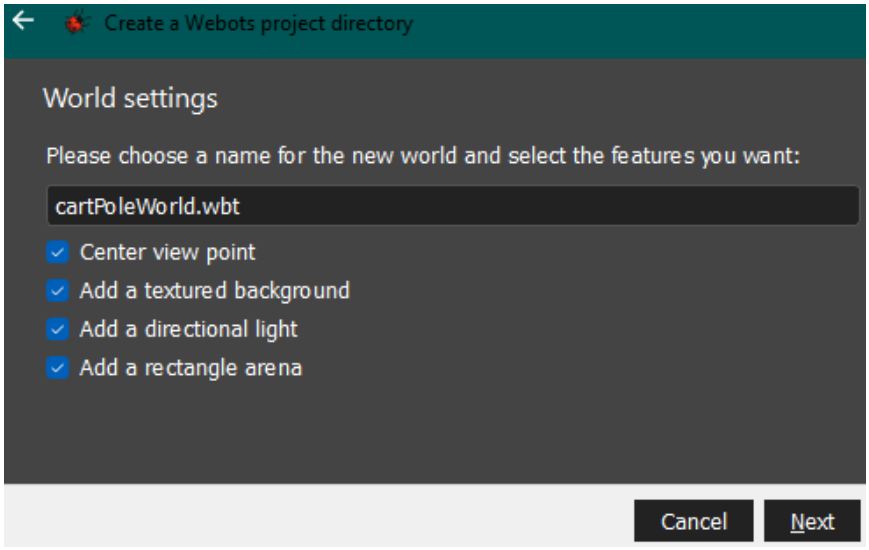
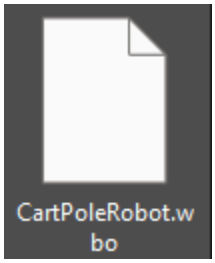
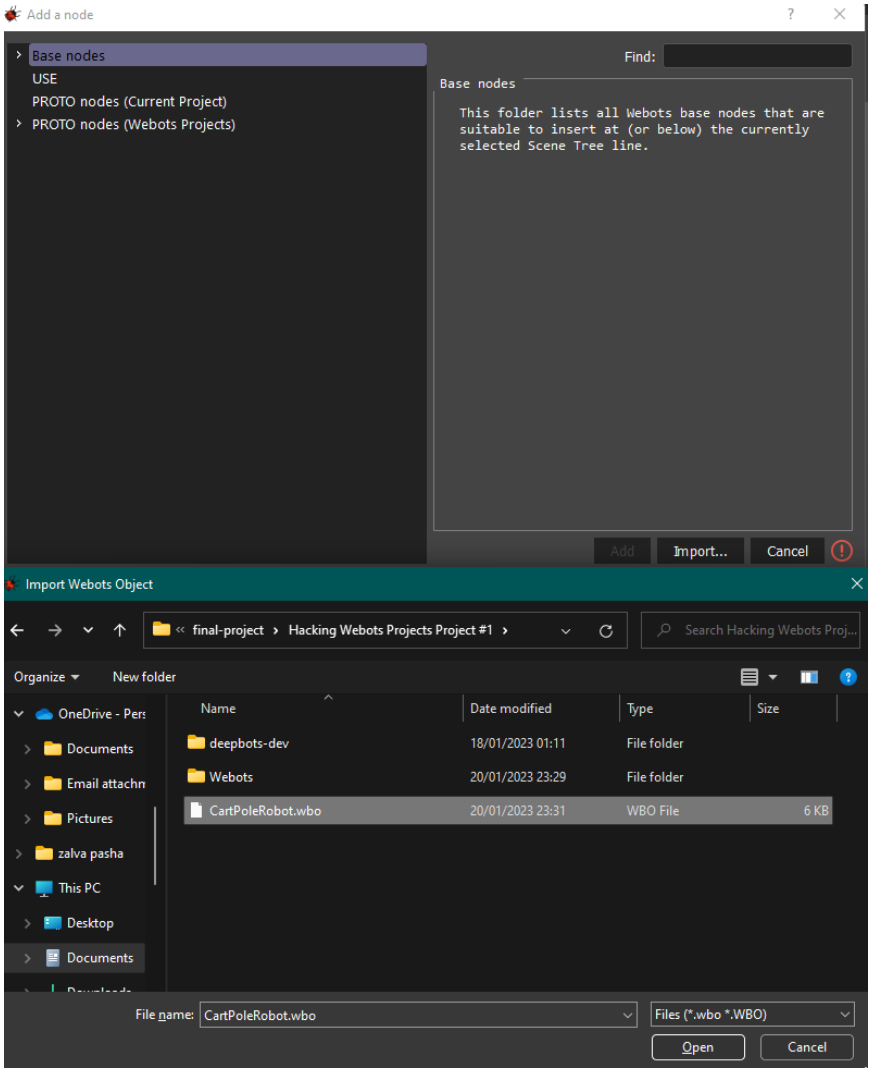
1. Melakukan instalasi PyTorch dan keperluan Deepbots.
```
pip3 install torch torchvision torchaudio
pip install -i https://test.pypi.org/simple/ deepbots
pip install gym
```
2. Membuat proyek baru dengan tambahan *rectangle arena*.



