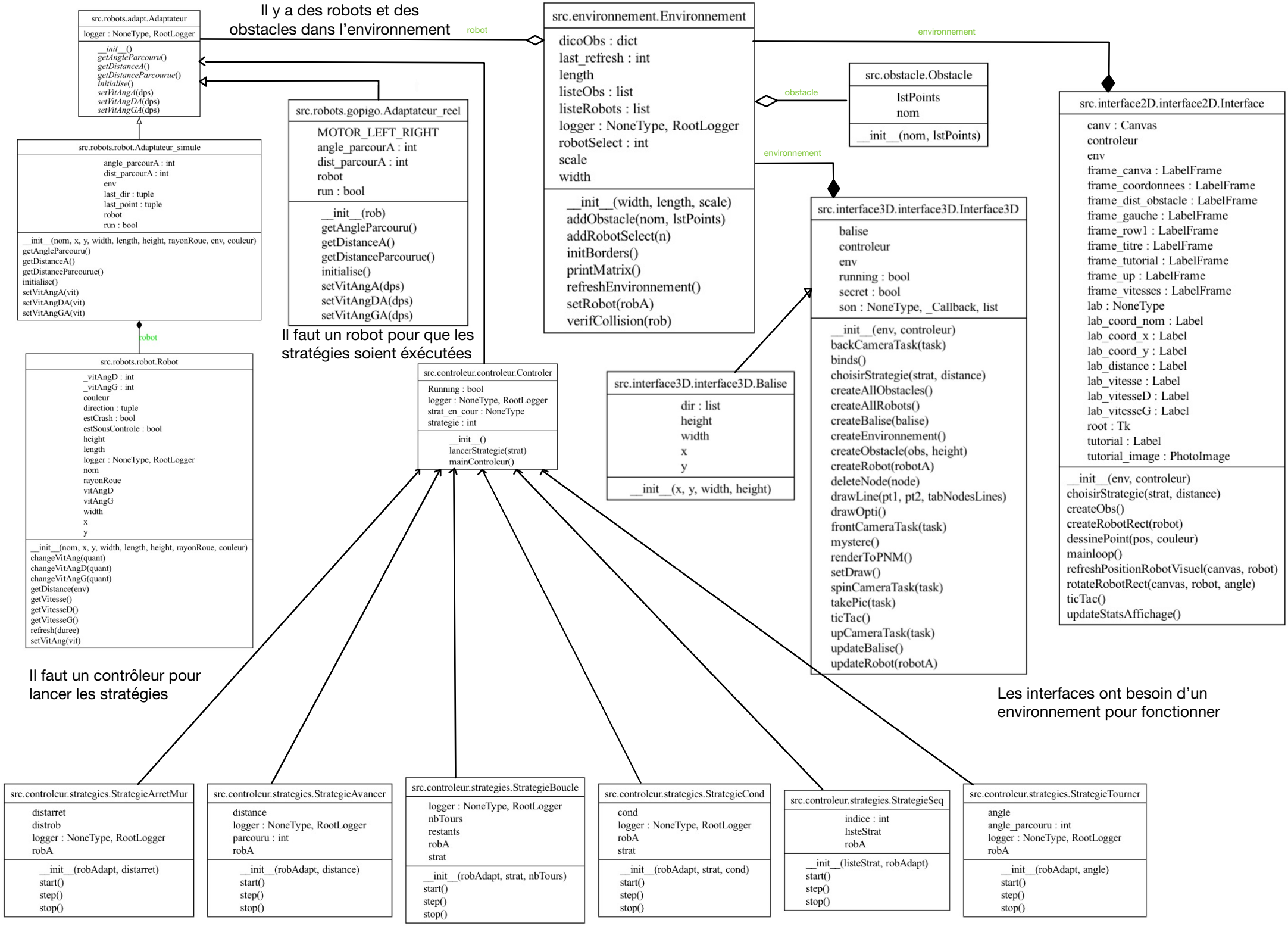
Il y a des robots et des obstacles dans l'environnement
robot
environnement
obstacle
environnement
robot
src.robots.adapt.Adaptateur
logger : NoneType, RootLogger
__init__()
getAngleParcouru()
getDistanceA()
getDistanceParcourue()
initialise()
setVitAngA(dps)
setVitAngDA(dps)
setVitAngGA(dps)
src.robots.robot.Adaptateur_simule
angle_parcourA : int
dist_parcourA : int
env
last_dir : tuple
last_point : tuple
robot
run : bool
__init__(nom, x, y, width, length, height, rayonRoue, env, couleur)
getAngleParcouru()
getDistanceA()
getDistanceParcourue()
initialise()
setVitAngA(vit)
setVitAngDA(vit)
setVitAngGA(vit)
robot
src.robots.robot.Robot
_vitAngD : int
_vitAngG : int
couleur
direction : tuple
estCrash : bool
estSousControle : bool
height
length
logger : NoneType, RootLogger
nom
rayonRoue
vitAngD
vitAngG
width
x
y
__init__(nom, x, y, width, length, height, rayonRoue, couleur)
changeVitAng(quant)
changeVitAngD(quant)
changeVitAngG(quant)
getDistance(env)
getVitesse()
getVitesseD()
getVitesseG()
refresh(duree)
setVitAng(vit)
src.robots.gopigo.Adaptateur_reel
MOTOR_LEFT_RIGHT
angle_parcourA : int
dist_parcourA : int
robot
run : bool
__init__(rob)
getAngleParcouru()
getDistanceA()
getDistanceParcourue()
initialise()
setVitAngA(dps)
setVitAngDA(dps)
setVitAngGA(dps)
Il faut un robot pour que les stratégies soient éxécutées
src.controleur.controleur.Controler
Running : bool
logger : NoneType, RootLogger
strat_en_cour : NoneType
strategie : int
__init__()
lancerStrategie(strat)
mainControleur()
Il faut un contrôleur pour lancer les stratégies
src.environnement.Environnement
dicoObs : dict
last_refresh : int
length
listeObs : list
listeRobots : list
logger : NoneType, RootLogger
robotSelect : int
scale
width
__init__(width, length, scale)
addObstacle(nom, lstPoints)
addRobotSelect(n)
initBorders()
printMatrix()
refreshEnvironnement()
setRobot(robA)
verifCollision(rob)
src.obstacle.Obstacle
lstPoints
nom
__init__(nom, lstPoints)
src.interface3D.interface3D.Balise
dir : list
height
width
x
y
__init__(x, y, width, height)
src.interface3D.interface3D.Interface3D
balise
controleur
env
running : bool
secret : bool
