Impact Crater Saturation Simulation

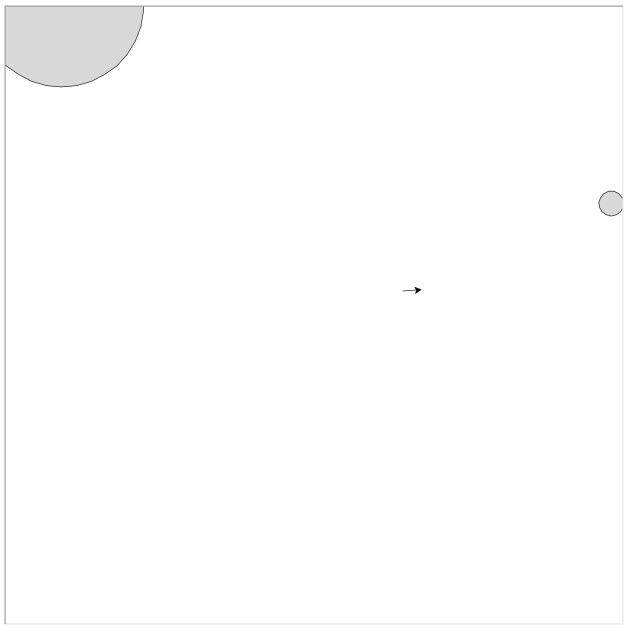
1. Initial Simulation

A. Assumptions I have made in the coding of the simulation are as follows:

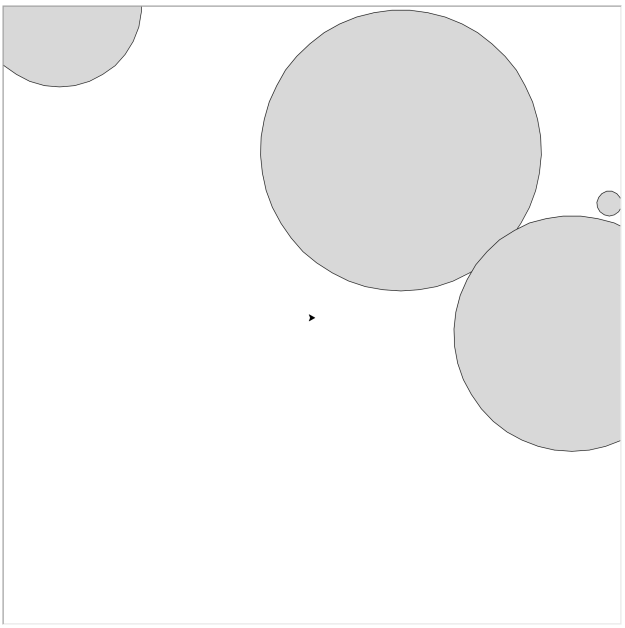
- As mentioned in the outline, one impact crater is formed every 1000 years.
- Simulated craters have a radius larger than 10 km. (outline)
- Simulated craters can have a radius as large as 150 km, the largest known on Earth.
- Simulated craters are perfect circles and their depths are not a factor in their annihilation of one another.
- An old crater will be erased if its epicenter lies inside the confines of a new crater and the radius of the old crater is smaller than that of the new crater.
- A surface is “saturated” with craters when the total area of visible the impact craters reaches 80% of the planets surface.
- There is a linear relationship between the size of impact craters and their frequency, as opposed to the logarithmic relationship observed in real life. I was unfortunately unable to figure out how to distribute the random number generation accordingly. In other words, the probability of an impactor creating a 150 km in radius crater is equal to that of an impactor creating a 10 km radius, which is far from what is observed in the real world.

