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

2018E15 鹿谷　陸悟

令 和 0 5 年（ 2 0 2 3 年 ）　　2 月 1 7 日 提 出

Abstract

The machine learning is being used in modern society. Q-learning is one of machine learning techniques. Q-learning advances learning by repeating trial and error so as to take actions that maximize the defined value. The goal of this research is to create an AI that clears a simple game with only images. It determines the operation for the game from the read data. Q-learning is used to determine operations. Data are read from the home screen using an image recognition. Transfer learning with VGG16 is used for image recognition.

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。

機械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。また、VGG16は2014年に開発された畳み込みニューラルネットワークアーキテクチャである。1400万を超える画像のデータセットであるImageNetで92.7％のテスト精度を叩き出した。画像分類手法で使用されており、実装が簡単なため人気がある。

本研究では機械学習の1手法としてQ学習を用いて簡単なゲームを解くAIを作成する。ゲームからAIに与えられる情報は画像のみであり、画像認識を用いて必要なデータを読み取ることにする。VGG16の転移学習で画像認識をして、Q学習でゲームを解く。

第2章　機械学習

2.1　機械学習の分類[1)]

機械学習 (Machine Learning) は人工知能 (AI) の一分野である。機械学習には教師あり学習、教師なし学習、強化学習、ディープラーニングなど多くの手法がある。

1. 教師あり学習

   教師あり学習 (Supervised Learning) とは、コンピューターに「入力」と「正しい出力」が紐づいた学習データを与え、ある入力を受けたときに正しい出力を返せるアルゴリズムを構築する学習方法である。連続する数値を予測したり、データの分類を行うときに用いられる。出力は事前に人間が与えたものの中から選ばれる。

2. 教師なし学習

   教師なし学習 (Unsupervised Learning) とは、コンピューターに「入力」データのみを与え、データの中に内在するパターンなどをコンピューターが独自で抽出する手法である。クラスタリングや次元削減などで用いられる。出力は事前に人が定めたものではない。

3. 強化学習

   強化学習 (Reinforcement Learning) では、システム自身が試行錯誤しながら、最適なシステム制御を実現する。今回の実験で使用した手法であり、詳しくは次項で説明する。

4. ディープラーニングディープラーニング (Deep Learning) では、コンピューターが自ら学習データから特徴量を抽出し、予測モデルを構築することができる。従来の機械学習と異なり人間による「特徴量抽出 (Feature Extraction)」が不要という特徴がある。

2.2　強化学習[1)]

強化学習では、コンピューターはある「環境」の中で、目的として設定された「報酬」を最大化するための行動を学習する。環境の中で行動を決定し、環境に対して影響を与えるものをエージェントと呼ぶ。強化学習の流れは以下のようになる。

i) エージェントがある環境の中に置かれ、その環境に対して「行動」を起こす

ii) 環境がエージェントに、行動により更新された「状態」と「報酬」をフィードバッ

クする

「状態」とは、エージェントが存在する空間の情報であり、エージェントが行動する度に更新される。「報酬」はエージェントの行動の指針となるものであり、マイナスの報酬は罰とも呼ばれる。

iii) 環境からのフィードバックを元に、「方策」を修正する

エージェントは、自身の行動に対する環境からの「状態」と「報酬」のフィードバックを元に、将来得られる「価値」を最大化する「方策」を導き出す。方策の導出に用いられるのが「価値関数」である。ここで用いられる価値とはより長いスパンで行動した結果得られる最終的なアウトプットを指し、一度の行動のみに対してのフィードバックである報酬とは異なる。

iv) これまでの一連の行動の結果として変化した環境の中で、再びエージェントが環境に対して行動を起こす

i) から iv) までの過程を繰り返し方策を修正しながら強化学習は進んでいく

強化学習の発展系としてディープラーニングを組み込んだ深層強化学習と呼ばれる手法が存在する。強化学習の価値関数の更新や状態の特徴量抽出などを、畳み込みニューラルネットワークや再帰型ニューラルネットワークで疑似的に再現している。深層強化学習のモデルである「R2D2」は Atari という強化学習の性能評価で一般的に用いられるゲームプレイにおいて、人間のプレイと比べて平均約 35 倍というパフォーマンスを叩き出している。

2.2.1 価値関数 [2)]

価値関数は「状態価値関数」と「行動価値関数」の 2 種類がある。状態価値関数は、ある状態 s における、エージェントが方策 π を実行した際に得られる価値を計算する。状態価値関数は次の式で表される。

$$v_\pi(s) = E[G_t|S_t = s] \tag{2.1}$$

ここで E は期待値である。カッコの中は条件付き確率を表しており、S_t は時間 t での状態である。状態が s の時の報酬和が G_t である。G_t は具体的に次の式で表される。γ は割引率、R は報酬である。γ は定数であり、大きいほど将来の Q 値を重視する。

$$G_t = \sum_{k=1}^{n} \gamma^{(k-1)} R_{(t+k)} \tag{2.2}$$

Figure 2.1　Maze.[2)]

一方で行動価値関数は、状態 s において行動 a を実行した後に方策 π を実行した際に得られる価値を計算する。状態価値関数との違いは初めに方策とは無関係の行動 a をとるか否かである。こうする事で、状態の価値を測るのか、行動の価値を測るのかを区別する。行動価値関数は次の式で表される。A_t は時間 t でとった行動である。

$$q_\pi(s,a) = E[G_t|S_t = s, A_t = a] \tag{2.3}$$

強化学習は、すべての状態にわたって価値関数を計算する必要がある。 これを効率よく計算するために用いられるのがベルマン方程式 (Bellman equation) といい、ある状態 s における価値関数と、その次の状態における価値関数と関連付ける方程式である。 なお、この方程式が成り立つ前提として、学習対象がマルコフ決定過程に従う必要がある。次にベルマン方程式を示す。

$$v_\pi(s) = E[R_{(t+1)} + \gamma v_\pi(S_{(t+1)})|S_t = s] \tag{2.4}$$

どちらも方策に応じた価値を測るものだが、状態価値関数は現在の状態の価値を表すのに対して、行動価値関数はその状態で選択した行動の価値を表す。

実例として、Figure 2.1 の迷路において、移動ルールに従って行動した時の各状態の状態価値、行動価値を求める。状態はどのマスにいるかで決まり、状態１～状態９と呼ぶことにする。方策 π は「外壁に突き当たるまでは下に進み突き当たったら右に進む」とする。報酬は以下の通り。割引率 γ は 1 とする。

Figure 2.2　State value function.[2)]

1. 移動すると -1
2. ゴールにたどり着くと$+5$
3. 太線を越えて移動すると -3
4. 太線を越えてゴールにたどり着くと$+2$

状態価値関数は Figure 2.2 の通りになった。それぞれのマスに状態の価値を記載してある。例として状態 4 の価値の求め方を説明する。状態 4 からスタートして方策に従って動くと、移動ルートは状態 4 →状態 7 →状態 8 →状態 9 になる。従って状態 4 の価値は次のように求められる。

$$v_\pi(S4) = (-1) + (-3) + 5 = 1 \tag{2.5}$$

ベルマン方程式を用いると次のようにも表せる。

$$v_\pi(S4) = (-1) + v_\pi(S7) = 1 \tag{2.6}$$

行動価値関数は Figure 2.3 の通りになった。それぞれの方向に移動する行動の価値を記載してある。例として状態 5 から上に行動しその後方策に従う場合の行動価値を求める。移動ルートは状態 5 →状態 2 →状態 5 →状態 8 →状態 9 になる。従って状態 5 から上に移動する行動の価値は次のように求められる。

$$q_\pi(S5, up) = (-1) + (-1) + (-1) + 5 = 2 \tag{2.7}$$

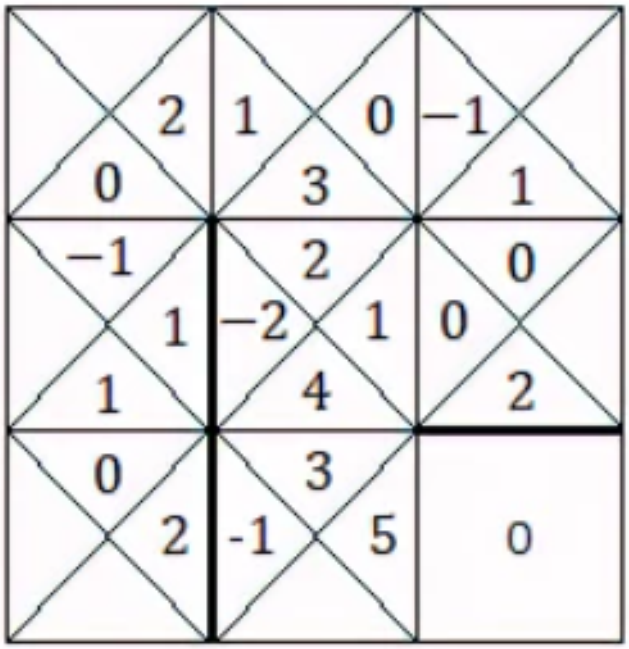


Figure 2.3　Action value function.[2)]

2.3　畳み込みニューラルネットワーク [3)]

ニューラルネットワークとは、人間の脳内にある神経細胞ニューロンとその繋がりの神経回路網を模した数式的なモデルである。ニューラルネットワークは主に、入力層、中間層、出力層から構成される (Figure 2.4)。入力データを受け取る層が入力層、予測値を返す層が出力層、入力層と出力層に挟まれたその他の層が中間層である。各層にはまるで神経細胞のように、前の層から受け取った情報を処理して次の層へ流すノードがある。ノード同士を結んでいるのがエッジで、それぞれ重みという情報を持っている。

畳み込みニューラルネットワーク (Convolution Neural Network) は画像認識に用いられるニューラルネットワークであり、中間層に畳み込み層とプーリング層を持つ。データから直接学習することができるディープラーニングのため手動での特徴抽出は不要である。畳み込み層は出力が入力の一部しか影響を受けない局所性の強い処理が行われ、入力データの特徴を抽出することができる。プーリング層は入力を各領域に区切りその領域の代表する値を抽出し情報量を削減する。畳み込み層とプーリング層を交互に並べることで、入力データから特徴抽出と情報量削減が何度も繰り返され最終的に抽出された特徴量は全結合層に渡される。全結合層とは隣り合う全てのノードとエッジを持つ層であり、畳み込みニューラルネットワークで得た特徴量に基づき結果を出力する。

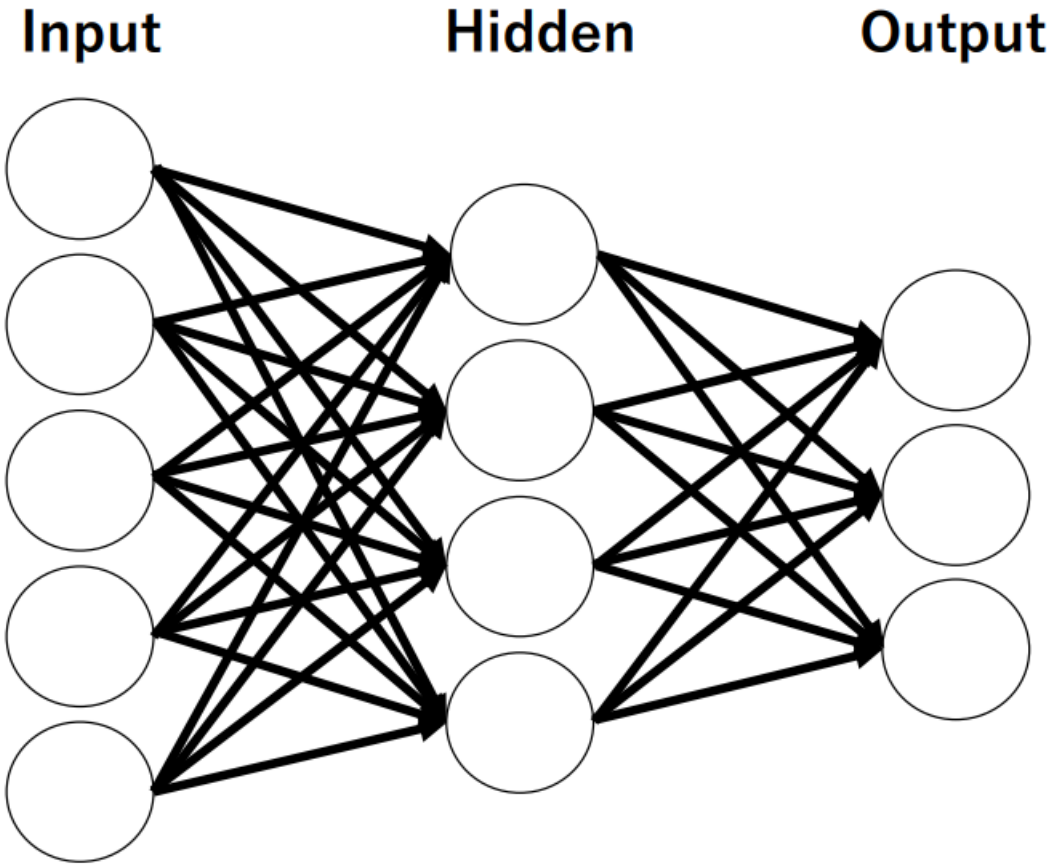


Figure 2.4　Neural network.[4)]

