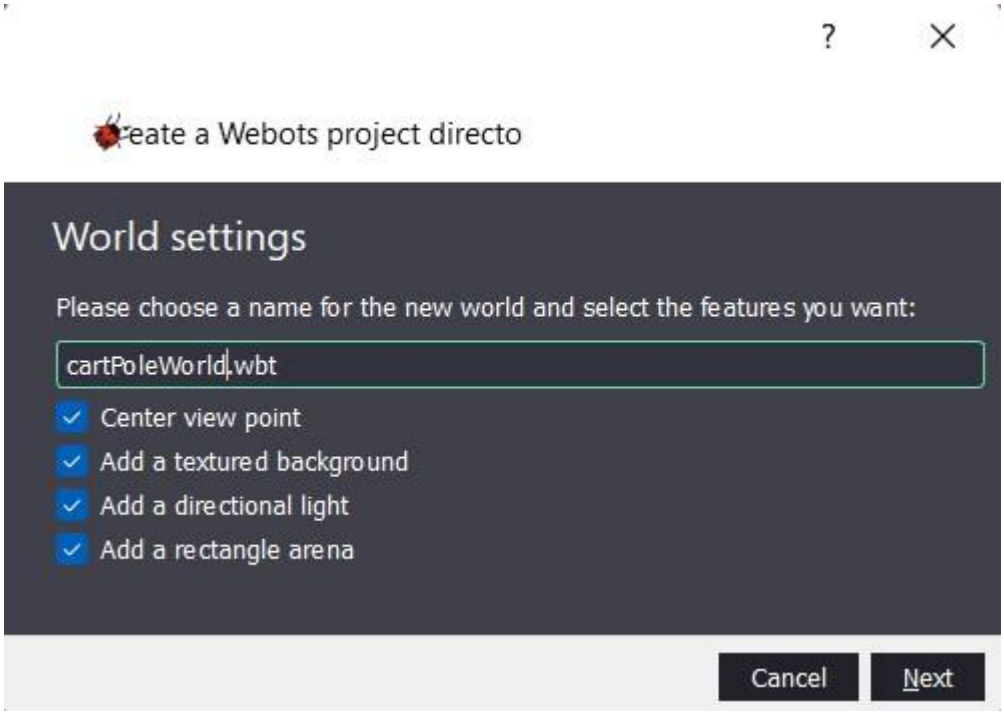
1. Install pytorch

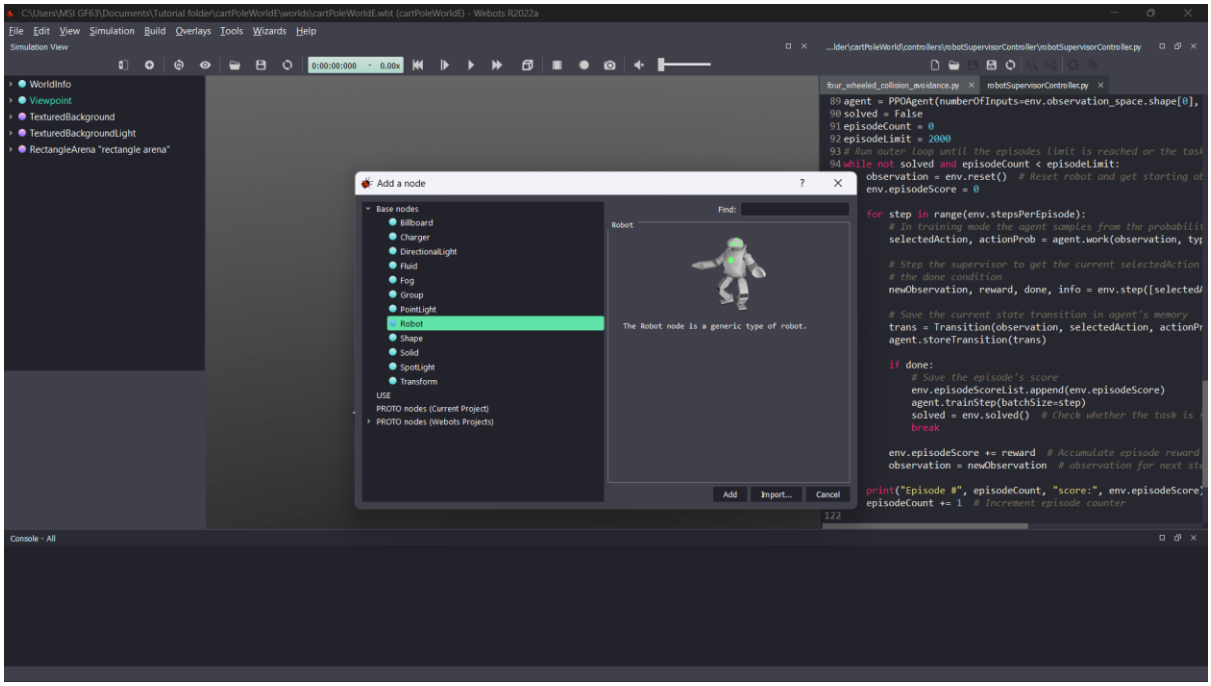
```
C:\Windows\System32>pip install gym
Collecting gym
  Downloading gym-0.26.2.tar.gz (721 kB)
     ---------------------------------------- 721.7/721.7 kB 5.0 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting gym-notices>=0.0.4
  Downloading gym_notices-0.0.8-py3-none-any.whl (3.0 kB)
Collecting cloudpickle>=1.2.0
  Downloading cloudpickle-2.2.1-py3-none-any.whl (25 kB)
Collecting importlib-metadata>=4.8.0
  Downloading importlib_metadata-6.0.0-py3-none-any.whl (21 kB)
Requirement already satisfied: numpy>=1.18.0 in c:\python38\lib\site-packages (from gym) (1.23.5)
Collecting zipp>=0.5
  Downloading zipp-3.11.0-py3-none-any.whl (6.6 kB)
Building wheels for collected packages: gym
  Building wheel for gym (pyproject.toml) ... done
  Created wheel for gym: filename=gym-0.26.2-py3-none-any.whl size=827646 sha256=2a70f331bd2bd31ac2f8392a2de12531a6ec47
76e0afd390047d90cb25de81c
  Stored in directory: c:\users\msi gf63\appdata\local\pip\cache\wheels\37\37\10\969be0b4bdeedbba8ecab056f1cbbf3a959cd7
ffd435c9a39
Successfully built gym
Installing collected packages: gym-notices, zipp, cloudpickle, importlib-metadata, gym
Successfully installed cloudpickle-2.2.1 gym-0.26.2 gym-notices-0.0.8 importlib-metadata-6.0.0 zipp-3.11.0

C:\Windows\System32>
```