B. Screenshots of the simulation in progress at arbitrary points in time are included below.

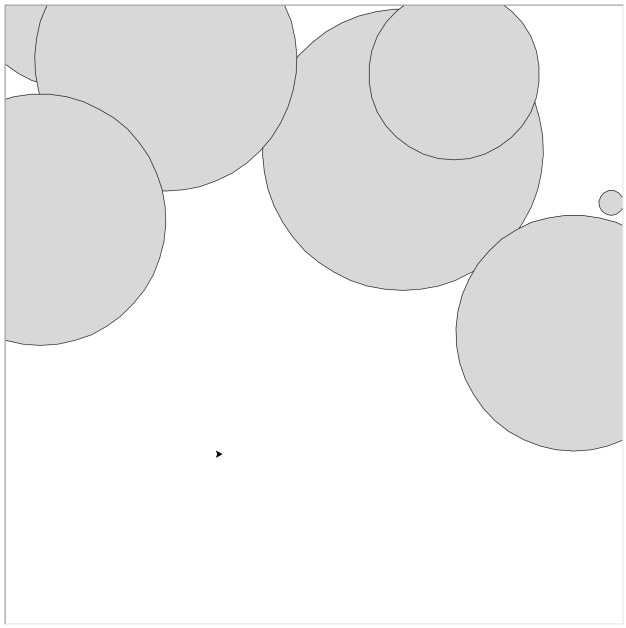
Picture 1: 2 visible impact craters



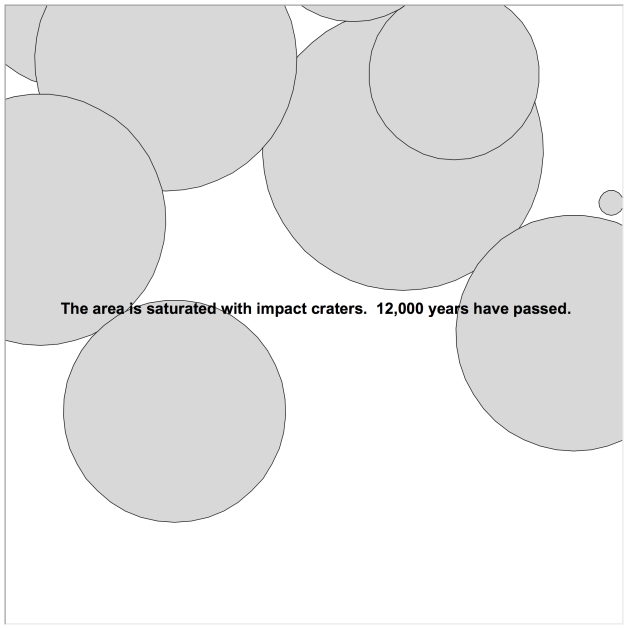
Picture 2: 4 visible impact craters



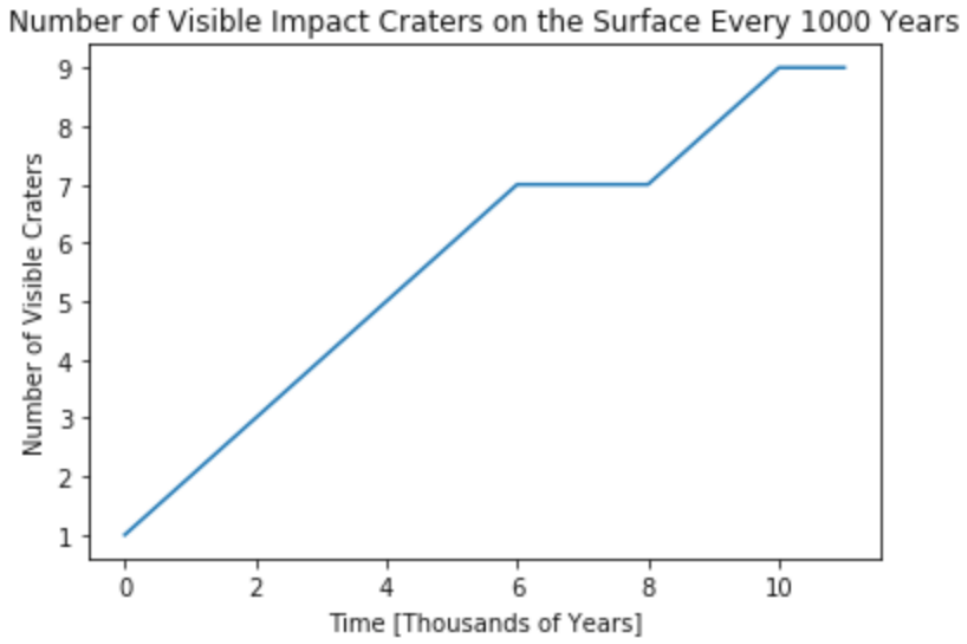
Picture 3: 7 visible impact craters



Picture 4: 9 visible impact craters



C. Below is a screenshot of the Number ofCraters vs. Time plot produced in iPython Jupyter Notebook which corresponds to the simulation that produced the screenshots above.




```
There are 9 total visible craters after 12 thousand years.
```

D. The results of this simulation show that theres a problem somewhere with my initial set of assumptions. It seems obvious that it has to do with the distribution of impact sizes, and as such I will change that assumption for part 2 of the project. The simulation took a mere 12,000 years to complete, which does not seem like a very long time when considering the extreme damage done to the surface. In the present on Earth we expect impacts that create ~20 km in diameter craters every million years or so, which implies that whatever planet we are simulating the surface of must either be in the early stages of its formation or have some astronomically bad luck. Many of the craters seen in the images above would indicate cataclysmic events that could very possibly extinguish all life on Earth. Comparing this result with scientific findings in the real world is sure to instill a certain amount of skepticism of the program.

