User Guide: Simulation of Brine Shrimps

Introduction

The intended readership and users for this program of Brine Shrimp Simulation includes end users and different types of system operators. There are two different types of end users. These include professionals or specialists that use the program for research, testing, operational functions, etc. and an example of this would be a Scientist who would use the program for research or on an experiment involving Brine Shrimp. The second type of end user would be a 'casual' user who would use this program for entertainment or occasional purposes, for an example a teacher would be able to use this program for teaching purposes. The other type of user for this program is the different types of system operators which can include game developers, data entry people and system installers.

Purpose

The purpose of the Simulation of Brine Shrimp program is to simulate Brine Shrimps showing their different stages in the lifecycle, different movement and how they would interact in the tank with the user's input for the parameters. This program will allow users to input varying parameters for the tank size, how many days the age steps, how many days the time step shows, the initial Brine Shrimp population, the death rate of the Brine Shrimps and the reproduction rate. The purpose of this user guide for this Brine Shrimp Simulation program is to help users to understand the features, functions and code of this program and how they would use it to simulate Brine Shrimps with their parameters. This program implements graphs showing the simulation of the Brine Shrimps and how they interact inside of the tank. Therefore users will be able to conduct their own experiments using these functions and methods.

Features

The Brine Shrimp Simulation program features two files, where shrimp.py is the class object containing different functions such as __init__, __str__, getAge, getSize, getColor, checkWall, getMovement, stepChange, checkCollisions and checkPos. The other file is the shrimpSimBase.py which also contains different functions such as checkRate, checkChance, checkValid and the main function that plots the data and therefore simulates the Brine Shrimp inside of the tank. The shrimp.py file is used and imported in shrimpSimBase.py where the functions and properties are called into the main function. Therefore this program is object orientated as it uses functions and methods with different attribute and properties to be imported into another file which uses these attributes and properties. This program allows the users to input their own parameters through command line arguments. Users are able to set the parameters for the tank size, population, age step, time step, death rate and reproduction rate. Through command line arguments users are able to input the numbers straight away with error handling and exception handling functions which prevent the program from crashing if users input strings, floats or negative integers. Once input parameters are successful each time step is shown through a graph where the dots represent the Brine Shrimp and the tank is represented by the XMAX and YMAX. There are different stages in the lifecycle of Brine Shrimp and this program helps users to differentiate between the states, ages and position of each Brine Shrimp:

- Egg (Age = 0 – 1 , ·)
- Hatchling (Age = 2 – 7, ●)
- Juvenile (Age = 8 – 21, ●)
- Adult (Age = 22 – 28, ●)
- Dead (Age >= 29, ×)

How To Use

To start the program users will need to be in terminal, in the correct directory where the files (shrimp.py and shrimpSimBase.py) are located and users will be using python3 in order to run the Brine Shrimp program. This program requires users to input the parameters by command line arguments and then the program will simulate shrimps showing different colours, shapes, sizes and positions depending on these different factors. The parameters that users input include:

- Tank size (XMAX = int(sys.argv[1]), YMAX = int(sys.argv[2]))
- Initial Brine Shrimp population (population = int(sys.argv[3]))
- Age step to increase the age by a certain number each time (agestep = int(sys.argv[4]))
- Time step to show users the amount of days or graphs of the increase in age step (timestep = int(sys.argv[5]))
- Death rate percentage (deathrate = int(sys.argv[6]))
- Reproduction rate percentage (reproducerate = int(sys.argv[7]))

An example of the command line argument that users input is demonstrated below:

```
python3 shrimpSimBase.py 500 500 50 5 5 10 50
```

In this example after python3 the index[0] is the program name, index[1] is the tank size XMAX = 500, index[2] is the tank size YMAX = 500, index[3] is the initial Brine Shrimp population = 50, index[4] is the age step = 5, index[5] is the time step = 5, index[6] is the death rate = 10% and index[7] is the reproduction rate = 50%. The program will then print out:

```
Tank size: ( 500 , 500 )

Shrimp Population:  50

Age Step:  5 Day(s)

Time Step:  5 Day(s)

Death Rate is:  10 %

Reproduction Rate:  50 %
```

With these input parameters there is also a chance of the program crashing if the wrong kind of input were to be inserted. Therefore this program has functions implemented for error and exception handling. Here are some examples of how the program will print an error message letting users know why and how it will close the program instead of crashing the program:

- If users were to input only 6 parameters instead of the 7:

```
python3 shrimpSimBase.py 500 500 50 5 5 10
Not enough arguments...
```