2.3.1　畳み込み層 [4)]

入力画像では、各ピクセルが隣同士のピクセルの値と強い関係性のある性質 (局所性) を持っている。この局所性を使用し、特徴を抽出するのが畳み込み層である。特徴抽出には複数のフィルタという正方行列を使用し、入力が画像と各フィルタを畳み込み演算することによって得られる。 Figure 2.5 では、入力画像サイズが 4×4 、フィルタサイズが 2×2 で畳み込みをした例である。入力行列の左上とフィルタの重なる各要素を乗算し、その総和を一つの要素とする。フィルタを右にずらしながら同じ計算を行い、最も右まで行きついたら下の段へ移動し同じ処理を行う。フィルタが移動できなくなるまで計算を行う。計算結果の行列を特徴マップ (feature map) という。フィルタの値によって特徴マップに現れる特徴が変わる。

2.3.2　プーリング層 [4)]

プーリング層は通常畳み込み層の直後に配置される。この層は、入力を領域に区切りその領域を代表する値を抽出し並べたものである。この処理により、元画像の位置がぼ

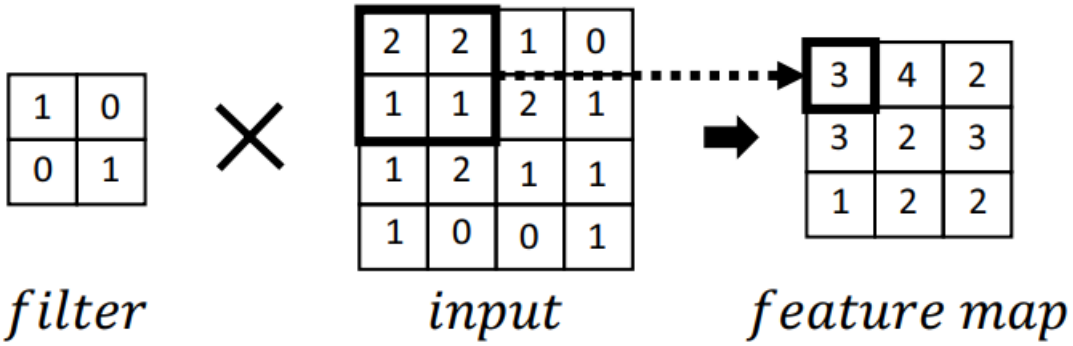


Figure 2.5　Convolution.[4)]

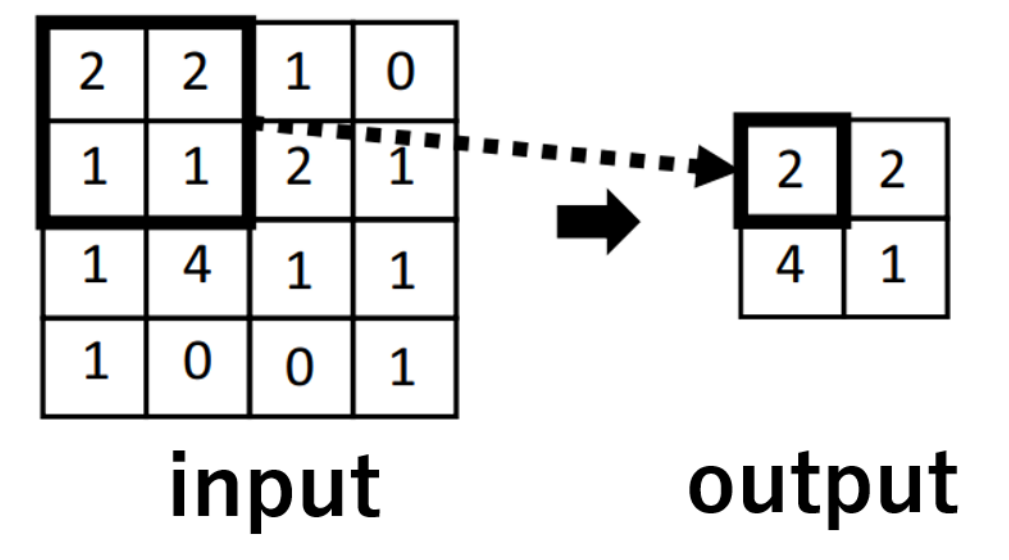


Figure 2.6　Pooling[4)].

かされる。さらに扱う情報量が削減され、計算も容易になる利点がある。 プーリングには、Figure 2.6 のように領域の最大値をとるマックスプーリング (max pooling) と領域の平均をとるアベレージプーリング (average pooling) がある。

第3章　実験で使用した技法

3.1　Q学習[1),5),6)]

Q学習とは、Q関数と呼ばれる行動価値関数を更新しながら学習する強化学習の手法の一つである。状態、行動、報酬の3要素が重要であり、これを変えると学習の速度や収束の仕方が変化する。エージェントが1度行動してから環境から状態・報酬が帰って来るまでのプロセスを1ステップと呼ぶ。また環境をリセットしてから次のリセットまでを1エピソードと呼ぶ。1ステップごとにQ関数は以下の式に従って更新される。

$$Q_{(s_t,a)} \leftarrow Q_{(s_t,a)} + \alpha[r_{t+1} + \gamma \max_a Q_{(s_{t+1},a)} - Q_{(s_t,a)}] \tag{3.1}$$

ここではs_tが時刻tでの状態、aがエージェントが取った行動、r_{t+1}が得た報酬、αが学習率、γが割引率を示す。$Q_{(s_t,a)}$が状態s_tにおける行動aの価値を表す。r_{t+1}が高いほど$Q_{(s_t,a)}$が増加し、マイナスの場合は$Q_{(s_t,a)}$が減少する。$\max_a Q_{(s_{t+1},a)}$は次の状態の行動価値関数の中で最大の値を表し、これが高いほど$Q_{(s_t,a)}$が増加する。αは$0 < \alpha < 1$を満たし、これが大きいほど更新されるQ値の変化量が大きくなる。γは$0 < \gamma < 1$を満たし、これが大きいほど将来のQ値を重視する。2ステップ後以降のQ値も影響するが、γにより将来のQ値ほど影響が少なくなる。またQ値をまとめたリストをQテーブルと呼ぶ。

3.1.1　Q学習の例:Cartpole問題[7)]

Cartpole問題はFigure 3.1のように直線上を動く台車に取り付けられた振り子を倒さないようにするにはどうすればいいかという問題である。できる操作は台車を左右どちらかに押すかの2種類だけである。振り子が一定の角度越えて傾いてしまうか台車が初期地点から一定の距離離れてしまうと失敗となる。振り子の角度と速度、台車の位置と速度が読み取れる。

Q学習の例としてCartpole問題を解く。状態は振り子の角度と速度、台車の位置と速度になる。行動は右に押すか左に押すかの2つである。報酬は1ステップごとに+1、振り子が倒れるか台車が遠くに移動して失敗となったら−200とする。失敗になるか200ステップ経過するまでを1エピソードとする。400エピソード学習した。

学習の様子をFigure 3.2に示す。合計報酬200のエピソードは200ステップ経過するまで振り子を倒していない。合計報酬が0未満のエピソードは途中で失敗している。学

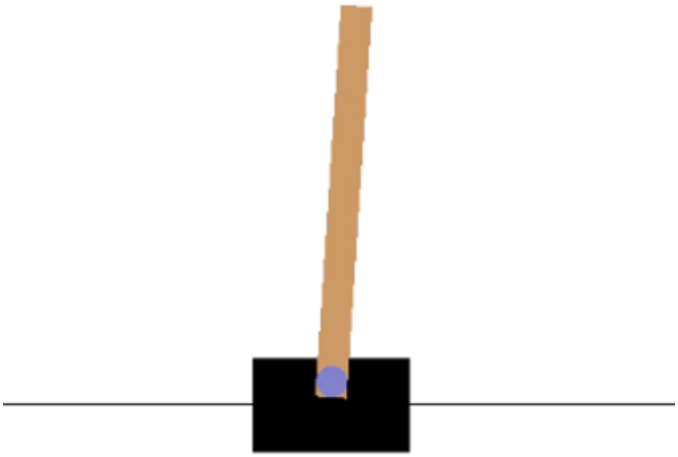


Figure 3.1　Cartpole.[7)]

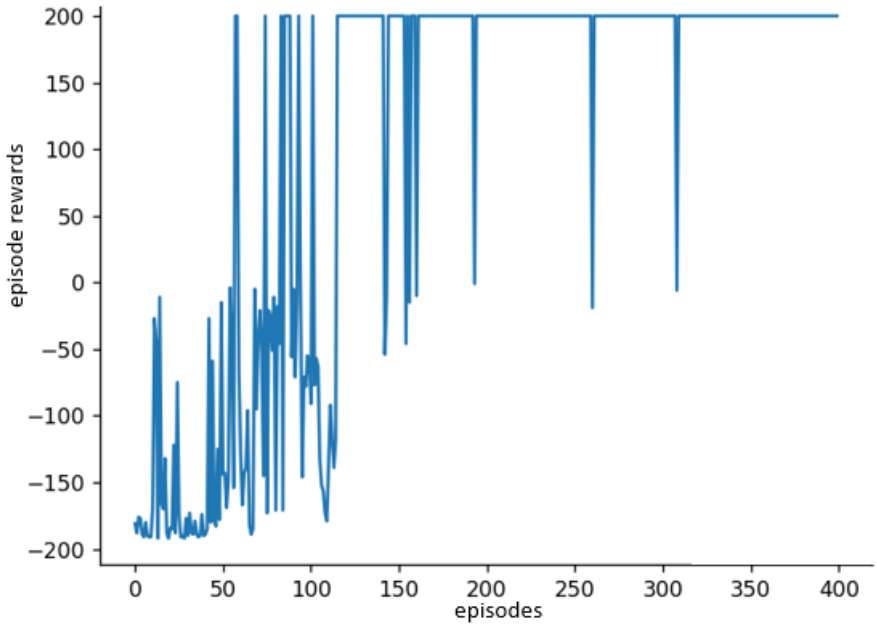


Figure 3.2　Cartpole episode reward.

