Invaders on Daisyworld: A Computer Simulation of Invasive Species

Jack Hosemans

September 4, 2015

Contents

1 Abstract

Climate change due to invasive species is a very real threat that mankind is constantly trying to fight via border protection and quarantine. Using an agent based implementation of the model "Daisyworld" proposed by Watson and Lovelock[1], this paper attempts to explore this threat to the global climate of a system. On top of the two native types of daisies, two new types of daisies are introduced that are identical besides the temperature that they flourish at. For this paper, the invasive daisies that warm the system prefer a warm climate and vice versa for the daisies that cool the climate. If enough of either type of invasive daisies exist in the system, the homeostatic property of the system should collapse, emulating the "point of no return" in any self regulating system.

2 Introduction

The introduction of new species to an environment can cause a change in the ecosystem that is detrimental to the native species of that system. Constant pressure from factors such as an invasive species can cause the entire system to collapse, altering the local environment drastically. Since expansive ecosystems can affect the global environment, climate change is almost inevitable if a sufficiently large ecosystem gets destroyed.

The goal of this project is to write an agent based computer simulation for the popular model "Daisyworld" and introduce invasive species to the system and study the change that can occur and it's affect on the homeostasis of the system.

In the original Daisyworld[1], there are only two kinds of daisies, black and white, that flourish at the same temperature. These two types of daisies modify their local environment by heating or cooling it through absorbing different amounts of energy depending on their color, or albedo.

The modification that will be implemented is to introduce two new types of daisies on top of the two currently in place. These new daisies are identical in every way bar the temperature that they flourish at. Although this can be set by the end user, the main exploration will be in the case of the invasive daisies flourishing at a temperature that their albedo tends to modify their local temperature towards. I.e: White invasive daisies grow better at a lower temperature, while black invasive daisies grow better in a higher temperature.

3 Method

The simulation will be in the form of an two dimensional agent based simulation that is a stochastic equivalent of the original deterministic model[1]. It will be implemented in the Python programming language utilising the pyQt graphics framework for user interaction.

3.1 Segregation of components

The simulation will be split up into several parts to ease the pain of implementation. They are listed below along with a high level overview of their functions.

- Sun
    - Determines the luminosity that is incident on the world
- Tile
    - Defines a "tile" in the world, that has a temperature and an albedo defined by the existence or nonexistence of a daisy at that point.
    - Handles the life-cycle of held daisies based on the temperature at the tile and age of the daisy
- World
    - Requests that each tile that it contains updates its temperature based on the current incident radiation.
    - Mixes the temperatures of the tiles adjacent to one another
    - Maintains the main loop and updating of the world at each time step
- Daisy
    - Determines the albedo for a tile
    - Has an albedo and an optimal growing temperature.

3.2 Abstract method description

The general program flow is outlined in Figure 1. This implementation is multi-threaded to ensure that user interaction does not interfere with the simulation. Furthermore, since updating is separated from the drawing thread, the simulation speed does not rely on the graphics power of the end user system.

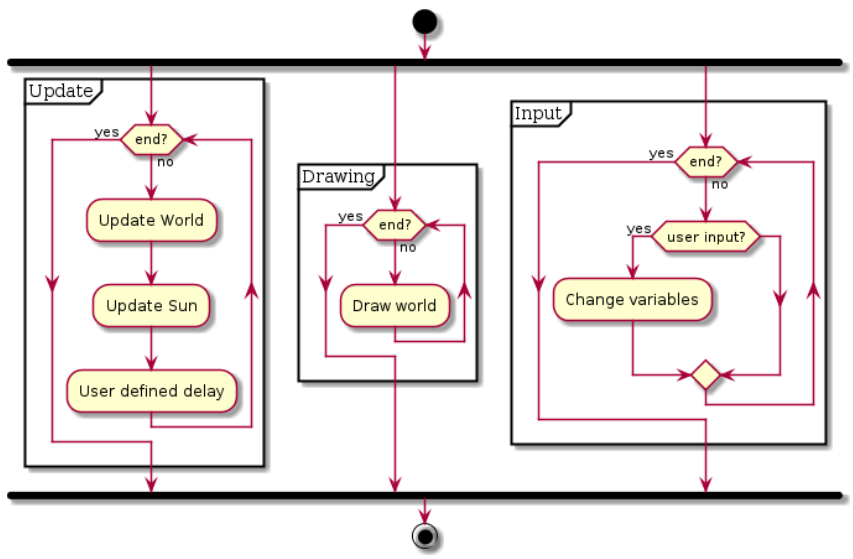


Figure 1: High level overview of the program flow

3.2.1 Updating daisy

If a daisy exists at a tile, the only changes that can occur, from the perspective of the daisy, is it's birth and death.

