プロジェクト研究A　活動内容報告

東方花映塚のGym環境構築とDoubleDQNを用いた強化学習

TAKUTO ITOI, Waseda Univ.

近年、囲碁の世界チャンピオンを相手に三連勝したAlphaGo[Silver et al. 2016]や、Atari社が開発したビデオゲームを人間の平均スコア以上のパフォーマンスでプレイする人工知能 [Mnih et al. 2013] が開発され、ゲームAIが注目されている。そこで、ゲームAIの有用性をより複雑なゲームにおいても示す必要があると考えた。東方花映塚というゲームは画面が大量の弾丸で覆われるため、囲碁のように大量のデータを処理する必要がある。さらにターン制ではないゲームであるため、Atari社のビデオゲームのようにリアルタイムでの処理が必要あるという特徴がある。これらの2つの特徴を東方花映塚が兼ね備えることから、囲碁やビデオゲームよりも複雑度の高いゲームであると考え、ゲームAIの有用性を示すために取り組む題材として最適であると考えた。

本紙では、エージェントが学習するために、東方花映塚をプレイすることができるGym環境を作成し、そのGym環境上で深層強化学習を用いて学習するためにどのような作業を行なったかを報告する。

CCS Concepts: • **Theory of computation → Reinforcement learning**; • **Computer systems organization → Client-server architectures**; • **Security and privacy →** *Software security engineering*.

Additional Key Words and Phrases: OpenAI, gym, PyTorch, Touhou, Kaeiduka

ACM Reference Format:
Takuto Itoi. 2020. プロジェクト研究 A　活動内容報告: 東方花映塚のGym 環境構築と DoubleDQN を用いた強化学習. *ACM Trans. Intell. Syst. Technol.* -, -, Article - (August 2020), 4 pages. https://doi.org/10.1000/182

1　はじめに

東方花映塚とは、正式名称「東方花映塚　～ Phantasmagoria of Flower View.」であり、上海アリス幻樂団が東方Projectの第9作目として、2005年に発売された対戦型弾幕シューティングゲームである。弾幕シューティングゲームとは、2D画面において、敵が画面を埋め尽くすほど撃ってくる低速で大量の弾に当たらないように自機を操作しながら、敵に攻撃するゲームである。また、花映塚では東方Projectの他の作品と異なり、対戦型であり、敵を倒すことで相手フィールドにより多くの弾幕を送りつけることができる仕様になっており、最終的に相手を先に撃墜することが目的となっている。プレイヤーは上下左右への移動と、弾丸を発射することができ、またスペルカードを使用することで相手フィールドへ弾幕を送りつける事ができる。

本研究では、深層強化学習を用いて、東方花映塚をプレイするエージェントを開発することが目的である。そのために、東方花映塚をプレイするGym環境を作成し、さらにこのGym環境上でDoubleDQNを用いて学習を行なった。この開発の経緯を報告する。

Author's address: Takuto Itoi, Waseda Univ.

2157-6904/2020/8-ART- $15.00
https://doi.org/10.1000/182

2　先行研究

2.1　花AI塚

東方花映塚では、プレイヤーは画面に表示されるフィールドで敵や弾の位置を把握し、キーボード入力によってキャラクターを操作することが想定されている。しかしこれはコンピュータのメモリ上で動作するエージェントからは操作がしづらいため、メモリ上の数値を直接読み込み、各オブジェクトの位置などの盤面の状態を取得するツールとして花AI塚 [ide_an 2014] がある。また、同ツールを用いて、花映塚がプレイヤーの押しているキーを格納するメモリに書き込むことができ、そこに適切な値を書き込むことでキーから入力せずにキー入力を擬態することができる。

一方で同ツールはLuaスクリプトを内部的に毎フレーム呼び出し、そこに記述した動作を実行するため、各フレームで独立した動作しかすることができず、過去の状態を保持することなどができない。そのため、深層強化学習は過去の状態を参考にして学習を行うため、そのまま転用することはできなかった。

2.2　Deep Q-Network with Pytorch

Pytorchを用いたDoubleDQNの例としてこの研究がある。これはAtari社が開発したビデオゲームの一つであるBreakoutをDoubleDQNを用いて学習するものである。

3　GYM環境構築

はじめに、どのような手順でGym環境を作成したかを時系列順に報告する。

3.1　花AI塚の問題点

Gym環境を構築するに当たり、はじめに花AI塚を使用してエージェントに東方花映塚をプレイさせることを考えた。しかし花AI塚は前述したとおり、フレームごとに別途記述したLuaスクリプトのmain関数を実行する方式をとっており、フレームをまたいだデータの保存ができないため、花AI塚を用いてそのまま学習することはできなかった。加えて東方花映塚は学習中のエージェントの行動決定のタイミングとは全く関係なく、30fpsを保つため、同様にLuaスクリプトのmain関数もエージェントと関係なく、毎秒30回実行される。

