Coding Document

Elijah Muzzi

Seaweed Competition Simulator

For my final project, I created a agent based model of seaweed competing for space. The seaweed growth is based on the BESeM (Biological Economic Seaweed Model) equations for modeling seaweed weight as a function of days since planting.

$$w_{f,g}(t) = \frac{w_{f,\max}}{1 + \left(\frac{w_{f,\max} - w_{f,0}}{w_{f,0}}\right) * e^{-rgrmax*t}}$$

To take advantage of this, I needed to create an object to represent a seaweed agent. This agent would need to keep track of its name, its initial size, growth rate and maximum size.

$$w_{f,g}(t) = newBiomass$$

$$w_{f,max} = maximumweight$$

$$w_{f,0} = initialsize$$

$$rgrmax = growthrate$$

Seaweed Agent

First, our seaweed agent needs a constructor:

Constructor

```
#imports
import math

class seaweed:


    #constructor
    def __init__(self, name,initialSize, growthRate, maxSize):
        self.name = name
        self.currentSize = initialSize
        self.initialSize = initialSize
        self.growthRate = growthRate
        self.maxSize = maxSize
```

The code above makes a seaweed class and allows us to create an agent by calling the constructor like seaweed('name',initialSize,growthRate,maxSize)

Next we add some helper functions to our class that we will need our seaweed to be able to do during our simulation:

Update Function

```
# update is a function of days since planting in relation to biomass
def update(self,t,x):
     newBiomass = (self.maxSize)/((1)+(((self.maxSize - 
self.initialSize)/self.initialSize)*math.exp(self.growthRate*-1*t)))
     if newBiomass < self.maxSize:
         if newBiomass > x:
             self.currentSize = x
         else:
             self.currentSize = newBiomass
```

The update function is where most of the magic happens, newBiomass is equal to the function for weight based on days since planted. It takes the arguments self so if can access information about the specific agent (this does not need to be passed when calling it), t which is the time in days since planted (or since last grown… more on this later). After we determine the new biomass, we do some quick logical checks to see if A) the seaweed has grown larger than its maximum size in which case it will set it to that maximim size. We also take in the argument x which is the amount of space that the seaweed agent has left to grow in the space. If the difference between the would be new biomass and the current size is greater than the space left to grow then the seaweed agent will only grow the amount of space it has left.

Remove Function

```
 # remove is a function to model what percent of seaweed is removed
 def remove(self,percent):
     newBiomass = self.currentSize*(percent/100)
     self.currentSize = self.currentSize - newBiomass
```

The remove function is used to simulate manual removal of a certain percent of the seaweed population, determined by the parameter 'percent'

RelativeX Function

```
# relativeX lets the agent know how much of X space is left for it to grow
based on how much space the other species is taking up
def relativeX(self,x):
    relativeX = x - self.currentSize
    return relativeX
```

The relvativeX function is used to let the other agent know how much of x space is left to grow into relative to the space that the agent is taking up already.

adjRGR Function

```
 # adjRGR is a function that adjusts the growth rate to reflect a suffciently
large population of collector urchins being present
 def adjRGR(self):
     self.growthRate -= (self.growthRate)*0.24
```

The adjRGR function is applied when the collector urchin mode is selected and it refelcts the change in growth rate that would occur when a suffciently large population of collector urchins are present. This is based on the fact that a suffciently large population of collector urchins will on an annual basis, consume about 24% of seagrass production. (https://en.wikipedia.org/wiki/Tripneustes_gratilla)

getName Function

```
# returns seaweed name
def getName(self):
    return self.name
```

The getName function returns the seaweeds name, this function is used for plotting purposed.