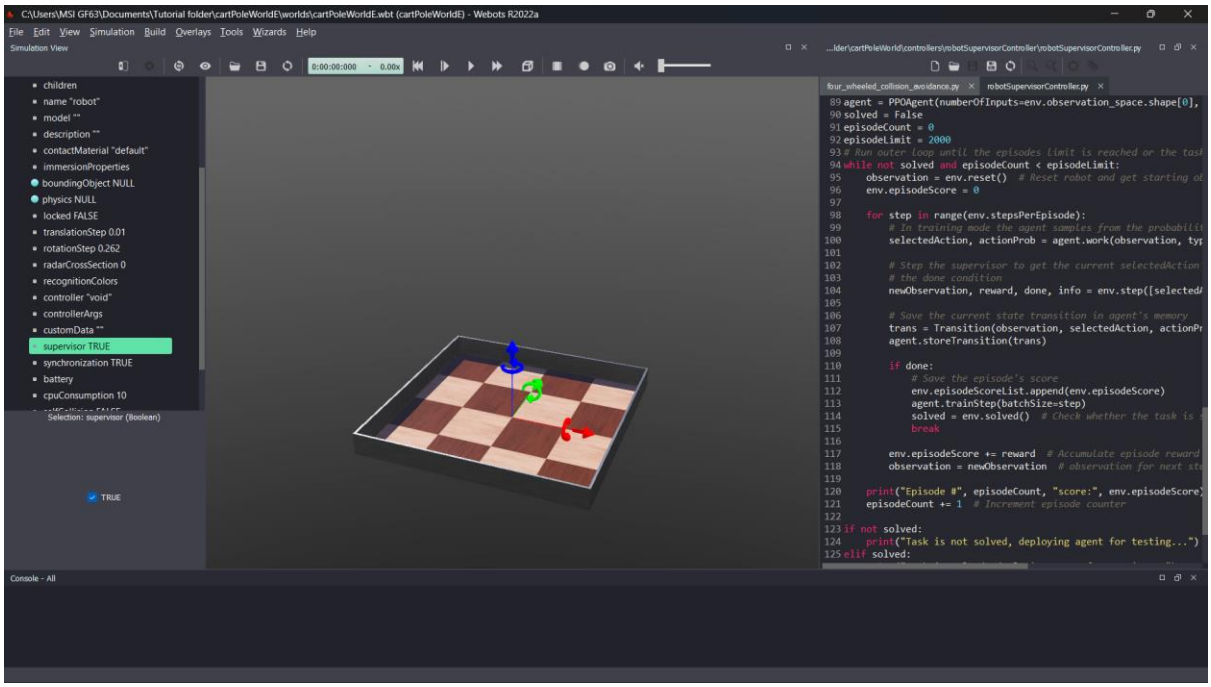
2. Membuat project directory baru



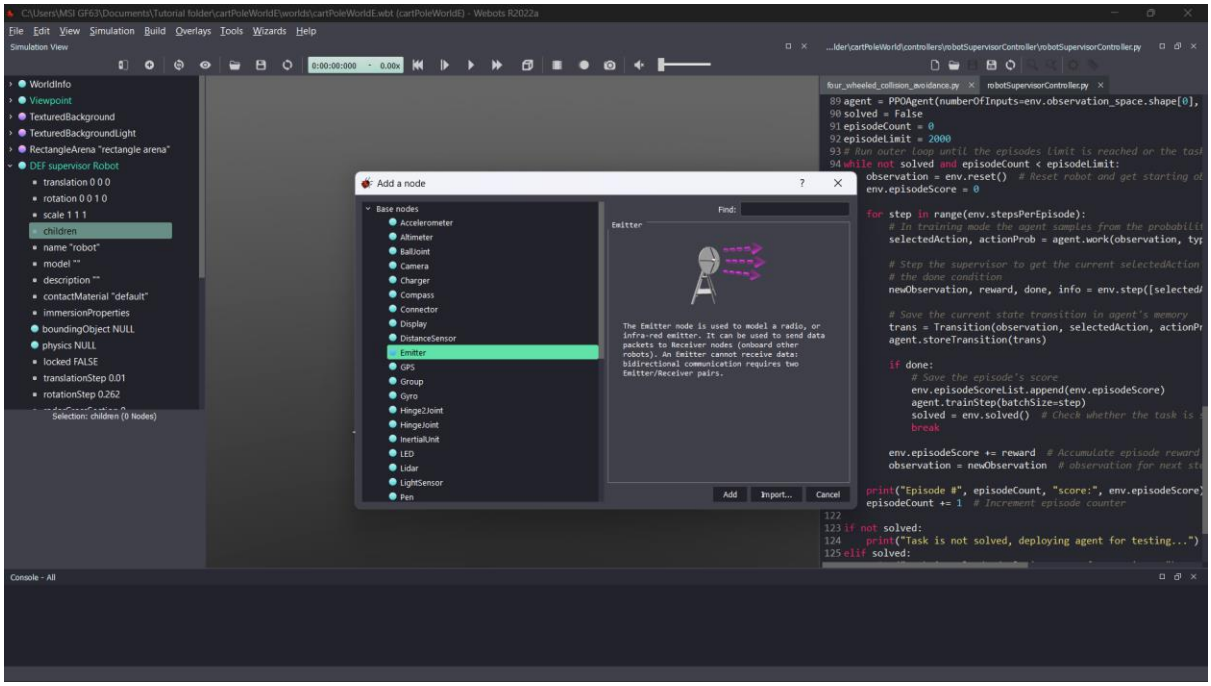
3. Masukkan nodes baru



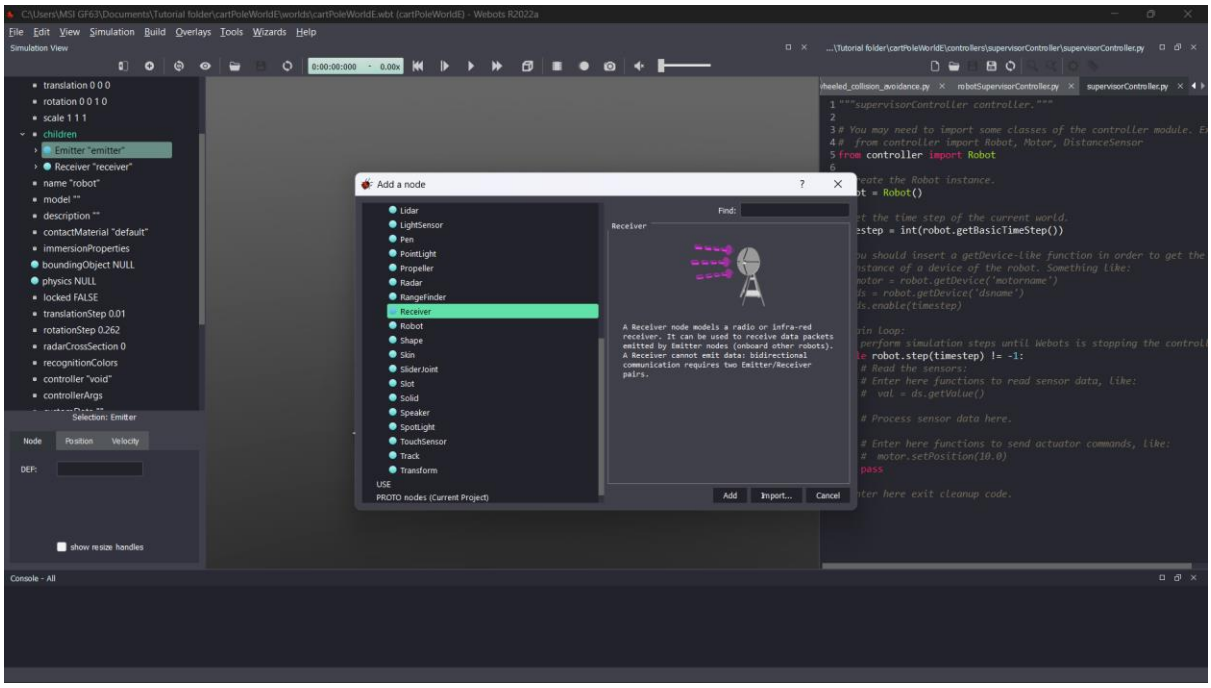
4. Set “supervisor” menjadi TRUE



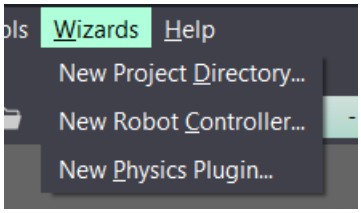
5. Lalu tambahkan nodes baru pada bagian children, tambahkan nodes emitter



