Gymnasium

Stable Baselines

Customizing OpenAi Gym Environment With Sable Baselines Reinforcement Learning Agents

Grupo W

André Sousa 202108571

David Scarin 202108314

Paulo Silva 202107359

Environment Description

id (for v2, v3, v4)	body part
0	worldBody (note: all values are constant 0)
1	torso
2	lwaist
3	pelvis
4	right_thigh
5	right_sin
6	right_foot
7	left_thigh
8	left_sin
9	left_foot
10	right_upper_arm
11	right_lower_arm
12	left_upper_arm
13	left_lower_arm

id (for v2, v3, v4)	joint
0	root
1	root
2	root
3	root
4	root
5	root
6	abdomen_z
7	abdomen_y
8	abdomen_x
9	right_hip_x
10	right_hip_z
11	right_hip_y
12	right_knee
13	left_hip_x
14	left_hiz_z
15	left_hip_y
16	left_knee
17	right_shoulder1
18	right_shoulder2
19	right_elbow
20	left_shoulder1
21	left_shoulder2
22	left_elfbow

Action Space	`Box(-0.4, 0.4, (17,), float32)`
Observation Space	`Box(-inf, inf, (376,), float64)`

Environment Description

Rewards

The reward consists of three parts:

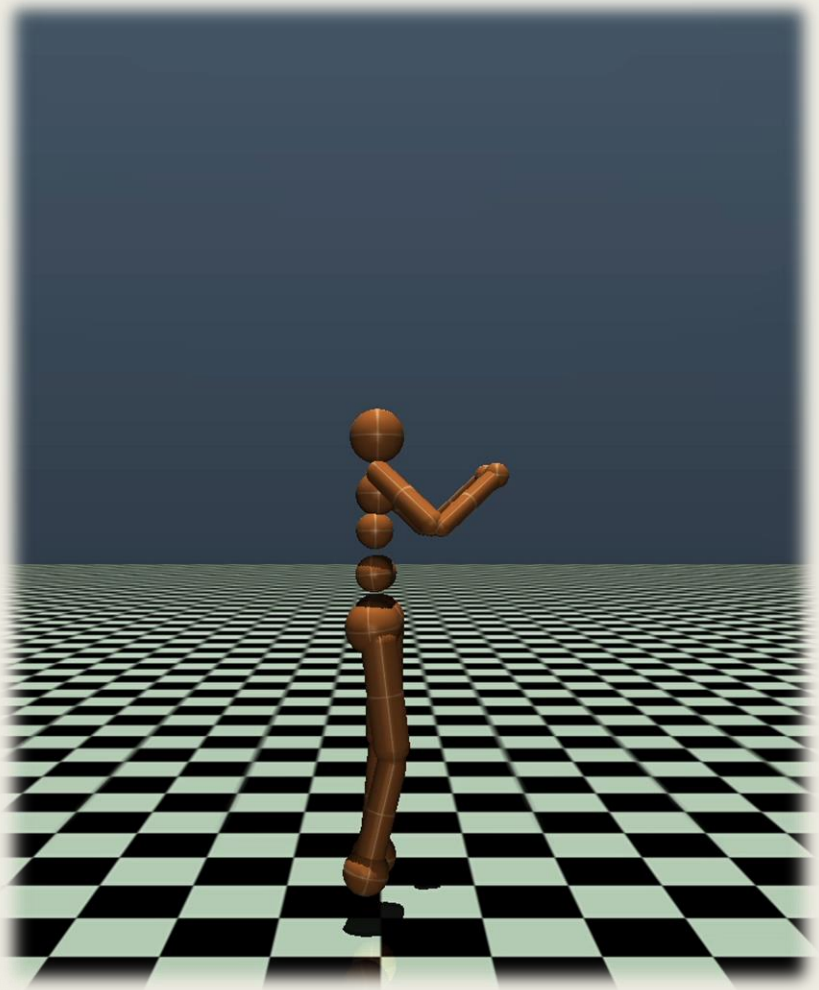
- *healthy_reward*: Every timestep that the humanoid is alive (see section Episode Termination for definition), it gets a reward of fixed value `healthy_reward`
- *forward_reward*: A reward of walking forward which is measured as `forward_reward_weight` * *(average center of mass before action - average center of mass after action)/dt*. *dt* is the time between actions and is dependent on the frame_skip parameter (default is 5), where the frametime is 0.003 - making the default *dt = 5 * 0.003 = 0.015*. This reward would be positive if the humanoid walks forward (in positive x-direction). The calculation for the center of mass is defined in the `.py` file for the Humanoid.
- *ctrl_cost*: A negative reward for penalising the humanoid if it has too large of a control force. If there are *nu* actuators/controls, then the control has shape `nu x 1`. It is measured as `ctrl_cost_weight` * *sum(control^2)*.
- *contact_cost*: A negative reward for penalising the humanoid if the external contact force is too large. It is calculated by clipping `contact_cost_weight` * *sum(external contact force^2)* to the interval specified by `contact_cost_range`.

