卒業論文

強化学習を利用したAIディレクターの制作

甲南大学　知能情報学部
知能情報学科
11771083
濱石 海地

2021年11月2日
指導教授　田中雅博

目 次

1　はじめに

ゲームにおいて難易度は, それ一つでゲームの面白さを左右する重要な要素であり, ゲーム開発において最も工数をかけるべき箇所のひとつである。一般的にはゲームの進行と共にプレイヤーの腕が向上するため, ゲームが進行するほど難易度も上がるよう調整される場合がほとんどである。しかし, プレイヤーの腕や, 面白く感じる難易度は人それぞれであり, またその調整は試行錯誤を繰り返すなどして調整する必要があり, 工数の多さはゲーム開発者にとって大きな負担となる。それを解決するべく, 本研究では「AI ディレクター」に着目した。AI ディレクターとは, Valve Software 社開発のシューティングゲーム「Left 4 Dead」に使用されている難易度調整システムの呼称であり, それはプレイヤー達の体力や所持品, 敵との戦い方からプレイヤーの状態を推定し, 敵の配置などを調整している [5, 6]。呼称や手法は異なるものの, プレイヤーの状況によって難易度を秘密裏に調整するシステムは「バイオハザード 4」など他のゲームにも見られる [7, 8]。これら「AI ディレクター」は, 多くはルールベースによって作られていると考えられる。本研究はこれを機械学習を用いて作成することで, ユーザーそれぞれに最適な難易度を提供すると共に, 開発者の負担を軽減することの実現可能性を探るものである。

2　研究の概要

本研究のために作成されたオリジナルのゲームにおいて, 登場する敵やアイテムの出現を強化学習 AI に制御させる。

2.1　ゲームの仕様

Python のコマンドライン上で動作する, 簡素な RPG を作成した。

2.1.1　ゲーム全体の流れ

以下の流れを繰り返すことで進行する。これを所定の回数 (10 回あるいは 20 回) 繰り返した後, プレイヤーが生存しているならゲームクリアとする。

1. ディレクター (強化学習 AI) によって 2 通りの行先が提示されるので, プレイヤーはそれを選ぶ。行先に何があるかは表示されている。
2. 選んだ行先に敵がいるなら, その敵と戦闘を行う。アイテムがあるなら, それを獲得する。

図 1 を参照のこと。

2.1.2　戦闘の流れ

戦闘ではプレイヤーと敵が, どちらかの生命力が 0 以下になるまで, 「プレイヤー, 敵, プレイヤー, 敵…」の順番で交互に行動する。戦闘における行動は, 表 1 に示すもののうち 1 つを選んで行う。プレイヤーの行動はコンソールからの手動入力, あるいはプログラムによる自動入力で選ばれる。敵の行動は, 敵自身が持つ数値が示す確率に応じてランダムに選ばれる (2.2 を参照)。

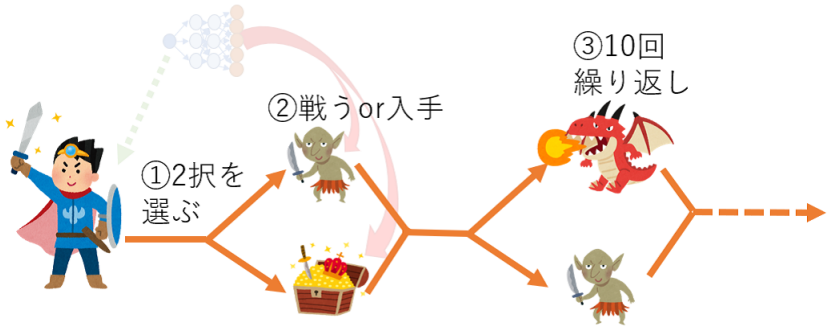

図 1: ゲームを遊ぶ流れのイメージ図

表 1: 戦闘における行動

攻撃	相手に自身の攻撃力と同値のダメージを与える (生命力を減らす)。
防御	相手から受けるダメージを, 自身の防御力の値だけダメージを減少させる。
薬草を使用	薬草所持数を 1 消費し, 自身の生命力を全回復。
爆弾を使用	爆弾所持数を 1 消費し, 攻撃力 2 倍の攻撃を行う。

2.1.3 プレイヤーおよび敵のステータス (能力値)

プレイヤーおよび敵は, 表 2 に示す値をステータスとして持ち, それらは戦闘中の行動に影響する。プレイヤーは 6 つ (生命力, 最大生命力, 攻撃力, 防御力, 薬草所持数, 爆弾所持数) の, 敵は 4 つ (生命力, 最大生命力, 攻撃力, 防御力) の値を持つ。

