修 士 論 文 の 和 文 要 旨

研究科・専攻	電気通信大学大学院　情報理工学研究科 情報・ネットワーク工学専攻　博士前期課程		
氏　　名	山岡 勇太	学籍番号	1831162
論 文 題 目	CFR 法と強化学習法の格闘ゲーム人工知能への適用		

要　旨

本研究では，二人不完全情報ゲームでありビデオゲームでもある対戦型格闘ゲームに CFR 法と強化学習法を適用し，両手法の性能を比較する．

CFR 法はゲーム木を深さ優先探索することで，二人不完全情報ゲームの ε 均衡点を求めることができる手法である．また，CFR 法の改良手法がいくつか提案されている．例えば，CFR 法にモンテカルロ木探索を用いた MCCFR が，ポーカーの一種であるヘッズアップノーリミットホールデムで成果を挙げた．

強化学習法は機械学習法の一種である．本研究では，強化学習法に Q 学習法を用いる．Q 学習法を改良した Deep Q-Network は，一人でプレイするビデオゲームである Atari2600 の 49 個のゲームのうち，29 個で人間と同等かそれ以上の性能を持つ AI の開発を可能にした．

本研究は CFR 法と Q 学習法を格闘ゲームに適用し，両手法の性能を搾取量と勝率の 2 つで比較する．格闘ゲームには，人工知能（AI）研究用格闘ゲーム FightingICE を用いる．FightingICE には，AI が受け取るゲーム状況が遅延する(これをディレイと呼ぶ）という特徴がある.FightingICE に CFR 法を適用するにあたりゲーム状況を抽象化するため，ゲームルールの調査と，調査結果に基づいてエミュレータを実装した．これを用いて CFR 法と Q 学習法の実験を行なった．ゲーム状況は，純戦略上の均衡点が存在する・しない，ディレイあり・なしの計 4 通りを用いた．実験の結果，純戦略上の均衡点が存在する場合は両手法とも同程度の性能であることを確認した．純戦略上の均衡点が存在しない場合，Q 学習法は決定論的なプレイヤに対して搾取されうるが，均衡戦略をとるプレイヤに対しては，良好な性能を発揮することがあることが確認された. CFR 法は均衡点を求めることはできるが，1 回の木の探索に要する計算時間やメモリ消費量の観点から，よりオリジナルのルールに近いゲームに適用することが困難であることがわかった．また，複合プレイヤを構成し Q 学習法の性能改善を試みた．結果として，搾取量は大幅に改善されたが，その一方で勝率はやや悪化した．

電気通信大学大学院情報理工学研究科
情報・ネットワーク工学専攻情報数理工学プログラム修士論文

CFR 法と強化学習法の格闘ゲーム人工知能への適用

令和 2 年 3 月 18 日

学籍番号 1831162

山岡勇太

指導教員　保木邦仁
伊藤毅志

目次

1　はじめに

ゲームを上手にプレイする人工知能 (AI) を開発するための研究は古くから行われており，近年では機械学習法を用いた AI の開発事例が数多く報告されている． バックギャモンでは 1994 年に，強化学習法の一種である TD(λ) 法と深層学習法を用いて開発された TD-Gammon が，人間の上級者に匹敵する性能を示した [1]． 強化学習と深層学習を用いた強い AI の開発事例としてはこれが初である．囲碁では 2017 年に，AlphaGo が世界トッププレイヤであるカ・ケツに勝利した [2]．AlphaGo は，強化学習法と深層学習法を用いて開発され，さらに豊富な計算資源を活用することで，通常では 100 年以上かかる学習を数日で行った．また同年には，ポーカーの一種であるヘッズアップノーリミットテキサスホールデム (HUNL) で，Deep Stack が初めてプロのプレイヤに勝利し [3]，2018 年には同ゲームで Libratus がプロの世界ランク上位 4 人と対戦し，統計的有意に勝ち越した [4]．Deep Stack は，2 人不完全情報ゲームの ε 均衡点を求めることができる CFR 法 [5] を改良した CFR+ [6] と深層学習法を用いて開発された．Libratus も，CFR+ と CFR 法の改良手法の一つである Monte Carlo CFR (MCCFR) [7] を用いて開発された．バックギャモンは状態数が 10^{20} 程度あるのに対し，囲碁は 10^{170} 程度，HUNL は 10^{160} 程度あると言われている．さらに，HUNL は不完全情報性を持つため，囲碁と同等かそれ以上に複雑なゲームと言われている．このように非常に膨大な状態集合を持つ複雑なゲームに対しても，豊富な計算資源を活用した機械学習法を用いることで人間のプロを超えるような AI が開発可能になった．

先に挙げたボードゲーム・カードゲームの他に，ビデオゲームの研究も行われている．2015 年には Atari2600 に含まれる 49 のゲームのうち，29 のゲームで人間の上級者と同等かそれ以上の性能を持つ AI が開発された．この AI は Deep Q-Network という，強化学習法の一種である Q 学習と深層学習法を併用した手法を用いて開発された [8]．Deep Q-Network はゲーム画像のみから学習を行うことが可能という点で非常に革新的であり，現在はこれを改良した手法が数多く提案されている．

2 人不完全情報ゲームであり，かつビデオゲームでもあるゲームの一つに対戦型格闘ゲームがある． 本研究の目的は，不完全情報ゲームで成果をあげた CFR 法と，ビデオゲームで成果をあげた強化学習法を格闘ゲームに適用し，両手法の性能を比較し評価することである．本研究で使用する格闘ゲームは，立命館大学の知能エンターテインメント研究室 (Intelligent Computer Entertainment Lab.，ICE) が開発した AI 研究用格闘ゲームの FightingICE である [9]．また同研究室は，モンテカルロ木探索を用いて勝率が最大となるように意思決定を行う MctsAi を公開した [10]． 本研究では，これとの性能比較も行う．

2　基礎知識

本章では，本研究で必要となる基礎知識を紹介する．

2.1　FightingICE

FightingICE とは，2013 年に ICE が開発した AI 研究用対戦型格闘ゲームである (図 1 参照)．FightingICE はゲームを進行させる対戦マネージャと AI 2 つで構成される．対戦マネージャは 2 体のキャラクタからなる現在のゲーム状況を AI に送り，AI は送られてきたゲーム状況を元にキャラクタの次の行動を決定し対戦マネージャに送る．対戦マネージャは 2 つの AI から行動を受け取った後にゲーム状況を更新し，再び AI に新たなゲーム状況を送る． これらのようなメッセージの通信を繰り返してゲームが進行する．本節では，ゲーム一回の流れを説明したのち，キャラクタの行動と体勢について説明し，最後に FightingICE の公式大会である Fighting Game AI Competition のルールを紹介する．

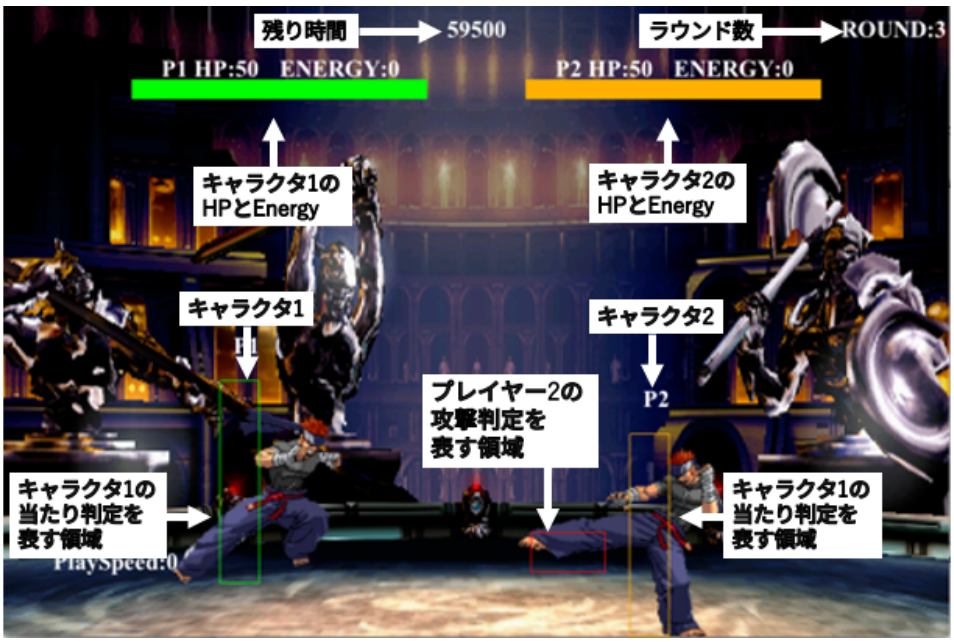

図 1　FightingICE のプレイ中のゲーム画面の一例． 2 つの AI それぞれがキャラクタを操り格闘する．

2.1.1　ゲーム 1 回の流れ

初めに，各 AI は使用するキャラクタのタイプを決定する． 使用可能なタイプは ZEN，GARNET，LUD の 3 種である．タイプを決定したのち格闘戦を開始する． 1 戦は 3 ラウンドからなり，1 ラウンドは 60 秒である．3 ラウンド後，勝利したラウンド数が多いプレイヤがゲームの勝者となる． 各ラウンドの勝利条件は以下の 2 つのどちらか一方を満たすことである．

1. 制限時間内に相手の HP を 0 にすること
2. ラウンド終了時の HP が相手よりも高いこと

ただし，制限時間内に自キャラクタと相手キャラクタの HP が同時に 0 になるか，またはラウンド終了時のお互いのキャラクタの HP が等しい場合は引き分けとなる．

各ラウンド中，プレイヤはフレーム単位でゲーム状況を受け取り，キャラクタの行動を決定する． フレームとはビデオゲームにおける時間の最小単位のことで，FightingICE では 1 秒間は 60 フレームである． 各フレームごとに各 AI が受け取るゲーム状況 (フレームデータ) は主に以下の要素からなる．

- 現在のラウンド数
- ラウンド開始からの経過フレーム数
- ラウンド開始からの経過時間
- 2 体のキャラクタが持つパラメータの値
  - HP
  - Energy (行動に必要なポイント)
  - Hit Box (キャラクタの当たり判定を表す領域で，プレイヤ 1 なら緑色の長方形，プレイヤ 2 なら橙色の長方形で表される) の左右の x 座標，上下の y 座標
  - キャラクタの移動速度と向いている方向
  - 行動 (後述) に関する情報
  - 体勢 (後述)
- エネルギー弾 (後述) に関する情報

ただし FightingICE では，プレイヤが受け取るフレームデータは 15 フレーム前のものである．

2.1.2 キャラクタの行動と体勢

まずキャラクタの行動について説明する． 行動には相手にダメージを与える攻撃行動と，それ以外の非攻撃行動がある．行動は 1 回のキー入力で実行されるものと，複数のキーを連続で入力することで実行されるものがある． 本研究では，複数のキーの連続入力は中断しないものと仮定する．

非攻撃行動には移動行動と防御行動がある．移動行動はキャラクタの位置を変更することができる． 例えば，ジャンプやダッシュやバックステップがある． 移動行動は，相手との距離を調整したり，相手の攻撃を避けるために用いられる．防御行動は相手の攻撃を防御することができる．

攻撃行動は主にパンチやキックなどの通常技と，防御できない投げ技と，ダメージの大きい必殺技の 3 つがある．必殺技は，格闘中に溜まるポイントを消費することで使用することができ，相手が防御している場合でも多少のダメージを与えることができる．また，攻撃行動には当たり判定があり，キャラクタの当たり判定を表す領域と重なると，そのキャラクタにダメージを与えることができる．

次にキャラクタの体勢について説明する． 体勢には基本体勢と復帰体勢の 2 種がある．基本体

勢はキャラクタが行動を選択可能であるときの体勢である． キャラクタが使用可能な行動は体勢に応じて変わる．復帰体勢は，キャラクタが相手の攻撃を受けたときや，地上に着地したときの体勢である． キャラクタが復帰体勢を取っている場合は行動の選択ができない．

2.1.3 Fighting Game AI Competition のルール

Fighting Game AI Competition には，参加者が提出した AI をトーナメント方式でプレイさせる Standard League と，モンテカルロ木探索により行動を決定する MctsAi を倒す早さを競う Speedrunning League の 2 種類がある．各リーグですべてのプレイヤは同じキャラクタタイプを使用しなければならない．また，各キャラクタごとに Standard League と Speedrunning League を行うので，計 6 セットのリーグを行う． 各リーグの共通ルールと個別ルールを以下に示す．

両リーグの共通ルール

- リーグの最終スコアは，自動車レースとして知られている F1 世界選手権のポイントシステムに従って付けられる．
- 各プレイヤに対し，ラウンド開始前 5 秒の処理時間が与えられる．
- 最大 1GB のメモリを使用することができる．
- 使用可能なスレッドは 1 つとする．
- ファイルの入出力を最大 50MB まで行うことができる． ただし，ファイル名は 20 文字以下でなければならない．
- 各キャラクタの初期 HP は 400，Energy は 0 とする．

Standard League のルール

- 自分以外の AI と 2 回ずつゲームを行う． また，1 回のゲームごとに各キャラクタの位置を入れ替える．
- 勝利したラウンドの数が多い AI ほど上位となる．

Speedrunning League のルール

- MctsAi と 5 回ゲームを行う． 自キャラクタの初期位置は左側，MctsAi の初期位置は右側とする．
- MctsAi を倒す時間の合計が少ない AI ほど上位となる．
- 60 秒以内に MctsAi を倒せなかった場合はペナルティとして 70 秒で倒したものとする．

参加者は，リーグの最終スコアすべての合計を競う．

2.2 Q 学習

本節の内容は，文献 [11], [12], [13] の内容を参考にして，本研究で用いる Q 学習法を要約し説明する．Q 学習は強化学習の一種である．強化学習とは，意思決定を行うエージェントが，エージェント以外のすべての要素から構成される環境と相互作用を行いながら，目標を達成するための挙動を学習する理論的枠組みのことである．エージェントが意思決定して，環境に行うはたらきかけのことを行動といい，行動によって変化する環境の観測可能な要素を状態と呼ぶ．また，行動の結果得られる環境で起こる様々なことを比較するための指標を報酬という．エージェントと環境が相互に作用する過程はいくつかの数理モデルによって表現される． もっとも代表的なモデルはマルコフ決定過程 (Markov Decision Process, MDP) である．

2.2.1 MDP とエージェントの方策

MDP とは，エージェントと環境が相互に作用する過程を状態，行動，報酬といった概念で表現する数理モデルの一種である． 本論文では，状態や行動の数は有限であるとする．マルコフ決定過程は，状態集合 $\mathcal{S}$，状態 s の行動集合 $\mathcal{A}(s)$，初期状態分布 P_0，状態遷移確率 $P(s'|s,a)$，報酬関数 $r(s,a,s')$ からなる確率過程である．以下，各要素について説明する．

状態集合 $\mathcal{S}$

$\mathcal{S}$ は，すべての状態からなる集合である． 例えば，状態が n 個ある場合の $\mathcal{S}$ は以下のようになる．

$$\mathcal{S} = \{s_1, \cdots, s_n\}$$

行動集合 $\mathcal{A}$

行動集合 $\mathcal{A}$ は，エージェントの行動すべてからなる集合である． 例えば，行動が m 個ある場合の $\mathcal{A}$ は以下のようになる．

$$\mathcal{A} = \{a_1, \cdots, a_m\}$$

状態 s の行動集合 $\mathcal{A}(s)$

行動集合 $\mathcal{A}(s)$ は，状態 s においてエージェントが選択可能な行動すべてからなる集合である．

初期状態分布 P_0

初期状態分布 P_0 は，初期状態 S_0 が生起する確率分布である． S_0 は，エージェントの意思決定とは無関係に生起する．

状態遷移確率 $P(s'|s,a)$

エージェントが状態 s において行動 a を取った場合，s' に遷移する確率は条件付き確率 $P(s'|s,a)$ で表される．このように，次の状態への遷移確率が直前の状態と行動にのみ依存するような確率過程の性質をマルコフ性という．