son : NoneType, _Callback, list
__init__(env, controleur)
backCameraTask(task)
binds()
choisirStrategie(strat, distance)
createAllObstacles()
createAllRobots()
createBalise(balise)
createEnvironnement()
createObstacle(obs, height)
createRobot(robotA)
deleteNode(node)
drawLine(pt1, pt2, tabNodesLines)
drawOpti()
frontCameraTask(task)
mystere()
renderToPNM()
setDraw()
spinCameraTask(task)
takePic(task)
ticTac()
upCameraTask(task)
updateBalise()
updateRobot(robotA)
src.interface2D.interface2D.Interface
canv : Canvas
controleur
env
frame_canva : LabelFrame
frame_coordonnees : LabelFrame
frame_dist_obstacle : LabelFrame
frame_gauche : LabelFrame
frame_row1 : LabelFrame
frame_titre : LabelFrame
frame_tutorial : LabelFrame
frame_up : LabelFrame
frame_vitesses : LabelFrame
lab : NoneType
lab_coord_nom : Label
lab_coord_x : Label
lab_coord_y : Label
lab_distance : Label
lab_vitesse : Label
lab_vitesseD : Label
lab_vitesseG : Label
root : Tk
tutorial : Label
tutorial_image : PhotoImage
__init__(env, controleur)
choisirStrategie(strat, distance)
createObs()
createRobotRect(robot)
dessinePoint(pos, couleur)
mainloop()
refreshPositionRobotVisuel(canvas, robot)
rotateRobotRect(canvas, robot, angle)
ticTac()
updateStatsAffichage()
Les interfaces ont besoin d'un environnement pour fonctionner
src.controleur.strategies.StrategieArretMur
distarret
distrob
logger : NoneType, RootLogger
robA
__init__(robAdapt, distarret)
start()
step()
stop()
src.controleur.strategies.StrategieAvancer
distance
logger : NoneType, RootLogger
parcouru : int
robA
__init__(robAdapt, distance)
start()
step()
stop()
src.controleur.strategies.StrategieBoucle
logger : NoneType, RootLogger
nbTours
restants
robA
strat
__init__(robAdapt, strat, nbTours)
start()
step()
stop()
src.controleur.strategies.StrategieCond
cond
logger : NoneType, RootLogger
robA
strat
__init__(robAdapt, strat, cond)
start()
step()
stop()
src.controleur.strategies.StrategieSeq
indice : int
listeStrat
robA
__init__(listeStrat, robAdapt)
start()
step()
stop()
src.controleur.strategies.StrategieTourner
angle
angle_parcouru : int
logger : NoneType, RootLogger
robA
__init__(robAdapt, angle)
start()
step()
stop()