2.2 ソースコードの構造および各クラスの仕様

ゲームを遊ぶプレイヤーと, ゲームに登場する敵・アイテムを操作するディレクターは, 該当部分のクラスを差し替えることで, 他のゲーム部分を編集することなく作り変えることが可能である。

- main クラス
  学習を始める際に初めに実行されるクラスである。学習のためにゲームを複数回実行する処理や, ゲームの結果をディレクターに渡す処理を持つ。

- game クラス
  関数としてゲームを実行し, 返り値として最終的なゲームの進行度を main クラスに渡す。ゲーム中はディレクターが提示した行先の選択肢をプレイヤーに渡す処理, プレイヤーと敵を戦わせる処理を行う。

表 2: プレイヤーおよび敵のステータス (能力値)

生命力	これが 0 になったら死亡。アイテム等により回復する。
最大生命力	生命力はこの値を超えて回復しない。アイテム拾得により増加。
攻撃力	戦闘において敵に与えるダメージ量に影響。 戦闘に勝利するか, 特定のアイテム拾得により増加。
防御力	戦闘において防御したとき, この値だけ受けるダメージ量が減る。 戦闘に勝利するか, 特定のアイテム拾得により増加。
薬草所持数	プレイヤー専用。戦闘中に消費して生命力を回復できる。 アイテム拾得により増加。
爆弾所持数	プレイヤー専用。戦闘中に消費して攻撃力 2 倍の攻撃ができる。 アイテム拾得により増加。

- character クラス
  player クラスおよび enemy クラスの親クラスであり, インタフェースとして機能する。表 2 に示す値のうち, プレイヤーと敵に共通する 4 つの値を持つ。また戦闘を行うにあたって必要な, 戦闘における行動を選ぶ関数や, ダメージによって生命力を減少させる関数を持つ。

- enemy クラス
  ゲーム中における敵。コンストラクタで設定可能な値として行動傾向を示す実数値リストを持ち, 戦闘中における各種行動 (表 1) のうち, それを選ぶ確率を示す。

- player クラス
  ゲーム中におけるプレイヤー (主人公)。character クラスに加えて, ディレクターの提示した選択肢を選ぶ関数と, 薬草所持数と爆弾所持数を示す 2 つの整数値を持つ。標準では, 選択肢と戦闘中の行動はコマンドライン上で手動指定する。

- ルールベース自動プレイヤークラス
  player クラスの子クラス。コマンドラインによる手動選択の代わりに, 条件分岐により自動的に選択を行う。学習中はこちらを使用する。

- treasure クラス
  ゲーム中でプレイヤーが入手するアイテム。プレイヤーの各ステータス値をいくら増減させるかの値を持つ。

- director クラス
  ディレクター。ディレクションとして選択肢を返す関数, 学習を行うためにゲームの結果を入力する返り値なしの関数の 2 つの関数を持つ。本研究における各手法は, このクラスの子クラスとして作成する。