1. Germination

   The germination of the daisy depends on the difference between the current temperature and the optimal temperature for the daisy in question. The lower the difference, the higher the chance of spawning that particular type of daisy.

   Therefore, the chance B of a daisy growing at some point can be given by:

   $$\text{B} = \begin{cases} 1 - \gamma_1|T|, & \gamma_1|T| > 0 \\ 0, & \text{Otherwise} \end{cases}$$

   Where T is the difference between the "optimal temperature" for a given type of daisy and the temperature of the current tile, and γ_1 is an arbitrary constant that defines the range of which the daisies can spawn over.

   Ideally, the daisies should only have a positive chance to grow in a range of $\pm 17°C$, so γ_1 should be set to $\frac{1}{17}$ to ensure this.

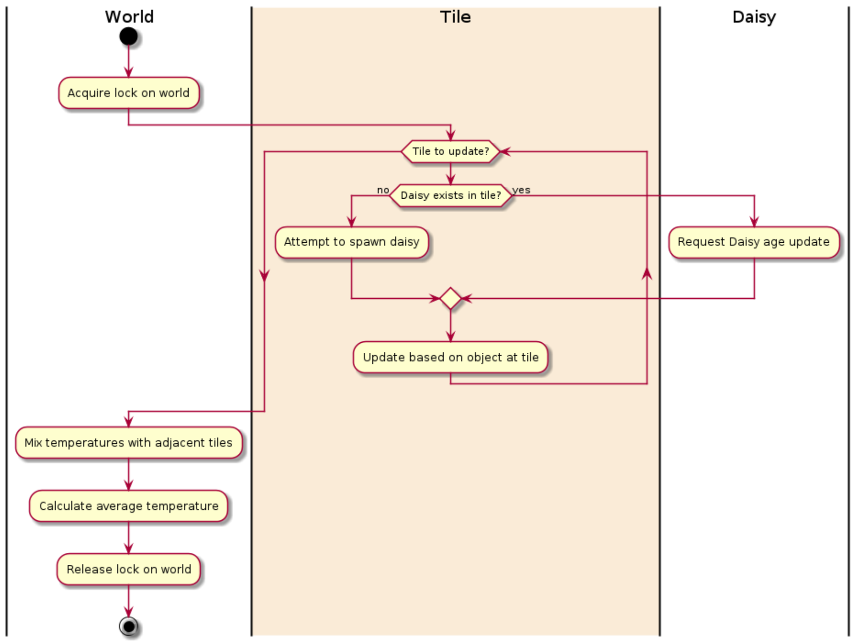


Figure 2: Process of updating the world

2. Type to germinate

   The type of daisy to spawn is chosen by the amount of that type of daisies in adjacent tiles, diagonal inclusive. Note that blank tiles are taken as wildcard daisies, as in they contribute slightly to the chance that any daisy can grow at that point. So, say a tile is surrounded by 3 White daisies, 4 Black Daisies and 1 blank tiles. The chance S_w to spawn a white daisy can be defined by:

$$\mathrm{S}_w = \tfrac{3+W}{8} B_w$$

   Likewise, the chance that a black daisy will grow can be defined by:

$$\mathrm{S}_b = \tfrac{4+W}{8} B_b$$

   Where W is a constant that has yet to be defined for blank tile contribution. Note that this should actually be very small, on the order of 0.01%, to seed the world for the rest of the daisies to grow.

3. Death

   The death of the daisy also depends on the difference between the optimal temperature and the current temperature, but also takes into account the current age of the daisy. This chance is reduced for the change in the temperature, but increases for the age of the daisy. Hence this chance D can be defined by:

   $$\text{D} = \begin{cases} \gamma_2|T|A, & \gamma_2|T|A > 0 \\ 1, & \text{Otherwise} \end{cases}$$

4. Growth

   The growth factor of the daisies will remain unchanged from the original definition[1]. The growth factor was defined as:

   G = 1 - 0.003265T^2

   This change in the growth will be accumulated over time and a daisy will be considered fully grown when this value exceeds (Some value that needs to be defined)

3.2.2 Updating tile

Updating the tile involves two parts, updating the daisy (which has already been outlined) and updating the local temperature.

