

**ROBOTICS PRACS**

**Practical 1**

**Aim:-** Write a program to create a robot (i) With gear (ii) Without gear and move it forward, left,right.

Description:

Description:

1. Gear(

Creates a gear instance with right motor plugged into port A, left motor plugged into port B.

2. backward(int duration)

Starts the backward movement for the given duration (in ms) and stops.

3. forward(int duration)

Starts the forward movement for the given duration (in ms) and stops.

4. left (int duration)

Starts to rotate left (center of rotation at middle of the wheel axes) for the given duration (in ms) and stops.

5. right(int duration) Starts to rotate right (center of rotation at middle of the wheel axes) for the given duration (in ms) and stops.

6. setSpeed(int speed) Sets the speed to the given value (arbitrary units).

7. stop()

Stops the movement.

8. NxtRobot()

Creates a turtle robot instance.

1. With gear

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class MoveWithGear {

MoveWithGear()

{

NxtRobot robot = new NxtRobot();

Gear g=new Gear();

robot.addPart(g);

g.forward(500);

g.setSpeed(30);

g.left(300);

g.forward(2000);

g.leftArc(500);

g.right(500);

g.forward(300);

robot.exit();

}

public static void main(String a[])

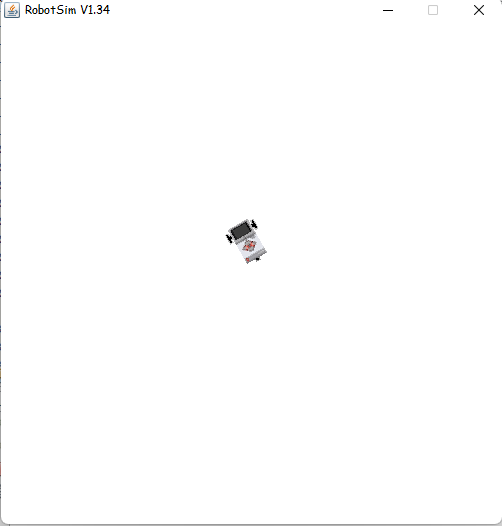
{

new MoveWithGear();

}

}

**Output:-**



1. Without gear

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class Movement {

Movement()

{

TurtleRobot t=new TurtleRobot();

t.forward(20);

t.left(45);

t.forward(20);

t.right(45);

t.forward(200);

t.right(90);

t.forward(200);

t.backward(20);

t.exit();

}

public static void main(String a[])

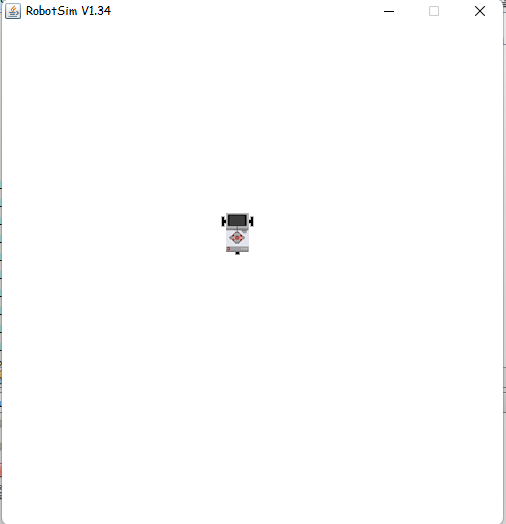
{

Movement m=new Movement();

}

}

**Output:-**



**Practical 2**

**Aim:-** Write a program to create a robot with a two motor and move it forward,left,right.

Description:

1. Motor(MotorPort port)

Creates a motor instance that is plugged into given port.

2. MotorPort A

A motor port for a motor connected to port A.

3. static MotorPort

B

A motor port for a motor connected to port B.

4. static Motor Port C

A motor port for a motor connected to port C.

5. delay(int duration) Suspends execution of the current thread for the given amount of time (unless the game grid window is disposed).

