TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Programmeerimise süvendatud algkursus ITI0140

2015

Teema

Objektorienteeritud programmeerimine (OOP)

- Objektid – atribuudid ja meetodid
- Klassid
- Konstruktorid
- Pärimine ja polümorfism

OOP

Python on oma olemuselt objektorienteeritud programmeerimiskeel (OOP).

Objektorienteeritud programmeerimine (OOP) on programmeerimise paradigma, mis kasutab "objekte".

Objekt

Objekt on andmestruktuur, mis koosneb meetoditest ja andmetest (atribuutidest).

objekti **funktsioonid**

Objekti atribuudid

len(*object*) => objekti pikkus (atribuut)

string e. sõne

```
len(string): len("Tere") => 4

len(list): len(['a', 'b']) => 2
```

list e. järjend

Objekti meetodid

dict e. sõnastik

```
object.method(...)
```

```
{'a': 1, 'b': 2}.items() => [('b', 2),
('a', 1)]
```

objekti meetod

```
"Tere".upper() => TERE
```

Klass

Klass on **šabloon**, mis kirjeldab millegi olemust.
Objekt on klassi **konkreetne eksemplar** (*instance*).

`"Tere"` = sõne eksemplar
`type("Tere")` => <class 'str'>

`['a', 'b']` = järjendi eksemplar
`type(['a', 'b'])` => <class 'list'>

Klass näide

Konstruktor – algväärtustab klassi muutujad

```
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def print_point(self):
        print("(%s, %s)" % (self.x, self.y))
```

Pythonis esimene argument alati "self"

Klass näide

Siin anname kaasa ainult "x" ja "y" – "self" argumenti ei ole

```
p1 = Point2D(1, 2)
p2 = Point2D(0, 7)
p1.print_point()
p2.print_point()
```

```
(1, 2)
(0, 7)
```

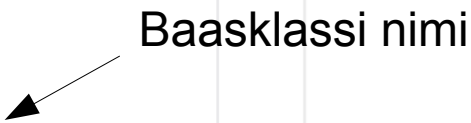
Pärimine

Baasklassi nimi

```
class Point3D(Point2D):
    def __init__(self, x, y, z):
        pass

p3 = Point3D(1, 2, 3)
p3.print_point()
```

Pärimine




```
class Point3D(Point2D):
    def __init__(self, x, y, z):
        pass

p3 = Point3D(1, 2, 3)
p3.print_point()
```

```
Traceback (most recent call last):
  File "x.py", line 14, in <module>
    p3.print_point()
  File "x.py", line 7, in print_point
    print("(%s, %s)" % (self.x, self.y))
AttributeError: 'Point3D' object has no
attribute 'x'
```

Pärimine

Baasklassi nimi

```
class Point3D(Point2D):
    def __init__(self, x, y, z):
        Point2D.__init__(self, x, y)

p3 = Point3D(1, 2, 3)
p3.print_point()
```

Kutsume välja baasklassi konstruktori

```
(1, 2)
```

Aga z-koordinaati pole

Pärimine

```
class Point3D(Point2D):
    def __init__(self, x, y, z):
        Point2D.__init__(self, x, y)
        self.z = z
    def print_point(self):
        print("(%s, %s, %s)" % (self.x, self.y, self.z))

p3 = Point3D(1, 2, 3)
p3.print_point()
```