報酬関数 $r(s,a,s')$

環境は，現在の状態 S_t と行動 A_t，および次の状態 S_{t+1} に応じて報酬 R_{t+1} を決定する．R_{t+1} は報酬関数 r によって定まる．

$$R_{t+1} = r(S_t, A_t, S_{t+1})$$

エージェントの挙動を定める方策 π とは，任意の状態 s に対して，$\mathcal{A}(s)$ 上の確率分布を定める関数である．任意の状態 s に対して，確率 1 である行動を選択する方策を決定論的方策と呼び，そうでない方策は確率的方策と呼ぶ．状態 s において行動 $a \in \mathcal{A}(s)$ を選択する確率は $\pi(a|s)$ で表される．

以上の要素によって生起する状態，行動，報酬の列 $S_0A_0S_1R_1 \cdots S_TR_T$ をエピソードと呼ぶ．ここで T は，エピソードの終了時間ステップである．強化学習の目的は，様々なエピソードにおける報酬の和 $R_1 + R_2 + \cdots + R_T$ の期待値を最大化することである．以降では，報酬の和を収益と呼ぶ．

2.2.2 価値関数

様々なエピソードにおける収益の期待値を最大化することを考える． まず，状態 s において，方策 π に従ったときに得られる収益の期待値 $V^\pi(s)$ を以下の式で定義する．

$$V^\pi(s) = \mathrm{E}\left[R_1 + R_2 + \cdots + R_T | S_0 = s\right] \tag{1}$$

$$= \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}} P\left(s'|s,a\right)\left\{r\left(s,a,s'\right) + V^\pi(s')\right\} \tag{2}$$

この $V^\pi(s)$ を s の状態価値と呼ぶ． また，この式は状態価値関数 V^π に関するベルマン方程式と呼ばれている．次に，この状態価値を用いて方策の良さを考える． 方策 π が方策 π' よりも良い方策であるとは，以下の条件が成立することである．

$$\begin{aligned} &\forall s \in S, V^\pi(s) \geq V^{\pi'}(s) \\ &\exists s \in S, V^\pi(s) > V^{\pi'}(s) \end{aligned} \tag{3}$$

これより，最も良い方策とは以下の条件を満たす方策 π^* である．

$$\forall s \in \mathcal{S}, \quad V^{\pi^*}(s) = \max_\pi V^\pi(s) \tag{4}$$

この条件を満たす π^* を最適方策と呼び，$V^* = V^{\pi^*}$ を最適状態価値関数と呼ぶ． V^* に関するベルマン方程式は以下のようになる．

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a)\left\{r(s,a,s') + V^*(s')\right\} \tag{5}$$

π^* は少なくとも一つは存在することが知られている．

状態価値と同様に，まず，状態 s において行動 a を選択したのち，方策 π に従ったときに得られる収益の期待値 $Q^\pi(s,a)$ を以下の式で定義する．

$$Q^\pi(s,a) = \mathrm{E}\left[R_1 + R_2 + \cdots + R_T | S_0 = s, A_0 = a\right] \tag{6}$$

$$= \sum_{s' \in \mathcal{S}} P\left(s'|s,a\right)\left\{r\left(s,a,s'\right) + V^\pi(s')\right\} \tag{7}$$

この $Q^{\pi}(s,a)$ を s および a の行動価値と呼ぶ． また，この式は行動価値関数 Q^{π} に関するベルマン方程式と呼ばれている．最適行動価値関数 Q^{*} を以下のように定義する．

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s), \quad Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{8}$$

次に，ベルマン方程式は以下のようになる．

$$Q^{*}(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left\{ r(s,a,s') + \max_{a' \in \mathcal{A}(s')} Q^{*}(s',a') \right\} \tag{9}$$

つまり，収益の期待値を最大化するためには π^{*} か V^{*} か Q^{*} を求める必要がある．

2.2.3 Q 学習アルゴリズム

Q^{*} を求めるための手法の一つに，Watkins が提案した Q 学習 [14] がある． Q 学習は方策オフ型 TD 法の一種である． 方策オフ型手法は，エピソードを生成するために用いられる方策と改善される方策が異なる手法である．前者を挙動方策といい，後者を推定方策という．

状態 s で挙動方策 π により行動 a を選択し，s' に遷移して報酬 r' を得たとする． このとき，以下の更新式に従って行動価値を更新する．

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \left\{ r' + \max_{a' \in \mathcal{A}(s')} Q(s',a') - Q(s,a) \right\} \tag{10}$$

ここで，$\leftarrow$ は値の更新を表し，α_t は時間ステップ t におけるステップサイズと呼ばれるパラメータで，このアルゴリズムの使用者が定める．状態 s で行動 a を選択し，状態 s' に遷移して報酬 r' を得た場合の行動価値のバックアップの様子を図 (2) に示す．

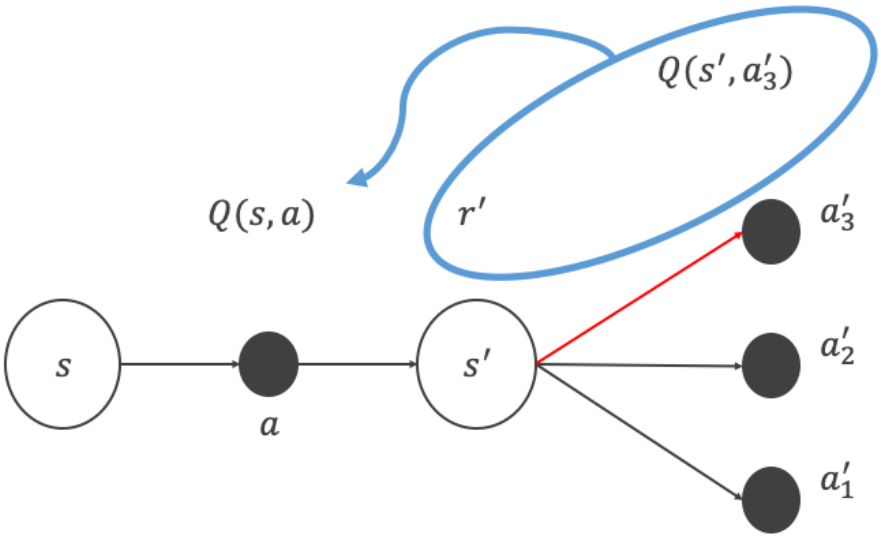

図 2　行動価値のバックアップ図

この更新式は以下のような構造になっている．

$$Q^{\text{new}} \leftarrow Q^{\text{old}} + \alpha(Q^{\text{target}} - Q^{\text{old}}) \tag{11}$$

つまり，この更新を繰り返していくことで Q^{new} は Q^{target} に漸近する．

この手法のメリット・デメリットとして，主に以下の項目が挙げられる．

メリット

- 環境の状態遷移確率 $P(s'|s,a)$ を知らなくても Q^* を推定できる．
- 式 (10) の a' がエージェントの方策に依存しないため，同じ行動価値関数に収束する．

デメリット

- 訪問されない状態や使用されない行動があると，行動価値関数が正しく収束しない．
- 環境の状態行動空間が膨大になると適用できない．

もし，アルゴリズムにおいて，挙動方策 π が

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s), \pi(s,a) > 0 \tag{12}$$

を満たし，ステップサイズ列 $\alpha_0, \alpha_1, \cdots$ が

$$\sum_{t=0}^{\infty} \alpha_t = +\infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < +\infty \tag{13}$$

を満たすならば，行動価値関数が最適行動価値関数に収束することが知られている [15].

2.2.4 関数近似版 Q 学習アルゴリズム

通常の Q 学習では，各状態行動対に対する行動価値をテーブルで保持する． しかし，状態や行動の数が膨大になると，すべての行動価値をテーブルで保持することは困難になる．このような場合，行動価値関数をパラメータ $\boldsymbol{\theta}$ で表現された関数 $\hat{Q}(\cdot|\boldsymbol{\theta})$ で近似する．

状態 s で挙動方策 π によって行動 a を選択し，状態 s' に遷移して報酬 r を得たとする． このとき，以下の更新式に従って $\boldsymbol{\theta}$ を更新する．

$$\varepsilon = r + \max_{a' \in \mathcal{A}(s')} \hat{Q}(s', a'|\boldsymbol{\theta}) - \hat{Q}(s, a|\boldsymbol{\theta}) \tag{14}$$

$$\Delta\boldsymbol{\theta} = \varepsilon \frac{\partial}{\partial \boldsymbol{\theta}} \hat{Q}(s, a|\boldsymbol{\theta}) \tag{15}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_t \Delta\boldsymbol{\theta} \tag{16}$$

ここで α_t は，時間ステップ t におけるステップサイズである． 関数近似版 Q-learning において，式 (12)，(13) に加え，次の二つの条件が成り立つとする．

1. 挙動方策 π は学習アルゴリズム実行中に変化しない．
2. $\hat{Q}(\cdot|\boldsymbol{\theta})$ は線形関数である．

このとき，行動価値関数が最適行動価値関数に収束することが知られている [16]． つまり，π が学習中に変化したり，$\hat{Q}$ に非線形関数を用いた場合は収束が保証されない．しかし，近年の研究では，$\hat{Q}$ に非線形関数を用いた場合でも行動価値を収束しやすくする経験的手法がいくつか提案されている．

2.3 深層学習

本節の内容は，文献 [17], [18] の内容を参考にし，本研究で用いる深層学習について要約し説明する．深層学習とは，多層のニューラルネットワーク (Neural Network，NN) を用いた機械学習法である． NN は人間の脳内にある神経回路網を模した数理モデルの一種である．NN にはいくつか種類があるが，本節では順伝播型 NN を扱う．

2.3.1 順伝播型 NN

順伝播型 NN は，人間の脳にあるニューロンを模したユニットが層状に複数個結合しあった構造を持ち，情報が入力側から出力側に伝播する．ユニットは複数の実数値を入力として受け取り，入力の総和を活性化関数に通した値を出力する．また，結合しているユニットとユニットの間には重みと呼ばれる実数値が割り当てられており，これをユニットの出力に掛けた値が次のユニットの入力に与えられる．ユニットの一例を図 3 に示す．

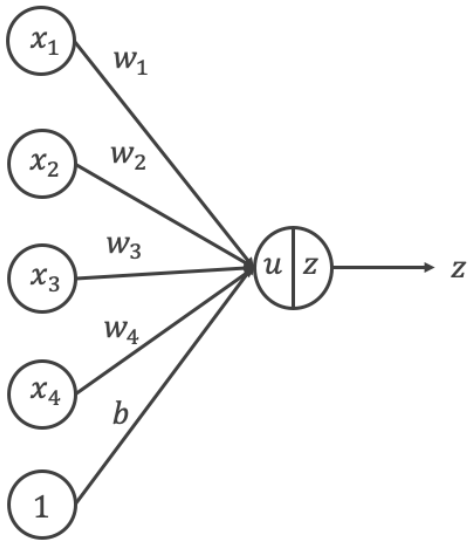

図 3　4 つの入力 $x_1, \cdots, x_4$ を受け取り値 z を出力するユニット

図 3 の場合，ユニットの出力 z は以下のようになる．

$$u = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b \tag{17}$$

$$z = f(u) \tag{18}$$

ここで b はバイアスと呼ばれる実数値，f は活性化関数を表す．図 3 のようなユニットが層状に複

数並べられ，隣接する層の間でユニット同士が結合したものが NN である．3 層の NN を図 4 に示す．L 層の NN の場合，第 l 層の i 個目のユニットの出力を $z_i^{(l)}$，第 l 層の i 個目のユニットと

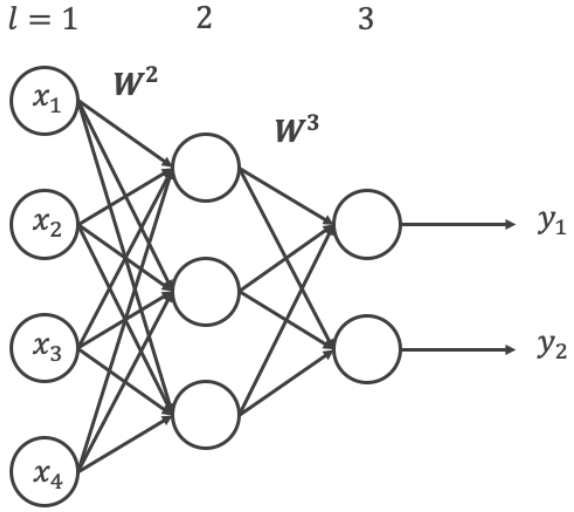

図 4　3 層 NN

第 $l+1$ 層の j 個目のユニットの間の重みを $w_{ji}^{(l+1)}$, 第 l 層の j 個目のユニットに対するバイアスを $b_j^{(l)}$，第 l 層の活性化関数を $f^{(l)}$ とすると，第 $l+1$ 層における j 個目のユニットの出力 $z_j^{(l+1)}$ は以下のようになる．

$$u_j^{(l+1)} = \sum_i z_i^{(l)} w_{ji}^{(l+1)} + b_j^{(l+1)} \tag{19}$$

$$z_j^{(l+1)} = f^{(l+1)}(u_j^{(l+1)}) \tag{20}$$

ただし，NN への入力を $\boldsymbol{x} = (x_1, \cdots, x_n)$，NN の出力を $\boldsymbol{y} = (y_1, \cdots, y_m)$ とすると，$z_j^{(1)} = x_j$，$z_j^{(l)} = y_j$ となる．式 (19), (20) をベクトルと行列を用いて表すと以下のようになる．

$$\boldsymbol{u}^{(l+1)} = \boldsymbol{W}^{(l+1)} \boldsymbol{z}^{(l)} + \boldsymbol{b}^{(l+1)} \tag{21}$$

$$\boldsymbol{z}^{(l+1)} = f^{(l+1)}(\boldsymbol{u}^{(l+1)}) \tag{22}$$

ただし，

$$\boldsymbol{u}^{(l+1)} = \begin{pmatrix} u_1^{(l+1)} \\ \vdots \\ u_J^{(l+1)} \end{pmatrix} \quad \boldsymbol{W}^{(l+1)} = \begin{pmatrix} w_{11}^{(l+1)} & \cdots & w_{1I}^{(l+1)} \\ \vdots & \ddots & \vdots \\ w_{J1}^{(l+1)} & \cdots & w_{JI}^{(l+1)} \end{pmatrix} \quad \boldsymbol{z}^{l} = \begin{pmatrix} z_1^{(l)} \\ \vdots \\ z_I^{(l)} \end{pmatrix}$$

$$\boldsymbol{z}^{(l+1)} = \begin{pmatrix} z_1^{(l+1)} \\ \vdots \\ z_J^{(l+1)} \end{pmatrix} \quad f^{(l+1)}(\boldsymbol{u}^{(l+1)}) = \begin{pmatrix} f(u_1^{(l+1)}) \\ \vdots \\ f(u_J^{(l+1)}) \end{pmatrix} \quad \boldsymbol{b}^{l} = \begin{pmatrix} b_1^{(l)} \\ \vdots \\ b_J^{(l)} \end{pmatrix}$$

である． これ以降，入力 $\boldsymbol{x}$ に対する L 層 NN の出力を，$W^{(2)}, \cdots, W^{(L)}$，$b^{(2)}, \cdots, b^{(L)}$ 全てを要素として持つベクトル $\boldsymbol{w}$ を用いて $y(\boldsymbol{x}|\boldsymbol{w})$ と表す．

2.3.2 活性化関数

一般に，活性化関数には単調増加する非線形関数がよく用いられる． 活性化関数の一例を挙げる．

- シグモイド関数

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{23}$$

- 双曲線正接関数

$$f(x) = \tanh(x) \tag{24}$$

- 正規化線形関数

$$f(x) = \max(x, 0) \tag{25}$$

2.3.3 誤差関数

NN を目的の関数に近づけるためには，訓練データを与えて NN の重みを調整する必要がある．訓練データとは，入力と入力に対する望ましい出力の組 $(\boldsymbol{x}, \boldsymbol{d})$ の集合である． すなわち，訓練データを D とすると，

$$D = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \cdots, (\boldsymbol{x}_m, \boldsymbol{d}_m)\}$$