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class MoveWithMotors {

MoveWithMotors()

{

NxtRobot r = new NxtRobot();

Motor m1 = new Motor(MotorPort.A);

Motor m2 = new Motor(MotorPort.B);

r.addPart(m1);

r.addPart(m2);

m1.forward();

Tools.delay(1000);

m2.forward();

Tools.delay(1000);

m1.stop();

Tools.delay(1000);

m1.forward();

Tools.delay(1000);

m2.stop();

Tools.delay(1000);

m2.forward();

Tools.delay(1000);

r.exit();

}

public static void main(String a[])

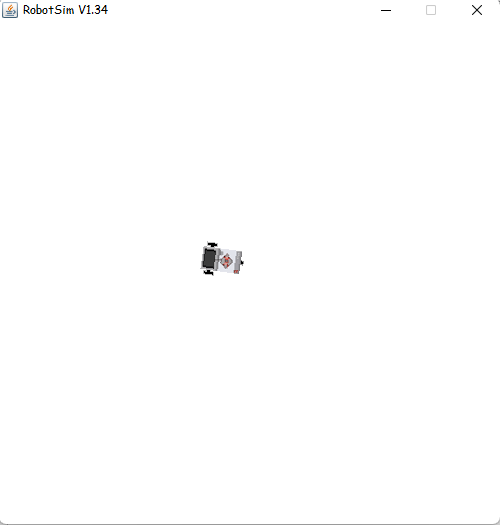
{

new MoveWithMotors();

}

}

**Output:-**



**Practical 3**

**Aim:-** Write a program to do a square using a while loop, doing steps with a for loop.

Description:

1. TurtleRobot() Creates a turtle robot instance.

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class WhileLoopSquare {

public WhileLoopSquare() {

TurtleRobot turtleRobot=new TurtleRobot();

while(true)

{

turtleRobot.forward(100);

turtleRobot.right(90);

//turtleRobot.exit();

}

}

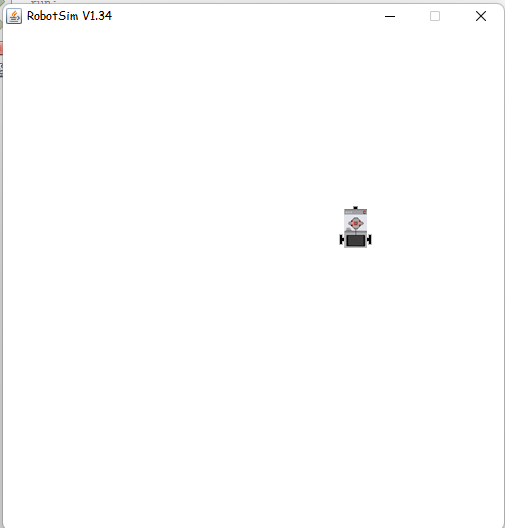
public static void main(String[] args){

new WhileLoopSquare();

}

}

**Output:-**



**Practical 4**

**Aim:-** Write a program to create a robot with light sensors to follow a line.

Description:

1. RobotContext()

Creates a RobotContext instance.

2. LegoRobot().

Creates a robot with its playground using defaults from RobotContext.

3. LightSensor(SensorPort port)

Creates a sensor instance pointing downwards connected to the given port.

4. LightSensor(SensorPort port, boolean upwards)

Creates a sensor instance connected to the given port.

5. addPart(Part part)

Assembles the given part into the robot.

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class RobotSensorLineFollower {

static {

RobotContext.setStartPosition(50, 470);

RobotContext.useBackground("sprites/road.gif");

}

public RobotSensorLineFollower() {

LegoRobot legoRobot = new LegoRobot();

Gear gearBox = new Gear();

LightSensor lightSensor = new LightSensor(SensorPort.S3);

legoRobot.addPart(gearBox);

legoRobot.addPart(lightSensor);

gearBox.forward();

gearBox.setSpeed(100);

while (true) {

int lightSensorValue = lightSensor.getValue();

if(lightSensorValue < 100)

gearBox.forward();

else if(lightSensorValue > 350 && lightSensorValue < 750)

gearBox.leftArc(0.05);

else if(lightSensorValue > 800)

gearBox.rightArc(0.05);