The plot in part C shows even more clearly why this version of the simulation is flawed. The plateaus or “landings” in the graph mathematically represent periods of time when new crater formation is “cancelled out”, so to speak, with the annihilation of old craters. That is to say, when the slope is 0, a crater is erased every time a crater is formed. One might conjecture that if the impacts were smaller, and, as such, were to take a longer period of time to saturate the surface with cratering, these flat portions of the graph would represent a much smaller portion of the time. However, if one were to base their assumptions regarding impact craters off of this graph alone, he or she might be led to believe that new craters do, in fact, cover up old craters a relatively large percentage of the time. To be specific, the flat areas make up about a quarter of the entire x axis, meaning that ~25% of impactors cover an old crater when they strike. Astronomers can tell this isn’t true when they observe the surfaces of near geologically dead planets such as Mercury, where we see only few large impact craters and an uncountable amount of small impact craters, which statistically have a lower chance of coinciding geographically.

Because I have adjusted the maximum crater radius for part 2, I have included below the section of the code in which I am able to edit this value. The code here appears as it did when I ran the second simulation, after changing my value for the maximum boundary of impact crater radius. 30 kilometers in radius still makes for a very large crater, however it seems much more reasonable than a 150 kilometers radius, especially considering I do not have control over the frequency with which different sizes of impactors hit.

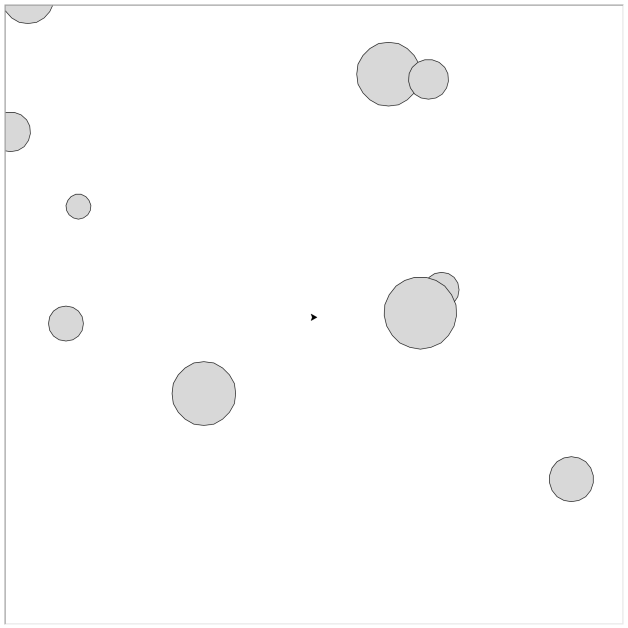
```
# Define important variables
width   = 1000 # window width  (1 kilometer = 2 pixels)
height  = 1000 # window height
satPrct = 0.8  # A fraction denoting the percentage of the surface that must be cratered to be considered "saturated"
rMin    = 20   # minimum crater radius given in outline
#rMax   = 300  # maximum crater radius based on first assumption in part 1 (the largest crater on Earth)
rMax    = 60   # max crater radius based on changed assumption in part 2 (smaller craters that are more likely to form at such a rate)
```