と表される． 訓練データの任意のサンプル $(\boldsymbol{x}_i, \boldsymbol{d}_i)$ に対し，$\boldsymbol{x}_i$ に対する NN の出力 $y(\boldsymbol{x}_i|\boldsymbol{w})$ と $\boldsymbol{d}_i$ を近づけるように重みを調整することを学習と呼ぶ．このとき，$\boldsymbol{d}_i$ と $y(\boldsymbol{x}_i|\boldsymbol{w})$ の近さをどのように定義するかが重要になる． この近さを定義する関数を誤差関数と呼ぶ．誤差関数は，目的に応じて二乗誤差や交差エントロピー誤差が用いられる． 本項では二乗誤差について説明する．

二乗誤差は，訓練データをよく再現するような関数を定める (回帰) 際に用いられる． サンプル $(\boldsymbol{x}_n, \boldsymbol{d}_n)$ 1 つに対する二乗誤差を $E_n(\boldsymbol{w}) = (1/2) \|\boldsymbol{d}_n - y(\boldsymbol{x}_n|\boldsymbol{w})\|^2$ とおくと，訓練データに対する二乗誤差 $E(\boldsymbol{w})$ は以下の通り．

$$E(\boldsymbol{w}) = \sum_{n=1}^{|D|} E_n(\boldsymbol{w}) \tag{26}$$

この $E(\boldsymbol{w})$ が最も小さくなるような $\boldsymbol{w}$ を選択する．

2.3.4 確率的勾配降下法

学習の目的は，誤差関数 $E(\boldsymbol{w})$ を最小にする $\boldsymbol{w}$ を求めることである． しかし，$E(\boldsymbol{w})$ は一般には凸関数ではなく，大域的な最小解を求めることは困難である．そこで，大域的最小解を求める代わりに局所的な極小点を見つけることを考える．

極小点を求める最も簡単な方法として勾配降下法がある． この手法における勾配とは，誤差関数の勾配 ∇E を指す． ∇E は以下のように表される．

$$\nabla E \equiv \frac{\partial E}{\partial \boldsymbol{w}} = \left(\frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_M} \right)^T \tag{27}$$

勾配降下法は，現在の $\boldsymbol{w}$ を負の勾配方向 $-\nabla E$ に少し動かすことを何度も繰り返す． 現在の重みを $\boldsymbol{w}^t$，動かした後の重みを $\boldsymbol{w}^{t+1}$ とすると，更新式は以下のようになる．

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \epsilon \nabla E \tag{28}$$

ここで ϵ は $\boldsymbol{w}$ の更新量の大きさを定める定数で学習係数と呼ばれる実数値である．初期値 $\boldsymbol{w}^{(1)}$ を適当に決め，式 (28) に従って重みを更新することで極小点に到達することができる．しかし，$E(\boldsymbol{w})$ の形状や ϵ の大きさによっては，重みを更新しても $E(\boldsymbol{w})$ の値が増加することもあるので，適切な ϵ を定めることが重要である．

勾配降下法では，訓練データのサンプルすべてを用いて勾配を計算していた． このような勾配の更新手法をバッチ学習と呼ぶ．これに対し，訓練データの一部を用いて $\boldsymbol{w}$ を更新する手法を確率的勾配降下法 (Stochastic Gradient Descent，SGD) と呼ぶ．SGD では，訓練データの一部のサンプルをひとまとめにし，この単位で重みを更新する． ひとまとめにしたサンプルの集合をミニバッチと呼ぶ．t 回目の重みの更新におけるミニバッチを D_t とすると，誤差関数 $E_t(\boldsymbol{w})$ は以下のように表される．

$$E_t(\boldsymbol{w}) = \frac{1}{|D_t|} \sum_{n \in D_t} E_n(\boldsymbol{w}) \tag{29}$$

SGD のメリットとして，局所的な極小解に陥るリスクを軽減したり, オンラインでの学習，すなわち訓練データの収集と勾配の更新を並行して行えることが挙げられる．

SGD は，ミニバッチのサイズを大きくすると学習速度が向上するが，SGD の恩恵を受けられなくなる．逆にミニバッチのサイズを小さくすると，SGD の恩恵を受けられる代わりに学習に時間がかかる．そのため，勾配法の学習速度を向上させる手法がいくつか提案されており，代表的な手法にモメンタムがある．モメンタムは，重みの修正量に前回の重みの修正量を定数倍して加算する．ミニバッチ $t-1$ に対する重みの修正量を $\Delta \boldsymbol{w}^{(t-1)} \equiv \boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}$ とすると，t 回目のミニバッチに対する更新は以下のようになる．

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \epsilon \nabla E_t(\boldsymbol{w}^{(\boldsymbol{t})}) + \mu \Delta \boldsymbol{w}^{(t-1)} \tag{30}$$

ここで μ は加算の割合を制御するパラメータである．勾配降下法は，誤差関数が深い谷状の形状を持ち，かつその谷底にあまり高低差がないときに非常に効率が悪くなることが知られているが，モメンタムはこの問題を解決し，勾配の収束を早める効果がある．

2.3.5 誤差逆伝播法

誤差逆伝播法は, ユニット間の重みとバイアスに関する勾配を計算する方法である．これ以降，表記を簡略化するために，$+1$ を常に出力する第 0 番ユニットを各層に導入し，バイアス $b_j^{(l)}$ を，

第 $l-1$ 層の 0 番目のユニットと第 l 層の j 番目のユニットの間の重み $w_{j0}^{(l)}$ として考えることとする．すなわち，第 l 層の j 番目のユニットの出力 $u_j^{(l)}$ は以下のようになる．

$$u_j^{(l)} = \sum_{i=1}^{I} w_{ji}^{(l)} z_i^{(l-1)} + b_j^{(l)} = \sum_{i=0}^{I} w_{ji}^{(l)} z_i^{(l)} \tag{31}$$

サンプル一つに対する誤差 $E_n(\boldsymbol{w})$ の $w_{ji}^{(l)}$ に関する勾配は次のように書ける．

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial u_j^{(l)}} \frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{32}$$

式 (32) の第 1 項について考える． $u_j^{(l)}$ は，ユニット j からの出力 $z_j^{(l)}$ を通して，第 $l+1$ 層の各ユニット k に対する入力の総和 $u_k^{(l+1)}$ を変化させることによってのみ，E_n に影響を与える．したがって，各 $u_k^{(l+1)}$ を経由した微分連鎖により

$$\frac{\partial E_n}{\partial u_j^{(l)}} = \sum_k \frac{\partial E_n}{\partial u_k^{(l+1)}} \frac{\partial u_k^{(l+1)}}{\partial u_j^{(l)}} \tag{33}$$

と書ける． ここで，第 l 層と第 $l+1$ 層の入力に関する微分 $\partial E_n / \partial u_j^{(l)}$ を

$$\delta_j^l \equiv \frac{\partial E_n}{\partial u_j^{(l)}} \tag{34}$$

とおく． この量をデルタと呼ぶ． $u_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} z_j^{(l)} = \sum_j w_{kj}^{(l+1)} f\left(u_j^{(l)}\right)$ より，$\partial u_k^{(l+1)} / \partial u_j^{(l)} = w_{kj}^{(l+1)} f'\left(u_j^{(l)}\right)$ となることから，式 (33) は

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \left(w_{kj}^{(l+1)} f'\left(u_j^{(l)}\right) \right) \tag{35}$$

と書ける． つまり，第 l 層の各ユニットにおけるデルタは，第 $l+1$ 層のデルタを用いて計算できる (図 5)． 式 (32) の第 2 項は $u_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)}$ より，

$$\frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}} = z_i^{(l-1)} \tag{36}$$

となる． したがって，式 (32) は式 (35), (36) より

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)} \tag{37}$$

となる．

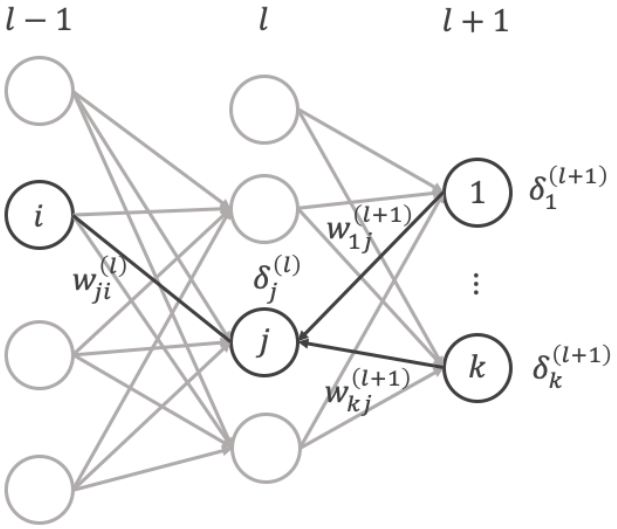

図 5　デルタの逆伝播の様子

出力層におけるデルタの計算を考える． 誤差関数には二乗誤差を，活性化関数に恒等関数を用いる場合，出力層のデルタ $\delta_j^{(L)}$ は次のように計算できる．

$$\delta_j^L = \frac{\partial E_n}{\partial u_j^{(L)}} \tag{38}$$

$$= \frac{\partial}{\partial u_j^{(L)}} \left(\frac{1}{2} \left\| \boldsymbol{y}(\boldsymbol{x}|\boldsymbol{w}) - \boldsymbol{d} \right\|^2 \right) \tag{39}$$

$$= \frac{\partial}{\partial u_j^{(L)}} \left(\frac{1}{2} \left\| \boldsymbol{u}^{(L)} - \boldsymbol{d} \right\|^2 \right) \tag{40}$$

$$= \left(u_j^{(L)} - d_j \right) \tag{41}$$

これで，各層における勾配の更新を行うことが可能になった． サンプル $(\boldsymbol{x}, \boldsymbol{d})$ が与えられたときの誤差逆伝播法のアルゴリズムを以下に示す．

誤差逆伝播法のアルゴリズム

1. $\boldsymbol{z}^{(1)} = \boldsymbol{x}$ とし，第 2 層から第 L 層まで順番に，$\boldsymbol{u}^{(l)}$ および $\boldsymbol{z}^{(l)}$ を求める．
2. 出力層の各デルタ $\delta_j^{(L)}$ を計算する．
3. 第 $L-1$ 層から第 2 層まで順番に，式 (35) にしたがって各デルタ $\delta_j^{(l)}$ を計算する．
4. 式 (37) にしたがって，各層のパラメータ $w_{ji}^{(l)} (l = 2, \cdots, L)$ の勾配を計算する．

2.4　確定的な展開形ゲーム

本節の内容は文献 [19] の内容を参考にし，本研究で用いる展開形ゲームについて要約し説明する．展開形ゲームとは，プレイヤの意思決定過程を木で表現するゲームの表現形式の一種であり，

ゲーム木 K，プレイヤ分割 P，情報分割 $\mathcal{I}$，利得関数 u の 4 つの要素で定義される．一般的な展開形ゲームは偶然的な事象を考慮しているが，本研究ではそのような事象を考慮しない確定的な展開形ゲームを用いる．

ゲーム木 K

ゲーム木 K は点と枝で構成され，初期点 0_1 を持つ． 0_1 と点 x を結ぶ点と枝の列を点 x へのパスという．二つの点 x, y に対し，y がパス x 上にあるとき，x は y の後にある，または y は x の前にあるといい，$x > y$ と表記する．$x > y$ であり，x と y が一本の枝 e で結ばれているとき，x を y の直後の点，または y は x の直前の点という． また，y を e の始点といい，x を e の終点という．点 w が $x > w$ となる点 x を持たないとき，w を木の頂点という． 頂点すべてからなる集合を W と表す．頂点以外の点を手番という． 手番すべてからなる集合を X と表す． 手番 x に対し，x と x の直後の点を結ぶ 1 本の枝を，手番 x における選択肢といい，x における選択肢すべてからなる集合を $A(x)$ と表す．

プレイヤ分割 P

プレイヤの集合を $N = \{1, \cdots, n\}$ とする． プレイヤ分割 $P = [P_1, \cdots, P_n]$ は，K の手番の集合 X の一つの分割である．P_i $(i = 1, \cdots, n)$ はプレイヤ i の手番すべてからなる集合である．偶然手番とは，プレイヤの意思決定とは無関係に枝が選択される手番のことである．

情報分割 $\mathcal{I}$

情報分割 $\mathcal{I} = [\mathcal{I}_1, \cdots, \mathcal{I}_n]$ はプレイヤ分割 $P = [P_1, \cdots, P_n]$ の一つの細分割である．$\mathcal{I}_i$ $(i = 1, \cdots, n)$ をプレイヤ i の情報分割といい，$I \in \mathcal{I}_i$ をプレイヤ i の情報集合という．プレイヤ i は I に属す要素を区別できない．

利得関数 u

利得関数 u は任意の頂点 $w \in W$ に対して，利得ベクトル $u(w) = (u_1(w), \cdots, u_n(w))$ を対応させる関数である．ただし，$u(w)$ の第 i 成分 $u_i(w)$ はプレイヤ i の利得を表す．

ゲーム木の一例を図 6 に示す． 図 6 の木が表すゲームは 2 人でプレイするゲームである．各プレイヤは $A = \{a_1, a_2\}$ という行動集合を持つ． 0_1 では自然が確率的に c_1 か c_2 のどちらかを選択する． $0_2, 0_3 \in P_1$ ではプレイヤ 1 が，$0_4, \cdots, 0_7 \in P_2$ ではプレイヤ 2 が意思決定を行う．

2.4.1 戦略と期待利得

展開形ゲームにおいて，プレイヤはゲームを行う前に，各情報集合でどのような行動を選択するかを計画することができる．このようなプレイヤの行動計画を戦略と呼ぶ．本項ではまず，戦略の概念として代表的な純戦略，混合戦略，行動戦略について説明する．

$\Gamma = (K, P, \mathcal{I}, u)$ を展開形ゲームとする． プレイヤ i の純戦略 s_i とは，プレイヤ i の任意の情報集合 $I \in \mathcal{I}_i$ に対して，I における一つの選択肢 $s_i(I) \in A(I)$ を対応させる関数である．プレイヤ i の純戦略の集合を S_i と表す．プレイヤ i の混合戦略 q_i とは，純戦略の集合 S_i 上の一つの確率分布である．プレイヤ i の混合戦略の集合を Q_i と表す．プレイヤ i の行動戦略 b_i とは，プレイヤ i

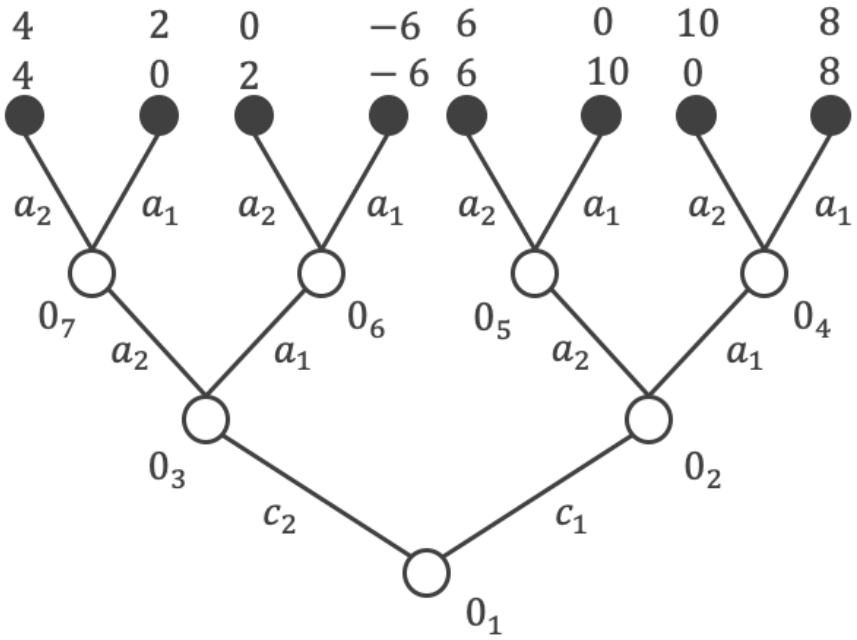

図 6　ゲーム木の例

の任意の情報集合 $I \in \mathcal{I}_i$ に対して，I における一つの局所戦略 b_{iI} を対応させる関数である．ここで，情報集合 $I \in \mathcal{I}_i$ におけるプレイヤ i の局所戦略 b_{iI} とは，I における選択肢の集合 $A(I)$ 上の一つの確率分布である．プレイヤ i の行動戦略の集合を B_i と表す．プレイヤ i の純戦略は，任意の情報集合 I に対し，一つの選択肢に確率 1 を付与する行動戦略とみなせる．

