Annexe XX : Code simulateur

*Ce code correspond à une version à jour du 26 mai 2015. Certaines modifications peuvent exister entre cette version et celle fournie au client.*

Simulation.py

'''

@auhtor : Corentin R

@date : February 2015

This file contains the function called by the programm when running on simulation

'''

import pygame,pylygon,threading,sys,time

from pygame.locals import \*

from modules.daarrt2d import \*

from modules.world import \*

from modules.constantes import \*

from modules.info import \*

HIGH = "1"

LOW = "0"

INPUT = "0"

OUTPUT = "1"

def simulate(ns,package,changeData,changeDataEnco,changeSonarLeft,changeSonarRight, changeSonarFront , changeSonarBack , changeCap):

#creation of objects required

myWorld=World(worldName)

worldLenght=myWorld.generer()

pygame.init()

simu=pygame.display.set\_mode((worldLenght[0]+windowWidth,worldLenght[1]))

info=Info(worldLenght)

bg=pygame.image.load(background).convert()

pygame.display.set\_caption(titre)

#first printing

pos=myWorld.afficher(simu,True)

daarrt2d=DAARRT2d(pos)

#simulation loop

pygame.key.set\_repeat(400, 30)

runSimu=True

display=0

while(runSimu==True) :

#Limitation vitesse

pygame.time.Clock().tick(speed)

simu.blit(bg,(0,0))

myWorld.afficher(simu,False)

package,changeData=cleanBus(package,changeData,changeDataEnco,daarrt2d)

runSimu=daarrt2d.update(package,myWorld,changeCap)

info.afficher(simu,daarrt2d,display)

daarrt2d.draw(simu)

virtualSonar(daarrt2d,myWorld,changeSonarLeft,changeSonarRight,changeSonarFront,changeSonarBack)

pygame.display.flip()

for event in pygame.event.get():

if event.type == pygame.QUIT :

runSimu = False

if event.type == pygame.MOUSEBUTTONDOWN and event.pos[0]<worldLenght[0] + 40 and event.pos[0] > worldLenght[0] + 10 and event.pos[1] < worldLenght[1] - 30 and event.pos[1] > worldLenght[1] - 60 :

display += 1

if display >= 2 :

display = 0

if event.type == KEYDOWN and event.key == K\_l :

if display == 2 :

display = 0

else :

display = 2

print "Ending Simulation..."

pygame.quit()

ns.isAlive=False

def cleanBus(package,changeData,changeDataEnco,daarrt):

'''

return the package updated and the element change Data cleaned

args :

package : dictionnary corresponding to the bus

changeData : list of elements which changed since the last update

'''

while(len(changeData)>0):

try :

package[changeData[-1][0]]=changeData[-1][1]

changeData.pop()

except : pass

while(len(changeDataEnco)>0):

try :

changeDataEnco.pop()

except : pass

changeDataEnco.append(["LeftEnco",daarrt.leftEnco])

changeDataEnco.append(["RightEnco",daarrt.rightEnco])

return package,changeData

def virtualSonar(daarrt2d,world,changeSonarLeft,changeSonarRight,changeSonarFront,changeSonarBack):

try :

if(len(changeSonarLeft)>1):

changeSonarLeft.pop()

except : print "FailLeft"

changeSonarLeft.append(daarrt2d.sonarDist[2])

try :

if(len(changeSonarRight)>1):

changeSonarRight.pop()

except : print "FailRight"

changeSonarRight.append(daarrt2d.sonarDist[3])

try :

if(len(changeSonarFront)>1) :

changeSonarFront.pop()

except : print "FailFront"

changeSonarFront.append(daarrt2d.sonarDist[0])

try :

if(len (changeSonarBack)>1):

changeSonarBack.pop()

except : print "FailBack"

changeSonarBack.append(daarrt2d.sonarDist[1])

Constantes.py

'''

@auhtor : Corentin R

@date : February 2015

This file contains robot's constants. If any element of the robot is changed,

please modify here

CAREFUL : 1px=1cm

'''

scale = 2 # Multiplicative scale for the robot

elementSize = 30 #Each element is the same size in pixels

#Sizes for the window

windowHeight = 0

windowWidth = 400

#Sizes for the robot

robotLenght = 23\*scale

robotWidth = 12\*scale

wheelLenght = 12.5\*scale

wheelWidth = 10\*scale

wheelRadius = 6.25\*scale

clawLenght = 10\*scale

clawWidthClosed = 2\*scale

clawWidthOpened = 2\*scale

clawBase = 10\*scale

nTicks = 50

#Sizes for sonar

sonarLenght = 400.0

sonarAngle = 10.0 #in degrees

nFrontSonar = 1#number

nLeftSonar = 1

nRightSonar = 1

nBackSonar = 1

#Elements for the simulation

speed=30

#Path to useful files and other stuff

titre = "Simulateur DAARRT"

worldName="vDaarrt/data/World/world1.txt"

mur = "vDaarrt/data/mur.png"

obstacle = "vDaarrt/data/obstacle.png"

nomRobot = "DAARRT"

background = "vDaarrt/data/background.jpg"

button ="vDaarrt/data/button.png"

logo = "vDaarrt/data/logo.png"