2. Changing the Assumption

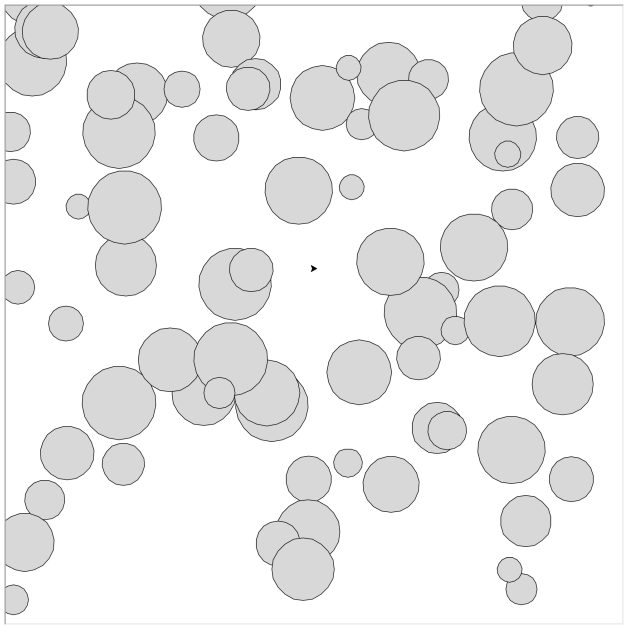
A. The assumption I have chosen to change is that the upper bound of crater radii is 150 km. By lowering this number to 30 km, the final time period calculated by the program should be more realistic, as larger diameter craters are formed very rarely in the real world. This will make for a longer and more interesting simulation.

