Name : Pusaka Manggala
NIM : 1103194021

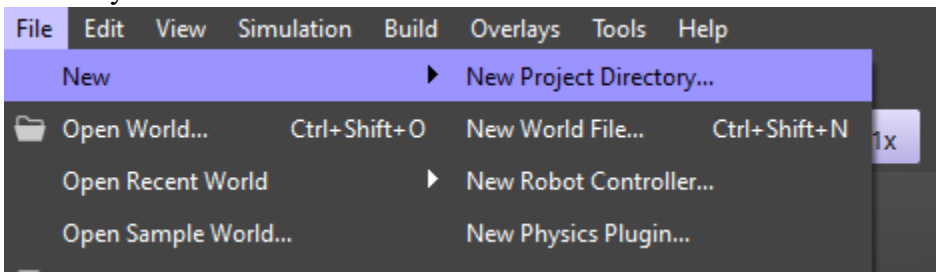
Deepbots - CartPole Beginner
Robot-Supervisor Scheme Tutorial

Prerequsites

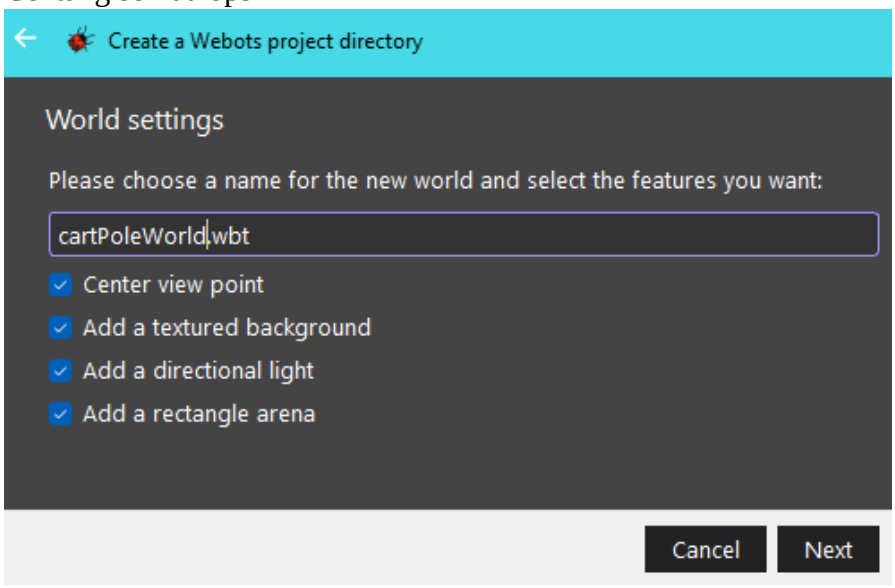
- Deepbots : pip install deepbots
- Python version 3.X.X (disarankan versi 3.8.3)
- Webots (disarankan versi R2020a)