これらクラスの関係性については、図 2 に示すクラス図を参照のこと。

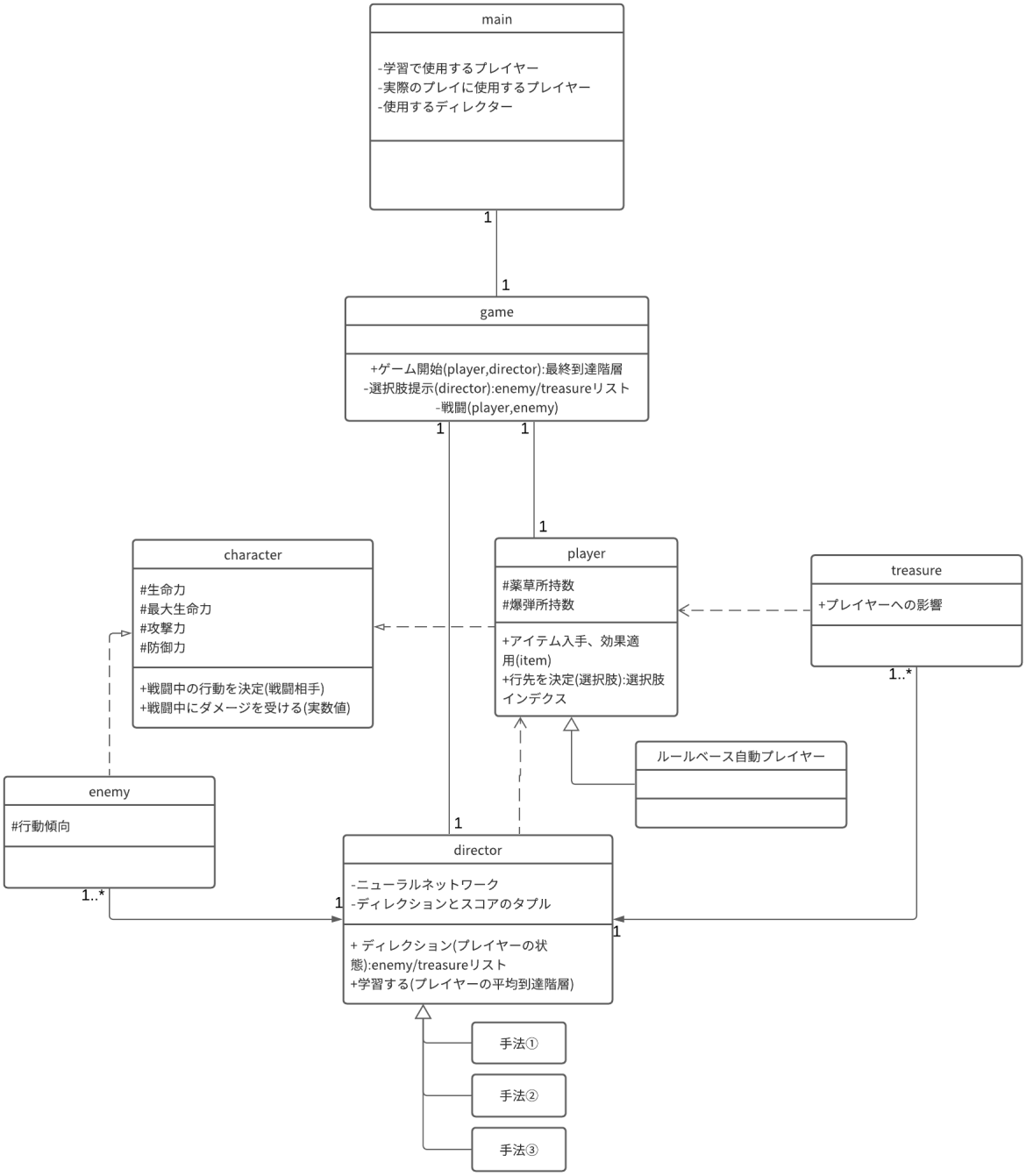

図 2: 本研究に使用したプログラムのクラス図

3　学習の方法と結果

以上を踏まえ, ディレクターを学習させることでゲームの難易度を最適なものとすることを試みる。本研究における最適な難易度とは, ルールベースで自動的にプレイするするプレイヤーが, ゲームを 7 割進行させた所でちょうどゲームオーバーとなるような難易度としている。

3.1　各手法に共通する流れ

ディレクターに適切な難易度を学習させるためには, 難易度を測る指標が必要となる。本研究ではディレクターの調整した難易度を測るために, 自動でゲームを複数回プレイし, それらのプレイでどこまで進めることができたか (以降, 最終到達階層と記す) の平均を求め, これを難易度の指標としている。図 3 に, この流れのイメージ図を表している。ディレクタークラスには 2 つのグローバル関数があり, 一つがディレクション