次に，プレイヤ i の期待利得について説明する． $b = (b_1, \cdots, b_n)$ をプレイヤすべての行動戦略の組，プレイヤ i が手番 x において枝 $e \in A(x)$ を選択する確率を $b_{ix}(e)$，ゲーム木の点 $x \in X$ に対して初期点 0_1 から x へのパス上における枝と枝の始点の組すべてからなる集合を $E(x)$ とする．$E(x)$ は，プレイヤ i $(= 1, \cdots, n)$ の枝と枝の始点の集合 $E_i(x)$ すべての和集合である．b に従ってゲームが行われるとき, x に到達する確率 $\pi^b(x)$ は以下の式で表される．

$$\pi^b(x) = \prod_{i=1}^{n} \prod_{(e,v) \in E_i(x)} b_{iv}(e) \tag{42}$$

この $\pi^b(x)$ を，行動戦略の組 b の下での点 x の実現確率という． $\pi^b(x) > 0$ のとき，x は b の下で到達可能であるという．$\pi^b(x)$ を用いて，プレイヤ i の期待利得は以下の式で表される．

$$U_i(b) = \sum_{w \in W} \pi^b(w) u_i(w) \tag{43}$$

最後に，最適応答について説明する． プレイヤ i の行動戦略 $b_i \in B_i$ が，他のプレイヤすべての戦略の組 $b_{-i} = (b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_n)$ に対する最適応答であるとは，

$$U_i\left(b_i, b_{-i}\right) = \max_{b' \in B_i} U_i\left(b', b_{-i}\right) \tag{44}$$

が成立することである．

2.4.2 ϵ 均衡点

Γ において，行動戦略の組 $b^* = (b_1^*, \cdots, b_n^*)$ が ϵ 均衡点であるとは，任意のプレイヤ $i \in N$ に対して，

$$U_i(b^*) + \epsilon \geq \max_{b_i \in B_i} U_i(b_i, b_{-i}^*) \tag{45}$$

が成立することである． ただし b_{-i}^* は，b^* から b_i^* を除いた行動戦略の組を表す． $\epsilon = 0$ のとき，ϵ 均衡点を単に均衡点と呼ぶ．

2.4.3 完全記憶ゲーム

Γ が完全記憶ゲームであるとは，任意のプレイヤ i の任意の異なる 2 つの情報集合 $I_1, I_2 \in \mathcal{I}_i$ に対して，I_2 のある手番 y が I_1 から枝 c によって到達可能ならば，I_2 の任意の手番は枝 c によって到達可能であることである．この定義は，あるプレイヤ i が手番 x で意思決定を行うときに，初期点から x までのパスに属すプレイヤ i の任意の手番 y に対し，

1. y での選択
2. y で知っていた全ての情報

を記憶していることを表す． 完全記憶ゲームの例を図 7 に，完全記憶ゲームでない例を図 8 に示す． このゲームでは，$P_1 = \{0_1, 0_4, \cdots, 0_7\}$，$P_2 = \{0_2, 0_3\}$ である．図 7 では，プレイヤ 1 が行動 a_1 を選択することによって I_2^1 の任意の点へ，a_2 を選択することによって I_3^1 の任意の点へ到達することができる．しかし図 8 の場合は，a_1 を選択した場合は $0_6, 0_7$ に，a_2 を選択した場合は $0_4, 0_5$ に到達することができない．以降，完全記憶ゲームのことを不完全情報ゲームと呼ぶ．

2.5 CFR 法

CFR 法とは，Zinkevich らによって提案された二人ゼロ和不完全情報ゲームの ε 均衡点を求める手法である [5]． ゼロ和ゲームとは，任意の行動戦略の組 $b = (b_1, \cdots, b_n)$ に対して，プレイヤすべての期待利得の総和 $\sum_{i \in N} U_i(b)$ が 0 となるゲームのことである．CFR 法を説明するにあたって，$\pi_i^b(x), \pi_{-i}^b(x), \pi^b(x, y), \pi^b(I), \pi_i^b(I), \pi_{-i}^b(I)$ は以下を表すこととする．

$\pi_i^b(x)$： $\pi^b(x)$ にプレイヤ i が寄与する確率 (すなわち $\pi_i^b(x) = \prod_{(e,v) \in E_i(x)} b_{iv}(e)$)
$\pi_{-i}^b(x)$： プレイヤ i を除くすべてのプレイヤが $\pi^b(x)$ に寄与する確率
$\pi^b(x, y)$： b の下で点 x から点 y へ到達する確率
$\pi^b(I)$： 情報集合 I に到達する確率 (すなわち $\pi^b(I) = \sum_{x \in I} \pi^b(x)$)
$\pi_i^b(I)$： $\pi^b(I)$ にプレイヤ i が寄与する確率
$\pi_{-i}^b(I)$： プレイヤ i を除くすべてのプレイヤが $\pi^b(I)$ に寄与する確率

本節では，CFR 法を行う上で必要な概念を紹介した後，アルゴリズムの擬似コードを示す．

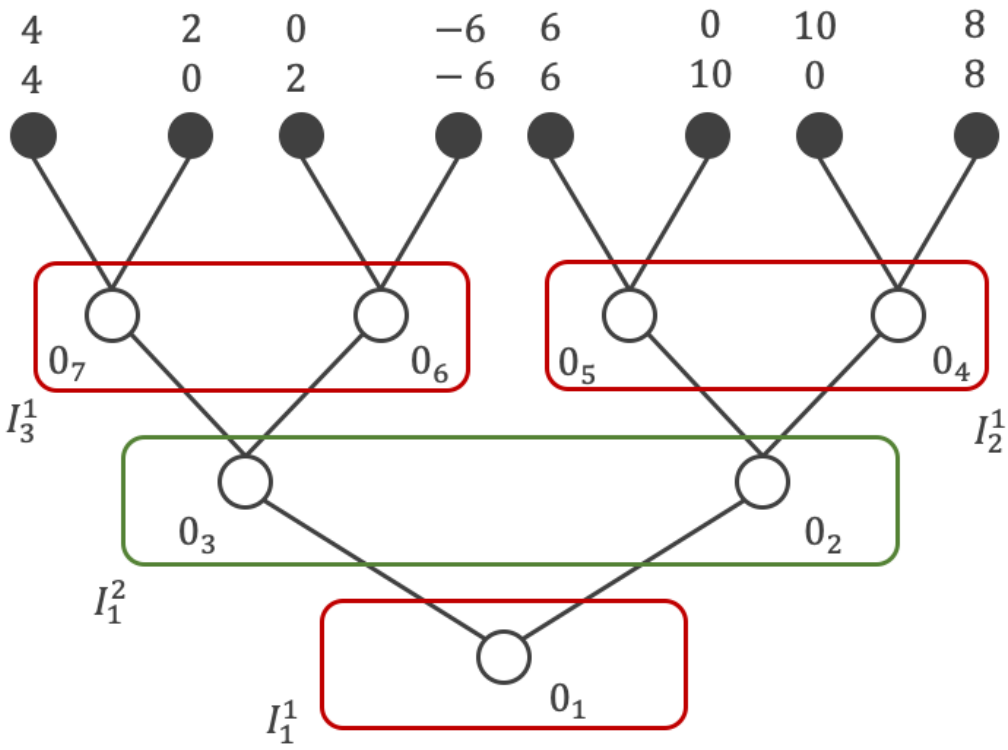

図 7　完全記憶ゲームの例

2.5.1　average overall regret と average strategy

ゲームを T 回行ったとする． t 回目のゲームにおけるプレイヤ i の行動戦略を b_i^t とおく．T 回目のゲーム後におけるプレイヤ i の average overall regret $R_{i,\mathrm{ave}}^T$ を次の式で定義する．

$$R_{i,\mathrm{ave}}^T = \frac{1}{T}\max_{b\in B_i}\sum_{t=1}^{T}\left\{U_i(b, b_{-i}^t) - U_i(b^t)\right\} \tag{46}$$

さらに T 回目のゲーム後において，プレイヤ i の任意の情報集合 $I \in \mathcal{I}_i$ の任意の行動 $a \in A(I)$ に対する average strategy $\bar{b}_i^T(I)(a)$ を次の式で定義する．

$$\bar{b}_i^T(I)(a) = \frac{\sum_{t=1}^{T}\pi_i^{b^t}(I)b^t(I)(a)}{\sum_{t=1}^{T}\pi_i^{b^t}(I)} \tag{47}$$

ここで次の定理を紹介する．

定理 2.1. 二人ゼロ和不完全情報ゲームの T 回目のゲーム後において，任意のプレイヤに対し，$R_{i,\mathrm{ave}}^T$ が ϵ より小さいならば，$\bar{b}^T = \left(\bar{b}_1^T, \bar{b}_2^T\right)$ は 2ϵ 均衡点である．

もし ϵ 均衡点を求めたいならば，T 回目のゲーム後の average overall regret が $\epsilon/2$ より小さくなるような $\bar{b}^T$ を求めればよい．任意のプレイヤ i に対して $R_{i,\mathrm{ave}}^T = 0$ のとき，$\bar{b}^T$ は均衡点となる．

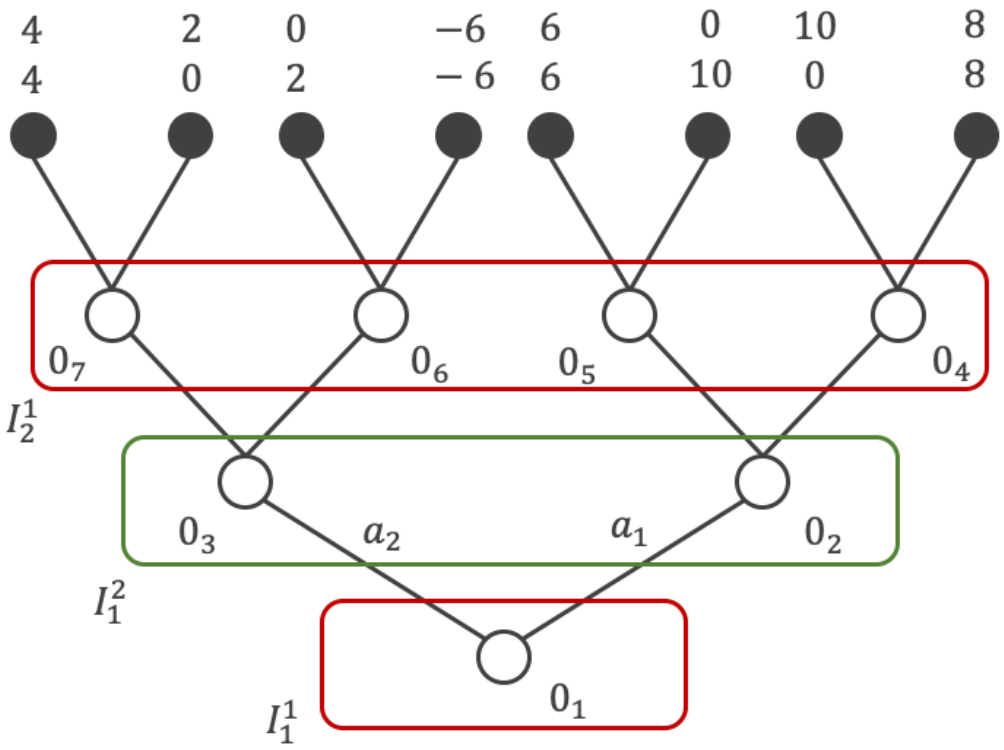

図 8　完全記憶ゲームでない例

2.5.2　counterfactual utility と immediate counterfactual regret

本項では，すべての $i \in N$ で $R^T_{i,\mathrm{ave}}$ よりも $R^{T+1}_{i,\mathrm{ave}}$ が小さくなるような b^{T+1} を，$b^1, \cdots, b^T$ から構成する方法を紹介する．まず，行動戦略の組 b に従ったときの，手番 x におけるプレイヤ i の期待利得を $u_i^b(x)$ とおく．b に従ったときの，情報集合 $I \in \mathcal{I}_i$ におけるプレイヤ i の counterfactual utility $u_i^b(I)$ を以下のように定義する．

$$u_i^b(I) = \frac{\sum_{x \in I, w \in W} \pi^b_{-i}(x) \pi^b(x,w) u_i(w)}{\pi^b_{-i}(I)} \tag{48}$$

この式は，プレイヤ i が情報集合 I を経由しようとした場合の，I における期待利得を表している．

次に，$R_i^T(I,a), R_i^{T,+}(I,a)$ を以下のように定義する．

$$R_i^T(I,a) = \frac{1}{T} \sum_{t=1}^{T} \pi^{b^t}_{-i}(I) \left(u_i^{b^t|_{I \to a}}(I) - u_i^{b^t}(I) \right) \tag{49}$$

$$R_i^{T,+}(I,a) = \max \left(R_i^{T,+}(I,a), 0 \right) \tag{50}$$

$R^{T,+}(I,a)$ を用いて，$T+1$ 回目のゲームにおける行動戦略 $b_i^{T+1}(I)(a)$ を以下のように定義する．

$$b_i^{T+1}(I)(a) = \begin{cases} \dfrac{R_i^{T,+}(I,a)}{\sum_{a' \in A(I)} R_i^{T,+}(I,a')} & \text{if } \sum_{a' \in A(I)} R_i^{T,+}(I,a') > 0 \\ \dfrac{1}{|A(I)|} & \text{otherwise.} \end{cases} \tag{51}$$

ただし，$R_i^0(I,a) = 0$ とする．ここで，次の定理を紹介する．

定理 2.2. A_i をプレイヤ i の行動すべてからなる集合，$\Delta_{u,i} = \max_{w \in W} u_i(w) - \min_{w \in W} u_i(w)$ とする．もしプレイヤ i が (51) に従って行動を決定するならば，以下の条件が成立する．

$$|A_i| = \max_{x \in P_i} |A(x)|,\ R_{i,\text{ave}}^T \leq \frac{\Delta_{u,i}|\mathcal{I}_i|\sqrt{|A_i|}}{\sqrt{T}} \tag{52}$$

この定理より，式 (51) に従って戦略を決定するプレイヤを繰り返し対戦させることで $R_{i,\text{ave}}^T$ を小さくすることができ，ϵ 均衡点となる $\bar{b}^T$ を求めることができる．CFR アルゴリズムの擬似コードを Algorithm 1 に示す．

Algorithm 1 CFR アルゴリズム

1: $\forall i \in N, \forall I \in \mathcal{I}_i, \forall a \in A(I), R_i^0(I,a) \leftarrow 0$
2: **for** $t = 1$ to T **do**
3: 　$\forall i \in N, b_i^t \leftarrow$ (R_i^{t-1} を元にプレイヤすべての行動戦略を式 (51) を用いて計算)
4: 　b^t の下での各 $R_i^t(I)(a)$ を木の頂点から初期点の方向に順番に計算
5: **end for**
6: $b^1 \sim b^T$ を元に $\bar{b}^T$ を計算
7: **return** $\bar{b}^T$

3　先行研究

本章では Q 学習をゲームに適用した先行研究と CFR をゲームに適用した先行研究と Fighting-ICE の AI に関する先行研究を紹介する．

3.1　Q 学習に関する先行研究

本節では Q 学習をゲームに適用した先行研究を紹介する．

3.1.1　Mario AI

Liao らは，Q 学習を用いてマリオ AI を開発した [20]． Q 学習を行うにあたり，Liao らはゲーム状態の離散化を行うことで，行動価値をテーブルで扱えるようにしている．Q 学習を適用した結果，およそ 90% の割合でゲームをクリアすることができるようになった．