Polümorfism

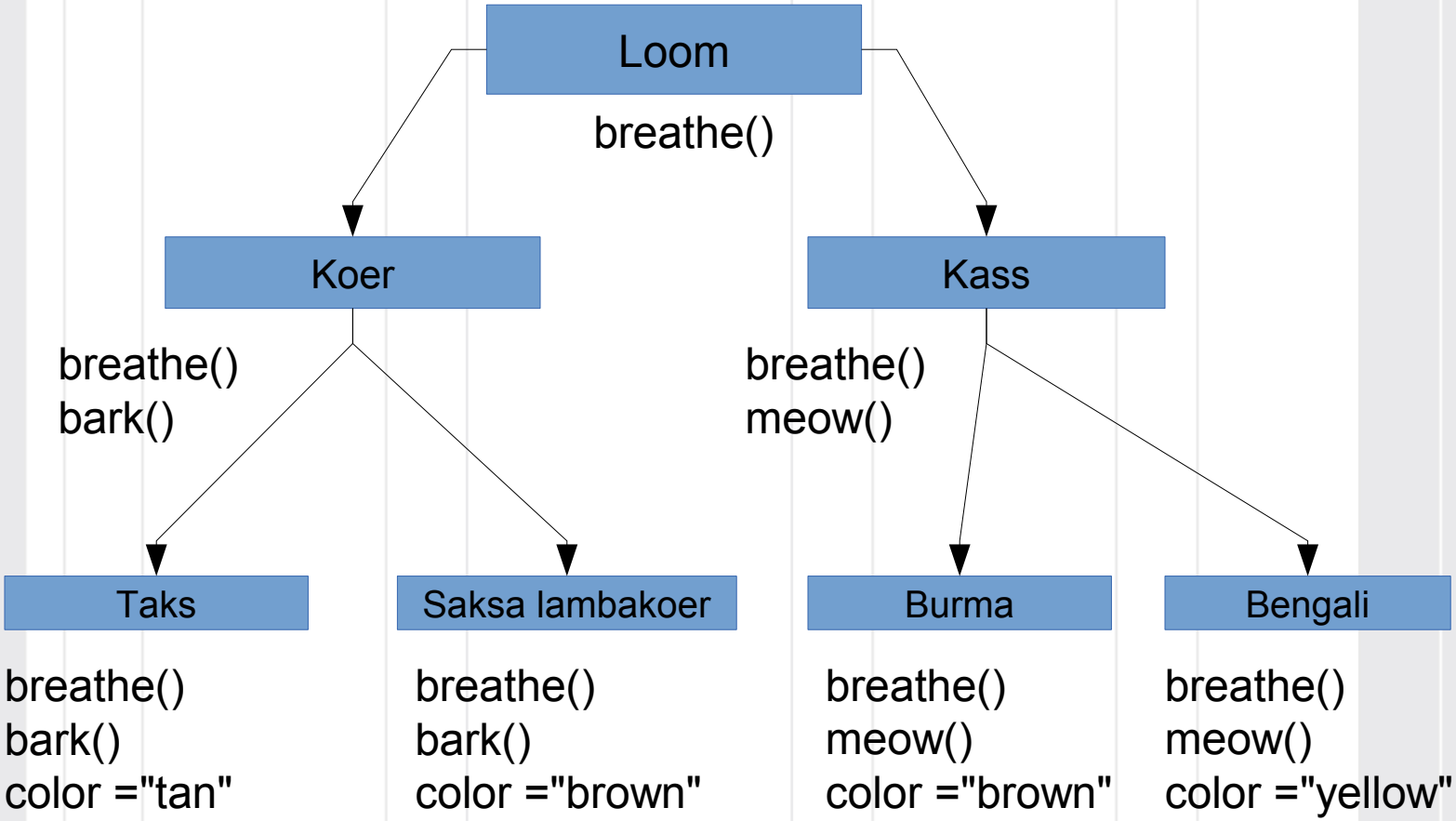
```
(1, 2, 3)
```

Objektid - olek

```
p2D = Point2D(3, 5)
p3D = Point3D(2, 3, 8)
p2D.print_point()
p3D.print_point()
p3D.x -= 1
p2D.print_point()
p3D.print_point()
p2D.x += 1
p2D.print_point()
p3D.print_point()
```

```
(3, 5)
(2, 3, 8)
(3, 5)
(1, 3, 8)
(4, 5)
(1, 3, 8)
```

Pärimine - hierarhiad



Ülesanne

Ülesanne on nähtaval

- **https://ained.ttu.ee**
- **https://courses.cs.ttu.ee/pages/ITI0140**

Ülesane lisakirjeldus

Aine kodulehel on link moodulile "**simulator.py**".

Teie loodud lahendus kasutab seda moodulit kasutades importi.

Ülesande lisakirjeldus

Teie esimeseks ülesandeks on luua klass ***Robot***, mis pärineb klassist ***Agent*** (*defineeritud moodulis simulator.py*).

