基于强化学习的蜂鸟采蜜智能体

朱正阳

摘　要　基于Unity引擎、强化学习工具包ML-Agents、Tensorflow训练框架、PPO近端策略优化算法，我设计了一个PC端中能够模仿蜂鸟采蜜的智能体HummingBird。我设计了训练环境、传感器、奖励规则函数，为了使智能体行为更接近真实，运用了数学和物理方法，诸如鸟喙对齐程度系数、平滑加速度系数等，使得智能体的行为更拟真化。最后，分析了整个训练过程的奖励累计曲线，通过调整参数等方法，实现了训练加速、防止智能体接触墙体等。最终展望了不足和后续的工作。

关键词　强化学习、ML-Agents、近端策略优化、智能体

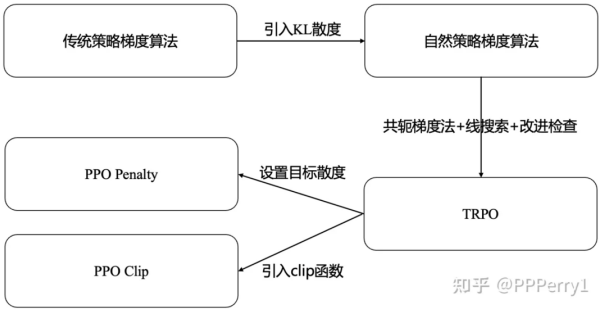

Fig. 1　PPO算法示意图

1　课题背景

1.1　游戏中的智能体

游戏中的智能体是重要组成部分。智能体可以在帮助新手学习游戏玩法、提高游戏技巧，起到引导作用。另一方面可以通过强度控制，提供竞技挑战。（王者荣耀绝悟人机）在实现智能体的其他算法中，常见的有决策树、有限状态机（FSM）等。但是往往需要复杂的逻辑流程控制，提高开发难度。

1.2　数据驱动决策

随着机器学习、数据挖掘等快速发展，数据驱动决策在金融、医疗领域崭露头角。强化学习算法能自动完成数据建模，依据数据直接产生决策。

1.3　强化学习基本框架

强化学习基本框架，包含智能体、环境、反馈、状态空间、动作空间等。从框架中感知当前环境s，执行一个决策动作a，得到新的状态s'和反馈r。通过一系列决策获得最佳的累计反馈。

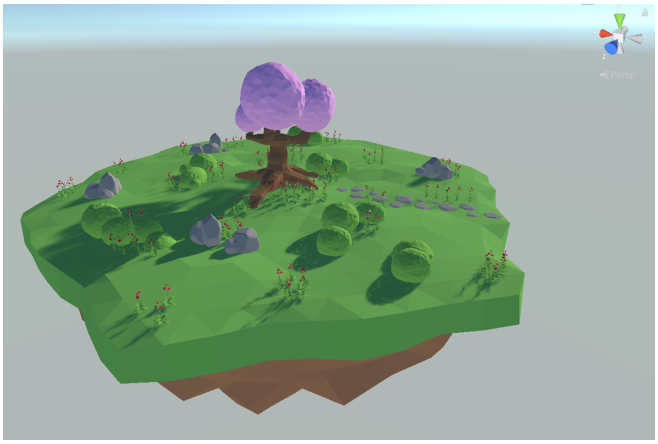

Fig. 2　环境：拥有花簇，灌木、树等的小岛。

2　具体内容

2.1　训练场景

2.1.1　环境

花丛(FlowerArea): 包含许多花簇
花簇(FlowerPlant): 包含3 4朵花
花(Flower): 包含花蜜(Nectar)、花瓣和其他实体碰撞体(Collider)
智能体(Agent): 采蜜的蜂鸟(HummingBird)

2.1.2　花簇、花束、花

花簇有若干花束，每个花束有若干多花。在每轮训练中，花簇的位置、朝向、旋转以及花束的数量都会随机化。每朵花有一个花蜜(Nectar)。智能体蜂鸟的目标就是以更标准的姿势吃到花蜜。