習初期は 200 ステップ経過する前に振り子を倒していたが、徐々に合計報酬が増えていき 150 エピソードを越えてからはほとんど失敗することはなくなった。

3.2　VGG16[8),9),10),11)]

VGG16 とは、13 層の畳み込み層と 3 層の全結合層の計 16 層からなる畳み込みニューラルネットワークである。ImageNet と呼ばれる大規模画像データセットで学習済みのモデルで、画像を 1000 個のカテゴリに分類できる。結果として、このネットワークは広範囲のイメージに対する豊富な特徴表現を学習している。

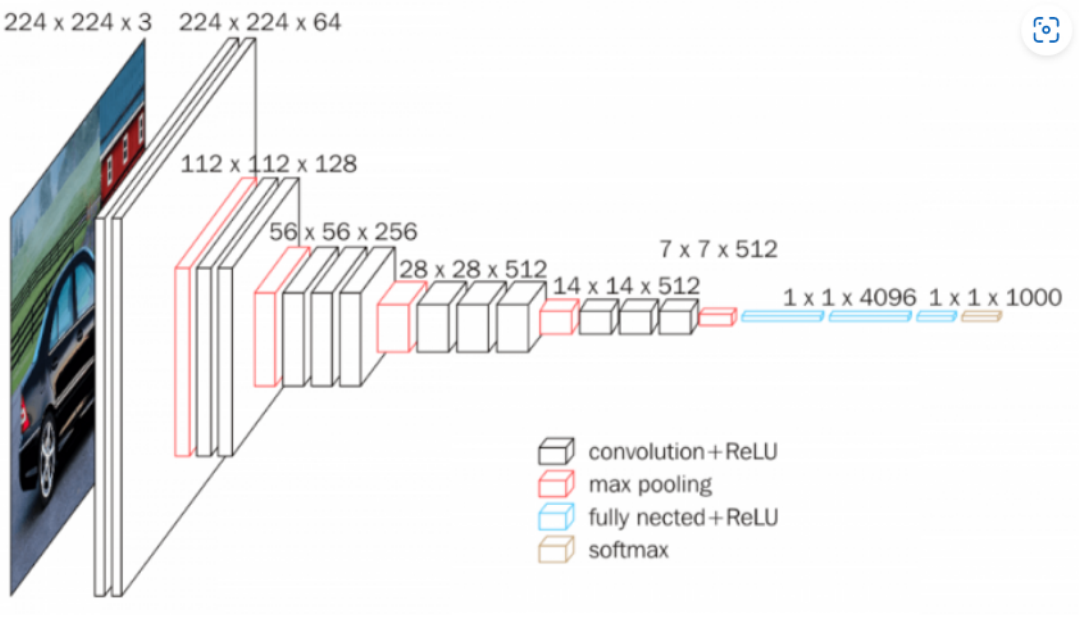


Figure 3.3　VGG16 architecture.[10)]

トレーニング中、最初の畳み込み層への入力は固定サイズの 224 x 224RGB 画像である。ここで行われる前処理は、各ピクセルからトレーニングセットで計算された平均 RGB 値を引くことだけである。画像は畳み込み層のスタックを通過する。2〜3 の畳み込み層の後にはマックスプーリング層があり、これが 5 ブロック存在する。マックスプーリングは、2 × 2 のピクセルウィンドウで実行される。畳み込み層のスタックに続く 3 つの全結合層がある。最初の 2 つはそれぞれ 4096 チャネルを持ち、3 番目は 1000 のカテゴリに分類するため 1000 チャネルを含む。最後の層はソフトマックス層である。ソフトマックス層は前の層の出力を指定した次元で総和 1 の確率に変換する。VGG16 の構造を Figure 3.3 に示す。

VGG16 の最後の全結合層を外して自作の層に置き換えることで学習済みの畳み込み層を残したまま、新たなモデルを作ることができる。この手法を転移学習 (transfer learning) と呼ぶ。転移学習を利用すると元の VGG16 とは異なる分類方法やカテゴリの数の画像分類ができる。

第4章　実験

4.1　目的

目的はゲームをクリアできるAIを制作することである。しかしいきなり深層強化学習を利用したり、複雑なゲームをクリアするAIを作るのは困難である。そこで目標として単純なゲームを画像だけでクリアするAIを制作することとした。

制作するAIモデルの概要をFigure 4.1に示す。まずゲーム画面から画像認識モデルを利用してデータを読み取る。次に読み取ったデータからゲームに対しての操作を決定する。画像認識モデルにはVGG16の転移学習を用いる。操作を決定するためにQ学習を用いる。

4.2　ゲームの概要

ゲーム画面をFigure 4.2に示す。フィールドは縦横それぞれ8マス、計64マスからなる。各マスは左上のマスを(0,0)とする二次元座標系で表す。フィールドにはプレイヤー(青い円)、目的地(緑の円)、敵(赤の円)が存在する。目的地と敵が同じマスに存在している場合は黄色の円で表示される。

プレイヤーは隣り合う上下左右のマスのどれかに移動する。ただしフィールドの端からフィールドの外側に移動した場合は移動前のマスに戻される。プレイヤーが目的地と同じマスへ移動するとスコアを1得て、目的地がプレイヤーとその周囲8マスを除いたランダムな座標に移動する。敵はプレイヤーが3回移動するたびにプレイヤーがいる方向に1マス移動する。

スコアが5になるとゲームクリアとなる。敵とプレイヤーが同じマスにいる、もしく

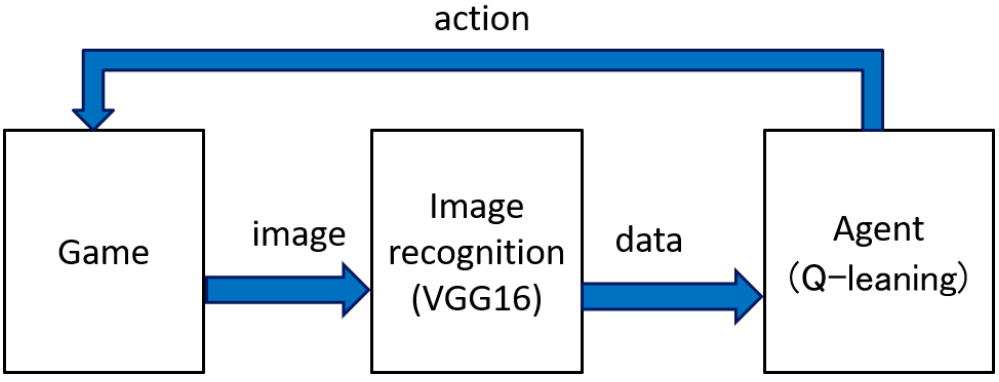


Figure 4.1　Model overview.

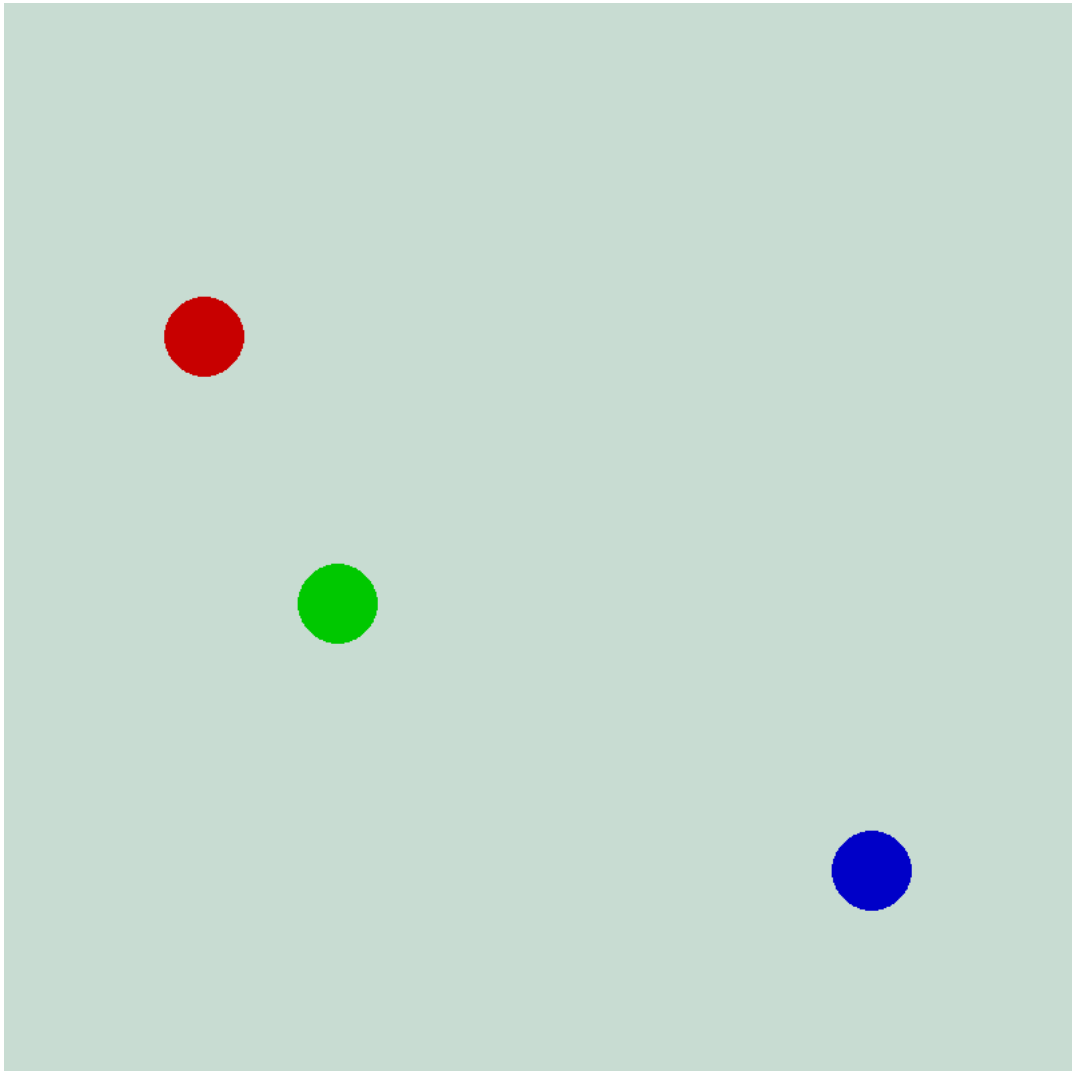


Figure 4.2　Game screen.

はプレイヤーが 200 回移動するとゲームオーバーとなる。

ゲーム開始時はプレイヤーの座標は (6,6)、目的地と敵の座標はプレイヤーとその周囲 8 マスを除いたランダムな座標になる。

4.3　VGG16 転移学習

ゲーム画面からゲームの状態を認識するために VGG16 の転移学習を行った。VGG16 の全結合層を外して新たな全結合層に置き換え、新しいモデルを作成した。Figure 4.3 に今回転移学習するモデルの構造を示す。dense(Dense) の層から、3 つの出力層 (output1, output2, output3) に接続されている。3 つの出力層はそれぞれプレイヤー、目的地、敵の座標を出力する。出力層同士はエッジを持たない。block4_pool (MaxPooling2D) の層までが元から学習済みの部分でこの部分の重みは更新しない。これ以降の重みを学習

Table 4.1　VGG16 result.

	training data		validation data	
	accuracy	loss	accuracy	loss
Player	1.0000	0.0062	1.0000	0.0028
Target	1.0000	0.0018	1.0000	0.0071
Enemy	1.0000	0.0028	0.9948	0.0198

