Unity ML-Agents 実行環境の構築と強化学習における報酬の影響の検討

岩田　員典（経営学部）
勝又　洋斗（経営学部）

要旨

近年，Artificial Intelligence（人工知能，以下 AI）の発展はめざましく，さまざまな分野に浸透してきている。特に機械学習の一種であるディープラーニングを利用したさまざまなアプリケーションは著しい成果を上げている。しかし，機械学習は初期値やパラメータにより学習結果が大きく変わる。そこで，本論文では機械学習の一つである強化学習において報酬がどのような影響を及ぼすかについて調査する。そのために，Unity が提供する Unity Machine Learning Agents（以下，ML-Agents）を使用する。しかし，ML-Agents を実行する環境を構築するのは，頻繁にアップデートされることもあり煩雑である。そこで，本論文では Windows，Linux，Mac における ML-Agents の実行環境の構築方法について説明をする。そして，この環境を使って，サンプルに含まれる人型エージェント「Walker」の報酬を変更した場合の学習結果について検証をする。

キーワード：Unity ML-Agents，機械学習，強化学習

1. はじめに

AI という言葉は今や，我々の日常生活の中で見かけることが珍しくないほど頻繁に使用されている。特に機械学習の一種であるディープラーニング[1]を利用したさまざまなアプリケーションは著しい成果を上げている。例えば，テーブルゲーム AI は急速に進化しており，完全情報ゲームであるオセロ[2]，チェス[3]，将棋[4]，碁[5]などでは AI はプロに勝利するほどに成長している。更に，プレイヤー同士の読みあいなどが発生する不完全情報ゲームである，大富豪（大貧民）[6]，ポーカー[7]や人狼ゲーム[8]などのテーブルゲームでも AI の研究が進められている。

このように様々な分野で活用されている機械学習ではあるが，初期値やパラメータにより学習結果が大きく変わる。そこで本論文では，機械学習を行う際にパラメータがどのような影響を及ぼすかについて調査する。そのため，Unity が提供する Unity ML-Agents[9]を使用し，サンプルに含まれている学習環境「Walker」の報酬を変更し，その影響について検証する。また，ML-Agents の実行環境の構築は，OS によって要求されるライブラリのバージョンが異なるなど

煩雑である。そこで，Windows，Linux，Mac における ML-Agents の環境構築方法と実行方法について説明する。

本論文の構成は次の通りである。まず，第 2 章で AI と機械学習について説明をする。次に，第 3 章で Unity ML-Agents と機械学習について述べる。そして，第 4 章で Unity ML-Agents の学習環境の構築方法について，第 5 章で学習の実行と結果の利用方法について説明をする。第 6 章でこれらを活用した報酬が与える影響と学習結果について論じる。最後に第 7 章でまとめと今後の課題について述べる。

2. AI と機械学習

本章では AI について簡単に説明をし，機械学習やディープラーニングの位置づけについて述べる。

2.1. AI の定義と現状

AI とは，「人工的にコンピュータ上などで人間と同様の知能を実現したもの」と定義されている。しかし，一般的に非常に広い概念をもった言葉で，専門家によっても定義が異なる[10]。

AI にはレベルというものがあり，一般に「汎用人工知能」と「特化型人工知能」に大別される。汎用人工知能は，人間の知能に迫って人間の仕事をこなせるようになり，幅広い知識と何らかの自意識を持つとされる。「人工知能によって人類が滅ぼされる，人間の知能を人工知能が上回る」と言った文脈で使われるものは汎用人工知能に分類できる。

一方，特化型人工知能は，全認知能力を必要としない程度の問題解決や推論を行うソフトウェアを指す。例えば，前述した囲碁のプロ棋士に勝利した AlphaGo[5] や，ディープラーニングなども，この特化型人工知能に区別される。

2022 年現在，汎用人工知能は実現されておらず，現存しているのは特化型人工知能のみである。

2.2. 機械学習

機械学習とは大量のデータの中から規則性を見つけ，推論に有用な規則，ルール，判断基準などを機械に生成させる手法である。AI の研究分野の 1 つである。

機械学習が用いられる以前の AI は，予測や判断を行うためのルールを全て人間が考える必要があった。よって AI の限界は，人間の限界によるものとなっていた。

しかし，機械学習の登場により，ある特定の分野においては，AI が人間の知能を超えることができるようになってきている。

2.2.1.「学習」と「推論」

機械学習には「学習」と「推論」という２つのプロセスが存在する。

「学習」とは，大量の学習データの統計的分布から特徴を抽出し，実際のデータから推論するための特徴の組み合わせパターン（推論モデル）を生成するためのプロセスである。

「推論」とは，分類や識別をしたいデータを，「学習」で生成した特徴の組み合わせパターン（推論モデル）に当てはめ，推論結果を導き出すプロセスである。

2.2.2. 機械学習の手法

機械学習の手法にはいくつかの種類がある。代表的なものとして「教師あり学習」，「教師なし学習」，「強化学習」などがある。

教師あり学習

教師あり学習（Supervised Learning）は，学習データに正解ラベルを付けて学習する手法である。予測の基礎となる正解ラベルと，学習に使用する学習データをセットで学習させることにより，入力されたデータに対して予測を出力する推論モデルを生成する。

教師あり学習は大きく「分類」と「回帰」の２つに分けることができる。

「分類」とは入力されたデータに対し，出力としてデータの属性または種類を返す手法である。迷惑メールかどうかを判断するスパムフィルタなどがこれに該当する。

「回帰」とは入力されたデータに対し，出力として数値を返す手法である。商品の販売予測などがこれに該当する。

教師なし学習

教師なし学習（Unsupervised Learning）は，データの構造を学習させる手法である。

学習データのみで学習させ，そのデータに含まれた潜在的なパターンを見つけ出す推論モデルを生成する。最も一般的な教師なし学習の手法は「クラスタリング」である。

「クラスタリング」は学習データのパターンを見つけ出し，似たパターンを持つ性質の近いデータ同士をまとめる手法である。類似購買者のグルーピングなどがこれに該当する。

強化学習

強化学習（Reinforcement Learning）は，ある「環境」に置かれた「エージェント」が環境から「状態」に関する情報を得る。そして，その「状態」に対し「行動」し，得られる「報酬」が最大化するような方策を求め，推論モデルを生成する手法である。「Unity ML-Agents」において，主に使用される学習方法である。他にも，将棋や囲碁などの対戦ゲーム AI や，車の自動運転などに利用される（図 1）。

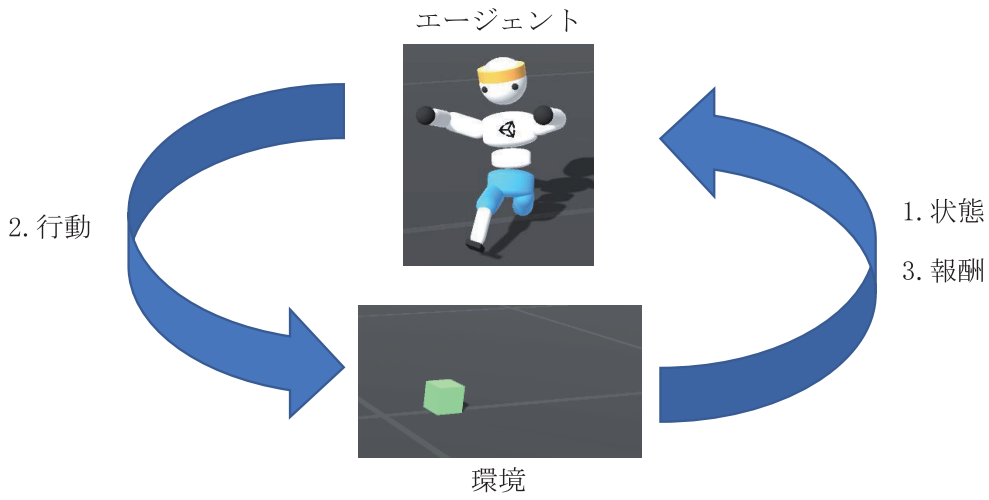

図１　強化学習の例

2.3. ディープラーニング

ディープラーニング[1)]は，教師あり学習の１つである「ニューラルネットワーク」を元に開発されたものである。本節ではニューラルネットワークとディープラーニングについて概要の説明をする。