Membuat project

1. Buka Webots, lalu lihat menu bar dan klik File -> New -> New Project Directory…

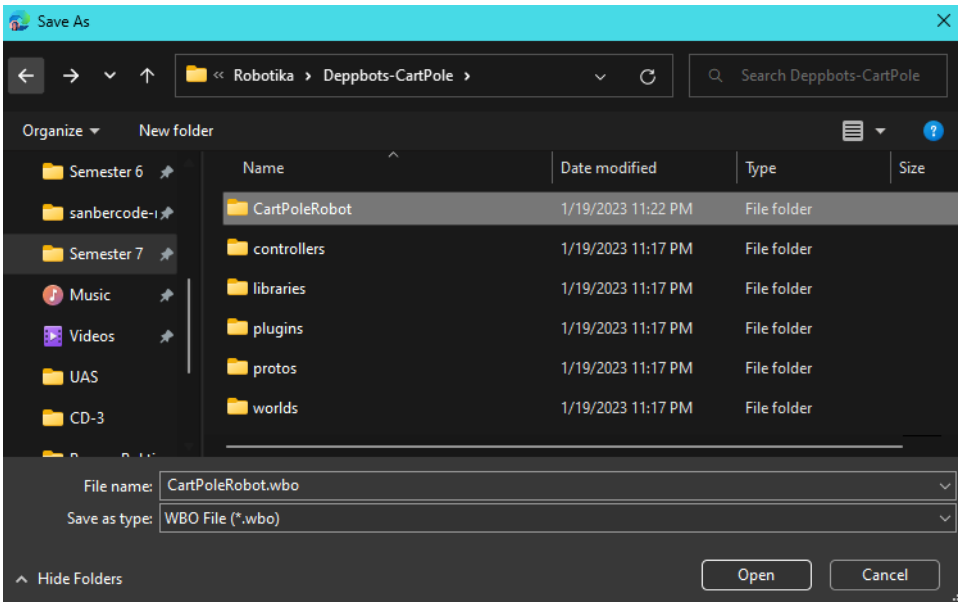


2. Centang semua opsi

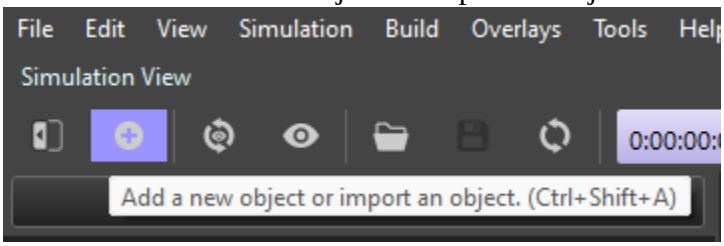