Daarrt2d.py

import time, math, sys,struct

import pygame,pylygon

from pygame.locals import \*

from numpy import array

import math

import os

import geometry as geo

from constantes import \*

def high\_low\_signed\_int(high\_byte, low\_byte):

'''

Convert low and low and high byte to signed int

'''

return (high\_byte << 8) + low\_byte

class DAARRT2d:

def \_\_init\_\_(self,daarrtpos):

#variable values

try :

self.posX = daarrtpos[0]

self.posY = daarrtpos[1]

except :

self.posX=250.0

self.posY = 250.0

self.leftSpeed = 0.0

self.rightSpeed = 0.0

self.robotCap =0.0#angle between 0 and 360

self.clawOpening = 0.0 #angle between 0 and 180 (0 close/180 open)

self.clawHeight = 0.0 #angle between 0 and 180 (0 up/180 bottom)

self.cmdClawOpening = 0.0

self.cmdClawHeight = 0.0

#geometry constant

self.clawLenght = clawLenght

self.clawWidthOpened = clawWidthOpened

self.clawWidthClosed = clawWidthClosed

self.clawBase=clawBase

self.robotLenght = robotLenght

self.robotWidth = robotWidth

self.wheelLenght = wheelLenght

self.wheelWidth = wheelWidth

self.wheelRadius = wheelRadius

self.sonarAngle= sonarAngle

self.sonarLenght = sonarLenght

self.nFrontSonar = nFrontSonar

self.nLeftSonar = nLeftSonar

self.nRightSonar = nRightSonar

self.nBackSonar = nBackSonar

#creating the robot

self.daarrt = geo.createRobot(self)

self.sonar = geo.createSonar(self)

self.sonarDist=[0.0,0.0,0.0,0.0]

self.leftEnco=0.0

self.rightEnco=0.0

def sonarCollide(self,world):

finalDist=[]

j=0

for beam in self.sonar:

dist=[]

for obstacle in world.obstacle :

if not beam.collidepoly(obstacle) is False :

tmp=1000

i=0

for beamPoint in beam.P :

distance=(math.sqrt((beamPoint[0]-self.posX)\*(beamPoint[0]-self.posX) + (beamPoint[1]-self.posY)\*(beamPoint[1]-self.posY)))

if distance<tmp :

index=i

tmp=distance

i+=1

x0=beam.P[index][0]

y0=beam.P[index][1]

ply=pylygon.Polygon([(x0,y0)])

tmp=ply.distance(obstacle)

dist.append(math.sqrt(tmp[0]\*tmp[0] + tmp[1]\*tmp[1]))

try :

finalDist.append(min(dist))

#finalDist.append(10\*j)

#j+=1

except :

finalDist.append(1000)

self.sonarDist=finalDist

return finalDist

def updateOdo(self,package) :

motorLeftAngSpeed = 0

motorRightAngSpeed = 0

dAG=math.sqrt(self.robotWidth\*\*2 + self.robotLenght\*\*2)/2

dBG=dAG

dTime = 1.0

theta=self.robotCap\*math.pi/180.0

thetaPrime = (self.leftSpeed-self.rightSpeed)/self.robotWidth

vGx =(((self.rightSpeed+self.leftSpeed)/self.robotWidth)\*math.cos(theta))

vGy = (((self.rightSpeed+self.leftSpeed)/self.robotWidth)\*math.sin(theta))

vAx = vGx + 0

vAy = vGy + dAG \* math.cos(theta)\* thetaPrime

vA=math.sqrt(vAx\*\*2 + vAy\*\*2)

dA = vA\*dTime

vBx = vGx + 0

vBy = vGy - dBG \* math.cos(theta)\* thetaPrime

vB=math.sqrt(vBx\*\*2 + vBy\*\*2)

dB = vB\*dTime

try :

self.leftEnco += ((nTicks\*dB/(math.pi\*wheelRadius\*2))\*(thetaPrime/abs(thetaPrime)))

self.rightEnco += ((nTicks\*dA/(math.pi\*wheelRadius\*2))\*(thetaPrime/abs(thetaPrime)))

except :

self.leftEnco += (nTicks\*dB/(math.pi\*wheelRadius\*2))

self.rightEnco += (nTicks\*dA/(math.pi\*wheelRadius\*2))

self.leftEnco = int(self.leftEnco) % 65335

self.rightEnco = int(self.rightEnco) % 65335

def update(self , package , world,changeCap):