させる。

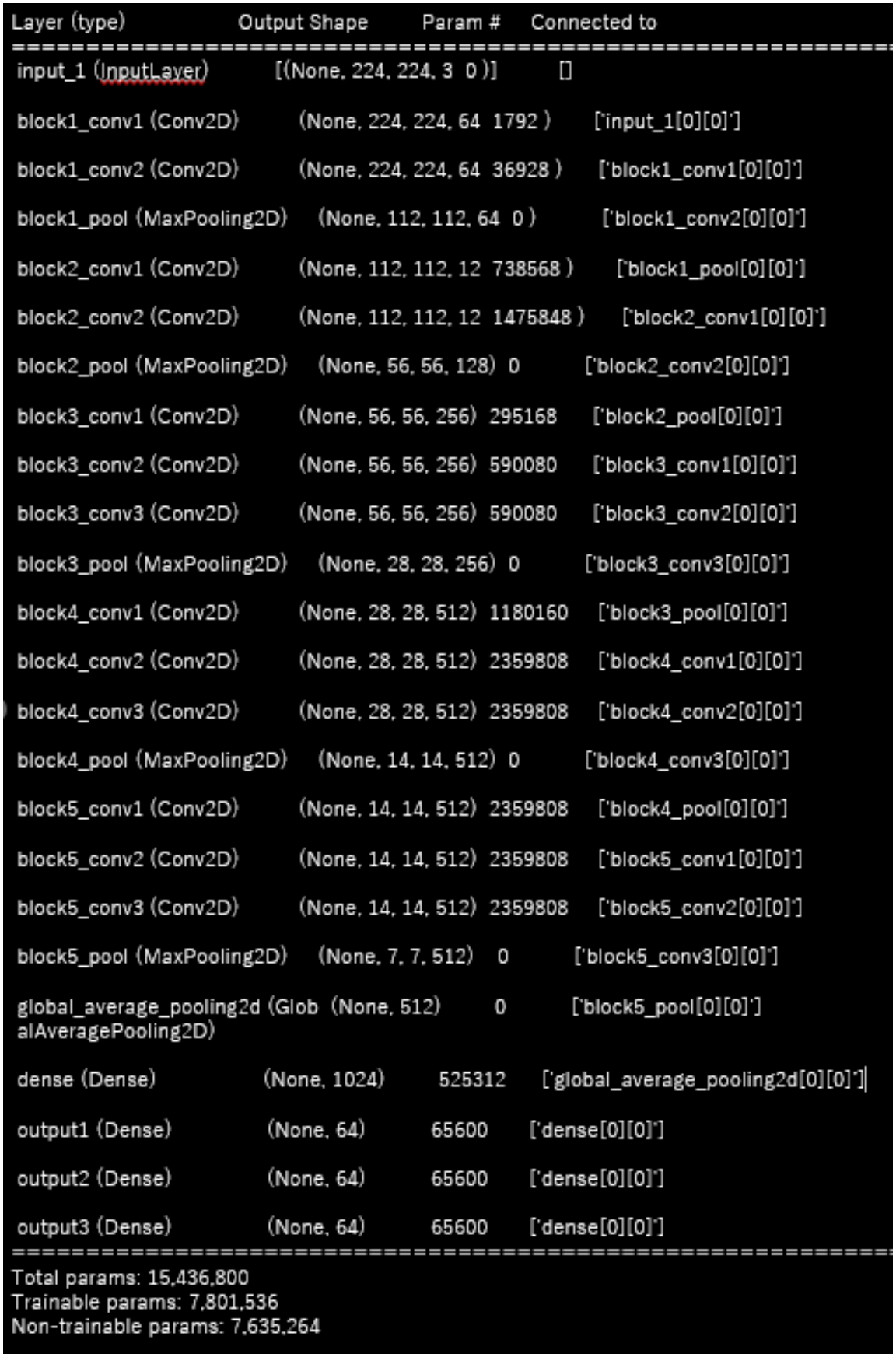
VGG16 はプレイヤー、目的地、敵の位置をそれぞれ 64 個のカテゴリに分類した結果を出力しなければならない。これはゲーム画面から $64^3 = 262144$ 通りの状態を判別できることを意味する。なお実際のゲームではプレイヤーが目的地のあるマスへ移動すると目的地がランダムな座標に移動するため、プレイヤーと目的地が同じマスに存在する $64^2 = 4196$ 通りの状態は起こりえない。またプレイヤーと敵が同じマスにいる $64^2 = 4196$ 通りの状態はゲームオーバーになった後ということになる。

転移学習のため、学習用の画像データ 9940 枚と検証用の画像データ 964 枚を用意した。学習用と検証用の画像はプレイヤー、目的地、敵をランダムで配置したものを使用した。ただしプレイヤーと目的地が同じマスに存在する画像とプレイヤーと敵が同じマスに存在する画像は学習が不要であるため除外した。

学習結果は Table 4.1 の通り。学習用データと検証用データに対する accuracy と loss を示す。accurcy は正解率であり、accuracy が 1 のとき 100 %で正解していることを示す。loss は損失関数であり、0 に近いほど学習が進んでいることを示す。敵の検証用データに対する accuracy は 1 に届かなかったものの、いずれの要素に対しても高い精度で検出できるようになった。

4.4　Q 学習

Q 学習の過程においてはゲーム画面から状態を読み取るのではなく、ゲームから直接座標を受け取って学習を進めた。これには 2 つの理由がある。まずは VGG16 を通して座標を認識しようとすると 1 ステップごとに十数倍の時間がかかるからである。もう一つは VGG16 がまれに誤った座標を出力した場合があり、それが Q 学習に悪影響を及ぼすことを防ぐためである。



Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 224, 224, 3	0)]	[]
block1_conv1 (Conv2D)	(None, 224, 224, 64	1792)	['input_1[0][0]']
block1_conv2 (Conv2D)	(None, 224, 224, 64	36928)	['block1_conv1[0][0]']
block1_pool (MaxPooling2D)	(None, 112, 112, 64	0)	['block1_conv2[0][0]']
block2_conv1 (Conv2D)	(None, 112, 112, 12	738568)	['block1_pool[0][0]']
block2_conv2 (Conv2D)	(None, 112, 112, 12	1475848)	['block2_conv1[0][0]']
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	['block2_conv2[0][0]']
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168	['block2_pool[0][0]']
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080	['block3_conv1[0][0]']
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080	['block3_conv2[0][0]']
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	['block3_conv3[0][0]']
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	['block3_pool[0][0]']
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808	['block4_conv1[0][0]']
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808	['block4_conv2[0][0]']
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	['block4_conv3[0][0]']
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808	['block4_pool[0][0]']
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808	['block5_conv1[0][0]']
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808	['block5_conv2[0][0]']
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	['block5_conv3[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0	['block5_pool[0][0]']
dense (Dense)	(None, 1024)	525312	['global_average_pooling2d[0][0]']
output1 (Dense)	(None, 64)	65600	['dense[0][0]']
output2 (Dense)	(None, 64)	65600	['dense[0][0]']
output3 (Dense)	(None, 64)	65600	['dense[0][0]']

Total params: 15,436,800
Trainable params: 7,801,536
Non-trainable params: 7,635,264

Figure 4.3 Model architecture.

学習を始める前に全ての Q 値をランダムな小さい数に設定しておく。1 ステップごとにランダムな数 $R(0 \leq R < 1)$ を生成する。$\epsilon \leq R$ ならば今の状態の Q 値のうち最大のものに対応する行動を選択する。$\epsilon > R$ ならばランダムな行動を選択する。ϵ は次の式で求められる。この処理により学習初期は「色々やってみる」、学習後期は「良さそうな方策を中心に学習する」ことができる。

$$\epsilon = \frac{1}{episode + 1} \tag{4.1}$$

Q 学習に必要な行動、状態、報酬を以下のように定めた。

4.4.1 行動

エージェントはプレイヤーを操作する。よって行動数は上下左右へ移動する 4 となる。

4.4.2 状態

ゲームの全ての状態は前述したように約 26 万通りもあり、これがエージェントに渡される。しかしそのまま利用すると Q テーブルの大きさは状態数と行動数の積になるので 100 万以上になってしまう。Q テーブルが大きいほど学習に時間がかかってしまい、よく似た状態であっても個別に学習する必要がある。例として「目的地がプレイヤーの上のマスにある場合、上に移動すればスコアが増える」ことを学習するためには、プレイヤーの座標 (最上段を除く 56 通り) と敵の座標 (プレイヤーと同じ位置を除く 63 通り) の積である 3528 個の Q 値を更新しなければならない。そこで状態数を減らすため、状態を大まかなプレイヤーの位置・目的地の相対位置・敵の相対位置の組み合わせに変換する。また敵が次にいつ動くかの情報は与えない。

Figure 4.4 を場面 A、Figure 4.5 を場面 B と呼ぶことにする。場面 A ではプレイヤーは (4,4)、目的地は (7,7)、敵は (4,3) の座標にある。場面 B ではプレイヤーは (0,1)、目的地は (0,2)、敵は (3,2) の座標にある。

プレイヤーの位置はフィールドの端に接しているかによって 9 状態に分ける (Figure 4.6,Figure 4.7)。フィールドの角にあれば 0,2,6,8、端にあれば 1,3,5,7、それ以外ならば 4 になる。例として場面 A では状態 4、場面 B では状態 3 になる。

目的地の相対距離はプレイヤーから見た距離と方向によって 29 状態に分ける (Figure 4.8、Figure 4.9)。プレイヤーを中心とした 5×5 マスにある時は 2 マス左上から状態 0 として 2 マス右下までを状態 24 とする 25 の状態に分ける。それより遠くにある場

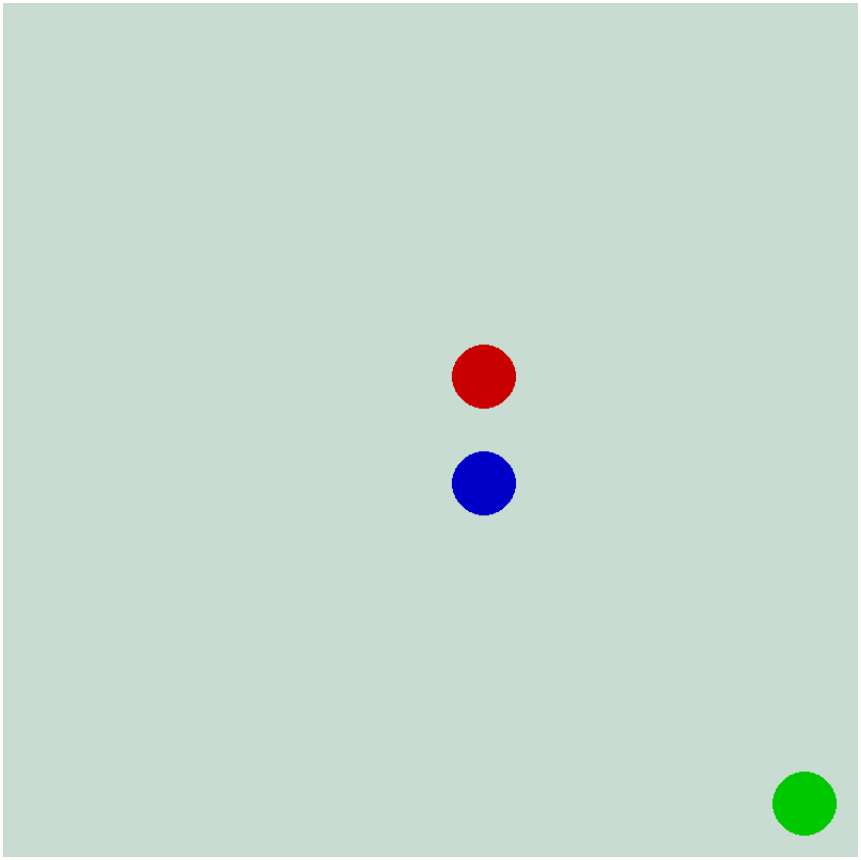


Figure 4.4　Scene A.

合は方向によって状態 26 から状態 28 とする。なお状態 12 はプレイヤーと目的地が同じマスにいるため本来ありえない状態だが、VGG16 が誤って判断することがあるため設定しておく。例として場面 A では状態 25、場面 B では状態 17 になる。エージェントからすれば 3 マス以上離れている場合はおおよその方向しか分からないが、2 マス以内ならば正確な相対位置が分かる。

敵の相対距離はプレイヤーから見た距離と方向によって 53 状態に分ける (Figure 4.10、Figure 4.11)。プレイヤーを中心とした 7×7 マスにある時は 3 マス左上から状態 0 として 3 マス右下までを状態 48 とする 49 の状態に分ける。それより遠くにある場合は方向によって状態 50 から状態 53 とする。なお状態 24 はプレイヤーと敵が同じマスにいるため本来ゲームオーバーした後の状態だが、VGG16 が誤って判断することがあるため設定しておく。例として場面 A では状態 17、場面 B では状態 34 になる。

エージェントからすれば 4 マス以上離れている場合はおおよその方向しか分からないが、3 マス以内ならば正確な相対位置が分かる。敵は目的地と異なりプレイヤーに向かって動くため 1 ステップに 2 マス近くなる場合がある。そのため目的地よりも正確に位置が分かる範囲を 1 マス広げた。

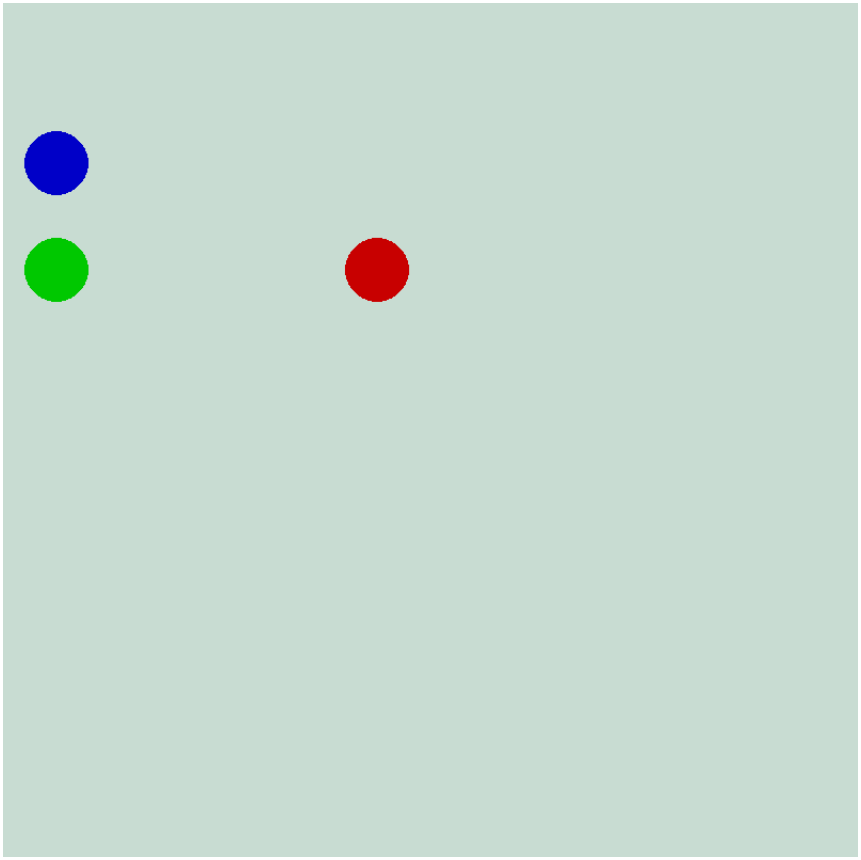


Figure 4.5　Scene B.