Liao らはゲーム状態を，AI プレイヤの状況や，敵や障害物の位置など 9 つの要素の組で定義した．これにより，ゲームの状態数を 2^{39} 程度まで削減した． さらに，この状態集合のうち実現可能な状態を剪定し，状態数を 20000 まで削減した．

学習では，挙動方策に ε-グリーディ方策を，推定方策をグリーディ方策とする Q 学習を行う．また，Q 学習における学習率などのパラメータには様々な値を使用し，AI の性能を比較している．

学習の結果，最も良い性能を持つ AI は約 90% の割合でゲームをクリアするようになった．また，ゲームクリアまでの時間や敵を倒す割合に関しても，性能の向上が確認された．

この研究で用いられたスーパーマリオは 1 人でプレイするビデオゲームであり，エピソードの長さには上限がない． また，行動集合の大きさは 10 程度である．これに対し，FightingICE は 2 人不完全情報ゲームであり，エピソードの長さは 10^3 程度，行動集合の大きさは 10 程度である．マリオと FightingICE の共通点は，ビデオゲームであること，行動集合の大きさが同程度であることの二つである．

3.1.2　Deep Q-Network

Deep Q-Network (DQN) は，Mnih らが提案した Q 学習の改良手法である [8]．これを Atari2600 に適用し，49 個あるゲームのうち 29 個で人間と同等かそれ以上の性能を持つ AI を開発した．

Mnih らは Atari2600 の膨大な状態行動集合を持つゲームに対応するため，行動価値関数を畳み込み層 3 層，全結合層 2 層からなる NN で近似している．この NN の入力には，直近 4 フレームのゲーム画像を 84×84 のグレースケール画像に変換したものを与える． NN の出力は各行動に対する推定行動価値である．

学習では，挙動方策に ε-グリーディ方策を，推定方策をグリーディ方策とする Q 学習を行う．しかし，行動価値関数を非線形関数で近似した場合は，行動価値の収束が不安定であることが知ら

れている．そのため Mnih らは，行動価値の収束を安定させるために experience replay と target network という二つの手法を導入した．

experience replay とは，ゲームプレイ中に観測した状態遷移のサンプル $(S_t, A_t, S_{t+1}, R_{t+1})$ をデータ集合（リプレイバッファ）に保存しておき，価値を更新する際にデータ集合の中からランダムにサンプルを選択し使用する． これを行うことで，データの偏りを無くし効率的に学習ができることが知られている．

target network は，行動価値の更新時に指標とする値に過去の行動価値を使用する． 指標とする価値は一定反復回数ごとに更新する．一般的な Q 学習では，行動価値の更新の度に指標とする行動価値が変化するため，学習が不安定になることがしばしばある．そのため，指標とする行動価値を固定することで，学習の安定化を行う．

学習の結果，Mnih らが開発した AI は Atari2600 の 49 あるゲームのうち，43 のゲームで既存の強化学習法を上回り，29 のゲームで人間のトッププレイヤと同等かそれ以上の性能となった．

Atari2600 は 1 人でプレイするゲームであり，行動集合の大きさは 10 程度，エピソードの長さは 10^3 程度である．一方で FightingICE は 3.1.1 で述べた性質を持つ．Atari2600 と FightingICE の共通点は，ビデオゲームであること，エピソードの長さと行動集合の大きさが同程度であることの二つである．したがって，DQN は FightingICE に有効ではないかと考えられる．

3.2 CFR 法に関する先行研究

本節では CFR 法をゲームに適用した先行研究を紹介する．

3.2.1 Deep Stack

Deep Stack は，Moravčík らが開発した HUNL をプレイする AI である．HUNL はポーカーの一種であり掛け金の制限を持たない． そのため，ゲームの意思決定点は 10^{160} を超えると言われている． Deep Stack は深層学習とゲーム木の深さを制限した探索により戦略を計算する AI であり，初めて HUNL で人間のプロプレイヤに有意に勝ち越した．

Deep Stack は戦略を計算するために，現在のゲーム状況を根に対応させたゲーム木を探索する．しかし，ゲーム木は非常に大きいため，掛け金を決定する行動を 3 つに制限し，さらに木の深さを制限して探索を行う．一定の深さに到達した場合は，その節点における評価値を NN で近似的に計算する．

NN は，ポットサイズ (ゲーム開始時のプレイヤすべての所持金額の合計でベットされた金額を割った値)，公開カード，各プレイヤが所持する非公開カードの確率分布の推定値を入力とし，各プレイヤの各非公開カードに対する推定価値を出力する．ネットワークは，ユニット数 500 の全結合層を 7 層重ねた構成になっており，活性化関数を持つユニットには PReLU [21] が用いられている．また，Deep Stack は公開カードの枚数が 3 枚の場合のネットワーク (flop network) と 4 枚の場合のネットワーク (turn network) の 2 種類のネットワークを使用している．公開カードが 5 枚の場合はネットワークを用いずに，抽象化したゲームを解くことで行動を決定している．

Flop network は，ランダムに生成された 100 万のゲーム状況から学習を行う． 出力の教師には，ゲーム状況を根に対応させたゲーム木に対し，CFR を改良した手法の一つである CFR+ を用いて計算した値を用いる．Turn network も同様に学習を行うが，flop network と違って，1000 万のゲーム状況を用いる．

Deep Stack は，33 人のプロプレイヤと合計 44852 ゲーム行い 486mbb/g [*1]勝ち越した．また，33 人のうち 10 人のプレイヤに対し有意に勝ち越した．

HUNL は 2 人不確定不完全情報ゲームであり，状態集合の大きさは 10^{160}，エピソードの長さは 10 未満，行動集合の大きさは所有している金額が多いほど大きくなる．ただしこの研究では，行動を抽象化することによって行動集合の大きさを 10 未満に抑えている．HUNL と FightingICE の共通点は，2 人不完全情報ゲームであること，行動集合の大きさが同程度であることの 2 つである．したがって，この研究において HUNL に適用された CFR+ の元となる CFR 法は格闘ゲームに対して有効ではないかと考えられる．

3.2.2 Libratus

Libratus は，Brown らが開発した HUNL をプレイする AI である． Libratus は CFR の発展形の一つである CFR+ と Monte Carlo CFR (MCCFR) を用いて開発され，人間のプロプレイヤのトップ 4 人と対戦し有意に勝ち越した．

Libratus はまず，掛け金を決定する行動や第 3,4 ベットラウンドにおいて構成できる役を抽象化する． これによりゲームの意思決定点を 10^{160} から 10^{12} 程度に削減することができる．このゲーム木に対し MCCFR を適用し戦略を計算する． この戦略はゲーム前に事前に計算しておき，第 1,2 ベットラウンドで使用する．

次に，第 3 ベットラウンドに到達した場合は，MCCFR で計算した戦略と，実際に相手プレイヤが取った行動を元に，現在のゲーム状況を根に対応させたゲーム木を構成し CFR+ で戦略を計算する． つまり，Libratus は対戦を行いながら動的に戦略を計算する．このとき，役に関する抽象化は行わず，掛け金に対する行動の抽象度は低くする． ここで計算した戦略は第 3,4 ベットラウンドで使用する．

最後に，Libratus はゲームをプレイしながら，MCCFR で計算された戦略を改善する． 改善前の戦略は行動や役を抽象化したゲームに対して計算されたもので，実際に相手プレイヤがとる行動は，抽象化された行動の中で一番近い行動として扱われる． しかし，実際にとった行動と抽象化された行動の間に無視できないほどの差異がある場合がある．このとき，Libratus はあらかじめ抽象化していたゲーム木に相手の取った行動を追加し，その行動に対する戦略を計算する．

Libratus はプロプレイヤのうち上位 4 人と対戦し 147mbb/g 勝ち越した．これは統計的に有意な結果であり，Libratus はポーカーで初めて人間のトッププレイヤを超えた AI となった．

HUNL は 3.2.1 で述べた性質を持つ． ただしこの研究では，ゲーム状況と行動を抽象化するこ

[*1] mbb/g とは，milli big blinds per game の略で，全試合の利得の総和をビッグブラインド (ゲームへの参加料) の 1000 分の 1 の値で割った後，全試合数で割った値のことである．

とにより状態集合の大きさを 10^{12} に，行動集合の大きさを約 10 分の 1 から 100 分の 1 程度に抑えている．格闘ゲームも膨大な状態集合を持つため，ゲーム状況の抽象化は有効ではないかと考えられる．

3.3 FightingICE の AI に関する先行研究

ICE は, モンテカルロ木探索で行動を決定する MctsAi を公開した [10]．MctsAi では，自キャラクタの行動列は木で表現されており，相手キャラクタの行動列はあらかじめ定められた確率分布に従って決定されるものとする．木の枝には可能な行動一つが対応し，点は初期点からの行動列から定まる．また，行動列の長さが 5 の点を頂点とする．

探索時に頂点まで到達した場合は試行を行う．試行とは，初期点から定まる自キャラクタの行動列と，あらかじめ定められた確率分布に従って決定された相手キャラクタの行動列をもとに，初期点に対応するゲーム状況から 60 フレーム分ラウンドを進行させることである．試行の評価値は，試行前と後における各キャラクタの HP から計算する．試行を一定時間繰返した後，初期点で最も実行された行動を選択する．

4　研究目的

本研究の目的は，格闘ゲームに CFR 法と強化学習法を適用し，両手法の性能を比較することである． 本研究で用いる強化学習法は Q 学習法とする．CFR 法と Q 学習法のメリットとデメリットを表 1 に示す．

表 1　CFR 法と Q 学習法のメリットとデメリット

	メリット	デメリット
CFR 法	二人ゼロ和不完全情報ゲームの ε 均衡点を求めることができる	ゲーム木が大きいと適用が困難
Q 学習法	行動価値関数を関数近似することで膨大なゲーム木を持つゲームに適用可能	関数近似を行うと行動価値が収束する保証がない 不完全情報ゲームを適切に扱うことができない

CFR 法をはじめとする CFR+ や MCCFR，Deep CFR [22] は HUNL で成果を収めた手法であるが，格闘ゲームに適用された前例はない．そのため，本研究では最も単純な CFR 法を FightingICE に適用し，格闘ゲームに対して CFR 法が有効かどうかを確認する．しかし，FightingICE を含む格闘ゲームのゲーム木は非常に大きいため，CFR 法をそのまま適用することは困難である．そのため，本研究では限定的なゲーム状況のみを考える．また，FightingICE はリアルタイムでゲームが進行するため，ゲーム木の点すべてを訪問する操作を行うことは困難である．そのため，まず FightingICE の詳細なルールを調査し，その結果を元にエミュレータを実装する．

Q 学習法はビデオゲームで成果を収めた手法である． 本研究では 3.1.1, 3.1.2 で用いられていた Q 学習法を FightingICE に適用する． また，Q 学習法の性能を CFR 法と比較するために，CFR 法と同じ限定的なゲーム状況に対してプレイヤの学習を行う．また，Q 学習法を FightingICE に適用するにあたって，行動価値関数を NN で関数近似する．

FightingICE において，AI が受け取るゲーム状況は 15 フレーム前のものである．これは，人間が格闘ゲームをプレイする状況に近づけるための仕様である [23]．n フレーム前のゲーム状況を AI が受け取るゲームの設定をディレイ n と呼ぶこととする．本研究では，ディレイ 0 の場合とディレイ 15 の場合で AI のふるまいにどのような違いが生じるかを確認する．

CFR 法と Q 学習法の性能の比較は，搾取量と勝率の 2 つの指標によって行う．搾取量とは，プレイヤすべての戦略の組を評価する 1 つの指標である [24]．搾取量については 6.2 で説明する．勝率は対戦実験を行うことで計算する． 対戦実験の詳細は 6.5 に示す．

本研究で行うことを箇条書きでまとめる．

- ゲームルールの調査を行う．

- 調査結果に基づきエミュレータを実装する.
- CFR 法と Q 学習法を適用する.
- 搾取量と勝率の 2 つを用いて両手法の性能を比較する.

5　ゲームルールの調査とエミュレータの実装および実験

本章では，ゲームルールの調査方法とその結果について述べた後，エミュレータの詳細と動作検証の結果を示す．

ゲームルールの詳細を調査するにあたって，FightingICE の公式サイト [23] で公開されているルールに関する情報 (PDF の文書並びに CSV 形式の表) や GitHub で公開されている FightingICE のソースコードを参照した [25]． 調査した結果を付録 A に示す．

調査したゲームルールに基づきエミュレータを実装した． エミュレータの仕様は以下の通りである．

- FightingICE の 3 種のキャラクタのうち ZEN のみ使用可能．
- リアルタイム性を排除し，対戦マネージャは AI の意思決定が終了するまで待つ．
- ディレイやゲームの制限時間，キャラクタのゲーム開始時の HP，Energy，位置などのパラメータを変更することができる．
- ゲーム画面の描画なし．
- 対戦マネージャと 2 つの AI は Linux 上で 1 つのプロセスの中で動作する．
- 実装する言語には C++ を用いる．

実装したエミュレータの動作確認を行う． エミュレータの動作は，FightingICE の既存 AI 3 つをエミュレータで動作するように実装し，各 AI の，FightingICE での勝率とエミュレータでの勝率が同程度になるかどうかで確認する．実験に使用する既存 AI は以下の 3 つである．

- ランダムに行動を決定するランダムプレイヤ
- モンテカルロ木探索で行動を決定する MctsAi
- IF-THEN ルールで行動を決定する Machete

どの AI のペアも 200 ラウンド対戦する (左右それぞれ 100 ラウンド)． ゲームパラメータは，Fighting Game AI Competition の Standard League のルールに従い設定された．また，勝率を計算するにあたり，勝ちを 1，負けを 0，引分けを 0.5 とする．95% 信頼区間は標準誤差から見積もった．

ランダムプレイヤは，FightingICE とエミュレータ両方において，他 AI 2 つに殆ど勝てなかった．また，Machete の MctsAi に対する勝率は 0.86 ± 0.10 (FightingICE) と 0.77 ± 0.11 (エミュレータ) であり，Machete が MctsAi に勝ちやすいという傾向はエミュレータ上でも再現されていた．ただし，FightingICE においては，MctsAi の意思決定はフレーム落ちによりしばしば無視されてしまうため，Machete の勝率が多少高くなるようである．さらに，思考時間が短いランダムプレイヤと Machete の対戦実験の計算は，エミュレータの方が約 16 倍速かった．

6　CFR 法と Q 学習法の実装および実験

本章では，CFR 法を用いた実験と Q 学習を用いた実験の共通設定を述べたあと，両実験の詳細と結果を示す． また，両実験によって得られた結果を搾取量と勝率によって比較する．

6.1　実験の共通設定

本節では，CFR 法を用いた実験と Q 学習法を用いた実験の共通設定を述べる．CFR 法と Q 学習法を適用するにあたり，本研究では以下のようにゲーム状況を抽象化する．

- 開始時の各キャラクタの HP は 2
- 開始時の各キャラクタの Energy は 150
- 開始時のキャラクタ間の距離が 4px
- 開始時の残り時間 1 秒
- 開始時の体勢は STAND
- 4 フレームごとに行動の決定が可能

この状況は，待機し続ける相手に対して任意の攻撃行動が当たる距離であり，キャラクタは任意の攻撃行動を行うことができる状況である．また，この状況はどの攻撃行動が当たっても負けてしまう状況である．これらに加えて，キャラクタの行動集合は以下の 2 通りを考える．

行動集合 1　攻撃行動すべてからなる集合
行動集合 2　THROW_A (投げ技弱)，STAND_D_DB_BB (必殺技強)，STAND_GUARD (防御) からなる集合