Simulation

First we will define the variable we will be needing in our simulation function

```
def sim(t,mode,size,agents,freq,percent):
    days = 0
    lastState = True    # Keeps track of the last state to know when to
update lastGrown
    x = size            # Total size that both seaweed agents have to share
    xaxis = []  #sets up an array to capture timesteps as we move through the
simulation
    N1Trace = []   # sets up an array to capture the values N1 biomass as we
move through time
    N2Trace = []    # sets up an array to capture the values N2 biomass as we
move through time
    N1 = agents[0]
    N2 = agents[1]
    adjValue = 0    # This will either hold the value of days or lastgrown
depending on the sequence of states
    lastgrown = 0
    firstTime = True   # First time lastgrown is updated this is set to false
    notAdjusted = True # If the growth rates have not yet been adjusted for
the collector urchins, it will adjust them then set this to false
```

Next we will get into the loop and logic of the simulation: (note that all of these blocks of code are in the sim function)

```
 while days < t:
     leftToGrow = ((N1.currentSize + N2.currentSize) < x) # checks at each
```

```
timestep to see if there is room left to grow
     #print(leftToGrow,days,lastgrown)
     if lastState == True and leftToGrow == False:   # Updates lastgrown to
be the day that the plant last was able to grow
         if firstTime == True:   # The first time this happens, lastgrown
will be equal to days-1
             #print("First Time True to False Change")
             lastgrown = (days-1)
             #print(lastgrown)
         else:
             #print("Incrimenting lastgrown...")
             lastgrown += 1
             #print(lastgrown)
     if lastState == True and leftToGrow == True:    # Updates lastgrown when
it grew both this state and the previous state
         #print("Incrimenting lastgrown...")
         lastgrown += 1

     if lastState == False and leftToGrow == True:   # If we could not grow
for a while and then change back to true
         #print((N1.currentSize + N2.currentSize))
         if firstTime == True:
             #print("First Time False to True Change")
             firstTime = False
             lastgrown += 1
             #print(lastgrown)
         adjValue = lastgrown    # While it hasn't grown for a while, the
next time it can grow adjValue should be equal to lastgrown

     if lastState == True and leftToGrow == True:
         #print((N1.currentSize + N2.currentSize))
         if firstTime == True:           # While it hasn't not grown,
adjValue should be equal to days
             adjValue = days

NameError: name 'days' is not defined
```

Next we can get into the code that will impliment the correct population control, manual removal of the seaweed, addition of collector urchins to the space, none or both.

```
if (mode == 'Manual Removal' or mode == 'Removal + Urchin') and days%freq ==
0:
    N2.remove(percent)
if mode == 'Collector Urchin':
    if notAdjusted:
        N1.adjRGR()
        N2.adjRGR()
        notAdjusted = False
if mode == 'Removal + Urchin':
    if notAdjusted:
        N1.adjRGR()
        N2.adjRGR()
        notAdjusted = False

NameError: name 'mode' is not defined
```

The next block of code checks to see if the seaweed has reached it maximum size or has run out of space to grow

```
 if N1.currentSize < N1.maxSize:
     if leftToGrow:
         N1x = N2.relativeX(x)
         N1.update(adjValue,N1x)

 if N2.currentSize < N2.maxSize:
     if leftToGrow:
         N2x = N1.relativeX(x)
         N2.update(adjValue,N2x)

NameError: name 'N1' is not defined
```

The final part of the loop adds the relevant values to their respective lists so we can graph a trace of the biomass over time.