}

}

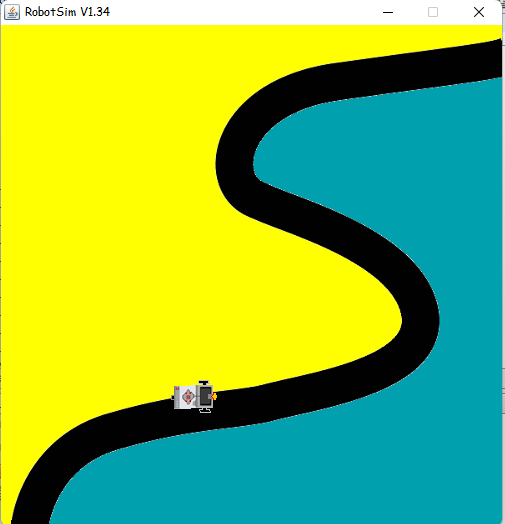
public static void main(String[] args) {

new RobotSensorLineFollower();

}

}

**Output:-**



**Practical 5**

**Aim:-** Write a program to create a robot that does a circle using 2 motors.

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class CircleWithMotors {

public CircleWithMotors(){

NxtRobot nxtRobot=new NxtRobot();

Motor motorLeft=new Motor(MotorPort.A);

Motor motorRight=new Motor(MotorPort.B);

nxtRobot.addPart(motorLeft);

nxtRobot.addPart(motorRight);

while(true){

motorLeft.forward();

motorRight.forward();

Tools.delay(300);

motorLeft.stop();

Tools.delay(300);

}

}

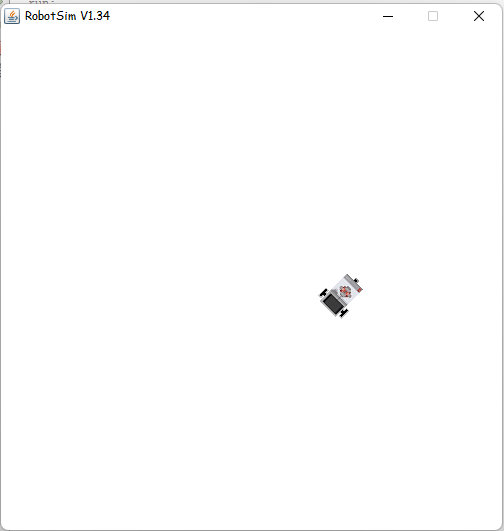
public static void main(String[] args){

new CircleWithMotors();

}

}

**Output:-**



**Practical 6**

**Aim:-** Write a program to create a path following robot.

Description:

1. left Arc(double radius)

Starts to move to the left on an are with given radius.

2. left Are(double radius, int duration) Starts to move left on an are with given radius for the given duration (in ms) and stops.

3. right(int duration)

Starts to rotate right (center of rotation at middle of the wheel axes) for the given duration (in ms) and stops.

4. rightArc(double radius)

+ Starts to move to the right on an are with given radius.

5. void rightArc(double radius, int duration) Starts to move right on an are with given radius for the given duration (in ms) and stops.

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.Gear;

import ch.aplu.robotsim.LegoRobot;

import ch.aplu.robotsim.LightSensor;

import ch.aplu.robotsim.RobotContext;

import ch.aplu.robotsim.SensorPort;

public class PathFollower {

static {

RobotContext.setStartPosition(105, 443);

RobotContext.useBackground("sprites/path.gif");

}

public PathFollower()

{

LegoRobot legoRobot = new LegoRobot();

Gear gearBox = new Gear();

LightSensor lightSensor1 = new LightSensor(SensorPort.S1);

LightSensor lightSensor2 = new LightSensor(SensorPort.S2);

legoRobot.addPart(gearBox);

legoRobot.addPart(lightSensor1);

legoRobot.addPart(lightSensor2);

gearBox.forward();

