Manojkumar Ravichandran

SIMULATION OF ZOMBIE APOCALYPSE IN PYTHON

This report is about the simulation of Zombie Apocalypse in Python with geographically separated locations.

Contents

Manojkumar Ravichandran

Objective

To write a model in Python to model a Zombie outbreak with geographically separated populations from the given formula below.

dS/dt = P - BSZ - dS dZ/dt = BSZ + GR - ASZ dR/dt = dS + ASZ - GR

with the following factors:

- S: the number of susceptible victims
- Z: the number of zombies
- R: the number of people "killed"
- P: the population birth rate
- d: the chance of a natural death
- B: the chance the "zombie disease" is transmitted (an alive person becomes a zombie)
- G: the chance a dead person is resurrected into a zombie
- A: the chance a zombie is destroyed

And add two new factors such as

- V: Speed of a zombie.
- v: Speed of a non-infected person.

Assumptions

I. Death of a human

In this model, I assumed that there is no natural cause of death for humans. (i.e.) A human must be killed by a zombie and the human will be turned into a zombie.

II. Death of a Zombie

I assumed in this model that a zombie can be killed only by human. (Based on a lot of zombie movies and TV series, where it can be killed by severed head or by burning the undead) So based on this model, the death of the zombie stops after the extinction of human.

So, based on the above formula and assumptions, I have created a script to execute them.

Manojkumar Ravichandran

Zombie Apocalypse Script

Software Requirements

- Spyder 3

Packages used

- Matplotlib
- Random
- NumPy
- Pylab
- tkinter

Project PyCX

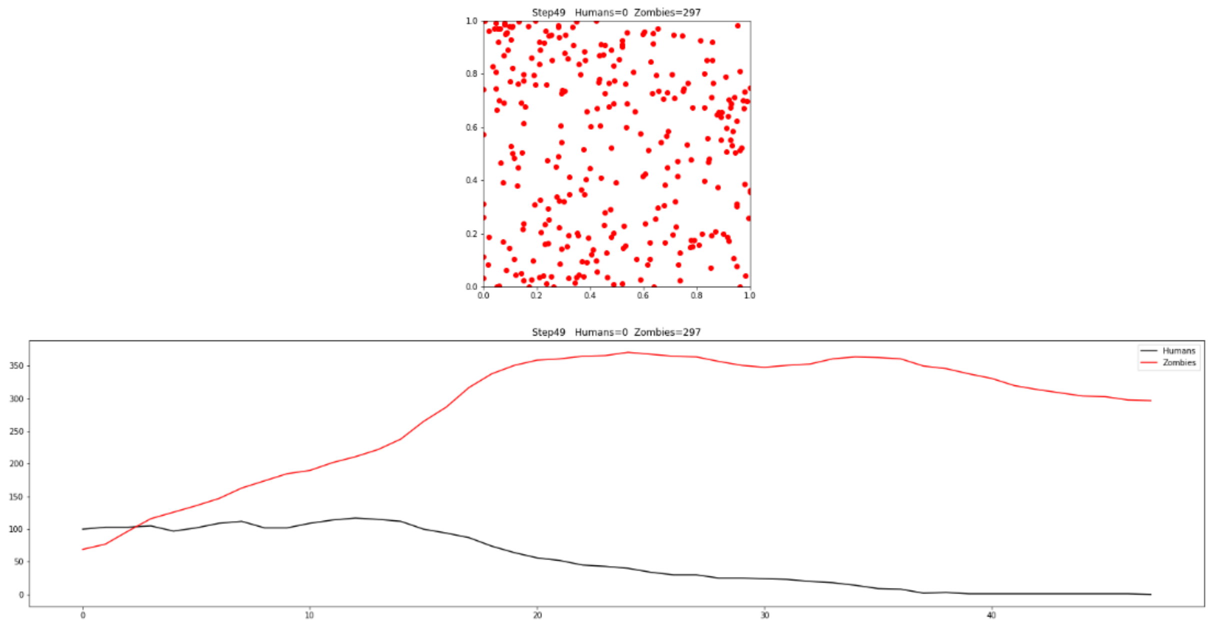
PyCX is an online repository of simple, crude, easy-to-understand sample codes for various complex systems simulation, including iterative maps, cellular automata, dynamical networks and agent-based models. *I used PyCX since it has a lot of factors to run and I can simulate it step by step so I can have a very clear understanding on how the factors will affect each other.*

So, there was a sample code of prey-predator model in Project PyCX which was the start of my script to develop the zombie outbreak. I used the sample code and modified it to script a zombie outbreak in Python.

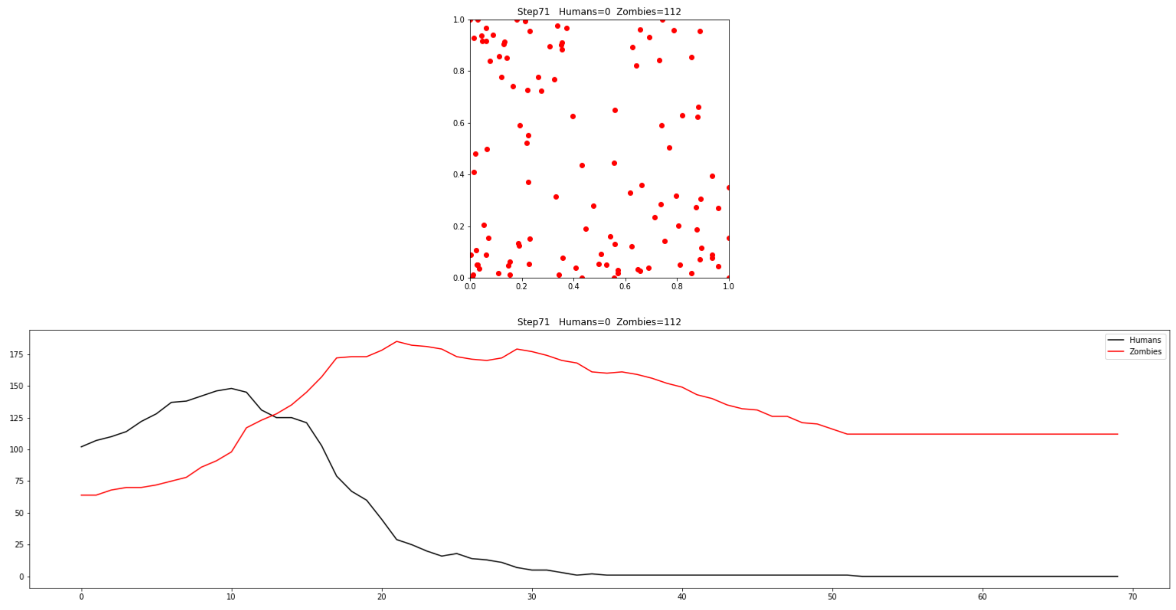
How the factors affect the outbreak

Effect of the human speed

- Considering the human doesn’t the have any vehicles to move faster. I used a human speed of 0.03 whereas zombie’s speed is 0.04 as they can run longer and faster
- I also considered human with vehicles and their speed were set to 1.0



Human Speed = 0.03



Human Speed = 1

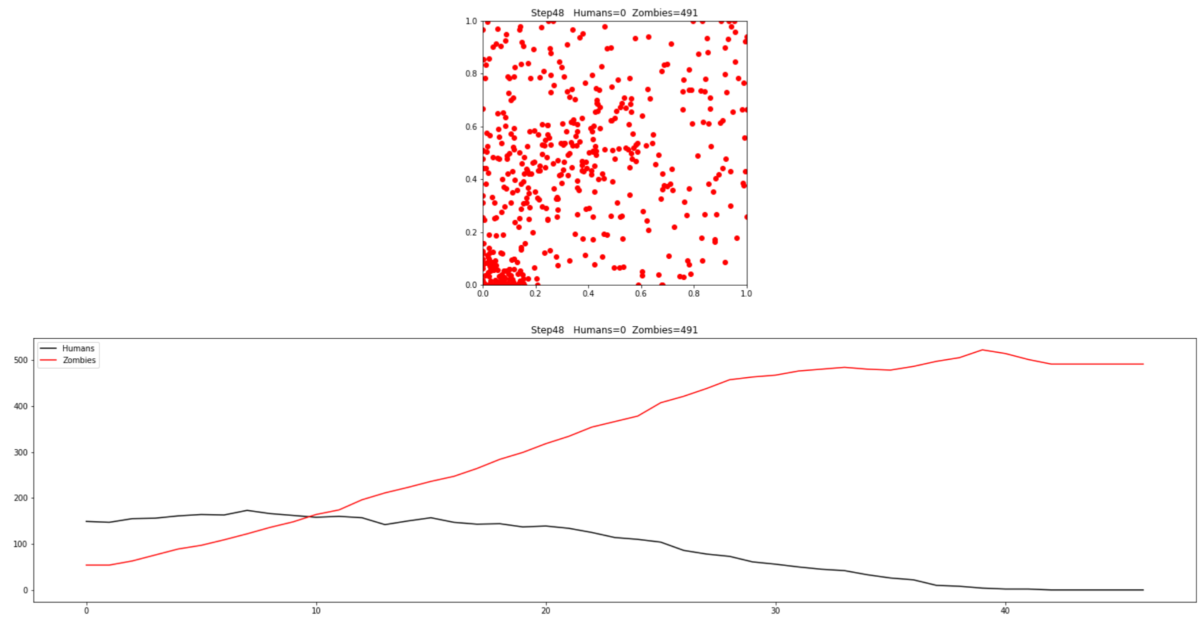
As you can see, human with the speed of 0.03 can only complete 49 steps. Whereas the human with the speed of 1.0 can complete 71 steps. This shows the speed of the humans affects the spread of the infection by a vast margin.

Initial human population

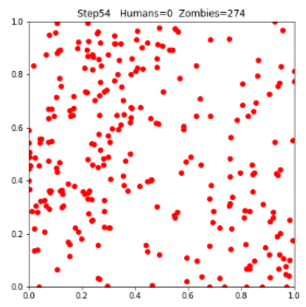
The initial human population also affects the spread of the zombie outbreak.

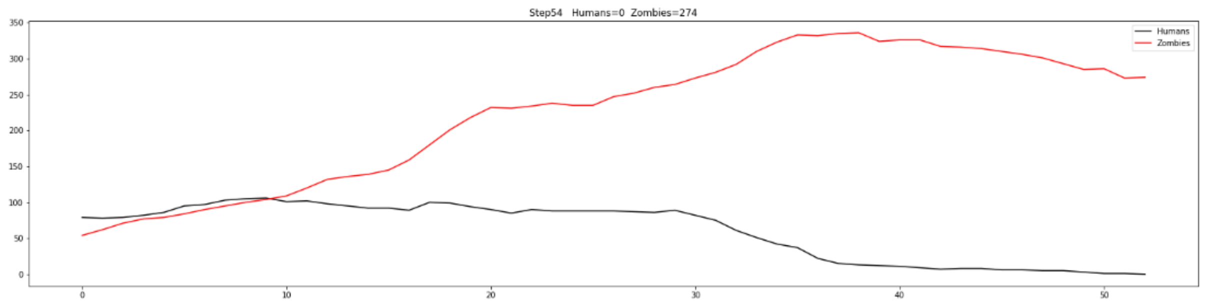
When comparing the results scripts with

- Human population 1.5 times the zombie population
- Human population 3 times the zombie population



Human population 1.5 times the zombie population



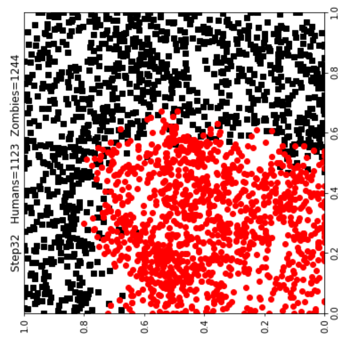


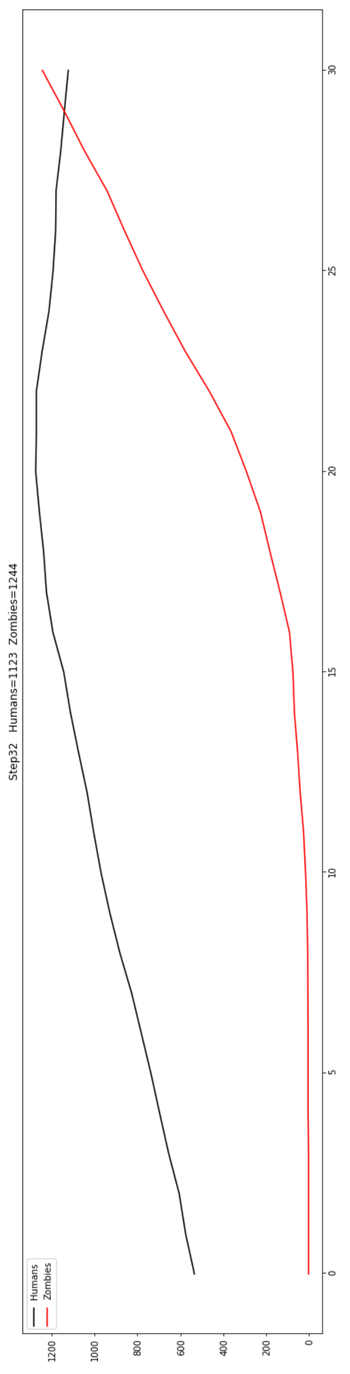
Human population 3 times the zombie population

It can be noticed that irrespective of the no. of human population, the human extinction happens approximately in 50 steps. So, we can learn that the infection rate is similar even if the population is twice the original size.

Simulating a real scenario

To experiment a real outbreak, I created a village of population 500 and an individual zombie. This shows how does a real outbreak might happen.


Step32   Humans=1123   Zombies=1244


Step32   Humans=1123   Zombies=1244
Humans
Zombies

As you can see the results are quite realistic. It shows how the infection spread so gradually. For the first 10 steps, the infection spread was very slow. Just after step 10, it started to gradually increase.

Just after 17 steps, there was a sudden outbreak of infection. The graphs show a sudden spike of zombie spreading.

It was so daunting that the zombie’s population crossed the human population by step 28.

Conclusion

This script shows a realistic simulation on how the zombie infection spreads, how the factors of speed affect the simulation, how an apocalypse takes place.

References

GitHub, 2016. *Data Science Lab PyCX*. [Online]
Available at: https://github.com/data-science-lab/data-science-lab.github.io/wiki/PyCX
[Accessed 27 07 2019].

Max Berggren, 2014. *Model of a zombie outbreak in Sweden, Norway and Finland (Denmark is fine).* [Online]
Available at: http://maxberggren.se/2014/11/27/model-of-a-zombie-outbreak/
[Accessed 28 07 2019].

Python, 2018. *A tiny package for cXense API.* [Online]
Available at: https://pypi.org/project/pyCx/
[Accessed 27 07 2019].

RIebling, J. R. & Schmitz, A., n.d. ZombieApocalypse: Modeling the social dynamics of infection and rejection. *Sage Journals.*

Sayama, H., 2015. *Introduction to the Modeling and Analysis of Complex Systems.* 1 ed. s.l.:SUNY Binghamton.

Sayama, H., 2018. *Project PyCX.* [Online]
Available at: http://pycx.sourceforge.net/
[Accessed 27 07 2019].

Scholarpedia, 2019. *Predator-prey model.* [Online]
Available at: http://www.scholarpedia.org/article/Predator-prey_model
[Accessed 27 07 2019].

Appendix

It includes two scripts – one is for the factors and logic and another is for the simulation using PyCX

PyCX Simulator script

```
#import matplotlib
#matplotlib.use('TkAgg')

import pylab as PL
import tkinter as Tk
from tkinter import ttk


class GUI:

    ## GUI variables
    titleText = 'PyCX Simulator'  # window title
    timeInterval = 0              # refresh time in milliseconds
    running = False
    modelFigure = None
    stepSize = 1
    currentStep = 0

    # Constructor
    def __init__(self, title='PyCX Simulator', interval=0, stepSize=1, parameterSetters=[]):
        self.titleText = title
        self.timeInterval = interval
        self.stepSize = stepSize
        self.parameterSetters = parameterSetters
        self.varEntries = {}
        self.statusStr = ""

        self.initGUI()


    # Initialization
    def initGUI(self):

        #create root window
        self.rootWindow = Tk.Tk()
        self.statusText = Tk.StringVar(value=self.statusStr) # at this point, statusStr = ""
        self.setStatusStr("Simulation not yet started")

        self.rootWindow.wm_title(self.titleText) # titleText = 'PyCX Simulator'
        self.rootWindow.protocol('WM_DELETE_WINDOW', self.quitGUI)
        self.rootWindow.geometry('600x450')
        self.rootWindow.columnconfigure(0, weight=1)
        self.rootWindow.rowconfigure(0, weight=1)

        self.notebook = ttk.Notebook(self.rootWindow)
        # self.notebook.grid(row=0,column=0,padx=2,pady=2,sticky='nswe') # commented out by toshi on
2016-06-21(Tue) 18:30:25
        self.notebook.pack(side=Tk.TOP, padx=2, pady=2)

        self.frameRun = Tk.Frame()
        self.frameSettings = Tk.Frame()
        self.frameParameters = Tk.Frame()
        self.frameInformation = Tk.Frame()

        self.notebook.add(self.frameRun,text="Run")
        self.notebook.add(self.frameSettings,text="Settings")
        self.notebook.add(self.frameParameters,text="Parameters")
        self.notebook.add(self.frameInformation,text="Info")
        self.notebook.pack(expand=Tk.NO, fill=Tk.BOTH, padx=5, pady=5 ,side=Tk.TOP)
        # self.notebook.grid(row=0, column=0, padx=5, pady=5, sticky='nswe')   # commented out by toshi on
2016-06-21(Tue) 18:31:02

        self.status = Tk.Label(self.rootWindow, width=40,height=3, relief=Tk.SUNKEN, bd=1,
textvariable=self.statusText)
        # self.status.grid(row=1,column=0,padx=5,pady=5,sticky='nswe') # commented out by toshi on 2016-
06-21(Tue) 18:31:17
        self.status.pack(side=Tk.TOP, fill=Tk.X, padx=5, pady=5, expand=Tk.NO)

        # -----------------------------------
        # frameRun
        # -----------------------------------
        # buttonRun
        self.runPauseString = Tk.StringVar()
        self.runPauseString.set("Run")
        self.buttonRun =
Tk.Button(self.frameRun,width=30,height=2,textvariable=self.runPauseString,command=self.runEvent)
        self.buttonRun.pack(side=Tk.TOP, padx=5, pady=5)
        self.showHelp(self.buttonRun,"Runs the simulation (or pauses the running simulation)")

        # buttonStep
        self.buttonStep = Tk.Button(self.frameRun,width=30,height=2,text='Step
Once',command=self.stepOnce)
        self.buttonStep.pack(side=Tk.TOP, padx=5, pady=5)
        self.showHelp(self.buttonStep,"Steps the simulation only once")

        # buttonReset
        self.buttonReset = Tk.Button(self.frameRun,width=30,height=2,text='Reset',command=self.resetModel)
        self.buttonReset.pack(side=Tk.TOP, padx=5, pady=5)
        self.showHelp(self.buttonReset,"Resets the simulation")

        # -----------------------------------
        # frameSettings
        # -----------------------------------
        can = Tk.Canvas(self.frameSettings)

        lab = Tk.Label(can, width=25,height=1,text="Step size ", justify=Tk.LEFT, anchor=Tk.W,takefocus=0)
        lab.pack(side='left')

        self.stepScale = Tk.Scale(can,from_=1, to=50,
resolution=1,command=self.changeStepSize,orient=Tk.HORIZONTAL, width=25,length=150)
        self.stepScale.set(self.stepSize)
        self.showHelp(self.stepScale,"Skips model redraw during every [n] simulation steps\nResults in a faster
model run.")
        self.stepScale.pack(side='left')
        can.pack(side='top')

        can = Tk.Canvas(self.frameSettings)
        lab = Tk.Label(can, width=25,height=1,text="Step visualization delay in ms ", justify=Tk.LEFT,
anchor=Tk.W,takefocus=0)
        lab.pack(side='left')
        self.stepDelay = Tk.Scale(can,from_=0, to=max(2000,self.timeInterval),
                              resolution=10,command=self.changeStepDelay,orient=Tk.HORIZONTAL,
width=25,length=150)
        self.stepDelay.set(self.timeInterval)
        self.showHelp(self.stepDelay,"The visualization of each step is delays by the given number of
milliseconds.")
        self.stepDelay.pack(side='left')

        can.pack(side='top')

        # --------------------------------------------
        # frameInformation
        # --------------------------------------------
        scrollInfo = Tk.Scrollbar(self.frameInformation)
        self.textInformation = Tk.Text(self.frameInformation,
width=45,height=13,bg='lightgray',wrap=Tk.WORD,font=("Courier",10))
        scrollInfo.pack(side=Tk.RIGHT, fill=Tk.Y)
        self.textInformation.pack(side=Tk.LEFT,fill=Tk.BOTH,expand=Tk.YES)
        scrollInfo.config(command=self.textInformation.yview)
        self.textInformation.config(yscrollcommand=scrollInfo.set)

        # --------------------------------------------
        # ParameterSetters
        # --------------------------------------------
        for variableSetter in self.parameterSetters:
            can = Tk.Canvas(self.frameParameters)

            lab = Tk.Label(can, width=25,height=1,text=variableSetter.__name__+"
",anchor=Tk.W,takefocus=0)
            lab.pack(side='left')

            ent = Tk.Entry(can, width=11)
            ent.insert(0, str(variableSetter()))

            if variableSetter.__doc__ != None and len(variableSetter.__doc__) > 0:
                self.showHelp(ent,variableSetter.__doc__.strip())

            ent.pack(side='left')

            can.pack(side='top')

            self.varEntries[variableSetter]=ent

        if len(self.parameterSetters) > 0:
            self.buttonSaveParameters = Tk.Button(self.frameParameters,width=50,height=1,
                                          command=self.saveParametersCmd,text="Save parameters to the running
model",state=Tk.DISABLED)
            self.showHelp(self.buttonSaveParameters,
                          "Saves the parameter values.\nNot all values may take effect on a running model\nA model
reset might be required.")
            self.buttonSaveParameters.pack(side='top',padx=5,pady=5)
            self.buttonSaveParametersAndReset = Tk.Button(self.frameParameters,width=50,height=1,
                                                  command=self.saveParametersAndResetCmd,text="Save parameters to the
model and reset the model")
```

```
            self.showHelp(self.buttonSaveParametersAndReset,"Saves the given parameter values and resets the
model")
            self.buttonSaveParametersAndReset.pack(side='top',padx=5,pady=5)
    # <<<<< Init
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>

    def setStatusStr(self,newStatus):
        self.statusStr = newStatus
        self.statusText.set(self.statusStr)


    # model control functions for changing parameters
    def changeStepSize(self,val):
        self.stepSize = int(val)

    def changeStepDelay(self,val):
        self.timeInterval= int(val)

    def saveParametersCmd(self):
        for variableSetter in self.parameterSetters:
            variableSetter(float(self.varEntries[variableSetter].get()))
            self.setStatusStr("New parameter values have been set")

    def saveParametersAndResetCmd(self):
        self.saveParametersCmd()
        self.resetModel()


    # <<<< runEvent >>>>>
    # This event is envoked when "Run" button is clicked.
    def runEvent(self):
        self.running = not self.running
        if self.running:
            self.rootWindow.after(self.timeInterval,self.stepModel)
            self.runPauseString.set("Pause")
            self.buttonStep.configure(state=Tk.DISABLED)
            self.buttonReset.configure(state=Tk.DISABLED)
            if len(self.parameterSetters) > 0:
                self.buttonSaveParameters.configure(state=Tk.NORMAL)
                self.buttonSaveParametersAndReset.configure(state=Tk.DISABLED)
        else:
            self.runPauseString.set("Continue Run")
            self.buttonStep.configure(state=Tk.NORMAL)
            self.buttonReset.configure(state=Tk.NORMAL)
            if len(self.parameterSetters) > 0:
                self.buttonSaveParameters.configure(state=Tk.NORMAL)
                self.buttonSaveParametersAndReset.configure(state=Tk.NORMAL)

    def stepModel(self):
        if self.running:
            self.modelStepFunc()
            self.currentStep += 1
            self.setStatusStr("Step "+str(self.currentStep))
            self.status.configure(foreground='black')
            if (self.currentStep) % self.stepSize == 0:
                self.drawModel()
            self.rootWindow.after(int(self.timeInterval*1.0/self.stepSize),self.stepModel)

    def stepOnce(self):
        self.running = False
        self.runPauseString.set("Continue Run")
        self.modelStepFunc()
        self.currentStep += 1
        self.setStatusStr("Step "+str(self.currentStep))
        self.drawModel()
        if len(self.parameterSetters) > 0:
            self.buttonSaveParameters.configure(state=Tk.NORMAL)

    def resetModel(self):
        self.running = False
        self.runPauseString.set("Run")
        self.modelInitFunc()
        self.currentStep = 0;
        self.setStatusStr("Model has been reset")
        self.drawModel()

    def drawModel(self):
        PL.ion() # bug fix by Alex Hill in 2013
        if self.modelFigure == None or self.modelFigure.canvas.manager.window == None:
            self.modelFigure = PL.figure()
        self.modelDrawFunc()
        self.modelFigure.canvas.manager.window.update()
        PL.show() # bug fix by Hiroki Sayama in 2016

    def start(self,func=[]):
        if len(func)==3:
            self.modelInitFunc = func[0]
            self.modelDrawFunc = func[1]
            self.modelStepFunc = func[2]
            if (self.modelStepFunc.__doc__ != None and len(self.modelStepFunc.__doc__)>0):
                self.showHelp(self.buttonStep,self.modelStepFunc.__doc__.strip())
            if (self.modelInitFunc.__doc__ != None and len(self.modelInitFunc.__doc__)>0):
                self.textInformation.config(state=Tk.NORMAL)
                self.textInformation.delete(1.0, Tk.END)
                self.textInformation.insert(Tk.END, self.modelInitFunc.__doc__.strip())
                self.textInformation.config(state=Tk.DISABLED)

            self.modelInitFunc()
            self.drawModel()
        self.rootWindow.mainloop()

    def quitGUI(self):
        PL.close('all')
        self.rootWindow.quit()
        self.rootWindow.destroy()

    def showHelp(self, widget,text):
        def setText(self):
            self.statusText.set(text)
            self.status.configure(foreground='blue')

        def showHelpLeave(self):
            self.statusText.set(self.statusStr)
            self.status.configure(foreground='black')
        widget.bind("<Enter>", lambda e : setText(self))
        widget.bind("<Leave>", lambda e : showHelpLeave(self))
```

