空戦 AI チャレンジ

内容説明 Minimum 版

主催 防衛装備庁

委託 Nishika 株式会社 Nishika

目次

1 本シミュレーションの内容

1.1 空対空目視外戦闘の定義

本コンペティションのシミュレーションが想定する戦闘を、以降空対空目視外戦闘と呼ぶ。空対空目視外戦闘は、彼我の複数の戦闘機が複数の誘導弾を搭載し、十分遠方から強力なセンサにより相手を探知・追尾し、戦闘機の速度に比べて十分高速な長射程の誘導弾を射撃することにより行われる戦闘を指すものとする。

1.2 戦闘空間の定義

1.2.1 シミュレーション及び行動判断の周期

シミュレーション時刻の最小単位をtickとし、あらゆる処理の周期を整数値で管理する。基準モデルにおいては、1[tick]の時間は$dt_{base} = 0.1$sとし、戦闘機やセンサ等の諸元更新は1[tick]ごとに行い、エージェントがOpenAI Gymインターフェースの外側で行動判断を行う周期n_{agent}は10[tick](= 1[Hz])とする。ただし、各エージェントが直前の行動判断結果に基づいて任意の制御量を1[tick]ごとに計算することは妨げない。

1.2.2 戦域の定義

(1) 戦域の座標系（以下「基準座標系」という。）は、NED(North-East-Down)の直交座標系とし、地球の曲率を無視するものとする。また、地表面を$z = 0$とし、$x - y$方向の原点を戦域の中心とする。

1.2.3 戦闘のルール

(1) 中心から東西にD_{line}[m]離れて引かれた南北方向の二本の直線を各陣営の防衛ラインとする。
(2) 南北方向は中心からD_{Out}[m]までを戦域とする。
(3) 戦闘の終了条件は以下の通りとし、終了時の得点が高い陣営を勝者とする。
 (a) いずれかの陣営が全滅（被撃墜または墜落）したとき
 (b) いずれかの陣営が相手の防衛ラインを突破したとき
 (c) 制限時間t_{Limit}[s]が経過したとき
 (d) いずれかの陣営の得点がp_{Disq}以下となったとき
(4) 同時に複数の終了条件が満たされた場合は、以下の優先度で一つだけが成立したものとみなす。また、終了条件(a),(b)及び(d)が成立したものとみなされた場合であっても、両陣営が同時に当該条件を満たしていた場合は終了条件(c)により終了したものとみなす。
 高 (a) > (b) > (d) >(c) 低
(5) 各陣営の得点の計算方法は以下の通りとする。
 (a) 相手を撃墜した数1機につきp_{Down}点を加算する。(撃墜された側には増減なし)
 (b) 終了条件(b)による終了となったとき、突破した陣営にp_{Break}点を加算する。
 (c) 終了条件(b)以外による終了となったとき、両陣営の最前線に位置する機体どうしを結んだ線分の中点のy座標の絶対値に応じ、より進出している陣営に1[km]あたりp_{Adv}点を加算する。
 (d) ペナルティとして、各機体が以下の条件を満たしたときにその所属陣営に減点を与える。
  - 墜落（地面に激突）したとき、p_{Down}点を減算する。
  - 各tickにおいて南北方向の場外に出ていたとき、1[s]、1[km]につきp_{Out}点を減算する。
(6) 各機体の初期状態は、以下の通りとする。
 (a) 東西方向の位置については、自陣営の防衛ライン上とする。
 (b) 南北方向の位置については、戦域中心から南北に$\pm D_{init}$[km]の位置に1機ずつとする。
 (c) 高度については、h_{init}[m]とする。
 (d) 速度については、V_{init}[m/s]とする。
 (e) 針路については、東側スタートの陣営は真西、西側スタートの陣営は真東に水平とする。

なお、学習を行う際は上記の初期条件については毎回ある程度の幅を持たせたランダムな値となるように設定することを推奨する。

1.3 戦闘機のモデル

1.3.1 運動・飛行制御モデル

1.3.1.1 運動モデル

戦闘機の運動は、横滑りを0と仮定した”Coordinated turn”が成立するものとみなして姿勢運動を以下の通り簡略化する。

(1) 燃料の消費や誘導弾の発射による質量変化は無視し、機体の質量mは一定とする。

(2) 機体の姿勢（機体座標系）は、センシングや行動判断を簡略化するために、一般的な機軸固定座標系でなく、機軸固定座標系をy軸まわりに回転させて速度ベクトルの方向がx軸に一致するようにした座標系を指すものとする。

(3) 機体に働く空気力の計算においては迎角αがパラメータとなるが、本モデルでは簡略化のために、αを直接的な制御量として瞬時に変更できるものとする。(2)と合わせることにより、αは機体の姿勢とは独立な変数として扱えるものとなる。

(4) (3)により、姿勢の変化は速度ベクトルの方向変化によるものと、ロール角変化$\Delta\phi$によるものの二種類によって生じるものとなる。ロール角は空気力の働く方向に影響するが、その変化は瞬時に行われるものとする。すなわち、各時刻の空気力を計算する際には、$\Delta\phi$だけ変化した後の姿勢を基準として空気力が計算される。ただし、状態量としての角速度の計算においては、$\Delta\phi$の変化に1tick要したものとして扱う。また、1tick中のロール角変化の上限値$\Delta\phi_{max}$は設定可能なパラメータとしている。

(5) 姿勢の表現にはクォータニオンを用いる。機体座標系成分で表示されたベクトル$\boldsymbol{v_b}$に対して左からqをかけることで慣性座標系成分で表示されたベクトル$\boldsymbol{v_i}$を得られるようなqを用いるものとする。

1.3.1.2 空気力モデル

戦闘機に働く空気力$\boldsymbol{F_A}$は、公刊文献から得られるF-16相当の性能値を用いて以下の通り計算するものとする。

$$\boldsymbol{F_A} = (C_X \boldsymbol{e_x} + C_Z \boldsymbol{e_z}) \cdot \tilde{q} S$$

$$C_X = C_x(\alpha, \delta_e)\cos\alpha + C_z(\alpha)\sin\alpha - C_{Dw}(M)$$

$$C_Z = C_z(\alpha)\cos\alpha - C_x(\alpha)\sin\alpha$$

$\delta_e = \delta_e(\alpha)$：エレベータ舵角(後述のとおり$\alpha$の関数として扱う)

$\boldsymbol{e_x}, \boldsymbol{e_z}$：機体座標系の$x$軸、$z$軸基底ベクトル

$$\tilde{q} = \frac{\rho V^2}{2}$$ ：動圧[Pa]

S：翼面積[m^2]

$C_x'(\alpha, \delta_e)$及び$C_z(\alpha)$は、[Stevens 15]に示されるF-16相当のテーブルデータを用いるが、[Stevens 15]のデータはαのみでなくエレベータ舵角δeも変数として使用しているため、本モデルでは同様に[Stevens 15]にテーブルデータが示されているピッチングモーメント係数$C_m(\alpha, \delta_e)$が0に維持されるようにδ_eが常に制御されているものとみなすことによりδ_eをαの関数として扱うことを可能とする。

また、$C_{DW}(M)$は造波抵抗を模擬する項であり、[Hendrick 08]に示される次式

$$C_{Dw}(M) = f_M C_{Dw0} \frac{k_{dw}}{(((M - k_{dwm})^2 - 1)^2 + k_{dw}^4)^{\frac{1}{4}}} \quad , \quad f_M = \frac{1}{1 + e^{-8\frac{M - (1 - \delta M/2)}{\delta M}}}$$

を用いる。ここで、$C_{Dw0}, k_{dwm}, k_{dw}, \delta M$の4個の値がパラメータとなるが、これらは$\boldsymbol{F_A}$の抵抗成分が[Webb 77]に示される各マッハ数における実際の抵抗値に近づくように数点ほどサンプリングしてカーブフィッティングした値を用いる。

1.3.1.3 エンジンモデル

戦闘機のエンジンモデルは、[Stevens 15]に示されるF-100エンジン相当の性能を有するモデルとする。本モデルではスロットルコマンド$p_{Cmd} \in [0,1)$に対して、高度hとマッハ数Mを変数とする3種のテーブルデータ(IDLE,MILITARY,MAX AB)を用いて対応する推力を計算するものである。ただし、[Stevens 15]のテーブルデータは超音速域に対応していないため、[Krus 19]に示される超音速域のMAX AB推力値を用いて拡張している。また、[Stevens 15]では推力制御の時間遅れも模擬しているが、本モデルにおいては遅れを無視し、スロットルコマンドに対して直ちに推力が追従するものとして扱う。また、本モデルにおいては燃料の消費を無視する。

１.３.１.４ 飛行制御モデル

前項までのモデル化により、本戦闘機モデルの運動を制御するための変数はスロットルコマンド$p_{Cmd} \in [0,1]$、迎角$\alpha \in [0, \alpha_{max}]$、ロール角変化$\Delta\phi \in [-\pi, \pi)$の 3 個である。これに対して 2 種類の飛行制御モデルを定義する。

(1)　直接制御モデル

エージェントが行動判断の結果として上記の制御量に相当する値を直接出力するような行動判断モデルを作成する場合に使用するものとして、直接制御モデルを定義する。

p_{cmd}については、加速目標値$c_{accel} \in [-1,1]$またはスロットルコマンド目標値$c_{throttle}$として与え、以下のように計算する。ただし、式中のp_sは現在の飛行状態において速度変化が 0 に最も近づくようなスロットルコマンドの値である。

$$p_{Cmd} = \begin{cases} p_s + (1 - p_s)c_{accel} & (c_{accel} \geq 0 \text{ のとき}) \\ p_s \cdot (1 + c_{accel}) & (c_{accel} < 0 \text{ のとき}) \\ c_{throttle} \end{cases}$$

αについては、αの最小値と最大値をそれぞれ-1 と 1 になるように線形変換させたコマンドc_{pitch}により、以下のように計算する。ただし、式中のα_sは現在の飛行状態において水平飛行を行う際の迎角値である。

$$\alpha = \min\left(\alpha_{max}, \max\left(\alpha_{min}, \alpha_s + (\alpha_{max} - \alpha_{min})c_{pitch}\right)\right)$$