3 つの状態を組み合わせた変換後の状態数は $9 \times 29 \times 53 = 13833$ となった。これは元の状態数の 5.3%である。変換後の状態は以下の式で求められる。

(プレイヤーの状態) = (プレイヤーの位置)+(目的地の相対位置)×9+(敵の相対位置)×9×29 (4.2)

例として場面 A、場面 B の状態は以下のようになる。

$$(\text{場面 A の状態}) = 4 + 25 \times 9 + 17 \times 9 \times 29 = 4666 \tag{4.3}$$

$$(\text{場面 B の状態}) = 3 + 17 \times 9 + 34 \times 9 \times 29 = 9030 \tag{4.4}$$

4.4.3　報酬

1 ステップごとの報酬は Table 4.2 に従って設定した。目的地へ移動した際にプラスの報酬が得られる。これにより目的地に向って行くようになる。常時微少の罰を与えることで目的地へできるだけ早く到着しようとするようになる。敵と重なるとゲームオーバーになる為、重なったときに大きい罰を与えることで敵を避けるようになる。200 ステップ経過したときも大きい罰を与え、ループを避けるようにする。フィールド外へ移動しようとした場合はその場から動けないため、追加で微少な罰を与える。これにより動かな

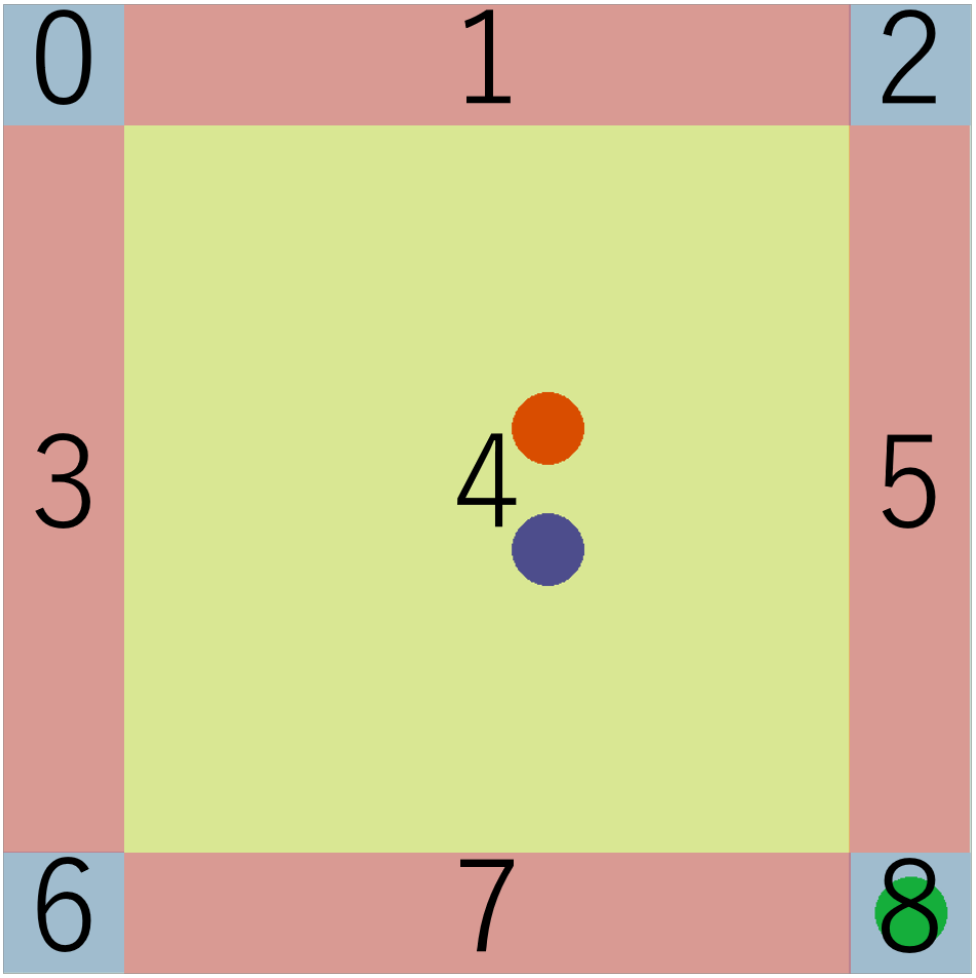


Figure 4.6　Scene A player state.

い行動を避けるようになる。

4.4.4　結果

ゲームをリセットしてからゲームクリアもしくはゲームオーバーするまでを 1 エピソードとして 2100 エピソード学習した。1500 エピソードを超えるとクリアする確率がほとんど上がらなくなっていた。学習過程の合計報酬とその平均値、中央値の遷移を Figure 4.12、スコアとその平均値、中央値の遷移を Figure 4.13 に示す。

学習終盤の 300 エピソードにおけるスコアとエピソード数を Table 4.3 にまとめた。271 エピソードでゲームクリアした。また最長 36 回連続でゲームクリアした。

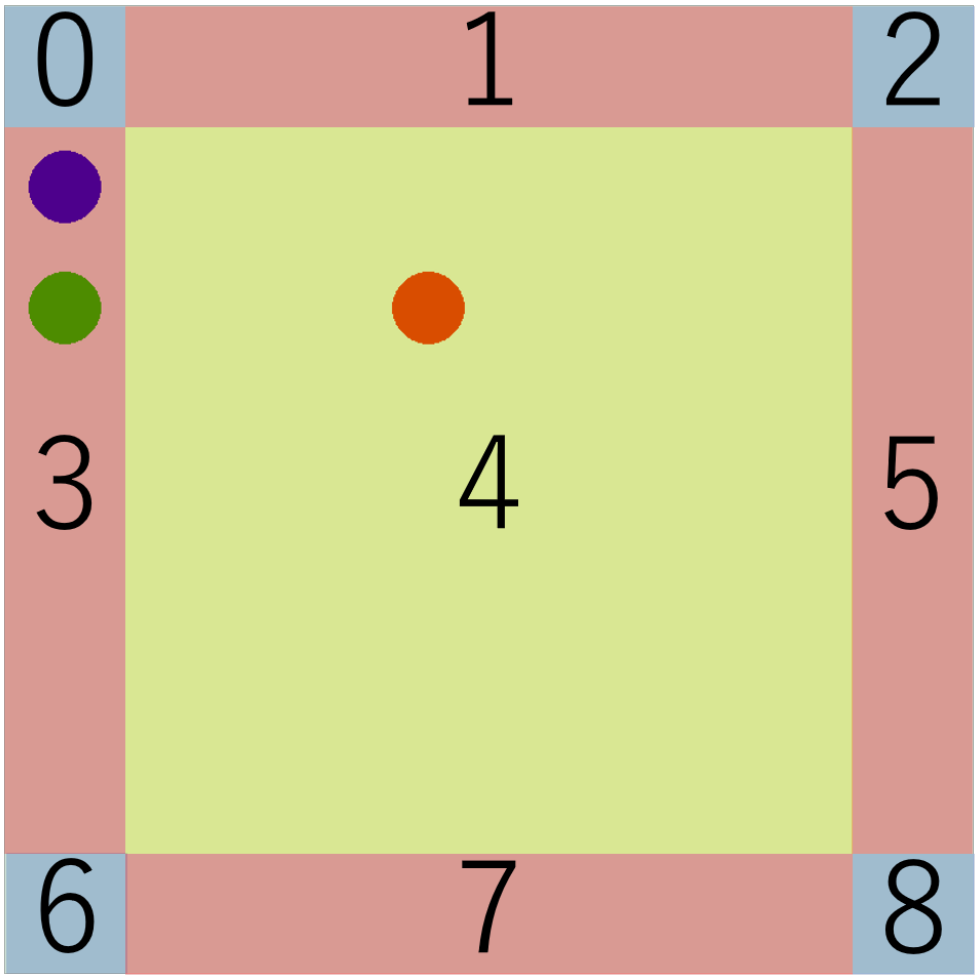


Figure 4.7 Scene B player state.

Table 4.2 How to reward.

Conditions	Reward
all the time	−1
move to target	+100
overlap with the enemy	−100
200 steps passed	−100
try to move out of the field	−1

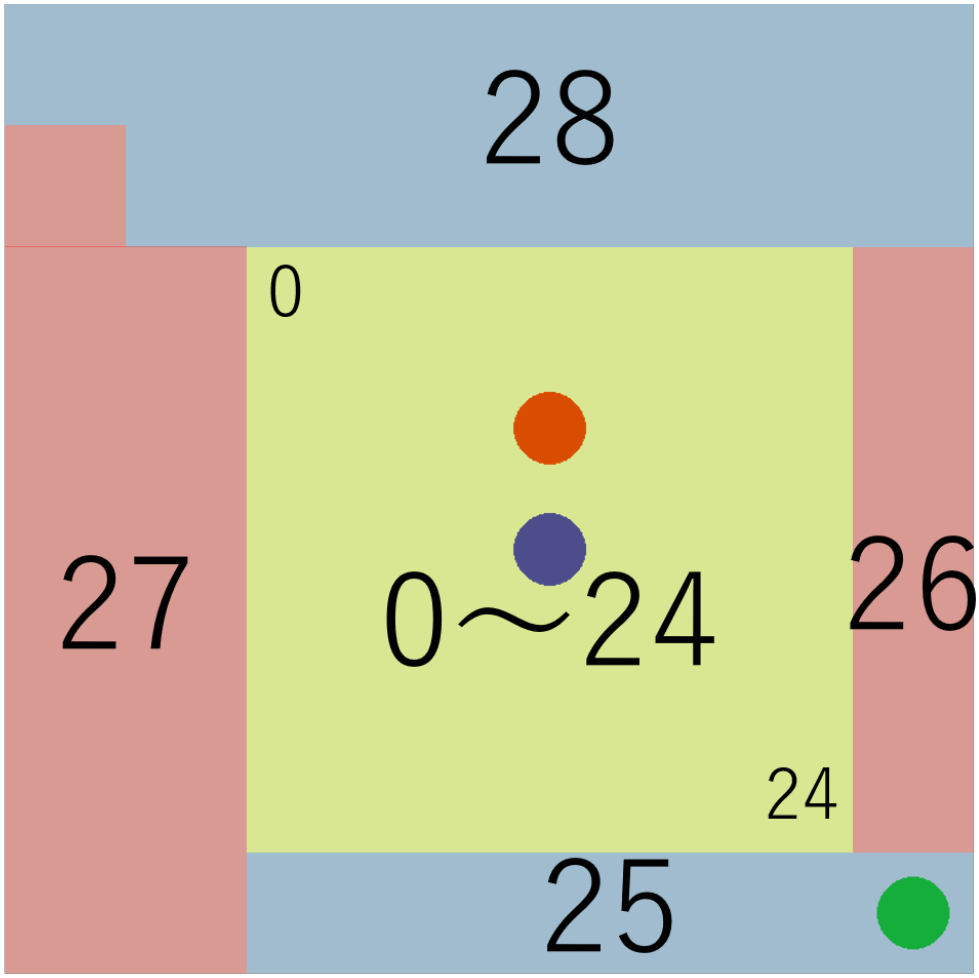


Figure 4.8 Scene A target state.