Menambahkan supervisor robot node

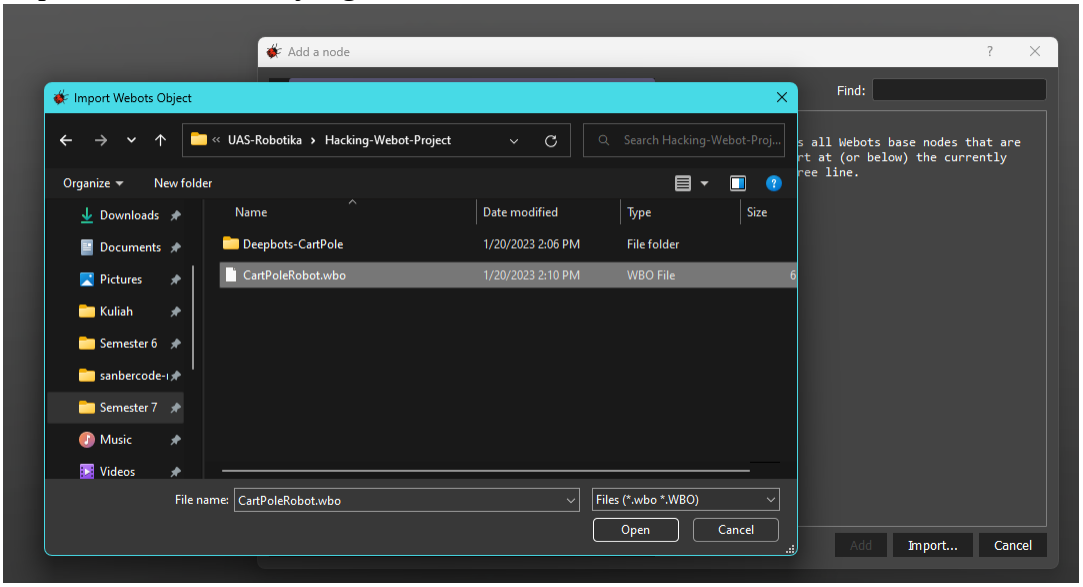
1. Download CartPole robot



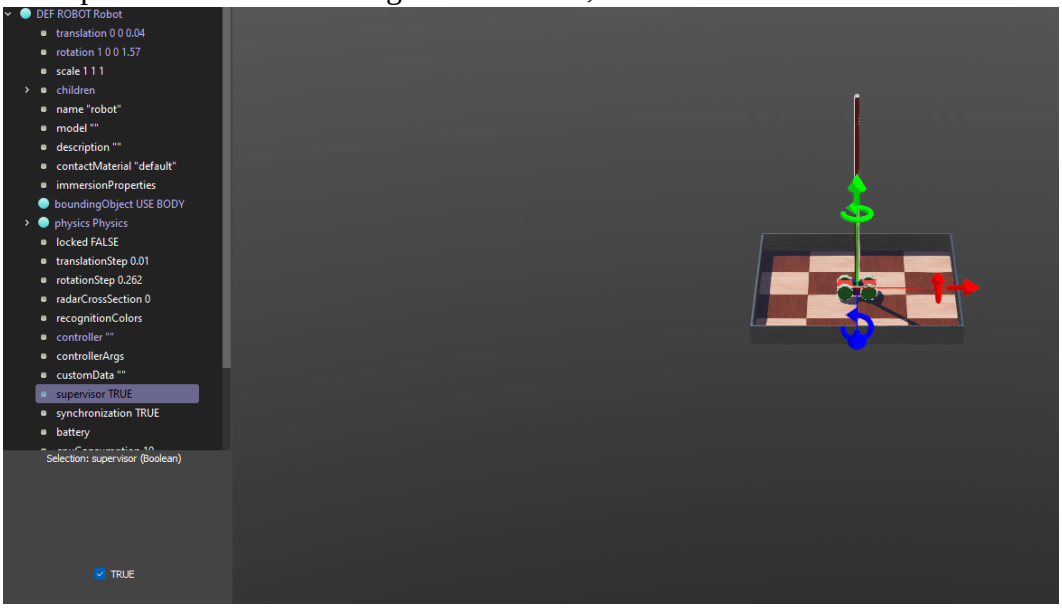
2. Klik tombol add a new object or import an object