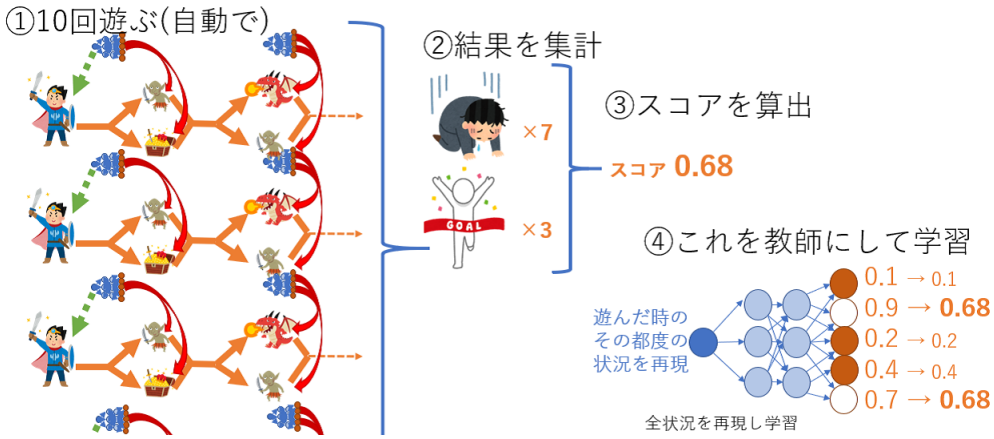

図 3: 学習全体の流れ。図中の数値は研究初期の一例

ンを返す関数, もう一つが学習を行わせる関数である。

ディレクションを返す関数には, 引数としてプレイヤーの各種能力, 出現する敵やアイテムのリストを渡す。返り値として出現させるべき敵およびアイテムのリストが渡される。内部ではプレイヤーの能力値を入力値に, 出現する敵やアイテムの価値を出力とするニューラルネットワークの前向き計算が行われる。それと同時にディレクションをクラス内に保存している。このニューラルネットワークの入出力値について, 図 4 に表している。

学習を行わせる関数には, 引数としてゲームの平均到達階層を渡す。返り値は無い。引数を基にディレクタークラス側で保存されていたディレクションのスコアを計算し, このスコアを基にニューラルネットワークの誤差逆伝播による最適化を行う。

3.1.1　教師データの形式

ゲーム中のディレクターは, ニューラルネットワークに対する入力値とその出力値を記憶する。スコアが算出された際には, 出力値のうち実際にゲームで提示した選択肢に対応するものをスコアに置き換え, これを教師データにして学習を行う。

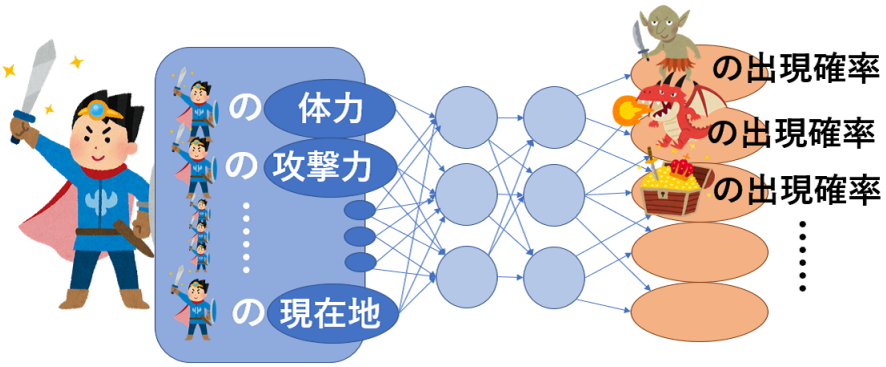

図 4: ディレクターの仕組み

3.2　一般的な ε-グリーディ法を使用した方法

まず最初に一般的な強化学習の手法に見られる [1, 2, 3, 4] ような, 出力値のうち最も高い値 2 つを選択肢として提示する方法をとった。

3.2.1　手法の手順

1. 自動でゲームを 10 回遊ぶ (ルールベース AI による)。
2. 1. において, AI ディレクターの出力 (以下, ディレクションとする) と, 最終到達階層を記録し, それらを基にスコアを算出する。
3. 1. と 2. を 10 回行い, データを溜める (最大で 2000 データ, 10 プレイ*最大 20 階*10 回)。
4. 3. を教師データとしてディレクターを学習させる。
5. 学習が終ったデータを削除し, 1. に戻る。

3.2.2　ゲーム中におけるディレクターの構造

入力値 7 個, 出力値 17 個の, Chainer を利用したニューラルネットワーク。入力値は, プレイヤーのステータス (表 2 を参照) の値 6 つに, 現在のプレイヤー位置の値を合わせた 7 個がある。出力値は, 出現する全ての敵 11 種類と, 出現する全てのアイテム 6 種類, それぞれの価値を決める実数値の 17 個がある。それらのステータスについては表 3 を参照。出力値のうち, 最も大きい値 2 つが示している敵またはアイテムがゲームに出現する。

3.2.3　スコアの算出方法

プレイヤーが死亡するまでに到達したゲームの進行度 10 回分の平均と, 目標値の差をスコアとする。この手法では目標の値を 14 としている (全 20 階のうち 14 階目の選択肢でちょうど力尽きる難易度に調整するため)。このときスコアは-14 から 0 の範囲となり, 0 が最高である。例えば, 力尽きた階層の平均が 10 階だった場合, スコアは-4 である。

