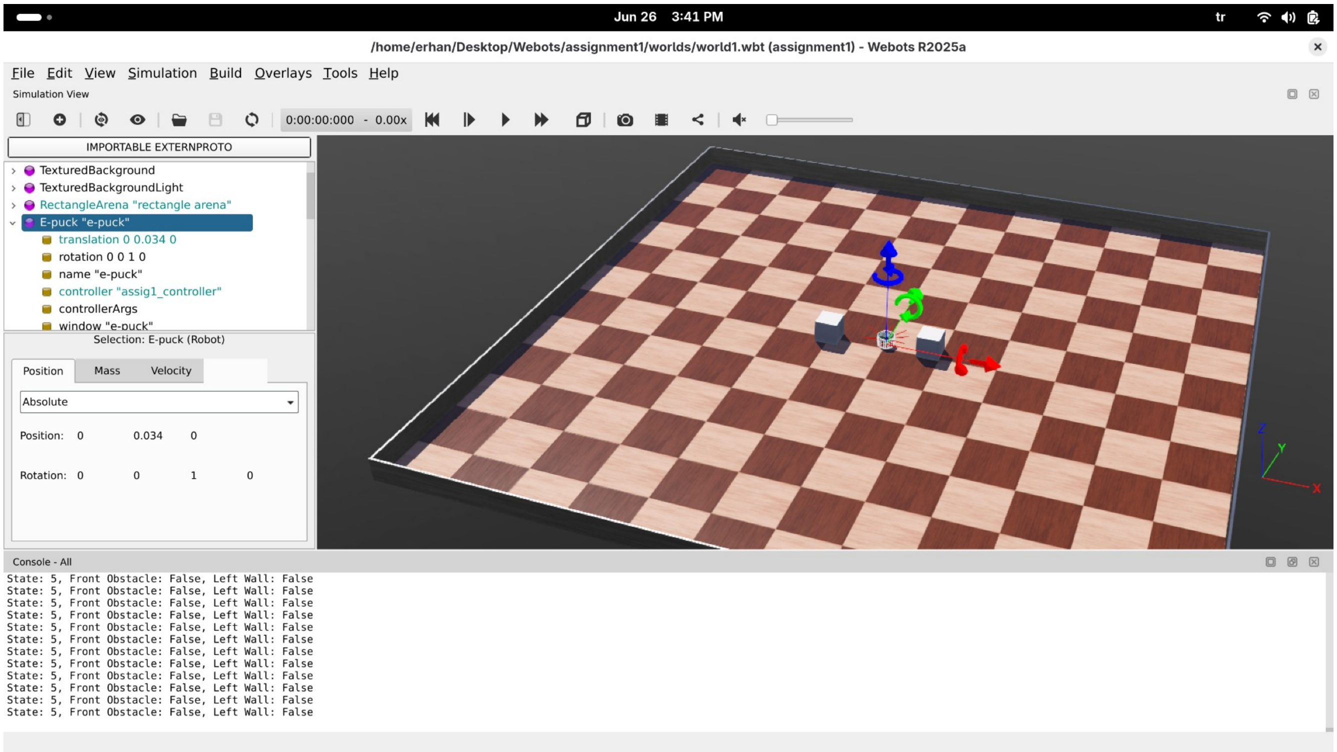
Peer-graded Assignment: Basic Reactive Behaviors

Screenshot of my project in WEBOTS:


Jun 26 3:41 PM
tr
/home/erhan/Desktop/Webots/assignment1/worlds/world1.wbt (assignment1) - Webots R2025a
File Edit View Simulation Build Overlays Tools Help
Simulation View
0:00:00:000 - 0.00x
IMPORTABLE EXTERNPROTO
Position
Mass
Velocity
Console - All
Z
Y
X

```
TexturedBackground
TexturedBackgroundLight
RectangleArena "rectangle arena"
E-puck "e-puck"
    translation 0 0.034 0
    rotation 0 0 1 0
    name "e-puck"
    controller "assig1_controller"
    controllerArgs
    window "e-puck"
```

Selection: E-puck (Robot)