そこで、エージェントと花AI塚が同調して動作できるように、別にサーバを立て、花AI塚はそこに毎秒30回現在の状況を送信することでサーバ内に保存されている盤面状況をアップデートし、エージェントはサーバに保存されている最新の状況を取得するという方式をとった。

加えて、東方花映塚はWindows上でのみ動作するが、今回の場合研究室のサーバを用いて学習を行うため、Linux(Ubuntu)上で動作する方法を模索する必要がある。

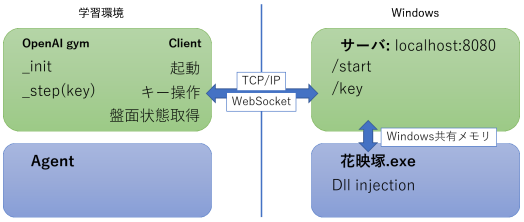

Fig. 1. それぞれのプログラムの動作状況

3.2 花 AI 塚の解析

はじめに、花 AI 塚がどのようにして動作しているかを解析した。花 AI 塚は DLL インジェクションを用いて DLL ファイルを東方花映塚に読み込ませることで、東方花映塚のメモリ内にどのようなデータが存在するかを取得していることがわかった。DLL インジェクションとは、Windows 上で動作するアプリケーションで読み込まれる DLL イメージに本来とは異なるものを注入することで、別のところで作成した DLL イメージを読み込ませる手法である。

花 AI 塚では、あらかじめゲーム開始時、ゲーム中の毎フレーム、ゲーム終了時に実行したい関数を DLL イメージとしてコンパイルしておき、東方花映塚を起動する際にこのイメージをインジェクションする。読み込まれた DLL イメージはプロセスの開始時に 1 度読み込まれるため、そのタイミングでゲーム開始時、ゲーム中の毎フレーム、ゲーム終了時に本来東方花映塚で実行されるはずだった関数のポインタを、自作した DLL 内の関数ポインタに書き換える。こうすることでゲーム開始時には、自作した関数が呼ばれる。ここで、もともと実行されるはずだった関数の処理も自作した関数に内包する必要があることに注意する。

3.3 盤面を保持するサーバの作成

はじめに、上で述べたサーバを作成した。サーバは東方花映塚に挿入された DLL イメージと、エージェントと通信する必要がある。ここで、東方花映塚が Windows 上で動作するため、同様にサーバも Windows 上で実行される必要がある。サーバは東方花映塚とは Windows の共有メモリをを用いて通信を行ない、エージェントとは WebSocket を用いて通信を行なった。共有メモリを用いた通信のほうが高速ではあるが、エージェントは Linux 上で実行されるため、カーネルに依存したこの機能は使用することができないため、共通のプラットフォームである TCP/IP を用いて通信をしている。

3.4 サーバの動作

サーバはエージェントと接続が確立されると、このエージェント用のゲームを起動する。エージェントにはそれぞれ専用の東方花映塚のゲームが起動されるようにしてある。こうすることで並列的な学習が可能になるように工夫した。また、ゲームを起動する際には、DLL インジェクションを行ない、さらに東方花映塚と通信ができるように共有メモリをアロケートする。この共有メモリは名称 (文字列) をつけてアロケートし、読み書きする際にはカーネルにこの登録した名称を渡すことでアドレスを取得している。複数のゲームを起動したときにこの名称が重複してしまうとアロケートが成功しないため、それぞれに独立した名称を書き込んだ DLL イメージを挿入させている。また、サーバにはワーカースレッドを立てておき、1 人のエージェントとのやり取りは 1 つのスレッドに担当させることで、一度に起動できるゲーム数を制限している。また、名称は 0 から始まる数字を割り当て、作成する個数はコンパイル時に指定できるようにした。

3.5 複数の東方花映塚の同時起動

東方花映塚は起動されると Windows カーネルに Mutex を登録し、ゲーム起動時にはすでに Mutex が登録されているかを確認することで、他のプロセスですでに起動されていないかを確認しているため、本来の東方花映塚では同時に 2 つ以上のゲームが起動できない。これはソースコード1に示すようなコードで判別しているため、判別しているアセンブリ命令 (cmp 命令) をゲームデータから消去した。

ソースコード 1. mutex の確認コード

```
hMSP = CreateMutex(NULL, TRUE, "MutexName");
if(GetLastError() == ERROR_ALREADY_EXISTS){
  MessageBox(NULL, "already running" ,"Multiplex starting prevention Test C", MB_OK);
  ReleaseMutex(hMSP);
  CloseHandle(hMSP);
  return FALSE;
}
```