3.2.4　Fixed Target Q-Network

ネットワークを毎回更新すると, いつまでも収束しないという事態が発生する可能性がある。そのため, 学習はスコアの取得 10 回毎にのみ行う。それまで教師データは保存される。

3.2.5　ε に基づくランダムなアクション

ε-グリーディ法に倣い, 学習初期は学習したものではなく乱数を出力に用いる。様々な出力を試行することで, より高いスコアの出力を発見する狙いがある。このランダムな試行は ε の確率で行われる。 ε の初期値は 1.0(100%), 学習が行われる (スコア取得 10 回) 毎に 0.05(5%) 減算される。

3.2.6　結果

提示される選択肢はゲームを通して同じものばかりになり, それに伴い学習に応じて結果が極端に上下するようになった。ニューラルネットワークでは, 入力層への値ひとつが多少変化したところで出力が劇的に変化するような学習は困難であるため [9], ゲームを通して同じ選択肢ばかりになったものと考察される。ゲームの結果とスコアの遷移は図 5 の通り。

表 3: ゲーム中に出現する敵およびアイテムのリスト

敵			
名称	生命力	攻撃力	防御力
スライム Lv1	4	1	1
スライム Lv2	5	1	1
スライム Lv3	5	1	2
スライム Lv4	6	1	2
スライム Lv5	7	2	2
スライム Lv6	8	2	2
スライム Lv7	9	3	3
スライム Lv8	10	3	3
スライム Lv9	11	3	4
スライム Lv10	12	3	4
ゴブリン Lv1	3	2	1
ゴブリン Lv2	4	3	1
ゴブリン Lv3	5	4	1
ゴブリン Lv4	6	5	1
ゴブリン Lv5	7	6	2
ゴブリン Lv6	8	7	2
ゴブリン Lv7	9	8	2
ゴブリン Lv8	10	9	2
ゴブリン Lv9	11	10	3
ゴブリン Lv10	12	11	3
ドラゴン	50	8	3

アイテム	
名称	効果
新しい鎧	最大生命力を 5 増加
新しい剣	攻撃力を 2 増加
新しい盾	防御力を 2 増加
爆弾	爆弾所持数を 1 増加
薬草	薬草所持数を 1 増加
宿屋	生命力を 10 回復

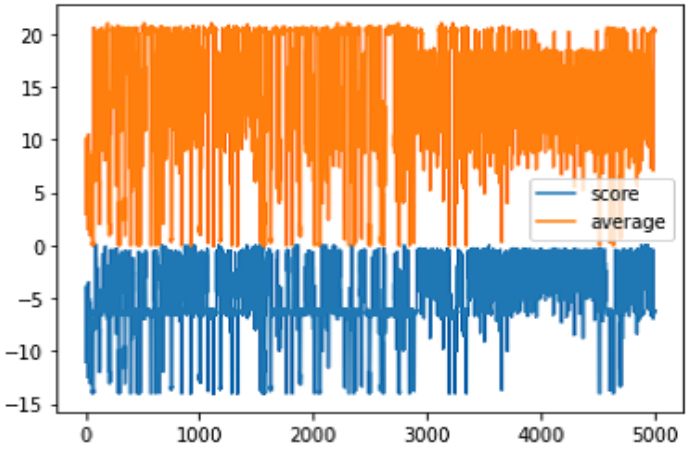

図 5: 一般的な ε-グリーディ法を使用した方法における平均到達階層とスコア

3.3 重み付きランダムを採用した方法

先の方法で取った手法に加えて, 選択肢を提示する方法を変更する。出力の最高値 2 つではなく, 出力値を重みとしたランダムな抽出で選択肢を提示する方法をとった。

3.3.1 この手法の手順

1. 自動でゲームを 10 回遊ぶ (ルールベース AI による)。
2. 1. において, ディレクションと最終到達階層を記録し, それらを基にスコアを算出する。
3. 1. と 2. を 10 回行い, データを溜める (最大で 2000 データ, 最大 10 階*10 プレイ*スコア算出 10 回)。
4. 3. を教師データとしてディレクターを学習させる。
5. 学習が終ったデータを削除し, 1. に戻る。

3.3.2 ゲーム中におけるディレクターの構造

入力値 7 個, 出力値 17 個の, Chainer を利用したニューラルネットワーク。入力値は, プレイヤーのステータス (表 2 を参照) の値 6 つに, 現在のプレイヤー位置の値を合わせた実数値 7 個。出力値は, 出現する全ての敵 11 種類と, 出現する全アイテム 6 種類, それぞれの価値を決める実数値 17 個。それらのステータスは先の手法に同じ。表 3 を参照。出力値を重みとし, 重み付きランダムで出現する敵またはアイテムを 2 つ選ぶ。