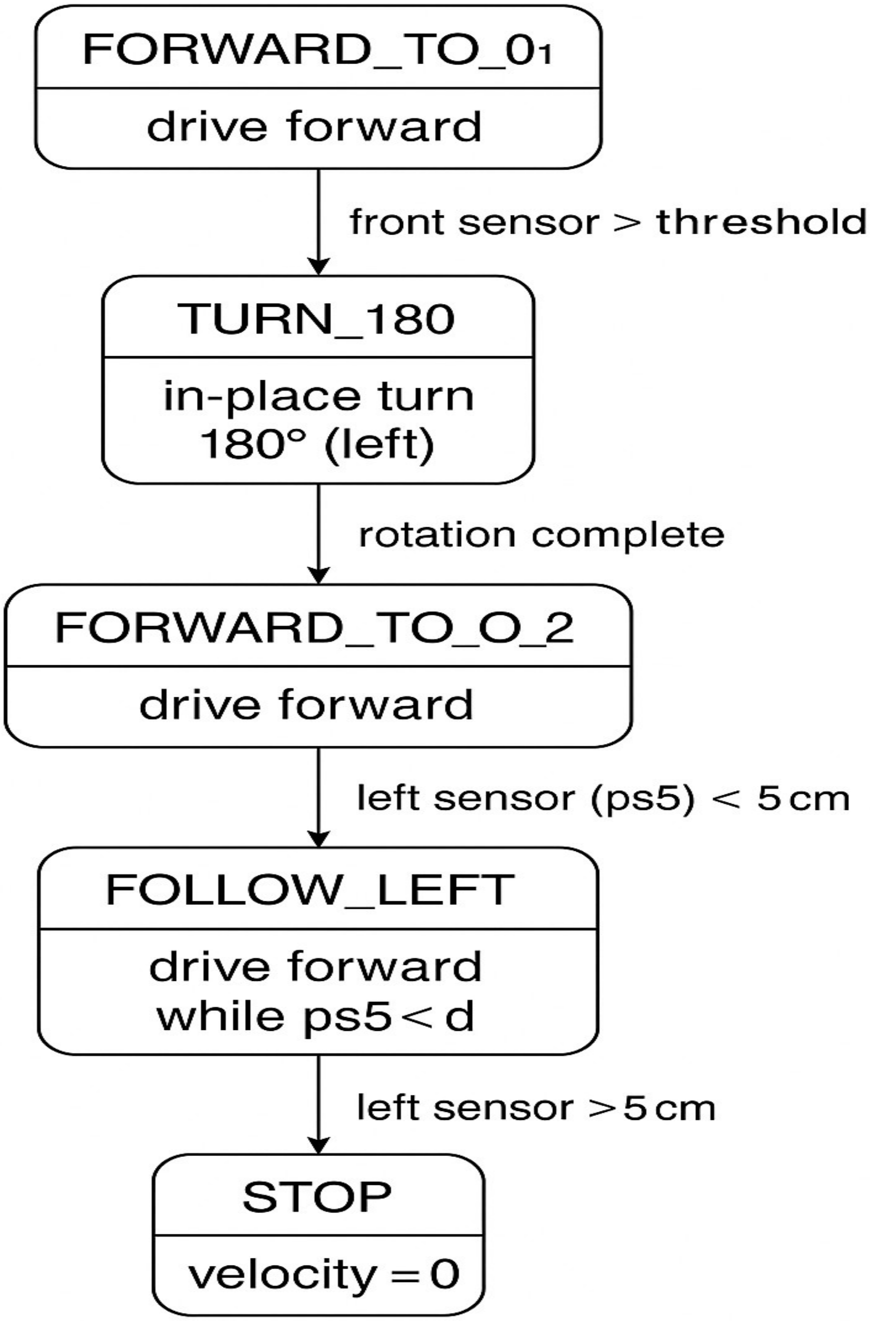
Absolute

Position: 0 0.034 0

Rotation: 0 0 1 0

```
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
State: 5, Front Obstacle: False, Left Wall: False
```

State-Machine Diagram



Final Result:
I completed the task given in the assignment correctly..! Here is my PYTHON Controller code:

```
from controller import Robot

robot = Robot()
timestep = int(robot.getBasicTimeStep())

# Constants
SPEED = 3.0  # Radians per second
THRESHOLD = 80  # Proximity value for ~5cm
WHEEL_RADIUS = 0.0205  # meters
AXLE_LENGTH = 0.053  # meters

# Calculate the required angular displacement for a 180-degree turn
# This is the angle each wheel needs to rotate, in radians.
# One wheel rotates +TURN_180_RAD, the other -TURN_180_RAD.
TURN_180_RAD = (3.1416 * AXLE_LENGTH) / (2 * WHEEL_RADIUS)
print(f"Required wheel rotation for 180-degree turn: {TURN_180_RAD:.3f} radians")

# Setup sensors
sensor_names = ['ps0', 'ps1', 'ps2', 'ps3', 'ps4', 'ps5', 'ps6', 'ps7']
sensors = [robot.getDevice(name) for name in sensor_names]
for s in sensors:
    s.enable(timestep)

# Motors and encoders
left_motor = robot.getDevice('left wheel motor')
right_motor = robot.getDevice('right wheel motor')
left_motor.setPosition(float('inf'))  # Set to infinite position control
right_motor.setPosition(float('inf')) # Set to infinite position control
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)

left_encoder = robot.getDevice('left wheel sensor')
right_encoder = robot.getDevice('right wheel sensor')
left_encoder.enable(timestep)
right_encoder.enable(timestep)

# State machine definitions
FORWARD_TO_O1 = 0
TURN_180 = 1
FORWARD_TO_O2 = 2
TURN_TO_LEFT_WALL = 3
FOLLOW_LEFT = 4
STOP = 5

state = FORWARD_TO_O1
```

```
# Variables to store initial encoder values for precise turns/movements
# These will be updated by reset_encoders()
initial_left_encoder_value = 0.0
initial_right_encoder_value = 0.0

def set_speed(l, r):
    left_motor.setVelocity(l)
    right_motor.setVelocity(r)

def reset_encoders():
    """Resets the reference point for encoder-based movements."""
    global initial_left_encoder_value, initial_right_encoder_value
    initial_left_encoder_value = left_encoder.getValue()
    initial_right_encoder_value = right_encoder.getValue()

def get_left_rotation():
    """Returns the absolute rotation of the left wheel since the last reset."""
    return abs(left_encoder.getValue() - initial_left_encoder_value)

def get_right_rotation():
    """Returns the absolute rotation of the right wheel since the last reset."""
    return abs(right_encoder.getValue() - initial_right_encoder_value)

# Initialize encoders at the start
reset_encoders()
print(f"Starting state: {state}")

while robot.step(timestep) != -1:
    # Read sensor values
    # ps0 and ps7 are front-left and front-right, ps1 and ps6 are side-front
    # For general "front" detection, it's good to check multiple front sensors.
    # ps0, ps7 are directly front
    # ps1, ps6 are front-side
    front_left_value = sensors[0].getValue()
    front_right_value = sensors[7].getValue()
    front_center_left_value = sensors[1].getValue()
    front_center_right_value = sensors[6].getValue()

    # We are checking ps5 for the left wall
    left_side_value = sensors[5].getValue()

    # Determine "front" obstacle detection
    # Consider multiple front sensors for robust detection
    is_front_obstacle = (front_left_value > THRESHOLD or
                         front_right_value > THRESHOLD or
                         front_center_left_value > THRESHOLD or
                         front_center_right_value > THRESHOLD)
    print(f"State: {state}, Front Obstacle: {is_front_obstacle}, Left Wall: {left_side_value > THRESHOLD}")

    if state == FORWARD_TO_O1:
        set_speed(SPEED, SPEED)
        if is_front_obstacle:
            # Stop before turning
            set_speed(0, 0)
            reset_encoders() # Reset encoders for the upcoming turn
            state = TURN_180
            print("Detected O1, transitioning to TURN_180")

    elif state == TURN_180:
        # One wheel forward, one backward for in-place turn
        set_speed(-SPEED, SPEED) # Left wheel backward, Right wheel forward (turns left)

        # Check if the desired rotation has been achieved by both wheels
        # For a point turn, both wheels should rotate by TURN_180_RAD in opposite directions
        if get_left_rotation() >= TURN_180_RAD and get_right_rotation() >= TURN_180_RAD:
            set_speed(0, 0) # Stop the turn
            reset_encoders() # Reset encoders for the next forward movement
            state = FORWARD_TO_O2
            print("180-degree turn complete, transitioning to FORWARD_TO_O2")

    elif state == FORWARD_TO_O2:
        set_speed(SPEED, SPEED)
        if is_front_obstacle:
            # Stop before the turn
            set_speed(0, 0)
            reset_encoders() # Reset encoders for the upcoming turn
            state = TURN_TO_LEFT_WALL
            print("Detected O2, transitioning to TURN_TO_LEFT_WALL")

    elif state == TURN_TO_LEFT_WALL:
        # Rotate clockwise until left sensor sees a wall
        set_speed(SPEED, -SPEED) # Left wheel forward, Right wheel backward (turns right/clockwise)
        if left_side_value > THRESHOLD:
            set_speed(0, 0) # Stop the turn
            # Optionally, you might want a small forward adjustment here if it overshoots
            reset_encoders() # Reset encoders for the next phase
            state = FOLLOW_LEFT
            print("Left wall detected, transitioning to FOLLOW_LEFT")

    elif state == FOLLOW_LEFT:
        # Simple wall following: if too far, turn slightly left; if too close, turn slightly right
        # For this simple setup, let's make it go straight if wall is present, else stop.
        # More advanced wall following would involve proportional control.
        if left_side_value > THRESHOLD:
            # Maintain forward speed
            set_speed(SPEED, SPEED)
        else:
            # Wall lost or end of path, stop
            set_speed(0, 0)
            state = STOP
            print("Lost left wall or reached end, transitioning to STOP")

    elif state == STOP:
        set_speed(0, 0) # Ensure motors are stopped
```