3.6 エージェントとの通信

次にサーバとエージェントとの通信を実装した。実装が簡単になるように、盤面の状況を取得するクラスは OpenAI の提供する Gym を継承した。サーバとエージェントの通信には websocket を用いた。サーバ側が websocket サーバの役割をし、Gym 環境内で websocket クライアントを起動し、サーバと通信を行なっている。エージェントが action をする際には移動する命令がクライアントから投げられ、それを受け取ったサーバが花 AI 塚にその命令を送ることで、花 AI 塚内で東方花映塚に対して擬似的なキー入力が行われる。反対に花 AI 塚からは毎フレーム更新された盤面の状況が送られてくるため、サーバが Gym に送信する。Gym は受け取った情報をメモリに格納しておき、エージェントから問い合わせが来た際にはメモリに格納されている盤面状況を返却するといった形で通信を行なうように実装をした。

また、はじめは json 形式のデータで盤面状況を送信していたが、盤面データは非常に多くのパラメータが存在するため json の生成に非常に時間がかかりボトルネックになっていた。そこで Google Protobuf を用いてデータを送受信する実装に変更した。この変更により、データの送受信が表1の下りに示したように、約 100 倍改善された。

Table 1. データ形式と通信速度の関係

	json [s]	protobuf [s]
上り	0.1	0.1
下り	0.9	0.01

また Gym 環境を作成するに当たり、gym を継承するクラスでは以下の 4 つのメソッドを定義する必要がある。

- reset(self)
- step(self, action)

- render(self, mode, close)
- close(self)

それぞれのメソッドで実装した処理を記述する。

reset が呼ばれた際にはサーバとの接続を確立する。このときにサーバ側ではワーカースレッドの 1 つが接続を確立させ、DLL インジェクションを行ない、ゲームを起動するといった処理が実行される。

step の引数である action はどのような操作を行なうかという意味である。今回の Gym 環境では、(action & 0b11) が上右下左を順番に意味し (0==上 ···)、(action & 0b100) で弾を打つか打たないかと言う意味で定義した。本来はためてから攻撃するといった動作も存在するが、現在はそのような攻撃方法はできないものとした。step が呼ばれた際には、action で指定された動作をするようにサーバにデータを送信する。

また、盤面の評価値の計算は後述する。

また、close が呼ばれた際にはサーバにゲームを終了する命令を送信し、websocket の接続を終了する。

3.7　Linux 上での動作

前述したとおり、東方花映塚は本来 Windows 上でのみ動作するため、研究室のサーバでの動作ができない。そこで、Wine を用いることで問題を解決した。Wine とは、x86 アーキテクチャを使用する OS(Linux) 上で Windows 用のアプリケーションを動作させることを可能にしているプログラムである。

以上で Gym 環境の構築が完了した。

4　深層強化学習の実装

次に作成した Gym 環境を実際に使用して学習を行なった。本研究ではすでに存在する DoubleDQN の実装 [Singh 2019] を参考にして実装を行なった。

4.1　モデルの定義

本研究で使用した DoubleDQN のニューラルネットは以下のように定義した。

ソースコード 2.　model.py

```
class DQN(nn.Module):
    def __init__(self, obs_size, n_actions, seed):
        super(DQN, self).__init__()
        hidden = int((obs_size + n_actions) * 2 /
             3)
        self.line1 = nn.Linear(obs_size, hidden)
        self.line2 = nn.Linear(hidden, hidden)
        # self.line3 = nn.Linear(hidden, hidden)
        self.line4 = nn.Linear(hidden, n_actions)

    def forward(self, x):
        x = F.relu(self.line1(x))
        x = F.relu(self.line2(x))
        # x = F.relu(self.line3(x))
        return self.line4(x)
        # return F.softmax(self.line4(x))
```

ニューラルネットは 3 層の全結合層であり、活性化関数は Relu を用いた。

入力次元数は 17326 次元であり、これは盤面データの配列を flatten を用いて整形したものである。また出力次元数は 5 次元であり、OneHot で上下左右を表す 4 次元と、弾を打つか打たないかを意味する 1 次元を結合したものとした。

Table 2.　学習のパラメータ

最適化手法	Adam
学習率	5e-4
誤差関数	MSELoss
バッチサイズ	64
NN の更新頻度	4 フレーム
hline 行動選択手法	ϵGreedy

4.2　学習のパラメータ

本学習では、学習のパラメータは以下のように設定した。

学習は 4 回の動作をするごとに、過去の盤面の状況を無作為に 64 個選択したものをニューラルネットに通し、評価値を計算させ、本来の評価値との差からパラメータを更新し学習を行なった。