- If users were to input a negative integer as all parameters need to be > 0

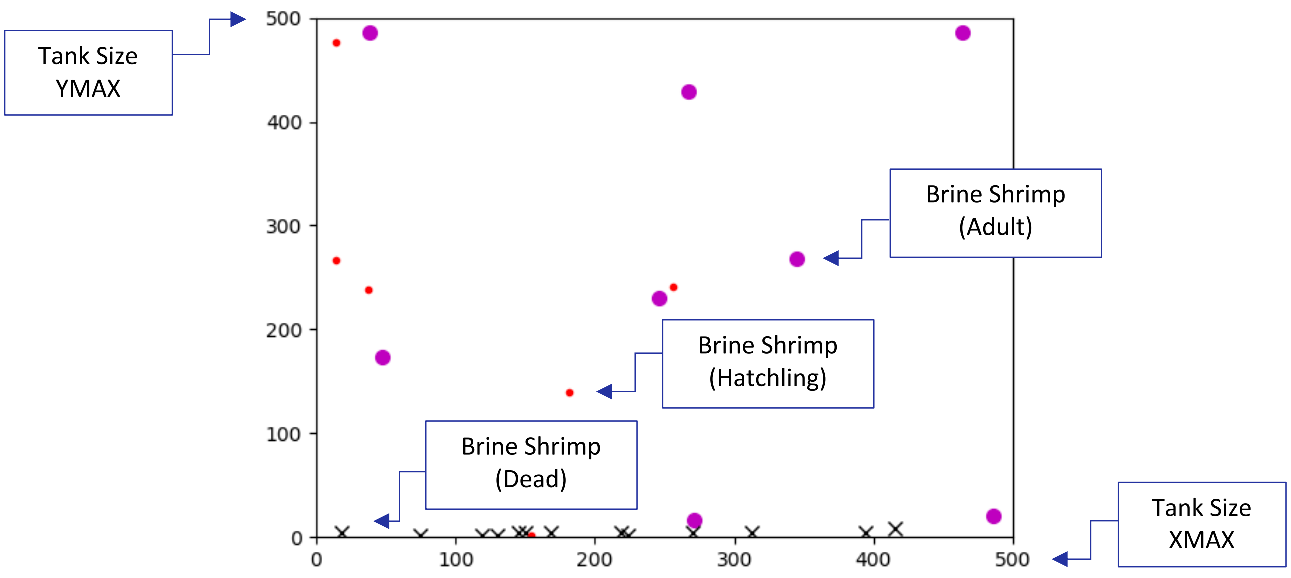
```
python3 shrimpSimBase.py 500 500 50 5 5 10 -50
Arguments must be a positive integer or percentage less
than 100%...
```

- If users were to input a string or float instead of an integer:

```
python3 shrimpSimBase.py 500 500 50 5 5 10 a
Arguments must be an integer...

python3 shrimpSimBase.py 500 500 50 5 5 10 5.0
Arguments must be an integer...
```

If the user's input parameters are successful then the program will plot the Brine Shrimps on the graph for each time step chosen and print it's state and it's position, as shown below:



```
### TIMESTEP  4 ###
adult @ [47, 174]
dead @ [130, 1]
dead @ [75, 1]
dead @ [313, 5]
adult @ [464, 486]
dead @ [394, 5]
adult @ [246, 230]
dead @ [150, 5]
adult @ [38, 486]
dead @ [415, 9]
dead @ [219, 5]
dead @ [168, 5]
adult @ [345, 268]
adult @ [486, 20]
adult @ [271, 17]
dead @ [18, 5]
dead @ [119, 1]
dead @ [224, 1]
dead @ [270, 5]
adult @ [267, 429]
hatchling @ [256, 241]
hatchling @ [154, 2]
hatchling @ [14, 476]
hatchling @ [14, 267]
hatchling @ [37, 239]
dead @ [145, 5]
hatchling @ [181, 140]
```

A caution for this Brine Shrimp program is that the plotted shrimps may seem out of the tank (graph boundaries) however the position of the shrimp (coordinates) never actually exceeds the boundaries, it is only because of its size that makes it appear like the Brine

Shrimps are out of the tank walls. Another caution is that the user's input for the parameters are required to be in the right order every time it is set (first XMAX, YMAX, population, age step, time step, death rate and then reproduction rate). If parameters are put in the wrong order then the program won't be able to detect it and therefore the user's parameters will be used for the wrong ones from intended.

