IMMERSIVE LIMIT

February 20, 2020

Reinforcement Learning Penguins (Part 2/4) | Unity ML-Agents

PENGUINS WITH UNITY ML-AGENTS PROJECT

In this tutorial you'll write all of the C# code needed for the penguin ML-Agents. These scripts manage Scene setup (such as randomized placement of the penguin agent), penguin decision making, fish movement, and interaction between the penguin agent and the Scene.

WRITING THE CODE

First, you'll create all of the C# scripts needed for this project. After you've created them, we'll walk through the code for each.

- Create a new folder in Unity called *Scripts* inside the *Penguin* folder.
- Create three new C# scripts inside the *Scripts* folder **(Figure 01)**:
    - PenguinArea
    - PenguinAgent
    - Fish

Figure 01: C# scripts in the Scripts folder.

PENGUINACADEMY.CS

The PenguinAcademy script is not needed as of version 0.14. Academy is now a singleton and we don't need our own version. I'm leaving this here in case you are coming from the older tutorial and are confused about where it went.

PENGUINAREA.CS

The PenguinArea **(Figure 02)** will manage a training area with one penguin, one baby, and multiple fish. It has the responsibilities of removing fish, spawning fish, and random placement of the penguins. There might be multiple PenguinAreas in a Scene for more efficient training.

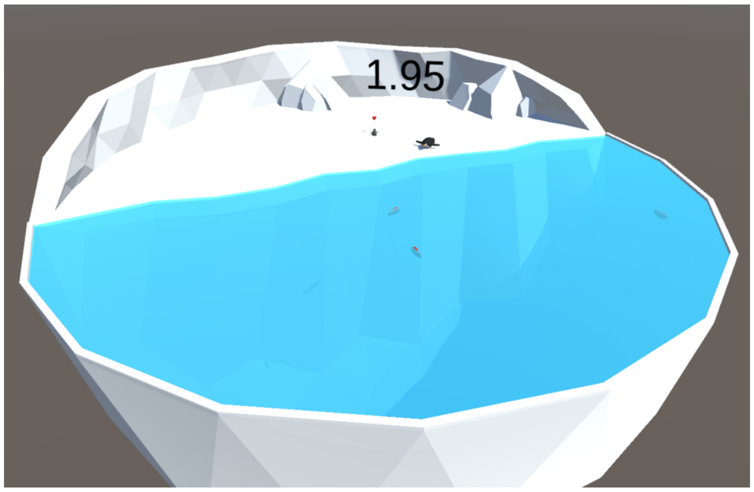

Figure 02: A single PenguinArea with penguins and fish, which you will be creating later and attaching the PenguinArea.cs script to.

- Delete the Update function.
- Add using statements for MLAgents and TMPro.
- Update the class definition to inherit from Area instead of Monobehaviour.

```
using System.Collections.Generic;
using UnityEngine;
using MLAgents;
using TMPro;

public class PenguinArea : MonoBehaviour
{

}
```

- Add the following variables inside the class (between { }).

```
[Tooltip("The agent inside the area")]
    public PenguinAgent penguinAgent;

    [Tooltip("The baby penguin inside th
    public GameObject penguinBaby;

    [Tooltip("The TextMeshPro text that 
    public TextMeshPro cumulativeRewardT

    [Tooltip("Prefab of a live fish")]
    public Fish fishPrefab;

    private List<GameObject> fishList;
```

variables in Unity later in this tutorial.

- Add a new ResetArea() function inside the class after the last private variable.

```
/// <summary>
    /// Reset the area, including fish a
    /// </summary>
    public void ResetArea()
    {
        RemoveAllFish();
        PlacePenguin();
        PlaceBaby();
        SpawnFish(4, Academy.Instance.Fl
    }
```

The functions in the code above do not exist yet, but we will create them later in this script.

If you're wondering what is going on in the SpawnFish function, it has to do with something called "curriculum learning". In short, we're going to make the task of feeding the baby progressively more difficult as the penguin agent gets better at the task. On this line, we are asking for the current "fish_speed" of the curriculum. In our curriculum, which we'll define later, we'll start with a speed of 0, but the default will be 0.5 if there's no active curriculum.

- Add a new RemoveSpecificFish() function.
- Add a new FishRemaining() function.

```
/// <summary>
    /// Remove a specific fish from the
    public void RemoveSpecificFish(GameC
    {
        fishList.Remove(fishObject);
        Destroy(fishObject);
    }

    /// <summary>
    /// The number of fish remaining
    /// </summary>
    public int FishRemaining
    {
        get { return fishList.Count; }
    }
```

When the penguin catches a fish, the PenguinAgent script will call RemoveSpecificFish() to remove it from the water.