The total reward returned is ***reward*** = *healthy_reward* + *forward_reward* - *ctrl_cost* - *contact_cost* and `info` will also contain the individual reward terms

`forward_reward_weight`	**float**	`1.25`	Weight for *forward_reward* term (see section on reward)
`ctrl_cost_weight`	**float**	`0.1`	Weight for *ctrl_cost* term (see section on reward)
`contact_cost_weight`	**float**	`5e-7`	Weight for *contact_cost* term (see section on reward)
`healthy_reward`	**float**	`5.0`	Constant reward given if the humanoid is “healthy” after timestep

Developed Work

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
ARS [1]	✔	✔	✖	✖	✔
A2C	✔	✔	✔	✔	✔
DDPG	✔	✖	✖	✖	✔
DQN	✖	✔	✖	✖	✔
HER	✔	✔	✖	✖	✔
PPO	✔	✔	✔	✔	✔
QR-DQN [1]	✖	✔	✖	✖	✔
RecurrentPPO [1]	✔	✔	✔	✔	✔
SAC	✔	✖	✖	✖	✔
TD3	✔	✖	✖	✖	✔
TQC [1]	✔	✖	✖	✖	✔
TRPO [1]	✔	✔	✔	✔	✔
Maskable PPO [1]	✖	✔	✔	✔	✔



Untrained Agent

```
agent = PPO('MlpPolicy', wrapped_env_train_1, verbose=0, tensorboard_log="./logs_changed")
```

Developed Work

- Separate training into 2 phases
- Split of 65/35 between first and second phases

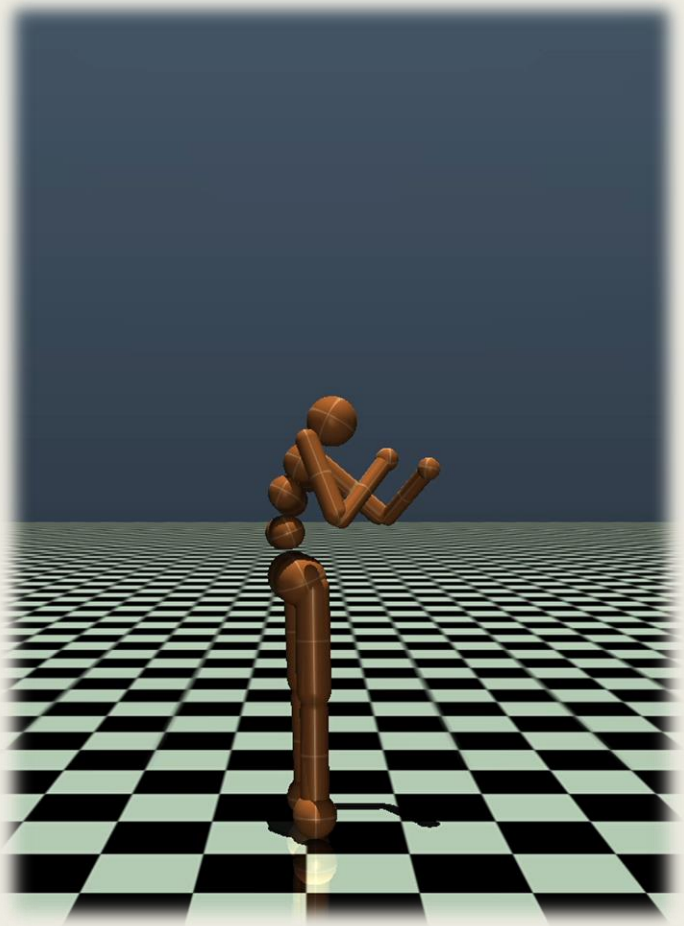
```
print(f"Starting 1st phase of training")
agent = PPO('MlpPolicy', wrapped_env_train_1, verbose=0, tensorboard_log="./logs_changed")
agent.learn(total_timesteps=700000,progress_bar=True, tb_log_name="PPO_1_"+str(seed), reset_num_timesteps=False)
agent.save("agent_changed_1_"+str(seed))
del agent
env_train_1.close()

print(f"Starting 2nd phase of training")
agent = PPO.load("agent_changed_1_"+str(seed), env = wrapped_env_train_2)
agent.learn(total_timesteps=1300000,progress_bar=True, tb_log_name="PPO_2_"+str(seed), reset_num_timesteps=False)
agent.save("agent_changed_2_"+str(seed))
del agent
env_train_2.close()
```