5　学習結果

評価値の計算を 2 通りの方法で行なったため、それぞれの結果を示す。

5.1　累計報酬

東方花映塚では敵を倒すと点がもらえ、その累計がプレイヤーの得点として表示されている。この点をそのまま盤面の評価値として学習させた。この条件で 50 試合を行なった得点とロスの結果を図2と図3に示す。

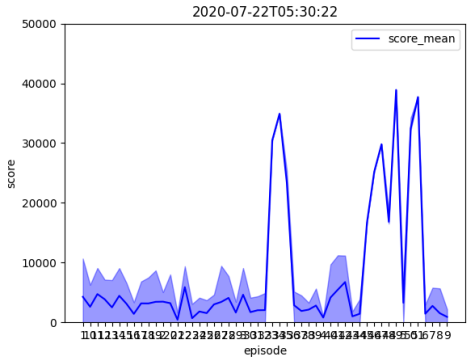

Fig. 2.　累計報酬での得点の推移

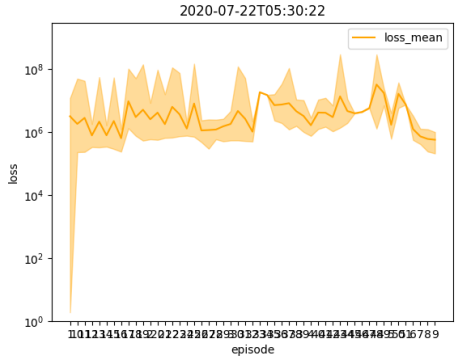

Fig. 3. 累計報酬でのロスの推移

5.2 差分報酬

次に、盤面の評価値を以下のように計算した。まず、毎フレームプレイヤーの HP を確認し、もし前フレームから減少していた場合、評価値を-1000 とした。また、減少していない場合には、そのフレームで取得した点数を評価値とした。東方花映塚では敵を倒すと点がもらえ、その累計がプレイヤーの得点として表示されているため、フレームごとに、前フレームでの得点との差を計算することで計算することができる。このような評価値にすることで、エージェントはなるべく死なず、かつなるべく敵を倒す行動を選択するように学習すると考えられる。この条件で 50 試合を行なった得点とロスの結果を図4と図5に示す。

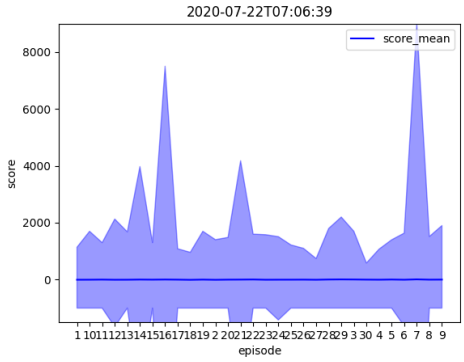

Fig. 4. 差分報酬での得点の推移

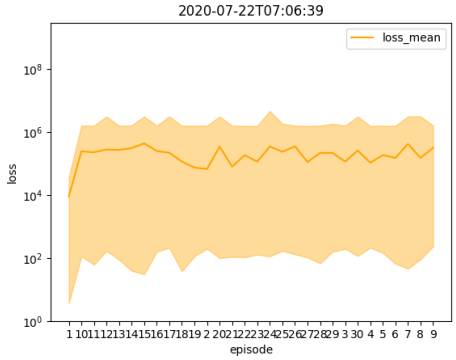

Fig. 5. 差分報酬でのロスの推移

6 考察

以上の結果から、どちらの報酬の設定においても得点の増加も、ロスの値の減少も確認することができない。そのため、学習モデルの定義を工夫する必要があると考える。

現在盤面を表す配列はすべてのデータをただ並べたものになっているが、弾の情報や、敵の情報は順番が関係ないため、PointNet などの手法を用いて、あらかじめ計算をしておくなどといった工夫が可能だと考える。

7 最後に

半年間でここまでの成果を得ることができたが、深層強化学習のモデルの定義をもう少し工夫することができると思う。夏休みや後期の時間でより上手に東方花映塚をプレイすることのできるエージェントの作成に挑戦したいと思う。

REFERENCES

ide_an. 2014. 東方花映塚 AI 自作ツール『花 AI 塚』の開発とその前後. http://www.usamimi.info/~ide/programe/touhouai/report-20140705.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

Unnat Singh. 2019. Deep Q-Network with Pytorch. https://medium.com/@unnatsingh/deep-q-network-with-pytorch-d1ca6f40bfda.