2.3.1. ニューラルネットワーク

ニューラルネットワークとは，人間の脳の神経細胞を模した数理モデルである。ニューラルネットワークは「入力層」,「中間層」,「出力層」を持つ階層構造で構成される（図 2)。

各層にはノード（図２中丸印）がいくつも配置され，それぞれのノードの間はエッジ（図２中矢印）で結ばれる。ニューラルネットワークでは入力層から中間層，中間層から出力層へとデータを伝播させていく。各層のノードには数値データが格納されている。異なる層間のノード同士はエッジで結合しており，ノードに格納された数値データは，エッジを介して次のノードへと伝播していく。このとき，エッジには「重み」が付与されており，データが入力層から伝播して出力層へたどり着くと出力値（予測値）を得ることができる。ニューラルネットワークにおいて一般的に,「重み」は特定の個体ごとに値を設定され,「重み」はシナプス結合の強さを表す。中間層では，予測の精度を高めるための学習が行われる。

2.3.2. ディープラーニング

ディープラーニングは，ニューラルネットワークの多数の中間層があるモデルを構築できるため,「ディープニューラルネットワーク」(Deep Neural Network：DNN) とも呼ばれる。従来のニューラルネットワークでは中間層は２～３層程度であっ

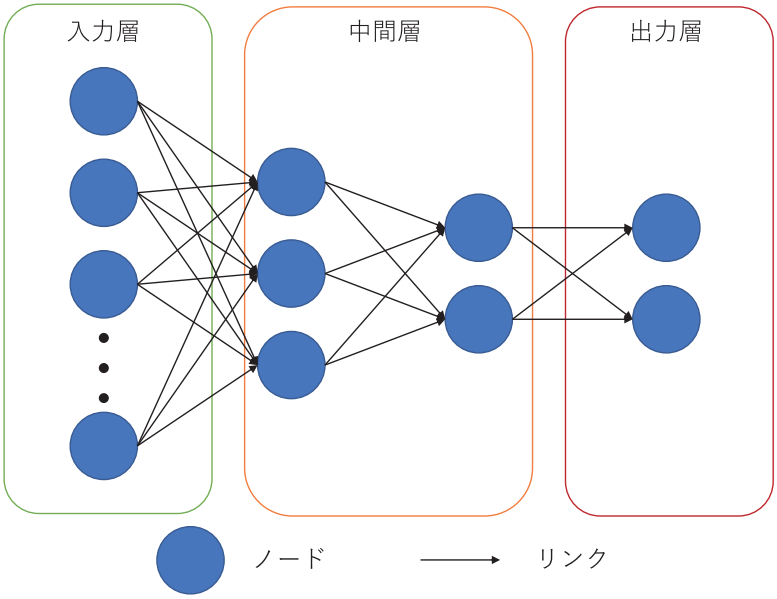

図２　ニューラルネットワークの基本構成

たが，ディープニューラルネットワークでは中間層は 100 を超える学習も可能になっている。こういった多層での学習が可能となった要因として，コンピュータ処理能力，特に CPU（Central Processing Unit）や GPU（Graphics Processing Unit：画像処理に特化したプロセッサのこと）の性能向上などがあげられる。

また，画像認識などにおいては畳み込みニューラルネットワーク（Convolutional Neural Network，以下 CNN）が用いられている。CNN では畳み込み層とプーリング層を有している。これらの層を使うことで CNN は画像から特徴を抽出し，画像内における対象の位置にかかわらず高い精度での認識が行える。

このように，ディープラーニングは特に画像や音声，言語などの非構造化データから特徴量を抽出し，精度の高い推論モデルを構築することに長けている。これらの非構造化データは説明変数の数が多く，特徴量の抽出が難しい。そのため従来の機械学習の手法を用いて精度の高い推論モデルを構築するためには，画像データや音声データ，自然言語処理の専門知識が必要である。一方，ディープラーニングを用いることにより，機械が自動的に特徴量を抽出するため，専門知識が得られないような場合にも精度の高い推論モデルを構築するが可能になっている。

3. Unity ML-Agents と機械学習

Unity[11] とはゲーム制作の大衆化を目

指して作られたゲームエンジンである。Unity を使用すれば 3D ゲーム開発や，物理エンジンなど，専門知識が少なくても導入することが可能である。

また，Unity ML-Agents は機械学習のためのライブラリ「PyTorch」[注1)] と Unity で作成した学習環境を連携させ，Unity のシーン内のキャラクターを学習させることが可能である。これを利用することにより，深い専門知識なしで，視覚的にわかりやすく機械学習を体験することができる。

Unity ML-Agents で利用可能な主な学習方法として「強化学習」「模倣学習」「強化学習＋模倣学習」がある。

3.1. 強化学習

強化学習は前述のように，ある「環境」に置かれた「エージェント」が環境に対し「行動」し，得られる「報酬」が最大化するような方策を求め，推論モデルを生成する手法である。学習に時間がかかるが，人間を超える能力を得られることがある。

3.2. 模倣学習

模倣学習は他人が行う行動を模倣して学習させる手法である。Unity ML-Agents では，人間（Player）が操作するエージェントの行動を模倣して学習する。そのため人間に似た能力を素早く得ることができる。

3.3. 強化学習＋模倣学習

強化学習＋模倣学習は，まずは模倣学習により「人間の行動」という「正解データ」を元に学習をする。そして，その学習結果を更に強化学習により改善をする。

3.4. その他の学習方法

その他にも，Recurrent Neural Network（RNN），Long Short-Term Memory（LSTM），カリキュラム学習等を利用することができる。

4. Unity ML-Agents の学習環境の構築

Unity ML-Agents を動作させるためには以下のソフトウェアを導入する必要がある。ただし，OS の種類によって動作するバージョンなどが異なるため各節で Windows，Linux，Mac の設定について説明する[注2)]。

- Unity Hub[12)]

注 1) 以前は TensorFlow が用いられていた。

注 2) 本論文執筆の 2022 年 11 月時点の情報

- Unity[12)]
- Unity-ML Agents[13, 14)]
- Python
- PyTorch[15)]
- CUDA[16)]・NVIDIA cuDNN[17)]（利用したい場合）

なお，Unity のインストールの際にユーザ登録やライセンスの取得が必要になる。大学教員の場合に，個人で使用する PC では Unity Educator Plan[18)] を使い，教室や研究室では教育機関向けライセンス[19)] を利用する必要がある。学生の場合は Unity Student プラン[20)] を利用するとよい。

4.1. Windows での設定

Windows10・11 では以下のバージョンの各ソフトウェアをインストールすることで，比較的容易に実行することができる。

- Unity Hub：3.3.0
- Unity：2021.3.21
- Unity ML-Agents：Release 19
- Python: Anaconda[21)] Python 3.9・64-Bit Graphical Installer を利用
- PyTorch：1.13

4.1.1. Unity Hub と Unity のインストール

Unity Hub をダウンロード（図 3）・インストールし，Unity Hub から Unity をインストールする。インストールは「初めて Unity をインストールする手順について＜ Windows 編＞」[22)] が参考になる。

Unity Hub の Preferences（図 4）から Appearance を選び Language を「日本語」にすることで，Unity Hub を日本語化できる（図 5）。