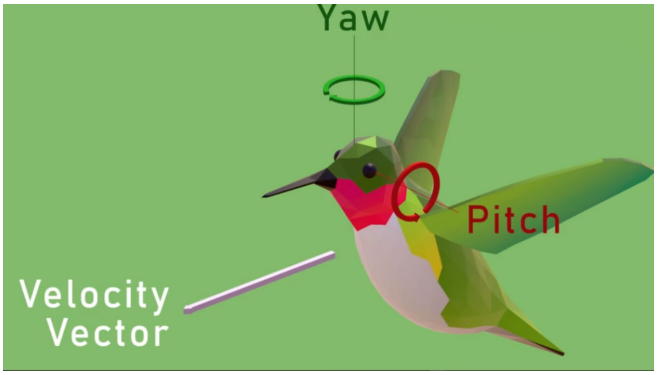

Fig. 3 偏航、俯冲、速度

2.2 状态信息与动作空间

状态信息：鸟的中心位置、鸟喙位置、已经获得的花蜜数量、速度向量。动作空间：偏航(Yaw)角度和速度、俯冲(Pitch)角度和速度、施加的力(moveForce)。

通过定义状态信息和动作空间，智能体可以根据当前的状态信息选择适当的动作来与环境交互。状态信息提供了环境的观测数据，而动作空间定义了智能体可以采取的行动范围。这些信息可以被用于训练智能体的决策模型，使其能够学习适应环境并做出有效的决策。

下面给出状态信息定义代码：

```
public Transform beakTip;
public Camera agentCamera;
public float NectarObtained { get; private set; }
public Vector3 BirdCenterPosition
{
    get { return transform.position; }
    private set { transform.position = value; }
}
public Vector3 BeakTipCenterPosition
{
    get { return beakTip.position; }
    private set { beakTip.position = value; }
}
public float moveForce = 2f;
public float pitchSpeed = 100f;
public float maxPitchAngle = 80f;
public float yawSpeed = 100f;
private float smoothPitchSpeedRate = 0f;
private float smoothYawSpeedRate = 0f;
private float smoothChangeRate = 2f;
```

2.3 观察、感知环境(CollectObservation)

在定义智能体收集观测数据的行为时，需要确定智能体需要哪些信息来进行训练和决策。观测数据是智能体对环境进行感知和理解的基础，它提供了环境状态的表示。

首先，定义观测空间：观测空间定义了智能体需要观测的状态的类型和范围。它可以是连续的（例如位置、速度）或离散的（例如分类标签）。我使用了Unity ML Agents框架提供的VectorSensor来定义观测空间，并且我们的观测信息都是连续的。

实现CollectObservations方法：在智能体的脚本中，CollectObservations方法来收集观测数据。这个方法在每个训练步骤或决策之前被调用。在该方法中，使用VectorSensor来添加观测数据到观测空间中。例如，添加智能体的位置、速度、感知到的物体等信息。

传递观测数据给决策算法：收集完观测数据后，需要将这些数据传递给智能体的决策算法进行处理。这可能涉及将观测数据输入到神经网络中，或者通过其他机器学习算法进行训练和决策。后文会给出代码。下面是定义的状态空间：

- 相对旋转四元数relativeRotation (x, y, z, w)
- 指向花向量toFlower (x, y, z)
- 位置对齐程度positionAlignment
- 喙对齐程度beakTipAlignment
- 距最近目标相对远离程度relativeDistance