Developed Work

Testing environment

```
env_test = gym.make("Humanoid-v4", healthy_z_range=(0.9,2.0))
```

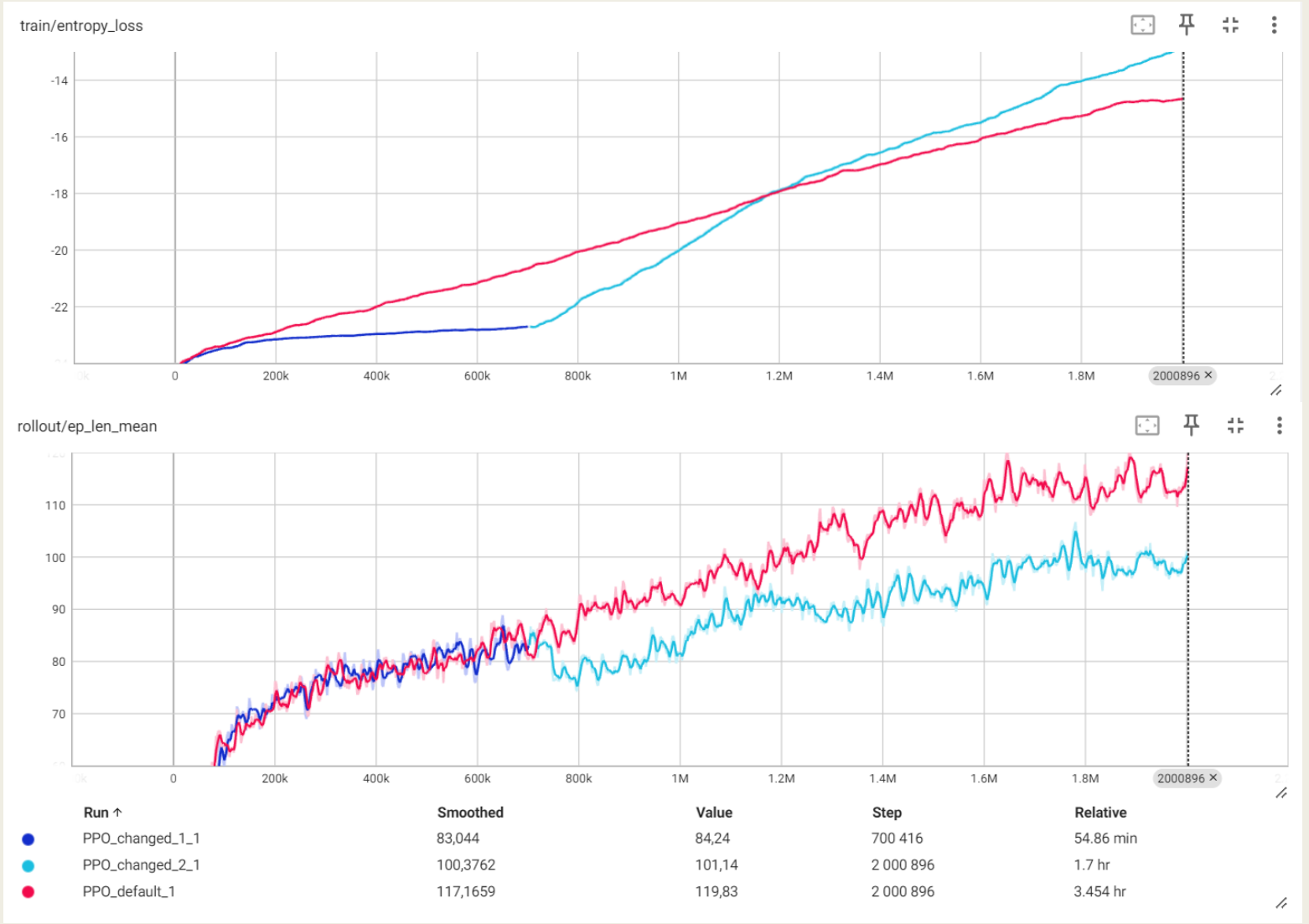


First phase of training with changed rewards

- 2 training phases with different weights

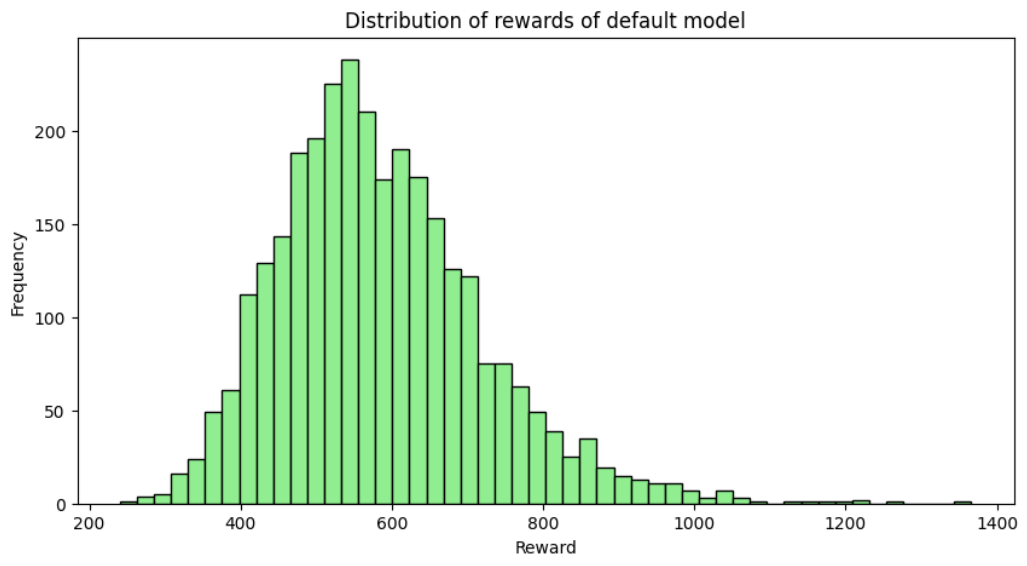
```
env_train_1 = gym.make("Humanoid-v4", healthy_reward = 10, forward_reward_weight = 0.625,  healthy_z_range=(0.9,2.0))
env_train_2 = gym.make("Humanoid-v4", healthy_reward = 2.5, forward_reward_weight = 2.5, healthy_z_range=(0.9,2.0))
env_test = gym.make("Humanoid-v4", healthy_z_range=(0.9,2.0))
```