Ülesande lisakirjeldus

Klassis ***Agent*** on defineeritud **kolm** olulist funktsiooni:

1) ***detect(****direction****)*** – sensorite abil ümbruse tajumine (0 = põhi, 1 = kirre, 2 = ida, 3 = kagu, 4 = lõuna, 5 = edel, 6 = lääs, 7 = loe)

2) ***turn_and_drive_straight(****way****)*** – pööramine vasakule või paremale (-2 = 90 kraadi vasakule, -1 = 45 kraadi vasakule, 1 = 45 kraadi paremale, 2 = 90 kraadi paremale)

3) **compass()** - tagastab hetke liikumissuuna

Ülesande lisakirjeldus

turn_and_drive_straight(*way****)***

See reaalselt tähendab, et n-ö pööratakse rattad soovitud asendisse, aga tegelikku sõitmist ei toimu selle käsuga.

Ülesande lisakirjeldus

```
world = simulator.World(
        width = 10,
        height = 10,
        sleep_time = 1,
        treasure = None,
        obstacles = [(5, 5)],
        reliability = 0.9,
        endurance = 10000)
```

Iga sammu viiteaeg sekundites (parem jälgida kui 1, aga võib ka 0)

None = random

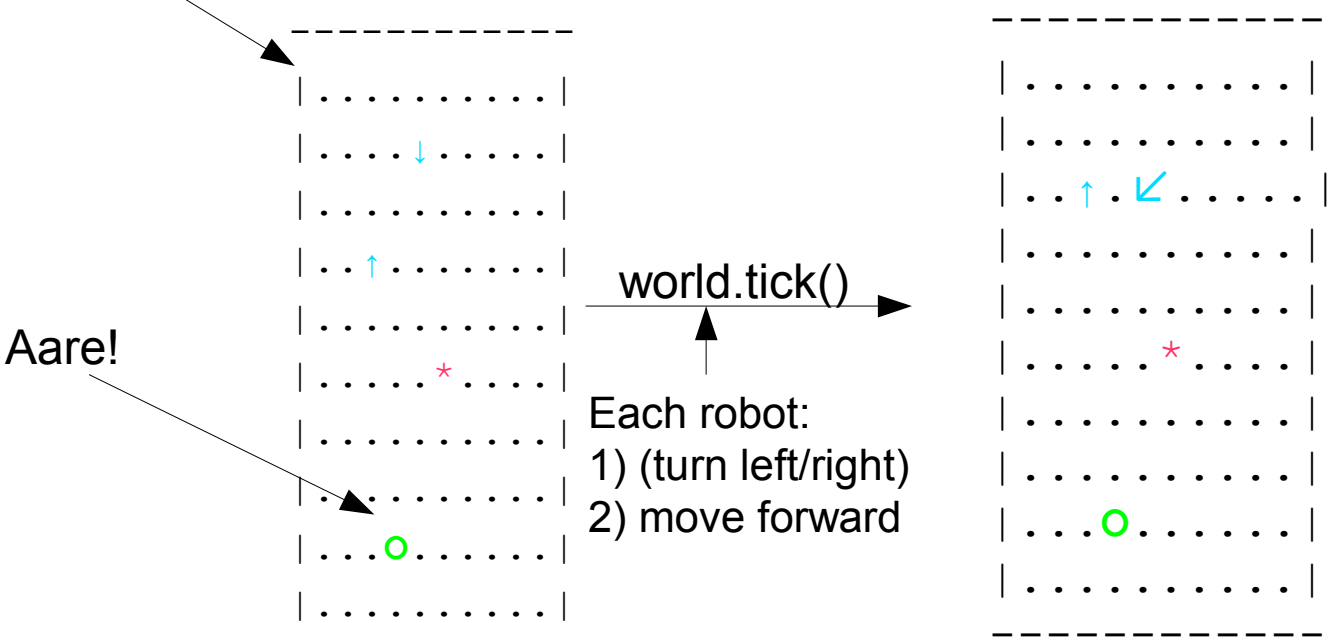
Robotite töökindlus = tõenäosus, et ei leki õli põrandale