図 3　Unity Hub のダウンロード

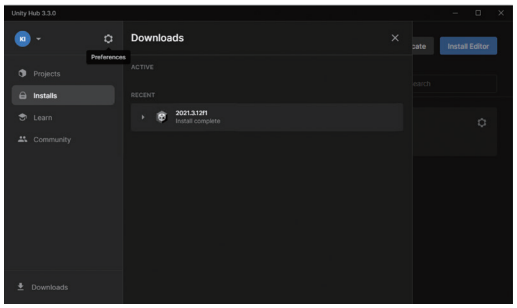

図 4　Unity Hub の Preference

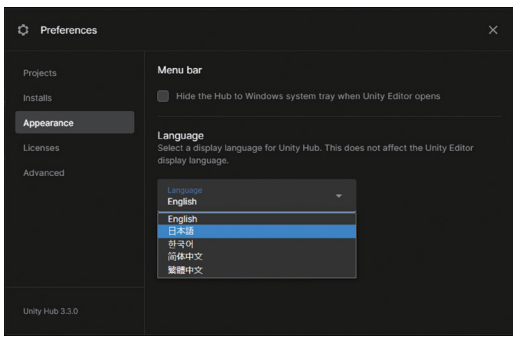

図 5　Unity Hub の言語の選択

Version	Release Date	Source	Documentation	Download	Python Package	Unity Package
main (unstable)	--	source	docs	download	--	--
Release 19	January 14, 2022	source	docs	download	0.28.0	2.2.1
Verified Package 1.0.8	May 26, 2021	source	docs	download	0.16.1	1.0.8

図 6　ML-Agents のバージョンとダウンロード

4.1.2. Unity ML-Agents のダウンロードと設定

利用したい Unity ML-Agents のバージョンを選びダウンロードする（図 6）。download をクリックすると zip ファイルが得られる。git を使っている場合は HTTPS 経由でも同様のファイルが得られる。

そして，上記のいずれかの方法でダウンロードした ml-agents-release_19.zip を展開し，任意の場所に移動させる。

次に，図 7 のように Unity Hub の「開く」から「ディスクから加える」を選び，ML-Agents のプロジェクトを加える。

加えるプロジェクトは「ml-agents¥Project」となる。ただし，ml-agents は展開したフォルダ名のため，「ml-agents-release_19」のような名称となることもある。

図 7　Unity へプロジェクトの追加

4.1.3. Python と PyTorch の準備

Python は Anaconda 経由でインストールする。

Anaconda と PyTorch のインストール方法は「コンピューターに PyTorch をインストールして構成する」[23] が参考になる。

また，NVIDA 製のグラフィックスボードを搭載している場合は CUDA をインストールすることで利用できる。

Anaconda Prompt を起動し以下のコマンドで Python の仮想環境を作成する。仮想環境の作成は必須では無いが，Anaconda の base 環境で pip3 を使うと Anaconda の動作がおかしくなることがあるため，こだわりがない限りは仮想環境を作成した方がよい。なお，仮想環境を作成する前に pip を最新にしておく。

```
> python.exe -m pip install
  --upgrade pip
```

ここで「>」はプロンプトを表している。またコマンドは一行だが誌面の都合上折り返して記載する（以下同様）。

また，Python のバージョン情報が必要なため，python--version で確認する。このバージョン（本論文執筆時では 3.9.13）を使い，以下のコマンドにより仮想環境を作成することができる。

```
> conda create --name ml-agents19
  python=3.9.13
```

仮想環境の作成ができたら以下のコマンドにより有効にする。

```
> conda activate ml-agents19
```

また，仮想環境を無効にし，元の環境に戻る場合は「conda deactivate」を実行する。なお，作成した仮想環境を確認する場合は「conda info -e」である。

PyTorch のインストールは CPU と CUDA によって異なるので注意が必要である。PyTorch のページ[15]から環境を選択すると下部に実行すべきコマンド Run this command：に表示される（図 8）ので，それをコピーして実行する。

PyTorch のインストール（CPU）

```
> conda install pytorch torchvision
  torchaudio cpuonly
  -c pytorch
```

PyTorch のインストール（CUDA）

```
> conda install pytorch torchvision
  torchaudio pytorch-cuda=11.7 -c
  pytorch -c nvidia
```

PyTorch がインストールできているかは，Python を起動し「import torch」を実行することで確認することができる。また，CUDA が利用できるようになっているかどうかは，「import torch」の後に「torch.cuda.is_available()」を実行することで確認が行える。

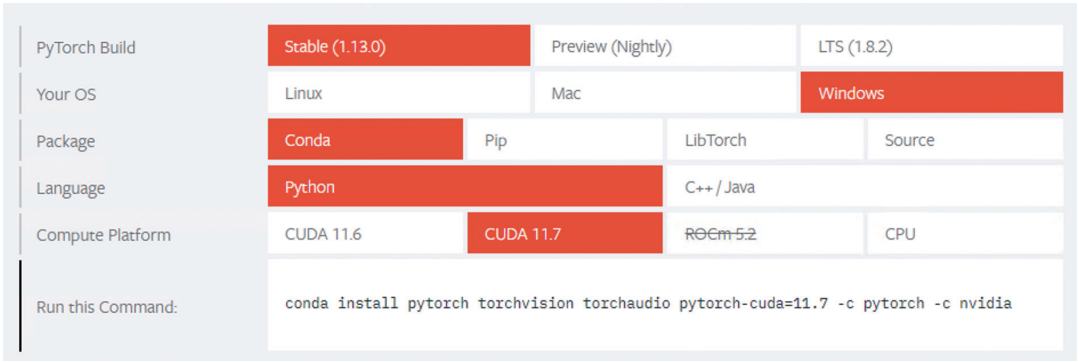

図 8　PyTorch の選択

```
> python
>>> import torch
>>> torch.cuda.is_available()
True
```

「True」と表示されればCUDAが利用できている。ここで「>>>」はPythonインタプリタの対話モードであることを表している。なお，対話モードから抜けるには「quit（）」と入力する。

4.1.4. Pythonによる ML-Agentsの設定

Anaconda Promptから，Downloadし展開したml-agentsのフォルダに移動する。例えば，Documentsの下に展開してあれば以下のようなコマンドになる。

```
> cd C:¥Users¥ユーザ名
  ¥Documents¥ml-agents-release_19
```

このフォルダ（ここではml-agents-release_19）以下にあるml-agentsとml-agents-envsの内容を編集可能な状態のパッケージとしてインストールする。

これにより，これらのフォルダ以下のファイルを編集しても，再インストールすることなしに編集結果を実行に反映できる。実行するコマンドは以下の通りである。

```
> pip3 install -e .¥ml-agents-envs¥
> pip3 install -e .¥ml-agents¥
```

上記の設定を行うとmlagents-envsとmlagentsの0.28.0がインストールされるが，python 3.9.13では0.29.0に変更しないとエラーが起きる。そこで以下のコマンドを実行する。なお，mlagents-envsのインストールで警告が出るが無視してよい。

```
> pip3 install mlagents-envs==0.29.0
> pip3 install mlagents==0.29.0
```

4.2. Linux（Ubuntu）での設定

本節ではUbuntu 22.04.1 LTSでML-Agentsを利用するための設定について説明をする。Ubuntuでは以下のバージョンの各ソフトウェアを使う。

- Unity Hub：2.4.6
- Unity：2022.1.23f1[注3)]
- Unity ML-Agents：Release 19
- Python：Anaconda Python 3.7・64-Bit Graphical Installerを利用
- PyTorch：1.8.2

注3) 本論文執筆時には2021.3.14f1(LTS)が推奨されるが，インストール時にエラーになる。また，いずれのバージョンを利用してもProjectがうまく読み込めないという問題が生じている。

4.2.1. Unity Hub と Unity のインストールと ML-Agents の設定

「Unity をダウンロード」[24] から「Unity Hub をダウンロード」（図 9）を選び，「UnityHub.AppImage」をダウンロードする（本来ならば Installing the Hub on Linux[25] に従って Unity Hub をインストールするべきだが，起動時にエラーが生じる）。

図 9　Unity Hub のダウンロード

ダウンロードができたら「UnityHub.AppImage」に実行権限を与える。Google Chrome からダウンロードした場合は，ホームディレクトリの下の Downloads に保存されているはずなので，次のようなコマンドを実行する。

```
> cd Downloads
> bash Anaconda3-YYYY.MM-Linux-x86_64.sh
```

また，fuse がインストールされている必要があるため，未インストールの場合は apt を使いインストールしておく。

```
> sudo apt install fuse libfuse2
> cd Downloads
> chmod 777 UnityHub.AppImage
```

実行権限を付与しても，そのまま実行するとエラーが生じるので，以下のように --no-sandbox オプションを付けて実行する。

```
> ./UnityHub.AppImage --no-sandbox
```