其中，为了使得鸟喙以对齐花开口方向这种更标准的姿势吃花蜜，我在这里定义了鸟喙对齐程度系数和位置对齐程度系数。在下文的奖励函数设计中有所使用。以下是公式表达:

$$pA = \frac{\overrightarrow{t}}{||\overrightarrow{t}||} \cdot \left(-\frac{\overrightarrow{up}}{||\overrightarrow{up}||} \right) \in [-1, 1]$$

$$bA = \frac{\overrightarrow{f}}{||\overrightarrow{f}||} \cdot \left(-\frac{\overrightarrow{up}}{||\overrightarrow{up}||} \right) \in [-1, 1]$$

$$r = \frac{||\overrightarrow{t}||}{R} \in (0, 1)$$

其中，pA代表位置对齐程度，bA代表鸟喙对齐程度，r代表相对距离系数。

下面是实现CollectObservations方法的实现代码：

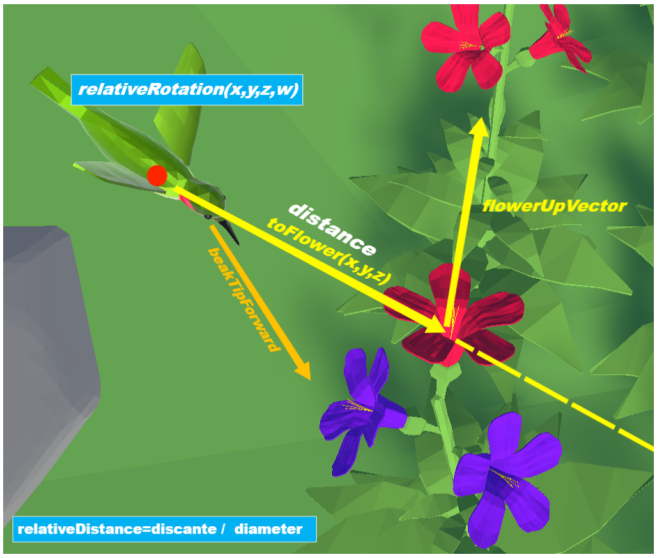

Fig. 4 偏航、俯冲、速度

```
public override void CollectObservations(
    VectorSensor sensor)
{
    if (nearestFlower == null)
    {
        sensor.AddObservation(new float[10]);
        return;
    }
    Quaternion relativeRotation = transform.
        localRotation.normalized;
    Vector3 toFlower = nearestFlower.
        FlowerCenterPosition -
        BeakTipCenterPosition;
    float positionAlignment = Vector3.Dot(
        toFlower.normalized, -nearestFlower.
        FlowerUpVector.normalized);
    float beakTipAlignment = Vector3.Dot(beakTip
        .forward.normalized, -nearestFlower.
        FlowerUpVector.normalized);
    float relativeDistance = toFlower.magnitude
        / FlowerArea.areaDiameter;
    sensor.AddObservation(relativeRotation);
    sensor.AddObservation(toFlower.normalized);
    sensor.AddObservation(positionAlignment);
    sensor.AddObservation(beakTipAlignment);
    sensor.AddObservation(relativeDistance);
}
```

2.4 传感器(Sensor)

射线传感器(RayCastSensor)，用于感知周围环境与碰撞信息。包括前方扇形射线、上和下两个射线传感器。

在Unity中，射线传感器（Raycast）是一种常用的技术，用于检测场景中的物体和表面。射线传感器通过发射一条无形的线（射线），并检测该线与场景中的物体是否相交，以及相交点的位置和其他属性。

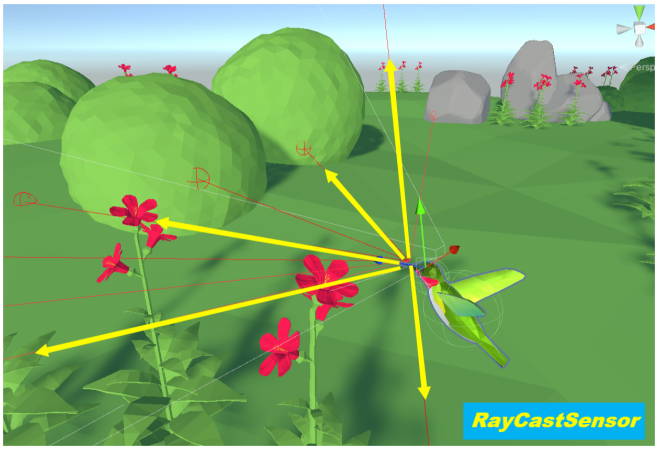

Fig. 5 传感器

射线传感器在游戏开发和虚拟现实应用中具有广泛的应用。它可以用于各种任务，例如检测碰撞、进行物体拾取、确定目标可见性等。

2.5 奖励函数

在强化学习中，奖励函数起着至关重要的作用。奖励函数是对智能体在环境中的行为进行评价和反馈的机制。它通过给予智能体正面或负面的奖励信号来指导智能体的学习和决策过程。