//gearBox.setSpeed(100);

while(true){

if(lightSensor1.getValue()>=10 && lightSensor2.getValue()>=10)

{

gearBox.forward();

}

else if(lightSensor1.getValue()<10 && lightSensor2.getValue()>10)

{

gearBox.leftArc(0.01);

}

else if(lightSensor1.getValue()>10 && lightSensor2.getValue()<10)

{

gearBox.rightArc(0.01);

}

}

}

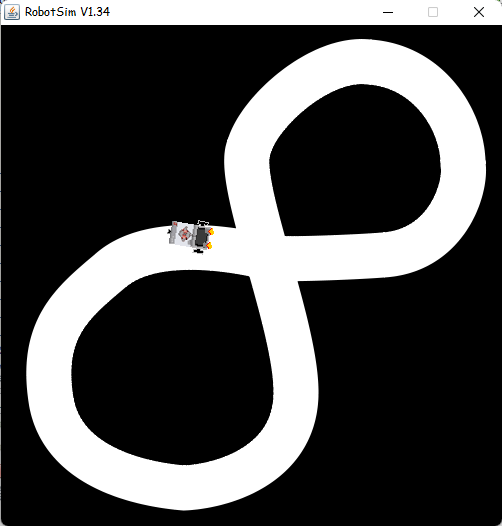
public static void main(String[] args) {

new PathFollower();

}

}

**Output:-**



**Practical 7**

**Aim:-** Write a program to resist obstacles.

Description:

1. setStartDirection(double direction) Sets the Nxt starting direction (zero to EAST).

2. static void setStartPosition(int x, int y) Sets the Nxt starting position (x-y-coordinates 0..500, origin at upper left). +

3. useBackground(java.lang.String filename) Use the given image as background (playground size 501 x 501).

4. TouchSensor(SensorPort port) Creates a sensor instance connected to the given port.

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

public class ResistObstacles {

public ResistObstacles()

{

LegoRobot r = new LegoRobot();

Gear g = new Gear();

TouchSensor t1 = new TouchSensor(SensorPort.S1);

TouchSensor t2 = new TouchSensor(SensorPort.S2);

r.addPart(g);

r.addPart(t1);

r.addPart(t2);

g.setSpeed(100);

g.right(200);

g.forward();

while(true)

{

Boolean b1 = t1.isPressed();

Boolean b2 = t2.isPressed();

if (b1 && b2) {

g.backward(100);

g.left(150);

g.forward();

}

else if (b1) {

g.backward(100);

g.left(100);

g.forward();

}

else if (b2) {

g.backward(100);

g.right(100);

g.forward();

}

}

}

public static void main(String args[])

{

new ResistObstacles();

}

static

{

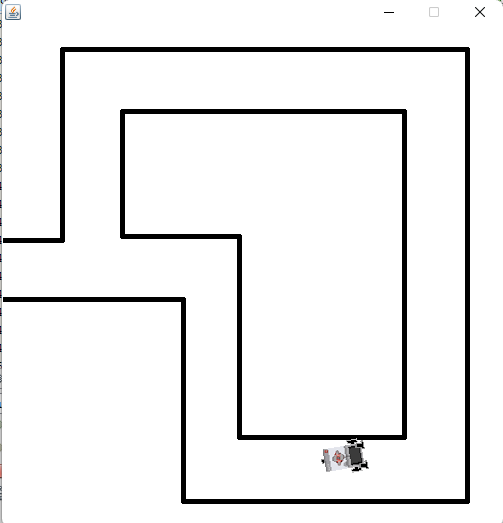
RobotContext.setStartPosition(50, 220);

RobotContext.useObstacle(RobotContext.channel);

}

}

**Output:-**



**Practical 8**

**Aim:-** Ultrasonic Sensor.

**Code:-**

package roboticsprac1;

import ch.aplu.robotsim.\*;

import java.awt.Color;

import java.awt.Point;

public class UltraSonic {

public UltraSonic()

{

LegoRobot robot = new LegoRobot();

Gear gear = new Gear();

robot.addPart(gear);