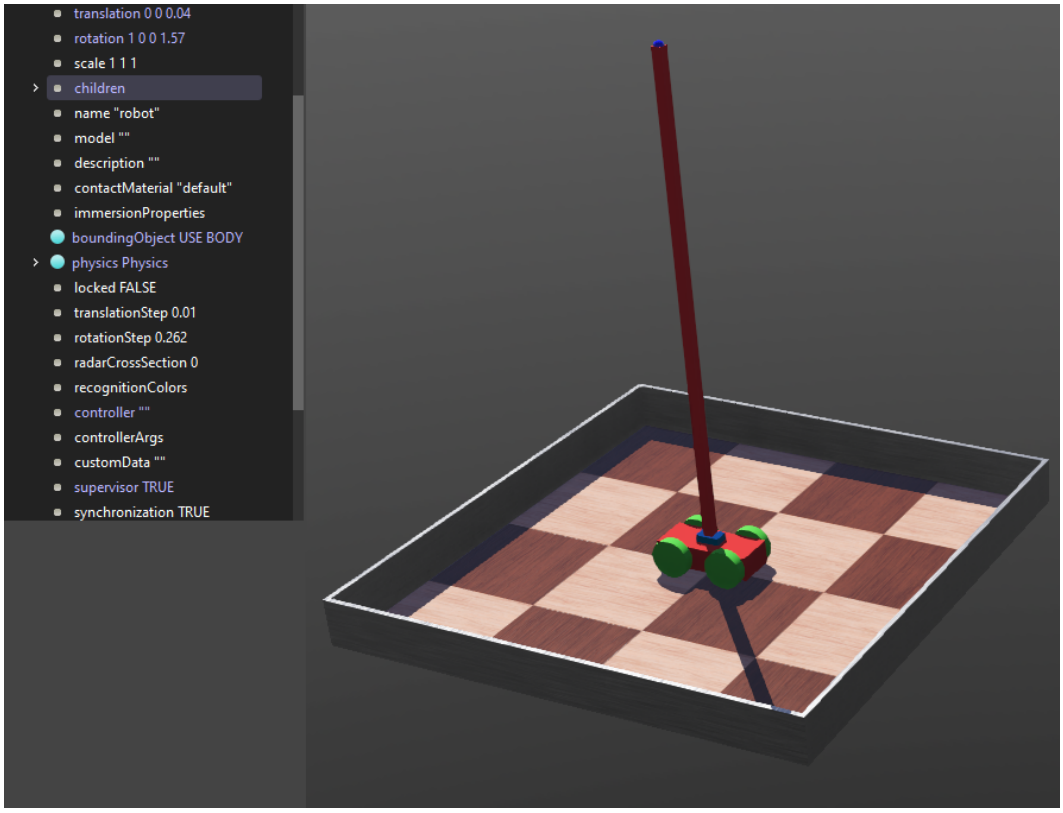
3. Download *CartPole robot definition* dengan menggunakan "*Save Link as…*".
akses link



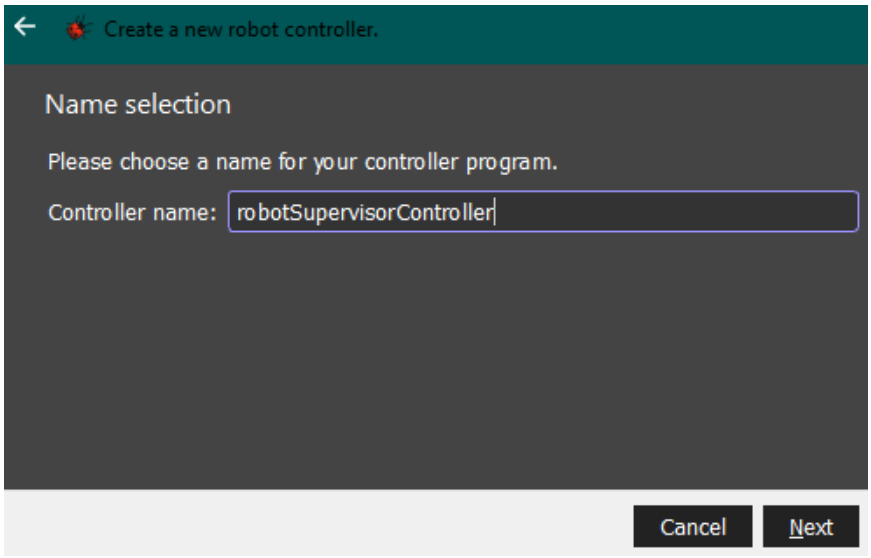
4. Melakukan *add nodes* lalu dilanjutkan dengan *import* file .wbo untuk mengambil objek.