The next few functions will handle placement of the animals in the area. It makes the most sense to spawn fish in the water and place the baby penguin on land. The penguin can move between land and water, so it can be placed in either. In **Figure 03** below, you can see where the script will randomly position each type of animal.

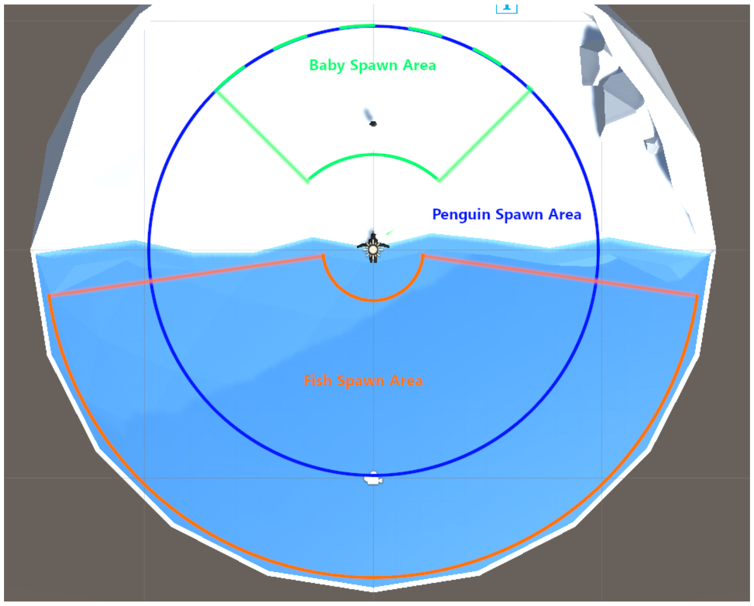

Figure 03: Placement regions for the penguin, the baby penguin, and fish.

- Add a new ChooseRandomPosition() function.

```
/// <summary>
    /// Choose a random position on the
    /// </summary>
    /// <param name="center">The center
    /// <param name="minAngle">Minimum a
    /// <param name="maxAngle">Maximum a
    /// <param name="minRadius">Minimum
    /// <param name="maxRadius">Maximum
    /// <returns>A position falling with
    public static Vector3 ChooseRandomPo
    {
        float radius = minRadius;
        float angle = minAngle;

        if (maxRadius > minRadius)
        {
            // Pick a random radius
        if (maxAngle > minAngle)
        {
            // Pick a random angle
            angle = UnityEngine.Random.R
        }

        // Center position + forward vec
        return center + Quaternion.Euler
    }
```

This function uses special radius and angle limits to pick a random position within wedges around the central point in the area. Read the comments in the code for more detail.

- Add a new RemoveAllFish() function.

```
/// <summary>
    /// Remove all fish from the area
    /// </summary>
    private void RemoveAllFish()
    {
        if (fishList != null)
        {
            for (int i = 0; i < fishList
            {
                if (fishList[i] != null)
    {
                    Destroy(fishList[i])
                }
            }
        }
    }
```

The ResetArea() function calls RemoveAllFish() to make sure no fish are in the area before spawning new fish.

- Add a new PlacePenguin() function.
- Add a new PlaceBaby() function.

```
/// <summary>
    /// Place the penguin in the area
    /// </summary>
    private void PlacePenguin()
    {
        Rigidbody rigidbody = penguinAge
        rigidbody.velocity = Vector3.zer
        rigidbody.angularVelocity = Vect
        penguinAgent.transform.position
        penguinAgent.transform.rotation
    }

    /// <summary>
    /// Place the baby in the area
    /// </summary>
    private void PlaceBaby()
    {
        Rigidbody rigidbody = penguinBab
        rigidbody.velocity = Vector3.zer
        rigidbody.angularVelocity = Vect
        penguinBaby.transform.position =
        penguinBaby.transform.rotation =
    }
```

things can happen when training for long periods of time at 100x speed. For example, the penguin could fall through the floor, then accelerate downward. When the area resets, the position would be reset, but if the downward velocity is not reset, the penguin might blast through the ground.

