■クラス化

Turtle Robotを作ろう

- Turtle Robotとは？
  - 「カメ（Turtle）」は地面にペンを下ろして移動しながら図形を描く

- Turtle Robotびクラス定義
  - クラスはオブジェクトの仕様書

タートルロボットに関するデータ（属性）
・速度
・ライフポイント

タートルロボットが動くための計算（機能）
・前進
・後退
・・・

→

```
class TurtleRobot():
    速度
    ライフポイント          フィールド

    前進()
    後退（）                メソッド
    ・・・
```

→ 属性とそれに関わる機能をまとめ，TurtleRobotという名前のクラスを定義

Turtle Robot（Python）

p.106：turtle_robot.py

初期設定

```
class TurtleRobot:
    def __init__(self, v, c):
        motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
        self.velocity = v
        self.life_count = c

    def forward(self, duration_ms):
        motor_pair.move_tank(motor_pair.PAIR_1, self.velocity, self.velocity)
        time.sleep_ms(duration_ms)
        motor_pair.stop(motor_pair.PAIR_1)
        self.life_count -= 1

    def backward(self, duration_ms):
        motor_pair.move_tank(motor_pair.PAIR_1, -self.velocity, -self.velocity)
        time.sleep_ms(duration_ms)
        motor_pair.stop(motor_pair.PAIR_1)
        self.life_count -= 1

    def pause(self, duration_ms):
        motor_pair.stop(motor_pair.PAIR_1)
        time.sleep_ms(duration_ms)
```

前進メソッド

後退メソッド

一時停止メソッド

続く →

Turtle Robot（Python）

続き →

```
from hub import port
import motor_pair
import time
import runloop

kame = TurtleRobot(500, 10)

async def main():
    kame.forward(1000)
    print(kame.life_count)
    kame.pause(1000)

    kame.backward(1000)
    print(kame.life_count)
    kame.pause(1000)

runloop.run(main())
```

速度の値

ライフカウントの初期値

時間[ms]

コマンド＋Turtle Robot（Python）

続き →

```
from hub import port
import motor_pair
import time
import runloop

kame = TurtleRobot(500, 10)
command = ["f", "b", "f"]

async def main():
    for i in command:
        if i == "f":
            kame.forward(1000)
            kame.pause(1000)
            print(kame.life_count)
        elif i == “b":
            kame.backward(1000)
            kame.pause(1000)
            print(kame.life_count)

runloop.run(main())
```

コマンドリスト

辞書リスト＋Turtle Robot（Python）

- 辞書リストとは？
  - 複数の辞書を1つのリストにまとめたデータ構造

- コマンドの辞書リスト
  - コマンドと時間を指定

コマンド “cmd”	時間 [ms] “msec”
f	500
b	1000
f	800
f	500
b	1000

commandList[0]
commandList[1]
commandList[2]
commandList[3]
commandList[4]

```
commandList = [
{“cmd”:"f", "msec":500},
{“cmd”:"b", "msec":1000},
{“cmd”:"f", “msec”:800},
{“cmd”:"f", “msec”:500},
{“cmd”:"f", “msec”:500}
]
```

辞書リスト＋Turtle Robot（Python）

p.107：turtle_robot_list.py

```
kame = TurtleRobot(500, 10)
commandList = [
    {"cmd":"f", "msec":500},
    {"cmd":"b", "msec":1000},
    {"cmd":"f", "msec":500}
    ]

async def main():

    for command in commandList:
        action = command["cmd"]
        duration = command["msec"]

        if action == "f":
            kame.forward(duration)
            kame.pause(duration)
            print(kame.life_count)
        elif action == "b":
            kame.backward(duration)
            kame.pause(duration)
            print(kame.life_count)

runloop.run(main())
```

コマンド辞書リスト

Turtle Robotの課題

- 課題１
  - 左旋回のメソッドleft()と右旋回のメソッドright()をクラスに追加してみましょう

```
commandList = [
    {“cmd”:"f", "msec":1000},
    {“cmd”:"l", "msec":600},
    {“cmd”:"b", “msec”:500},
    {“cmd”:"r", “msec":700}
    ]
```