Updating the temperature of a tile depends on the albedo of the object at that point, be it a type of daisy or bare ground. The higher the albedo, the lower the amount of incident energy to take into account when updating the temperature.

The temperature of the tile after the update can be defined by:

$\text{T}_{t+1} = T_t + 1 - \gamma_3 AR$

3.2.3 Updating global temperature

After updating every tile, the temperatures at every tile have to be mixed. This simulates the natural mixing of temperatures due to wind and other factors.

This is done by taking the average temperature of the tiles adjacent to each tile, exclusive of diagonals, then adding a fraction of that to the current tile. This change in temperature is buffered to ensure correct calculation of the change in temperature at each tile.

T = $\text{T}_{i,j} + \gamma_4(T_{i-1,j} + T_{i,j-1} + T_{i+1,j} + T_{i,j+1})$

4 User interaction

The user of the final simulation should be able to:

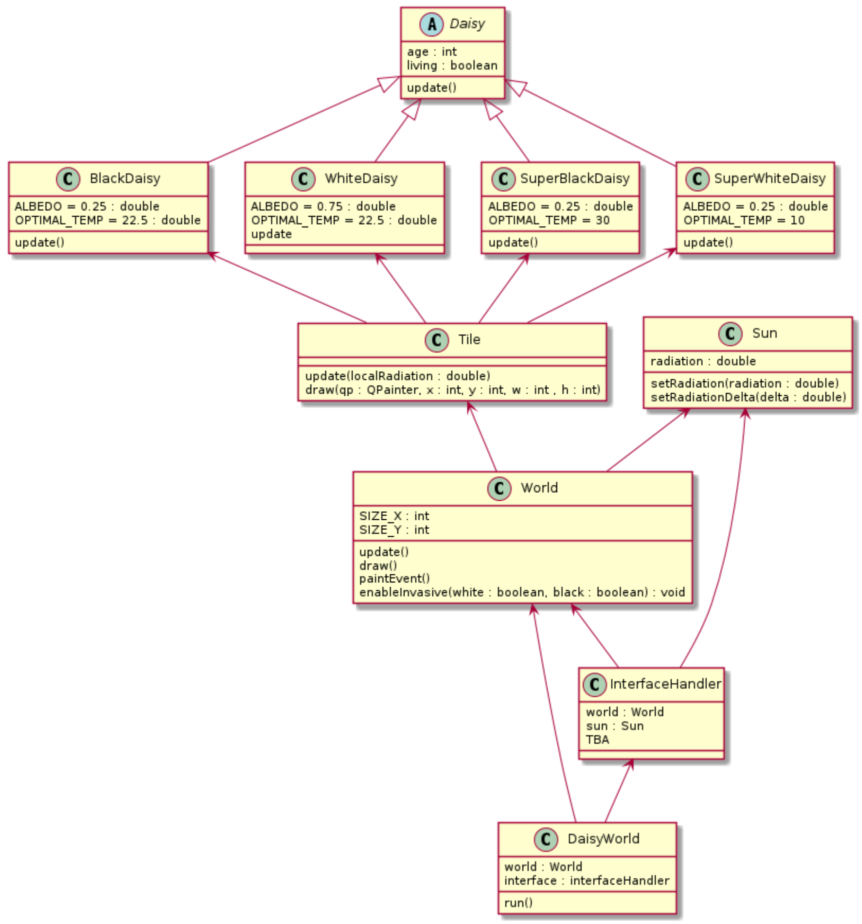
- Run a simulation of Daisyworld
- Modify Daisyworld to include either type of invasive daisies, or both at once
- Set the "Optimal Temperature" for germination of invasive daisies
- Set the luminosity and how it changes over time, if at all
- Set the speed of the simulation
- Output luminosity vs temperature to a file for post-processing
- Display graphical representation of Daisyworld

5 Project plan

5.1 Project Time-line

- Weeks 1 to 4
  - Read relevant documentation on Daisyworld
  - Figure out implementation of agent based models and how it can be applied to simulation
  - Choose modification of model
- Week 5
  - Implement initial prototype of simulation
- Week 6 & 7
  - Implement UI interaction for simulation
  - Increase accuracy of simulation by properly defining equations
  - Modify simulation slightly, with user option to enable/disable invasive properties
- Week 8 onwards
  - Gather data for different parameters
  - Write final report

5.2 Program architecture



6 References

References

[1] Watson, A. J. and J. E. Lovelock, 1983: Biological homeostasis of the global environment: the parable of daisyworld. *Tellus*, **35B**, 284–289.