3.3.3　学習におけるディレクターの構造

この手法の手順 (3.3.1) で示した通り記録された, ディレクターがゲーム中で使用した入出力の値のうち, 選ばれた選択肢 (重み付きランダムで選ばれていた 2 つ) を, 算出されたスコア値に変更し, 誤差修正関数にかけることで学習を行う。

3.3.4　結果

3000 回目前後の学習以降, スコアが安定して-5 以上出る結果となった。選択肢を直接決めるのではなく, 何が出やすいのか傾向を決めさせることで, 選択毎に変化が生じるゲームとなった。また ε によらない通常の学習でもランダム性を持たせることで, 現状よりも良い選択を発見する助けにもなっているのではないかと考察できる。しかし, 学習 3000 回目以降も学習を続けても, -5 より高いスコアを安定して出すことは叶わなかった。この原因は学習のし過ぎで高いスコアの状態で難易度が安定していないものと仮定された。ゲームの結果とスコアの遷移は図 6 の通り。

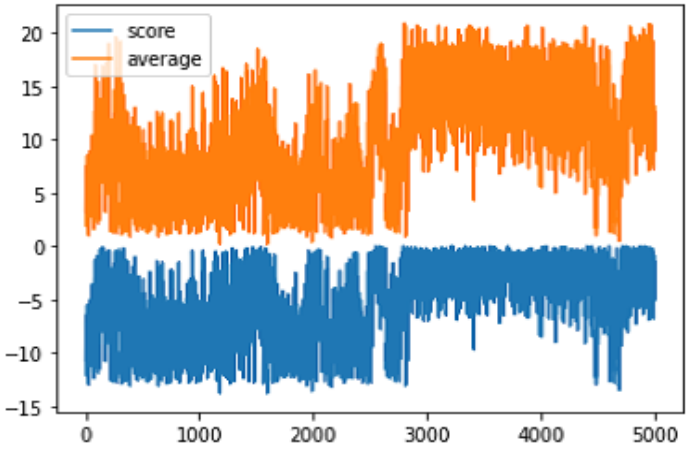

図 6: 重み付きランダムを採用した方法における平均到達階層とスコア

3.4　学習に応じて学習率を低下させていく方法

3.3 章の手法を改良し, 直近のスコアから学習の進み具合を見て, 学習率を低下させる方法をとった。図 6 において学習 3000 回目以降, 目標付近で揺れ動いている average(橙の線) を目標に収束させることが狙いである。

3.4.1　この手法の手順

1. 自動でゲームを 10 回遊ぶ (ルールベース AI による)。
2. 1. において, AI ディレクターの出力 (以下, ディレクションとする) と, 最終到達階層を記録し, それらを基にスコアを算出する。
3. 1. と 2. を 10 回行い, データを溜める (最大で 2000 データ, 最大 20 階*10 プレイ*スコア算出 10 回)。
4. 3. のスコアに応じて, 学習率 lr を減少させる。
5. 3. を教師データとしてディレクターを学習させる。
6. 学習が終ったデータを削除し, 1. に戻る。

3.4.2　ディレクターの構造

この手法は, 3.3 章の重み付きランダムを採用した方法を改良したものであり, ディレクターの仕組みは, 学習率の変更が為される箇所を除き 3.3 章と同様である。

3.4.3　学習率の変更

スコアの取得 10 回毎に行われる学習の直前に, その 10 回のスコアの平均 avg に応じて学習率 lr を変更する。$avg > -5$ のときは式 (1), $avg <= -5$ のときは式 (2)。

$$lr = 0.01/10^{\frac{5+avg}{2}} \tag{1}$$

$$lr = 0.01 \tag{2}$$

3.4.4　結果

結果, 狙い通りにはならず, スコア-5 以上のスコアを安定して出すことはできなかった。しかし, 高いスコアが先の手法よりも早く出るようになった。-5 以上のスコアが多く出てはいるが, 学習が不足しているのか, しばしば低いスコアが出るという不安定なものとなった。ゲームの結果とスコアの遷移は図 7 の通り。