- Add a new SpawnFish() function.

```
/// <summary>
    /// Spawn some number of fish in the
    /// </summary>
    /// <param name="count">The number t
    /// <param name="fishSpeed">The swim
    private void SpawnFish(int count, fl
    {
        for (int i = 0; i < count; i++)
        {
            // Spawn and place the fish
            GameObject fishObject = Inst
            fishObject.transform.positic
            fishObject.transform.rotatic

            // Set the fish's parent to
            fishObject.transform.SetPare

            // Keep track of the fish
            fishList.Add(fishObject);

            // Set the fish speed
            fishObject.GetComponent<Fish
        }
    }
```

the code for more detail.

- Add a new Start() function and call ResetArea().

```
/// <summary>
    /// Called when the game starts
    /// </summary>
    private void Start()
    {
        ResetArea();
    }
```

- Add a new Update() function.

```
/// <summary>
    /// Called every frame
    /// </summary>
    private void Update()
    {
        // Update the cumulative reward
        cumulativeRewardText.text = peng
    }
```

This function updates the cumulative reward display text on the back wall of the area every frame. It is not necessary for training, but it helps you see how well the penguins are performing.

That's all for the PenguinArea script!

PENGUINAGENT.CS

The PenguinAgent class, which inherits from the Agent class, is where the cool stuff happens. It handles

- Open PenguinAgent.cs.
- Delete the Start() function.
- Delete the Update() function.
- Add a MLAgents using statement.
- Change the class definition to inherit from Agent instead of Monobehaviour.

```
using UnityEngine;
using MLAgents;

public class PenguinAgent : Agent
{

}
```

- Add public variables to keep track of the move and turn speed of the penguin agent as well as the Prefabs for the heart and regurgitated fish.

```
[Tooltip("How fast the agent moves forwa
    public float moveSpeed = 5f;

    [Tooltip("How fast the agent turns")
    public float turnSpeed = 180f;

    [Tooltip("Prefab of the heart that a
    public GameObject heartPrefab;

    [Tooltip("Prefab of the regurgitated
    public GameObject regurgitatedFishPr
```

- Add private variables to keep track of things.

```
    new private Rigidbody rigidbody;
    private GameObject baby;
    private bool isFull; // If true, per
    private float feedRadius = 0f;
```

InitializeAgent() is called once, automatically, when the agent wakes up. It is not called every time the agent is reset, which is why there is a separate ResetAgent() function. We'll use it to find a few objects in our Scene.

- Override InitializeAgent()

```
/// <summary>
    /// Initial setup, called when the a
    /// </summary>
    public override void InitializeAgent
    {
        base.InitializeAgent();
        penguinArea = GetComponentInPare
        baby = penguinArea.penguinBaby;
        rigidbody = GetComponent<Rigidbo
    }
```

AgentAction() is where the agent receives and responds to commands. These commands may originate from a neural network or a human player, but this function treats them the same.

The vectorAction parameter is an array of numerical values that correspond to actions the agent should take. For this project, we are using "discrete" actions, which means each integer value (e.g., 0, 1, 2, ...) corresponds to a choice. The alternative is "continuous" actions, which instead allows a choice of any fractional value

In this case:

- vectorAction[0] can either be 0 or 1, indicating whether to remain in place (0) or move forward at full speed (1).
- vectorAction[1] can either be 0, 1, or 2, indicating whether to not turn (0), turn in the negative direction (1), or turn in the positive direction (2).

The neural network, when trained, actually has no concept of what these actions do. It only knows that when it sees the environment a certain way, some actions tend to result in more reward points. This is why it will be very important to create an effective observation of the environment later in this script.

After interpreting the vector actions, the AgentAction() function applies the movement and rotation and then adds a small negative reward. This small negative reward encourages the agent to complete its task as quickly as possible.

In this case, a reward of -1 / 5000 is given for each of the 5,000 steps. If the penguin finishes early — in 3,000 steps, for example — the negative reward added from this line of code would be -3000 / 5000 = -0.6. If the penguin takes all 5,000 steps, the total negative reward would be -5000 / 5000 = -1.