[2] VON BLOH, W.; BLOCK, A.; SCHELLNHUBER, H. J.. Self-stabilization of the biosphere under global change: a tutorial geophysiological approach. Tellus B, [S.l.], v. 49, n. 3, Dec. 2011.

7 Appendix

This is the current (very basic) implementation of daisyworld. The final program will be much more accurate and possibly more complex.

To run it you need to execute the "World.py" file with Python 3.+ and have the pyQt5 libraries installed on your system.

7.1 World

```
import sys
import threading

from PyQt5.QtWidgets import QWidget, QApplication
from PyQt5.QtGui import QPainter, QColor, QFont
from PyQt5.QtCore import Qt

from Tile import Tile
from Sun import Sun

class World(QWidget):
    SIZE_X = 20
    SIZE_Y = 20
    START_TEMP = 30
    def __init__(self, sun):
super().__init__()

self.sun = sun

# calculate start temp from sun
self.avgTemp = self.START_TEMP
# TODO: variation dependent on the position of the tile?
self.worldTiles = [[Tile(self, World.START_TEMP) \
    for x in range(self.SIZE_X)] \
   for y in range(self.SIZE_Y)]
self.worldLock = threading.Lock()

self.update()

self.setGeometry(300, 300, 280, 170)
self.setWindowTitle('Points')
self.show()


    def update(self):
self.worldLock.acquire()

threading.Timer(0.001,self.update).start()
self.avgTemp = 0
# let the tiles update their temps
for i in range(self.SIZE_X):
    for j in range(self.SIZE_Y):
self.worldTiles[i][j].update(self.sun.radiation)


# TODO: let the temperature of adjacent tiles affect each other
# Cannot do this in place, have to make an array of whatever
# change is required and apply it, multiple times for multiple
# stages?

deltaTempTiles = [[0 for x in range(self.SIZE_X)] \
   for y in range(self.SIZE_Y)]


# let adjacent tiles affect one another
for i in range(self.SIZE_X):
    for j in range(self.SIZE_Y):
deltaTempTiles[i][j] += self.worldTiles[i][j].temp - \
  self.worldTiles[i-1][j].temp
deltaTempTiles[i][j] += self.worldTiles[i][j].temp - \
  self.worldTiles[i][j-1].temp
deltaTempTiles[i][j] += self.worldTiles[i][j].temp - \
self.worldTiles[(i+1)%self.SIZE_X][j].temp
deltaTempTiles[i][j] += self.worldTiles[i][j].temp - \
self.worldTiles[i][(j+1)%self.SIZE_Y].temp
deltaTempTiles[i][j] /= 4 # TODO: fix factor...
deltaTempTiles[i][j] *= 0.1

for i in range(self.SIZE_X):
    for j in range(self.SIZE_Y):
self.worldTiles[i][j].temp -= deltaTempTiles[i][j]
```

```
for i in range(self.SIZE_X):
    for j in range(self.SIZE_Y):
self.avgTemp += self.worldTiles[i][j].temp

self.worldLock.release()
self.avgTemp /= self.SIZE_X*self.SIZE_Y
self.sun.update()
print(str(self.avgTemp) + " , " + str(self.sun.radiation))


    def draw(self, qp):
size = self.size()
incX = size.width()/self.SIZE_X
incY = size.height()/self.SIZE_Y

# let the tiles draw themselves
self.worldLock.acquire()
for i in range(self.SIZE_X):
    for j in range(self.SIZE_Y):
self.worldTiles[i][j].draw(qp,i*incX,j*incY,
   incX+1,incY+1)
self.worldLock.release()


    def paintEvent(self, e):
qp = QPainter()
qp.begin(self)
self.draw(qp)
qp.end()
super().update()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = World(Sun())
    sys.exit(app.exec_())
```

7.2 Sun

```
class Sun:
    def __init__(self):
# radiation value of 1 means that if albedo is 0.5,
# temperature should eventually become 22.5
self.radiation = 1.5

    def update(self):
self.radiation -= 0.001
# TODO: make radiation an actual equation that takes in time
```