奖励函数的作用是激励智能体朝着期望的目标进行行动。当智能体采取一个好的行动或接近目标时，奖励函数给予正面奖励，增强智能体这种行为的倾向性。相反，当智能体采取不良的行动或偏离目标时，奖励函数给予负面奖励，抑制智能体这种行为的倾向性。通过不断调整奖励信号的大小和方向，奖励函数可以引导智能体朝着更优化的策略进行学习和决策。

触发停留事件(OnTriggerStay):当智能体的碰撞体触碰到花蜜的时候，有可能是身体的任意部分接触。需要确定是否为鸟喙部分进入，并捕捉鸟喙与花开口方向对齐程度。

数学表达如下：

$$R_t = R(\ a_t \mid s_t\ ) = R_\beta + R_{\lambda_1} * pA + R_{\lambda_2} * bA$$

其中，R_β表示吃花蜜得到的基础奖励，R_{λ_1}，R_{λ_2}分别是对齐程度的基础奖励。

由于我发现在训练过程中，鸟会撞击地面、出界等，因此奖励函数增加了惩罚项，对鸟的活动进行限制:

$$R_t = R(\ a_t \mid s_t\ ) = R_\beta + R_{\lambda_1} * pA + R_{\lambda_2} * bA - R_{\lambda_3}$$

奖励函数实现代码如下：

```
private void OnTriggerStay(Collider collider)
{
    if (collider.CompareTag("Nectar"))
    {
        Vector3 closePointToBeakTip = collider.ClosestPoint(BeakTipCenterPosition);
        if (Vector3.Distance(closePointToBeakTip, BeakTipCenterPosition) < BeakTipRadius)
        {
            Flower flower = flowerArea.GetFlowerFromNectar(collider);
            float nectarReceived = flower.Feed(.01f);
            NectarObtained += nectarReceived;
            if (trainingMode)
            {
                float forwardAlignment = Vector3.Dot(transform.forward.normalized, -nearestFlower.
                    FlowerUpVector.normalized);
                float bonus = .02f * Mathf.Clamp01(forwardAlignment);
                float baseIncrement = .01f;
                float increment = baseIncrement + bonus;
                AddReward(increment);
            }
        }
        if (!nearestFlower.HasNectar)
        {
            UpdateNearestFlower();
        }
    }
}
 private void OnCollisionEnter(Collision collision)
{
    if (collision.collider.CompareTag("Boundary") && trainingMode)
    {
        AddReward(-0.5f);
    }
}
```

2.6 决策、动作、拟真

接收10维动作向量，并用物理学方式更新状态。在每次Agent（可以是玩家、神经网络或其他形式的决策实体）接收到一个新行为的时候调用(action received)。

根据接收到的行为更新Agent的状态、执行特定的动作或触发相关的事件。这个方法允许根据不同的行为来驱动Agent在游戏中进行相应的操作和决策，

为了使得蜂鸟动作更加拟真，而非突兀的切换动作、速率阶跃，增加平滑速率，控制加速度平滑递增：

$$\alpha_{pitch} = \Delta T \times r_{\alpha} + \alpha_0$$

$$\nu_{pitch} = \Delta T \times \alpha_{pitch} + \nu_0$$

代入可得：

$$\begin{aligned}\nu_{pitch} &= \Delta T \times (\Delta T \times r_{\alpha} + \alpha_0) + \nu_0 \\ &= (\Delta T)^2 \times r_{\alpha} + \Delta T \times \alpha_0 + \nu_0\end{aligned}$$