dt = 1.0

self.updateSpeed(package)

self.updateClaw(package)

self.sonarCollide(world)

self.updateOdo(package)

theta=self.robotCap\*math.pi/180.0

self.posX += (((self.rightSpeed+self.leftSpeed)/self.robotWidth)\*math.cos(theta))\*dt

self.posY += (((self.rightSpeed+self.leftSpeed)/self.robotWidth)\*math.sin(theta))\*dt

self.robotCap += (self.leftSpeed-self.rightSpeed)\*dt/self.robotWidth

self.robotCap = self.robotCap %360

self.daarrt = geo.createRobot(self)

self.sonar=geo.createSonar(self)

while (len(changeCap)>0):

try:

changeCap.pop()

except : pass

changeCap.append(self.robotCap)

if(self.collide(world)):

print "Collision detectee"

return False

return True

def updateClaw(self,package):

cHeight=high\_low\_signed\_int(package['servo\_1\_high\_byte'],package['servo\_1\_low\_byte'])

cOpening=high\_low\_signed\_int(package['servo\_2\_high\_byte'],package['servo\_2\_low\_byte'])

if cHeight<=0:

self.cmdClawHeight=0.0

else :

if cHeight>180 :

self.cmdClawHeight=180.0

else :

self.cmdClawHeight=cHeight

if cOpening<=0 :

self.cmdClawOpening=0.0

else :

if cOpening>180 :

self.cmdClawOpening=180.0

else :

self.cmdClawOpening=cOpening

if self.clawHeight<self.cmdClawHeight-1 :

self.clawHeight+=0.5

elif self.clawHeight>self.cmdClawHeight+1:

self.clawHeight-=0.5

if self.clawOpening<self.cmdClawOpening-1 :

self.clawOpening+=0.5

elif self.clawOpening>self.cmdClawOpening-1 :

self.clawOpening-=0.5

def updateSpeed(self , package):

lSpeed=high\_low\_signed\_int(package['lm\_speed\_high\_byte'],package['lm\_speed\_low\_byte'])

rSpeed=high\_low\_signed\_int(package['rm\_speed\_high\_byte'],package['rm\_speed\_low\_byte'])

a=200.0/(255.0-80.0)

b=-80.0\*a

if(lSpeed < 80 and lSpeed>-80):

self.leftSpeed = 0.0

elif(lSpeed>=0) :

self.leftSpeed = a\*lSpeed + b

else :

self.leftSpeed= a\*lSpeed -b

if(rSpeed < 80 and rSpeed>-80):

self.rightSpeed=0.0

elif(rSpeed>=0) :

self.rightSpeed=a\*rSpeed + b

else :

self.rightSpeed=a\*rSpeed -b

def collide(self,world):

for element in self.daarrt:

for obstacle in world.obstacle :

try :

(bool)(element.collidepoly(obstacle))

except :

return 1

return 0

def draw(self,simulation):

i=0

for element in self.daarrt :

if(i==6 or i==7):

pygame.draw.polygon(simulation,(0,0,255),element.P,0)

elif(i!=0):

pygame.draw.polygon(simulation,(255,0,0),element.P,0)

else:

pygame.draw.polygon(simulation,(255,0,0),element.P,1)

i+=1

for sonar in self.sonar :

pygame.draw.polygon(simulation,(40,100,83),sonar.P,1)

geometry.py

'''

@auhtor : Corentin R

@date : February 2015

This file contains geometric objects' constructors. Nothing should be changed

in this file as it would possibly change the comportment of the robot

'''

import math

import pylygon

from numpy import array

from constantes import \*

def createSonar(daarrt):

'''

return :

list of pylygon objects containing each sonar of the DAARRT

arguments :

daarrt object

'''

sonar=[]

theta = daarrt.sonarAngle\*math.pi/180.0

for i in range(daarrt.nFrontSonar) :

if daarrt.nFrontSonar>1 :

x0 = daarrt.robotLenght/2 + 2

y0 = daarrt.robotWidth\*i/(daarrt.nFrontSonar-1) - daarrt.robotWidth/2

else :

x0 = daarrt.robotLenght/2 + 2

y0 = 0

sonar.append(pylygon.Polygon([(x0,y0),(x0+sonarLenght\*math.cos(theta),y0+sonarLenght\*math.sin(theta)),(x0+sonarLenght\*math.cos(-theta),y0+sonarLenght\*math.sin(-theta))]))

for i in range (daarrt.nBackSonar) :

if daarrt.nBackSonar >1 :

x0 = -daarrt.robotLenght/2 + 2

y0 = daarrt.robotWidth\*i/(daarrt.nBackSonar-1) - daarrt.robotWidth/2

else :