UnityHub が起動したら 4.1.1・4.1.2 と同様に Unity や Unity ML-Agents の設定を行う。

4.2.2. Python と PyTorch の準備

Windows と同様に Python は Anaconda 経由で設定をする。Anaconda のインストールは「【Ubuntu Server 18.04】Anaconda をインストールする」[26] が参考になる。

Anaconda は「ANACONDA DISTIRUTION」[21] にアクセスし，Download をクリックする。Anaconda3-YYYY.MM-Linux-x86_64.sh をダウンロードすることができるはずなので，以下のようにこのファイルを実行する（YYYYY には西暦が，MM には月が入る）。

ライセンスの認証には「yes」と入力し，インストール先はデフォルトの「/home/<USERNAME>/anaconda3」を選択する。

これらの設定が完了したら，利用する python を Anaconda のものにするために，ターミナルを再起動するなど，シェルの設定を読み込み直す。以下のように python の場所を調べて，上記のインストール先の bin 以下にある python が指定されていればよい。

```
> which python
/home/<USERNAME>/anaconda3/
bin/python
```

その後は，4.1.3 と同様に設定を完了させる。ただし，Ubuntu 22.04.1 LTS 上では python のバージョンが 3.7 でないとエラーになるため，下記のコマンドを使い設定を行う。

- pip のアップグレードと ML-Agents のダウンロード

```
> python --m pip install --upgrade
  pip
> git clone
  https://github.com/Unity-
  Technologies/ml-agents.git
```

- 仮想環境の設定と PyTorch のインストール

```
> cd ml-agents
> conda create --name ml-agents19
  python=3.7
> conda activate ml-agents19
> conda install pytorch torchvision
  torchaudio cpuonly -c pytorch-lts
```

4.2.3. Python による ML-Agents の設定

ML-Agents の設定も 4.1.4 と同じように行うことができる。前項のコマンドを実行している場合は ml-agents ディレクトリに移動済みのはずなので，以下のコマンドを実行することで設定が完了する。

```
> pip install -e ./ml-agents-envs/
> pip install -e ./ml-agents/
> pip install mlagents-envs==0.29.0
> pip install mlagents==0.29.0
> pip install importlib-
  metadata==4.4
```

なお，これらの実行が完了したら，パスの設定を更新するために再度ターミナルを再起動する必要がある[注4)]。

4.3. Mac での設定

本節での説明は Intel プロセッサと Apple シリコンの両方に対応している。また，macOS Ventura 13.0.1 が対象である。

利用するソフトウェアは以下の通りである。ただし，執筆時点で ML-Agents に対応する PyTorch が Mac では動作しないため，Python や PyTorch は Docker[27)] 上で動作させる。この Docker の設定についても後述する。

- Unity Hub：3.3.0

注 4) rehash などを使ってもよい

- Unity：2021.3.13f1
- Unity ML-Agents：Release 19
- Docker v4.13.1

  Docker 内部の設定
  - Ubuntu 22.04
  - Python: 3.7.1
  - PyTorch: 1.8.1

4.3.1. Unity Hub と Unity のインストール

「Unity をダウンロード」[24]から「Unity Hub をダウンロード」をクリックすると「UnityHubSetup.dmg」がダウンロードされるはずなので，ダブルクリックしインストールをする。インストールが完了したら，アプリケーションフォルダから UnityHub を起動し 4.1.1 と同様に設定を進める。ここで，ML-Agents は Docker からダウンロードするため 4.1.2 の設定はまだ行わない。

4.3.2. Docker を利用した Python と PyTorch の準備

前述のように ML-Agents に対応する PyTorch が Mac 向けには提供が終了しているため，Docker を利用して ML-Agents を動かせるようにする。Docker 内部で起動させた OS は Ubuntu 22.04 のため，基本的な設定は 4.1.3・4.1.4 と同様である。また，Dockerfile や各種スクリプトは GitHub に「macOS で Unity ML-Agents19 を動作させるための Docker の設定」[28]として公開している（付録 1 ～ 5）。

Agent のスクリプトなどを編集しやすくするために ml-agetns は Mac 上に置き，バインドマウントで Docker にマウントを行う方式を採用している（図 10）。

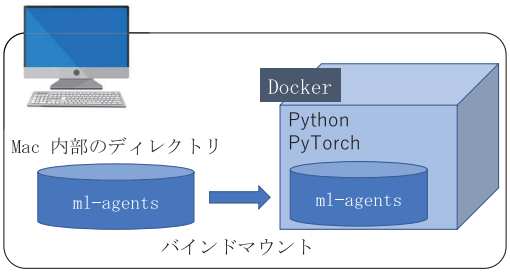

図 10　ml-agents のマウント

ml-agents を置きたいディレクトリに移動し，以下のコマンドを実施することでダウンロードから build まで行うことができる。ただし，あらかじめ docker や git が使えるように設定がなされている必要がある。

```
> git clone https://github.com/kazunori-iwata/ml-agents19-for-macOS.git
> sh ./docker-build.sh
```

4.3.3. Unity ML-Agents の設定

Docker 上で ML-Agents を動作させるには，Unity から Linux 対応のビルドを実施する必要がある。少し情報が古いが「Docker で Unity ML-Agents を動作させてみた（v0.11.0 対応）」[29]の「Unity に Linux ビルドサポートコンポーネントを追加する」が参考になる。

図 11　Unity へのモジュールの追加

まず，以下の手順でモジュールを追加する。

1. Unity Hub の「インストール」から Unity の右側にある歯車をクリックして「モジュールを加える」を選択する（図 11）
2. 表示されたモジュールを加えるにある「Linux Build Support（IL2CPP）」にチェックを入れて「インストール」ボタンをクリックする（図 12）

次に，ML-Agents に含まれているサンプルを学習できるようにする。4.1.2 の図 7 と同様にプロジェクトの追加をする。その際に，4.3.2 で保存したディレクトリ以下を指定する必要がある。

また，本論文ではサンプルにある Walker を利用するため，プロジェクトが読み込めたら「Project」タブから「Asserts → ML-Agents → Examples → Walker → Scenes」にある「Walker」をダブルクリックする（図 13）。