B. Screenshots of the simulation in progress at arbitrary points in time are included below.

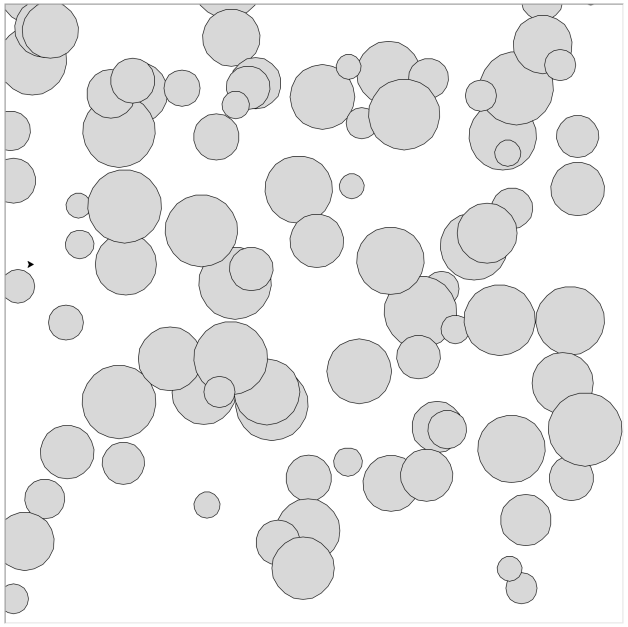
Picture 1: 10 visible impact craters



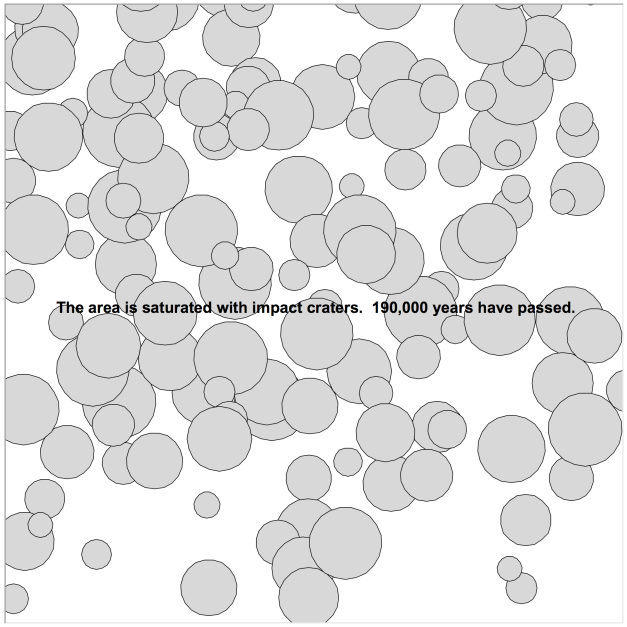
Picture 2



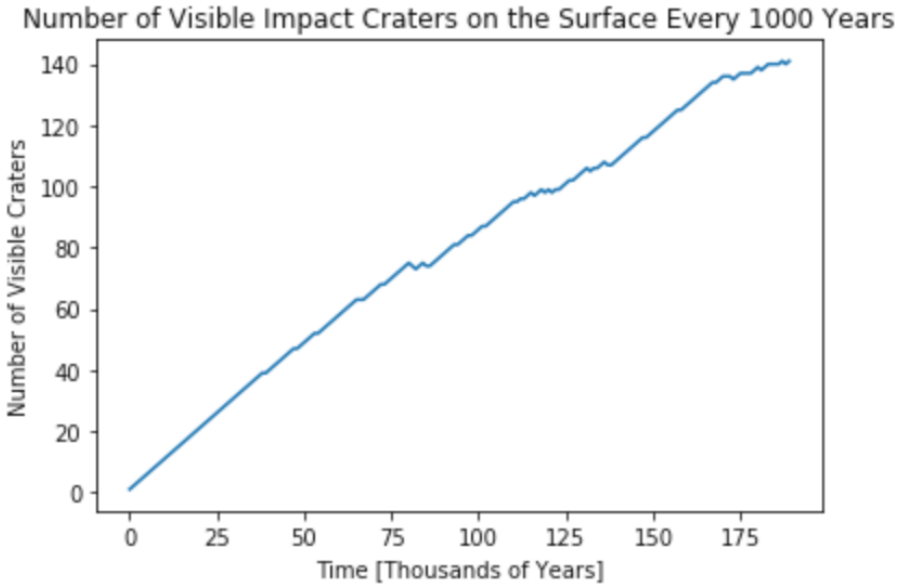
Picture 3



Picture 4: 141 visible impact craters



C. I have included a screenshot of the Number of Craters vs. Time plot produced in iPython Jupyter Notebook which corresponds to the simulation that produced the screenshots above.




```
There are 141 total visible craters after 190 thousand years.
```

D. This result is far more interesting, and likely more accurate than the first simulation. We can see from the pictures that the simulated surface resembles that of a real planet more so than in the original simulation. This is because scientists do not observe craters with radii of hundreds of kilometers forming as often as my simulation would entail on celestial bodies in our solar system. The average crater radius was so large in the first simulation that the program considered the surface sufficiently saturated after only 12,000 years, an astonishingly short period of time on a cosmic scale. In this simulation we see that it takes 190,000 years to saturate the surface, which is significantly longer than in the first simulation.

We also see a higher frequency of “dips” and “landings” in the plot for this simulation. This means that when impactors are not unrealistically huge, more impact craters are covered up by the formation new ones, which would follow logically from the idea that smaller impactors take a considerably longer amount of time to saturate the surface with craters. By this, I do not mean smaller impact craters are covered up *more often* by new ones, but that more of them are covered up due to the extensive amount of time it takes to saturate the surface with small craters. Because radii are generated randomly, and the area of a circle increases with the square of its radius, it is statistically less likely for smaller craters to annihilate each other than for larger craters, especially when the total area of the simulation is confined to certain boundaries.

Lastly, it is worth noting that, because this graph shows a data set collected over a longer amount of time, we have more insight into the trend of visible craters over time. we see that as the number of craters grows larger and larger, old craters are covered up more often. There is nothing surprising about this, and I will do my best to explain why. On a completely smooth surface, there is a 0% chance that a new crater will cover an old one. Because new craters are tiny compared to the simulated area, it takes a while for this probability to increase. We see this in the first 30,000 years of the plot, in which a new visible crater is formed every millennium (the graph has a slope of 1). On the other hand, a surface which is completely covered in craters will show a significant probability that a new crater will destroy an old one. We see this in the graph as well, where, towards the end of the simulation, the slope begins to taper off. In other words, the more impact cratering there is, the more craters get covered up. It is similar to the buck shot cannon experiment that was shown in class. Assuming that impactors strike at constant rate and their sizes fall within a certain range, this is exactly the trend that we should expect.