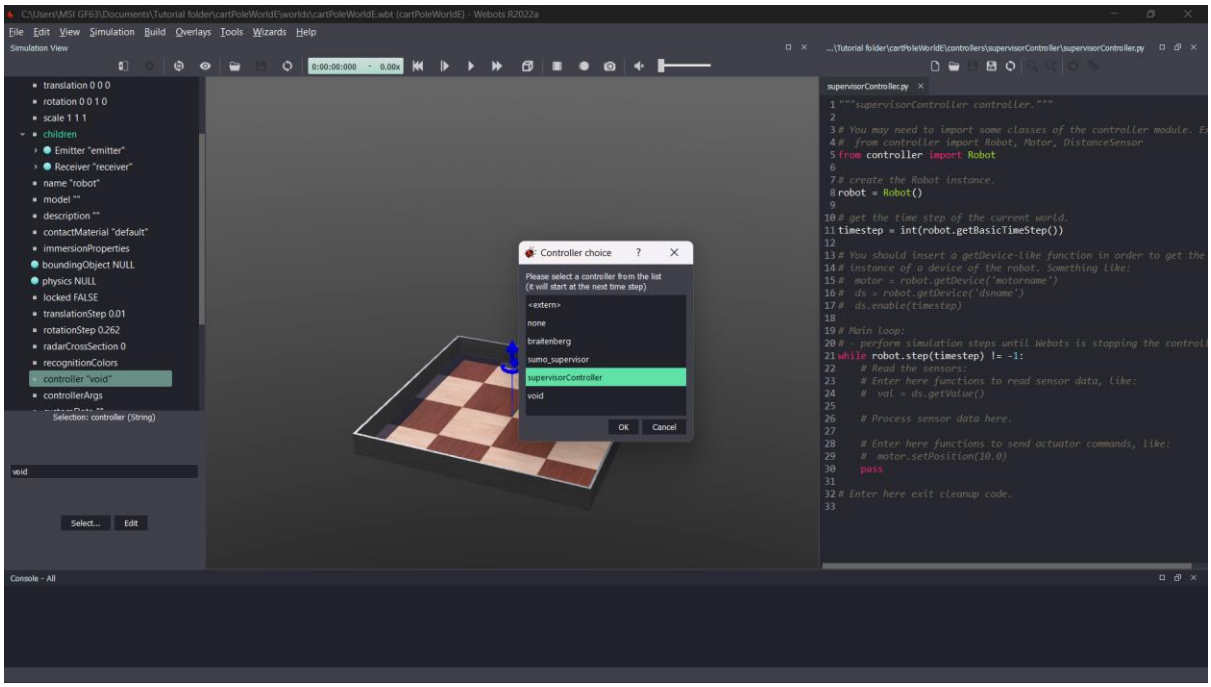
6. Tambahkan juga nodes receiver dibawahnya