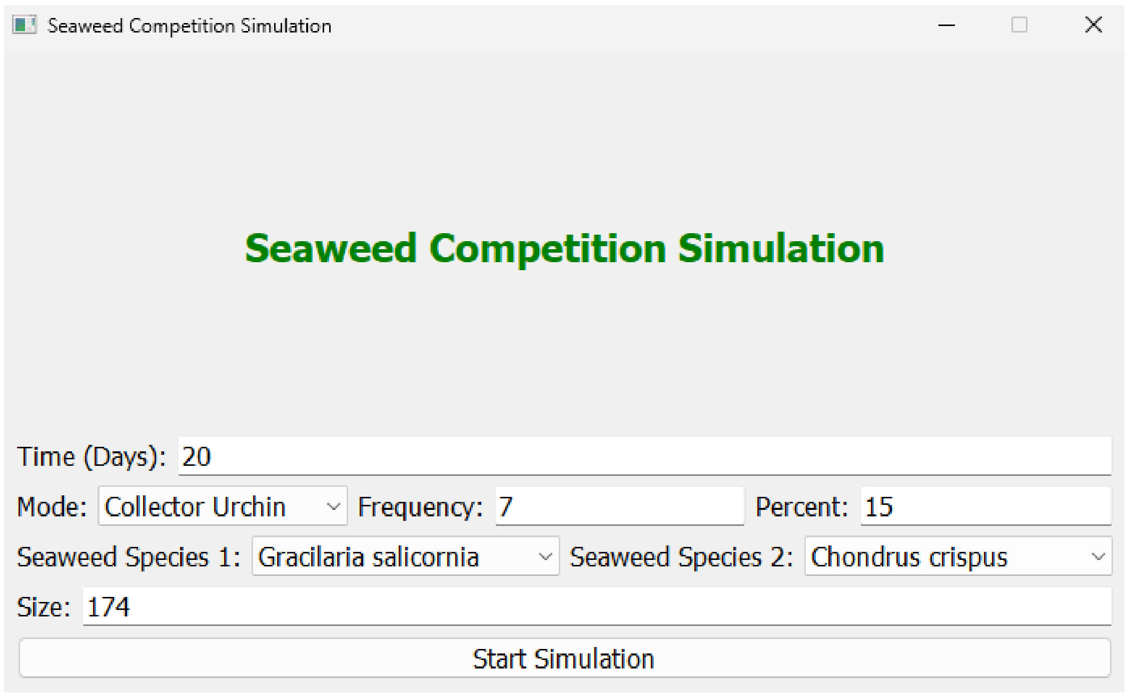
```
    N1Trace.append(N1.currentSize)
    N2Trace.append(N2.currentSize)
    xaxis.append(days)
    lastState = leftToGrow
    days += 1

plot(xaxis,N1Trace,N2Trace,N1,N2)

NameError: name 'N1Trace' is not defined
```

The above block of code makes use of the plot helper function that I wrote

plot function

```
def plot(x,values1, values2,N1,N2):
    plt.style.use('Solarize_Light2')  # Using a predefined style for nicer 
plots
    plt.cla()
    plt.ion() # makes interactive
    plt.plot(x, values1, label=f'{N1.getName()} Biomass')
    plt.plot(x, values2, label=f'{N2.getName()} Biomass')
    plt.legend()
    plt.title("Trace of Seaweed Biomass over Time")
    plt.xlabel('Time (Days)')
    plt.ylabel('Biomass (KG)')
    plt.show()
```

All of this simulation logic is hidden behind a PyQT GUI that takes the parameters as input and makes graphs when a button is clicked


Seaweed Competition Simulation
Seaweed Competition Simulation
Start Simulation

Time (Days): 20
Mode: Collector Urchin
Frequency: 7
Percent: 15
Seaweed Species 1: Gracilaria salicornia
Seaweed Species 2: Chondrus crispus
Size: 174

The Time field takes an integer input to represent the number of days you want the simulation to run for. Mode is a dropdown where you can select the population control method you want to model. Frequency is for if you select an option that involves Manual collection and it will remove the percent field amount of population of Seaweed Species 2 every frequency days. (In this example 15% of the Seaweed 2 population will be removed every 7 days). The Seaweed species 1 and 2 are both dropdowns where you can select the speices of seaweed you want to simulate. Size is a values indicating the maximum amount of biomass allowed in the space.

The code for this GUI is as follows:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel,
QLineEdit, QPushButton, QComboBox, QHBoxLayout

class SeaweedSimulatorGUI(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Seaweed Competition Simulation')
        self.setFixedSize(700, 400)
        layout = QVBoxLayout()

        header = QLabel('<h2 style="color: green; text-align: center; line-
height: 1">Seaweed Competition Simulation</h2>')
        layout.addWidget(header)
```

```
         # Timesteps Input
        timestepsLayout = QHBoxLayout()
        self.timestepsInput = QLineEdit(self)
        timestepsLayout.addWidget(QLabel('Time (Days):'))
        timestepsLayout.addWidget(self.timestepsInput)
        layout.addLayout(timestepsLayout)

        # Mode Dropdown
        modeLayout = QHBoxLayout()
        self.modeDropdown = QComboBox(self)
        self.modeDropdown.addItems(['None', 'Manual Removal', 'Collector
Urchin','Removal + Urchin'])  # Add modes here
        self.frequencyInput = QLineEdit(self)
        self.percentInput = QLineEdit(self)
        modeLayout.addWidget(QLabel('Mode:'))
        modeLayout.addWidget(self.modeDropdown)
        modeLayout.addWidget(QLabel('Frequency:'))
        modeLayout.addWidget(self.frequencyInput)
        modeLayout.addWidget(QLabel('Percent:'))
        modeLayout.addWidget(self.percentInput)
        layout.addLayout(modeLayout)

         # Seaweed Species Dropdowns
        speciesLayout = QHBoxLayout()
        self.species1Dropdown = QComboBox(self)
        self.species1Dropdown.addItems(['Gracilaria salicornia', 'Kappaphycus
alvarezii','Chondrus crispus','Pterocladiella capillacea'])  # Add seaweed
species here
        speciesLayout.addWidget(QLabel('Seaweed Species 1:'))
        speciesLayout.addWidget(self.species1Dropdown)

        self.species2Dropdown = QComboBox(self)
        self.species2Dropdown.addItems(['Gracilaria salicornia', 'Kappaphycus
alvarezii','Chondrus crispus','Pterocladiella capillacea'])  # Add seaweed
species here
        speciesLayout.addWidget(QLabel('Seaweed Species 2:'))
        speciesLayout.addWidget(self.species2Dropdown)

        layout.addLayout(speciesLayout)

       # Size Input
        sizeLayout = QHBoxLayout()
        self.sizeInput = QLineEdit(self)
        sizeLayout.addWidget(QLabel('Size:'))
        sizeLayout.addWidget(self.sizeInput)
        layout.addLayout(sizeLayout)

        # Start Simulation Button
        self.startButton = QPushButton('Start Simulation', self)
        self.startButton.clicked.connect(self.start_simulation)
        layout.addWidget(self.startButton)




        self.setLayout(layout)
    # This is the function that runs when the start simulation button is
pushed.
    def start_simulation(self):
        agents = []
        timesteps = int(self.timestepsInput.text())
        mode = self.modeDropdown.currentText()
        species1 = self.species1Dropdown.currentText()
        species2 = self.species2Dropdown.currentText()
        freq = int(self.frequencyInput.text())
        percent = int(self.percentInput.text())
        size = float(self.sizeInput.text())
        print(f"Starting simulation with: Timesteps: {timesteps}, Mode:
{mode}, Species 1: {species1}, Species 2: {species2}, Size: {size}")
        if species1 == 'Gracilaria salicornia':
            agents.append(seaweed('Gracilaria salicornia',1,0.61,196.57))
        if species2 == 'Gracilaria salicornia':
            agents.append(seaweed('Gracilaria salicornia',1,0.61,196.57))
        if species1 == 'Kappaphycus alvarezii':
            agents.append(seaweed('Kappaphycus alvarezii',1,0.70,190.60))
        if species2 == 'Kappaphycus alvarezii':
            agents.append(seaweed('Kappaphycus alvarezii',1,0.70,190.60))
        if species1 == 'Chondrus crispus':
            agents.append(seaweed('Chondrus crispus',1,1.86,137.00))
        if species2 == 'Chondrus crispus':
            agents.append(seaweed('Chondrus crispus',1,1.86,137.00))
        if species1 == 'Pterocladiella capillacea':
            agents.append(seaweed('Pterocladiella capillacea',1,2.35,40.00))
        if species2 == 'Pterocladiella capillacea':
            agents.append(seaweed('Pterocladiella capillacea',1,2.35,40.00))


        sim(timesteps, mode, size,agents,freq,percent)
```