Unity 上部に図 14 のようなエージェントが表示されれば Scene の設定は完了である。

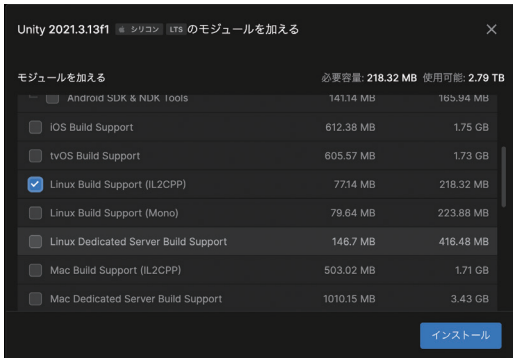

図 12　モジュールのインストール

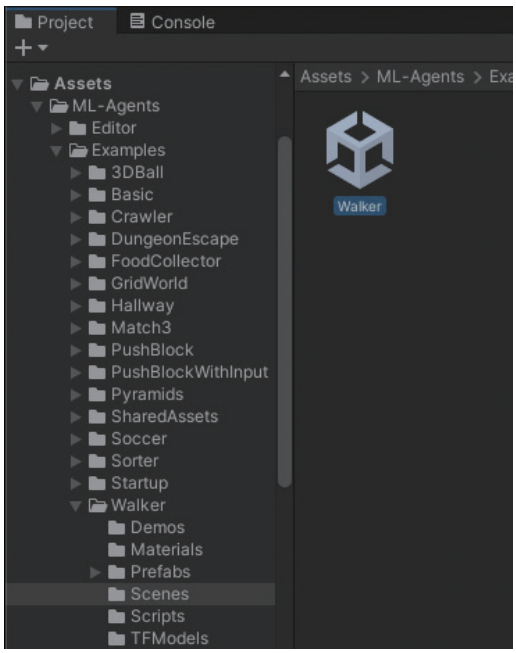

図 13　Scene を開く

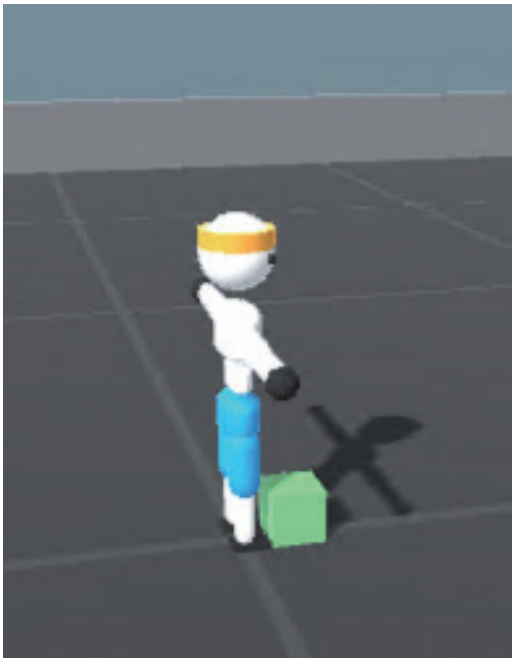

図 14　エージェントの設置例

Scene が設置できたら，ビルドの設定をし Linux で動作するようにする。

1. Unity の「File」メニューから「Build Settings...」を選ぶ（図 15）
2. 「Target Platform」から「Linux」を選択する。
3. 「Add Open Scenes」をクリックすると「ML-Agents/Examples/Walker/Scenes/Walker」が追加されるので，Scenes In Build にチェックマークが入っていることを確認する（図 16）。
4. 「Build」をクリックし（図 16），保存先を指定する。保存先は ML-Agents のトップディレクトリである「ml-agents」の下の「unity-volume」にする。また，ファイル名は「Walker」とする（図 17）。

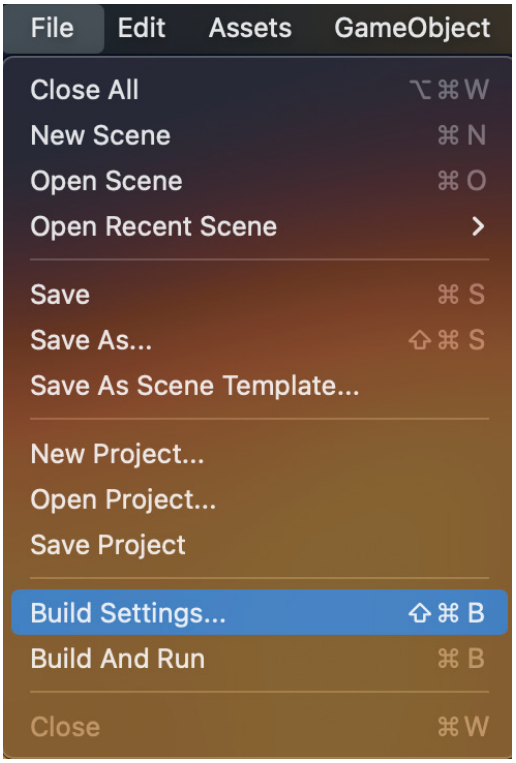

図 15　Build Settings

5. ML-Agents の学習の実行と結果の利用

ML-Agents では 17 種類のサンプル[30]が用意されており，さまざまな環境においてエージェントを学習させることができる。本論文では前述のように Walker を対象とする。

5.1. Walker の概要

Walker は以下のパーツに関して自由度 26 を有した人型ロボットである（図 18）。

- 腰
- 胸部

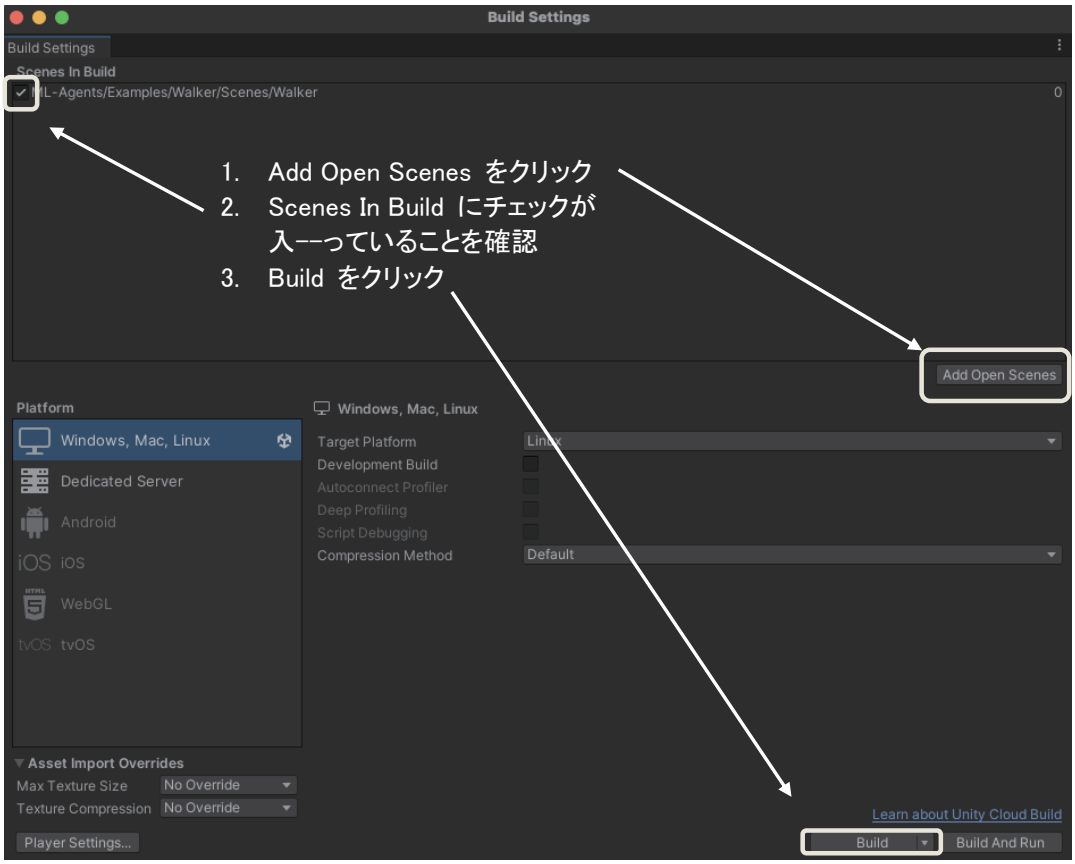

図 16　Build する Scene の設定

図 17　Build の保存先の指定

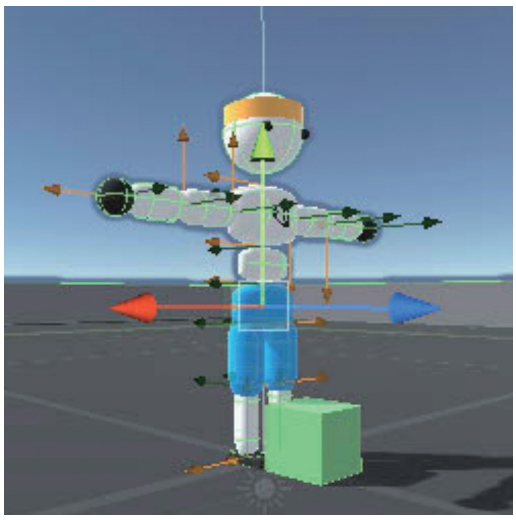

図 18　Walker エージェント

- 背骨
- 頭部
- 大腿部
- 脛骨部
- 足部
- 腕
- 前腕
- 手

このサンプルにおいてエージェントは，ランダムな位置に配置されるターゲット（ボックス）に到達することを目的とする。

5.2. 報酬（Rewards）の計算方法

強化学習の報酬は以下の方針に従って計算される。

- FixedUpdate のタイミングで報酬を受け取る。これは 0.02 秒に 1 回実施される。
  本来の強化学習ではエピソードが終了した際に合計値を受け取るが，学習を早くするためにこの方式が採用されている。
- 以下のいずれかの値になる。
  - ターゲットに到達したステップは 1
  - それ以外のステップは以下の 2 値の積
    - 全パーツの平均ベクトルがターゲットの方向にどの程度一致しているかを基準に判断する[注5)]。0 以上 1 以下の値に正規化されている。転んだ場合は 0 となる
    - 顔（頭部）がターゲットの方を向いているかどうかを基に計算される。0 以上 1 以下の値に正規化され，転んだ場合は 0 となる。

