self-Play での学習前よりも、相手の攻撃範囲に入り、相手の攻撃に対して反撃する行動を多くとっていることが分かる。山札がなくなったときに負ける回数が 200 回程度増加していたが、これは相手に攻撃する回数が少なくなったことで、山札が無くなるまで試合が続くことが多くなったためだと考えられる。

第4章　結論

本研究の目標はゲーム「En Garde」で基本的な方策に勝つことができるAIを制作することであった。ゲームの学習手法にはQ学習を用い、Qテーブルを使う手法とDeep Q-Networkの2つの手法で学習を行った。

Qテーブルを使う手法では、プレイヤーの位置、相手との距離、プレイヤーの手札の情報のみを用いて学習を行った。パラメータ調整を行い学習した結果、AIは評価用の対戦相手に対して81 % の確率で勝つことができた。100 % には達しなかったが、ランダム要素があるゲームでは100 % の確率で勝つことが難しいため、制作したAIは基本的な方策に勝つことができていると言える。

Deep Q-Networkでは、さらに扱う情報を増やし、場に公開されているカード、山札の枚数の情報も用いて学習を行った。しかし、各状態を連続値としてQ-Networkに入力すると、評価用の対戦相手に対する勝率は52 % と、ほぼ互角の結果になった。この対策として、各状態にOne-Hotエンコーディングを行うことによって、勝率を70 % まで上げることができた。扱う情報を増やすことにより勝率が向上することを期待したが、結果はQテーブルを用いたAIよりも勝率が下がってしまった。Deep Q-Networkによる学習では状態の表し方によって勝率が大きく変わることが分かったため、よりよい状態の表し方を模索することで勝率を向上できると考える。

Deep Q-Networkを用いたAIにSelf-Playの手法を用いて学習すると、評価用の対戦相手に対しては弱くなったが、学習前のAIに対しては強くなった。基本的な方策に勝つという目的ではなく、他の方策にも通用する強いAIを制作するという目的ならば、Self-Playの手法を使うことが適切だと考えられる。

本研究の課題として、Deep Q-Networkを用いたAIの勝率がQテーブルを用いたAIの80 % に届かなかった原因が分かっていない。この原因を調べるために、Qテーブルの学習で使用した状態をOne-Hotベクトルで表し、それを用いてQ-Networkの学習を行うという実験をする必要がある。また、第3.7.7項の実験では、プレイヤーの手札、プレイヤーから見えないカードに対してOne-Hotエンコーディングとは違う手法でベクトル化を行ったが、この手法が有効かどうかが分かっていない。この手法の有効性を調べるために、One-Hotエンコーディングを利用した場合との比較を行う必要がある。

参考文献

1) 総務省, “AI に関する基本的な仕組み”, 総務省, https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/html/nd113210.html, (参照 2023-12-20).
2) 総務省統計研究研修所監修 株式会社 Rejoui 制作, “「高等学校における「情報 II」のためのデータサイエンス・データ解析入門」”, 総務省統計局, https://www.stat.go.jp/teacher/comp-learn-04.html, (参照 2023-12-20).
3) Ryan Holbrook, “Intro to Deep Learning”, Kaggle, https://www.kaggle.com/learn/intro-to-deep-learning, (参照 2024-2-3).
4) TensorFlow, “tf.keras.activations.relu”, TensorFlow, https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu, (参照 2024-1-31).
5) Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, arXiv, https://arxiv.org/abs/1502.01852, (参照 2024-1-31).
6) TensorFlow, “tf.keras.initializers.HeNormal”, TensorFlow, https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeNormal, (参照 2024-1-31).
7) Diederik P. Kingma, Jimmy Ba, “Adam: A Method for Stochastic Optimization”, arXiv, https://arxiv.org/abs/1412.6980, (参照 2024-2-4).
8) The TF-Agents Authors, “Introduction to RL and Deep Q Networks”, TensorFlow, https://www.tensorflow.org/agents/tutorials/0_intro_rl, (参照 2024-1-4).
9) Thomas Simonini, “Deep Reinforcement Learning Course”, Hugging Face, https://huggingface.co/learn/deep-rl-course/unit0/introduction, (参照 2024-1-4).
10) 中部大学 機械知覚&ロボティクスグループ, “深層強化学習と活用するためのコツ”, 電子情報通信学会, https://www.ieice.org/~prmu/jpn/ieice/2018/dt_01_004s.pdf, (参照 2024-1-6).
11) PyTorch, “HUBERLOSS”, PyTorch, https://pytorch.org/docs/stable/generated/torch.nn.HuberLoss.html, (参照 2024-1-6).
12) TensorFlow, “TensorFlow Agents”, TensorFlow, https://www.tensorflow.org/

agents?hl=ja, (参照 2024-1-23).

13) Rob Robinson, “English Rules Translation v1.2”, BoardGameGeek, https://boardgamegeek.com/filepage/231134/en-garde-english-rules-full-colour-original-format, (参照 2023-12-28).

14) 鹿野和宏, “重複組合せの現代的解法”, 数研出版, https://www.chart.co.jp/subject/sugaku/suken_tsushin/85/85-5.pdf, (参照 2024-1-13).

謝辞

最後に、本研究を進めるにあたり、ご多忙中にも関わらず多大なご指導をしていただきました出口利憲先生、また共に勉学に励んだ同研究室のメンバーに厚く御礼申し上げます。