UltrasonicSensor us = new UltrasonicSensor(SensorPort.S1);

robot.addPart(us);

us.setBeamAreaColor(Color.green);

us.setProximityCircleColor(Color.lightGray);

double arc = 0.5;

gear.setSpeed(50);

gear.rightArc(arc);

boolean isRightArc = true;

int oldDistance = 0;

while (true)

{

Tools.delay(100);

int distance = us.getDistance();

if (distance == -1)

continue;

if (distance < oldDistance)

{

if (isRightArc)

{

gear.leftArc(arc);

isRightArc = false;

}

else

{

gear.rightArc(arc);

isRightArc = true;

}

}

oldDistance = distance;

}

}

public static void main(String[] args)

{

new UltraSonic ();

}

static

{

Point[] mesh\_bar =

{

new Point(10, 200), new Point(-10, 200),

new Point(-10, -200), new Point(10, -200)

};

RobotContext.useTarget("sprites/bar1.gif", mesh\_bar, 200, 250);

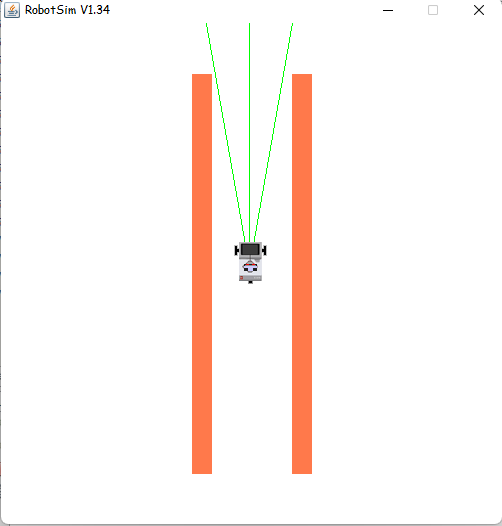
RobotContext.useTarget("sprites/bar1.gif", mesh\_bar, 300, 250);

RobotContext.setStartPosition(250, 460);

}

}

**Output:-**



**Practical 9**

**Aim:-** Drag and Bot Simulator Demo

**Description:**

Drag & Bot as a software platform for development of ROS-based industrial applications

Drag & Bot is the software platform for simple graphical setup and programming of robot systems. Drag & Bot enables manufacturing companies to flexibly and economically auto small let Drag & Bot pursues to be for ROS what Windows was for MS-DOS

Drag & Bot STUDIO: develop

Website running on browser

Intuitive

Graphical

Drag & Bot RUNTIME: produce

IPC-hased execution environment

ROS is running inside

Drag & Bot STUDIO: develop

•Website running on browser

Intuitive

Graphical

Drag & Bot RUNTIME produce

ROS namming inside in the cloud

Includes a robot simulator with different URDF models and inverse kinematics Includes a gripper and a machine tending interactive environment

Drag & Bot allows to create ROS-based industrial applications with an easy interface for the user.

Users are able to use, configure and reprogram industrial cells.

With Drag & Bot you can fast prototype new applications

New hardware or software modules can be integrated in a couple of days or even hours

Drag & Bot transforms industrial robots into flexible production tools.

Our software is the easiest way for flexible programming of robot systems.

It enables your workers to use robots and other machines independently for automation

Drag & Bot is an app for controlling industrial robots. The software is for use by production and industrial companies, enabling system programming

with little effort. A special feature of the app is that it lets staff program tasks independently, which allows them

to adapt robots to change production tasks and benefit from automation advantages. Staff should ideally be able to use available robots like they would use tools for performing specific tasks.

**Practical 10**

**Aim:-** Pick-up Object using Drag and Bot Simulator.

**Description:**

Why drag & bot for flexible robots in pick & place applications?