实现代码如下：

```
public override void OnActionReceived(
    ActionBuffers actions)
{
    if (frozen) return;
    var vectorAction = actions.ContinuousActions
        ;
    Vector3 targetMoveDirection = new Vector3(
        vectorAction[0], vectorAction[1],
        vectorAction[2]);
    rigidbody.AddForce(targetMoveDirection *
        moveForce);
    Vector3 curRotation = transform.rotation.
        eulerAngles;
    float targetPitchSpeedRate = vectorAction
        [3];
    float targetYawSpeedRate = vectorAction[4];
    smoothPitchSpeedRate = Mathf.MoveTowards(
        smoothPitchSpeedRate,
        targetPitchSpeedRate, smoothChangeRate *
         Time.fixedDeltaTime);
    smoothYawSpeedRate = Mathf.MoveTowards(
        smoothYawSpeedRate, targetYawSpeedRate,
        smoothChangeRate * Time.fixedDeltaTime);
    float pitch = curRotation.x +
        smoothPitchSpeedRate * Time.
        fixedDeltaTime * pitchSpeed;
    float yaw = curRotation.y +
        smoothYawSpeedRate * Time.fixedDeltaTime
         * yawSpeed;
    if (pitch > 180f) pitch -= 360f;
    pitch = Mathf.Clamp(pitch, -maxPitchAngle,
        maxPitchAngle);
    transform.rotation = Quaternion.Euler(pitch,
         yaw, 0);
}
```

2.7 启发式控制与输入控制兼容

兼容训练模式和启发式控制，即允许玩家手动控制蜂鸟。

在后续还可以引入AI对战人机功能。启发式控制主要通过按键捕获，将动作决策传递给物理更新的函数。

实现代码如下：

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    Vector3 left = Vector3.zero;
    Vector3 up = Vector3.zero;
    Vector3 forward = Vector3.zero;
    float pitch = 0f;
    float yaw = 0f;

    if (Input.GetKey(KeyCode.W)) forward = transform.forward;
    else if (Input.GetKey(KeyCode.S)) forward = (-1f) * transform.forward;

    if (Input.GetKey(KeyCode.LeftArrow)) left = (-1f) * transform.right;
    else if (Input.GetKey(KeyCode.RightArrow)) left = transform.right;

    if (Input.GetKey(KeyCode.Space)) left = transform.up;
    else if (Input.GetKey(KeyCode.LeftControl)) left = (-1f) * transform.up;

    if (Input.GetKey(KeyCode.UpArrow)) pitch = -1f;
    else if (Input.GetKey(KeyCode.DownArrow)) pitch = 1f;

    if (Input.GetKey(KeyCode.A)) yaw = -1f;
    else if (Input.GetKey(KeyCode.D)) yaw = 1f;

    Vector3 combinedDirection = (forward + up + left).normalized;

    actionsOut.ContinuousActions.Array[0] = combinedDirection.x;
    actionsOut.ContinuousActions.Array[1] = combinedDirection.y;
    actionsOut.ContinuousActions.Array[2] = combinedDirection.z;
    actionsOut.ContinuousActions.Array[3] = pitch;
    actionsOut.ContinuousActions.Array[4] = yaw;
}
```

2.8 启发式随机搜索合适位置

考虑最大递归深度地、启发式随机搜索合理空间位置：需满足具有合理的碰撞情况。。

设置随机参数inFrontFlower，以概率的形式确定随机得到的朝向与花开口朝向的关系

实现代码如下：

```
private void MoveToSafeRandomPosition(bool inFrontOfFlower)
{
bool safePositionFound = false;
int attemptsRemaining = 100;
Vector3 potentialPosition = Vector3.zero;
Quaternion potentialRotation = new Quaternion();
while (!safePositionFound && attemptsRemaining > 0)
{
    if (inFrontOfFlower)
    {
        int flowersCount = flowerArea.Flowers.Count;
        Flower randomFlower = flowerArea.Flowers[Random.Range(0, flowersCount)];

        float distanceFromFlower = Random.Range(.1f, .2f);
        Vector3 offset = randomFlower.FlowerUpVector * distanceFromFlower;
        potentialPosition = randomFlower.transform.position + offset;