Mitu 90 kraadist pööret robot teha saab

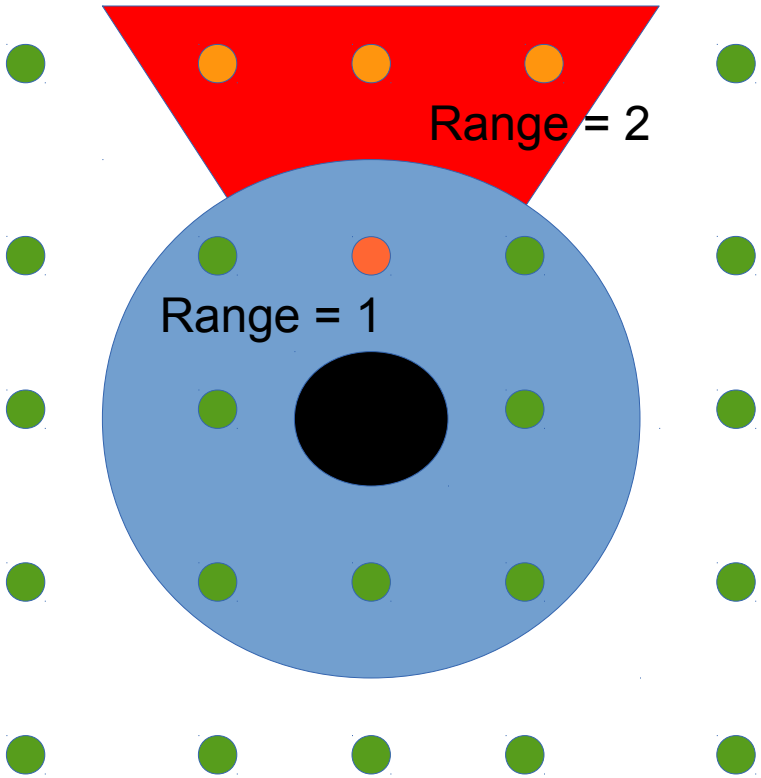
```
import simulator
import random
world = simulator.World(width = 10, height = 10, sleep_time = 1, treasure =
None, obstacles = [(5, 5)], reliability = 0.9, endurance = 10000)

robots = []
robots.append(Robot(world, 4, 1, 4))
robots.append(Robot(world, 2, 3, 0))
while True:
    for robot in robots:
        robot.turn_and_drive_straight(random.randint(-1, 1))
    world.print_state()
    world.tick()
```

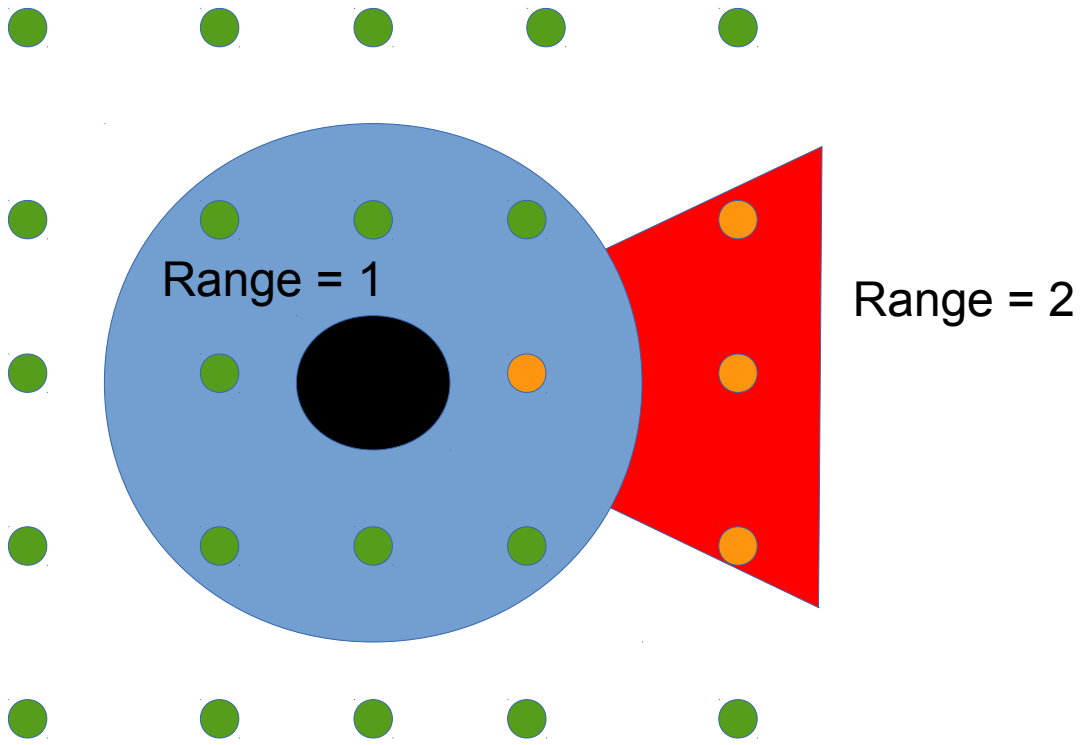
Siin peaks välja kutsuma decide(), see on lihtsalt näide