x0 = -daarrt.robotLenght/2 + 2

y0 = 0

sonar.append(pylygon.Polygon([(x0,y0),(-x0-sonarLenght\*math.cos(theta),y0+sonarLenght\*math.sin(theta)),(-x0-sonarLenght\*math.cos(-theta),y0+sonarLenght\*math.sin(-theta))]))

for i in range (daarrt.nLeftSonar):

if daarrt.nLeftSonar>1 :

x0 = -daarrt.robotLenght/2 + daarrt.wheelLenght/2 + (daarrt.robotLenght-daarrt.robotWidth)\*i/(daarrt.nLeftSonar-1)

y0 = -daarrt.robotWidth/2 + 2

else :

x0 = 0

y0=-daarrt.robotWidth/2 + 2

sonar.append(pylygon.Polygon([(x0,y0),(x0+sonarLenght\*math.sin(theta),-y0-sonarLenght\*math.cos(theta)),(x0+sonarLenght\*math.sin(-theta),-y0-sonarLenght\*math.cos(-theta))]))

for i in range (daarrt.nRightSonar):

if daarrt.nRightSonar>1 :

x0 = -daarrt.robotLenght/2 + daarrt.wheelLenght/2 + (daarrt.robotLenght-daarrt.robotWidth)\*i/(daarrt.nLeftSonar-1)

y0 = daarrt.robotWidth/2 + 2

else :

x0 = 0

y0=daarrt.robotWidth/2 + 2

sonar.append(pylygon.Polygon([(x0,y0),(x0+sonarLenght\*math.sin(theta),+y0+sonarLenght\*math.cos(theta)),(x0+sonarLenght\*math.sin(-theta),+y0+sonarLenght\*math.cos(-theta))]))

sonar=[poly\_translate(poly\_rotate(element,daarrt.robotCap),daarrt.posX,daarrt.posY) for element in sonar]

return sonar

def createBody(daarrt,offsetX,offsetY):

'''

return :

list corresponding to the daarrt's body

arguments :

daarrt object

offsetX : optionnal offset on X axis

offsetY : optionnal offset on Y axis

'''

x0 = offsetX

y0 = offsetY

x1 = offsetX + daarrt.robotLenght

y1 = offsetY

x2 = offsetX + daarrt.robotLenght

y2 = offsetY + daarrt.robotWidth

x3 = offsetX

y3 = offsetY + daarrt.robotWidth

body = [(x0,y0) , (x1,y1) , (x2,y2) , (x3,y3)]

return body

def createWheel(daarrt,offsetX,offsetY):

'''

return :

list corresponding to a daarrt wheel

arguments :

daarrt object

offsetX : optionnal offset on X axis

offsetY : optionnal offset on Y axis

'''

x0 = offsetX

y0 = offsetY

x1 = offsetX

y1 = offsetY - daarrt.wheelWidth

x2 = offsetX + daarrt.wheelLenght

y2 = offsetY - daarrt.wheelWidth

x3 = offsetX + daarrt.wheelLenght

y3 = offsetY

wheel=[(x0,y0) , (x1,y1) , (x2,y2) , (x3,y3)]

return wheel

def createClaw(daarrt,offsetX,offsetY):

'''

return :

list corresponding to the daarrt's claw

arguments :

daarrt object

offsetX : optionnal offset on X axis

offsetY : optionnal offset on Y axis

'''

x0 = offsetX

y0 = offsetY

x1 = offsetX + daarrt.clawLenght

y1 = offsetY

x2 = offsetX + daarrt.clawLenght

y2 = offsetY + daarrt.clawWidthClosed/2

x3 = offsetX

y3 = offsetY + daarrt.clawWidthClosed/2

claw = [(x0,y0),(x1,y1),(x2,y2),(x3,y3)]

return claw

def createClawBase(daarrt,offsetX,offsetY):

'''

return :

list corresponding to the daarrt's clawBase

arguments :

daarrt object

offsetX : optionnal offset on X axis

offsetY : optionnal offset on Y axis

'''

x11 = offsetX

y11 = offsetY

x12 = offsetX + daarrt.clawBase

y12 = offsetY

x13 = offsetX + daarrt.clawBase

y13 = offsetY + daarrt.clawWidthClosed

x14 = offsetX

y14 = offsetY + daarrt.clawWidthClosed

clawBase=[(x11,y11),(x12,y12),(x13,y13),(x14,y14)]

return clawBase

def createRobot(daarrt):