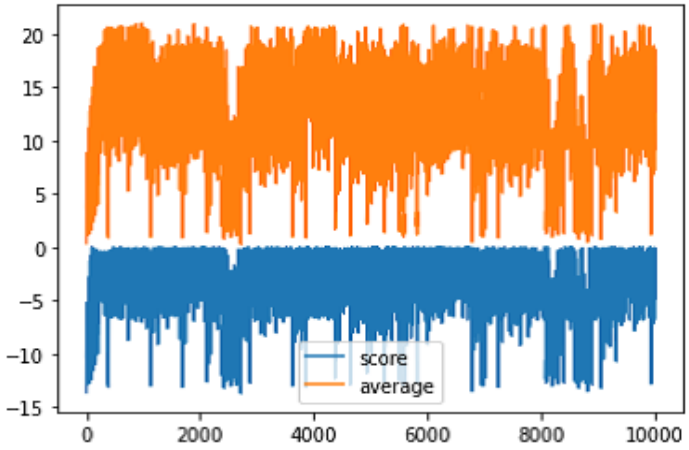

図 7: 学習に応じて学習率を低下させていく方法における平均到達階層とスコア

3.5 出現する敵を動的に生成する方法

-5 より高いスコアが出ない原因として, 一段階易しい選択肢を提示したときに易しすぎ, 逆に一段階難しい選択肢を提示したときに難しすぎる, という「帯に短し襷に長し」状態になっているがために振れ幅が大きくなっているという仮説を立てた。これまでの手法では, あらかじめ静的に決められた強さを持つ敵を十数種類から選んでいた。これを動的なものに変更する。

3.5.1 手法の手順

1. 自動でゲームを 10 回遊ぶ (ルールベース AI による)。
2. 1. において, ディレクションと最終到達階層を記録し, それらを基にスコアを算出する。
3. 2. を 10 回行い, データを溜める (最大で 1000 データ, 最大 10 階*10 プレイ*スコア算出 10 回)。
4. 3. を教師データとしてディレクターを学習させる。
5. 学習が終ったデータを削除し, 1. に戻る。

3.5.2 ディレクターの仕組み

敵とアイテムの 2 択をディレクションとして出力する。

敵を生成するために, ニューラルネットワークを 2 つ使用する。片方は, 出現する 3 種類の敵から決める (実数値 3 つを出力し, そのうち最大のもの)。もう片方は, 敵の能力の倍率を決める実数値 1 つを出力する。この手順は, 図 3.5.2 に表している。

更にもう 1 つのニューラルネットワークを使用し, 出現するアイテムを決める (実数値 6 つを出力し, そのうち最大のもの)。

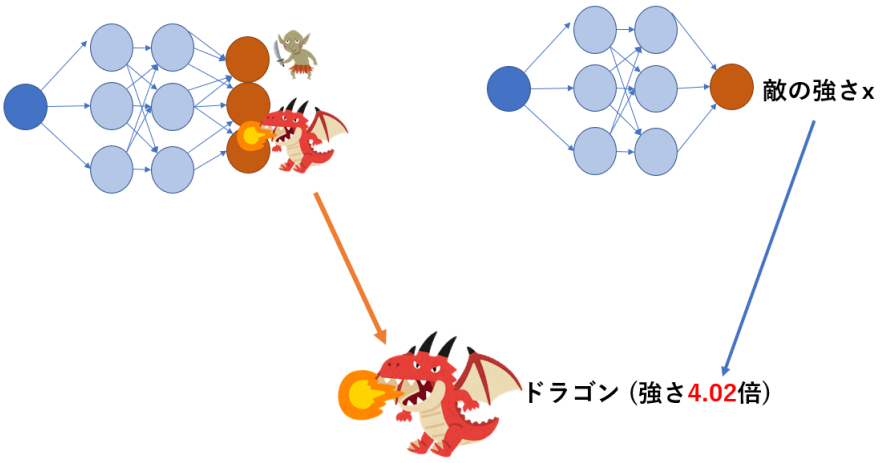

図 8: 出現する敵を動的に生成する方法

この手法で登場する敵およびアイテムは, 表 4 の通り。

3.5.3 結果

スコアが-3 前後となり, これまでの手法よりも高いスコアとなった。しかし, これ以上続けてもスコアの向上は認められなかった。また, この手法ではニューラルネットワークを 3 つ使用するため処理時間が長くなり、1 ゲームのプレイ時間が長くなってしまった (3.2 章〜3.4 章の場合 1 ゲームに最長約 0.016 秒, この手法の場合最長約 0.245 秒)。そのため規模の大きいゲームの学習には向かないものと思われる。ゲームの結果とスコアの遷移は図 9 の通り。