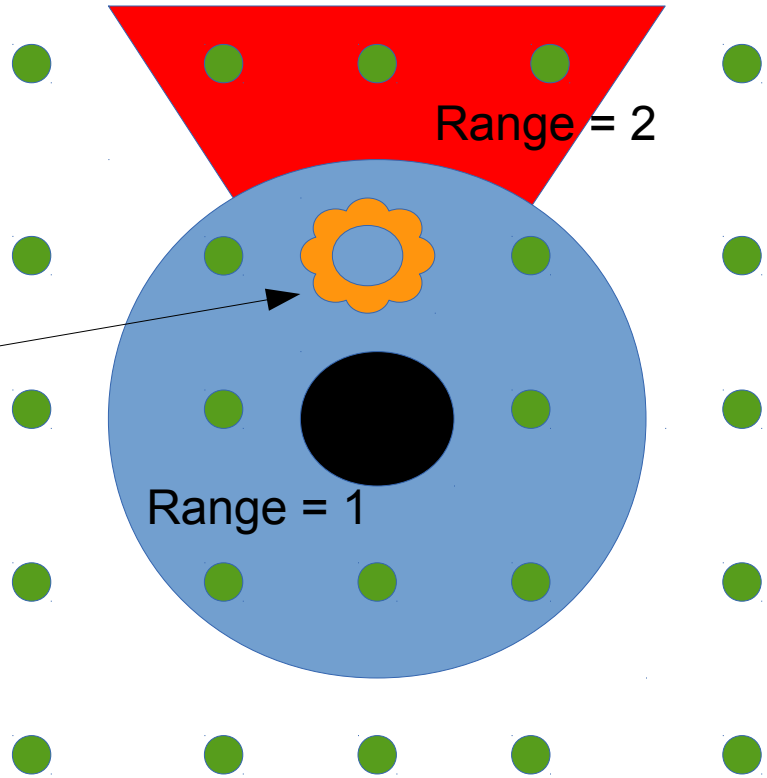
Detect(0) = North



Detect(2) = East

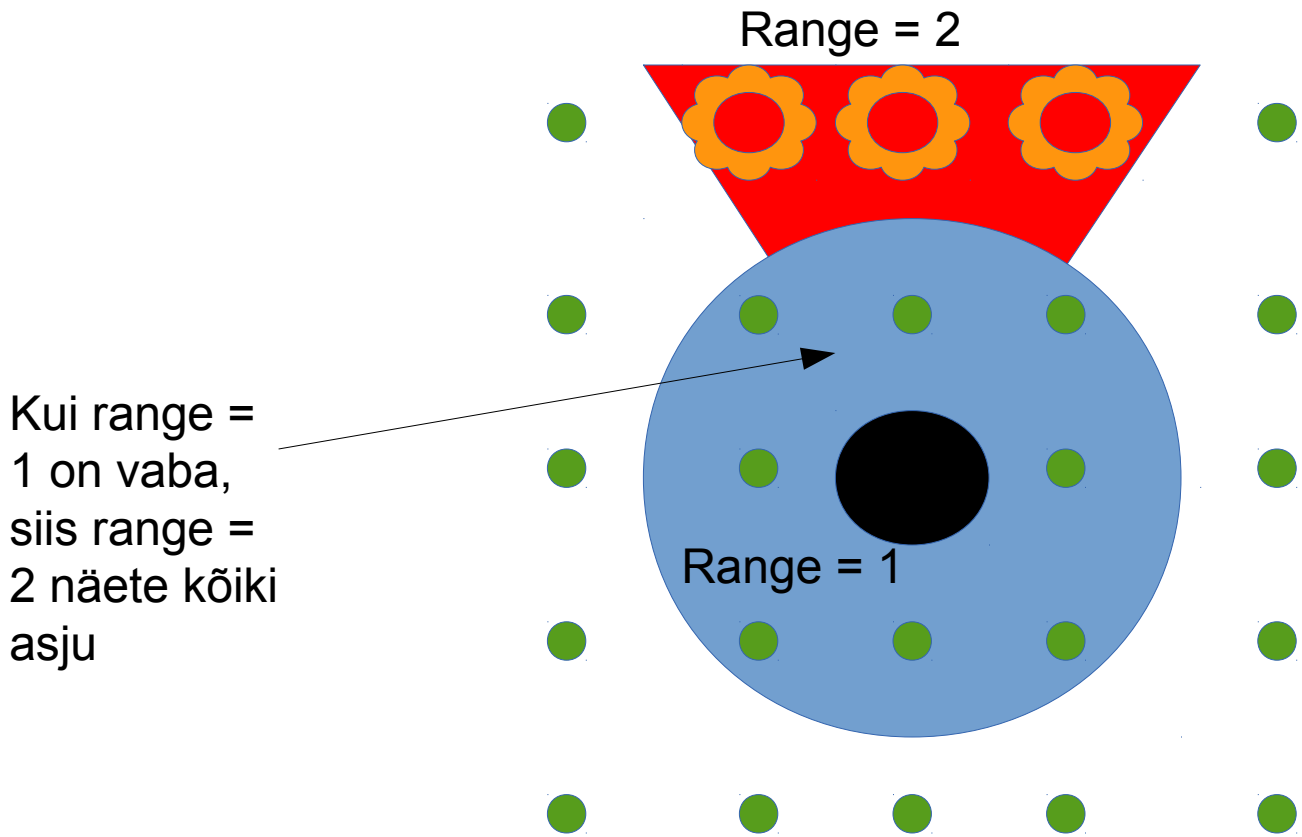


Detect(0) = North

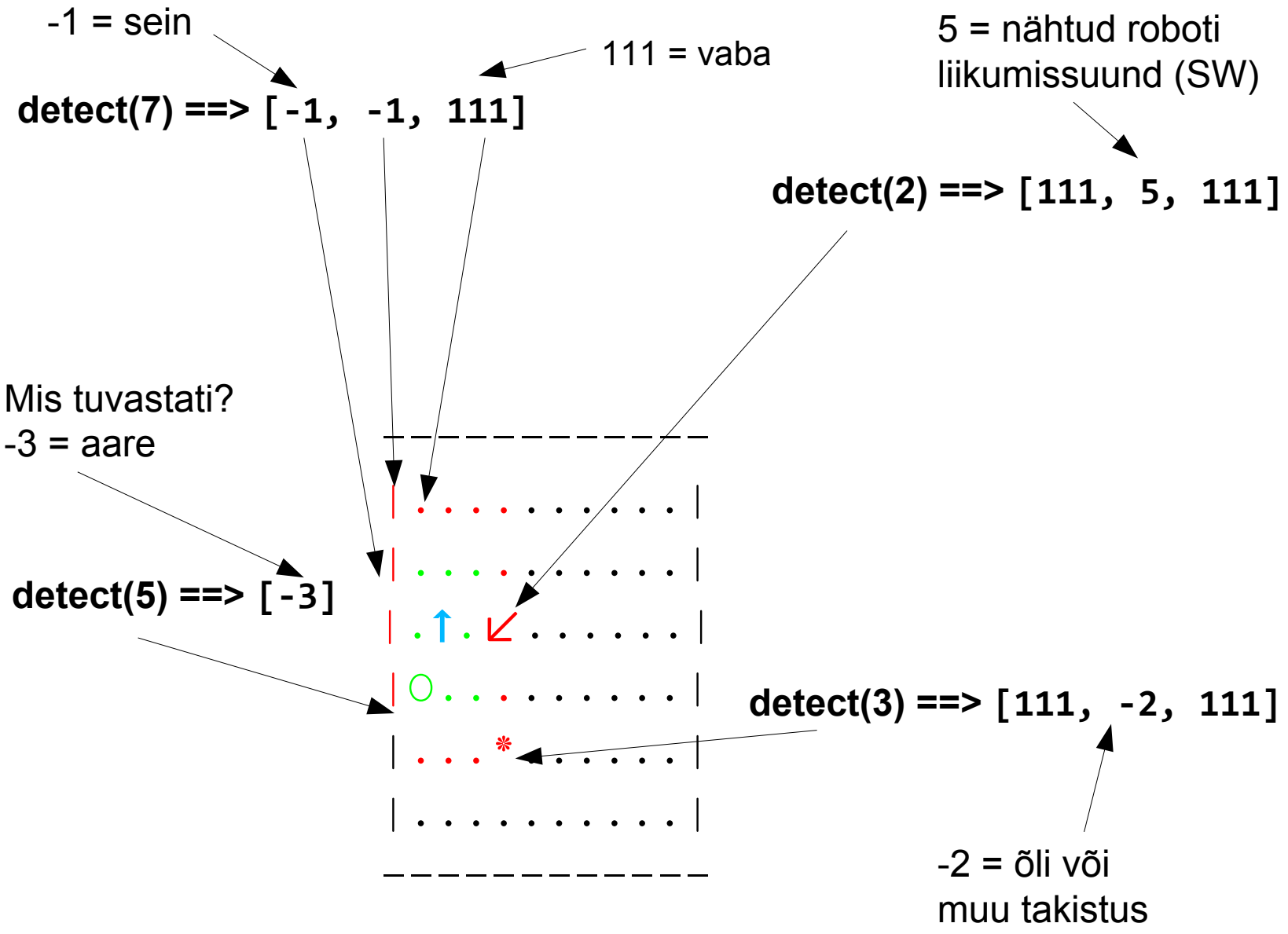