'''

return :

list pylygon object corresponding to the daarrt (wheel )

arguments :

daarrt object

'''

rotationCenter = [ -daarrt.robotLenght/2 , -daarrt.robotWidth/2 ]

body = createBody(daarrt , rotationCenter[0] , rotationCenter[1])

wheel1 = createWheel(daarrt , -daarrt.wheelLenght/2 + rotationCenter[0] , rotationCenter[1])

wheel2 = createWheel(daarrt , daarrt.robotLenght - daarrt.wheelLenght/2 + rotationCenter[0] , rotationCenter[1])

wheel3 = createWheel(daarrt , daarrt.robotLenght - daarrt.wheelLenght/2 + rotationCenter[0] , daarrt.robotWidth + daarrt.wheelWidth + rotationCenter[1])

wheel4 = createWheel(daarrt , -daarrt.wheelLenght/2 + rotationCenter[0] , daarrt.robotWidth + daarrt.wheelWidth + rotationCenter[1])

clawBase = createClawBase( daarrt , daarrt.robotLenght + rotationCenter[0] , daarrt.robotWidth/2 - daarrt.clawWidthOpened/2 + rotationCenter[1])

claw1 = createClaw(daarrt , daarrt.robotLenght + daarrt.clawBase + rotationCenter[0] , daarrt.robotWidth/2 - daarrt.clawWidthClosed/2 - daarrt.clawWidthOpened \* daarrt.clawOpening/180.0 + rotationCenter[1])

claw2 = createClaw(daarrt , daarrt.robotLenght + daarrt.clawBase + rotationCenter[0] , daarrt.robotWidth/2 + daarrt.clawWidthOpened \* daarrt.clawOpening/180.0 + rotationCenter[1])

robotTmp = [pylygon.Polygon(body) , pylygon.Polygon(clawBase) , pylygon.Polygon(wheel1) , pylygon.Polygon(wheel2) , pylygon.Polygon(wheel3) , pylygon.Polygon(wheel4) , pylygon.Polygon(claw1) , pylygon.Polygon(claw2)]

robot = [poly\_translate(poly\_rotate(element,daarrt.robotCap),daarrt.posX,daarrt.posY) for element in robotTmp]

return robot

def createObstacle(world,x,y):

'''

return :

pylygon object corresponding to an obstacle

arguments :

x : position on x Axis

y : position on y Axis

'''

obs=pylygon.Polygon([(x,y),(x+world.elementSize,y),(x+world.elementSize,y+world.elementSize),(x,y+world.elementSize)])

return obs

def createWall(world,num\_case,num\_line):

'''

return :

list of pylygon object corresponding to the walls of a World

arguments :

num\_case : number of case

num\_line : number of line

'''

polyObs1=[(0,0),((num\_case) \* world.elementSize,0),((num\_case) \* world.elementSize,world.elementSize),(0,world.elementSize)]

polyObs2=[(0,world.elementSize),(world.elementSize,world.elementSize),(world.elementSize,(num\_line) \* world.elementSize),(0,(num\_line) \* world.elementSize)]

polyObs3=[((num\_case - 1) \* world.elementSize,world.elementSize),((num\_case) \* world.elementSize,world.elementSize),((num\_case) \* world.elementSize,(num\_line - 1) \* world.elementSize),(((num\_case - 1) \* world.elementSize,(num\_line - 1) \* world.elementSize))]

polyObs4=[(0,(num\_line - 1) \* world.elementSize),(num\_case \* world.elementSize,(num\_line - 1) \* world.elementSize),(num\_case\*world.elementSize,num\_line \* world.elementSize),(0,num\_line \* world.elementSize)]

walls=[pylygon.Polygon(polyObs1),pylygon.Polygon(polyObs2),pylygon.Polygon(polyObs3),pylygon.Polygon(polyObs4)]

return walls

def poly\_translate (p, tx, ty):

'''

return :

pylygon object corresponding to the argument translated

arguments :

p : pylygon object to translate

tx : translation on X axis

ty : translation on Y axis

'''

points = p.P

pointsTranslates = array([(x+tx,y+ty) for (x,y) in points])

p.P = pointsTranslates

return pylygon.Polygon(p)

def poly\_rotate (p, theta):

'''

return :

pylygon object corresponding to the argument rotated

arguments :

p : pylygon object to rotate

theta : rotation angle in degrees

'''

theta=theta \* math.pi/180.0

points=p.P

pointsTournes = array([(x \* math.cos(theta)-y \* math.sin(theta),

x \* math.sin(theta) + y \* math.cos(theta)) for (x,y) in points])

p.P=pointsTournes

return pylygon.Polygon(p)

info.py

'''

@auhtor : Corentin R

@date : February 2015

This file contains Info stuff. It represents the right part of the window where

other views of the robot are available

'''

from constantes import \*

import pygame,pylygon

from pygame.locals import \*

from numpy import array

import math

class Info :

def \_\_init\_\_(self,sizeWorld) :

self.sizeWorld = sizeWorld

self.offsetX = self.sizeWorld[0]+10

self.offsetY = elementSize

self.myfont = pygame.font.SysFont("monospace", 16)

self.author\_display = self.myfont.render("Corentin Rifflart",1,(0,0,0))

self.date\_display = self.myfont.render("2015",1,(0,0,0))

self.team\_display = self.myfont.render("The Daarrt Team",1,(0,0,0))

self.cap\_string = "Cap : "

self.lSpeed\_string = "Vitesse Gauche : "

self.rSpeed\_string = "Vitesse Droite : "

self.logoImg = pygame.image.load(logo).convert\_alpha()

self.buttonImg = pygame.image.load(button).convert\_alpha()

def afficher(self,simu,daarrt2d,display) :