As different as manufacturing companies and their production processes are, the handling of parts of different materials, shapes and properties is present almost everywhere, In almost every operation, parts must be removed from bins, boxes, containers, workpiece carriers or grids and placed back in a different orientation or exact positioning

When done manually by a worker this can quickly become very monotonous and tedious. That is why more and more smaller companies are using robots for these tasks. Pick&Place industrial robots dis not get tired and make fewer mistakes. They are easy to acquire, quick to install. They often have a high cycle time, increasing throughput in production

The processes are often simple, a standard gripper on the arm is often sufficient special sensors and intermediate stations are not necessary. This saves investment costs and leads to a fast implementation of the projects. Robots in the pick&place area have one of the best return un-investment (ROI) calculations compared to other robot applications. The use of robots often pays for itself in less than a year.

With drag&hor, pick&place applications are easy to operate by workers, flexible to use and maximally cost efficient. More than every second robot sold worldwide is now used in the pick&place sector and the trend is rising

Loading/unloading machines

Machine loading with robots is one of the most common pick and place tasks. We work with many machine builders who want to offer a flexible automation solution to their machines. Typical extensions of these robot systems are cleaning quality checks and or marking of the workpieces. The video shows an automatic bending press machine from Placke, equipped with a Fanuc 1R Mate 200t) and a vacuum gripper.

**ROBOTICS ASSIGNMENT**

**Assignment 1:**

**Aim: Create a Robot to perform square and circular motion without using gear.**

Square Motion

Source Code:

package robotics;

import ch.aplu.robotsim.\*;

public class Assignment1\_SquareMovement {

Assignment1\_SquareMovement()

{

TurtleRobot t = new TurtleRobot();

for (int i = 0; i < 4; i++) {

t.forward(100);

t.left(90);

}

}

public static void main(String a[])

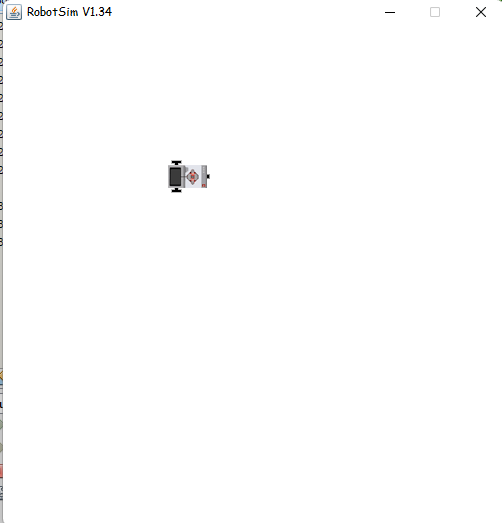
{

new Assignment1\_SquareMovement();

}

}

Output:



Circular Motion

Source Code:

package robotics;

import ch.aplu.robotsim.\*;

/\*\*

\*

\* @author admin

\*/

public class Assignment1\_CircularMotion {

Assignment1\_CircularMotion()

{

TurtleRobot t = new TurtleRobot();

while (true) {

t.forward(2);

t.left(2);

}

}

public static void main(String a[])

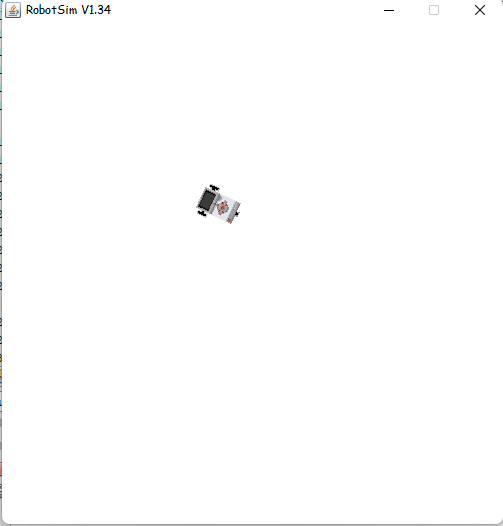
{

new Assignment1\_CircularMotion();

}

}

Output:



**Assignment 2:**

**Aim: Create a Robot to perform rectangular and circular motion using gears.**

Circular Motion using gears.