Table 4.3 Final scores.

Scores	Times
0	1
1	11
2	6
3	4
4	7
5	271

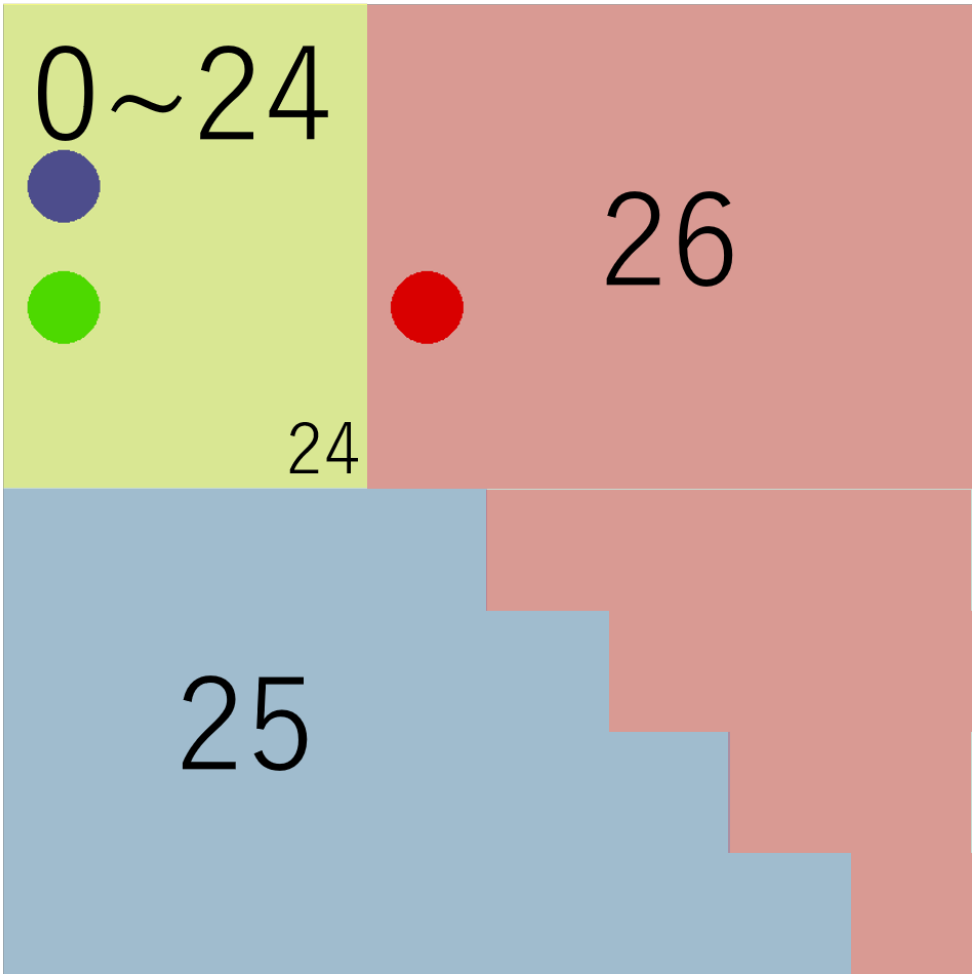


Figure 4.9　Scene B target state.

4.4.5　状態の変換をしなかった場合との比較

4.4.2 項に示した状態の変換をしなかった場合と、前項の結果で学習の様子を比較する。前項の結果を学習結果 A、状態の変換をしなかった場合の結果を学習結果 B とする。それぞれの学習過程の合計報酬の平均値の遷移を Figure 4.14、スコアの平均値の遷移を Figure 4.15 に示す。学習結果 B は合計報酬もスコアもほとんど増えていない。このまま学習を続けていっても十分に学習できるまでには膨大な時間がかかると考えられる。学習結果 A と B の違いから状態を変換し、状態数を削減することの効果が確かめられた。

4.4.6　報酬の与え方が異なる場合との比較

報酬の与え方が異なる場合と、学習結果 A で学習の様子を比較する。報酬の与え方を「目的地へ移動したときに大きいプラスの報酬を与える」、「敵と重なったときに大きい罰を与える」、の 2 つだけとする。この場合の結果を学習結果 C とする。それぞれの学習

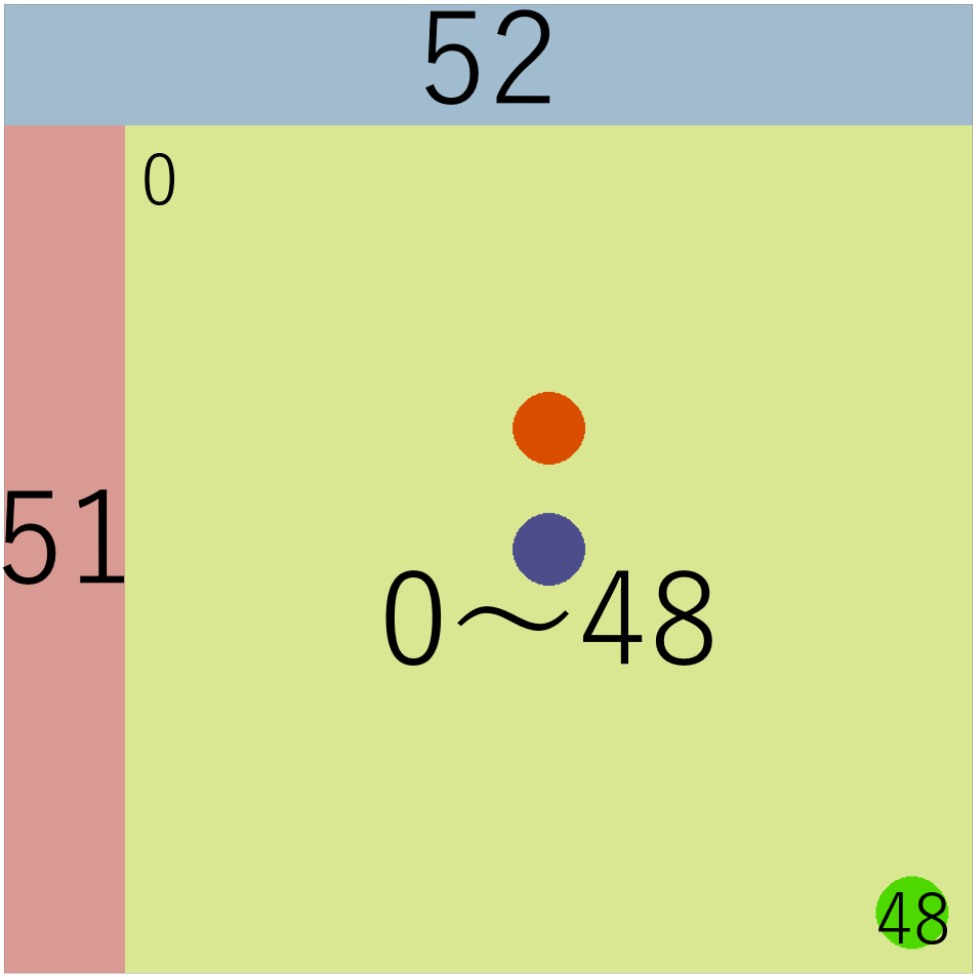


Figure 4.10　Scene A enemy state.

過程の合計報酬の平均値の遷移を Figure 4.16、スコアの平均値の遷移を Figure 4.17 に示す。エピソードが経過するに連れて学習結果 C の合計報酬は増加しているが、スコアはあまり増加していない。よって学習結果 C の報酬の与え方はこの問題には適していないということが分かる。学習結果 C ではほとんどのエピソードで 200 ステップが経過するようになった。動きを観察すると目的地へ向かおうとする傾向はあるが、学習結果 A よりはあまり積極的に近づこうとはしていないように見えた。同じ場所をループする挙動も見られた。一方で敵を避けることは学習結果 A よりも上達していた。学習結果 A との差異は敵に重なること以外に罰を与えられることがないからだと考えられる。常時微少な罰を与えることは報酬に繋がらない行動の価値を下げ、学習過程において価値の下がった行動は選択される可能性が下がる。しかし学習結果 C では微少な罰を受けないためエージェントが「敵に重ならなければそれでいい」と判断していると考えられる。

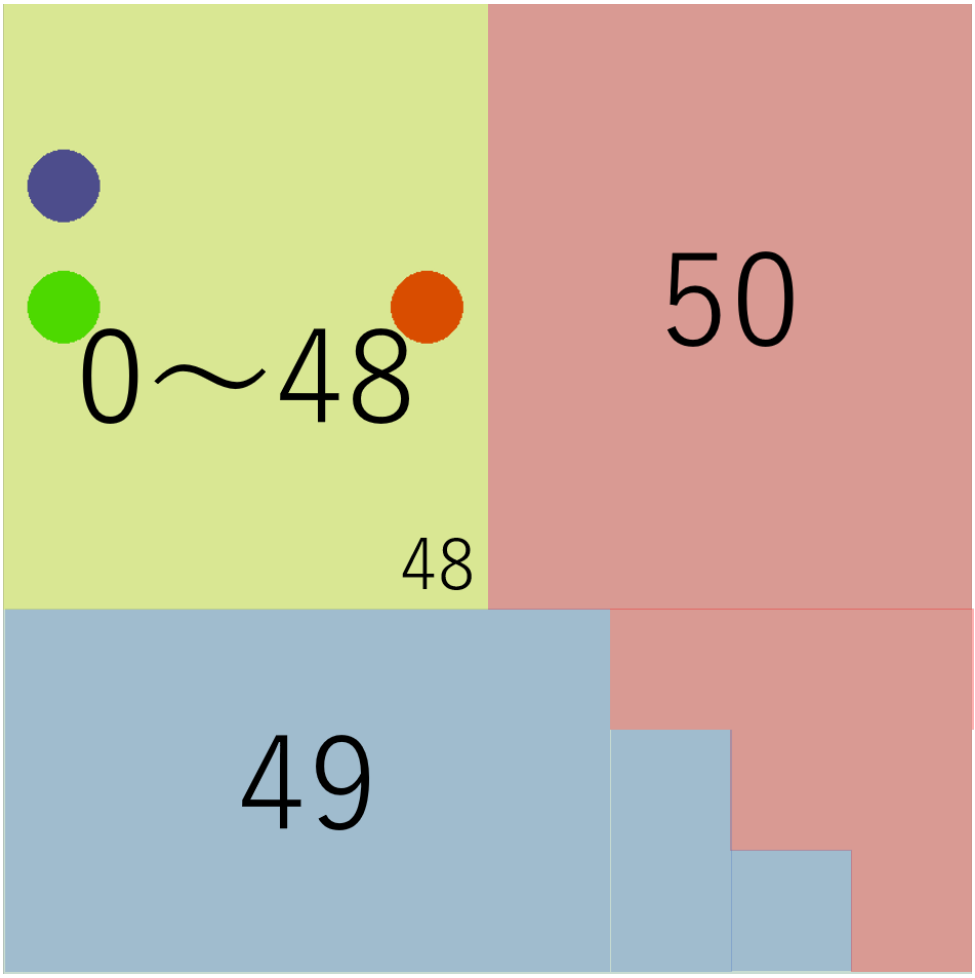


Figure 4.11　Scene B enemy state.

Table 4.4　Resulting scores.