'''

return :

void

args :

simu : pygame object

daarrt2d : daarrt2d object

display : type of display (int)

draw the right part of the window (informations, views)

'''

pygame.draw.rect(simu,(0,0,0),(self.offsetX,self.offsetY,windowWidth-20,self.sizeWorld[1]/2),2)

self.\_\_afficherName(simu,daarrt2d)

if (display == 0 or display ==1) :

self.\_\_afficherScale(simu,daarrt2d,display)

self.\_\_afficherClaw(simu,daarrt2d,display)

if (display == 2 ):

self.\_\_afficherLogo(simu,daarrt2d,display)

self.\_\_afficherInfos(simu,daarrt2d)

self.\_\_afficherButton(simu)

def \_\_afficherInfos(self,simu,daarrt2d) :

simu.blit(self.myfont.render(self.cap\_string+str(daarrt2d.robotCap%360),1,(0,0,0)),(self.offsetX,self.offsetY+self.sizeWorld[1]/2 +5))

simu.blit(self.myfont.render(self.lSpeed\_string+str(daarrt2d.leftSpeed),1,(0,0,0)),(self.offsetX,self.offsetY+self.sizeWorld[1]/2 +20))

simu.blit(self.myfont.render(self.rSpeed\_string+str(daarrt2d.rightSpeed),1,(0,0,0)),(self.offsetX,self.offsetY+self.sizeWorld[1]/2 +35))

def \_\_afficherButton(self,simu):

simu.blit(self.buttonImg,(self.offsetX+10,self.sizeWorld[1]-60))

def \_\_afficherClaw(self,simu,daarrt2d,display) :

if(display==0):

pygame.draw.rect(simu,(0,255,0),(self.offsetX+70,self.offsetY+190-daarrt2d.clawHeight,10,10),1)

pygame.draw.rect(simu,(0,255,0),(self.offsetX+80+daarrt2d.clawOpening,self.offsetY+190-daarrt2d.clawHeight,10,10),1)

else :

cap =- (daarrt2d.clawHeight+270)%360

cap = cap\*math.pi/180.0

pygame.draw.line(simu,(255,0,0),(self.offsetX+50,self.offsetY+100),(self.offsetX+50+100\*math.cos(cap),self.offsetY+100+100\*math.sin(cap)),1)

def \_\_afficherLogo(self,simu,daarrt2d,display) :

simu.blit(self.logoImg,(self.offsetX + 70 ,self.offsetY + 90 ))

def \_\_afficherScale(self,simu,daarrt2d,display) :

pygame.draw.line(simu,(0,0,0),(self.offsetX+50,self.offsetY+10),(self.offsetX+50,self.offsetY+190),1)

pygame.draw.line(simu,(0,0,0),(self.offsetX+40,self.offsetY+10),(self.offsetX+60,self.offsetY+10),1)

pygame.draw.line(simu,(0,0,0),(self.offsetX+40,self.offsetY+190),(self.offsetX+60,self.offsetY+190),1)

pygame.draw.line(simu,(0,0,0),(self.offsetX+40,self.offsetY+100),(self.offsetX+60,self.offsetY+100),1)

simu.blit(self.myfont.render("180",1,(0,0,0)),(self.offsetX+5,self.offsetY))

simu.blit(self.myfont.render("90",1,(0,0,0)),(self.offsetX+5,self.offsetY+90))

simu.blit(self.myfont.render("0",1,(0,0,0)),(self.offsetX+5,self.offsetY+180))

if(display == 0) :

pygame.draw.line(simu,(0,0,0),(self.offsetX+80,self.offsetY+200),(self.offsetX+260,self.offsetY+200),1)

pygame.draw.line(simu,(0,0,0),(self.offsetX+80,self.offsetY+190),(self.offsetX+80,self.offsetY+210),1)

pygame.draw.line(simu,(0,0,0),(self.offsetX+170,self.offsetY+190),(self.offsetX+170,self.offsetY+210),1)

pygame.draw.line(simu,(0,0,0),(self.offsetX+260,self.offsetY+190),(self.offsetX+260,self.offsetY+210),1)

simu.blit(self.myfont.render("180",1,(0,0,0)),(self.offsetX+260,self.offsetY+210))

simu.blit(self.myfont.render("90",1,(0,0,0)),(self.offsetX+170,self.offsetY+210))

simu.blit(self.myfont.render("0",1,(0,0,0)),(self.offsetX+80,self.offsetY+210))