- 課題２
  - ライフポイントが0となったら終了するにように変更してみましょう
    ヒント：break

- 課題3
  - クラス定義を変更してTurtle Robotを拡張してみましょう

    例：ライフポイントが減ると速度が遅くなる

■並列タスク

シングルタスク

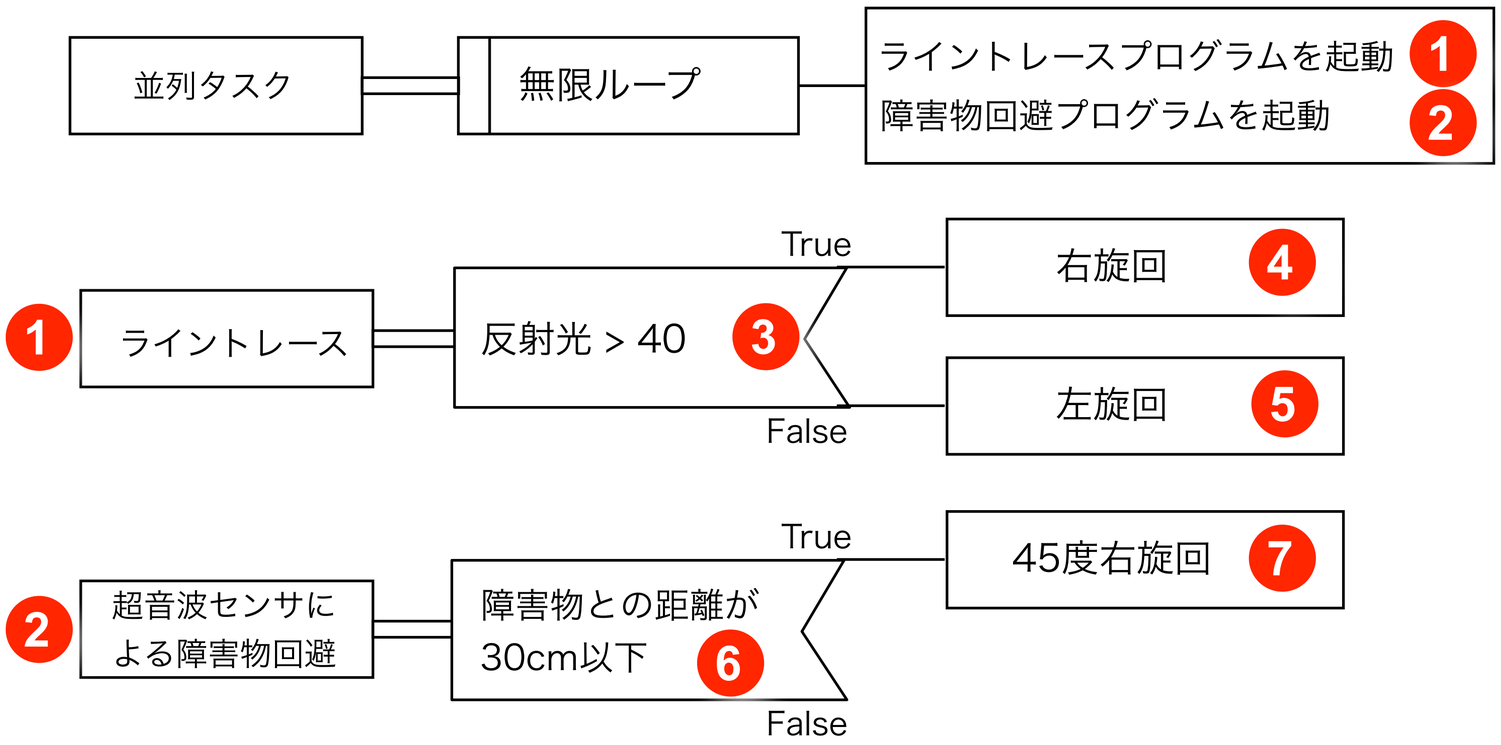
- シングルタスクと並列タスク

(a) シングルタスク　　(b) 並列タスク

→ライントレースと障害物回避を同時に行うには並列タスクを利用

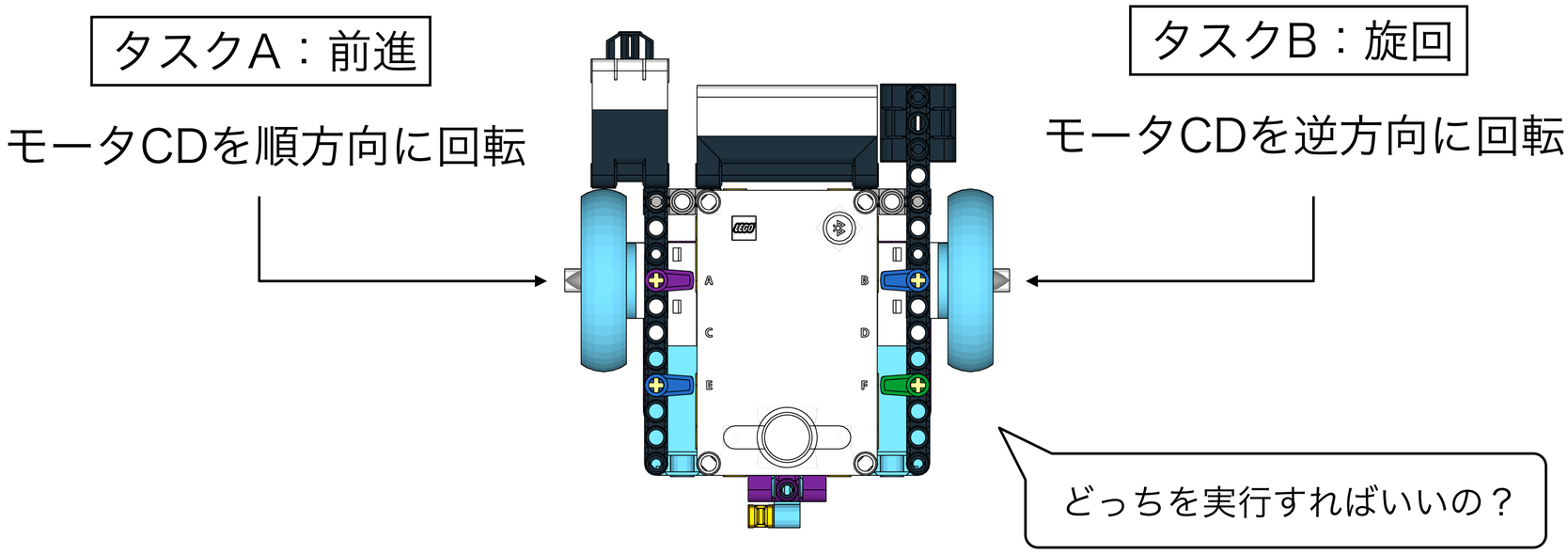
PADによる並列タスクの図示

- 並列タスク



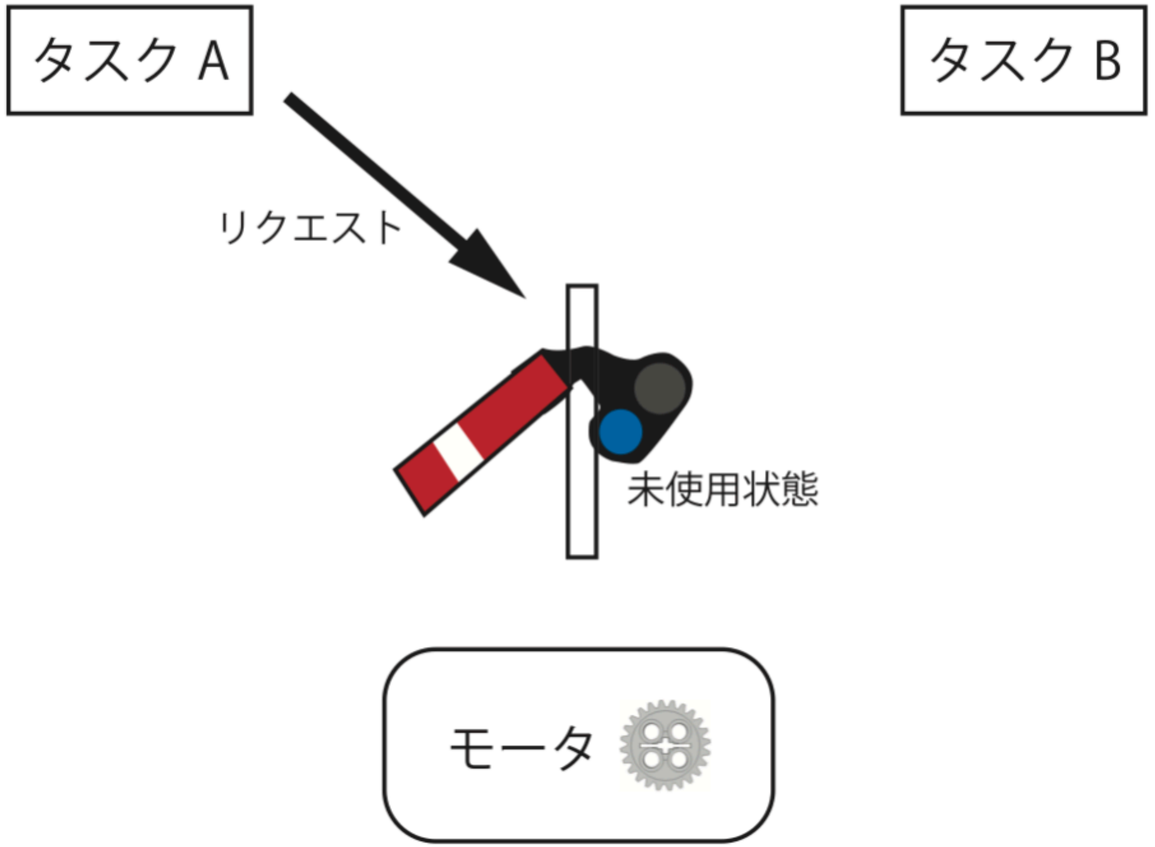
並列タスクにおける問題

- 命令の衝突（コンフリクト）
  - タスクAがタスクBの実行を妨害
  - 同時にモータを制御



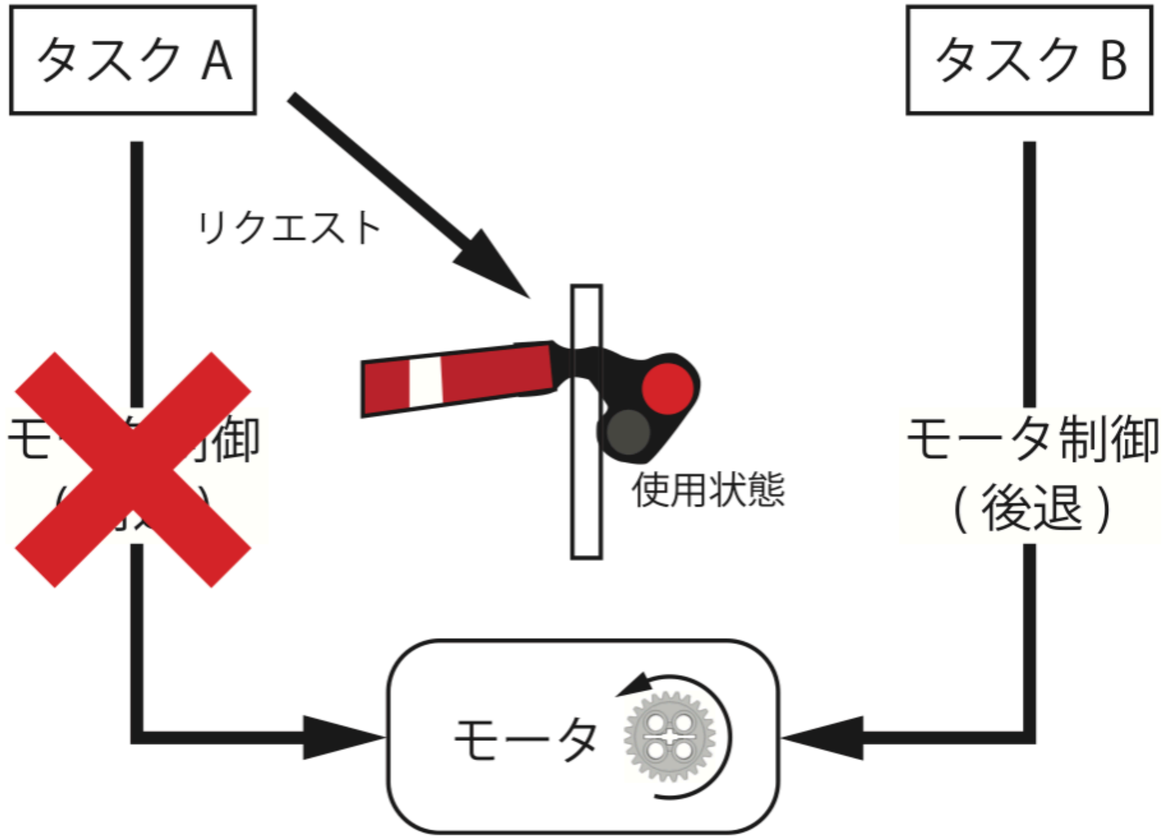
セマフォ（信号灯）

① タスクA使用時



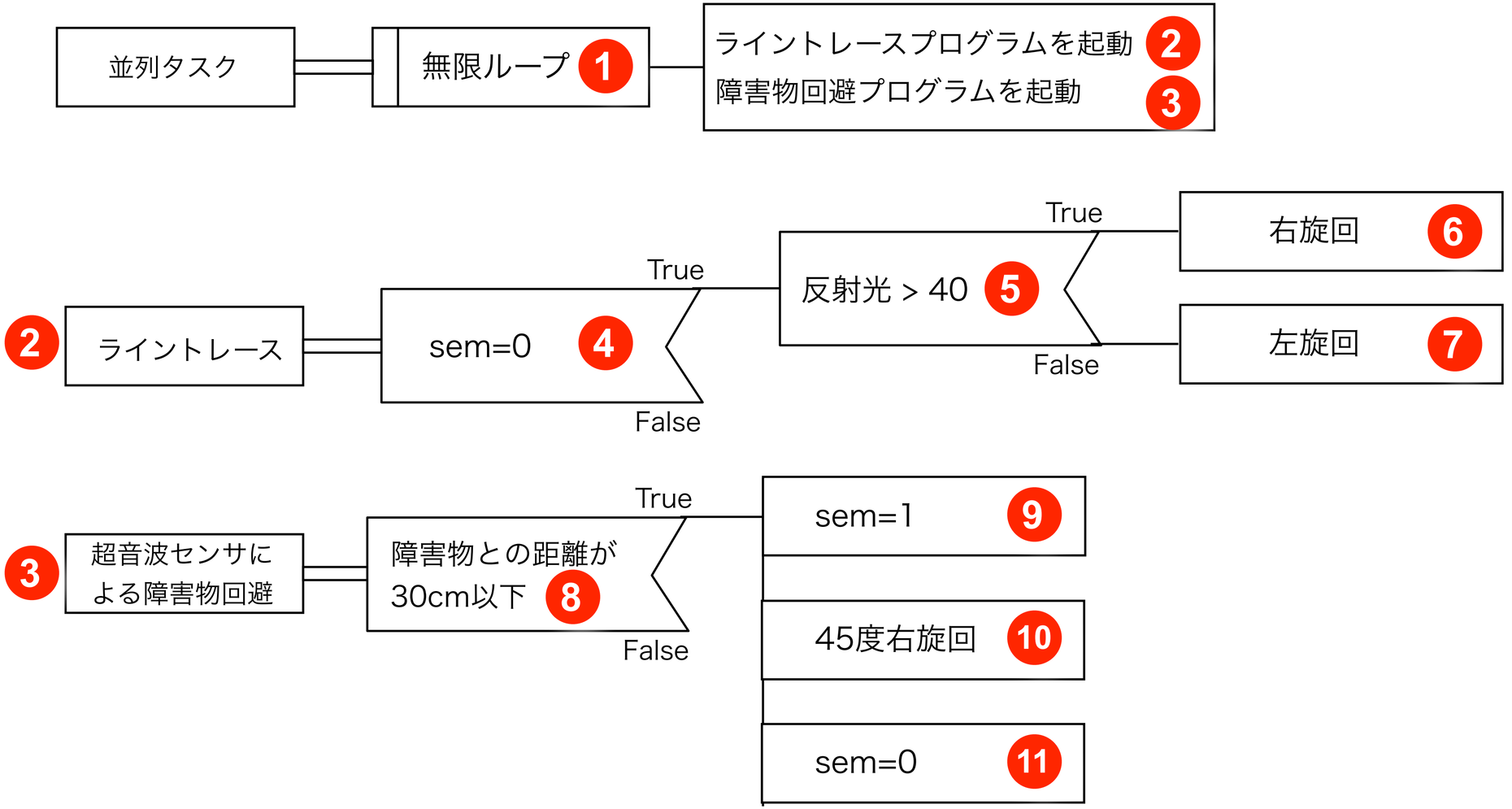
セマフォ（信号灯）

② タスクBによる制御



PADによる並列タスクの図示

- 並列タスク



並列タスク（Python）

- ライントレースと障害物回避を同時に実現

p.111：parallel_task.py

```
from hub import port
from hub import motion_sensor
import color_sensor
import distance_sensor
import motor_pair
import time