$\Delta\phi$については$\Delta\phi_{max}$ で正規化した$c_{roll} \in [-1,1]$により、以下のように計算する。

$$\Delta\phi = \Delta\phi_{max} \cdot c_{roll}$$

また、結果として得られる横方向 G が上限G_{lim}を超過する場合、超過しないようにαを補正する。

(2)　方向・速度指示モデル

エージェントが行動判断の結果として「進みたい方向」と「進みたい速度」のような、制御量と直接の関連が薄いような値を出力するような行動判断モデルを作成する場合に使用するものとして、以下の通り方向・速度指示モデルを定義する。

まず、現在の進行方向を$\boldsymbol{t}$、進行方向回転軸の目標値$\boldsymbol{b}$、これらと右手系をなす直交基底を$\boldsymbol{n} = \boldsymbol{b} \times \boldsymbol{t}$とする。また、目標回転面内の角速度を$\dot{\theta}$、面外の角速度を$\dot{\epsilon}$とする。また、現在の機体座標系の$y$軸の基底ベクトルを$\boldsymbol{e_y}$としたとき、現在のロール角$\phi$を$\sin\phi = \boldsymbol{t} \cdot (\boldsymbol{b} \times \boldsymbol{e_y}), \cos\phi = \boldsymbol{b} \cdot \boldsymbol{e_y}$を満たすような$\phi$とする。（図　１．３－１）

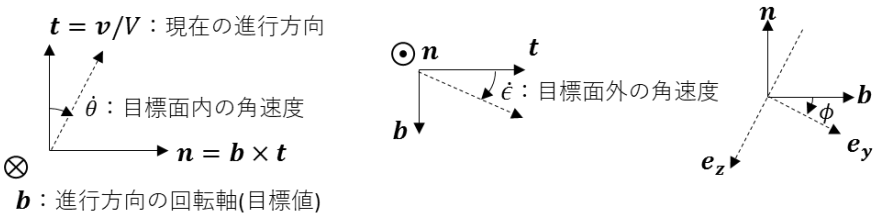

図　１．３－１　回転軸、回転角の定義

このとき、機体が受ける推力、空気力、重力の合力$\boldsymbol{F}$は

$$\begin{aligned} \boldsymbol{F} = (T(p_{cmd}) + C_x(\alpha)\tilde{q}S)(\boldsymbol{t}\cos\alpha + \boldsymbol{n}\sin\alpha\cos(\phi + \Delta\phi) + \boldsymbol{b}\sin\alpha\sin(\phi + \Delta\phi)) \\ - C_z(\alpha)\tilde{q}S(-\boldsymbol{t}\sin\alpha + \boldsymbol{n}\cos\alpha\cos(\phi + \Delta\phi) + \boldsymbol{b}\cos\alpha\sin(\phi + \Delta\phi)) + m\boldsymbol{g} \end{aligned}$$

であり、速度変化及び制御平面内外方向の角速度はそれぞれ

$$\dot{V} = \frac{\boldsymbol{F} \cdot \boldsymbol{t}}{m} = \frac{1}{m}\left\{(T + C_x\tilde{q}S)\cos\alpha + C_z\tilde{q}S\sin\alpha - C_{D_w} + m\boldsymbol{g} \cdot \boldsymbol{t}\right\}$$

$$\dot{\theta} = \frac{\boldsymbol{F} \cdot \boldsymbol{n}}{mV} = \frac{1}{mV}[\{(T + C_x\tilde{q}S)\sin\alpha - C_z\tilde{q}S\cos\alpha\}\cos(\phi + \Delta\phi) + m\boldsymbol{g} \cdot \boldsymbol{n}]$$

$$\dot{\epsilon} = \frac{\boldsymbol{F} \cdot \boldsymbol{b}}{mV} = \frac{1}{mV}[\{(T + C_x\tilde{q}S)\sin\alpha - C_z\tilde{q}S\cos\alpha\}\sin(\phi + \Delta\phi) + m\boldsymbol{g} \cdot \boldsymbol{b}]$$

となる。方向・速度指示の形式に応じて$\dot{V}, \dot{\theta}, \dot{\epsilon}$それぞれに対応する評価関数$f_V, f_\theta, f_\epsilon$を定義し、全体の評価関数を

$$f(\boldsymbol{x}) = f_V + f_\theta + f_\epsilon \quad , \quad \boldsymbol{x} = (p_{cmd}, \alpha, \Delta\phi)^T$$

とし、横方向 G 制限について不等式制約

$$g(\boldsymbol{x}) = \left(\dot{\theta}^2 + \dot{\epsilon}^2\right)V^2 - {G_{limit,fgtr}}^2 \leq 0$$

として考慮すると、

$$\text{minimize } f(\boldsymbol{x}) \text{ s.t. } g(\boldsymbol{x}) \leq 0$$

の最小化問題を解くことにより制御量が得られる。

方向の指示方法は次の 3 通りとし、この指示により回転軸の目標値$\boldsymbol{b}$と評価関数f_θ, f_ϵを定める。

(a-1) 目標進行方向$\boldsymbol{d_d}$として指定する場合

所要回転角度θ_dが最小となる回転軸は$\boldsymbol{b'} = \boldsymbol{t} \times \boldsymbol{d_d}/|\boldsymbol{d_d}|$として求まり、$\theta_d = |\boldsymbol{t} \times \boldsymbol{d_d}|$である。

ここで、θ_dが一定値$\theta_{d,thr}$以上である場合、かつ回転中のピッチ角の絶対値がある上限値γ_{limit}を超える場合、回転中の最大のピッチ角の絶対値がγとなるように以下の要領で回転軸を補正する。ただし、現在のピッチ角γ_{bef}または目標進行方向のピッチ角γ_{aft}の絶対値がγ_{limit}より大きい場合は、これらのうち最大の値を上限値として扱うものとする。

当初の回転面内でピッチ角の絶対値が最大となる方向$\boldsymbol{u}$及びそのときのピッチ角の絶対値$|\gamma_u|$は、

$$\boldsymbol{u} = \frac{\boldsymbol{b'} \times ((0,0,1)^T \times \boldsymbol{b'})}{|\boldsymbol{b'} \times ((0,0,1)^T \times \boldsymbol{b'})|}, |\gamma_u| = \sin^{-1}(|(0,0,1)^T \times \boldsymbol{b'}|)$$

であるため、$|\gamma_u|$が$\gamma' = \max(\gamma_{limit}, \max(|\gamma_{bef}|, |\gamma_{aft}|))$よりも大きく、かつ$\boldsymbol{t}$から$\boldsymbol{b}$までの回転中に$\boldsymbol{u}$または$-\boldsymbol{u}$を通過する場合に補正が必要となり、$|\gamma_u| = \gamma'$となるように補正する。

補正後の回転軸$\boldsymbol{b}$は

$$\boldsymbol{b_h} = \frac{\boldsymbol{t} \times (0,0,1)^{\mathrm{T}}}{|\boldsymbol{t} \times (0,0,1)^{\mathrm{T}}|}, \boldsymbol{b_v} = \boldsymbol{b_h} \times \boldsymbol{t}$$

とおくと、パラメータδを用いて

$$\boldsymbol{b} = \boldsymbol{h}\cos\delta + \boldsymbol{v}\sin\delta$$

と表せる。このとき、δが満たすべき式は

$$\sin|\gamma_u| = |(0,0,1)^T \times (\boldsymbol{h}\sin\delta + \boldsymbol{v}\cos\delta)| = \sin\gamma'$$

となるが、$|(0,0,1)^T \times \boldsymbol{h}| = 1$及び$|(0,0,1)^T \times \boldsymbol{v}| = \sin|\gamma_{bef}|$であることから、上式は

$$\sqrt{\cos^2\delta + \sin^2\delta \sin^2|\gamma_{bef}|} = \sin\gamma'$$

となる。これを解くと

$$\sin^2\delta = \frac{\cos^2\gamma'}{\cos^2|\gamma_{bef}|}$$

となるから、条件を満たす$\boldsymbol{b}$は

$$\boldsymbol{b} = \pm\boldsymbol{h}\sqrt{1 - \frac{\cos^2\gamma'}{\cos^2|\gamma_{bef}|}} \pm \boldsymbol{v}\sqrt{\frac{\cos^2\gamma'}{\cos^2|\gamma_{bef}|}}$$

の 4 個となる。これらのうち、$\boldsymbol{b'}$に最も近いものを補正後の目標回転軸$\boldsymbol{b}$とする。

なお、回転軸を補正すると所要回転角度は増加するが、これに関する補正は行わず、$\theta_d = |\boldsymbol{t} \times \boldsymbol{d_d}|$のままとする。この$\theta_d$を用いて、$f_\theta = k_\theta\{\dot{\theta} - \lambda_\theta \min(\theta_d, \theta_{d,clamp})\}^2$、$f_\epsilon = k_\epsilon\dot{\epsilon}^2$とする。

(a-2) 目標角速度$\boldsymbol{\omega_d}$として指定する場合

$\boldsymbol{b} = \boldsymbol{t} \times (\boldsymbol{\omega_d} \times \boldsymbol{t})/|\boldsymbol{t} \times (\boldsymbol{\omega_d} \times \boldsymbol{t})|$として、$f_\theta = k_\omega\{\dot{\theta} - \boldsymbol{b} \cdot \boldsymbol{\omega_d}\}^2$、$f_\epsilon = k_\epsilon\dot{\epsilon}^2$とする。

(a-3) 目標旋回軸$\boldsymbol{e_{y,d}}$目標迎角α_dとして指定する場合

$\boldsymbol{b} = \boldsymbol{t} \times (\boldsymbol{e_{y,d}} \times \boldsymbol{t})/|\boldsymbol{t} \times (\boldsymbol{e_{y,d}} \times \boldsymbol{t})|$として$f_\theta = k_\alpha\{\alpha - \min(\alpha_{max}, \max(\alpha_{min}, \alpha_d))\}^2$、$f_\epsilon = k_\epsilon\dot{\epsilon}^2$とする。

また、速度の指示方法は次の 4 通りとし、この指示により評価関数f_Vを定める。

(b-1) 目標速度V_dとして指定する場合

$f_V = k_V\{\dot{V} - (-\lambda_V(V - V_d))\}^2$とする。

(b-2) 目標加速度a_dとして指定する場合

$f_V = k_a(\dot{V} - a_d)^2$とする。

(b-3) 目標推力T_dとして指定する場合

$f_V = k_p\left\{p_{cmd} - \min\left(1, \max\left(0, \frac{T_d - T_{min}}{T_{max} - T_{min}}\right)\right)\right\}^2$

(b-4) 目標スロットルコマンド$p_{cmd,d}$として指定する場合

$f_V = k_p\left\{p_{cmd} - \min\left(1, \max\left(0, \frac{T(p_{cmd,d}, h, M) - T_{min}}{T_{max} - T_{min}}\right)\right)\right\}^2$とする。

参考文献：
[Stevens 15] Stevens, Brian L., et al. “Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems.” John Wiley & Sons, 2015.
[Krus 19] Krus, Petter, and Abdallah, Alvaro. “Modelling of Transonic and Supersonic Aerodynamics for Conceptual Design and Flight Simulation.” Proceedings of the 10th Aerospace Technology Congress, 2019.
[Hendrick 08] Hendrick, P., Bourdiaudhy, K., and Herbiet, J. F. “A Flight Thrust Deck for the F100 Turbofan of the F-16 Aircraft.” 26th Congress of International Council of the Aeronautical Sciences (ICAS), 2008.
[Webb 77] Webb, T. S., Kent, D. R., and Webb, J. B. “Correlation of F-16 aerodynamics and performance predictions with early flight test results.” Agard Conference Proceedings. N 242, 1977.