ただし，THROW_A，STAND_D_DB_BB，STAND_GUARD については付録 A の表 13, 14 を参照されたい．先のゲーム状況において行動集合 1 の場合は，純戦略の組 (STAND_D_DB_BB, STAND_D_DB_BB) が均衡点である．行動集合 2 の場合は，純戦略の組の均衡点はありそうにない． もしディレイがなければ，行動集合 2 の各要素の関係は以下のようになっている．

- THROW_A は STAND_GUARD に勝てるが，STAND_D_DB_BB には負ける．
- STAND_GUARD は STAND_D_DB_BB に勝てるが，THROW_A には負ける．
- STAND_D_DB_BB は THROW_A には勝てるが，STAND_GUARD には負ける．

ただし，STAND_GUARD が STAND_D_DB_BB に勝つためには，STAND_GUARD を選択した直後の意思決定において，STAND_D_DB_BB を選択する必要がある．STAND_D_DB_BB を選択しない場合は負けとなる． つまり，行動集合 2 の各要素同士はジャンケンのグー・チョキ・パーのような関係になっている．

このような 2 つの行動集合に対し，相手の行動列の認識がディレイ 0 とディレイ 15 の場合で

CFR 法と Q 学習法を適用し，両手法の性能を比較する．

6.2 CFR 法を用いた実験

本実験では，先に述べた抽象化を行ったゲームに対し，CFR 法を適用する．

まず，2 種の行動集合とディレイで，情報集合，木の枝数，木の点，木の頂点数を表 2 に示す．ここで，ゲーム木の点は選択可能な行動の数が 2 つ以上存在する意思決定点とした．表 2 より，各

表 2　4 つのゲーム木の大きさ

	行動集合 1		行動集合 2	
	ディレイ 0	ディレイ 15	ディレイ 0	ディレイ 15
情報集合	74	47	86	24
枝	1766	1320	348	276
内部節点	106	78	116	92
葉節点	1661	1243	233	185
葉の経路の最大長	4	4	30	24

行動集合においてディレイが大きい方が情報集合，枝，頂点の数が少なくなっていることがわかる． これはディレイの仕様に起因すると考えられる (付録 B 参照)．また，行動集合 1 は枝と頂点の数が，行動集合 2 と比べて非常に多くなっている． これは行動集合 1 のサイズが，行動集合 2 のサイズと比べて大きいことが要因である．しかし，点の数は行動集合 2 の場合の方が多い． これは行動集合 2 に，選択した直後でも他行動の選択が可能な STAND_GUARD が属していることが要因であると考えられる．

次に，抽象化したゲーム状況に対し CFR 法を適用したときの搾取量を計算する． 搾取量の定義は次の通りである．$b = (b_1, b_2)$ をプレイヤすべての行動戦略の組とし，$\mathrm{br}(b_i)$ をプレイヤ i の行動戦略 b_i に対する最適応答の一つとする．このとき搾取量は以下の式で計算される．

$$\epsilon = \frac{U_1(\mathrm{br}(b_2), b_2) + U_2(b_1, \mathrm{br}(b_1))}{2} \tag{53}$$

二人ゼロ和ゲームでは，b は 2ϵ 均衡点であり，搾取量が 0 に近づけば近くほど，b は均衡点の利得に近い利得を与える戦略の組である．各ゲーム状況に対し，CFR 法を適用したときの搾取量の変化を図 9 に示す． ここで，図の横軸は反復回数，縦軸は搾取量を表す． 図 9 では，行動集合 1 のディレイ 0 とディレイ 15 の場合の曲線がほぼ重なっている．この図より，戦略の組が徐々に均衡点に近づいていくことがわかる．また，行動集合 1 の場合の方が搾取量の減少が早いことがわかる． これは，純戦略の組の均衡点が存在することに起因すると考えられる．

CFR 法で 10^4 回反復を行うのに，およそ 3，4 日程度の時間を要した． また，CFR 法を適用するためには，各情報集合での行動集合の大きさが全て同じだと仮定した場合，情報集合の数 × 行動集合の大きさ × プレイヤの数 ×4 程度の浮動小数点数を保持する必要がある．CFR 法の実験環境を表 3 に示す．

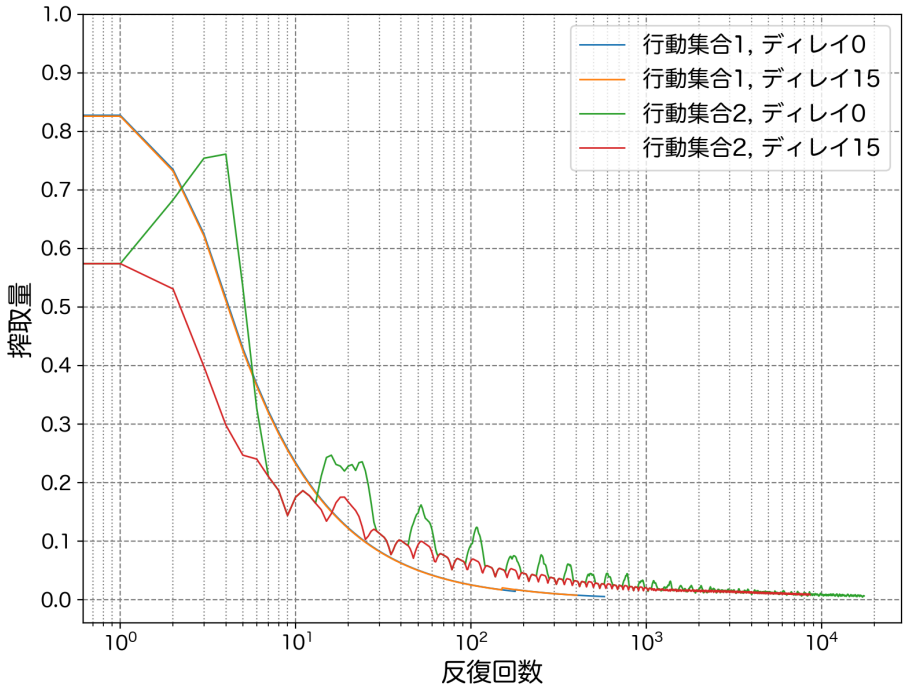

図 9　各ゲーム状況での搾取量の変化

表 3　CFR 法の実験環境

OS	メモリ	CPU
Ubuntu 18.04.3 LTS	16GB	Intel Xeon(R) CPU E31275@ 3.40GHz x 8

6.3　Q 学習法を用いた実験

本実験では先に述べた抽象化を行ったゲームに対し Q 学習法を適用する．挙動方策は ε グリーディ方策，推定方策はグリーディ方策とする．学習の際は，ε グリーディ方策のランダム行動の割合を，学習開始から 10^5 フレーム経過するまでの間に 1.0 から 0.1 に単調減少させる．10^5 フレーム経過後は 0.1 で固定する．また，行動価値関数を NN で近似し， 3.1.2 の experience replay と target network を導入する．NN は表 2 のゲーム木の点すべての行動価値関数を表現するのに十分な数の重みを持つ（表 4 参照）．NN の入力には，各キャラクタのパラメータ (162 ユニット，表 5 参照) とラウンド開始からの経過フレーム数 (1 ユニット) を用いる．学習における報酬は，勝てば 1，負ければ −1，それ以外は 0 とする．また，損失関数には二乗誤差を，重みの最適化手法には確率的勾配降下法を用いる．学習に関するパラメータを表 6 に示す．ここで，リプレイサイズは experience replay のパラメータであり，状態，行動，報酬，次状態の組を保持する数を表す．対戦相手にはランダムプレイヤを用い，30 万エピソード分の学習を行う．

各ゲーム状況に対し，反復回数 1000 回ごとの損失関数値の平均の変化を図 10 に示す．ここで反復とは，ミニバッチを用いた重みの更新処理 1 回を指す． グラフの横軸は学習の反復回数，縦

表 4　NN の構成

	入力の数	出力の数	ReLU	重みの数
入力	-	163	-	0
中間層 1	163	128	✓	20992
中間層 2	128	128	✓	16512
中間層 3	128	128	✓	16512
出力	128	41	-	5289

表 5　キャラクタの特徴．当り判定は box は左右の x 座標および上下の y 座標で表現

ユニット数	特徴
1	HP
1	Energy
1	向いている方向
4	キャラクタの当り判定 box
1	x 方向の移動速度
1	y 方向の移動速度
41	効果発動中の行動
4	拳・蹴りの当り判定 box
15	態勢
12	エネルギー弾の当り判定 box

表 6　学習のパラメータ

パラメータ名	値
学習率	5.0×10^{-6}
モメンタム	0.9
リプレイサイズ	10^5
ミニバッチサイズ	32

軸は反復回数 1000 回あたりの損失関数値の平均を表す．この結果から，学習が進んでいることが確認できた．

反復を 10^6 回行うのに，およそ 1 時間を要した． また，Q 学習を行う場合，ネットワークの重みの数だけ浮動小数点数を保持すれば良い．

Q 学習法と MctsAi は，期待勝率が最大となるように意思決定を行う点で類似している．しかし，MctsAi は 1 回の意思決定に 16 ミリ秒を要するが，Q 学習法は，本研究のネットワークの規模であれば 1 ミリ秒以内で意思決定を行うことができる．Q 学習法の実験環境を表 7 に示す．

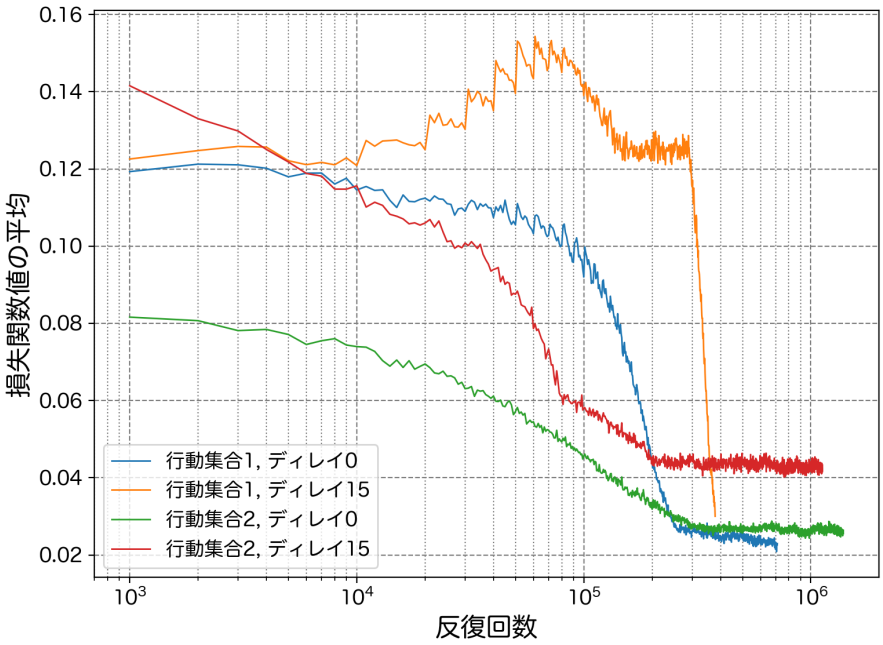

図 10　各ゲーム状況での損失関数値の平均の変化

表 7　Q 学習法の実験環境

OS	メモリ	CPU
Ubuntu 18.04.3 LTS	16GB	Intel Core i7 2.5GHz

6.4　搾取量による性能比較

本節では，CFR 法で求めた行動戦略を元に行動を決定する CFR プレイヤと，Q 学習法で得られた行動価値を基に行動を決定する Q 学習プレイヤの性能を搾取量によって比較する．ここで，Q 学習プレイヤは ε グリーディ方策に基づいて行動するものとし，$\varepsilon = 0.05$ とする．各ゲーム状況での最終的な搾取量を表 8 に示す．表 8 より，純戦略の組で均衡点をおおよそ表現できる状況で

表 8　各ゲーム状況に対する CFR 法と Q 学習法の搾取量

	行動集合 1		行動集合 2	
	ディレイ 0	ディレイ 15	ディレイ 0	ディレイ 15
CFR プレイヤ	0.006	0.008	0.007	0.009
Q 学習プレイヤ	0.044	0.044	0.966	0.950

は，CFR 法も Q 学習法も搾取量がほぼ 0 であることが確認された．しかし，混合戦略が必要になる状況では，Q 学習プレイヤは搾取される可能性があることが確認された．また，表には示されていないが，MctsAi の搾取量も，ほとんど決定論的な意思決定を行うため，1 付近になることがわ

かった．

6.5 勝率による性能比較

CFR 法と Q 学習法で得られた行動戦略を勝率によって比較する．これらのプレイヤに加え，MctsAi を用いて左右 1000 ラウンドずつ対戦を行い勝率を計算した．ただし，Q 学習プレイヤは ε グリーディ方策に基づいて行動するものとし，$\varepsilon = 0.05$ とする．勝ちを 1，負けを 0，引き分けを 0.5 としたときの，各ゲーム状況における Q 学習プレイヤの CFR プレイヤと MctsAi に対する勝率を推定した (表 9)．純戦略の組の均衡点が存在する場合，両手法の勝率は 5 割程度であった．

表 9　Q 学習プレイヤの勝率

	行動集合 1		行動集合 2	
	ディレイ 0	ディレイ 15	ディレイ 0	ディレイ 15
対 CFR の勝率	0.48 ± 0.01	0.49 ± 0.01	0.50 ± 0.04	0.51 ± 0.04
対 MctsAi の勝率	0.64 ± 0.03	0.64 ± 0.03	0.40 ± 0.03	0.48 ± 0.03

混合戦略が必要な場合，Q 学習プレイヤは搾取されうる戦略を取っているにもかかわらず，均衡戦略をとるプレイヤに対し勝率が 5 割程度であった．また，Q 学習プレイヤは MctsAi とおおよそ同等の性能を示した．

6.6 プレイヤープールを用いた実験

前々項では Q 学習プレイヤと MctsAi の搾取量がほとんど 1 であることを確認した．そこで本研究では，Multi-Agent Reinforcement Learning [26] のような方法で Q 学習プレイヤの性能を改善することを試みる．本研究ではまず，とりあえずの感触を掴むために，以下に示す簡潔な手順でプレイヤを開発する．

1 リプレイバッファと重みの履歴の組を N 個用意（リプレイバッファは空であり，重みの履歴はランダムに初期化された重み 1 個とする）
2 以下繰り返し
  2-1 N 個の組から重複を許さずランダムに 2 組選択し，プレイヤ 1 とプレイヤ 2 を構成
  2-2 ラウンドを最新の重みに基づく挙動方策で 1 つ生成
  2-3 各プレイヤのリプレイバッファを更新
  2-4 リプレイバッファが溜まっていれば，各プレイヤの最新の重みをそれぞれ一定回数更新し，重みの履歴を更新

この手順で開発されたプレイヤを複合プレイヤと呼ぶ．複合プレイヤの意思決定方法は以下の通りである．

1 ランダムに組を 1 つ選択

2 その組の最新の重みのグリーディ方策で意思決定

先の 4 つのゲーム状況のうち，行動集合 2 の場合における複合プレイヤの搾取量と複合プレイヤの CFR プレイヤに対する勝率を計算した（表 10）．勝率の計算方法は前項と同様である．ただし $N = 16$ とし，挙動方策には ε グリーディを用いた．その他の Q 学習法に関するパラメータは第 6.3 項のものと同じである．

表 10　各ゲーム状況における複合プレイヤに対する搾取量（最善で 1，最悪で 0）と複合プレイヤの CFR プレイヤに対する勝率

	行動集合 2	
	ディレイ 0	ディレイ 15
搾取量	0.352	0.476
対 CFR の勝率	0.40 ± 0.04	0.43 ± 0.04

結果として，Q 学習プレイヤに対する搾取量は大幅に改善された．その一方で，均衡戦略を取るプレイヤに対する勝率は少し小さくなってしまった．これの改善案としては，意思決定の際に組を選択する確率を何らかの指標に基づいて設定することや，意思決定の度に組を選択せず，例えばラウンドを通して組を固定することなどが考えられる．

7　おわりに

本研究では，不完全情報ゲームでありビデオゲームである格闘ゲームの限定的な状況を考えて，CFR 法と Q 学習法を適用し，それぞれの手法の性能を調査した．[8] で用いられていた experience replay と target network を導入し，Q 学習法の行動価値関数は NN で関数近似した． また，CFR 法を適用するために，ゲームルールの詳細を調査し，エミュレータを実装した．結果として，CFR 法は，本研究で扱ったような小さいゲーム状況に対しても，10^4 回の反復で 3,4 日を要した．ゲーム木が大きくなるにつれて，計算時間や必要なメモリが膨大に増えてしまうため，よりオリジナルのルールに近いゲーム状況に対する適用は困難だということがわかった．Q 学習法は決定論的なプレイヤに対して搾取されうることと，均衡戦略を取るプレイヤに対しては，良好な性能を発揮することがあることが確認できた．また，学習の反復を 10^6 回行うのに要した時間はおよそ 1 時間であった．学習を行うためにはリプレイバッファと NN の重みを 2 セットだけ保持すれば良いため，メモリ消費量も少なかった．

さらに，本研究では Q 学習プレイヤの性能の改善を試みた．結果として，Q 学習プレイヤに対する搾取量は大幅に改善された．その一方で，均衡戦略を取るプレイヤに対する勝率はやや悪化した．

謝辞

本研究を行うにあたり，ご指導をいただいた保木邦仁准教授に深く感謝いたします．また，村松・高橋研究室の皆様には，日々の議論を通して様々な助言を頂きました．心から感謝いたします．最後に，日頃から相談や助力をいただいた弊研究室の皆様に感謝いたします．

参考文献

[1] Tesauro, G., Temporal difference learning and TD-Gammon, *Communications of the ACM*, Vol. 38, No. 3, pp. 58–68, 1995.