- Override AgentAction()

```
/// <summary>
    /// Perform actions based on a vecto
    /// </summary>
    {
        // Convert the first action to f
        float forwardAmount = vectorActi

        // Convert the second action to
        float turnAmount = 0f;
        if (vectorAction[1] == 1f)
        {
            turnAmount = -1f;
        }
        else if (vectorAction[1] == 2f)
        {
            turnAmount = 1f;
        }

        // Apply movement
        rigidbody.MovePosition(transform
        transform.Rotate(transform.up *

        // Apply a tiny negative reward
        if (maxStep > 0) AddReward(-1f /
    }
```

The Heuristic() function allows control of the agent without a neural network. This function will read inputs from the human player via the keyboard, convert them into actions, and return a list of those actions. In our project:

- The default forwardAction will be 0, but if the player presses 'W' on the keyboard, this value will be set to 1.

be set to 1 or 2 respectively to turn left or right.

- Override the Heuristic() function.

```
/// <summary>
    /// Read inputs from the keyboard ar
    /// This is called only when the pla
    /// Behavior Type to "Heuristic Only
    /// </summary>
    /// <returns>A vectorAction array of
    public override float[] Heuristic()
    {
        float forwardAction = 0f;
        float turnAction = 0f;
        if (Input.GetKey(KeyCode.W))
        {
            // move forward
            forwardAction = 1f;
        }
        if (Input.GetKey(KeyCode.A))
        {
            // turn left
            turnAction = 1f;
        }
        else if (Input.GetKey(KeyCode.D)
        {
            // turn right
            turnAction = 2f;
        }

        // Put the actions into an array
        return new float[] { forwardActi
    }
```

all of the fish. We will use it to empty the penguin's belly and reset the area. We check the feed radius here because it may have changed since the last reset (during curriculum learning). The default feed radius is 0, meaning the penguin must actually touch its baby to feed it.

```
/// <summary>
    /// Reset the agent and area
    /// </summary>
    public override void AgentReset()
    {
        isFull = false;
        penguinArea.ResetArea();
        feedRadius = Academy.Instance.Fl
    }
```

The penguin agent observes the environment in two different ways. The first way is with raycasts. This is like shining a bunch of laser pointers out from the penguin and seeing if they hit anything. It's similar to LIDAR, which is used by autonomous cars and robots. Raycast observations are added via a RayPerceptionSensor component as of version 0.12, which we'll add in the Unity Editor later.

The second way the agent observes the environment is with numerical values. Whether it's a true/false value, a distance, an XYZ position in space, or a quaternion rotation, you can convert an observation into a list of numbers and add it as an observation for the agent. Check out the comments in the code to understand what we're adding.

about its environment, it will not be able to complete its task. Imagine your agent is floating in space, blindfolded. What would it need to be told about its environment to make an intelligent decision?

This penguin agent, as currently implemented, doesn't have any memory. We need to help it out by telling it where things are every update step so that it can make a decision. It's possible to use memory in ML-Agents, but that's beyond the scope of this tutorial. You can read more about it in the ML-Agents Recurrent Neural Network documentation.

- Override the CollectObservations() function.

```
/// <summary>
    /// Collect all non-Raycast observat
    /// </summary>
    public override void CollectObservat
    {
        // Whether the penguin has eaten
        AddVectorObs(isFull);

        // Distance to the baby (1 float
        AddVectorObs(Vector3.Distance(ba

        // Direction to baby (1 Vector3 
        AddVectorObs((baby.transform.pos

        // Direction penguin is facing (
        AddVectorObs(transform.forward);

        // 1 + 1 + 3 + 3 = 8 total value
    }
```

penguin is close enough to the baby and then try to regurgitate the fish to feed it. We do a check inside RegurgitateFish() to see if it has a full belly before doing so.

- Add a new FixedUpdate() function.

It used to be the case (in earlier versions of ML-Agents) that decisions were requested automatically. That is no longer the case, so now we have to call it ourselves. We'll request a decision every FixedUpdate steps and otherwise just request an action. It is common in reinforcement learning to ask for a decision and take that action a few times in a row before asking for a new decision. It saves on processing and probably reduces jitter.

```
private void FixedUpdate()
    {
        // Request a decision every 5 st
        // but for the steps in between,
        // of the previous decision
        if (GetStepCount() % 5 == 0)
        {
            RequestDecision();
        }
        else
        {
            RequestAction();
        }

        // Test if the agent is close en
        if (Vector3.Distance(transform.p
        {
        }
    }
```

Next we'll implement OnCollisionEnter() and test for collisions with items that have the tag "fish" or "baby" and respond accordingly.