1.3.2 センサモデル

戦闘機の搭載センサは相手側の戦闘機を探知するレーダと、相手側の誘導弾を探知する MWS の 2 種類とする。

1.3.2.1 相手側戦闘機の探知（レーダ）

相手側戦闘機の探知については、機体正面から偏角$\theta_{FOR,radar}$以内の範囲で基準探知距離を$L_{ref,radar}$、目標の RCS スケールをσとしたとき、自機との距離Lが$L \leq L_{ref,radar} \cdot \sigma^{1/4}$以内の目標について、その真の三次元位置及び速度を遅延・欠損なく観測できるものとする。また、誤警報及び誤相関の発生は考慮しない。

1.3.2.2 相手側誘導弾の探知（MWS）

相手側誘導弾の探知については、１．４．３項で定める誘導弾センサが起動し、かつ自機を捕捉している状態のものに限り、その誘導弾の自機からみた真の方向を遅延・欠損なく観測できるものとする。

1.3.3 ネットワークによる情報共有

味方の機体の諸元は、全て真の諸元を遅延・欠損なく共有できるものとする。
相手側の戦闘機及び誘導弾の諸元は、自陣営のいずれかの機体がセンサで捉えている場合に、同等の情報を遅延・欠損なく取得できるものとする。このとき、複数の機体が同一の目標を探知できている場合は、それらの探知情報を算術平均により合成して扱うものとする。

1.3.4 人間による介入の模擬

エージェントの行動判断に対する人間による介入の模擬として、エージェントが射撃行動を出力した際に、一定時間の判断遅延を経て射撃行動の承認が行われるような介入モデルを導入する。
介入モデルは、エージェントから射撃行動の出力を認知するたび、その射撃対象と出力時刻の組を最大C_h組記憶する。記憶組数が上限に達している状態で認知した場合は無視する。また、記憶してから$\Delta t_{h,delay}$秒経過した組について、射撃行動を承認し、戦闘機モデルに射撃コマンドを送信し、記憶から消去する。また、エージェントからごく短時間で複数の射撃行動を出力されたとしても反応できないことを模擬するため、最後に射撃行動の出力を認知、記憶した時点から$\Delta t_{h,cooldown}$経過するまでは新たな射撃行動の出力を認知できないものとする。

1.3.5 武装モデル

戦闘機モデルは１．４項に示す誘導弾を１機あたりN_{msl}発搭載しているものとし、１．３．４項に示す人間介入モデルから射撃コマンドが送信された場合直ちに誘導弾の発射処理を行うものとする。

1.3.6 戦闘機に関する観測可能な情報

本モデルにおいて、エージェントが観測してよい情報(observables)は次の通りとする。

表 1.3-1 戦闘機に関する観測可能な情報(json として表現)(1／2)

キー名(インデントは階層を表す)	本文中の記号	型	概要
isAlive		bool	生存中か否か
spec		object	性能に関する値（戦闘中不変）
dynamics		object	運動性能に関する値
rollMax	$\Delta\phi_{max}$	double	ロール角速度の上限値[rad/s]
weapon		object	武装に関する値
numMsls	N_{msl}	unsigned int	初期誘導弾数
stealth		object	被探知性能に関する値
rcsScale	σ	double	RCS スケール(無次元量として扱う)
sensor		object	センサに関する値
radar		object	レーダに関する値
Lref	$L_{ref,radar}$	double	基準探知距離[m]
thetaFOR	$\theta_{FOR,radar}$	double	探知可能覆域[rad]
mws		object	MWS に関する値　※該当なし
motion		object	現在の運動状態に関する値。後述する MotionState クラスの json 表現である。
sensor		object	現在の探知状況に関する値
radar		object	自機レーダの探知状況に関する値
track		array(object)	自機レーダが探知した 3 次元航跡のリスト。各要素は後述する Track3D クラスの json 表現である。
track		array(object)	編隊内で共有し統合されたレーダ航跡のリスト。各要素は後述する Track3D クラスの json 表現である。
trackSource		array(array(string))	編隊内で共有し統合された 3 次元航跡それぞれの、統合元となった機体名のリストのリスト。
mws		object	自機 MWS の探知状況に関する値
track		array(object)	自機 MWS が探知した 2 次元航跡のリスト。各要素は後述する Track2D クラスの json 表現である。

表 1.3-1 戦闘機に関する観測可能な情報(json として表現)(2／2)

キー名(インデントは階層を表す)	本文中の記号	型	概要
weapon		object	現在の武装状況に関する値
remMsls		unsigned int	現在の残弾数
nextMsl		unsigned int	次に射撃する誘導弾の ID
launchable		bool	現在射撃可能な状態か否か。残弾数が 0 でなく、かつ人間介入モデルの記憶容量が上限に達していない場合に可となる。
missiles		array(object)	各誘導弾に関する observables の配列。その内訳は後述する誘導弾モデルに記載のとおり。
shared		object	味方機の observables。自身を含む各機の名称をキーとして各機の observable の一部を格納。含まれないものは以下の 3 要素である。 ・/shared 以下全て ・/sensor/track ・/sensor/trackSource

1.3.7 戦闘機に関する制御情報

本モデル全体として、エージェントから受け取る制御情報コマンドは次の通りとする。

表 1.3-2 戦闘機に関するエージェントからの制御情報(json として表現)

キー名(インデントは階層を表す)	本文中の記号	型	概要
motion		object	運動に関する値
dstV	V_d	double	目標速度[m/s]
dstAccel	a_d	double	目標加速度[m/s^2]
dstThrust	T_d	double	目標推力[N]
dstThrottle	$p_{cmd,d}$	double	目標スロットルコマンド
dstDir	$\boldsymbol{d_d}$	array(double)	目標進行方向(単位ベクトル)
dstTurnRate	$\boldsymbol{\omega_d}$	array(double)	目標角速度[rad/s]
dstAlpha	α_d	double	目標迎角[rad]
ey	$\boldsymbol{e_{y,d}}$	array(double)	目標とする機体y軸方向(単位ベクトル)
weapon		object	射撃に関する値
launch	－	bool	射撃するか否か
target	－	object	射撃する対象の 3 次元航跡の json 表現

1.4 誘導弾のモデル

1.4.1 運動・飛行制御モデル

誘導弾の性能については、[Ekker 94]及び[Redmon 80]に記載されている AMRAAM 相当の値を用いるものとする。

1.4.1.1 空気力モデル

誘導弾に働く空気力は[Ekker 94]に記載されているモデルを用いるが以下の 2 点の改変を加える。
(1) 全て MKS 単位系(角度は rad)を用いるものとする。
(2) 遷音速域($M = 0.95\sim1.2$)について一部の関数が適用対象外となるが、モデルを簡略化するために、亜音速域または超音速域の計算方法をそのまま準用して計算するものとする。

1.4.1.2 推力モデル

誘導弾の推力モデルは、射撃後に燃焼時間t_{burn}が経過するまでの間は一定の推力T_{msl}を発生し、それ以降の推力は 0 とする単純なモデルとする。また、燃焼による誘導弾の質量変化は無視する。t_{burn}及びT_{msl}の値は[Ekker 94]及び[Redmon 80]に基づき、推進剤の比推力I_{sp}と、参照高度h_{boost}における最高到達マッハ数M_{boost}から求めたものを用いる。

1.4.1.3 運動モデル

誘導弾の運動は以下の通り簡略化したものを用いる。
(1) 誘導弾の姿勢（座標系）は、戦闘機モデルと同様に、弾体軸固定座標系をy軸まわりに回転させて速度ベクトルの方向がx軸に一致するようにした座標系を指すものとする。姿勢表現は戦闘機モデルと同様とする。
(2) 誘導弾の姿勢運動について、x軸周りの回転は無視する。また、y,z軸周りの回転は直ちに所要の姿勢をとれるものとして扱う。
(3) (2)により、迎角αと舵角δを直接制御量として扱うものとする。ただし、1.4.1.1項の空気力モデルにおいてαとδにより生じるピッチングモーメントが 0 に釣り合うことを制約条件とする。

1.4.1.4 飛行制御モデル

誘導弾の飛行制御は、目標の航跡情報(位置及び速度)に基づき、ゲインGの単純な比例航法により計算した必要角速度を実現するために必要な横力を生じさせるαとδの組を計算することによって行うものとする。また、横方向加速度には上限$G_{limit,msl}$を設ける。

参考文献：
[Ekker 94] Ekker, David A. Missile Design Toolbox. Diss. Monterey, California. Naval Postgraduate School, 1994.
[Redmon 80] Redmon, Danny Ray. Tactical Missile Conceptual Design. Naval Postgraduate School Monterey CA, 1980.