else :

pass

def \_\_afficherName(self,simu,daarrt2d):

simu.blit(self.author\_display, (self.offsetX+windowWidth-190, self.offsetY+self.sizeWorld[1]-70))

simu.blit(self.date\_display, (self.offsetX+windowWidth-190, self.offsetY+self.sizeWorld[1]-58))

simu.blit(self.team\_display, (self.offsetX+windowWidth-190, self.offsetY+self.sizeWorld[1]-48))

vRazor.py

import struct

from multiprocessing import Process,Manager

class vRazorIO():

def \_\_init\_\_(self):

manager=Manager()

self.changeCap=manager.list()

self.changeCap.append(0.0)

def getAngles(self):

try :

output=struct.pack("fff",self.changeCap[-1],0.0,0.0)

except:

print "Can't get value"

output=struct.pack("fff",0.0,0.0,0.0)

return output

def getRawSensorData(self):

output=struct.pack("fffffffff",0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)

return output

def getCalibratedSensorData(self):

output=struct.pack("fffffffff",0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)

return output

vSonar.py

from multiprocessing import Process,Manager

HIGH = "1"

LOW = "0"

INPUT = "0"

OUTPUT = "1"

class vSonar():

def \_\_init\_\_(self, id, GPIO\_ECHO, GPIO\_TRIGGER):

self.id = id

self.GPIO\_ECHO = vGPIO(GPIO\_ECHO)

self.GPIO\_TRIGGER = vGPIO(GPIO\_TRIGGER)

manager=Manager()

if(id == 1):

self.changeSonarLeft=manager.list()

elif (id == 2):

self.changeSonarRight=manager.list()

elif (id == 3):

self.changeSonarFront=manager.list()

elif (id == 4):

self.changeSonarBack=manager.list()

else :

pass

def getValue(self) :

try :

if(self.id==1):

return self.changeSonarLeft[-1]

elif (self.id == 2):

return self.changeSonarRight[-1]

elif (self.id == 3):

return self.changeSonarFront[-1]

elif (self.id == 4):

return self.changeSonarBack[-1]

else :

pass

except :

return -1

class vGPIO():

def \_\_init\_\_(self, i) :

pass

def set(self, state) :

pass

def read(self) :

pass

def write(self, state) :

pass

def clean(self) :

pass

vTrex.py

from multiprocessing import Process,Manager

import struct

def high\_byte(integer):

'''

Get the high byte from a int

'''

return integer >> 8

def low\_byte(integer):

'''

Get the low byte from a int

'''

return integer & 0xFF

class vTrex():

def \_\_init\_\_(self):

manager=Manager()

self.package=manager.dict()

self.changeData=manager.list()

self.changeDataEnco=manager.list()

self.initPackage()

self.leftEnco=0

self.rightEnco=0

def i2cWrite(self):

'''while len(self.changeData)>2 :

try :

self.changeData.pop()

except : pass'''

for key in self.package :

if self.package[key] != self.oldPackage[key]:

self.oldPackage[key]=self.package[key]

self.changeData.append([key,self.oldPackage[key]])

def i2cRead(self):

try :

self.leftEnco = int(self.changeDataEnco[0][1])

self.rightEnco = int(self.changeDataEnco[1][1])

except :

pass

return struct.pack(">BBhhhhhhhhhhh",self.package['start\_byte']

,0

,self.package['battery\_high\_byte']\*256 + self.package['battery\_low\_byte']

,self.package['lm\_speed\_high\_byte']\*256 + self.package['lm\_speed\_low\_byte']

,high\_byte(self.leftEnco)\*256 + low\_byte(self.leftEnco)

,self.package['rm\_speed\_high\_byte']\*256 + self.package['rm\_speed\_low\_byte']

,high\_byte(self.rightEnco)\*256 + low\_byte(self.rightEnco)

,0

,0

,0

,0

,0

,0)

def initPackage(self):