- Add a new OnCollisionEnter() function.

```
/// <summary>
    /// When the agent collides with som
    /// </summary>
    /// <param name="collision">The coll
    private void OnCollisionEnter(Collis
    {
        if (collision.transform.CompareT
        {
            // Try to eat the fish
            EatFish(collision.gameObject
        }
        else if (collision.transform.Com
        {
            // Try to feed the baby
            RegurgitateFish();
        }
    }
```

Now we can add a function to eat fish, assuming the penguin doesn't already have a full stomach. It will remove that fish from the area and get a reward.

- Add a new EatFish() function.

```
    /// Check if agent is full, if not,
    /// </summary>
    /// <param name="fishObject">The fis
    private void EatFish(GameObject fish
    {
        if (isFull) return; // Can't eat
        isFull = true;

        penguinArea.RemoveSpecificFish(f

        AddReward(1f);
    }
```

Finally, we'll add a function to regurgitate fish and feed the baby. We'll spawn a regurgitated fish blob on the ground as well as a heart floating in the air to show how much the baby loves its parent for feeding it. We'll also set an auto-destroy timer. The agent gets a reward, and if there are no fish remaining, we call Done(), which will automatically call AgentReset().

- Create a new RegurgitateFish() function.

```
/// <summary>
    /// Check if agent is full, if yes,
    /// </summary>
    private void RegurgitateFish()
    {
        if (!isFull) return; // Nothing

        isFull = false;

        // Spawn regurgitated fish
        GameObject regurgitatedFish = In
        regurgitatedFish.transform.paren
        // Spawn heart
        GameObject heart = Instantiate<G
        heart.transform.parent = transfo
        heart.transform.position = baby.
        Destroy(heart, 4f);

        AddReward(1f);

        if (penguinArea.FishRemaining <=
        {
            Done();
        }
    }
```

That's all for the PenguinAgent script!

FISH.CS

The Fish class will attach to each fish and make it swim. Unity doesn't have water physics built in, so our code just moves them in a straight line toward a target destination to keep things simple.

- Open Fish.cs.
- Delete the Start() function.
- Delete the Update() function.
- Add several variables as shown.

Here's an overview of the variables:

- fishSpeed controls the average speed of the fish.
- randomizedSpeed is a slightly altered speed that we will change randomly each time a new swim destination is picked.

- targetPosition is the position of the destination the fish is swimming toward.

```
using UnityEngine;

public class Fish : MonoBehaviour
{
    [Tooltip("The swim speed")]
    public float fishSpeed;

    private float randomizedSpeed = 0f;
    private float nextActionTime = -1f;
    private Vector3 targetPosition;
}
```

FixedUpdate is called at a regular interval of 0.02 seconds (it is independent of frame rate) and will allow us to interact even when the agent is training at an increased game speed, which is common for training ML-Agents. In it, we check if the fish should swim and, if so, call the Swim() function.

- Add a new FixedUpdate() function.

```
/// <summary>
    /// Called every timestep
    /// </summary>
    private void FixedUpdate()
    {
        if (fishSpeed > 0f)
        {
            Swim();
```

Next, we'll add swim functionality. At any given update, the fish will either pick a new speed and destination, or move toward its current destination.

When it is time to take a new action, the fish will:

- Choose a new randomized speed between 50% and 150% of the average fish speed.
- Pick a new random target position (in the water) to swim toward.
- Rotate the fish to face the target.
- Calculate the time needed to get there.

Otherwise, the fish will move toward the target and make sure it doesn't swim past it.

- Add a new Swim() function.

```
/// <summary>
  /// Swim between random positions
  /// </summary>
  private void Swim()
  {
      // If it's time for the next a
      // Else, swim toward the desti
      if (Time.fixedTime >= nextActi
      {
          // Randomize the speed
          randomizedSpeed = fishSpee

          // Pick a random target
          targetPosition = PenguinAr

          // Rotate toward the targe
```

```
            // Calculate the time to 
            float timeToGetThere = Vec
            nextActionTime = Time.fixe
        }
        else
        {
            // Make sure that the fis
            Vector3 moveVector = rand
            if (moveVector.magnitude 
            {
                transform.position +=
            }
            else
            {
                transform.position = 
                nextActionTime = Time
            }
        }
    }
```

That's all for the Fish script!

CONCLUSION

You should now have all of the code you need to train the penguins to catch fish and feed their babies. In the next tutorial, you will set up your Scene to use this code.

TUTORIAL PARTS

Reinforcement Learning Penguins (Part 1/4)

Reinforcement Learning Penguins (Part 2/4)

Reinforcement Learning Penguins (Part 3/4)

Reinforcement Learning Penguins (Part 4/4)

IMMERSIVE LIMIT

COMMENTS (13) Most Liked Subscribe via e-mail

Preview Post Comment…

Thorbjørn Rysgaard

2 months ago · 1 Like

Hi thanks for making this tutorial!

It it my first try at ml in Unity and I have some trouble with this tutorial. The GetStepCount() function called in FixedUpdate is not implemented.

Then there are some new way of calling CollectObservations() that now needs a parameter like this: CollectObservations(VectorSensor sensor).

This parameter is now used to call the functions inside like this:

```
sensor.AddObservation(isFull);
sensor.AddObservation(Vector3.D
sensor.AddObservation((baby.tra
sensor.AddObservation(transform
```

Duddumpudi

2 weeks ago · 1 Like

As of v0.15.1 this can now be resolved by :

public override void CollectObservations(MLAgents.Sensors.VectorSensor sensor)
{
// Whether the penguin has eaten a fish (1 float = 1 value)
sensor.AddObservation(isFull);

```
    // Distance to the baby (1
    sensor.AddObservation(Vect

    // Direction to baby (1 Ve

    sensor.AddObservation((bab

    // Direction penguin is fa
    sensor.AddObservation(tran

    // 1 + 1 + 3 + 3 = 8 total
&#125;
```

Krishna Duddumpudi

2 weeks ago · 0

Likes

Again, as of v0.15.1, the GetStepCount() method can be replaced with StepCount https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Migrating.md

Aakash Jha 2 weeks ago ·

0 Likes

I am getting 2 errors and a warning and im stuck.
mlagents version : 0.15.1
Assets\Penguin\Scripts\PenguinAgent.cs(95,26): error CS0115: 'PenguinAgent.OnEpsiodeBegin()': no suitable method found to override

EndLayoutGroup: BeginLayoutGroup must be called first.
UnityEngine.GUIUtility:ProcessEvent(Int32, IntPtr, Boolean&)

Should not be capturing when there is a hotcontrol
UnityEngine.GUIUtility:ProcessEvent(Int32, IntPtr, Boolean&)

Danai Kal

2 weeks ago · 0 Likes

I think that OnEpisodeBegin() function has been replaced by InitializeAgent(function)!

IMMERSIVE LIMIT

A week ago · 0 Likes

Try using the v0.14 release for this tutorial so that you understand how ML-Agents works, then you can try using v0.15+. There are lots of small changes that will make it tricky to follow, but the fundamental concepts are the same.

Tsung Wei Hsu

4 weeks ago · 0 Likes

Hey,
thanks you guys for this awesome tutorial! At the beginning, I had hard time starting to write the script because of the missing class "Area". It seems like the class has be moved to other dependency. Instead of using "MLAgents", I used "MLAgentsExamples" to locate the "Area" class. I don't know whether others also had this kind of problem.

Adam Kelly

3 weeks ago · 0 Likes

Thanks for the heads up. The examples code was recently removed from ML-Agents and I forgot to remove that dependency. It's not needed as there's no actual functionality in the Area class that is needed.

A month ago · 0 Likes

Hi,
I Have a problem with inheiriting class Area. It says that namespace name cant be found. I have done everything just like in these tutorial ('using MLAgents', added package). I've been stuck with this for a long time and couldnt find any solutions on google. I am using 0.14.1 version. Just wanted to add that basic project with 3dBalls worked properly.

Tsung Wei Hsu

4 weeks ago · 0 Likes

You could use my solution, add "using MLAgentsExamples" instead of "MLAgents". I found this solution in one of the official examples.

Thorbjørn Rysgaard

2 months ago · 0 Likes

Hi again,
I added the GetStepCount() function. But the change to CollectObservations() was not correct with the version 14. My bad! So now it is running and making my computer sweat ;-)

2 months ago · 0 Likes

I'm pretty sure this happened because you are using the "master" branch of the code instead of using the v0.14 release from the Releases page. The master branch is impossible to make a tutorial for because it changes daily. This tutorial uses the v0.14 release. https://github.com/Unity-Technologies/ml-agents/releases/tag/0.14.0

Thorbjørn Rysgaard

2 months ago · 0 Likes

You are right! After using the v0.14 release it worked.

PREVIOUS

Reinforcement Learning Penguins (Part 3/4) | Unity ML-Agents

NEXT

Reinforcement Learning Penguins (Part 1/4) | Unity ML-Agents

I ERSIVE LIMIT

ABOUT CONNECT