Apocalypse logic and factors script

```
import matplotlib
matplotlib.use('TkAgg')
import random
import copy as cp
import numpy

nhuman = 2000 # Count
HumanPopulation = 500 # initial human population
humanspeed = 0.03 # magnitude of movement of Human
HumanNaturaldeathRate = 1.0 # death rate of Human when it faces Zombies
rr = 0.1 # reproduction rate of Human

ZombiePopulation = 1 # initial Zombie population
zombieSpeed = 0.04 # magnitude of movement of Zombies
zombiedeathrate = 0.025 # chance a zombie is totally destroyed
zombiebirthrate = 1.0 # reproduction rate of Zombies
cd = 0.02 # radius for collision detection
cdsq = cd ** 2
COUNT=1


class agent:
    pass

def initialize():
    global agents, humandata, zombiedata
    agents = []
    humandata = []
    zombiedata = []
    for i in range(HumanPopulation + ZombiePopulation):
        ag = agent()
        ag.type = 'h' if i < HumanPopulation else 'z'
        ag.x = random.random()
        ag.y = random.random()
        agents.append(ag)

def observe():
    global agents, humandata, zombiedata
    global COUNT
    matplotlib.pyplot.subplot(2, 1, 1)
    matplotlib.pyplot.cla()
    Human = [ag for ag in agents if ag.type == 'h']
    if len(Human) > 0:
        x = [ag.x for ag in Human]
        y = [ag.y for ag in Human]
        matplotlib.pyplot.plot(x, y, 'ks')
    Zombies = [ag for ag in agents if ag.type == 'z']
    if len(Zombies) > 0:
        x = [ag.x for ag in Zombies]
        y = [ag.y for ag in Zombies]
        matplotlib.pyplot.plot(x, y, 'ro')

    matplotlib.pyplot.axis('image')
    matplotlib.pyplot.title( "Step"+ str(COUNT) +'   Humans=' + str(sum(p.type == "h" for p in agents)) + '
Zombies=' + str(sum(p.type == "z" for p in agents)))
    matplotlib.pyplot.axis('image')
    matplotlib.pyplot.axis([0, 1, 0, 1])
    matplotlib.pyplot.subplot(2, 1, 2)
    matplotlib.pyplot.cla()
    matplotlib.pyplot.plot(humandata, 'k',label='Humans')
    matplotlib.pyplot.plot(zombiedata,'r', label='Zombies')
    matplotlib.pyplot.legend()
    matplotlib.pyplot.title( "Step"+ str(COUNT) +'   Humans=' + str(sum(p.type == "h" for p in agents)) + '
Zombies=' + str(sum(p.type == "z" for p in agents)))

    COUNT += 1

def update():
    global agents
    suk = 0
    ag = agents[numpy.random.randint(len(agents))]
    for p in agents:
        if p.type == "h":
            suk+=1
    if suk ==0:
        return

    # simulating random movement
    m = humanspeed if ag.type == 'h' else zombieSpeed
    ag.x += random.uniform(-m, m)
    ag.y += random.uniform(-m, m)
    ag.x = 1 if ag.x > 1 else 0 if ag.x < 0 else ag.x
    ag.y = 1 if ag.y > 1 else 0 if ag.y < 0 else ag.y

    # detecting collision and simulating death or birth
    neighbors = [nb for nb in agents if nb.type != ag.type
                 and (ag.x - nb.x)**2 + (ag.y - nb.y)**2 < cdsq]


    if ag.type == 'h':
        if len(neighbors) > 0: # if there are Zombies nearby
            if random.random() < HumanNaturaldeathRate:
                agents.remove(ag)
                return
        if random.random() < rr*(1-sum(1 for x in agents if x.type == 'h')/nhuman):
            agents.append(cp.copy(ag))
    else:
        if len(neighbors) == 0: # if there are no Human nearby
            if random.random() < zombiedeathrate:
                agents.remove(ag)
                return
        else: # if there are Human nearby
```

```
            if random.random() < zombiebirthrate:
                agents.append(cp.copy(ag))

def update_one_unit_time():
    global agents, humandata, zombiedata
    t = 0.
    while t < 1. and len(agents) > 0:
        t += 1. / len(agents)
        update()

    humandata.append(sum(1 for x in agents if x.type == 'h'))
    zombiedata.append(sum(1 for x in agents if x.type == 'z'))

import pycxsimulator
pycxsimulator.GUI().start(func=[initialize, observe, update_one_unit_time])
```