Understanding The Code

shrimp.py:

```
def __init__(self, pos, XMAX, YMAX, agestep):
    self.pos = pos
    self.state = self.states[0]
    self.age = 0
    self.XMAX = XMAX
    self.YMAX = YMAX
    self.direction = self.directions[random.randint(0,3)]
    self.agestep = agestep
```

Function initialises shrimps and contains the necessary parameters that go into the main function as all shrimp objects require a position, tank size and agestep (pos, XMAX, YMAX, agestep). This also initialises the state of the shrimp to eggs, the age to 0 and to have a random direction between the four choices (upmax, downmax, leftmax and rightmax).

```
def __str__(self):
    return self.state + " @ " + str(self.pos)
```

Function that returns the state of the Brine Shrimps plotted and at which position they are in the tank (coordinates).

```
def getAge(self):
    if (self.age <= 1 and self.age >= 0) and self.state != "dead":
        self.state = "egg"
    elif (self.age <= 7 and self.age > 1) and self.state != "dead":
        self.state = "hatchling"
    elif (self.age <= 21 and self.age > 7) and self.state != "dead":
        self.state = "juvenile"
    elif (self.age <= 28 and self.age > 21) and self.state !=
    “dead”:
        self.state = "adult"
    else:
        if self.age >= 29:
            self.state = "dead"
```

Function that checks the age of shrimps and if they are not dead and then sets their state depending on their age. It also checks if its dead first because it must be alive to change its state as it doesn't make sense to go from dead to adult.

```
def getSize(self):
    if self.state == "egg":
        size = 1
    elif self.state == "hatchling":
        size = 3
    elif self.state == "juvenile":
        size = 5
    elif self.state == "adult":
        size = 7
    else:
        size = 7
    return size
```
Function that checks the state of the Brine Shrimps and sets their size according to their state and then returns the results to the caller in the main function.

```
def getColor(self):
    if self.state == "egg":
        color = 'yo'
    elif self.state == "hatchling":
        color = 'ro'
    elif self.state == "juvenile":
        color = 'bo'
    elif self.state == "adult":
        color = 'mo'
    else:
        color = 'kx'
    return color
```
Function that checks the state of shrimps and sets their colour depending on their state. the letters represent colours and shapes of the shrimps, where ‘o’ is for dots and ‘x’ is for crosses. The results are then returned to the caller in the main function.

```
def checkWall(self):
    if self.pos[1] >= self.YMAX:
        self.pos[1] = self.YMAX - 15
        self.direction = "downmax"
    elif self.pos[1] <= 0:
        self.pos[1] = 15
        self.direction = "upmax"
    elif self.pos[0] >= self.XMAX:
        self.pos[0] = self.XMAX - 15
        self.direction = "leftmax"
    elif self.pos[0] <= 0:
        self.pos[0] = 15
        self.direction = "rightmax"
```
Function that checks if shrimps swim into the tank walls (the position is equal to the boundaries) and then sets them at a position near the wall and changes its direction to go the opposite way.

```
def getMovement(self, xspeed, yspeed):
    if self.direction == "downmax":
        self.pos[1] -= yspeed
    elif self.direction == "upmax":
        self.pos[1] += yspeed
    elif self.direction == "leftmax":
        self.pos[0] -= xspeed
    elif self.direction == "rightmax":
         self.pos[0] += xspeed
    self.checkWall()
```

Function that checks the direction of the shrimps and then sets their movement with the matching speed. For an example “downmax” is -=yspeed as its going downwards and “rightmax” is +=xspeed as its going to the right. It then calls the checkWall function to check for collisions with the tank walls and to handle the collisions.

```
def stepChange(self):
    self.age += self.agestep
    self.getAge()
    if self.state == "hatchling":
        xspeed = round(self.XMAX/random.randint(8,10))
        yspeed = round(self.YMAX/random.randint(8,10))
        self.getMovement(xspeed, yspeed)
    elif self.state == "juvenile":
        xspeed = round(self.XMAX/random.randint(5,8))
        yspeed = round(self.YMAX/random.randint(5,8))
        self.getMovement(xspeed, yspeed)
    elif self.state == "adult":
        xspeed = round(self.XMAX/random.randint(2,5))
        yspeed = round(self.YMAX/random.randint(2,5))
        self.getMovement(xspeed, yspeed)
    elif self.state == "dead":
        self.pos[1] = 5
```

Function that checks the state and age of shrimps (by calling getAge function), then increases their age by the user’s input for age step. It then sets their speed in each direction for each state. It also calls the getMovement function to get their movement and direction, to then check for collisions with the tank walls. It also sets the dead state to be near the bottom of the tank at all times.

```
def checkCollisions(self, m):
    if self.direction == "upmax" and m.direction == "downmax
        self.pos[1] -= 10
        m.pos[1] += 10
    elif self.direction == "downmax" and m.direction == "upmax":
        self.pos[1] += 10
        m.pos[1] -= 10
    elif self.direction == "leftmax" and m.direction == "rightmax":
        self.pos[0] += 10
        m.pos[0] -= 10
    elif self.direction == "rightmax" and m.direction == "leftmax":
        self.pos[0] -= 10
        m.pos[0] += 10
```

Function that checks if shrimps are colliding with opposite directions and then changes their positions so they don’t collide.

```
def checkPos(self, m):
    x = m.pos[0]
    y = m.pos[1]
    if self.pos[0] == x and self.pos[1] == y:
        self.checkCollisions(m)
        m.getMovement(10,10)
        self.getMovement(10,10)
```

