HW5 – Train a walker

- HW5 asks you to train a humanoid walker using PPO and SAC provided by Unity ML agent and the SAC code in Prof. Sun's GitHub.
- Train 5~7M steps and compare rewards, policy loss, actor loss at 1M, 3M, 5M, … as well as the test performance. (Refer to HW1)
- Due: next class meeting
- You can do this homework in a group with 1 to 3 members
- Upload ppt to Teams

Result and discussion – Training performance

	Training performance								
	Reward			Policy loss			Actor loss		
	1M	3M	…	1M	3M	…	1M	3M	…
PPO									
SAC									
SAC (Prof. code)									

Result and discussion – Test performance

1M

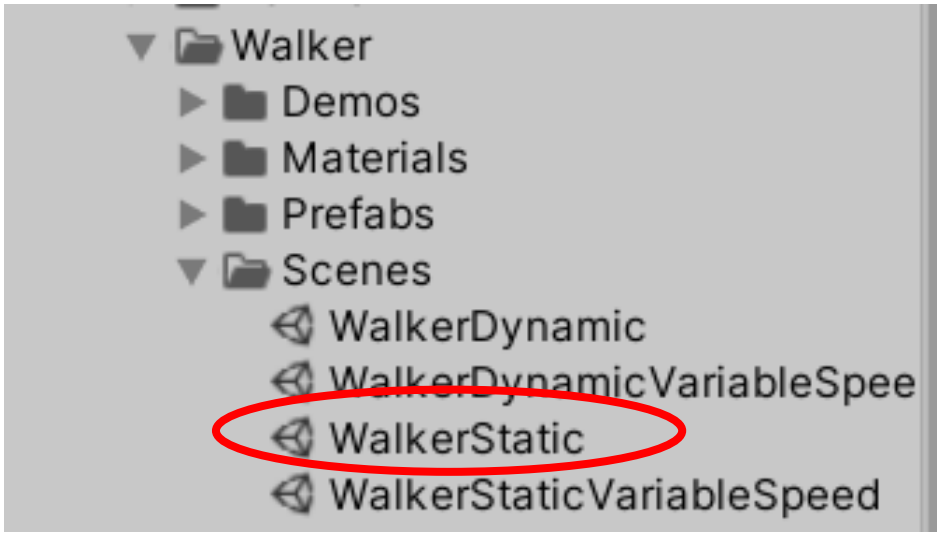
3M

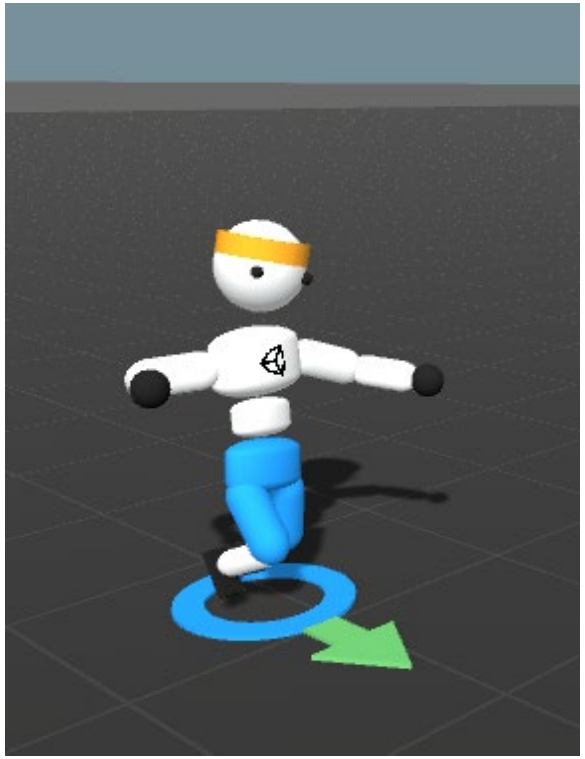
5M

...