        Vector3 toVFlower = potentialPosition - randomFlower.FlowerCenterPosition;
        potentialRotation = Quaternion.LookRotation(toVFlower, Vector3.up);
    }
    else
    {
        float height = Random.Range(1.2f, 2.5f);
        float radius = Random.Range(2f, 7f);
        Quaternion direction = Quaternion.AngleAxis(Random.Range(-180f, 180f), Vector3.up);

        Vector3 offset = Vector3.up * height + direction * Vector3.forward * radius;
        potentialPosition = flowerArea.FlowerAreaCenter + offset;

        float pitch = Random.Range(-60f, 60f);
        float yaw = Random.Range(-180f, 180f);
        potentialRotation = Quaternion.Euler(pitch, yaw, 0);
    }

    safePositionFound = Physics.CheckSphere(potentialPosition, 0.05f);
    attemptsRemaining--;
}
BirdCenterPosition = potentialPosition;
transform.rotation = potentialRotation;
}
```

根据最大递归深度和启发式随机搜索的原理，该代码实现了在合理的空间位置中搜索一个安全的随机位置。在搜索过程中，需要满足一定的碰撞条件。

代码中引入了一个随机参数inFrontOfFlower，通过概率的方式确定了随机得到的位置与花朝向的关系。

通过这段代码，可以实现在给定的空间范围内，按照一定的规则和限制，随机生成一个安全的位置，可以是面朝花朵或者是随机选择地面上的位置。该功能可以用于模拟游戏中对象的移动行为，确保对象在移动过程中不会发生碰撞或其他不合理的情况。

3 训练过程

3.1 训练参数设置

AnacondaPrompt中，激活虚拟环境、设置配置参数，指定PPO算法、批和缓冲区大小、线性学习率、隐藏层维度、隐藏层数等信息。

以下是参数配置文件，设定了使用PPO算法、最大迭代步数、每隔50万轮训练生成一个模型等。