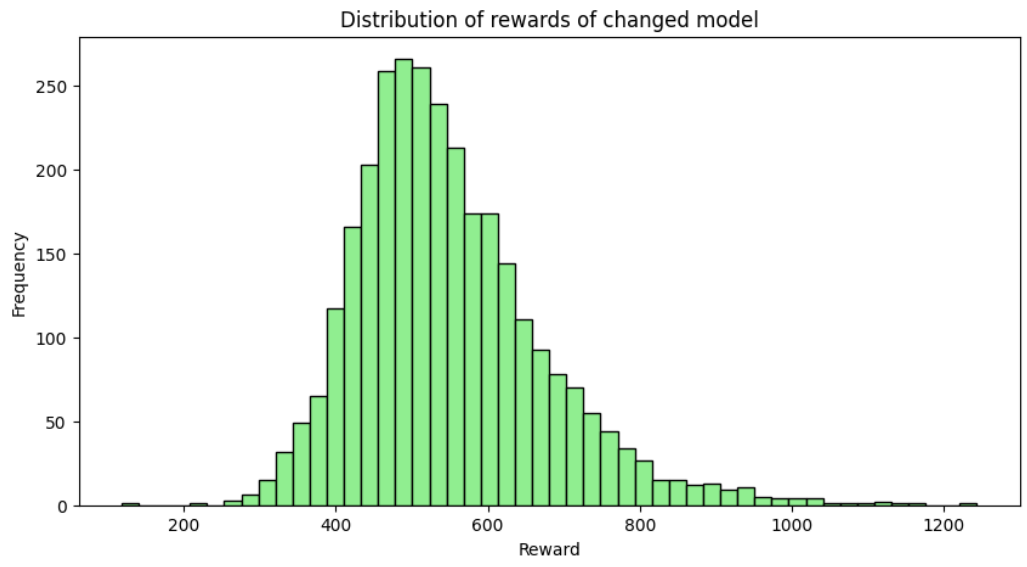
Training curves

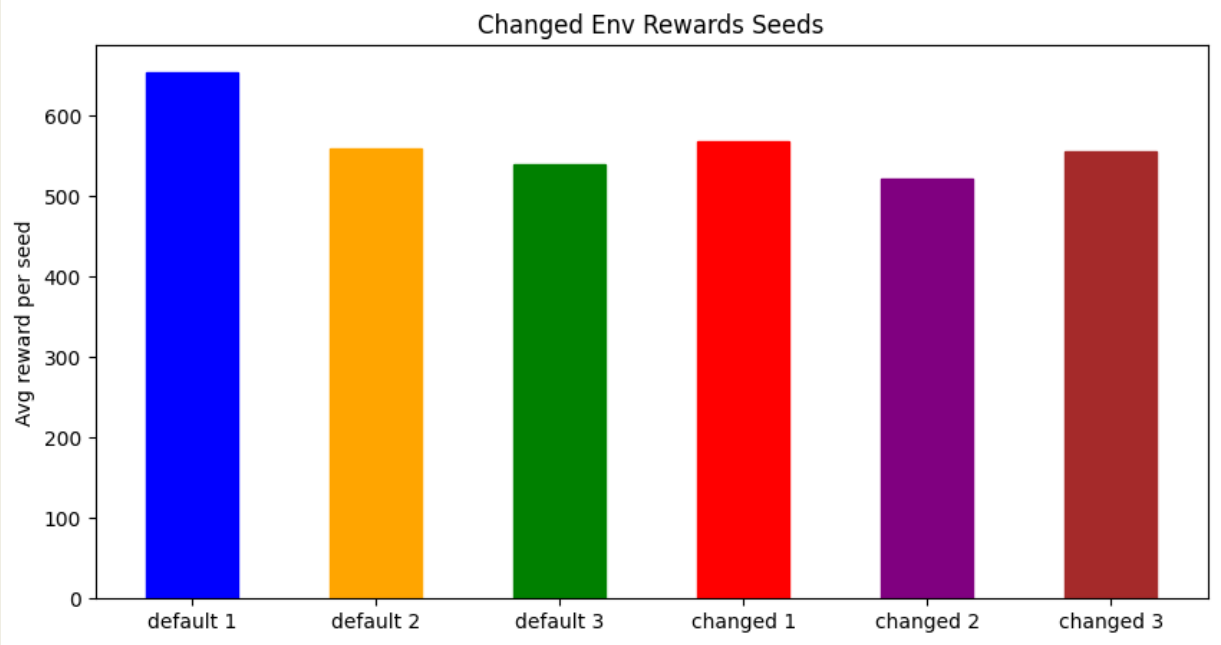


Run	Smoothed	Value	Step	Relative
PPO_changed_1_1	83,044	84,24	700 416	54.86 min
PPO_changed_2_1	100,3762	101,14	2 000 896	1.7 hr