1.4.2 最大飛翔時間及び命中判定

誘導弾の有効飛翔時間は$t_{M,max}$秒間とし、その時間内に自陣営以外のいずれかの戦闘機との距離がd_{hit}以下となった場合には起爆するものとし、その時点で命中半径内に存在している全ての自陣営以外の戦闘機に対し撃墜判定を与えるものとする。また、燃焼終了後に飛翔速度が$V_{M,min}$を下回った場合は、飛翔時間が残っていたとしても当該誘導弾を無効化するものとする。

1.4.3 センサモデル

誘導弾自身の諸元は真の値を遅延・欠損なく取得できるものとする。目標機の航跡情報は発射母機が保持している場合は発射母機と同一の情報を遅延・欠損なく取得できるものとする。

自身の保持している目標情報との距離がd_{Ms}以内となった場合、シーカを起動するものとし、以後は発射母機からの航跡供給の有無によらず、目標が自身の覆域内（正面から偏角$\theta_{FOR,seeker}$以内の範囲）に存在し、かつ目標の推定方向を中心とした視野内（偏角$\theta_{FOV,seeker}$以内の範囲）に存在している場合に目標の真の諸元を遅延・欠損なく取得できるものとする。また、覆域の計算においては簡略化のために迎角の影響は無視し、その中心が常に進行方向と一致するものとする。

発射母機とシーカのいずれも目標を探知できていない場合は、目標が等速直線運動をしていると仮定して外挿を行い、誘導を継続するものとする。

1.4.4 誘導弾に関する観測可能な情報

本モデルにおいて、エージェントが観測してよい情報(observables)は次の通りとする。

表 1.4-1 誘導弾に関する観測可能な情報(json として表現)

キー名	本文中の記号	型	概要
isAlive		bool	生存中か否か
hasLaunched		bool	発射済みか否か
launchedT		double	発射時刻。未発射の場合は-1
mode		string	目標の追尾状況であり、以下の 3 種類のいずれか。 “guided“・・・母機から供給された航跡を使用 “self”・・・自身のセンサで捉えた航跡を使用 “memory”・・・メモリトラックにより外挿中
target		object	目標の 3 次元航跡。後述する Track3D クラスの json 表現である。
motion		object	現在の運動状態に関する値。後述する MotionState クラスの json 表現である。

1.5 航跡及び運動状態の表現

1.5.1 3 次元航跡の表現

本シミュレータにおいて 3 次元航跡を表す Track3D クラスは、慣性系での位置、速度及び生成時刻を保持するものとして扱う。また、航跡は必ずそれがどの Asset を指したものかを示す情報を付加し、誤相関は発生しないものとして扱う。3 次元航跡を json 化した際の表現は次の通りとする。

表 1.5-1 Track3D クラスの json 表現

キー名	本文中の記号	型	概要
truth		str	この 3 次元航跡が指す対象の Asset を特定する UUID(バージョン 4)を表す文字列。
time		array(double)	この航跡を生成した時刻
pos		array(double)	位置ベクトル(慣性系)
vel		array(double)	速度ベクトル(慣性系)
buffer		array(object)	この 3 次元航跡と同一の対象を指すものとして外部から追加された 3 次元航跡のリスト。merge 関数によって平均値をとる際に用いられる。

1.5.2 2次元航跡の表現

本シミュレータにおいて2次元航跡を表すTrack2Dクラスは、慣性系での観測点の位置、観測点から見た目標の方向及び角速度並びに生成時刻を保持するものとして扱う。また、航跡は必ずそれがどのAssetを指したものかを示す情報を付加し、誤相関は発生しないものとして扱う。2次元航跡をjson化した際の表現は次の通りとする。

表 1.5-2 Track2Dクラスのjson表現

キー名	本文中の記号	型	概要
truth		str	この2次元航跡が指す対象のAssetを特定するUUID(バージョン4)を表す文字列。
time		array(double)	この航跡を生成した時刻
dir		array(double)	方向ベクトル(慣性系)
origin		array(double)	観測者の位置ベクトル(慣性系)
omega		array(double)	角速度ベクトル(慣性系)
buffer		array(object)	この2次元航跡と同一の対象を指すものとして外部から追加された2次元航跡のリスト。merge関数によって平均値をとる際に用いられる。

1.5.3 運動状態の表現

本シミュレータにおいて一般的な運動状態はMotionStateクラスにより表現するものとし、位置、速度、姿勢、角速度及び生成時刻を保持するものとして扱う。また、姿勢に関する追加情報として、方位角、ピッチ角の情報と、方位角をそのままに$x-y$平面を水平面と一致させた座標系の情報を付加するものとする。

また、基本的なobservablesとして得られるものは慣性系における値とし、内部の状態量として使用するものは親Assetの座標系における値とする。運動状態をjson化した際の表現は次の通りとする。

表 1.5-3 MotionStateクラスのjson表現

キー名	本文中の記号	型	概要
pos		array(double)	位置ベクトル
vel		array(double)	速度ベクトル
omega		array(double)	角速度ベクトル
q		array(double)	現在の姿勢。クォータニオンを実部⇒虚部の順に並べた4次元ベクトルとして記述。
qh		array(double)	現在の方位をx軸正方向として$x-y$平面を水平にとった座標系。クォータニオンを実部⇒虚部の順に並べた4次元ベクトルとして記述。
az		double	現在の方位角(真北を0として東側を正)
el		double	現在のピッチ角(下向きを正)
time		double	このMotionStateを生成した時刻

1.6 地球モデル

本シミュレータにおいて、地形については模擬せず、地球の曲率も無視する。また、重力場は重力加速度$g = 9.8[\mathrm{m/s^2}]$の一様場とする。大気についてはISO 2533:1975で定められている標準大気モデルを用いるものとする。

1.7 基準パラメータ

前項までで定義した空対空目視外戦闘環境の設定パラメータ及び基準値の一覧は以下に示す通りである。

表 1．7-1　戦闘のルールに関する基準パラメータ

実装クラス	本文書中の記号	基準シミュレータにおける変数名	意味	基準値	単位
SimulationManager	dt_{base}	baseTimeStep	シミュレーション時刻の最小単位（=1tick）	0.1	s
	n_{agent}	agentInterval	エージェントの行動判断の周期	10	tick
Ruler	t_{Limit}	maxTime	最大戦闘時間	1200	s
R3BVRRuler01	D_{line}	dLine	戦域中心から防衛ラインまでの距離	100000	m
	D_{out}	dOut	戦域中心から戦域の南北端までの距離	75000	m
	H_{limit}	hLim	戦域の高度上限	20000	m
	—	westSider	西側からスタートする陣営の名称	“Red”	—
	—	eastSider	東側からスタートする陣営の名称	“Blue”	—
	p_{Disq}	pDisq	減点過剰により失格とする点数	-100	—
	p_{Break}	pBreak	相手の防衛ラインを突破したときに得る得点	10	—
	p_{Down}	pDown	相手を 1 機撃墜するごとにえる得点　及び 自陣営の機体が 1 機墜落するごとに失う得点	5	—
	p_{Adv}	pAdv	双方の進出度合いに応じ優勢側が得る加点量	0.1	km^{-1}
	p_{Out}	pOut	南北方向の戦域逸脱度合いに応じた減点量	0.1	$(km \cdot s)^{-1}$
Fighter (instanceConfig として指定)	D_{init}	—	各機の南北方向の初期位置(中央からの距離)	10000	m
	h_{init}		各機の初期高度	10000	m
	V_{init}		各機の初期速度	300	m/s
	—		各機の針路方位(真北を 0 とし東側を正)	270/90	deg

表 1．7-2　戦闘機モデルに関する基準パラメータ(1／2)

実装クラス	本文書中の記号	基準シミュレータにおける変数名	意味	基準値	単位
Fighter	σ	rcsScale	RCS スケール	1	—
	N_{msl}	numMsls	搭載誘導弾数	10	—
Fighter:: HumanIntervention	C_h	capacity	人間介入モデルの記憶可能な射撃行動出力の組数	1	組
	$\Delta t_{h,delay}$	delay	人間介入モデルの射撃行動の承認までの遅延時間	3	s
	$\Delta t_{h,cooldown}$	cooldown	人間介入モデルの射撃行動出力を認知できる間隔	0.999	s
MassPointFighter	$a_{F,min}$	aMin	最小加速度	-1	G
	$a_{F,max}$	aMax	最大加速度	1	G
	$\omega_{F,x,max}$	rollMax	最大角速度（ロール軸）	15	deg/s
	$\omega_{F,y,max}$	pitchMax	最大角速度（ピッチ軸）	15	deg/s
	$\omega_{F,z,max}$	yawMax	最大角速度（ヨー軸）	15	deg/s
	$V_{F,min}$	vMin	最小速度	150	m/s
	$V_{F,max}$	vMax	最大速度	300	m/s
CoordinatedFighter	m	m	質量	20,500	lbs
	S	S	主翼面積	300	ft^2
	$\Delta\phi_{max}$	rollMax	1tick 中のロール角変化の上限値	180	deg
	$G_{limit,fgtr}$	sideGLimit	横方向 G 制限	10	G
	α_{min}	minAoa	迎角の最小値	-10	deg
	α_{max}	maxAoa	迎角の最大値(揚力が単調増加となる範囲で設定)	36.35	deg
	$c_x(\alpha, \delta_e)$	cxTable	機軸固定座標系x軸方向の空気力のテーブル	省略	
	$c_m(\alpha, \delta_e)$	cmTable	機軸固定座標系y軸方向の空気力によるモーメントのテーブル	省略	
	$c_z(\alpha)$	czTable	機軸固定座標系z軸方向の空気力のテーブル	省略	
	$C_{Dw0}, k_{dwm}, k_{dw}, \delta M$	cdwTable	造波抵抗のパラメータをベクトルとして並べたもの	省略	
	—	aoaTable	テーブルデータのα軸の参照点	省略	deg
	—	deTable	テーブルデータのδ_e軸の参照点	省略	deg

表 1.7-2 戦闘機モデルに関する基準パラメータ(2／2)

実装クラス	本文書中の記号	基準シミュレータにおける変数名	意味	基準値	単位
Propulsion	—	tminTable	IDLE 時($p_{cmd}=0$)の推力テーブル	省略	
	—	tmilTable	MILITARY 時($p_{cmd}=0.5$)の推力テーブル	省略	
	—	tmaxTable	MAX AB 時($p_{cmd}=1$)の推力テーブル	省略	
	—	alts	推力テーブルの高度軸の参照点	省略	
	—	machs	IDLE,MILITARY 時の推力テーブルのマッハ数軸の参照点	省略	
	—	machsEx	MAX AB 時の推力テーブルのマッハ数軸の参照点	省略	
CoordinatedFighter:: FlightController	λ_V	lambdaVel	速度ゲイン	1	
	λ_θ	lambdaTheta	角度ゲイン	1	
	$\theta_{d,clamp}$	clampTheta	角度差フィードバックのカットオフ上限	45	deg
	k_V	kVel	速度に関する評価関数の重み	10	
	k_a	kAccel	加速度に関する評価関数の重み	10	
	k_p	kPower	推力に関する評価関数の重み	20	
	k_θ	kTheta	目標角度差に関する評価関数の重み	10,000	
	k_ω	kOmega	面内角速度に関する評価関数の重み	10,000	
	k_α	kAoa	迎角に関する評価関数の重み	10,000	
	k_ϵ	kVel	面外角速度に関する評価関数の重み	0.1	
	γ_{limit}	pitchLimit	ピッチ角制限値	0	deg
	$\theta_{d,thr}$	pitchLimitThreshold	ピッチ角制限の対象とする下限旋回角度	0	deg
AircraftRadar	$L_{ref,radar}$	Lref	戦闘機搭載センサの基準探知距離	100,000	m
	$\theta_{FOR,radar}$	thetaFOR	戦闘機搭載センサの覆域	90	deg

表 1.7-3 誘導弾モデルに関する基準パラメータ

実装クラス	本文書中の記号	基準シミュレータにおける変数名	意味	基準値	単位
Missile	—	mass	質量	339	lbs
	λ_θ	tMax	最大飛翔時間	150	s
	d_{hit}	hitD	命中判定距離	300	m
	I_{sp}	Isp	比推力	260	s
	G_{boost}	boostMaxG	推力計算に用いる加速度	30	G
	M_{boost}	boostMaxM	推力計算に用いる到達マッハ数	4	
	h_{boost}	boostAlt	推力計算に用いる飛翔高度	10000	ft
	—	length	全長	12	ft
	—	lcg	重心位置(先端から)	6.5553	ft
	—	diameter	直径	0.6	ft
	—	lengthN	ノーズの長さ	1.2	ft
	—	thicknessRatio	翼厚比	0.04	
	—	spanW	主翼スパン	2.5817	m
	—	spanT	尾翼スパン	1.2950	m
	—	locationW	主翼位置(先端から)	5.5767	m
	—	areaW	主翼面積	2.3798	m^2
	—	areaT	尾翼面積	0.5988	m^2
	—	maxA	迎角上限	30	deg
	—	maxD	舵角上限	30	deg
	$G_{limit,msl}$	maxLoadG	横方向 G 制限	30	G
	$V_{M,min}$	minV	飛翔下限速度	150	m/s
PropNav	G	gain	比例航法ゲイン	15	
MissileSensor	$L_{ref,seeker}$	Lref	誘導弾搭載センサの基準探知距離	10,000	m
	$\theta_{FOR,seeker}$	thetaFOR	誘導弾搭載センサの覆域	60	deg
	$\theta_{FOV,seeker}$	thetaFOV	誘導弾搭載センサの視野角	15	deg

2 本シミュレータの構成・使用方法

本シミュレータの構成及び使用方法について説明する。

2.1 ディレクトリ構成

simulator.zip として配布される本シミュレータのディレクトリ構成は以下の通り。
root

```
├addons：本シミュレータの追加機能として実装されたサブモジュール
│ ├AgentIsolation：行動判断モデルをシミュレータ本体と隔離して動作させるための評価用機能
│ └rayUtility：ray RLlib を用いて学習を行うための拡張機能
├ASRCAISim1：本シミュレータの Python モジュールを構成する一式(一部はビルド時に生成)
├include：コア部を構成するヘッダファイル
├sample：戦闘環境を定義し学習を行うためのサンプル
│ ├config：サンプル用のコンフィグファイル
│ ├MinimumEvaluation：各エージェントを隔離して対戦させる最小限の評価環境
│ ├OriginalModelSample：独自の Agent クラスと報酬クラスを定義するためのサンプル
│ └Standard：初期行動判断モデルを対戦相手として基本的な強化学習を行うためのサンプル
├src：コア部を構成するソースファイル
└thirdParty：外部ライブラリの格納場所(同梱は改変を加えたもののみ)
  └include
    └pybind11_json　※オリジナルを改変したものを同梱
```

2.2 環境の構築

2.2.1 環境構築手順

2.2.1.1 手法 1 Ubuntu ベースのコンテナ環境を構築

simulator.zip を解凍し、解凍した simulator の root ディレクトリに Dockerfile を置き、以下を実行する。

```
docker build .
```

この環境が提出されたエージェント同士の対戦環境にもなっている。
コンテナ環境の中に入り、以下動作確認用サンプルを実行し問題なければ成功である。

```
cd sample/Standard
python FirstSample.py
```

上記コンテナ環境は GUI 非対応であるため、後に示す、戦闘画面の可視化を行いたい場合は手法 2 を推奨する。

2.2.1.2 手法 2 ホスト OS 上で直接環境構築（Linux, macOS） ※推奨

Linux の場合は py37_linux_whl.zip、macOS の場合は py37_mac_whl.zip を解凍し、2 つの whl ファイルが存在することを確認する。
Python3.7 がインストールされた仮想環境等にて、以下を実行する。

```
pip install "ray[default, tune, rllib]==1.9.1"
pip install ASRCAISim1-1.0.0-py3-none-any.whl
pip install OriginalModelSample-1.0.0-py3-none-any.whl
```

nlopt をインストールする。

```
curl -sSL https://github.com/stevengj/nlopt/archive/v2.6.2.tar.gz | tar xz
cd nlopt-2.6.2 && cmake . && make && make install
```

nlopt の共有オブジェクトの場所を指定する。Linux の場合は、
```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```
macOS の場合は、
```
export LIBRARY_PATH=$LIBRARY_PATH:/usr/local/lib
```
により指定できる。
simulator.zip を解凍し、以下動作確認用サンプルを実行し問題なければ成功である。
```
cd sample/Standard
python FirstSample.py
```

2.2.1.3 手法 2 ホスト OS 上で直接環境構築（Windows, MSVC）　※非推奨

py37_win_msvc_whl.zip を解凍し、2 つの whl ファイルが存在することを確認する。
Python3.7 がインストールされた仮想環境等にて、以下を実行する。
```
pip install "ray[default, tune, rllib]==1.9.1"
pip install ASRCAISim1-1.0.0-py3-none-any.whl
pip install OriginalModelSample-1.0.0-py3-none-any.whl
```

nlopt をインストールする。
```
mkdir c:¥simulator;cd simulator
curl -sSL -o nlopt-2.6.2.tar.gz https://github.com/stevengj/nlopt/archive/v2.6.2.tar.gz
tar -xzf nlopt-2.6.2.tar.gz
cd nlopt-2.6.2
mkdir build;cd build
cmake ..
```
Visual Studio を起動して、nlopt.sln を開いて Release モードでビルドする。

システム環境変数 Path に、以下の nlopt.dll へのパスと ASRCAISim1 の Core.dll へのパス（自身のパスに読み替えること）を追加する。
C:¥simulator¥nlopt-2.6.2¥build¥Release
C:¥Users¥...¥lib¥site-packages¥ASRCAISim1

simulator.zip を解凍し、以下動作確認用サンプルを実行し問題なければ成功である。
```
cd sample/Standard
python FirstSample.py
```

2.2.1.4 手法 2 ホスト OS 上で直接環境構築（Windows, MSYS）　※非推奨

py37_win_msys_whl.zip を解凍し、2 つの whl ファイルが存在することを確認する。
Python3.7 がインストールされた仮想環境等にて、以下を実行する。
```
pip install "ray[default, tune, rllib]==1.9.1"
pip install ASRCAISim1-1.0.0-py3-none-any.whl
pip install OriginalModelSample-1.0.0-py3-none-any.whl
```

nlopt をインストールする。
```
mkdir c:¥simulator;cd simulator
curl -sSL -o nlopt-2.6.2.tar.gz https://github.com/stevengj/nlopt/archive/v2.6.2.tar.gz
tar -xzf nlopt-2.6.2.tar.gz
cd nlopt-2.6.2
mkdir build;cd build
cmake .. -G "MSYS Makefiles"
make; make install
```

システム環境変数 Path に、以下の nlopt.dll へのパスと ASRCAISim1 の Core.dll へのパス（自身のパスに読み替えること）を追加する。
C:¥simulator¥nlopt-2.6.2¥build¥Release
C:¥Users¥...¥lib¥site-packages¥ASRCAISim1

simulator.zip を解凍し、以下動作確認用サンプルを実行し問題なければ成功。
```
cd sample/Standard
python FirstSample.py
```

2.2.2 動作確認済環境(主要な項目のみ)

本シミュレータは、以下の環境で動作確認を行っている。
- OS：Ubuntu 20.04 LTS
- Python 3.7.12
- Tensorflow：2.6.2
- PyTorch：1.9.1+cpu
- ray：1.9.1
- gcc：11.2.0(>=9)
- Eigen：3.4.0 (>=3.3.9)
- pybind11：2.8.0 (>=2.6.2)
- nlohmann’s json：3.9.1
- pybind11_json：0.2.11(を改変)
- NLopt：2.6.2
- Magic Enum C++：0.7.3
- Boost：1.65.1
- CMake：3.21 (>=3.12)

なお、C++コンパイラについては C++17 の機能を使用しているため、以下の通りのバージョン要求がある。(以下は最も要求バージョンが厳しい Magic Enum C++による。)
- clang/LLVM >= 5
- MSVC++ >= 14.11 / Visual Studio >=2017
- Xcode >= 10
- gcc >= 9
- MinGW >= 9

2.2.3 深層学習用フレームワークのインストール

本シミュレータは純粋な OpenAI Gym 環境ではなく、マルチエージェントのシミュレーションを前提としていることから ray RLlib を強化学習フレームワークとして想定しており、RLlib の MultiAgentEnv クラスにインターフェースを合わせている。２．２．１項における本シミュレータのインストール作業において、依存ライブラリとして RLlib を指定しているため RLlib は自動的にインストールされるが、深層学習フレームワークについては自動的にはインストールされない。
RLlib は Tensorflow または PyTorch のいずれかを使用することを前提に実装されており、本シミュレータの学習サンプルは RLlib による学習を実施するため、どちらもインストールされていない場合は別途インストールすることを推奨する。

2.2.4 動作確認用サンプル

root/sample/Standard 配下の、動作確認用サンプルについて説明する。

2.2.4.1 シミュレータとしての動作確認用サンプル

FirstSample.py は、ルールベースの初期行動判断モデル同士の対戦が実行される。
```
cd sample/Standard
```

```
python FirstSample.py
```
で実行される。なお、初期行動判断モデルは Observation や Action の入出力は伴わない。

2.2.4.2 OpenAI gym 環境としての I/F 確認用サンプル

SecondSample.py では、サンプルの Agent モデル２種（Blue 側は 1 機ずつ行動、Red 側は２機分一纏めに行動）同士のランダム行動による対戦が実行される。