Source Code:

package robotics;

import ch.aplu.robotsim.\*;

public class Assignment2\_CircularMotionUsingGears {

Assignment2\_CircularMotionUsingGears()

{

NxtRobot nxtRobot = new NxtRobot();

Gear g = new Gear();

nxtRobot.addPart(g);

while (true) {

g.setSpeed(60);

g.forward(100);

g.left(80);

}

}

public static void main(String a[])

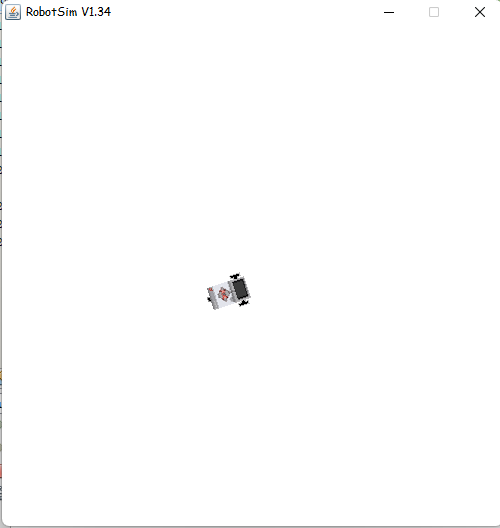
{

new Assignment2\_CircularMotionUsingGears();

}

}

Output:



Square motion using gears

Source Code:

package robotics;

import ch.aplu.robotsim.\*;

public class Assignment2\_SquareUsingGears {

Assignment2\_SquareUsingGears()

{

NxtRobot nxtRobot = new NxtRobot();

Gear g = new Gear();

nxtRobot.addPart(g);

for (int i = 0; i < 4; i++) {

g.setSpeed(50);

g.forward(2000);

g.left(550);

}

nxtRobot.exit();

}

public static void main(String a[])

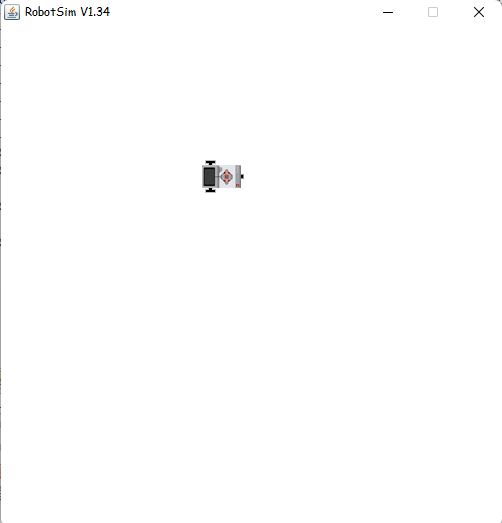
{

new Assignment2\_SquareUsingGears();

}

}

Output:



**Assignment 3:**

**Aim: Write a program to do a square using while or for loop, change the direction based on condition and controls motor movement using switch case.**

Source Code:

package robotics;

import ch.aplu.robotsim.\*;

import java.io.BufferedReader;

import java.io.\*;

public class Ass3\_MoveWithMotors\_WithSwitchCase {

public static void MoveWithMotor(String dir)

{

NxtRobot r = new NxtRobot();

Motor m1 = new Motor(MotorPort.A);

Motor m2 = new Motor(MotorPort.B);

r.addPart(m1);

r.addPart(m2);

m1.forward();

Tools.delay(1000);

m2.forward();

Tools.delay(1000);

switch(dir)

{

case "Left":

for (int i = 0; i < 4; i++) {

m1.stop();

Tools.delay(1000);

m1.forward();

Tools.delay(1000);

}

break;

case "Right":

for (int i = 0; i < 4; i++) {

m2.stop();

Tools.delay(1000);

m2.forward();

Tools.delay(1000);

}

break;

}

r.exit();

}

public static String getInput() throws IOException

{

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(isr);

System.out.println("Which direct you want to select (Left/Righ) ?");

String dir = br.readLine();

return dir;

}

public static void main(String a[]) throws IOException

{

String dir = getInput();

System.out.println("Direction : " + dir);

MoveWithMotor(dir);

}

}

Output:

