Programming with Python
Part 2
Replacing console interactions by mouse-click events
emerging-mind.org eJournal ISSN 2567-6466
(info@emerging-mind.org)

Gerd Doeben-Henisch
gerd@doeben-henisch.de

October 16, 2017

Contents

Abstract

Taking the proposal from Part 1 for an environment-actor demo and enhance it with replacing all console interactions with mouse-click events.

1 Problem to be Solved

Having a first sketch for a simple environment and a simple input-output system as an actor we want now introduce a graphical systen interface (GSI) of the program for the user.

We take as a starting point the requirements from the concert in Hamburg wich will happen at November-9, 2017 in the evening.

We have the following experimental setups:

1. The *behavior function* ϕ of the actor is 'empty' $\phi = \emptyset$. The actor functions like an 'envelope': you can see the body of the actor on the screen, but his behavior depends completely from the inputs given by a human person.

2. The *behavior function* ϕ of the actor is driven by one, fixed rule $\phi(i) = const$. The actor will do always the same, independent from the environment.

3. The *behavior function* ϕ of the actor is driven by a source of random values; therefore the output is completely random $\phi(i) = rand$.

4. The *behavior function* ϕ of the actor is driven by a source of random value but simultaneously the actor has some simple memory μ remembering the last n steps before finding food. Therefore the behavior is partially random, partially directed depending from the distance to the goal food: $\phi : I \times IS \longmapsto IS \times O$ with internal states IS as a simple memory which can collect data from the last n-many steps before reaching the goal. If the memory μ is not empty then it can happen, that the actual input maps the actual memory-content and then the memory gives the actor a 'direction' for the next steps.

2 How to Program

We will first replace every console input by an interaction with the graphic window.

There are the following console interactions to be replaced:

1. Asking for step-wise continuation during the program run

2. Asking for the position of the actor in the beginning

3. Quitting the whole program at the end

2.1 Continuation with Timer instead of Console Interaction; Quit

It will give more comfort in the usage of the program when the continuation will happen 'by the program itself' and not by asking every time for a continuation. This can easily be implemented with the python-timer function 'sleep(n)' for n-many seconds. The new program looks like the preceding program 'gdh-win10.py' with only a small change here:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 16 08:19:31 2017

@author: gerd doeben-henisch
Extend the program gdh-win10.py by a timer-mechanism for automatic
continuation of the event-loop
"""

# Program: gdh-win11.py

def main():
import graphics as grph        #Graphics objects from Zelle
import environment as env      #Functions to generate grids
import numpy as np             #numerical library numpy
import acctor as acc           #Acctor related functions
import time as tm              #Timer library <----- New
....

     energy=a1.getEnergy()
if energy <0:
print('ATTENTION: Actor ',label,' has no more Energy!!!')
FINISH=-1

#Continuation delayed by the timer function
tm.sleep(2) #python timer , sleep n secs
<----- in the original source is here an indent which marks the end of the while-eve

print('THE GAME SAYS GOODBYE!')
tm.sleep(10)
win.close()
main()
```

The new 'end' of the program run has additionally been changed by taking into account if the actor has died on account of loosing all his energy.

2.2 Inserting an Actor by Mouse-Click

In the following code example we are using the settings from the last main program 'gdh-win10.py' for the generation of a graphics window with a grid.

Then we allow a mouse-click interaction for the position of the new actor. The window coordinates of the mouse-click event will be adjusted to the underlying grid and then the actor will be inserted and shown.

If the actor will be place on the cell of an obstacle the obstacle is deleted by this insertion.

If the actor would coincide with a food then the food is gone.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 16 08:19:31 2017

@author: gerd doeben-henisch
Transforming an example of   Johen Zelle (2002)
for usage in the environment-actor project
"""

# Program: gdh-environment-actor-inserted.py

import graphics as grph
import environment as env
import numpy as np

def main():

#Star a graphics window for the environment
xmax = 700
ymax = 700

win = grph.GraphWin('ENVIRONMENT 1', xmax, ymax)
win.setBackground('HoneyDew')

#State some values for the parameters for a grid in the graphics window

first = 1    #First element of the list
last = 7     # Last element of the list used
distance = 100   #distance of two lines in the grid
maxXcoord = 7  #size of the data array in the background
maxYcoord = 7
nobj = 20    #objects: percentage of the array space !
nfood = 1    #food: percentage of the array space !
```

```
#generating the lines of the grid

env.grid(win,xmax, ymax, first, last, distance)

message = grph.Text(grph.Point(80, 15), "Click on one point")
message.draw(win)

# Fill the grid with spaces and objects
gr2=env.fillgridobj2(win,first, last, maxXcoord,maxYcoord,nobj)

# Fill the grid with food
gr2=env.fillgridfood(win,first, last, maxXcoord,maxYcoord,nfood,gr2)

# Get and draw the coordinate of a point
p1 = win.getMouse()
p1.draw(win)
x=p1.getX()
y=p1.getY()
print('P1 ',p1,(x,y))

#Transform the window coordinates into the array coordinates
xs = int(x/100)
ys = int(y/100)
print('Center Array  ',(xs+1, ys+1))

#neutralize cell on window to delete  a possible obstacle

rect1 = grph.Rectangle(grph.Point(xs*100,ys*100), grph.Point((xs+1)*100,(ys+1)*100))
rect1.setFill('HoneyDew')
rect1.draw(win)

#translating the fillgridob coordinates into the graphics coordinates

gr2[xs,ys]=3

#Transform the array coordinates in new windows coordinates
xwin = (xs*distance)+(distance/2)
ywin = (ys*distance)+(distance/2)

#gnerate a red circle
center = grph.Point(xwin,ywin)
circ = grph.Circle(center, distance/2)
circ.setFill('red')
circ.draw(win)
```

```
#gnerate a name as a lable
name = grph.Text(center, "A1")
name.draw(win)

print('MAP OF ARRAY \n')
print(str(gr2))
print('ATTENTION: Columns represent Rows on screen!\n')

# Wait for another click to exit
message.setText("Click anywhere to quit.")
p=win.getMouse()
win.close()

main()
```

Figure 1 shows you a grid with obstacles and food and then the inserted actor after a click into the grid. A text above reminds you that you have to click if You wants to close the window.

2.3 Putting Things Together

To complete the task of replacing all console inputs we have only to put the two solutions together: inserting an actor by mouse clicks and continue with a timer function. The program finishes then after the fulfillment of given criteria after some seconds automatically.

The following source code is still far from being optimal, but he lets you recognize possible directions for further improvements.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 16 08:19:31 2017

@author: gerd doeben-henisch
Extend the program gdh-win10.py by a timer-mechanism for automatic
continuation of the event-loop
"""

# Program: gdh-win12b.py

def main():
```

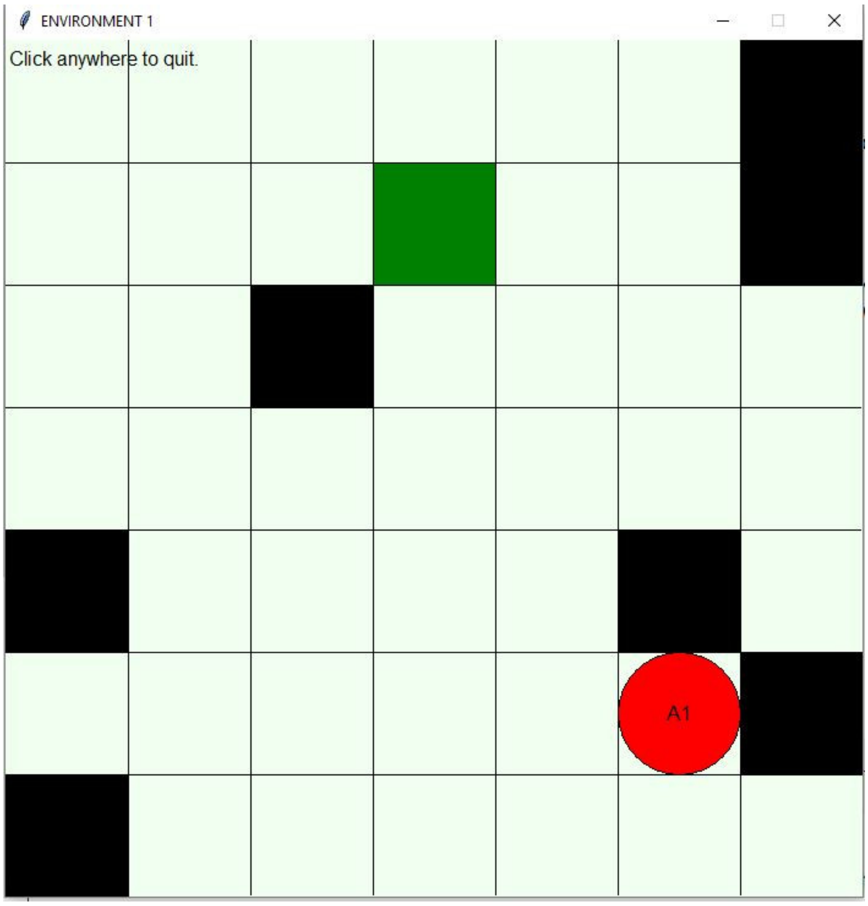

Figure 1: Actor after insertion into a grid with objects and food

```
import graphics as grph        #Graphics objects from Zelle
import environment as env      #Functions to generate grids
import numpy as np             #numerical library numpy
import acctor as acc           #Acctor related functions
import time as tm              #Timer library

##
## SET UP THE ENVIRONMENT
##

e1=env.ENVOBJ('env1',0)

xmax = 700
ymax = 700

win = grph.GraphWin('ENVIRONMENT 1', xmax, ymax)
win.setBackground('HoneyDew')


first = 1    #First element of the list
last = 7     # Last element of the list used
distance = 100   #distance of two lines in the grid
maxXcoord = 7  #size of the data array in the background
maxYcoord = 7
nobj = 20    #objects: percentage of the array space !
nfood = 1    #food: percentage of the array space !

#generating the lines of the grid

env.grid(win,xmax, ymax, first, last, distance)

# Fill the grid with spaces and objects
gr2=env.fillgridobj2(win,first, last, maxXcoord,maxYcoord,nobj)

# Fill the grid with food
gr2=env.fillgridfood(win,first, last, maxXcoord,maxYcoord,nfood,gr2)

# Introducing an actor for the event-loop

time=e1.getClock()  #Setting the time of 'birth'

#Get the array coordinates for the actor
message = grph.Text(grph.Point(80, 15), "Click on one point")
message.draw(win)
```

```
# Get and draw the coordinate of a point
p = win.getMouse()
p.draw(win)
xc=p.getX()
yc=p.getY()
print('P ',p,(xc,yc))

#Transform the window coordinates into the array coordinates
xs = int(xc/100)
ys = int(yc/100)
x = xs+1
y=ys+1
print('Center Array  ',(x,y))

#Insert the actor in the data array
gr2[xs,ys]=3

#Introduce more parameter
dir=1
color = 'red'
name="A1"
energy=2000
energyrate=2

#Generate an instance of the actor class ACTOBJ
a1=acc.ACTOBJ(name,time,x,y,dir,color, energy,energyrate)

a1.setPos(x,y)
outputMessage='yesMove'  #This tells the environment whether an actor
#wants to move
a1.setOutputMessage(outputMessage)

#Realize an actor on the window
env.introduceActor(x,y,dir,name,color,energy,distance,win)

print('MAP OF ARRAY \n')
print(str(gr2))
print('ATTENTION: Columns represent Rows on screen!\n')

##
## START AN EVENT LOOP FOR THE ENVIRONMENT
##
## ASK ACTOR-BEHAVIOR-FUNCTION WHAT TO DO
```

```
## CHECK WETHER ACTIONS WANTED ARE POSSIBLE
## MODIFY THE ENVIRONMENTR IF NECESSARY
## REPEAT AS LONG AS ACTORS ARE ALIVE
##

# condition for while

FINISH=100

while FINISH >0:

# Look to the actor class for new responses

dir=a1.getDir()
print('Direction = ', dir)
p=a1.getPos()
print('Actual Position = ',p)
xold=p[0]
yold=p[1]

if a1.getOutputMessage() == 'yesMove':

# If there are responses then compute the next possible move

Err, xnew, ynew = env.newPosition(dir,xold,yold)
print('Err = ', Err)
print('New Position planned = ', (xnew,ynew))

#Check whether this new position  is possible

if gr2[xnew-1,ynew-1] != 0:
#Telling the actor that the planned new position is occupied
a1.setInputMessage("NoMove")
print('actor =',name,' has not moved')
else:
type=3 #actor
#First change position in data array
env.moveArray(type,xold,yold,xnew,ynew,gr2)
#Tell the actor his new position
a1.setPos(xnew,ynew)
#Next change position in window
env.introduceActor(xnew,ynew,dir,name,color,energy,distance,win)
env.deleteActor(p,win)
else:
```

```
print('No move wanted by actor.')

# Get the actual environment time

t2=e1.getClock()
print('Environment Time =', t2)

#Call the actor

acc.behaviorAc(e1,a1)


# The actor will compute it's response and will update the class

#counting FINISH down for to stop after finitely many steps
FINISH=FINISH-1

envClock = e1.getClock()
print('Actual Environment Time = ',envClock)
envClock= envClock+1
e1.setClock(envClock)
print('New  Environment Time = ',envClock)

energy=a1.getEnergy()
if energy <0:
print('ATTENTION: Actor ',name,' has no more Energy!!!')
FINISH=-1

#Continuation delayed by the timer functionm
tm.sleep(2) #python timer , sleep 5 secs

print('THE GAME SAYS GOODBYE!')
tm.sleep(10)
win.close()
main()
```

In the environment.py file only the function ' introduceActor(x,y,dir,name,color,energy,distance,wir has changed a little bit. This relates to the feature that one has to delete the whole cell before inserting the actor. There could be another quadratic object be on that place before.

```
def introduceActor(x,y,dir,name,color,energy,distance,win):

# Convert array coordinates (x,y) into window coordinates (xwin,ywin)
import numpy as np
import graphics as grph

rect1 = grph.Rectangle(grph.Point(x*100,y*100), grph.Point((x+1)*100,(y+1)*100))
rect1.setFill('HoneyDew')
rect1.draw(win)

#insert actor
xwin = ((x-1)*distance)+(distance/2)
ywin = ((y-1)*distance)+(distance/2)

#gnerate a red circle
center = grph.Point(xwin,ywin)
circ = grph.Circle(center, distance/2)
circ.setFill(color)
circ.draw(win)

#gnerate a name as a lable
label = grph.Text(center, name)
label.draw(win)
```

In the actor class ACTOBJ only the head of the function has changed because there was a doubling of 'name' and 'label' which did have the same reference. Therefore the parameter 'label' has been abandoned.

```
class ACTOBJ:
#The actor object has more propeties as a simple environment object.

def __init__(self,name,time,x,y,dir,color, energy,energyrate):
self.name = name
self.time = time
self.position = (x,y)
self.dir = dir
self.color = 'red'
self.energy = energy
self.energyrate = energyrate

def getName(self):
return self.name
```

```
def setName(self,name):
#This should be a short name different from others; a number
# is enough ...
self.name = name
...
```

References