Function that checks if shrimps are colliding at the same positions and then calls on other functions to handle the collisions. Calls on checkCollisions function to change their position according to which direction they are colliding with. Calls on getMovement function for m and self to change their xspeed and yspeed and therefore their positions.

shrimpSimBase.py:

```
def checkRate(m, deathrate):
    if (random.randint(0,100)) <= deathrate:
        m.state = "dead"
```

Function that sets the user's death rate input to be a percentage and changes the state of the shrimps to dead depending on the probability.

```
def checkChance(m, XMAX, YMAX, agestep, reproducerate):
    newegg = None
    if ((random.randint(0,100)) <= reproducerate) and m.state ==
    "adult":
        randX = np.random.randint(0,XMAX)
        randY = np.random.randint(0,YMAX)
        newegg = Shrimp([randX,randY], XMAX, YMAX, agestep)
    return newegg
```

Function that sets the 'newegg' to none and sets the reproduction rate to the user's input to be a percentage. It also checks if the state of the shrimps are adults and if so then newegg is added (new shrimp object with random positions) depending on the probability. It then returns the result to the caller in the main function.

```
def checkValid(XMAX, YMAX, population, agestep, timestep, deathrate,
reproducerate):
    valid = False
    if (XMAX > 0) and (YMAX > 0) and (population > 0) and (agestep >
    0) and (timestep > 0) and (deathrate > 0) and (deathrate < 100)
    and (reproducerate > 0) and (reproducerate < 100):
        valid = True
    return valid
```

Function that checks if the user's input values are valid for use. Sets the valid to 'False' and checks if it's a positive integers and if death rate and reproduction rate are less than 100. If these statements are true then valid is changed to 'True' and the result is returned to the caller in the main function.

```
def main():
    monkeys = []
    try:
        if len(sys.argv) == 8:
            XMAX = int(sys.argv[1])
            YMAX = int(sys.argv[2])
            population = int(sys.argv[3])
            agestep = int(sys.argv[4])
            timestep = int(sys.argv[5])
            deathrate = int(sys.argv[6])
            reproducerate = int(sys.argv[7])
            if checkValid(XMAX, YMAX, population, agestep, timestep,
            deathrate, reproducerate) == True:
                 print('\nTank size: (',XMAX, ',',YMAX, ')')
                 print('\nShrimp Population: ', population)
                 print('\nAge Step: ', agestep, 'Day(s)')
                 print('\nTime Step: ', timestep, 'Day(s)')
                 print('\nDeath Rate is: ', deathrate, '%')
                 print('\nReproduction Rate: ', reproducerate, '%')
                 for i in range(population):
                     randX = np.random.randint(0,XMAX)
                     randY = np.random.randint(0,YMAX)
                     monkeys.append(Shrimp([randX,randY], XMAX,
                     YMAX, agestep))

                 for i in range(timestep):
                     print("\n ### TIMESTEP ", i, "###")
                     for m in monkeys:
                          checkRate(m, deathrate)
                          m.stepChange()
                          newegg = checkChance(m, XMAX, YMAX,
                          agestep, reproducerate)
                          if newegg is not None:
                              monkeys.append(newegg)
                          for m2 in monkeys:
                              m.checkPos(m2)
                          print(m.__str__())
                          plt.plot(m.pos[0], m.pos[1], m.getColor(),
                          markersize=m.getSize())
                          plt.xlim(0,XMAX)
                          plt.ylim(0,YMAX)
                      plt.show()
            else:
                print('Arguments must be a positive integer or
                percentage less than 100%...')
        else:
            print('Not enough arguments...')
    except ValueError:
        print('Arguments must be an integer...')
```

The main function puts the plotted shrimps in the 'monkeys' list. It then has an exception handling function which inputs the user's parameters at the index shown next to the variable. The function then calls on the checkValid function to check if the inputs are true and if so then the function prints the parameters for the user to see. Next is a for loop for the population with random positions (population that the user chooses) and plots these shrimps with the same parameters as def __init__ (pos, XMAX, YMAX, agestep). The other for loop is for the time step (amount of graphs shown) and inside this loop it calls on the function checkRate to input the user's death rate and also calls on the function stepChange to add the user's choice of age step and to change the shrimps' movements according to their states and ages. The function checkChance is then called to input the user's choice of reproduction rate and checks if newegg isn't equal to 'None' then it adds the new eggs with the same parameters. Next the checkPos function is called to check the position, direction and collisions for the shrimps and then handles the collisions. The main function then prints the states of the shrimps at it's positions at each time step. The plot code plots the shrimps' positions as well as calling the getColor and getSize functions to set their colours and sizes according to their states. The end of the main function then handles errors and exceptions by using ‘except ValueError’ and therefore prevents the program from crashing as mentioned in the features of this program.