Kui näete range = 1 peal midagi, siis range = 2 ei näe, sest range = 1 blokeerib

Detect(0) = North



detect() näitab maksimaalselt kauguseni 2 ruutu(märgitud punasega)

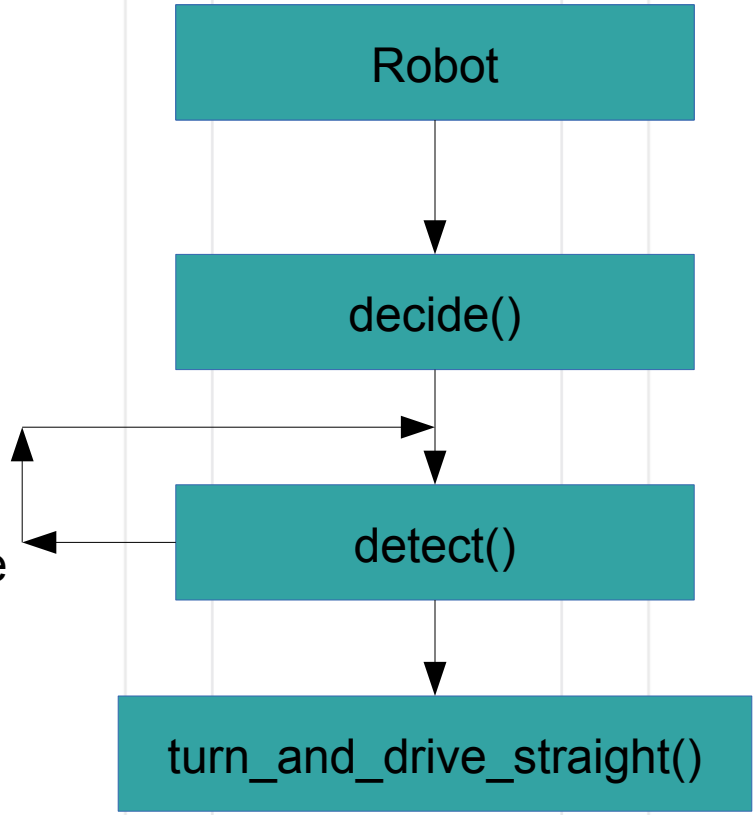


Teie põhiülesandeks on kirjutada teie loodud **Robot** klassile funktsioon **decide*()***, mis kasutades funktsiooni ***detect(****direction****)*** annab robotile juhtkäske rakendades ***turn_and_drive_straight(****way****)*** funktsiooni.