Scores	Times
0	2
1	4
2	5
3	1
4	4
5	84

4.5　最終結果

学習の完了した VGG16 と Q テーブルを使用してエージェントに 100 回ゲームをプレイさせた。行動は上下左右のうち最も高い Q 値を選択させた。結果を Table 4.4 にまとめた。84 回ゲームクリアした。最長 14 回連続でクリアした。

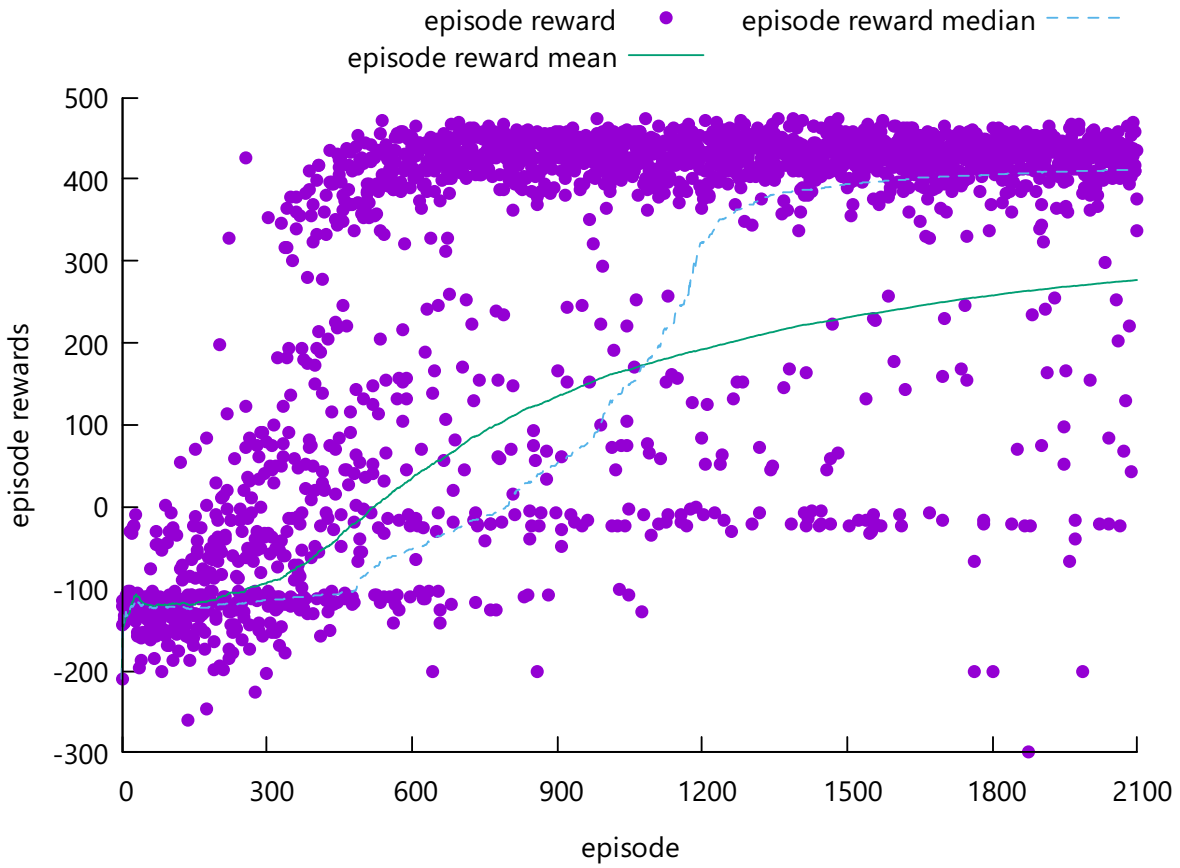


Figure 4.12　Q-learning episode reward.

ゲームオーバーとなった 16 回の原因を Table 4.5 にまとめた。最多となった原因は、直前にプレイヤーと敵の距離が 2 マスのときに互いに近づいた結果、同じマスに移動しゲームオーバーになったことであり、8 回起こった。次に多かった原因は VGG16 がプレイヤーや敵の座標を誤判断した結果ゲームオーバーになったことであり、5 回起こった。その他は敵が隣にいる時にフィールド外側に移動しようとして敵に追いつかれたこと、敵と目的地が同じ座標にいるにも関わらず目的地に移動したこと、単純に敵へ移動したことがそれぞれ 1 回ずつ起こった。200 ステップ経過してゲームオーバーになることは一度も起こらなかった。

敵との距離が 2 マスのときに互いに近づいた結果起こるゲームオーバーが取り除けなかった理由を考察する。今回のゲームでは敵は 3 回プレイヤーが移動するたびに 1 回しか移動することはない。そのため同じ状態であってもゲームオーバーになる場合とならない場合が発生する。距離が 2 マスとなる状態は多数あり、そのうちいくつかの状態が Q 学習の過程でゲームオーバーになる場合を経験しなかったと考えられる。

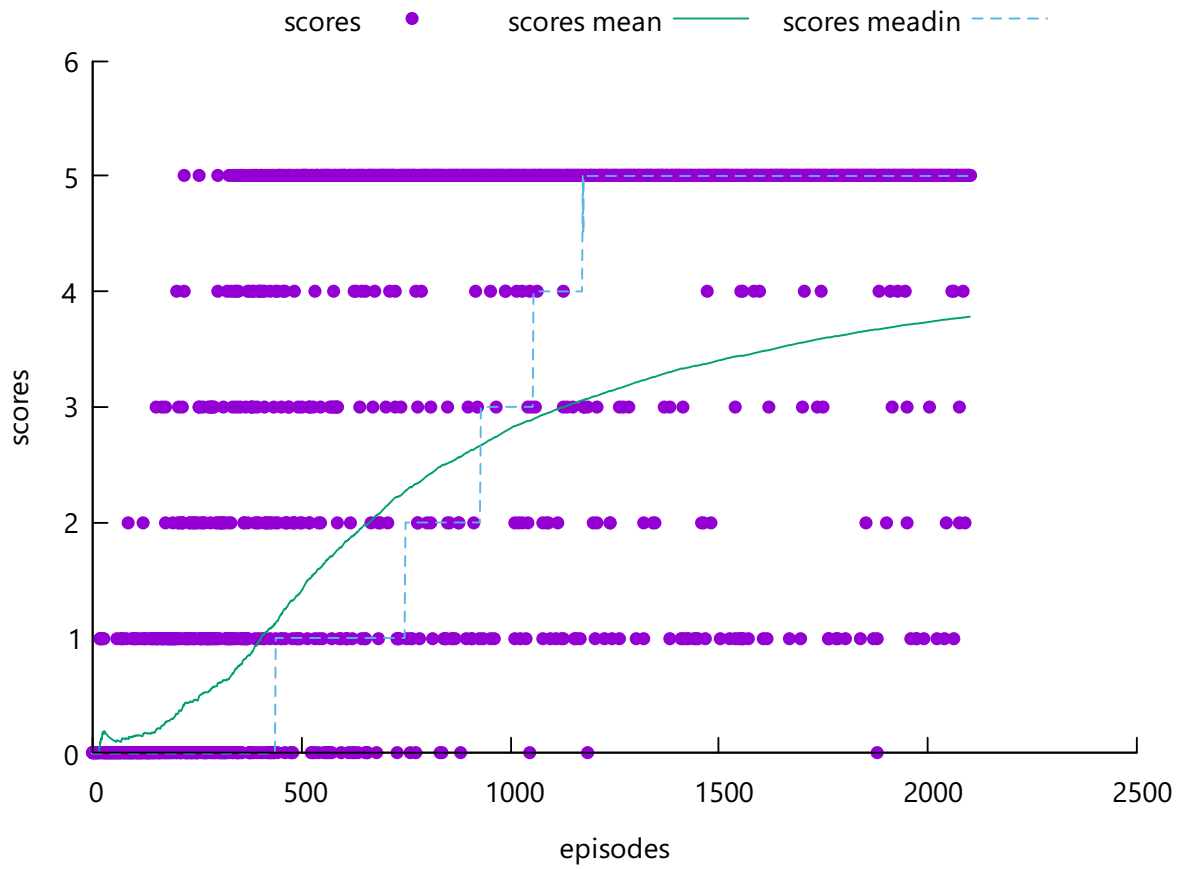


Figure 4.13 Q-learning score.

Table 4.5 Failure causes.

Cause	Times
distance 2 squares	8
Misjudgment of VGG16	5
tried to move outside	1
Enemy and target overlapped	1
moved to enemy	1
200 steps passed	0

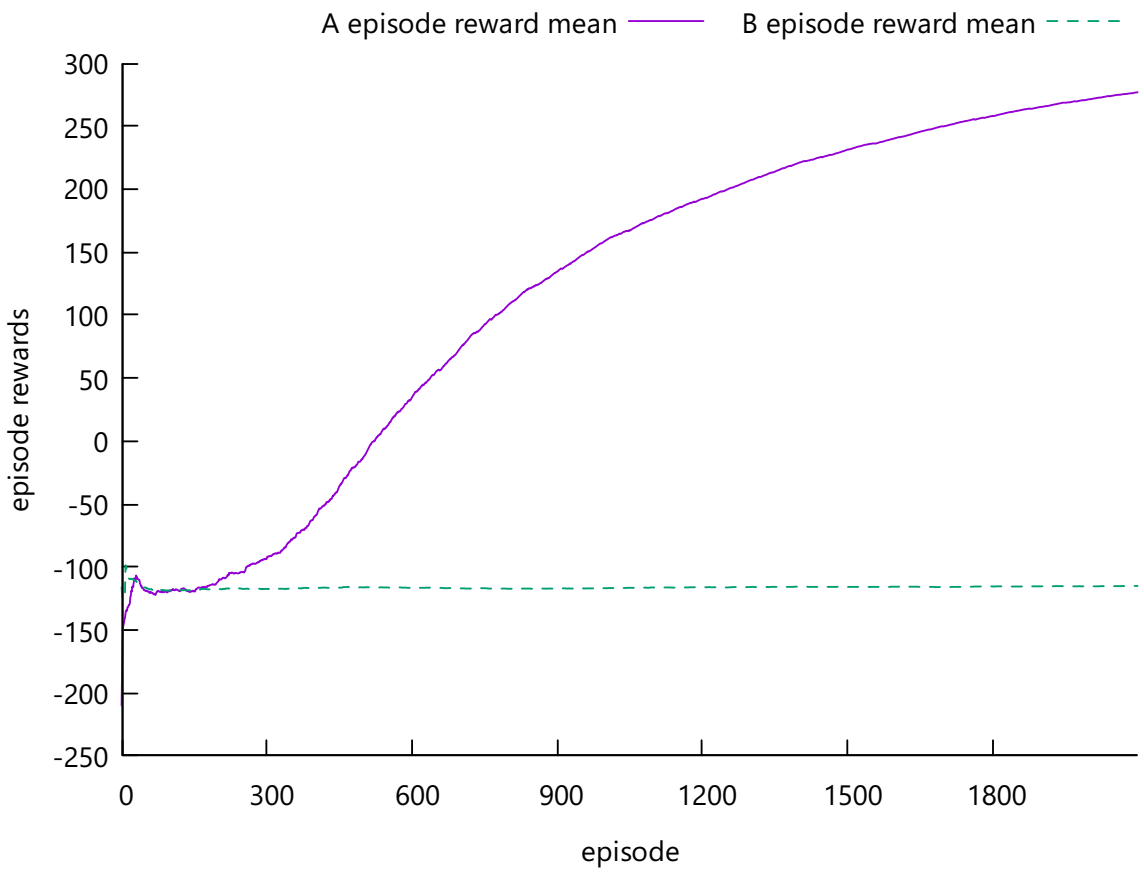


Figure 4.14 Episode reward mean comparison.