5.3. 学習の実行

学習は「mlagents-learn」（Windows では mlagents-learn.exe）によって行える。このプログラムはターミナルなどから実行する。Windows の場合は Anaconda Powershell Prompt で実行する必要がある[注6)]。

mlagents-learn の引数に学習の設定ファイルを指定し，オプション「--run-id」を使い学習結果を識別するための ID を指定する。この ID で指定された値を使って「ml-agents/results/<ID ＞」以下に実行結果が保存される。Walker を利用する場合に，学習の設定ファイルは「config/

注 5) 原文では「Body velocity matches goal velocity」と書かれているが，コードを読むと Body は bodyPartsList のベクトルの平均値を計算しており，goal velocity はボックスの方向に固定値 m_TargetWalkingSpee を乗算した値を利用している。

注 6) Anaconda Prompt では実行時にエラーになる。

ppo/Walker.yaml」である[注7)]。

また，「--torch-device=cuda:0」のように指定することでCUDAを利用することができる。

これらオプションについては「Unity ML-Agents Release 18 の mlagents-learn」[31)]で詳しく説明がされている。また，学習には時間がかかるためアプリ化して学習環境を複数起動することで高速化が可能である[32)]。

IDをfirstとした場合の実行方法は以下の通りである。

- Windows

```
> mlagents-learn.exe
  .¥config¥ppo¥Walker.yaml
  --run-id=first
```

- Linux

```
> mlagents-learn
  config/ppo/Walker.yaml
  --run-id=first
```

- Mac（Docker 利用）

```
> sh ./docker-run.sh
（以下 Docker 内）
> cd ml-agents
> mlagents-learn
  config/ppo/Walker.yaml
  --run-id=first
```

なお，コンテナを既に起動しており，exit している場合は，sh ./docker-attach.sh で再利用できる。

起動に成功すると図19のような表示の後に「Listening on port 5004. Start training by pressing the Play button in the Unity Editor.」と出力されるので，Unityの上部にある再生ボタンをクリックすることで学習が始まる。

図19　ml-agents の起動画面

学習が開始されると以下のように経過が表示される。

```
[INFO] Walker. Step: 30000. Time Elapsed: 12.147 s. Mean Reward: 0.247. Std of Reward: 1.623. Training.
```

ここで，Step が経過ステップ数，Mean Reward は1000ステップ毎の報酬の合計の平均値（この場合は30000ステップまでの1000ステップ毎），Std of Reward はその標準偏差を表す。

なお，本論文では以下の2台を利用し

注7)　本論文では Proximal Policy Optimization（PPO）を利用する

エージェントの学習を進めた。

- Ubuntu 22.04.1 LTS
  CPU: AMD Ryzen 7 5700G 3.8GHz
  RAM: 64.0 GB
- Mac Studio
  OS: macOS Ventura 13.0.1
  CPU: Apple M1 Ultra 3.2GHz
  20 コア CPU，64 コア GPU
  RAM: 128.0GB

5.4. 学習状況の監視・確認

学習がどのように進んだかは TensorBoard を利用することで確認することができる。

logdir オプションでは「results/<ID>」ディレクトリを指定する。ここでは，ID が first の場合について示す。

- Windows

```
> tensorboard.exe
  --logdir=.¥results¥first
```

- Linux

```
> tensorboard
  --logdir=./results/first
```

- Mac（Docker を利用）

```
> sh ./docker-attach.sh
（以下 Docker 内）
> cd ml-agents
> tensorboard
  --logdir=./results/first
  --host=0.0.0.0
```

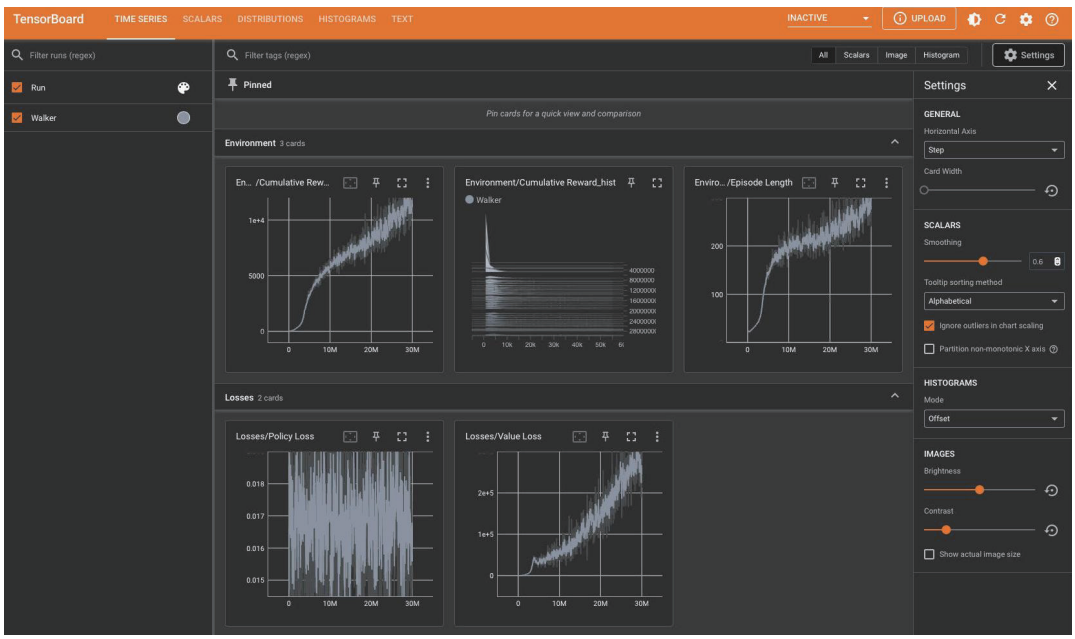

図 20　TensorBoard の例

これらのコマンドを実行して「TensorBoard 2.11.0 at http://localhost:6006/」のように表示されたらWebブラウザからhttp://localhost:6006/にアクセスすることで学習状況を監視したり確認したりすることができる（図20）。

5.5. 学習結果の利用

学習結果を利用するには以下の手順で設定をし，Unityの再生ボタンをクリックすればよい[33]。

1. results/<ID＞以下にあるWalker.onnxファイルをProject/Assets/ML-Agents/Examples/Walker/TFModels以下に置く。
2. UnityからPrefabsにあるBehavior ParametersのModelで配置したWalker.onnxファイルを指定しする
3. Behavior Typeをinferenceにする

しかし，本論文執筆時にはUnityからこれらを設定しても学習結果を利用できない[注8)]。そこで，Unity側ではなく，mlagents-learnを推論モードで実行することで，学習結果を利用する方法について説明をする。

mlagents-learnに「--inference」と「--resume」を与えることで推論モードとして起動することができる。--run-idには学習時に利用したIDを指定する。また，学習のTime Scale（表示速度）が初期値では20倍のため，--time-scale=1を指定する。

- Windows

```
> mlagents-learn.exe
  .¥config¥ppo¥Walker.yaml
  --run-id=first
  --inference --resume
  --time-scale=1
```

- Linux

```
> mlagents-learn
  config/ppo/Walker.yaml
  --run-id=first
  --inference --resume
  --time-scale=1
```

- Mac（Dockerを利用）

```
> sh ./docker-attach.sh
（学習でDockerのコンテは起動済みのはず。以下Docker内）
> cd ml-agents
> mlagents-learn
  config/ppo/Walker.yaml
  --run-id=first
  --inference --resume
  --time-scale=1
```

注8) 筆者の調査不足の可能性もあるため，確定情報ではない。また，参考文献33)の最新版が2022年12月に発刊が予定されており，その中で正確な手法が説明されているかもしれない。