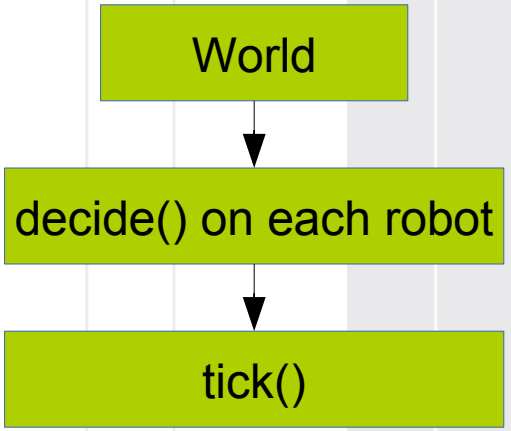
Eesmärk: leida aare kasutades ***decide****()* funktsiooni ja sõita robotiga sellele ruudule.

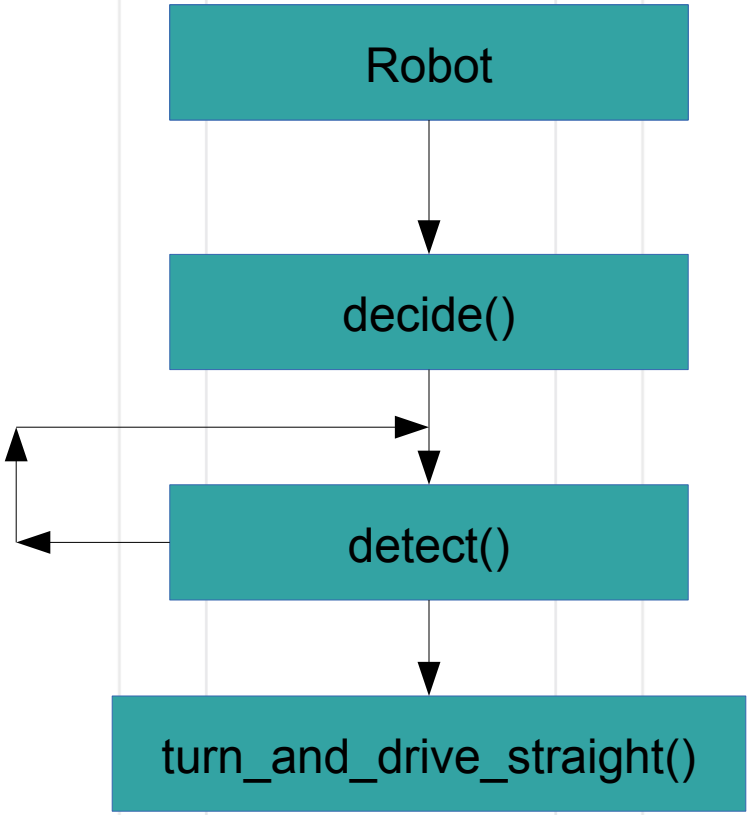
Üldine ülesehitus

Näiteks vaatab ette, vasakule, paremale ja otsustab selle põhjal



Üldine ülesehitus





Näiteks vaatab ette, vasakule, paremale ja otsustab selle põhjal

Piirangud:

Robot liigub iga ajaühikuga alati ühe ruudu võrra edasi (ei ole võimalik peatuda).

Robot saab pöörata ühe ajaühiku jooksul ainult ühe korra (turn_and_drive_straight(way) kasutamisel toimub liikumine kas vasakule->otse või paremale->otse, ei saa pöörata mitu korda (vasakule-vasakule->otse))

Robotid ei tohi sõita vastu seina, takistuste pihta ega põrgata teineteisega kokku.

Ei tohi "häkkida" simulaatorit ja kasutada simulaatoris olevaid olekumuutujaid robotite ja maailma kohta. Robot suhtleb simulatsioonikeskkonnaga ainult detect() ja turn() funktsioonide kaudu.