[2] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., Mastering the game of Go with deep neural networks and tree search, *Nature*, Vol. 529, No. 7587, pp. 484–489, 2016.

[3] Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M., Deepstack: Expert-level artificial intelligence in heads-up no-limit poker, *Science*, Vol. 356, No. 6337, pp. 508–513, 2017.

[4] Brown, N. and Sandholm, T., Superhuman AI for heads-up no-limit poker: Libratus beats top professionals, *Science*, Vol. 359, No. 6374, pp. 418–424, 2018.

[5] Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C., Regret minimization in games with incomplete information, In *Advances in neural information processing systems*, pp. 1729–1736, 2008.

[6] Tammelin, O., Solving large imperfect information games using CFR+, *arXiv preprint arXiv:1407.5042*, 2014.

[7] Lanctot, M., Waugh, K., Zinkevich, M., and Bowling, M., Monte carlo sampling for regret minimization in extensive games, In *Advances in neural information processing systems*, pp. 1078–1086, 2009.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., Human-level control through deep reinforcement learning, *Nature*, Vol. 518, No. 7540, pp. 529–533, 2015.

[9] Lu, F., Yamamoto, K., Nomura, L. H., Mizuno, S., Lee, Y., and Thawonmas, R., Fighting game artificial intelligence competition platform, In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pp. 320–323, IEEE, 2013.

[10] Yoshida, S., Ishihara, M., Miyazaki, T., Nakagawa, Y., Harada, T., and Thawonmas, R., Application of monte-carlo tree search in a fighting game ai, In *2016 IEEE 5th Global Conference on Consumer Electronics*, pp. 1–2, IEEE, 2016.

[11] Sutton, R. S. and Barto, A. G., 強化学習, 森北出版, 2000.

[12] Szepesvari, C., 速習 強化学習 - 基礎理論とアルゴリズム -, 共立出版, 2017.

[13] 牧野貴樹, 澁谷長史, 白川真一, 浅田稔, 麻生英樹, 荒井幸代, 飯間等, 伊藤真, 大倉和博, 黒江康明, 杉本徳和, 坪井祐太, 銅谷賢治, 前田新一, 松井藤五郎, 南泰浩, 宮崎和光, 目黒豊美, 森村哲郎, 森本淳, 保田俊行, 吉本潤一郎, これからの強化学習, 森北出版, 2016.

[14] Watkins, C. J. C. H., *Learning from delayed rewards*, PhD thesis, Cambridge, 1989.

[15] Watkins, C. J. and Dayan, P., Q-learning, *Machine learning*, Vol. 8, No. 3-4, pp. 279–292, 1992.

[16] Melo, F. S., Meyn, S. P., and Ribeiro, M. I., An analysis of reinforcement learning with function approximation, In *Proceedings of the 25th international conference on Machine learning*, pp. 664–671, ACM, 2008.

[17] 岡谷貴之, 深層学習 (機械学習プロフェッショナルシリーズ), 講談社, 2015.

[18] 瀧雅人, 機械学習スタートアップシリーズ これならわかる深層学習入門 (KS 情報科学専門書), 講談社, 2017.

[19] 岡田章, ゲーム理論 新版, 有斐閣, 2011.

[20] Liao, Y., Yi, K., and Yang, Z., Cs229 final report reinforcement learning to play mario, Technical report, Technical report, Stanford University, 2012.

[21] He, K., Zhang, X., Ren, S., and Sun, J., Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[22] Brown, N., Lerer, A., Gross, S., and Sandholm, T., Deep counterfactual regret minimization, *arXiv preprint arXiv:1811.00164*, 2018.

[23] Fighting Game AI Competition, `http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-2h.html` (last access: Jan 26, 2020).

[24] Johanson, M., Waugh, K., Bowling, M., and Zinkevich, M., Accelerating best response calculation in large extensive games, In *Twenty-second international joint conference on artificial intelligence*, 2011.

[25] TeamFightingICE/FightingICE - GitHub, `https://github.com/TeamFightingICE/FightingICE` (last access: Jan 26, 2020).

[26] Zhang, K., Yang, Z., and Başar, T., Multi-agent reinforcement learning: A selective overview of theories and algorithms, *arXiv preprint arXiv:1911.10635*, 2019.

付録 A　キャラクタの行動と体勢の詳細

本付録では，エミュレータを実装する際に調査したゲームルールの詳細について述べる． まず，キャラクタの非攻撃行動について説明する． 次に，キャラクタの攻撃行動について説明する．最後に，キャラクタの体勢について説明する． FightingICE にはキャラクタのタイプが 3 つあるが，以降ではタイプ ZEN についてのみ紹介する．

非攻撃行動は，キャラクタの位置を変更する移動行動と，相手キャラクタの攻撃行動を防ぐ防御行動と，何もしない待機行動の 3 つに分けられる．非攻撃行動の一覧を表 11 に示す．

表 11　キャラクタの非攻撃行動

行動名	種類	概要
FORWARD_WALK	移動	前方へ一定距離歩く
DASH	移動	前方へ一定距離走る
JUMP	移動	真上にジャンプする
FOR_JUMP	移動	前方へ一定距離ジャンプする
BACK_JUMP	移動	後方へ一定距離ジャンプする
BACK_STEP	移動	後方へ一定距離飛び退く
STAND_GUARD	防御	その場で防御する
CROUCH_GUARD	防御	しゃがんだ状態で防御する
AIR_GUARD	防御	空中にいる状態で防御する
NEUTRAL	待機	何もしない

行動が JUMP，FOR_JUMP，BACK_JUMP ならば，行動を選択した直後に他行動の選択が不可になり，キャラクタが移動を開始する (図 11 の t_1)． t_1 から数フレーム経過後，他行動の選択が可能になる (t_2)． t_2 から数フレーム経過したのち，キャラクタが移動を終了する．終了した直後は，キャラクタの体勢が LANDING (後述) となり，数フレームの間は他行動を選択することができない (t_3)． t_3 から数フレーム経過したのち，体勢が STAND (後述) となり，行動の選択が可能になる (t_4)．

行動が STAND_GUARD，CROUCH_GUARD，AIR_GUARD，FORWARD_WALK ならば，行動を選択した直後にキャラクタが防御または移動を開始する (図 12 の t_5)． 数フレーム経過したのち，キャラクタが防御または移動を終了する (t_6)．AIR_GUARD を選択した場合 (図 13, 14 の t_{a}) のゲーム状況の変化は，防御を終了 (図 13 の t_{b}) したのちに着地する場合と，防御を終了する前 (図 14 の t_{c}) に着地する場合の 2 通りある． それぞれの場合のゲーム状況の変化の様子を図 13， 14 に示す．STAND_GUARD，CROUCH_GUARD，AIR_GUARD に関しては，防御の途中で相手の攻撃を受けた場合 (図 15，16，17 の t_7)，体勢が STAND_GUARD_RECOV，CROUCH_GUARD_RECOV，AIR_GUARD_RECOV となり，数フレームの間，他行動の選択ができない．t_7 から数フレーム経過したのち，STAND_GUARD，CROUCH_GUARD ならばそれぞれ体勢が STAND，CROUCH となり (図 15 の t_8)，AIR_GUARD ならば体勢が LANDING

(図 16 の t_3) または AIR (図 17 の t_d) となる．

行動が DASH ならば，DASH を選択したフレーム (図 18 の t_9) から 2 フレーム後 (t_{10}) にキャラクタが移動を開始する．t_9 から t_{10} の間は，特定のキーを続けて入力する．t_{10} から数フレームが経過したのち，キャラクタは移動を終了する (t_{11})．

行動が BACK_STEP ならば，BACK_STEP を選択したフレーム (図 19 の t_{12}) から 2 フレーム後 (t_{13}) に他行動の選択が不可になり，キャラクタが移動を開始する．t_{12} から t_{13} の間は，DASH と同様に，特定のキーを続けて入力する．t_{13} から数フレーム経過後，キャラクタの移動が終了する (t_{14})．t_{14} から数フレーム経過したのち，他行動の選択が可能になる (t_{15})．

NEUTRAL は何もしない行動を表す．

タイプ ZEN の非攻撃行動の詳細を表 12 に示す． 表の MotionLevel については後述する．

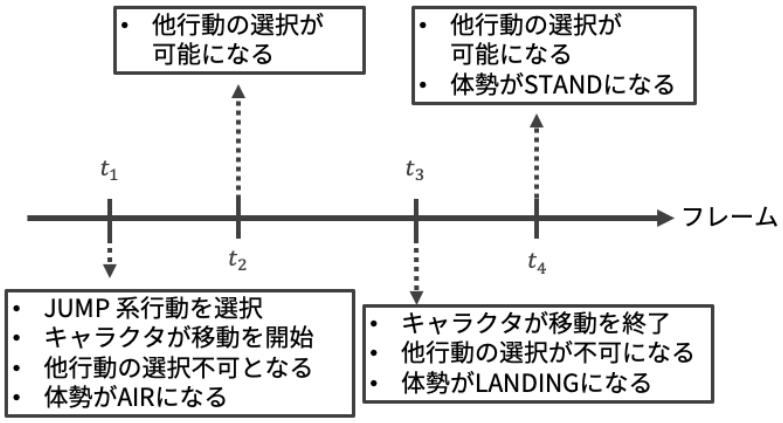

図 11　JUMP 系行動を選択した場合のゲーム状況の変化

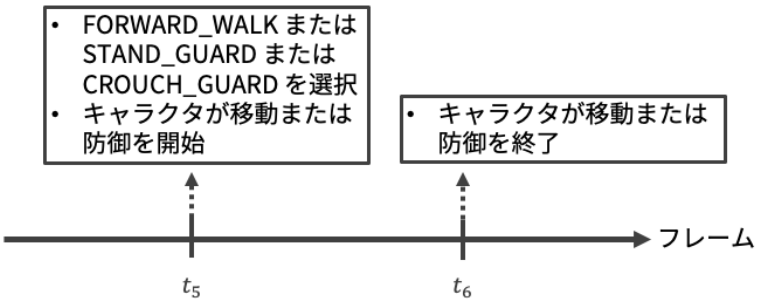

図 12　STAND_GUARD または CROUCH_GUARD または FORWARD_WALK を選択した場合のゲーム状況の変化

攻撃行動は相手にダメージを与えることができる行動である． 攻撃行動の一覧を付録の表 13 に示す．行動名に STAND，CROUCH，THROW がついている行動は地上で選択可能な行動を表し，AIR がついている行動は空中で選択可能な行動を表す．攻撃行動は主に Startup，Active，

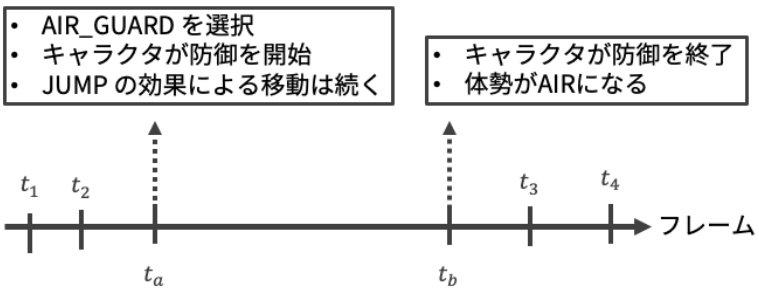

図 13　AIR_GUARD を選択した場合のゲーム状況の変化 (防御を終了したのちに着地する場合)

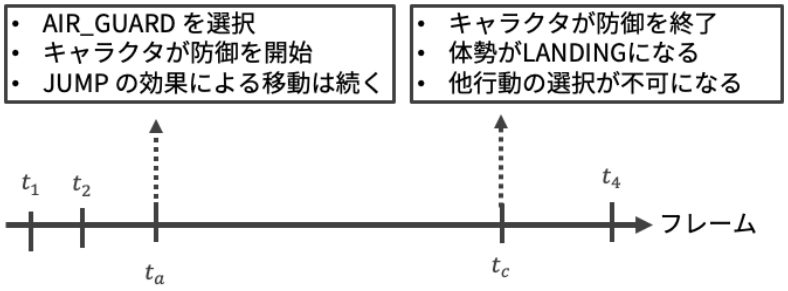

図 14　AIR_GUARD を選択した場合のゲーム状況の変化 (防御を終了する前に着地する場合)

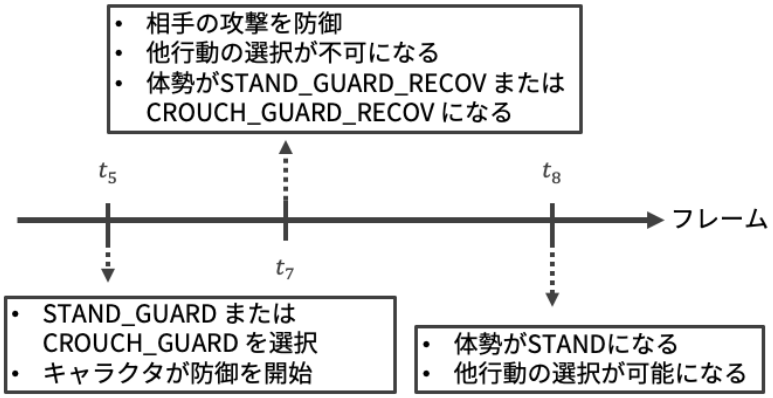

図 15　STAND_GUARD または CROUCH_GUARD で相手の攻撃を防御した場合のゲーム状況の変化

Recovery の 3 ステップからなる．

Startup　行動を選択してから攻撃判定が発生するまで．
Active　攻撃判定が発生しているとき．攻撃判定は赤い長方形の領域で表される．
Recovery　攻撃判定が無くなり，他行動が選択可能になるまで．

各ステップのフレーム数は行動によって異なる．ただし，例外として STAND_D_DF_FA，STAND_D_DF_FB，STAND_D_DF_FC，AIR_D_DF_FA，AIR_D_DF_FB は Active ステップの

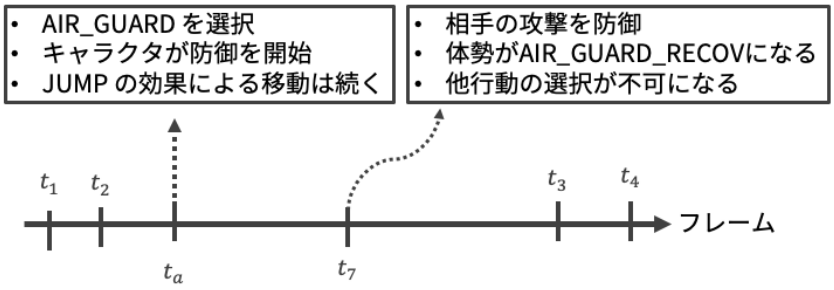

図 16 AIR_GUARD で相手の攻撃を防御した場合のゲーム状況の変化 (AIR_GUARD_RECOV 後の体勢が LANDING の場合)