3. Import file robot .wbo yang sudah didownload

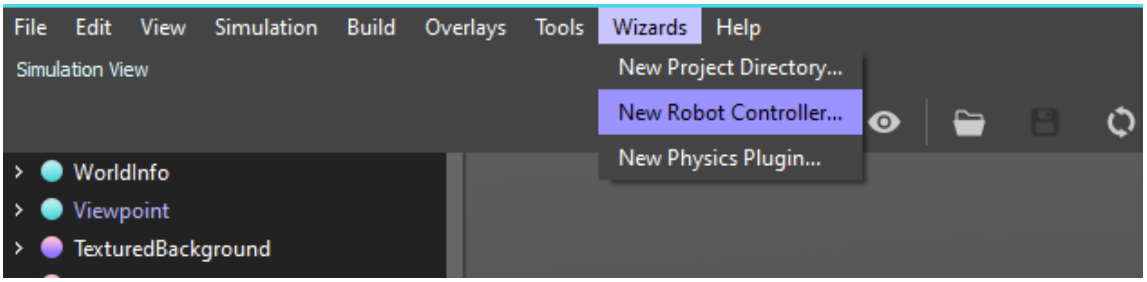


4. Set supervisor ke TRUE di bagian node robot, lalu klik save

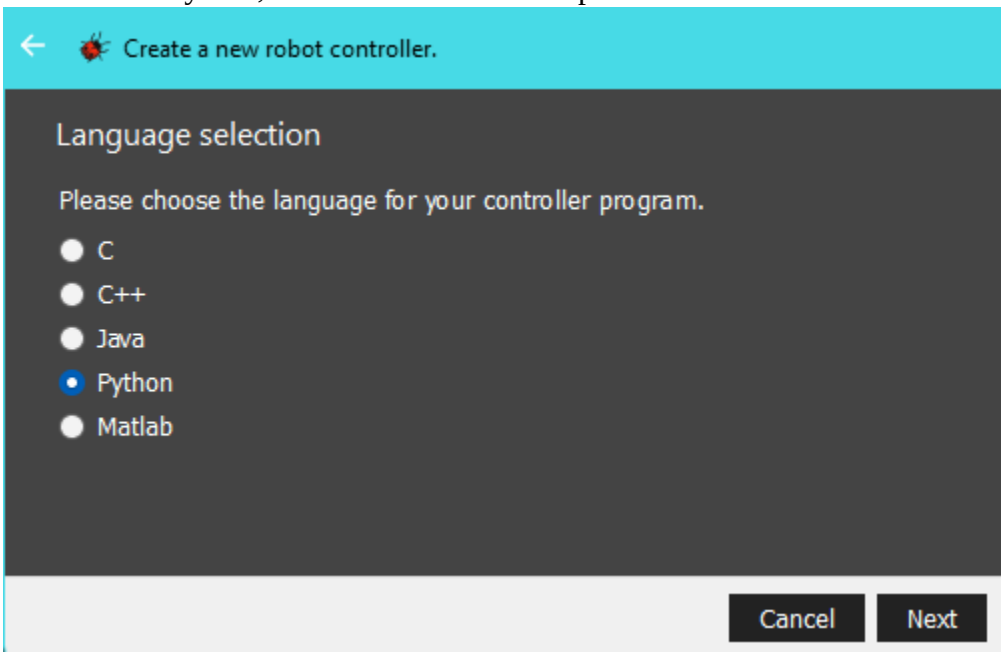


Menambahkan Controller

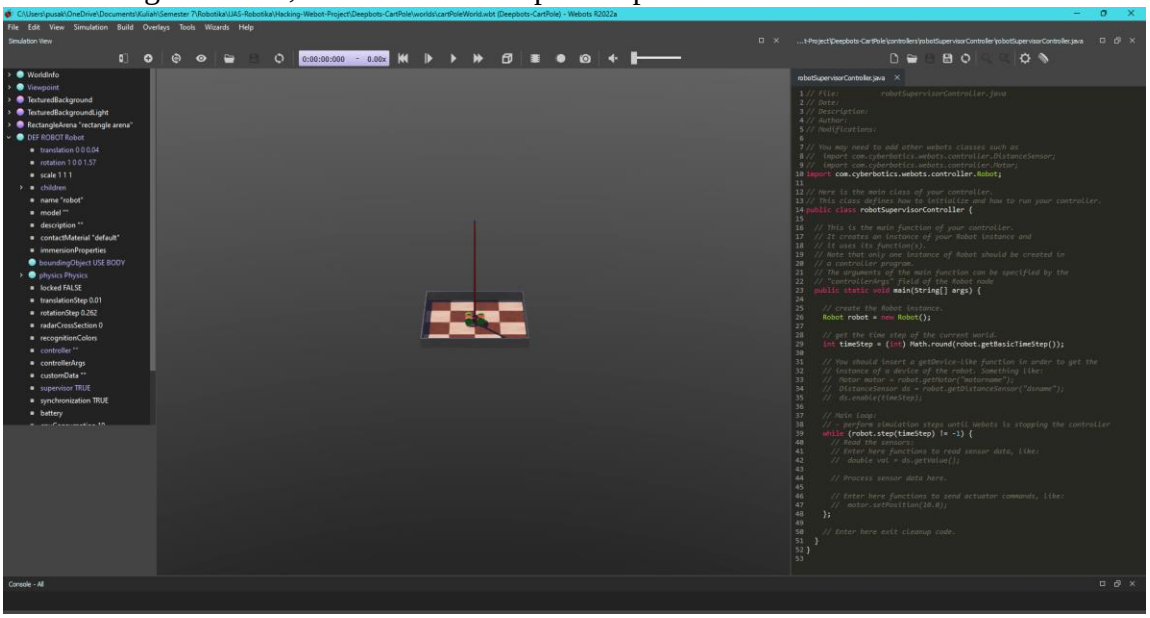
1. Pada bar menu, klik "Wizards -> New Robot Controller..."