import asyncio

class MissionRobot:
    def __init__(self, v):
        motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
        motion_sensor.set_yaw_face(motion_sensor.FRONT)
        self.velocity = v
        self.sem = 0

    def forward_zero(self):
        motor_pair.move_tank(motor_pair.PAIR_1, self.velocity, self.velocity)

    def backward_zero(self):
        motor_pair.move_tank(motor_pair.PAIR_1, -self.velocity, -self.velocity)
```


続く →

並列タスク（Python）

続き →

```
    def forward_ms(self, duration_ms):
        motor_pair.move_tank(motor_pair.PAIR_1, self.velocity, self.velocity)
        time.sleep_ms(duration_ms)
        motor_pair.stop(motor_pair.PAIR_1)

    def backward_ms(self, duration_ms):
        motor_pair.move_tank(motor_pair.PAIR_1, -self.velocity, -self.velocity)
        time.sleep_ms(duration_ms)
        motor_pair.stop(motor_pair.PAIR_1)

    def turn_right_zero(self):
        motor_pair.move_tank(motor_pair.PAIR_1, self.velocity, 0)

    def turn_left_zero(self):
        motor_pair.move_tank(motor_pair.PAIR_1, 0, self.velocity)

    def turn_right_ms(self, motor_angle, duration_ms):
        motor_pair.move_tank_for_degrees(motor_pair.PAIR_1,motor_angle, self.velocity,\
            -self.velocity)
        time.sleep_ms(duration_ms)
        motor_pair.stop(motor_pair.PAIR_1)
```


続く →

並列タスク（Python）

続き →

```
    def turn_left_ms(self, motor_angle, duration_ms):
        motor_pair.move_tank_for_degrees(motor_pair.PAIR_1,motor_angle, \
            -self.velocity, self.velocity)
        time.sleep_ms(duration_ms)