```
cd sample/Standard
python SecondSample.py
```
で実行される。本サンプルにより、Observation や Action の定義例を確認できる。

2.3 任意の陣営間の対戦の実行（陣営ごとに Agent や Policy を隔離した状態で対戦）

陣営ごとに Agent や Policy の動作環境や設定ファイルをパッケージ化し、互いに隔離した状態で対戦させて評価を行うことを可能にするための機能を root/sample/MinimumEvaluation 及び root/addons/AgentIsolation に実装している。

2.3.1 Agent 及び Policy のパッケージ化の方法

Agent と Policy の組を一意に識別可能な名称をディレクトリ名とし、__init__.py を格納して Python モジュールとしてインポート可能な状態とし、インポートにより以下の 4 種類の関数がロードされるような形で実装されていれば、細部の実装方法は問わないものとする。

(1) getUserAgentClass()･･･Agent クラスオブジェクトを返す関数
(2) getUserAgentModelConfig()･･･Agent モデルの Factory への登録用に modelConfig を表す json(dict)を返す関数
(3) isUserAgentSingleAsset()･･･Agent の種類(一つの Agent インスタンスで 1 機を操作するのか、陣営全体を操作するのか)を bool で返す関数(True が前者)
(4) getUserPolicy()･･･３．４項で示す StandalonePolicy を返す関数

2.3.2 パッケージ化された Agent 及び Policy の組を読み込んで対戦させるサンプル

Agent 及び Policy を隔離しない場合のサンプルを root/sample/MinimumEvaluation/evaluator.py に実装しており、run 関数の引数として Agent・Policy パッケージの識別名を二つ与えることで、それらを読み込んで互いに対戦させることができる。

また、Agent 及び Policy を隔離する場合は、sep_config.json に Blue と Red のそれぞれに対応する edge の IP アドレス、ポート名、Agent・Policy パッケージの識別名を記述しておく。設定例は以下。

```
{
    "blue":{
        "userID":"RuleBased",
        "server":"localhost",
        "agentPort":51000,
        "policyPort":51001
    },
    "red":{
        "userID":"User001",
        "server":"localhost",
        "agentPort":51002,
        "policyPort":51003
    },
    "seed":null
}
```

次に、Blue を動かす edge 環境において python sep_edge.py blue を、Red を動かす edge 環境において python sep_edge.py red をそれぞれ実行した後に、center 環境において python sep_main.py を実行することで対戦させることが可能となっている。

なお、本サンプルでは戦闘場面は以下に示す BVR2v2_rand.json を root/sample/MinimumEvaluation に格納して読み込むこととしているため、異なる場面の対戦を実行したい場合は適切に変更されたい。

表 2.3-1 戦闘場面指定の config 例

パス	概要
root/sample/config/ BVR2v2.json	固定された初期条件で Red と Blue の 2 対 2 の対戦場面を設定する例
root/sample/config/ BVR2v2_rand.json	ランダムな初期条件で Red と Blue の 2 対 2 の対戦場面を設定する例

2.3.3 Agent 及び Policy のパッケージ化のサンプル

Agent 及び Policy のパッケージ化を行うサンプルとして root/sample/MinimumEvaluation に以下に示す 4 種類を格納している。

表 2.3-2 Agent 及び Policy のパッケージ化を行うサンプルの一覧

識別名	概要	4 種類の関数が返す値	
RuleBased	初期行動判断モデル	getUserAgentClass	R3InitialFighterAgent01
		getUserAgentModelConfig	config.json を読み込み
		isUserAgentSingleAsset	True
		getUserPolicy	None を返すダミーPolicy
User001	1 体で 1 機を操作する Agent モデル 及び ランダム行動をとる Policy	getUserAgentClass	R3PyAgentSample01
		getUserAgentModelConfig	config.json を読み込み
		isUserAgentSingleAsset	True
		getUserPolicy	ランダム行動を返すダミーPolicy
User002	1 体で陣営全体を操作する Agent モデル 及び ランダム行動をとる Policy	getUserAgentClass	R3PyAgentSample02
		getUserAgentModelConfig	config.json を読み込み
		isUserAgentSingleAsset	False
		getUserPolicy	ランダム行動を返すダミーPolicy
User003	1 体で 1 機を操作する Agent モデル 及び ray で学習された Policy	getUserAgentClass	R3PyAgentSample01
		getUserAgentModelConfig	config.json を読み込み
		isUserAgentSingleAsset	True
		getUserPolicy	trainer_config.json 及び weights.dat を読み込んで生成された VTraceTorchPolicy を実体として持つ StandaloneRayPolic

2.4 戦闘画面の可視化方法

戦闘場面の可視化を行う例として、pygame と OpenGL を用いた単純な可視化を行う GodView クラスを root/ASRCAISim1/viewer/GodView.py に Python クラスとして実装している。図 2.4－1 に示すように、戦域を真上から見た図平面図と、真南から見た鉛直方向の断面図が表示される。また、画面の左上には時刻、得点、報酬の状況が表示される。

なお、現在の実装では 1 枚のサーフェスのまま強引に 2 種類の図を描画しているため、一方の描画範囲外になったオブジェクトがもう一方の図にはみ出て描画されてしまうことがある。

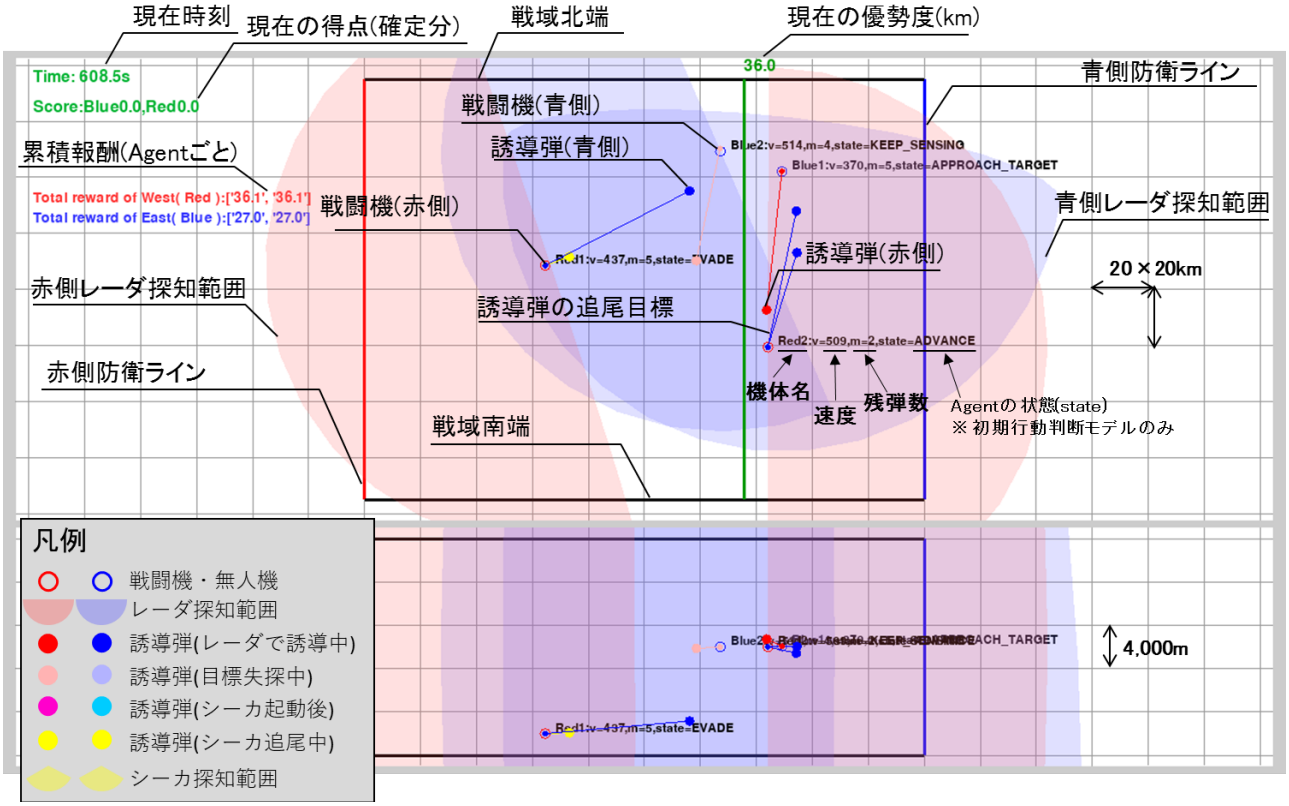

図　２．４－２　GodView の画面表示例(上側が平面図、下側が側面図)

例えば root/sample/MinimumEvaluation 配下のサンプルでは、sep_main.py の config を"ViewerType": "God"と書き換えることで戦闘画面が可視化される。また、２．２．４項で使用した二つのサンプルでも同様の変更で可視化が可能である。

```
    configs = [
        os.path.join(os.path.dirname(__file__), "common/BVR2v2_rand.json"),
        agentConfig,
        {
            "Manager": {
                "Rewards": [],
                "seed":seed,
                "ViewerType":"God",
                "Loggers":{
                }
            }
        }
    ]
```

なお、戦闘画面可視化の際、OpenGL に関して以下のエラーが発生することが確認されている（macOS Big Sur にて確認）。

```
Warning: Godview was not imported collectly. The exception occured is,
('Unable to load OpenGL library', 'dlopen(OpenGL, 10): image not found', 'OpenGL', None)
To continue the simulation, a dummy no-op Viewer will be used instead.
```

この場合、import している OpenGL の OpenGL/platform/ctypesloader.py のコードを以下の通り修正する。

```
（修正前）
fullName = util.find_library( name )

（修正後）
fullName = '/System/Library/Frameworks/OpenGL.framework/OpenGL'
```

また、GodViewStateLogger を用いて戦闘場面の可視化に必要な情報をログとして出力しておき、後から SimulationManager と独立に動作可能な可視化用クラスの例として、GodViewStateLoader クラスを root/ASRCAISim1/viewer/GodViewLoader.py に実装している。

root/sample/MinimumEvaluation 配下のサンプルでは、sep_main.py の config の"Loggers"を以下の通り書き換えることでログが出力される。

```
                "Loggers":{
                    "GodViewStateLogger":{
                        "class":"GodViewStateLogger",
                        "config":{
                            "prefix":"./results/GodViewStateLog",
                            "episodeInterval":1,
                            "innerInterval":1
                        }
                    }
                }
```

出力したログを元に戦闘画面を再度可視化するには、root/sample/MinimumEvaluation/replay.py の第一引数としてログの.dat ファイルを、第二引数として動画や連番画像を保存するファイルパス名 prefix（以下の例の場合./movies/movie_e0001.mp4 として保存される）を指定して実行することで、可視化される。

```
python replay.py "./results/GodViewStateLog_YYYYMMDDhhrrss_e0001.dat" "./movies/movie"
```

3 本シミュレータの機能・処理の流れ

本シミュレータの機能・処理の流れについて説明する。

本シミュレータは、内部の処理を C++で記述したものを pybind11 経由で Python モジュールとして公開する形で作成されている。

3.1 クラス、モデル、ポリシーの考え方及び Factory によるインスタンス生成

本シミュレータにおいて、クラス、モデル、ポリシーについて以下のように定義するものとする。

クラス　…プログラム上で実装されたクラスそのもの。
　　(例：レーダクラス、誘導弾クラス)
モデル　…あるクラスに対して、特定のパラメータセットを紐付けたもの
　　(例：50km 見えるレーダ、100km 見えるレーダ)
ポリシー…与えられた Observation に対して対応する Action を計算するもの
　　(例：ニューラルネットワーク、ルールベースなど)

このうち、クラスとモデルは本シミュレータの内部で管理されるものであり、ポリシーは外部の、例えば強化学習フレームワークによって管理されるものである。

ある登場物のインスタンスを生成する際にはクラスまたはモデルを指定することで行うものとし、本シミュレータはクラスとモデルを登録して共通のインターフェースでインスタンスを生成できる Factory クラスを実装している。また、インスタンスの生成時にはモデルのパラメータセットのみでなく、そのインスタンスの初期状態や「親」となる登場物等、インスタンス固有のパラメータセットも必要となるため、本シミュレータにおいてはモデルのパラメータセットを modelConfig という名称の json 型変数で、インスタンス固有のパラメータセットを instanceConfig という名称の json 型変数で取り扱うこととしている。クラス名を指定してインスタンスを生成する場合は Factory に対してクラス名とともに modelConfig と instanceConfig を与え、モデル名を指定して生成する場合はモデル名とともに instanceConfig を与えることで行う。

3.2 シミュレーションの登場物

本シミュレータの登場物は大きく 2 種類に分けられる。一つはシミュレーション中に実際に様々な行動をとる主体となる Asset であり、もう一つは勝敗の判定や得点、報酬の計算、あるいはログの出力や場面の可視化等、Asset の動きに応じて様々な処理を行いシミュレーションの流れを管理する Callback である。両者はともに Factory に登録して生成するものとし、３．１項で述べた modelConfig と instanceConfig をコンストラクタ引数にとる共通の基底クラス Entity を継承したものとして実装されている。Asset と Callback はそれぞれ図３．２－１のように、役割に応じてもう一段階細かい分類をしている。

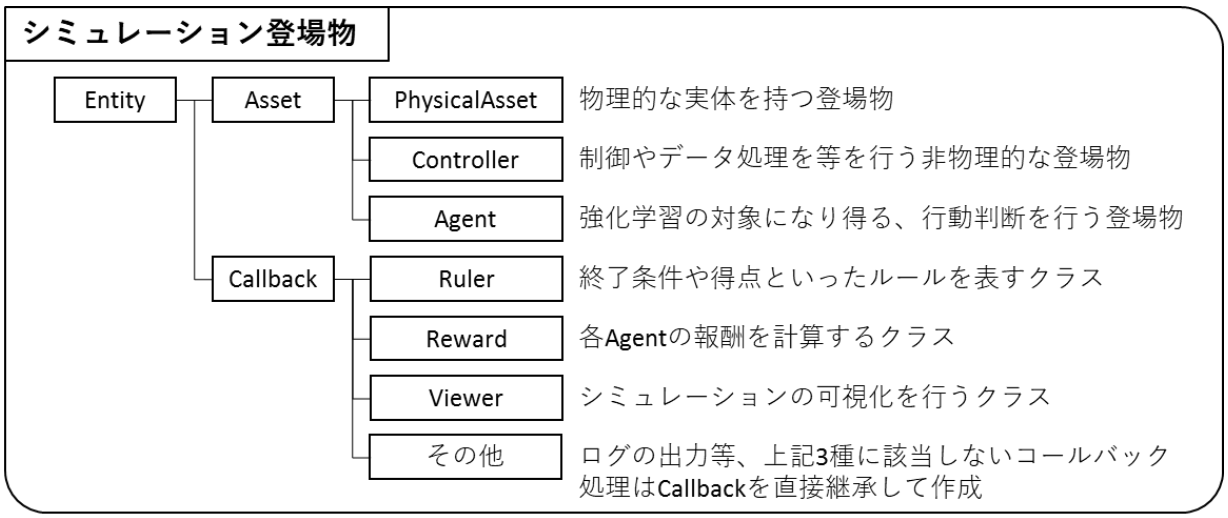

図 ３．２－１　シミュレーション登場物の分類

3.2.1 Asset

Asset は、観測(perceive)、制御(control)、行動(behave)の 3 種類の処理をそれぞれ指定された周期で実行することで環境に作用する。また、各 Asset の処理の中には他の Asset の処理結果に依存するものがあるため、同時刻に処理を行うこととなった際の処理優先度を setDependency 関数によって指定することができる。

Asset は自身の生存状態を返す関数 bool isAlive()と、生存状態 false にするための関数 void kill()を持つ。生存中のみ前述の 3 種類の処理が実行され、一度生存状態が false となった Asset はそのエピソード中に復活することはない。ただし、インスタンスとしてはエピソードの終了まで削除せずに維持しており、例えば Callback 等から特定の処理のために変数を参照することができるようにしている。

また、各 Asset は共通のメンバ変数として、「観測してよい情報」である observables と、「自身の行動を制御するための情報」である commands を json 型変数として持ち、他の Asset が具体的にどのようなクラスであるかを気にせずに相互作用を及ぼすインターフェースとして用いることができる。ただし、他 Asset との相互作用を必ずしもこれらの変数を介して行わなければならないものではなく、各クラスの定義次第では直接互いのメンバ変数やメンバ関数を参照することも許容される。

Asset は大きく physicalAsset、Contoller、Agent の 3 種類のサブクラスに分類され、それぞれの概要は以下の通りである。

3.2.1.1 PhysicalAsset

PhysicalAsset は Asset のうち物理的実体を持つものを表すクラスであり、独立した Asset として生成されるものと、他の PhysicalAsset に従属してその「子」として生成されるものとする。

自身の位置や速度、姿勢といった運動に関する状態量(1.5.3 項)を持つほか、従属関係にあるものについてはその運動が「親」に束縛されているか否かを指定することができる。また、自ら「子」となる PhysicalAsset と Controller を生成することができる。

3.2.1.2 Controller

Controller は Asset のうち物理的実体を持たないものを表すクラスであり、一つの PhysicalAsset に従属してその「子」として生成されるものである。PhysicalAsset の複雑な制御を実現するために用いることを想定しているが、control だけでなく、必要に応じて perceive と behave で処理を行うことも許容される。

また、自ら「子」となる PhysicalAsset と Controller を生成することができる。

3.2.1.3 Agent

Agent は Controller と同様に物理的実体を持たないものであり、一つ以上の PhysicalAsset に従属してその「子」として生成されるものとする。Controller との最大の違いは、強化学習の対象として、一定周期でシミュレータの外部に Observation を供給し、外部から Action を受け取る役割を持っていることと、「親」である PhysicalAsset の実体にはアクセスできず、前述の observable と commands を介して作用することしか認められていないことである。

Agent は親 Asset の observables を読み取って Observation を生成する makeObs 関数と、Action を引数として受け取り自身のメンバ変数 commands を更新する deploy 関数を持っており、親 Asset は Agent の commands を参照して自身の動作を決定する。また、Agent は perceive、control、behave の処理も行えるため、これらを活用して Observation、Action の変換のために様々な中間処理を行うことも許容される。

3.2.2 Callback

Callback は、シミュレーション中に周期的に呼び出される処理を記述してシミュレーションの流れを制御するクラスであり、次の 6 種類のタイミングで対応するメンバ関数が呼び出される。

(1) onEpisodeBegin…各エピソードの開始時(＝reset 関数の最後)

(2) onStepBegin…step 関数の開始時(＝外部から与えられた Action を Agent が受け取った直後)
(3) onInnerStepBegin…各 tick の開始時(＝Asset の control の前)
(4) onInnerStepEnd…各 tick の終了時(＝Asset の perceive の後)
(5) onStepEnd…step 関数の終了時（＝step 関数の戻り値生成の前または後※1)
(6) onEpisodeEnd…各エピソードの終了時（step 関数の戻り値生成後)

Callback は大きく Ruler、Reward、Viewer、その他の 4 種類に分けられ、それぞれの概要は以下の通りである。

3.2.2.1 Ruler

Ruler は、エピソードの終了判定の実施と各陣営の得点の計算を主な役割としている、名前の通りシミュレーションのルールを定義するクラスである。そのため、単一のエピソード中に存在できる Ruler インスタンスは一つのみに限られる。また、Ruler は Asset と同様に「観測してよい情報」である observable を json 型変数として持っており、Agent クラスからは得点以外には observable にしかアクセスできないようにしている。

3.2.2.2 Reward

Reward は、エピソードにおける各 Agent の報酬の計算を行うためのクラスであり、Ruler と異なり、単一のエピソード中に複数存在させることができる。Reward は陣営単位の報酬を計算するものと Agent 単位の報酬を計算するもの 2 種類に分類でき、生成時に計算対象とする陣営や Agent の名称を与えることで個別に報酬関数をカスタマイズできる。

3.2.2.3 Viewer

Viewer は、エピソードの可視化を行うためのクラスである。Viewer は Ruler と同様、単一のエピソード中にただ一つのインスタンスのみ存在できる。

3.2.2.4 その他

その他の Callback は特別な共通サブクラスはなく、Callback クラスを直接継承してよい。例えば、ログの出力や、次のエピソードのコンフィグの書き換えを行うために用いることができる。更には、前述の Ruler や Reward の処理結果や Asset の状態量を強引に改変したり、Agent から本来はアクセス出来ない情報を Agent に伝達したりすることも可能な作りとなっており、かなり広い範囲にわたってシミュレーションの挙動に干渉することができるものとしている。

また、その他に該当する Callback のうち、ログ出力に該当するものを Logger として通常の Callback とは明示的に分けて生成する。これは Logger が Viewer の可視化結果を参照できるようにするためであり、シミュレーション中は Logger 以外の Callback→Viewer→Logger の順で処理される。

3.3 シミュレーションの処理の流れ

本項では、シミュレーションの処理の流れの概要をまとめる。一般的な OpenAI Gym 環境と同じく、本シミュレータの外部から見た場合に各エピソードは初期 Observation を返す reset 関数から始まり、その後は Action を入力して Observation、Reward、Done、Info の 4 種類の値を返す step 関数を繰り返し、全ての Agent に対する Done が True となった時点で終了となる。３．２．１項に挙げた Asset と、３．１項で挙げたポリシーに関するエピソード中の処理フローは図 ３．３－１の通りである。現時点での実装では、全ての Agent 行動判断周期(ポリシーと Observation、Action をやりとりする周期)は同一とし、n[tick]ごとに行うものとしている。また、図中青色で示している step 関数内のn[tick]分の時間進行処理は、必ずしも全ての Asset が1[tick]ごとに処理を行うものではなく、各 Asset クラスまたはモデルごとに指定された周期で、実行すべき時刻が来たときに実行される。ただし、現時点で実装済のクラス及びモデルは全て1[tick]ごとに処理するものとして実装されている。

また、３．２．２に挙げた Callback の 6 種類の処理の実行タイミングは Callback の種類によって異なり、図 ３．３－２に示すタイミングで実行される。

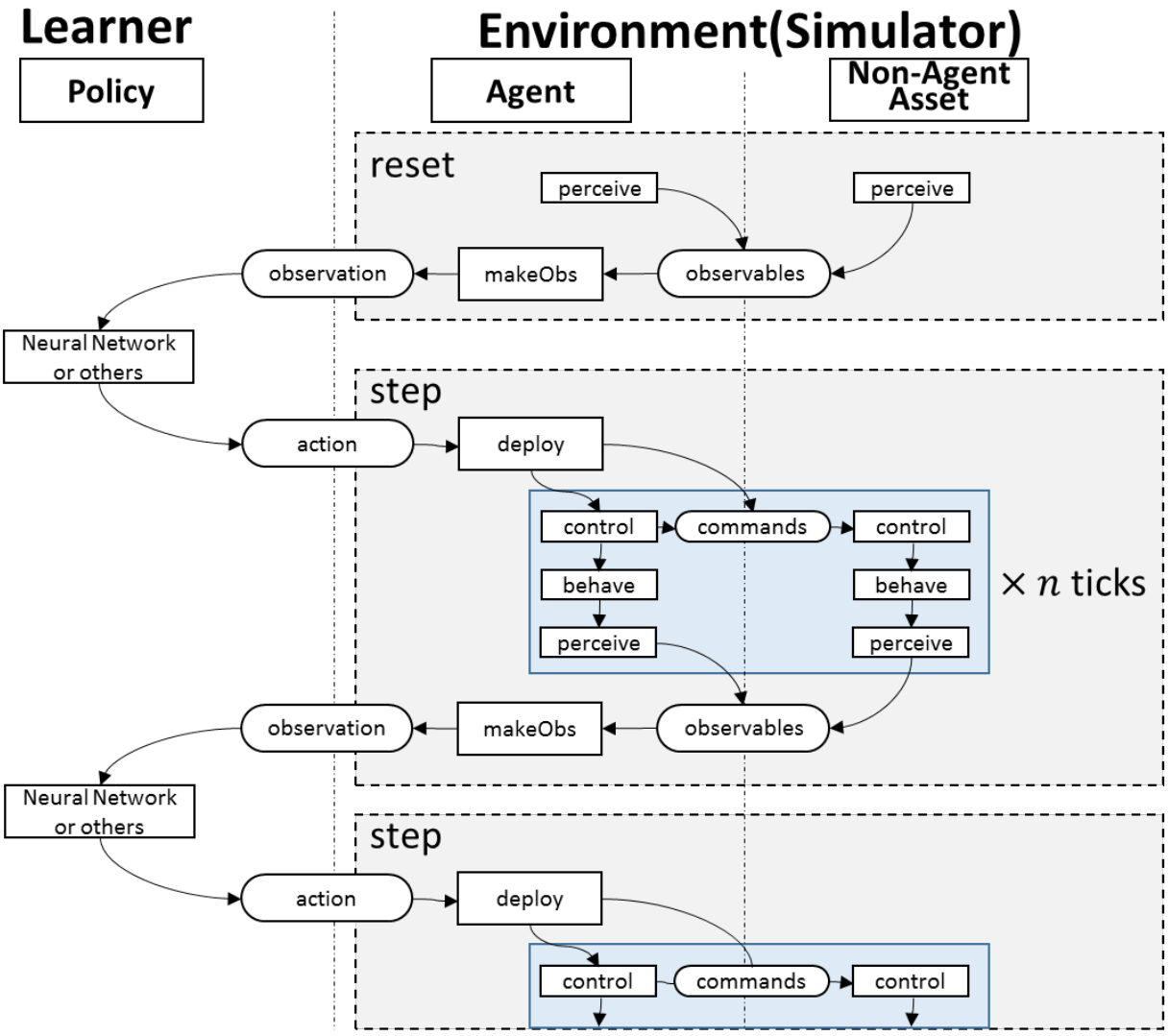

図 ３．３－１　シミュレーション中の処理の流れ(行動判断に関するもののみ)

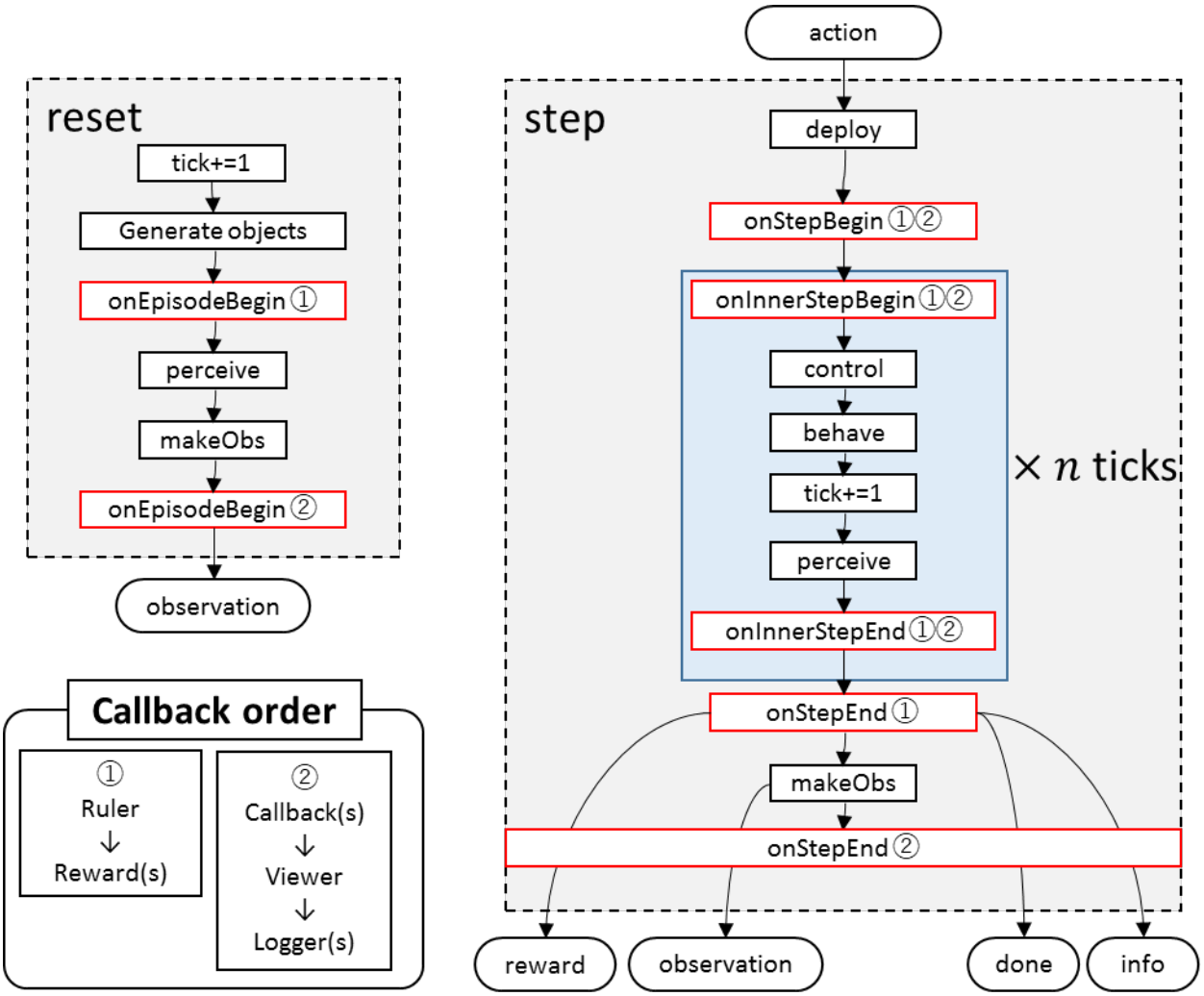

図　3．3－2　コールバックの処理タイミング

3.4 ポリシーの共通フォーマット

通常、OpenAI gym 環境においてエピソードの実行は外部から制御されるため、ポリシーが Action を計算する処理の実装方法は制約が無い。しかし、一部の Agent について環境の内部で処理を済ませたい場合には環境側が主となってポリシーの Action 計算処理を呼び足す必要がある。

そのため、本シミュレータでは StandalonePolicy クラスとしてポリシーの共通インターフェースを定義している。Standalone クラスは引数、返り値なしの reset 関数と、Observation 等を入力して Action を出力する step 関数の二つを有するものとし、それぞれ環境側の reset, step と対になるものである。step 関数の引数として与えられる変数は以下の通りである。

表　3．4－1　StandalonePolicy の step 関数の引数

引数	型	概要
observation	Any	環境から得られた、計算対象の Agent の observation
reward	float	環境から得られた、計算対象の Agent の reward
done	bool	環境から得られた、計算対象の Agent の done
info	Any	環境から得られた、計算対象の Agent の info
agentFullName	str	計算対象の Agent の完全な名称であり、 agentName:modelName:policyName の形をとる
observation_space	gym.spaces.Space	与えられた Observation の Space
action_space	gym.spaces.Space	計算すべき Action の Space