7. Lalu buat controller baru



8. Lalu set controller, menggunakan select



9. Masukkan codingan

```python
import numpy as np
from deepbots.supervisor.controllers.supervisor_emitter_receiver import SupervisorCSV
from PPOAgent import PPOAgent, Transition
from utilities import normalizeToRange
from deepbots.robots.controllers.robot_emitter_receiver_csv import RobotEmitterReceiverCSV

class CartpoleRobot(RobotEmitterReceiverCSV):
    def __init__(self):
        super().__init__()
        self.positionSensor = self.robot.getPositionSensor("polePosSensor")
        self.positionSensor.enable(self.get_timestep())
        self.wheel1 = self.robot.getMotor('wheel1')  # Get the wheel handle
        self.wheel1.setPosition(float('inf'))  # Set starting position
        self.wheel1.setVelocity(0.0)  # Zero out starting velocity
        self.wheel2 = self.robot.getMotor('wheel2')
        self.wheel2.setPosition(float('inf'))
        self.wheel2.setVelocity(0.0)
        self.wheel3 = self.robot.getMotor('wheel3')
        self.wheel3.setPosition(float('inf'))
        self.wheel3.setVelocity(0.0)
        self.wheel4 = self.robot.getMotor('wheel4')
        self.wheel4.setPosition(float('inf'))
        self.wheel4.setVelocity(0.0)

    def create_message(self):
        # Read the sensor value, convert to string and save it in a list
        message = [str(self.positionSensor.getValue())]
        return message

    def use_message_data(self, message):
        action = int(message[0])  # Convert the string message into an action integer
```

```
        if action == 0:
            motorSpeed = 5.0
        elif action == 1:
            motorSpeed = -5.0
        else:
            motorSpeed = 0.0

        # Set the motors' velocities based on the action received
        self.wheel1.setVelocity(motorSpeed)
        self.wheel2.setVelocity(motorSpeed)
        self.wheel3.setVelocity(motorSpeed)
        self.wheel4.setVelocity(motorSpeed)

# Create the robot controller object and run it
robot_controller = CartpoleRobot()
robot_controller.run()  # Run method is implemented by the framework, just need to call it

class CartPoleSupervisor(SupervisorCSV):
    def __init__(self):
        super().__init__()
        self.observationSpace = 4  # The agent has 4 inputs
        self.actionSpace = 2  # The agent can perform 2 actions

        self.robot = None
        self.respawnRobot()
        self.poleEndpoint = self.supervisor.getFromDef("POLE_ENDPOINT")
        self.messageReceived = None  # Variable to save the messages received from the robot

        self.episodeCount = 0  # Episode counter
        self.episodeLimit = 10000  # Max number of episodes allowed
        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 respawnRobot(self):
        if self.robot is not None:
            # Despawn existing robot
            self.robot.remove()

        # Respawn robot in starting position and state
        rootNode = self.supervisor.getRoot()  # This gets the root of the scene tree
        childrenField = rootNode.getField('children')  # This gets a list of all the children, ie. objects of the scene
        childrenField.importMFNode(-2, "CartPoleRobot.wbo")  # Load robot from file and add to second-to-last position

        # Get the new robot and pole endpoint references
        self.robot = self.supervisor.getFromDef("ROBOT")
        self.poleEndpoint = self.supervisor.getFromDef("POLE_ENDPOINT")

    def get_observations(self):
        # Position on z axis, third (2) element of the getPosition vector
        cartPosition = normalizeToRange(self.robot.getPosition()[2], -0.4, 0.4, -1.0, 1.0)
        # Linear velocity on z axis
        cartVelocity = normalizeToRange(self.robot.getVelocity()[2], -0.2, 0.2, -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)

        # Update self.messageReceived received from robot, which contains pole angle
        self.messageReceived = self.handle_receiver()
        if self.messageReceived is not None:
            poleAngle = normalizeToRange(float(self.messageReceived[0]), -0.23, 0.23, -1.0, 1.0, clip=True)
        else:
            # Method is called before self.messageReceived is initialized
            poleAngle = 0.0

        return [cartPosition, cartVelocity, poleAngle, endpointVelocity]

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

```
    def is_done(self):
        if self.messageReceived is not None:
            poleAngle = round(float(self.messageReceived[0]), 2)
        else:
            # method is called before self.messageReceived is initialized
            poleAngle = 0.0
        if abs(poleAngle) > 0.261799388:  # more than 15 degrees off vertical
            return True

        if self.episodeScore > 195.0:
            return True

        cartPosition = round(self.robot.getPosition()[2], 2)  # Position on z 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 reset(self):
        self.respawnRobot()
        self.supervisor.simulationResetPhysics()  # Reset the simulation physics to start over
        self.messageReceived = None
        return [0.0 for _ in range(self.observationSpace)]
```

```
    def get_info(self):
        return None
supervisor = CartPoleSupervisor()
agent = PPOAgent(supervisor.observationSpace, supervisor.actionSpace)
solved = False
# Run outer loop until the episodes limit is reached or the task is solved
while not solved and supervisor.episodeCount < supervisor.episodeLimit:
    observation = supervisor.reset()  # Reset robot and get starting observation
    supervisor.episodeScore = 0

    for step in range(supervisor.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 = supervisor.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
            supervisor.episodeScoreList.append(supervisor.episodeScore)
            agent.trainStep(batchSize=step)
            solved = supervisor.solved()  # Check whether the task is solved
            break

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

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

```

```
167 if not solved:
168     print("Task is not solved, deploying agent for testing...")
169 elif solved:
170     print("Task is solved, deploying agent for testing...")
171 observation = supervisor.reset()
172 while True:
173     selectedAction, actionProb = agent.work(observation, type_="selectActionMax")
174     observation, _, _, _ = supervisor.step([selectedAction])
175
176
177
```

10. Berikut hasil running project diatas