Walker and goal

- Set-up: Physics-based Humanoid agents with 26 degrees of freedom. These DOFs correspond to articulation of the following body-parts: hips, chest, spine, head, thighs, shins, feet, arms, forearms and hands.
- Goal: The agents must move its body toward the goal direction without falling.





https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Examples.md

Reward

- Agent Reward Function (independent): The reward function is now geometric meaning the reward each step is a product of all the rewards instead of a sum, this helps the agent try to maximize all rewards instead of the easiest rewards.
  - Body velocity matches goal velocity. (normalized between (0,1))
  - Head direction alignment with goal direction. (normalized between (0,1))

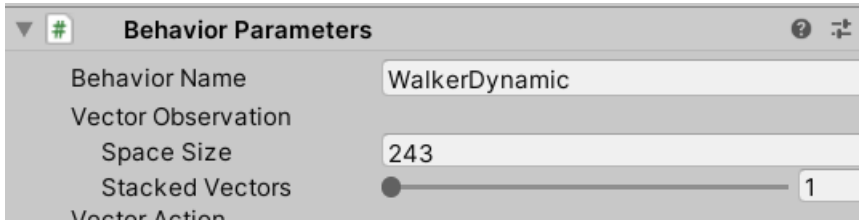
```
AddReward(matchSpeedReward * lookAtTargetReward);
```

```
public void TouchedTarget()
{
    AddReward(1f);
```

```
// Set reward for this step according to mixture of the following elements.
// a. Match target speed
//This reward will approach 1 if it matches perfectly and approach zero as it deviates
var matchSpeedReward = GetMatchingVelocityReward(cubeForward * MTargetWalkingSpeed, GetAvgVelocity());

// b. Rotation alignment with target direction.
//This reward will approach 1 if it faces the target direction perfectly and approach zero as it deviates
var lookAtTargetReward = (Vector3.Dot(cubeForward, head.forward) + 1) * .5F;
```

State

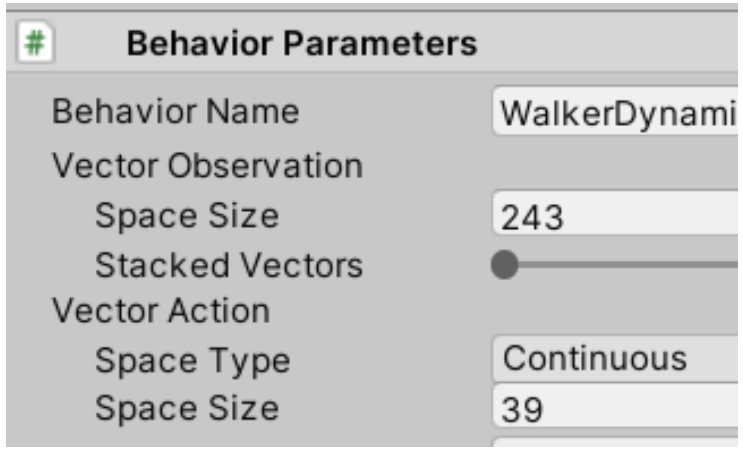
- Behavior Parameters:
  - Vector Observation space: 243 variables corresponding to position, rotation, velocity, and angular velocities of each limb, along with goal direction.




```
//current ragdoll velocity. normalized
sensor.AddObservation(Vector3.Distance(velGoal, avgVel));
//avg body vel relative to cube
sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(avgVel));
//vel goal relative to cube
sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(velGoal));

//rotation deltas
sensor.AddObservation(Quaternion.FromToRotation(hips.forward, cubeForward));
sensor.AddObservation(Quaternion.FromToRotation(head.forward, cubeForward));

//Position of target position relative to cube
sensor.AddObservation(m_OrientationCube.transform.InverseTransformPoint(target.transform.

foreach (var bodyPart in m_JdController.bodyPartsList)
{
    CollectObservationBodyPart(bodyPart, sensor);
}
```

Actions

- Behavior Parameters:
  - Actions: 39 continuous actions, corresponding to target rotations and strength applicable to the joints.

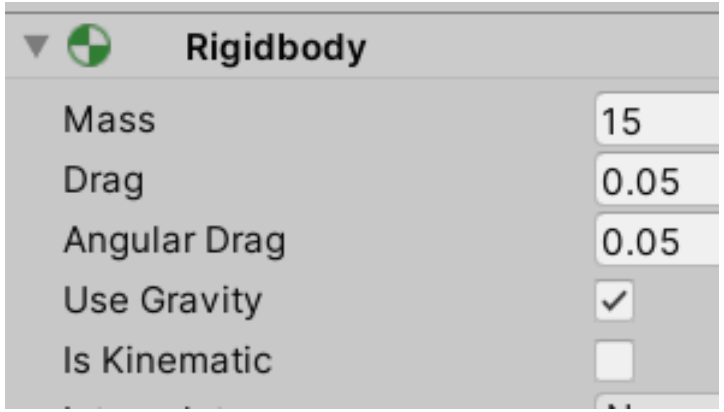



```
public override void OnActionReceived(ActionBuffers actionBuffers)

{
    var bpDict = m_JdController.bodyPartsDict;
    var i = -1;

    var continuousActions = actionBuffers.ContinuousActions;
    bpDict[chest].SetJointTargetRotation(continuousActions[++i], conti
    bpDict[spine].SetJointTargetRotation(continuousActions[++i], conti

    bpDict[thighL].SetJointTargetRotation(continuousActions[++i], cont
    bpDict[thighR].SetJointTargetRotation(continuousActions[++i], cont
    bpDict[shinL].SetJointTargetRotation(continuousActions[++i], 0, 0)
    bpDict[shinR].SetJointTargetRotation(continuousActions[++i], 0, 0)
    bpDict[footR].SetJointTargetRotation(continuousActions[++i], conti
    bpDict[footL].SetJointTargetRotation(continuousActions[++i], conti

    bpDict[armL].SetJointTargetRotation(continuousActions[++i], contin
    bpDict[armR].SetJointTargetRotation(continuousActions[++i], contin
    bpDict[forearmL].SetJointTargetRotation(continuousActions[++i], 0,
    bpDict[forearmR].SetJointTargetRotation(continuousActions[++i], 0,
    bpDict[head].SetJointTargetRotation(continuousActions[++i], contin
```

Gravity and mass

Float Properties: Four

- gravity: Magnitude of gravity
  - Default: 9.81
  - Recommended Minimum:
  - Recommended Maximum:
- Hip mass: Mass of the hip component of the walker
  - Default: 8
  - Recommended Minimum: 7
  - Recommended Maximum: 28
- Chest mass: Mass of the chest component of the walker
  - Default: 8
  - Recommended Minimum: 3
  - Recommended Maximum: 20
- Spine mass: Mass of the spine component of the walker
  - Default: 8
  - Recommended Minimum: 3
  - Recommended Maximum: 20



Yaml file

```
WalkerStatic:
  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
  network_settings:
    normalize: true
    hidden_units: 512
    num_layers: 3
    vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.995
    strength: 1.0
keep_checkpoints: 5
max_steps: 30000000
time_horizon: 1000
summary_freq: 30000
threaded: true
```

```
WalkerStatic:
  trainer_type: sac
  hyperparameters:
    learning_rate: 0.0003
    learning_rate_schedule: constant
    batch_size: 1024
    buffer_size: 2000000
    buffer_init_steps: 0
    tau: 0.005
    steps_per_update: 30.0
    save_replay_buffer: false
    init_entcoef: 1.0
    reward_signal_steps_per_update: 30.0
  network_settings:
    normalize: true
    hidden_units: 256
    num_layers: 3
    vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.995
    strength: 1.0
keep_checkpoints: 5
max_steps: 15000000
time_horizon: 1000
summary_freq: 30000
threaded: true
```

Benchmark Mean Reward : 2500