7.3 Tile

```
import random
from PyQt5.QtWidgets import QWidget, QApplication
from PyQt5.QtGui import QPainter, QColor, QFont
from PyQt5.QtCore import *

from Daisy import Daisy

class Tile:
    BARE_ALBEDO = 0.5
    def __init__(self, parentWorld, temp):
self.parent = parentWorld
# TODO: Make different types of object that occupy this tile
self.obj = None
self.temp = temp

    def update(self, rad):
if self.obj is not None:
    self.obj.update()
    # obj died, clear
    if self.obj.living is False:
self.obj = None
# else, the tile is unoccupied, attempt to spawn?
else:
    self.spawn()


# update temp
if self.obj is not None:
    self.temp += 0.2*(rad*self.obj.albedo-0.5)
```

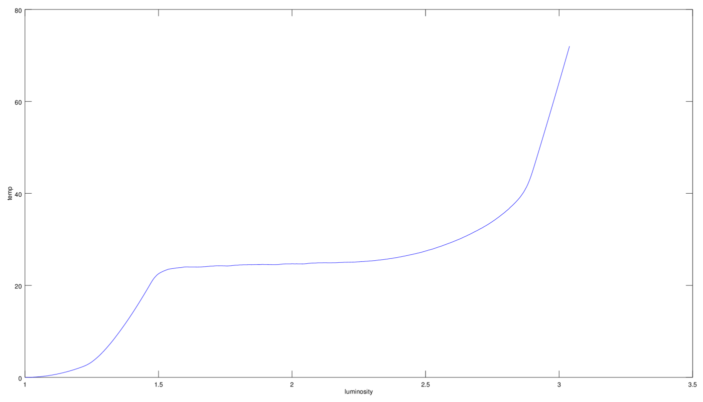
```
else:
    self.temp += 0.2*(rad*self.BARE_ALBEDO-0.5)



    def spawn(self):
chance = random.random()
# TODO: make spawn chance dependent on the daisy type
if chance < 1-abs(22.5-self.temp)/22.5-0.1: # TODO: fix magics
    # TODO: spawn different types of
    # daisy dependent on temperature
    if random.random() > (self.temp-22.5)/5+0.5:
# TODO: make albedo not magic
self.obj = Daisy(self, 0.7)
    else:
self.obj = Daisy(self, 0.2)

    def draw(self, qp, x, y, w, h):
# TODO: change this so it sets color, not albedo
if self.obj is None:
    albedo = self.BARE_ALBEDO
else:
    albedo = self.obj.albedo

# convert albedo from "darkness" to lightness
albedo = 255 - 255*albedo

qp.setBrush(QColor(albedo,albedo,albedo))
# draw in the albedo
qp.drawRect(x,y,w,h)

# overlay temp rect over the top
qp.setBrush(QColor(min(max((self.temp-22.5)*50-255,0),255),
 min(max(255-abs((self.temp-22.5)*50),0),255),
   min(max(128-(self.temp-22.5)*50,0),255),25))


qp.drawRect(x,y,w,h)
qp.setPen(QColor(min(max((self.temp-22.5)*50-255,0),255),
 min(max(255-abs((self.temp-22.5)*50),0),255),
 min(max(128-(self.temp-22.5)*50,0),255)))

# Write the temperature on the tile
qp.drawText(QRectF(QPointF(x,y+h),
   QPointF(x+w,y)),
    Qt.AlignCenter,
    str(round(self.temp,1)))

qp.setPen(Qt.black);
```

7.4 Daisy

```
import random

class Daisy:
    MAX_GROWTH = 100
    # TODO: add optimal temps for types of daisy
    def __init__(self, parentTile, albedo):
self.growth = 0
self.living = True
self.albedo = albedo
self.age = 0

    def grow(self):
self.growth += 1

    def update(self):
"""This should be called at every time interval"""
self.age += 1
# TODO: Growth dependent on tile incident radiation
# TODO: Death dependent on age and temp at tile
# TODO: death dependent on things other than age...
if self.age/self.MAX_GROWTH > random.random():
    self.die()

    def die(self):
self.living = False
```

7.5 Preliminary results

This is a plot of the average temperature over time for the given simulation. Homeostasis is clearly evident in the "flat section" at around $25°C$


temp
luminosity