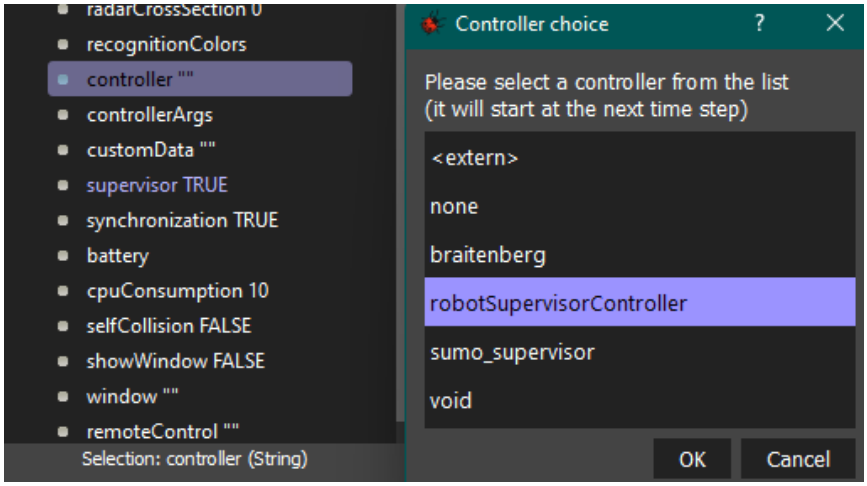
5. Webots akan menampilkan node “robot” dan juga model yang di import, mengubah supervisor menjadi “True”.



6. Menambahkan Controller baru yang menggunakan Python diberikan nama "robotSupervisorController".

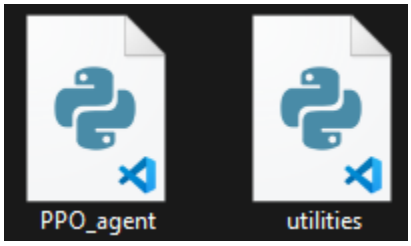


7. Skrip Python Controller baru harus dibuat dan dibuka di Webots text editor
8. Mengubah controller yang digunakan robot menjadi "robotSupervisorController".

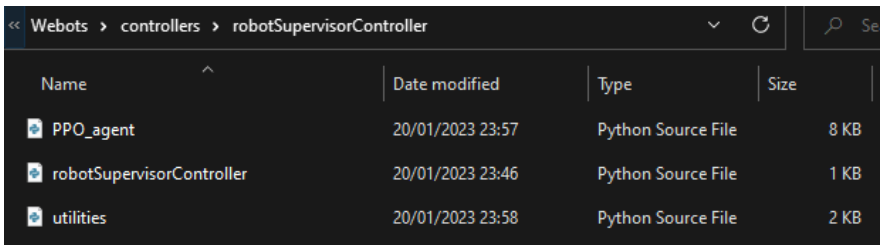


Script

1. klik disini untuk melakukan download PPO Agent dan klik disini untuk melakukan download *utilities script*.



2. Pemindahan *directory* kedua file tersebut ke folder robotSupervisorController pada project.



3. Mengubah isi dari file robotSupervisorController.py

robotSupervisorController.py

```python
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


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

        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)
        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

    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 x of endpoint
        endpointVelocity = normalizeToRange(self.poleEndpoint.getVelocity()[3], -1.5, 1.5, -1.0, 1.0, clip=True)

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

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

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()[0], 2)  # Position on x axis
    if abs(cartPosition) > 0.39:
        return True

    return False

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

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

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)

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

def get_info(self):
    return None
```

```python
env = CartpoleRobot()
agent = PPOAgent(numberOfInputs=env.observation_space.shape[0], numberOfActorOutputs=env.action_space.n)
solved = False
episodeCount = 0
episodeLimit = 2000
# 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
    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")
        # 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)
        if done:
            # Save the episode's score
            env.episodeScoreList.append(env.episodeScore)
            agent.trainStep(batchSize=step + 1)
            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
    print("Episode #", episodeCount, "score:", env.episodeScore)
    episodeCount += 1  # Increment episode counter

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])
```

Hasil Running Project