4 おわりに

ゲームのバランス調整には, あらかじめ設定された敵の中から選択させる離散的な手法よりも, 敵を動的に生成する連続的な手法のほうが適していることが分かった。また今回の場合, 学習のためにゲームをプレイする回数が少なくとも 20, 000 回程度, 万全を期すなら 100, 000 回以上必要であることが分かった。
今回のゲームは今日遊ばれているようなゲームと比べてかなり単純なものであるため, 実際のゲームで強化学習を使いゲームバランスを調整する場合, さらに多くの試行回数が必要であることが予測される。これを人力で行うことは現実的ではなく, 自動的にプレイする機構が必要であると考えらる。また, それができるゲームの種類は限られてくるものと推測される。

表 4: ゲーム中に出現する敵およびアイテムのリスト

敵 (x は第 2 ニューラルネットワークの出力値)			
名称	生命力	攻撃力	防御力
スライム	4*x	1*x	2*x
ゴブリン	3*x	2*x	1*x
ドラゴン	4*x	2*x	2*x

アイテム	
名称	効果
新しい鎧	最大生命力を 5 増加
新しい剣	攻撃力を 2 増加
新しい盾	防御力を 2 増加
爆弾	爆弾所持数を 1 増加
薬草	薬草所持数を 1 増加
宿屋	生命力を 10 回復

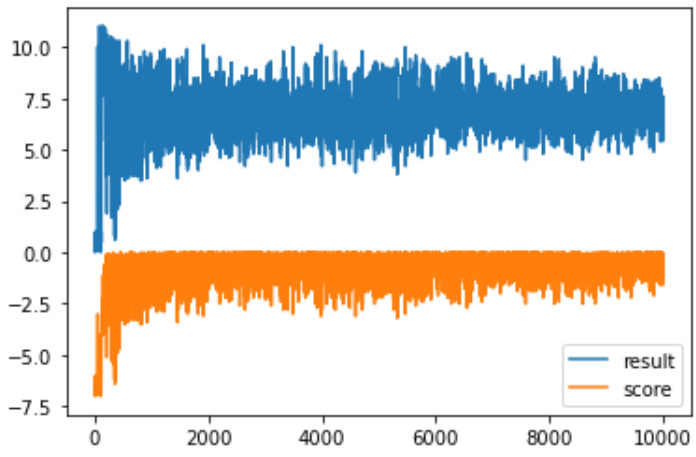

図 9: 出現する敵を動的に生成する方法における平均到達階層とスコア

参考文献

[1] Sebastian Raschka, Vahid Mirjalili, 株式会社クイーブ: [第 3 版]Python 機械学習プログラミング 達人データサイエンティストによる理論と実践, 株式会社インプレス, 2020 年 10 月 21 日

[2] ishizakiiii: OpenAI Gym 入門
yyyy 年 mm 月 dd 日最終確認
https://qiita.com/ishizakiiii/items/75bc2176a1e0b65bdd16

[3] ishizakiiii: DQN（Deep Q Network）を理解したので, Gopher くんの図を使って説明
yyyy 年 mm 月 dd 日最終確認
https://qiita.com/ishizakiiii/items/5eff79b59bce74fdca0d#q-learning

[4] icoxfog417, Takahiro Kubo: ゼロから Deep まで学ぶ強化学習
yyyy 年 mm 月 dd 日最終確認
https://qiita.com/icoxfog417/items/242439ecd1a477ece312
https://www.slideshare.net/takahirokubo7792/python-openai-gym

[5] Mark Brown, Game Maker's Toolkit: What Makes Good AI?
yyyy 年 mm 月 dd 日最終確認
https://www.youtube.com/watch?v=9bbhJi0NBkk&t=646s

[6] Michael Booth, Valve: The AI Systems of Left 4 Dead
yyyy 年 mm 月 dd 日最終確認
https://steamcdn-a.akamaihd.net/apps/valve/2009/ai_systems_of_l4d_mike_booth.pdf

[7] Mark Brown, Game Maker's Toolkit: What Capcom Didn't Tell You About ResidentEvil4
yyyy 年 mm 月 dd 日最終確認
https://www.youtube.com/watch?v=zFv6KAdQ5SE

[8] ファミ通: バイオハザード 4 解体真書, スタジオベントスタッフ, 2005 年 4 月 2 日

[9] ???: ???, ????年??月??日

付録　ソースコード

当研究で使用したゲームおよび学習システムのソースコードは, GitHub 上で公開されている。
https://github.com/KaichiHamaishi/aiDirectedRPG