self.package = {

'start\_byte' : 15, # Start byte

'pwm\_freq' : 6, # PWMfreq

'lm\_speed\_high\_byte' : 0, # Left speed high byte

'lm\_speed\_low\_byte' : 0, # Left Speed low byte

'lm\_brake' : 0, # Left brake

'rm\_speed\_high\_byte' : 0, # Right Speed high byte

'rm\_speed\_low\_byte' : 0, # Right Speed low byte

'rm\_brake' : 0, # Right brake

'servo\_1\_high\_byte' : 0, # Servo 1 high byte

'servo\_1\_low\_byte' : 0, # Servo 1 low byte

'servo\_2\_high\_byte' : 0, # Servo 2 high byte

'servo\_2\_low\_byte' : 0, # Servo 2 low byte

'servo\_3\_high\_byte' : 0, # Servo 3 high byte

'servo\_3\_low\_byte' : 0, # Servo 3 low byte

'servo\_4\_high\_byte' : 0, # Servo 4 high byte

'servo\_4\_low\_byte' : 0, # Servo 4 low byte

'servo\_5\_high\_byte' : 0, # Servo 5 high byte

'servo\_5\_low\_byte' : 0, # Servo 5 low byte

'servo\_6\_high\_byte' : 0, # Servo 6 high byte

'servo\_6\_low\_byte' : 0, # Servo 6 low byte

'devibrate' : 50, # Devibrate

'impact\_sensitivity\_high\_byte' : 0, # Impact sensitivity high byte

'impact\_sensitivity\_low\_byte' : 50, # Impact sensitivity low byte

'battery\_high\_byte' : 0, # Battery voltage high byte

'battery\_low\_byte' : 50, # Battery voltage low byte

'i2c\_address' : 7, # I2C slave address

'i2c\_clock' : 0 # I2C clock frequency

}

self.oldPackage= {

'start\_byte' : 15, # Start byte

'pwm\_freq' : 6, # PWMfreq

'lm\_speed\_high\_byte' : 0, # Left speed high byte

'lm\_speed\_low\_byte' : 0, # Left Speed low byte

'lm\_brake' : 0, # Left brake

'rm\_speed\_high\_byte' : 0, # Right Speed high byte

'rm\_speed\_low\_byte' : 0, # Right Speed low byte

'rm\_brake' : 0, # Right brake

'servo\_1\_high\_byte' : 0, # Servo 1 high byte

'servo\_1\_low\_byte' : 0, # Servo 1 low byte

'servo\_2\_high\_byte' : 0, # Servo 2 high byte

'servo\_2\_low\_byte' : 0, # Servo 2 low byte

'servo\_3\_high\_byte' : 0, # Servo 3 high byte

'servo\_3\_low\_byte' : 0, # Servo 3 low byte

'servo\_4\_high\_byte' : 0, # Servo 4 high byte

'servo\_4\_low\_byte' : 0, # Servo 4 low byte

'servo\_5\_high\_byte' : 0, # Servo 5 high byte

'servo\_5\_low\_byte' : 0, # Servo 5 low byte

'servo\_6\_high\_byte' : 0, # Servo 6 high byte

'servo\_6\_low\_byte' : 0, # Servo 6 low byte

'devibrate' : 50, # Devibrate

'impact\_sensitivity\_high\_byte' : 0, # Impact sensitivity high byte

'impact\_sensitivity\_low\_byte' : 50, # Impact sensitivity low byte

'battery\_high\_byte' : 0, # Battery voltage high byte

'battery\_low\_byte' : 50, # Battery voltage low byte

'i2c\_address' : 7, # I2C slave address

'i2c\_clock' : 0 # I2C clock frequency

}

World.py

'''

@auhtor : Corentin R

@date : February 2015

This file contains world generation stuff.

It loads a .txt file and generates a structure understable by the program.

'''

import time, math, sys

import pygame,pylygon

from pygame.locals import \*

from numpy import array

import math

import os

import geometry as geo

from constantes import \*

class World :

def \_\_init\_\_(self,fichier):

self.nom = fichier

self.structure = 0

self.obstacle = []

self.numberCase = 0

self.numberLine = 0

self.elementSize = elementSize

def generer(self):

"""

return a list containing lenght and width for the windows

args :

void

generate a world from a .txt file. The path must be indicated in

constantes.py

"""

num\_case = 0

num\_line = 0

with open(self.nom,"r") as fichier :

structure\_monde = []

for line in fichier :

line\_monde = []

for element in line :

if element != '\n':

line\_monde.append(element)

num\_case += 1

if element == 'L':

num\_case = 0

structure\_monde.append(line\_monde)

num\_line += 1

self.structure = structure\_monde

#num\_line -= 2

self.numberCase = num\_case

self.numberLine = num\_line

self.obstacle = geo.createWall(self,self.numberCase,self.numberLine)

return [self.numberCase\*30 , self.numberLine\*30]

def afficher(self,simulation,initiate):

"""

no return

args :

simulation : pygame object

initiate : boolean, true if it is the first call, false otherwise

draw the world in the pygame object

"""

num\_line = 0

compteur = 0

myWall = pygame.image.load(mur).convert\_alpha()

myObstacle = pygame.image.load(obstacle).convert\_alpha()

for ligne in self.structure :

num\_case = 0

for element in ligne :

x = num\_case\*elementSize

y = num\_line\*elementSize

if element == 'M':

simulation.blit(myWall,(x,y))

elif element == 'O':

simulation.blit(myObstacle,(x,y))

if( initiate == True ):

self.obstacle.append(geo.createObstacle(self,x,y))

elif element == 'D' :

daarrtpos=[x,y]

num\_case += 1

num\_line += 1

try :

return daarrtpos

except : return [250.0,250.0]