        motor_pair.stop(motor_pair.PAIR_1)

    def turn_right_deg(self, turn_angle):
        motion_sensor.reset_yaw(0)
        while motion_sensor.tilt_angles()[0] > -turn_angle:
            motor_pair.move_tank(motor_pair.PAIR_1, self.velocity, -self.velocity)
        motor_pair.stop(motor_pair.PAIR_1)

    def turn_left_deg(self, turn_angle):
        motion_sensor.reset_yaw(0)
        while motion_sensor.tilt_angles()[0] > turn_angle:
            motor_pair.move_tank(motor_pair.PAIR_1, -self.velocity, self.velocity)
        motor_pair.stop(motor_pair.PAIR_1)

    def pause(self, duration_ms):
        motor_pair.stop(motor_pair.PAIR_1)
        time.sleep_ms(duration_ms)
```


続く →

並列タスク（Python）

続き →

```
2 async def line_trace(self):
    4 if robot.sem == 0:
        5 if color_sensor.reflection(port.B) > 40:
            6 robot.turn_right_zero()
          else:
            7 robot.turn_left_zero()


3 async def avoidance(self):
    8 if distance_sensor.distance(port.E)<300 and \
              distance_sensor.distance(port.E) != -1:
        9 robot.sem = 1
        10 robot.turn_right_deg(450)
          robot.pause(500)
        11 robot.sem = 0

robot = MissionRobot(300)

1 async def main():
    robot.sem=0
    while True:
  2 3   await asyncio.gather(robot.line_trace(),robot.avoidance())

asyncio.run(main())
```

演習課題

課題１：クラス定義を変更してTurtle Robotを拡張してみよう（前回の課題３の続き）。

（1）Turtle Robotをどのように拡張したかを文章で説明して下さい。
（2）上記に合わせて、どのようにclass定義を変更したのか、具体的に説明して下さい。
（3）プログラム全体のPADを作成して下さい。

課題２：ライントレース、障害物回避、ライトマトリクス表示の並列タスクを実現しましょう。
　プログラムとその解説、PAD、工夫した点をGoogleドキュメントにまとめて提出して下さい。