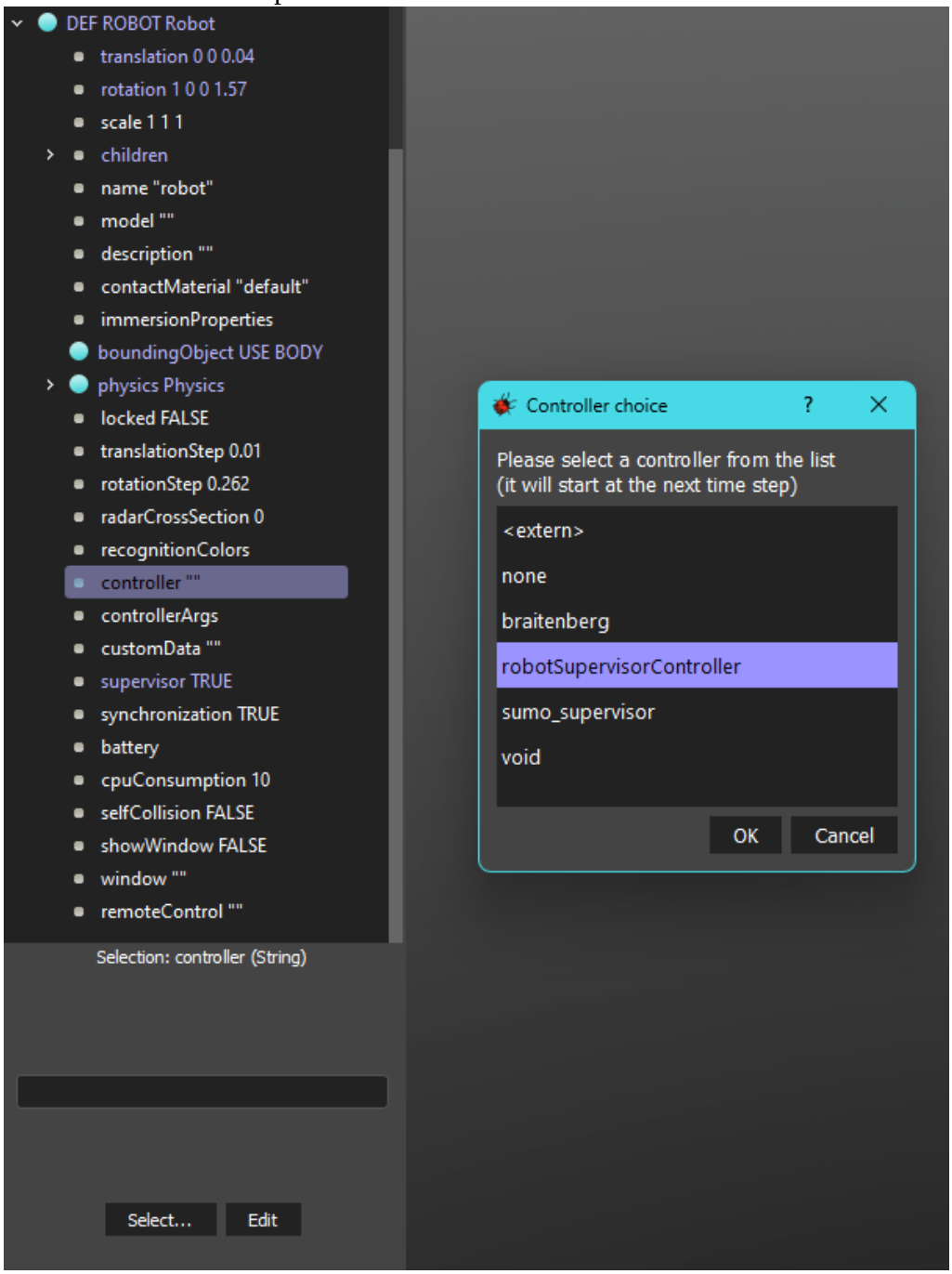
2. Pilih bahasa Python, dan beri nama “robotSupervisorController”



3. Klik Finish
4. Setelah langkah diatas, maka muncul tampilan seperti ini



5. Tambahkan controller pada node robot



6. Klik OK dan Save Project

Pengkodean

1. Download file PPO agent pada link ini
2. Download utilities script pada link ini
3. Simpan kedua file tersebut kedalam Controllers/robotSupervisorController/
4. Hapus semua code yang ada pada robotSupervisorController

5. Import RobotSupervisor class beserta beberapa fungsi utility, implementasi PPO agent, gym space untuk aksi dan ruang observasi, dan numpy.

```
from deepbots.supervisor.controllers.robot_supervisor import RobotSupervisor
from utilities import normalizeToRange, plotData
from PPO_agent import PPOAgent, Transition

from gym.spaces import Box, Discrete
import numpy as np
```

6. Lalu kita memerlukan sebuah class yang menurunkan beberapa fungsi penting

```
class CartpoleRobot(RobotSupervisor):
    def __init__(self):
        super().__init__()
```

7. Ruang aksi menentukan output dari neural network, yaitu 2. Satu untuk gerakan maju dan satu untuk gerakan mundur robot.

```
        self.observation_space = Box(low=np.array([-0.4, -np.inf, -1.3, -np.inf]),
                                     high=np.array([0.4, np.inf, 1.3, np.inf]),
                                     dtype=np.float64)
        self.action_space = Discrete(2)
```

8. Lalu kita dapatkan referensi kepada node robot, inisiasi pole sensor, dapatkan referensi untuk endpoint pole dan inisiasi roda motor.

```
        self.robot = self.getSelf()  # Grab the robot reference from the supervisor to access various robot methods
        self.positionSensor = self.getDevice("polePosSensor")
        self.positionSensor.enable(self.timestep)

        self.poleEndpoint = self.getFromDef("POLE_ENDPOINT")
        self.wheels = []
        for wheelName in ['wheel1', 'wheel2', 'wheel3', 'wheel4']:
            wheel = self.getDevice(wheelName)  # Get the wheel handle
            wheel.setPosition(float('inf'))  # Set starting position
            wheel.setVelocity(0.0)  # Zero out starting velocity
            self.wheels.append(wheel)
```