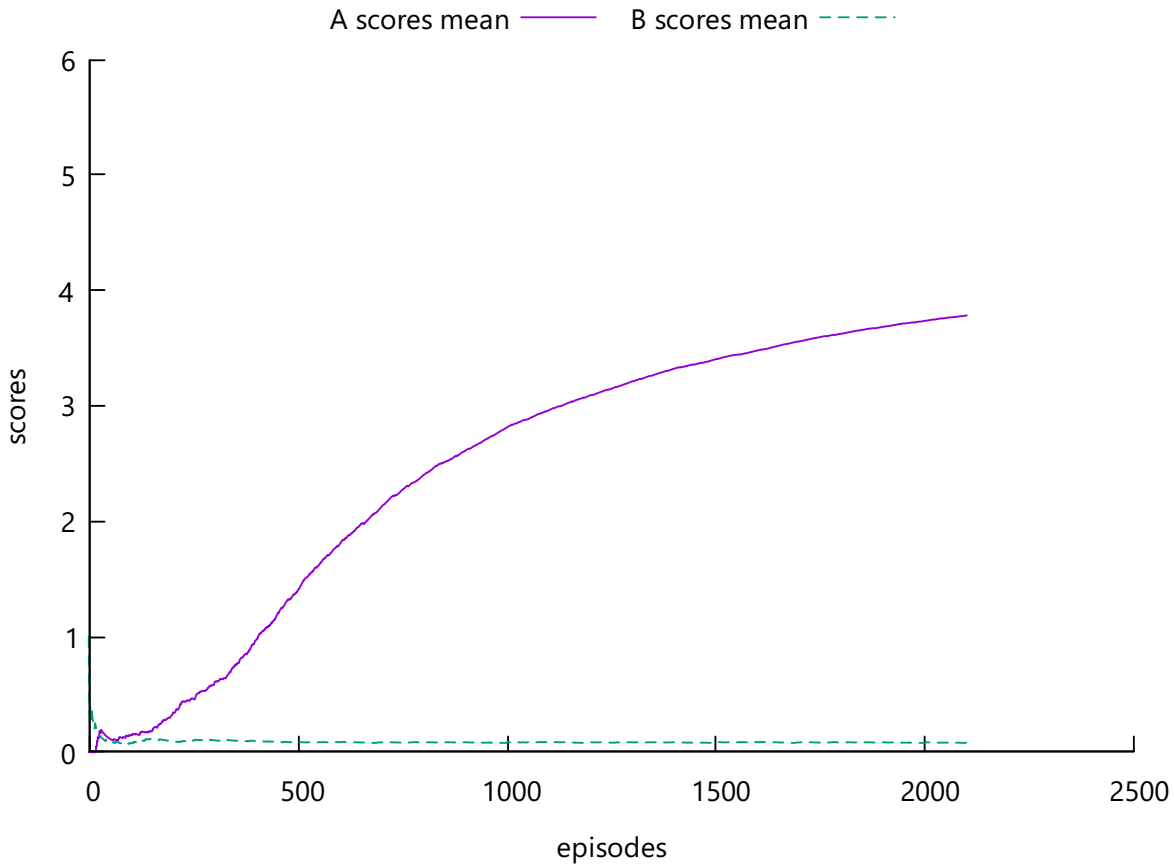


Figure 4.15 score mean comparison.

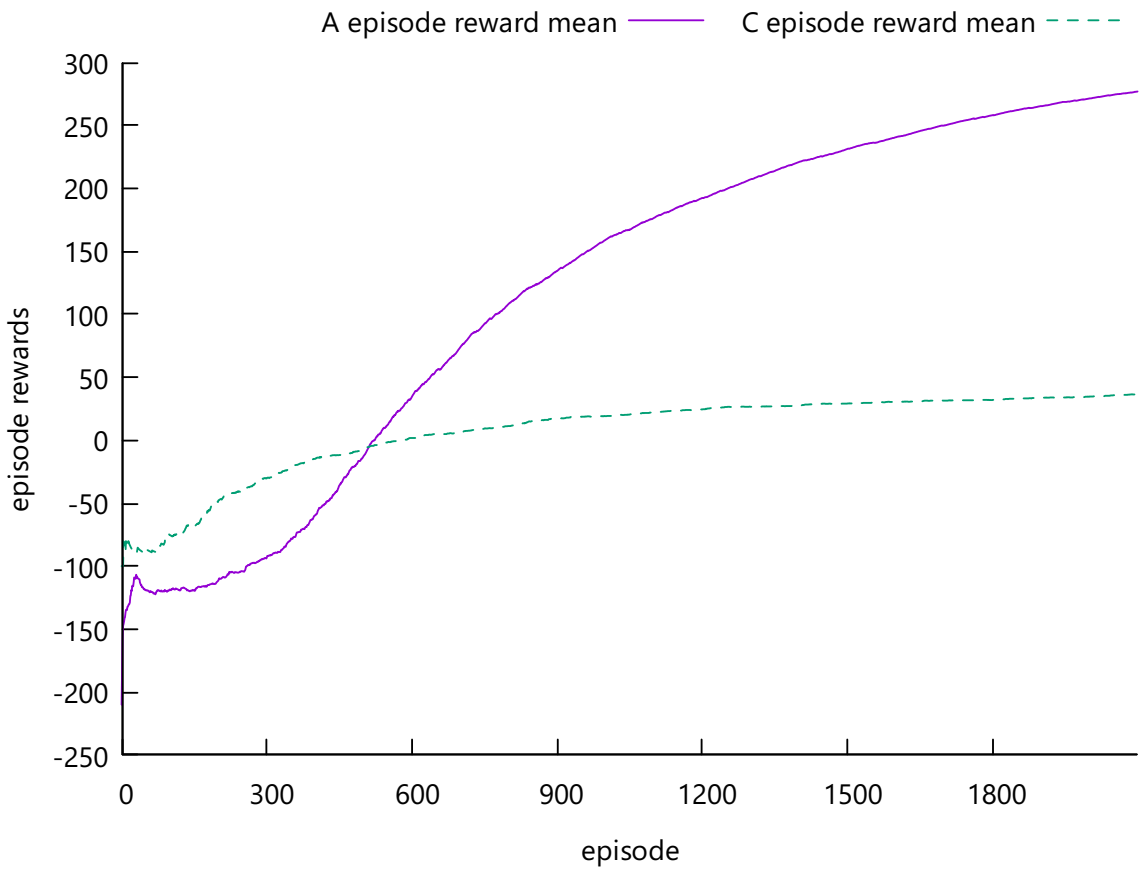


Figure 4.16 Episode reward mean comparison.

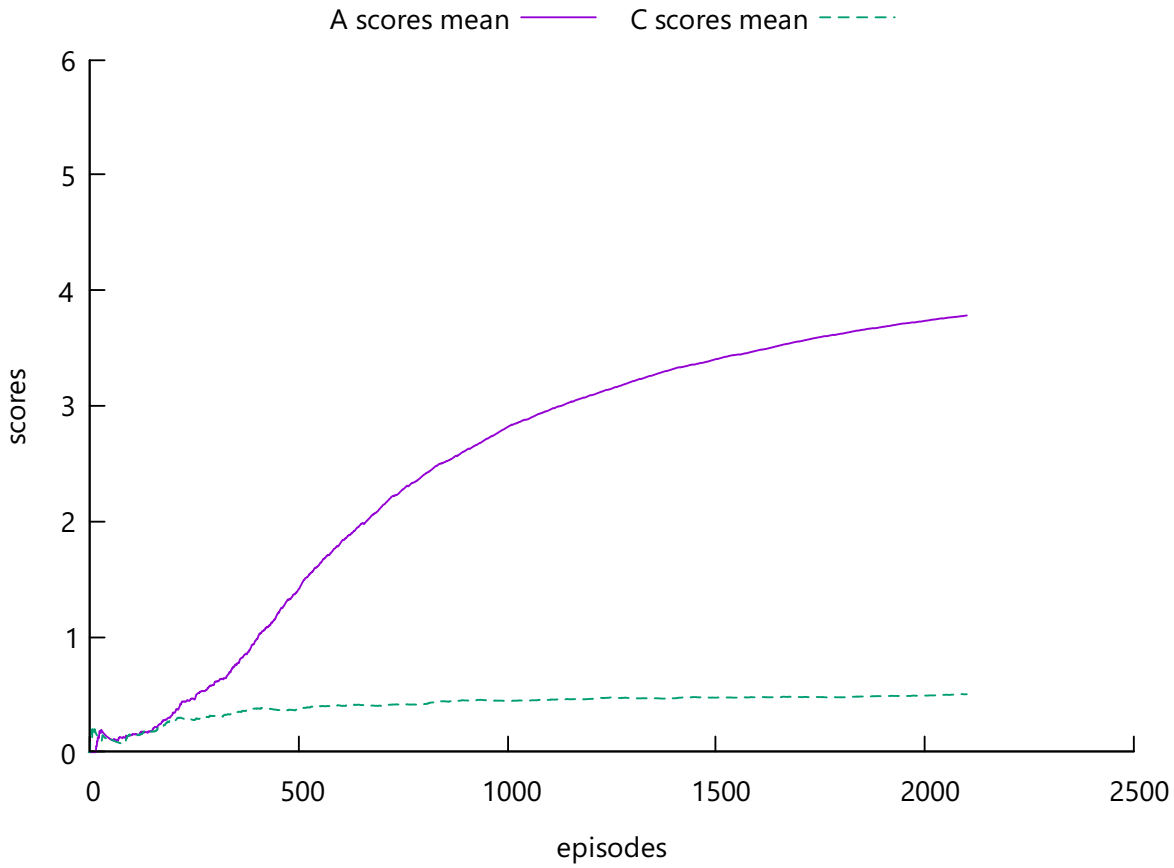


Figure 4.17 score mean comparison.

第5章　結論

本研究の目標は単純なゲームを画像だけでクリアするAIを制作することであった。結果として概ねゲームクリアすることができたが100％には届かなかった。ゲームオーバーとなる原因はQ学習とVGG16の両方から生まれていた。

興味深い挙動としてプレイヤーがフィールドの角にいて敵が斜めに接しているとき(例:プレイヤーの座標(0,0)、敵の座標(1,1))、プレイヤーはフィールド外に移動する行動を選択しその場に留まった。そして敵が隣に来ると角から離れていった。これは敵が次にいつ動くか情報を与えないことによって発生した挙動であると考えられる。学習過程でプレイヤーをその場に留まると微少な罰を与えていたが、敵が動くのを待つ方が多くの報酬を得られることを学習しているようだ。人間の操作ではこのような判断にはならないだろう。AI特有の判断と思われるため興味深い。

Q学習では状態の変換方法や報酬の与え方を変えると学習の様子が変わった。その中でもクリアする確率が高くなり、学習が早いものを最終結果に使用した。報酬や罰を与える条件によっては目的地を無視して敵から離れるだけになったり、狭い範囲を往復するだけになったりした。敵が次にいつ動くか情報を与えてみても良かったかもしれない。

VGGの転移学習では約1万のデータから26万以上に分類でき高い正解率が出た。新たな全結合層を作ることで異なる分類に対応できるため、非常に汎用性が高く今後も使っていきたい。

今後の展望として、深層強化学習を実験したい。そうすればより複雑な環境でも満足のいく結果が得られると期待している。

参考文献

1) Ledge.ai . 強化学習とは — 機械学習との違い・深層強化学習・活用事例やその未来まで徹底解説
https://ledge.ai/reinforcement-learning/

2) 価値関数　ベルマン方程式
http://arduinopid.web.fc2.com/N83.html

3) MathWorks. 畳み込みニューラルネットワークとは？これだけは知っておきたい 3 つのこと
https://jp.mathworks.com/discovery/convolutional-neural-network-matlab.html

4) 日下部完：CNN を利用した柔道競技におけるポイントの判定
https://www.gifu-nct.ac.jp/elec/deguchi/sotsuron/kusakabe.pdf

5) AIZINE . 強化学習の手法の一つ Q 学習とは？要点と基本を簡単理解しよう
https://aizine.ai/glossary-q-learning/

6) AIZINE . 強化学習を勉強するなら必須！「Q 学習」の基礎〜実装まで完全ガイド
https://aizine.ai/q-learning0920/

7) 強化学習の基礎を学んで Cart Pole で遊んでみた（単純な Q Learning を実装）
https://qiita.com/Fumio-eisan/items/e59996b144d8cbb35efa

8) Qiita .VGG16 を転移学習させて「まどか☆マギカ」のキャラを見分ける
https://qiita.com/God_KonaBanana/items/2cf829172087d2423f58

9) MathWorks .vgg16
https://jp.mathworks.com/help/deeplearning/ref/vgg16.html

10) ICHI.PRO .VGG16 とは何ですか？—VGG16 の概要
https://ichi.pro/vgg-16-to-wa-nani-desu-ka-vgg-16-no-gaiyo-267001881294357

11) 少ない画像から画像分類を学習させる方法（keras で転移学習：fine tuning)
https://spjai.com/keras-fine-tuning/

謝辞

本研究を進めるにあたり、御多忙中にも関わらず多大なご指導を賜りました出口利憲先生に深く感謝すると共に、同研究室において共に勉学に励んだ西川司優氏、武藤昇吾氏、森駿氏に厚く御礼を申し上げます。