6. 報酬が与える影響と学習結果

サンプルに用意されている設定で学習を進めるとWalkerはすり足でターゲットに接近をする。これは転倒すると報酬が0になるため，転倒しにくい移動方法を学習したからであると考えられる。

そこで，本論文では報酬を決定する方法を変更することで，すり足ではなく普通に人が移動しているような歩き方に近づけることを目差す。修正はProject/Assets/ML-Agents/Examples/Walker/Scriptsに置かれている「WalkerAgent.cs」に対して行う。

6.1.1. 情報取得部分の修正

姿勢の制御に関わる情報が不十分なため，一部修正する。姿勢の制御については「物理ベースのキャラを強化学習向けに調整する」にある「手順：既存のスクリプトを修正する」[34]を基に修正をする（付録6）。

6.1.2. 報酬部分の修正

報酬に関してはメソッド「FixedUpdate」で決定しているため，この部分を編集する。変更内容は以下の通りである（詳細は付録を参照）。

- 左右いずれかの足先が，逆の足よりも高い位置にあり，腰よりも低ければ高さに応じた報酬を与える。0以上1以下の値に正規化をする（付録7）。
- 頭の位置が腰の位置よりも高い場合に報酬を与える。こちらも0以上1以下の値に正規化をする（付録8）。

二つ目の報酬は設定しなくても足を上げて歩行を行うが，バランスをとろうと頭を大きく振ってしまう。そこで，頭の位置をあまり移動させないようにするため，この条件を加えている。

これらの学習の結果をYouTubeで公開している[35]（図21のリンク先）。

図21　学習の経過動画へのリンク（YouTube）

7. まとめと今後の課題

本論文では機械学習の一つである強化学習において報酬がどのような影響を及ぼすかについて検証した。この検証にはUnityが提供するML-Agentsを利用した。このML-Agentsの実行環境の構築方法や利用方法は複雑な上，OSによって異なるためWindows，Linux，Macの各OSでの方法について解説した。そして，この環境を使って，サンプル「Walker」の報酬を変更した場合の学習結果について

考察した。

今後は，本論文では動作が確認出来なかったUnityにおけるonnxの利用方法や，その他のエージェントの動作について検証を行う予定である。

参考文献

1）Yann LeCun, Yoshua Bengio, Geoffrey Hinton, “Deep Learning”. Nature. 521 (7553) : 436-444. Bibcode:2015Natur.521..436L. doi:10.1038/nature14539. PMID 26017442. S2CID 307409, (2015).

2）LOGISTELLO, 2002年11月更新. https://skatgame.net/mburo/log.html, ［アクセス日 : 2022年10月30日］.

3）Stockfish Outlasts “Rybkamura”, 2014年8月25日更新. https://www.chess.com/news/view/stockfish-outlasts-nakamura-3634, ［アクセス日 : 2022年10月30日］.

4）山本 一成, 下山 晃, 齋藤 真樹, 藤田 康博, 秋葉 拓哉, 土井 裕介, 菊池 悠太, 奥田 遼介, 須藤 武文, 大川 和仁, “第27回世界コンピュータ将棋選手権 Ponanza Chainer アピール文章”, 第27回世界コンピュータ将棋選手権, (2017)

5）Google Research Blog, “AlphaGo: Mastering the ancient game of Go with Machine Learning”, 2016年1月27日更新. https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html, ［アクセス日 : 2022年10月31日］.

6）UECコンピュータ大貧民大会 2022（UECda-2022), 2022年9月31日 更 新. http://www.tnlab.inf.uec.ac.jp/daihinmin/2022/［アクセス日 : 2022年10月31日］.

7）Noam Brown, Tuomas Sandholm, “Superhuman AI for multiplayer poker”. Science. 365 (6456) : 885-890. doi:10.1126/science.aay2400, 2019.

8）人狼知能プロジェクト, 2022年10月4日更新. http://aiwolf.org/［アクセス日 : 2022年10月31日］.

9）Unity Machine Learning Agents, https://unity.com/ja/products/machine-learning-agents［アクセス日 : 2022年10月31日］.

10）人工知能学会, What’s AI, https://www.ai-gakkai.or.jp/whatsai/［アクセス日: 2022年10月31日］.

11）Unity, https://unity.com/ja［アクセス日 : 2022年10月31日］.

12）Unity download archive, https://unity3d.com/get-unity/download/archive?_ga=2.39544163.1505483174.1667441172-525719297.1667441172［アクセス日 : 2022年10月31日］

13）ML-Agents 2022年1月14更新. https://github.com/Unity-Technologies/ml-agents［アクセス日 : 2022年10月31日］.

14）Installing ML-Agents for Windows, 2018年3月5日 更 新. https://github.com/miyamotok0105/unity-ml-agents/blob/master/docs/Installation-Windows.md［アクセス日 : 2022年10月31日］.

15）PyTorch https://pytorch.org/［アクセス

日 : 2022 年 10 月 31 日].

16) Accelerated Computing Tools https://developer.nvidia.com/accelerated-computing-toolkit［アクセス日 : 2022 年 11 月 7 日].

17) NVIDIA cuDNN, https://developer.nvidia.com/cudnn［アクセス日 : 2022 年 11 月 7 日].

18) UNITY EDUCATOR プラン, https://unity.com/ja/products/unity-educator［アクセス日 : 2022 年 10 月 31 日].

19) UNITY 教育機関向けライセンス, https://unity.com/ja/products/unity-education-grant-license［アクセス日 : 2022 年 10 月 31 日].

20) UNITY STUDENT プラン, https://unity.com/ja/products/unity-student［アクセス日 : 2022 年 10 月 31 日].

21) ANACONDA DISTRIBUTION, https://www.anaconda.com/products/distribution［アクセス日 : 2022 年 10 月 31 日].

22) 初めて Unity をインストールする手順について＜ Windows 編＞, 2022 年 2 月 24 日更新. https://forpro.unity3d.jp/tutorial/unity-install-windows/［アクセス日 : 2022 年 10 月 31 日].

23) コンピューターに PyTorch をインストールして構成する 2022 年 9 月 22 日更新. https://learn.microsoft.com/ja-jp/windows/ai/windows-ml/tutorials/pytorch-installation［アクセス日 : 2022 年 10 月 31 日].

24) Unity をダウンロード https://unity3d.com/jp/get-unity/download［アクセス日 : 2022 年 10 月 31 日].

25) Installing the Hub on Linux. https://docs.unity3d.com/hub/manual/InstallHub.html?_ga=2.126622349.1839134172.1669444065-522614030.1668563380#install-hub-linux［アクセス日 : 2022 年 10 月 31 日].

26)【Ubuntu Server 18.04】Anaconda をインストールする, 2019 年 5 月 4 日更新. https://paperface.hatenablog.com/entry/2019/05/04/【Ubuntu_Server_18.04】Anaconda をインストールする［アクセス日 : 2022 年 10 月 31 日].

27) Docker. https://www.docker.com［アクセス日 : 2022 年 10 月 31 日].

28) 岩田 員典, “macOS で Unity ML-Agents19 を動作させるための Docker の設定”, 2022 年 11 月 28 日 更 新. https://github.com/kazunori-iwata/ml-agents19-for-macOS［アクセス日 : 2022 年 11 月 28 日].

29) Docker で Unity ML-Agents を動作させてみた（v0.11.0 対応), 2019 年 12 月 23 日更新. https://qiita.com/kai_kou/items/6fbb8d7aa9d39820428b［アクセス日 : 2022 年 10 月 31 日].

30) Example Learning Environments, 2022 年 11 月 19 日更新. https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md［アクセス日 : 2022 年 11 月 29 日].

31) Unity ML-Agents Release 18 の mlagents-learn, 2021 年 8 月 17 日更新 .https://note.

com/npaka/n/n65bc79178744［アクセス日 : 2022 年 11 月 29 日］.

32）【Unity】ML-Agents の環境をアプリ化して学習を高速化する（ML-Agents で機械学習＆強化学習をやってみる　その 6），2022 年 1 月 28 日更新．https://wakky.tech/unity-ml-agents6-app-quick/［アクセス日 : 2022 年 11 月 29 日］.

33）布留川 英一（著），佐藤 英一（編），“Unity ML-Agents 実践ゲームプログラミング v1.1 対応版（Unity ではじめる機械学習・強化学習）”，ボーンデジタル，（2020）.

34）物理ベースのキャラを強化学習向けに調整する：Unity（ML-Agents（Walker）），Unity Chan，VRM（VRoid），DAZ，2022 年 5 月 6 日更新．https://mkitys.xoxxox.net/doc/setcmm_20220504143934669886895 7.htm［アクセス日 : 2022 年 11 月 29 日］.

35）岩田 員典，“Unity ML-Agents Release 19「Walker」における報酬の影響 ”，2022 年 12 月 1 日更新．https://youtu.be/xarzJGUXpjE［アクセス日 : 2022 年 12 月 1 日］.

付録 1 Dockerfile

Dockerfile

```
FROM ubuntu:22.04
MAINTAINER Kazunori Iwata

ENV PATH /usr/bin:/usr/local/bin:$PATH

RUN apt-get -y upgrade && apt-get update && apt-get install -y curl software-properties-common vim && add-apt-repository ppa:deadsnakes/ppa

ENV LANG C.UTF-8

ENV TZ=Asia/Tokyo
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get install -y python3.7
RUN update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.7 1
RUN apt-get -y install python3.7-distutils && apt-get install -y python3-pip git

RUN python3 -m pip install --upgrade pip
RUN pip3 install torch==1.8.1 torchvision==0.9.1 torchaudio==0.13.0 --extra-index-url https://download.pytorch.org/whl/lts/1.8/cpu

COPY setup.sh /

RUN mkdir /home/tmpuser

RUN useradd tmpuser
RUN USER=tmpuser && ¥
    GROUP=tmpuser && ¥
    curl -SsL https://github.com/boxboat/fixuid/releases/download/v0.5/fixuid-0.5-linux-amd64.tar.gz | tar -C /usr/local/bin -xzf - && ¥
    chmod 4755 /usr/local/bin/fixuid && ¥
    mkdir -p /etc/fixuid && ¥
    printf "user: $USER¥ngroup: $GROUP¥n" > /etc/fixuid/config.yml

RUN chown tmpuser:tmpuser /home/tmpuser
USER tmpuser:tmpuser
ENV PATH /home/tmpuser/.local/bin:$PATH

#WORKDIR /ml-agents/ml-agents-envs
#RUN pip3 install -e .
#WORKDIR /ml-agents/ml-agents
#RUN pip3 install -e .

# port 5004 is the port used in Editor training.
# port 6006 is the port used by tensorboard.
EXPOSE 5004 6006

ENTRYPOINT ["fixuid"]
```

付録 2 setup.sh

Docker-build.sh

```
#! /bin/sh

cd ml-agents

if [ ! -f .setup ]; then
    pip3 install -e ./ml-agents-envs
    pip3 install -e ./ml-agents
    touch .setup
fi
```

付録 4 docker-build.sh

Docker-build.sh

```
#!/bin/sh

localUID=$(id -u $USER)
localGID=$(id -g $USER)

docker run -it -u ${localUID}:${localGID} -p 5004:5004 -p 6006:6006 -v ${PWD}/ml-
agents:/ml-agents --name ml-agents ml-agents /bin/bash
```

付録 5 docker-run.sh

Docker-run.sh

```
#! /bin/sh
docker build -t ml-agents .

if [ ! -d ml-agents ]; then
    git clone https://github.com/Unity-Technologies/ml-agents.git
fi
```

付録 6 姿勢情報の取得に関する WalkerAgent.cs の修正部分

```
--- WalkerAgent.cs-org        2022-11-29 16:41:34
+++ WalkerAgent.cs            2022-11-30 12:04:26
@@ -15,6 +15,11 @@ public class WalkerAgent : Agent
     //The walking speed to try and achieve
     private float m_TargetWalkingSpeed = 10;

+    private Quaternion hipsInitialRatation; /*MOD*/
+
+    private Vector3 hipsInitialDirection; /*MOD*/
+    private Vector3 headInitialDirection; /*MOD*/
+
     public float MTargetWalkingSpeed // property
     {
         get { return m_TargetWalkingSpeed; }
@@ -85,6 +90,11 @@ public class WalkerAgent : Agent

         m_ResetParams = Academy.Instance.EnvironmentParameters;

+            hipsInitialRatation = hips.rotation; /*MOD*/
+            hipsInitialDirection = (m_OrientationCube.transform.rotation * Quaternion.Inverse(hips.rotation)) * m_OrientationCube.transform.forward;
+            headInitialDirection = (m_OrientationCube.transform.rotation * Quaternion.Inverse(head.rotation)) * m_OrientationCube.transform.forward;
+
         SetResetParameters();
     }

@@ -100,7 +110,8 @@ public class WalkerAgent : Agent
         }

         //Random start rotation to help generalize
-        hips.rotation = Quaternion.Euler(0, Random.Range(0.0f, 360.0f), 0);
+        // hips.rotation = Quaternion.Euler(0, Random.Range(0.0f, 360.0f), 0);
+            hips.rotation = Quaternion.Euler(0, Random.Range(0.0f, 360.0f), 0) * hipsInitialRatation; /*MOD*/

         UpdateOrientationObjects();

@@ -154,8 +165,10 @@ public class WalkerAgent : Agent
         sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(velGoal));

         //rotation deltas
-        sensor.AddObservation(Quaternion.FromToRotation(hips.forward, cubeForward));
-        sensor.AddObservation(Quaternion.FromToRotation(head.forward, cubeForward));
+        //sensor.AddObservation(Quaternion.FromToRotation(hips.forward, cubeForward));
+        //sensor.AddObservation(Quaternion.FromToRotation(head.forward, cubeForward));
+            sensor.AddObservation(Quaternion.FromToRotation(hips.rotation * hipsInitialDirection, cubeForward)); /*MOD*/
+        sensor.AddObservation(Quaternion.FromToRotation(head.rotation * headInitialDirection, cubeForward)); /*MOD*/

         //Position of target position relative to cube
         sensor.AddObservation(m_OrientationCube.transform.InverseTransformPoint(target.transform.position));
@@ -252,7 +265,64 @@ public class WalkerAgent : Agent
             );
         }
```

付録 7 報酬に関する WalkerAgent.cs 変更（足を上げる）

```
        AddReward(matchSpeedReward * lookAtTargetReward);
```

を以下のように修正

```
        float footReward = 0.001F;
        float diffFootHeight = Mathf.Abs(footL.position.y - footR.position.y);
        float diffFootAndHip = 0;

        if(footL.position.y > footR.position.y){
          diffFootAndHip = hips.position.y - footL.position.y;
        } else {
          diffFootAndHip = hips.position.y - footR.position.y;
        }

        if((diffFootAndHip > 0 ) && (0.0F < diffFootHeight ) || (0.2F >= diffFootHeight )){
          footReward = diffFootHeight /0.2F * .3F;
        }

        AddReward(matchSpeedReward * lookAtTargetReward * footReward);
```

付録 8 報酬に関する WalkerAgent.cs 変更（足を上げ，頭を高く保つ）

```
        AddReward(matchSpeedReward * lookAtTargetReward);
を以下のように修正
        float footReward = 0.001F;
        float diffFootHeight = Mathf.Abs(footL.position.y - footR.position.y);
        float diffFootAndHip = 0;

        if(footL.position.y > footR.position.y){
          diffFootAndHip = hips.position.y - footL.position.y;
        } else {
          diffFootAndHip = hips.position.y - footR.position.y;
        }

        if((diffFootAndHip > 0 ) && (0.0F < diffFootHeight ) || (0.2F >= diffFootHeight )){
          footReward = diffFootHeight /0.2F * 0.3F;
        }

        float headHeight = head.position.y - hips.position.y;
        float headReward = headHeight / 0.2F * 0.3F;

        AddReward(matchSpeedReward * lookAtTargetReward * footReward * headReward);
```