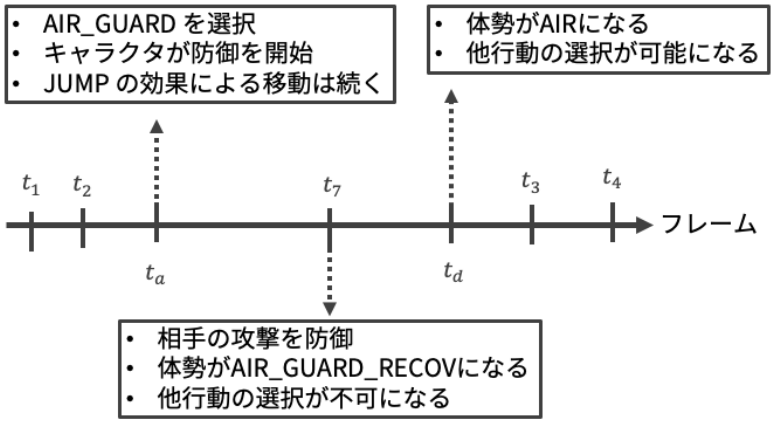

図 17 AIR_GUARD で相手の攻撃を防御した場合のゲーム状況の変化 (AIR_GUARD_RECOV 後の体勢が AIR の場合)

フレーム数が 0 であり，Recovery ステップに移行すると同時に攻撃判定を持つ物体 (エネルギー弾) が生成される．エネルギー弾は，AI の意思決定とは無関係に，一定の速度で一方向に進み，一定フレーム数が経過すると消える．攻撃は，攻撃判定を表す領域と，相手キャラクタの当たり判定を表す領域が重なったときに当たる．

攻撃を連続で 4 回以上当て続けると，通常のダメージに加えて追加ダメージを相手に与えることができる．このような一連の攻撃をコンボと呼ぶ． コンボの追加ダメージは

$$\left\lfloor 5 \times \frac{4}{\text{攻撃の連続ヒット数}} \right\rfloor \tag{54}$$

で計算される． ただし，二つの攻撃が連続で当たるとは，最初の攻撃が当たってから 30 フレーム

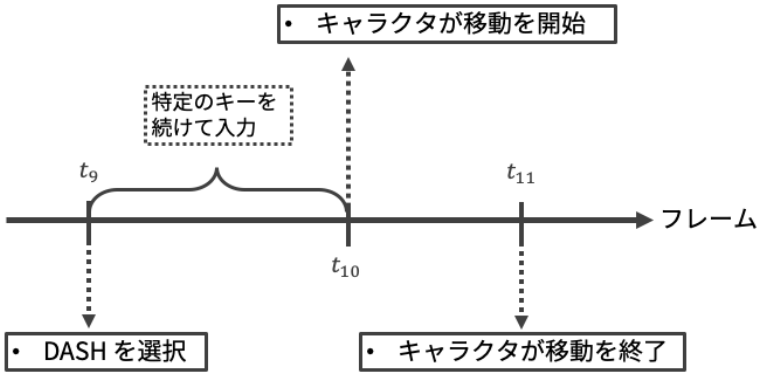

図 18　DASH を選択した場合のゲーム状況の変化

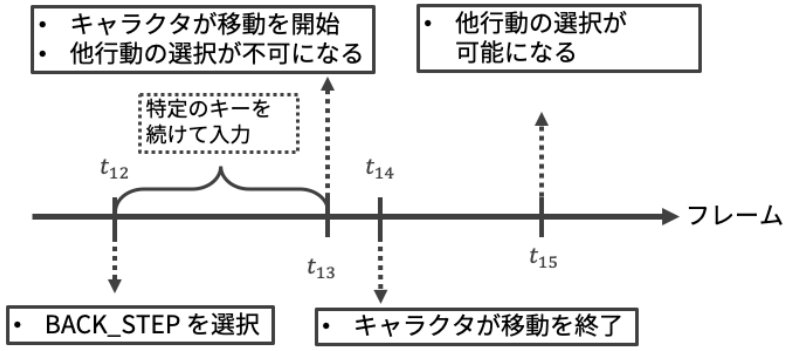

図 19　BACK_STEP を選択した場合のゲーム状況の変化

以内に次の攻撃が当たることである．

Recovery の途中から最後までのフレームは CancelAbleFrame と呼ばれる． このフレームで別の行動を選択すると，現在の行動が終了し別の行動の効果が発動する． ただし，効果が発動中の行動 a と次に行いたい行動 b は次の条件を満たす必要がある: a の MotionLevel より b の MotionLevel が小さい． MotionLevel とは，各行動に割り当てられている値である．

攻撃行動は各行動ごとにダメージ，使用 Energy，取得 Energy，ダウン可能かどうか (IsDown)，攻撃タイプ (AttackType) が異なる．ダメージは相手の HP の減少数を表す． 使用 Energy は行動を実行するために必要で消費される Energy を表す．取得 Energy は攻撃を相手に当てた際の Energy の増加量を表す．ダウン可能かどうかは，攻撃が相手にヒットした際に，相手の体勢 (後述) を DOWN にできるかどうかを表す．攻撃タイプは high，middle，low，throw の 4 つのいずれかである．攻撃タイプが high なら任意の防御行動で，middle なら STAND_GUARD または

表 12　ZEN の非攻撃行動の詳細

行動名	行動の効果が終了するフレーム数	MotionLevel	他行動が選択可能になるまでのフレーム数
FORWARD_WALK	5 (図 12 の $t_6 - t_5$)	10	0
DASH	12 (図 18 の $t_{11} - t_9$)	10	2 (図 18 の $t_{10} - t_9$)
JUMP	24 (図 11 の $t_3 - t_1$)	4	4 (図 11 の $t_2 - t_1$)
FOR_JUMP	22 (図 11 の $t_3 - t_1$)	4	4 (図 11 の $t_2 - t_1$)
BACK_JUMP	22 (図 11 の $t_3 - t_1$)	4	4 (図 11 の $t_2 - t_1$)
BACK_STEP	25 (図 19 の $t_{14} - t_{12}$)	2	25 (図 12 の $t_{15} - t_{12}$)
STAND_GUARD	23 (図 12 の $t_6 - t_5$)	10	0
CROUCH_GUARD	23 (図 12 の $t_6 - t_5$)	10	0
AIR_GUARD	23 (図 13 の $t_{\mathrm{b}} - t_{\mathrm{a}}$)	10	0
NEUTRAL	0	10	0

AIR_GUARD で，low なら CROUCH_GUARD で防ぐことができる．throw は防御できない行動を表す．タイプ ZEN の攻撃行動の詳細を表 14 に示す．

表 13　キャラクタの攻撃行動

行動名	概要
THROW_A	投げ技 (ダメージ弱)
THROW_B	投げ技 (ダメージ強)
STAND_A	通常技 (ダメージ弱)
STAND_B	通常技 (ダメージ強)
CROUCH_A	通常技 (ダメージ弱)
CROUCH_B	通常技 (ダメージ強)
STAND_FA	通常技 (ダメージ弱)
STAND_FB	通常技 (ダメージ強)
CROUCH_FA	通常技 (ダメージ弱)
CROUCH_FB	通常技 (ダメージ強)
STAND_D_DB_BA	必殺技 (ダメージ弱)
STAND_D_DB_BB	必殺技 (ダメージ強)
STAND_D_DF_FA	必殺技 (ダメージ弱)
STAND_D_DF_FB	必殺技 (ダメージ中)
STAND_D_DF_FC	必殺技 (ダメージ強)
STAND_F_D_DFA	必殺技 (ダメージ弱)
STAND_F_D_DFB	必殺技 (ダメージ強)
AIR_A	通常技 (ダメージ弱)
AIR_B	通常技 (ダメージ弱)
AIR_DA	通常技 (ダメージ弱)
AIR_DB	通常技 (ダメージ弱)
AIR_FA	通常技 (ダメージ弱)
AIR_FB	通常技 (ダメージ弱)
AIR_UA	通常技 (ダメージ弱)
AIR_UB	通常技 (ダメージ弱)
AIR_D_DF_FA	必殺技 (ダメージ弱)
AIR_D_DF_FB	必殺技 (ダメージ強)
AIR_F_D_DFA	必殺技 (ダメージ弱)
AIR_F_D_DFB	必殺技 (ダメージ強)
AIR_D_DB_BA	必殺技 (ダメージ弱)
AIR_D_DB_BB	必殺技 (ダメージ強)

表 14　ZEN の地上 (上部) と空中 (下部) における攻撃行動

行動名	ダメージ	StartUp	Active	Recovery	使用 Energy	取得 Energy	AttackType	IsDOWN	MotionLevel	エネルギー弾の出現フレーム数
THROW_A	10	5	1	24	5	2	throw	×	10	0
THROW_B	20	20	1	9	20	10	throw	×	10	0
STAND_A	5	4	3	11	0	2	high	×	7	0
STAND_B	10	5	3	15	0	5	high	×	7	0
CROUCH_A	5	3	3	12	0	3	low	×	7	0
CROUCH_B	10	11	3	6	0	5	low	×	7	0
STAND_FA	5	5	3	28	0	2	high	×	6	0
STAND_FB	12	12	3	28	0	10	middle	×	6	0
CROUCH_FA	5	4	3	31	0	2	low	×	6	0
CROUCH_FB	10	12	3	45	0	5	low	○	6	0
STAND_D_DF_FA	10	20	0	40	2	3	high	×	3	150
STAND_D_DF_FB	30	15	0	45	30	10	high	×	3	100
STAND_D_DF_FC	120	15	0	33	150	30	high	○	10	100
STAND_F_D_DFA	10	10	20	40	0	5	high	○	3	0
STAND_F_D_DFB	40	10	10	40	50	15	low	○	3	0
STAND_D_DB_BA	10	31	5	21	0	5	middle	×	3	0
STAND_D_DB_BB	25	5	10	45	50	15	middle	○	3	0
AIR_A	8	4	4	6	0	5	middle	×	7	0
AIR_B	10	6	9	8	0	10	middle	×	7	0
AIR_DA	8	5	10	33	5	5	middle	×	6	0
AIR_DB	10	15	10	25	5	10	middle	×	6	0
AIR_FA	8	11	15	16	0	5	middle	×	6	0
AIR_FB	10	7	13	15	0	10	middle	×	6	0
AIR_UA	10	5	10	33	0	5	middle	×	10	0
AIR_UB	20	8	10	32	0	10	middle	×	10	0
AIR_D_DF_FA	10	20	0	40	0	0	middle	×	3	150
AIR_D_DF_FB	30	20	0	40	50	15	middle	×	3	100
AIR_F_D_DFA	10	5	15	40	10	5	middle	×	3	0
AIR_F_D_DFB	20	14	8	38	40	15	middle	×	3	0
AIR_D_DB_BA	10	10	15	35	10	5	middle	×	3	0
AIR_D_DB_BB	40	14	15	41	50	15	middle	○	3	0

キャラクタの体勢には基本体勢と復帰体勢の 2 種がある． 基本体勢はキャラクタの行動時の体勢である．基本体勢には以下の 3 つの体勢がある:

- STAND
- AIR
- CROUCH

復帰体勢はキャラクタが攻撃を受けてから行動可能な体勢に戻るまでの体勢や，地面に着地してから行動可能な体勢に戻るまでの体勢や，攻撃を防御してから行動可能な体勢に戻るまでの体勢である．復帰体勢は以下の 12 種である．

- STAND_GUARD_RECOV
- CROUCH_GUARD_RECOV
- AIR_GUARD_RECOV
- STAND_RECOV
- CROUCH_RECOV
- AIR_RECOV
- CHANGE_DOWN
- DOWN
- RISE
- LANDING
- THROW_HIT
- THROW_SUFFER

STAND_GUARD_RECOV，CROUCH_GUARD_RECOV，AIR_GUARD_RECOV は相手の攻撃行動を防御した際の体勢を表す． CHANGE_DOWN，DOWN，RISE は相手をダウンさせる攻撃行動を受けたときの体勢を表す． 攻撃を受けたのち CHANGE_DOWN，DOWN，RISE の順で体勢が変化する．これらの体勢をとるときのゲーム状況の変化の様子を図 20 に示す．STAND_RECOV，CROUCH_RECOV，AIR_RECOV は攻撃行動のうち，相手をダウンさせることができない行動を受けたときの体勢を表す．LANDING は空中から地上に着地する際の体勢を表す．THROW_HIT は相手に投げ技を当てた際の体勢を表す．THROW_SUFFER は投げ技を当てられた際の体勢を表す．STAND_RECOV，CROUCH_RECOV，AIR_RECOV，THROW_HIT，THROW_SUFFER をとるときのゲーム状況の変化の様子を図 21 に示す．また，各復帰体勢は，その体勢をとるフレーム数が決まっている．タイプ ZEN の各体勢をとるフレーム数を表 15 に示す．

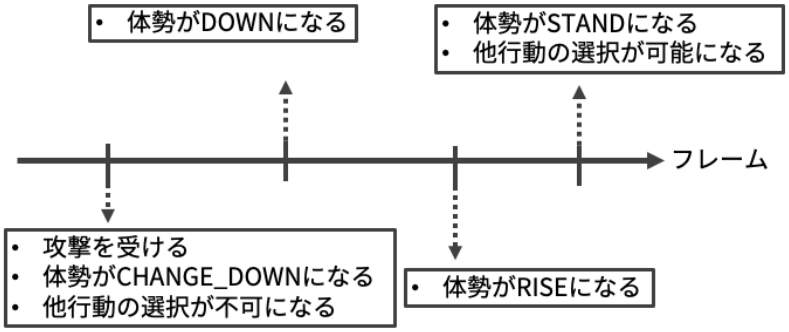

図 20　キャラクタをダウンさせる攻撃を受けた場合のゲーム状況の変化の様子

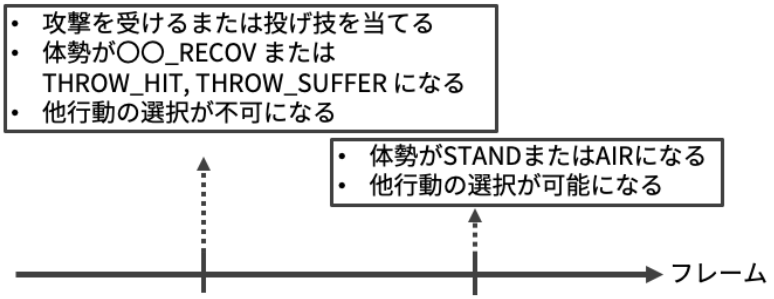

図 21　キャラクタをダウンさせることができない攻撃を受けた場合のゲーム状況の変化

表 15　ZEN の復帰体勢の詳細

体勢名	体勢をとるフレーム数
STAND_GUARD_RECOV	6
CROUCH_GUARD_RECOV	6
AIR_GUARD_RECOV	6
STAND_RECOV	28
CROUCH_RECOV	28
AIR_RECOV	28
CHANGE_DOWN	6
DOWN	21
RISE	29
LANDING	6
THROW_HIT	30
THROW_SUFFER	59

付録 B　ディレイの仕様について

本付録では，ディレイの仕様について説明する． ディレイとは，AI に与えられるゲーム状況を表すデータが数フレーム遅れる設定のことである．ディレイが n フレームの場合，ゲーム開始から n フレーム経過するまで，各キャラクタは待機行動を繰り返す． その後，各 AI に n フレーム前のゲーム状況が送られる．例えば，制限時間が 1 秒 (60 フレーム) でディレイ 15 の場合，AI は最大で 45 回しか意思決定を行うことができない．