PPO_default_1	117,1659	119,83	2 000 896	3.454 hr

Reward Analysis



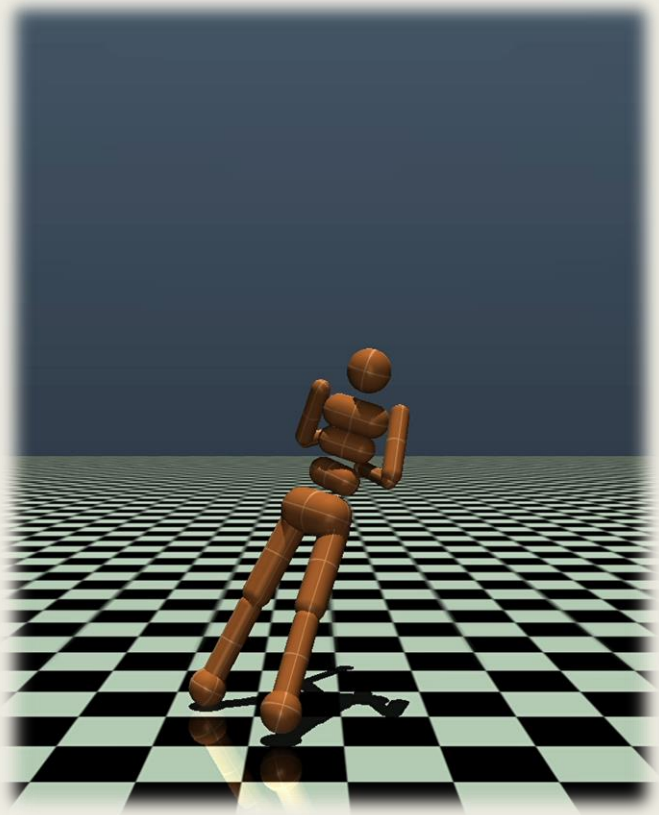




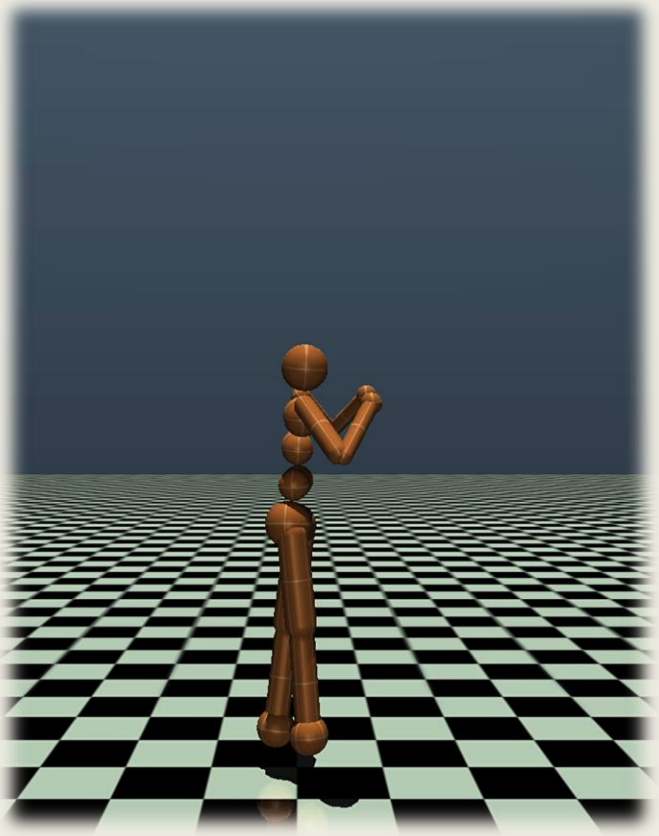

```
Test Results
statistic: 10.696914047665128
degrees of freedom: 5955.495487925635
pvalue: 1.827726145871 55e-26
```

Comparing Results

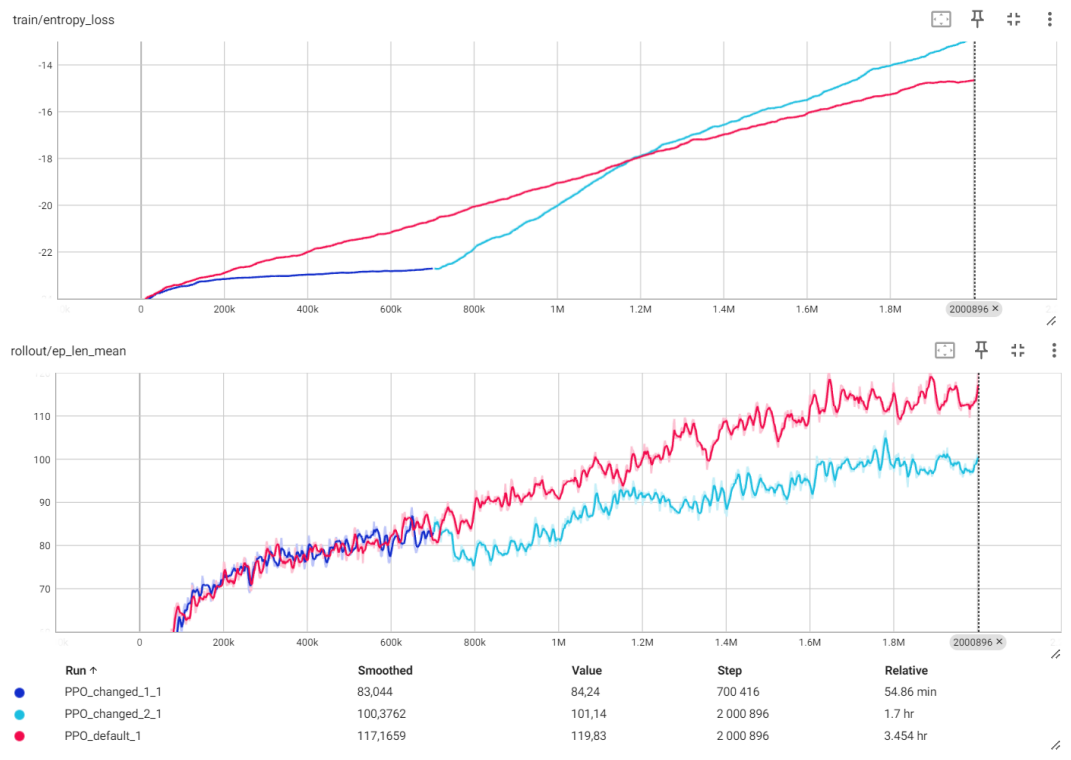
Agent trained on default environment



Agent trained on changed environment



Conclusions and Future Work




```
Test Results
statistic: 10.696914047665128
degrees of freedom: 5955.495487925635
pvalue: 1.82772614587155e-26
```

The results return a very low p-value, indicating that there is indeed a significant difference in the means of the two sets of rewards.

Therefore, we can state that there is a difference in the agent trained with or without the custom changes to the environment.

We verify that the agent trained with the default reward settings achieved a better performance for this number of timesteps in training.

However, we can see from the training curves that our changed agent appears to converge faster, so it would be a worthwhile experiment to further train our agents for more timesteps.

Another possible option is to attempt to finetune our learning algorithm by changing parameters such as learning rate and optimizer or even the architecture of the network.