```yaml
Hummingbird:
  trainer_type: ppo
  hyperparameters:
    batch_size: 2048
    buffer_size: 20480
    learning_rate: 0.0003
    beta: 0.005
    epsilon: 0.2
    lambd: 0.95
    num_epoch: 3
    learning_rate_schedule: linear
    beta_schedule: linear
    epsilon_schedule: linear
  network_settings:
    normalize: false
    hidden_units: 256
    num_layers: 2
    vis_encode_type: simple
    memory: null
    goal_conditioning_type: hyper
    deterministic: false
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
      network_settings:
        normalize: false
        hidden_units: 128
        num_layers: 2
        vis_encode_type: simple
        memory: null
        goal_conditioning_type: hyper
        deterministic: false
  init_path: null
  keep_checkpoints: 5
  checkpoint_interval: 500000
  max_steps: 5000000
  time_horizon: 128
  summary_freq: 10000
  threaded: false
  self_play: null
  behavioral_cloning: null
```

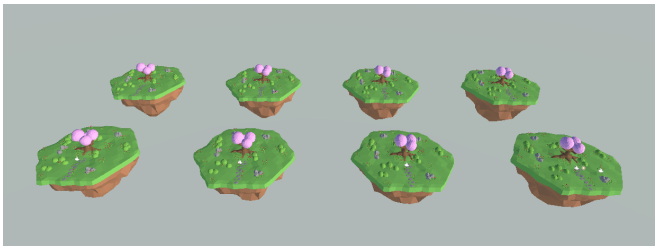

Fig. 6 复制了8份岛屿

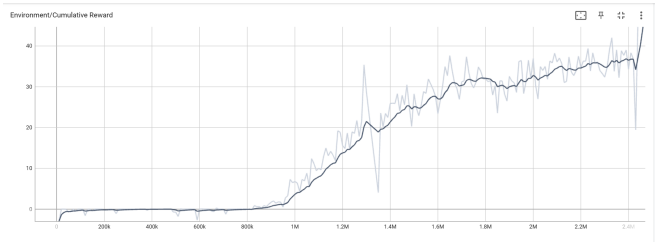

Fig. 7 前半段累计奖励图

3.2 加速训练

为了加速训练过程，复制了8份独立的训练场景。并且将将物理时间缩放(TimeScale)提高30倍。这并不会负面影响得到的奖励。

3.3 训练结果和累计奖励折线图

得到了拥有18层、5个连续型向量输出、4个空间状态信息向量输入的迭代模型(最大迭代次数500万次)。后续可以考虑梯度能力模型。（类似王者荣耀段位绝悟人机）

在初始的0到800k步迭代过程中，奖励累计一直为负数。这是由于在随机化智能体的决策中，我对惩罚项(碰撞墙体、地面、出界等)设置较大。前200万次训练结果如下

200到400万的训练中，由于部分参数可以训练时调整，我将单个Episode的训练最大步数提高到2倍。得到的累计奖励也对应的在一段时间后趋向于两倍。在320万次的时候，我将场景复制到16份，出现短暂的累计奖励骤降情况，猜测可能是初始化到过多惩罚情况。在400 500万次以后，又将单个Episode的训练最大步数恢复到原来。

4 总结和展望

本实验主要完成了对智能体对环境的感知、惩罚奖励规则、启发式随机搜索与初始化状态、动作空间与决策的物理更新等内容，在Unity引擎中使

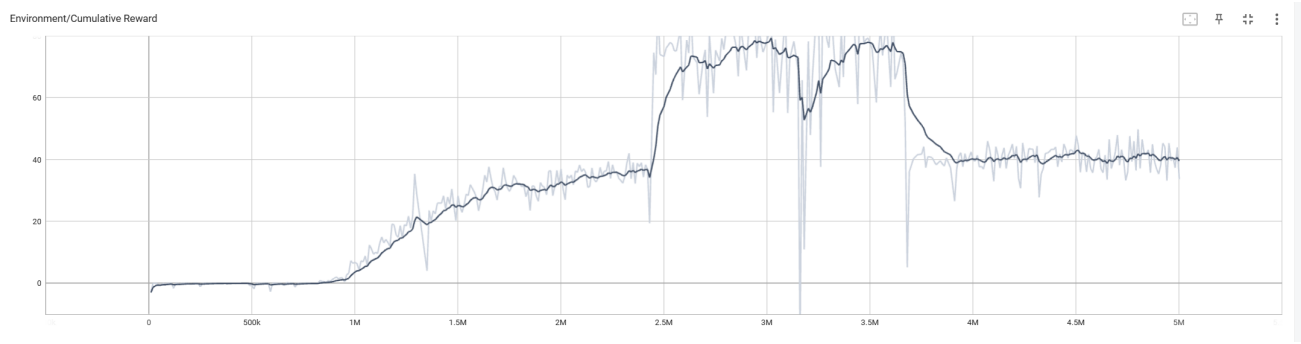

Fig. 8 完整累计奖励图

用MLAgents强化学习工具包，通过设定PPO算法在训练了3小时、500万轮后得到了较为可观的采蜜蜂鸟的模型。

训练完成的模型在蜂鸟上体现出了强化学习的强大：由于动作空间的连续性而非离散性，难以使用状态机或者决策树的方式调整。强化学习的算法使得模型对输出连续向量空间成为了可能。

后续工作：目前是单一场景中单一蜂鸟采蜜，没有考虑到多只蜂鸟同时盯上一束花时的竞争关系行为。后续可以采用自对弈的方式进行对抗学习。

由于启发式控制输入目前来说，不够优秀，人类玩家无法超过蜂鸟智能体。可以采用模型强度评估系统，针对不同迭代次数的模型，得到不同难度的AI，从而应用到更为实际的游戏中。