9. Inisiasi beberapa variable yang dipakai ketika percobaan. Perhatikan bahwa self.stepsPerEpisode di set 200 berdasarkan masalah yang didefinisikan. Ini termasuk metode __init__()

```
self.stepsPerEpisode = 200  # Max number of steps per episode
self.episodeScore = 0  # Score accumulated during an episode
self.episodeScoreList = []  # A list to save all the episode scores, used to check if task is solved
```

10. Kita akan mulai untuk implementasi beberapa metode yang dibutuhkan. Dimulai dengan metode get_observations(), lalu metode utility normalizeToRange() untuk normalisasi nilai ke jangkauan [-1.0, 1.0].

```
def get_observations(self):
    # Position on x axis
    cartPosition = normalizeToRange(self.robot.getPosition()[0], -0.4, 0.4, -1.0, 1.0)
    # Linear velocity on x axis
    cartVelocity = normalizeToRange(self.robot.getVelocity()[0], -0.2, 0.2, -1.0, 1.0, clip=True)
    # Pole angle off vertical
    poleAngle = normalizeToRange(self.positionSensor.getValue(), -0.23, 0.23, -1.0, 1.0, clip=True)
    # Angular velocity y of endpoint
    endpointVelocity = normalizeToRange(self.poleEndpoint.getVelocity()[4], -1.5, 1.5, -1.0, 1.0, clip=True)

    return [cartPosition, cartVelocity, poleAngle, endpointVelocity]
```

11. Sekarang kita definisikan metode get_reward(), yang mengembalikan nilai 1 untuk setiap step

```
  def get_reward(self, action=None):
    return 1
```

12. Definiskan metode is_done(), yang berisi beberapa episode kondisi :
    - Episode 1 jika score lebih dari 195.0
    - Episode 2 jika pole sudah jatuh melebihi sudut yang masih bisa dipulihkan (+- 15 degrees)
    - Episode 2 jika robot menabrak tembok, yang dihitung berdasarkan posisi z axis

```
  def is_done(self):
    if self.episodeScore > 195.0:
      return True

    poleAngle = round(self.positionSensor.getValue(), 2)
    if abs(poleAngle) > 0.261799388:  # 15 degrees off vertical
      return True

    cartPosition = round(self.robot.getPosition()[2], 2)  # Position on z axis
    if abs(cartPosition) > 0.39:
      return True

    return False
```

13. Kita pisahkan kondisi solved ke metode lain, metode solved() karena membutuhkan handling berbeda. Kondisi solved bergantung pada episode yang diselesaikan.

```
  def solved(self):
    if len(self.episodeScoreList) > 100:  # Over 100 trials thus far
      if np.mean(self.episodeScoreList[-100:]) > 195.0:  # Last 100 episodes' scores average value
        return True
    return False
```

14. Implementasi metode get_default_observations(), yang berfungsi untuk mengembalikan nilai 0 vector. Singkat nya metode ini bertujuan untuk mereset robot.

```
  def get_default_observation(self):
    return [0.0 for _ in range(self.observation_space.shape[0])]
```

15. Sekarang kita definisikan metode apply_action(), yang akan mendapatkan aksi dari agent output dan merupah itu ke motion fisik robot. Agent output adalah antara 0 atau 1 yang menyatakan gerakan maju atau mundur menggunakan motor robot.

```
def apply_action(self, action):
    action = int(action[0])

    if action == 0:
        motorSpeed = 5.0
    else:
        motorSpeed = -5.0

    for i in range(len(self.wheels)):
        self.wheels[i].setPosition(float('inf'))
        self.wheels[i].setVelocity(motorSpeed)
```

16. Terakhir kita tambahkan implementasi dummy metode get_info() dan render(), karena pada cnoth kali ini metode tersebut tidak sebenarnya dipakai, tapi dibutuhkan oleh framework dan gym pada latar belakang.

```
def render(self, mode='human'):
    print("render() is not used")

def get_info(self):
    return None
```

RL Training Loop

1. Inisiasi supervisor object dan PPO agent, dilengkapi dengan ruang observasi dan aksi. Perhatikan bahwa kita mengekstrak angka 4 sebagai numberOfInput dan 2 sebagai numberOfActorOutputs dari gym space, karena algoritma implementasi mengharapkan integer untuk argumen yang diinisialisasi dari neural network input dan neuron output.

```
env = CartpoleRobot()
agent = PPOAgent(numberOfInputs=env.observation_space.shape[0],
numberOfActorOutputs=env.action_space.n)
```

2. Set solved ke false

```
solved = False
```

3. Sebelum kita mengatur RL Loop, kita definiskan penghitung episode dan limit untuk jumlah episode yang akan dijalankan.

```
episodeCount = 0
episodeLimit = 2000
```

4. Sekarang kita definiskan outerloop yang menjalankan jumlah dari episode yang telah didefinisikan sebelumnya, serta me reset world untuk memulai observasi. Kita juga me reset episode ke nilai 0

```
# Run outer loop until the episodes limit is reached or the task is solved
while not solved and episodeCount < episodeLimit:
    observation = env.reset()  # Reset robot and get starting observation
    env.episodeScore = 0
```

5. Kita mulai memanggil metode agent.work(), ang memiliki nilai 0 dari metode reset() untuk step pertama.

```
    for step in range(env.stepsPerEpisode):
      # In training mode the agent samples from the probability distribution, naturally implementing exploration
      selectedAction, actionProb = agent.work(observation, type_="selectAction")
```

6. Lalu buat Transisi, yang dinamai tuple berisikan name suggests, transisi diantara observasi sebelumnya ke newObservation.

```
      # Step the supervisor to get the current selectedAction's reward, the new observation and whether we reached
      # the done condition
      newObservation, reward, done, info = env.step([selectedAction])

      # Save the current state transition in agent's memory
      trans = Transition(observation, selectedAction, actionProb, reward, newObservation)
      agent.storeTransition(trans)
```

7. Lalu cek apakah episode bisa dijalankan, jika tidak maka kita tambahkan step reward ke episodeScore accumulator, simpan newObservation sebagai observation dan loop ke episode selanjutnya

```
        if done:
          # Save the episode's score
          env.episodeScoreList.append(env.episodeScore)
          agent.trainStep(batchSize=step)
          solved = env.solved()  # Check whether the task is solved
          break

      env.episodeScore += reward  # Accumulate episode reward
      observation = newObservation  # observation for next step is current step's newObservation
```

8. Tambahkan print pernyataan dan increment dari penghitung episode untuk menyelesaikan outer loop

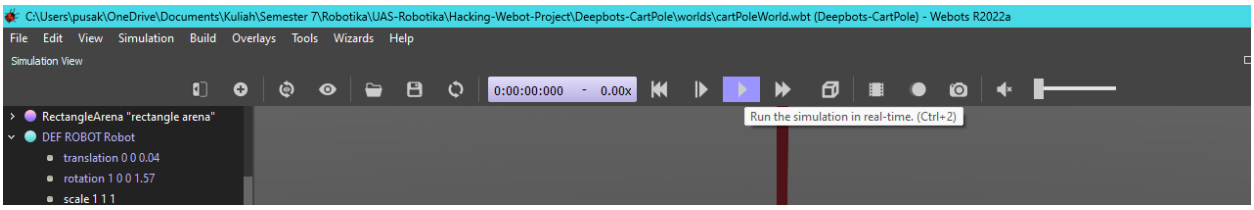
```
    print("Episode #", episodeCount, "score:", env.episodeScore)
    episodeCount += 1  # Increment episode counter
```

9. Akhirnya metode step() dipanggil, tetapi kali ini kita hanya menyimpan observasiya agar environment – agent loop tetap berjalan.

```
if not solved:
    print("Task is not solved, deploying agent for testing...")
elif solved:
    print("Task is solved, deploying agent for testing...")
observation = env.reset()
while True:
    selectedAction, actionProb = agent.work(observation, type_="selectActionMax")
    observation, _, _, _ = env.step([selectedAction])
```

Kesimpulan

Setelah proses pengkodean selesai, maka kita bisa menjalankan simulai dengan cara menekan tombol Run.



Kamu bisa menggerakan pole pada robot dengan Alt + klik kiri dan berusaha untuk menyeimbangkan pole pada robot.