Code for copy/pasting into Jupyter Notebook if need be.

```
# Course Project: Impact Cratering Saturation Simulation
# ASTR 3750: Planets, Moons, & Rings
# Professor Nick Schneider
# Programmed by Michael Carter
# November 16th, 2017
#-------------------------------------------------------------------------------------------------------------------------------------------

# Import necessary packages
from matplotlib import pyplot as plt
import turtle
import random
import math

#-------------------------------------------------------------------------------------------------------------------------------------------

# Define important variables
width   = 1000 # window width  (1 kilometer = 2 pixels)
height  = 1000 # window height
satPrct = 0.8  # A fraction denoting the percentage of the surface that must be cratered to be considered "saturated"
rMin    = 20   # minimum crater radius given in outline
#rMax   = 300  # maximum crater radius based on first assumption in part 1 (the largest crater on Earth)
rMax    = 60   # max crater radius based on changed assumption in part 2 (smaller craters that are more likely to form at such a rate)


#-------------------------------------------------------------------------------------------------------------------------------------------

# Draws light grey circles that represent impact craters
# r - radius of the crater
# x - x coordinate in window of crater
# y - y coordinate in window of crater
def drawCrater(r, x, y):

    t = turtle.Pen()  # create pen "t"
    t.speed(15)     # set pen to ludicrous speed
    t.penup()       #  lift cursor
    t.goto(x,y)     #  move cursor
    t.pendown()     # place cursor
    t.begin_fill()         # fill end
    t.circle(r)            # draw circle
    t.color('light grey')  # fill color
    t.end_fill()           # fill end
    t.hideturtle()  # remove cursor

#-------------------------------------------------------------------------------------------------------------------------------------------

# Generates a random impact location and crater radius
def getCraterInfo():

    i = [] # A three element list containing crater information. Many of these will be stored in a 2D list allCraters

    i.append(random.randrange(rMin, rMax, 1))  # Generate random radii between bounds defined up top
    i.append(random.randrange(- width/2,   width/2, 1)) # Generate a random x value within the window
    i.append(random.randrange(-height/2,  height/2, 1)) # Generate a random y value within the window

    return i

#-------------------------------------------------------------------------------------------------------------------------------------------

# Determines if new crater covers any old craters, and removes covered craters from allCraters
# newC - the 3 value list of information for the new crater
# allC - the 2D list containing the information of all craters so far
def removeCovered(newC, allC):

    i = 0 # Initial value indicating current index of allC being accessed for comparison

    # Compare all old craters in allC to newC and determine if old craters should be removed from allC
    while i < len(allC):

        # Old crater epicenter distance from new crater epicenter
        d = ((newC[1] - allC[i][1])**2 + (newC[2] - allC[i][2])**2)**0.5

        # If the old crater's center is within the new crater and the old crater has a smaller radius, remove the old crater from allC
        if (d <= newC[0] and allC[i][0] < newC[0]):

            # Create a temporary list to indicate which crater must be removed from list
            oldC = []; oldC = allC[i]

            # Remove old crater info from allCraters
            allC.remove(oldC) # (No need to increment i, as the next crater will now assume the position of the removed crater)
        else:
            i += 1 # increment i, check next crater

    return allC

#-------------------------------------------------------------------------------------------------------------------------------------------

# Determines if the section of the planets surface is "saturated" to end cratering simulation period
# Assumes "saturated" to mean that the total area of visible craters is 80% of the window's area
# allC - a 2D list containing all data for craters so far
def notSaturated(allC):

    thresholdA = width * height * satPrct # Threshold for what is or is not considered saturated [pixels^2]

    crateredA = 0 # Initial cratered area to be recalculated with every new crater [pixels^2]

    # Add up the area of all visible craters drawn [pixels^2]
    for n in range(len(allC)):
        crateredA += (allC[n][0]**2 * math.pi)

    # Compare total area of drawn craters to area of surface section
    if (crateredA <= thresholdA):
        return True  # returns True if area is not saturated
    else:
        return False #  returns False if area is saturated
```

```
    #- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    # Ideal method would involve examining the colors of all the pixels after drawing every crater
    # Method would count the white pixels at the end of the simulation and calculate ratio of white pixels to total pixels
    # This ratio would represent the proportion of the surface unaffected by impacts

#-----------------------------------------------------------------------------------------------------------------------------------

# A method that creates a 3D list containing the final values for all craters in index [0] and total crater amounts in [1]
def createCraters():

    allCraters = []  # a 2D list that will contain the information of all craters created, to be utilized by removeCovered()
    numCraters = []  # a 1D list containing the number of craters for every iteration of the loop, or every 1000 years

    notSat  = True # a boolean variable that will represent the area's status as saturated or not

    # Iterate a loop for every new crater formed, ultimately creating a final allCraters containing only visible craters
    while notSat:

        # Retrieve new crater information. cc = current crater: refers to the crater being created this loop iteration
        cc = getCraterInfo()

        # Remove old craters covered by cc from allCraters
        updatedCraters = removeCovered(cc, allCraters)

        # Replace allCraters with new version that is missing covered craters
        allCraters = updatedCraters

        # Store new crater info in the next index of allCraters
        allCraters.append(cc)

        # Store the total number of visible craters as of this millenium in an array for plotting.
        numCraters.append(len(allCraters)) # (The length of numCraters will equal the number of kiloyears passed in the simulation)

        # Check if area is saturated and if so, end loop
        notSat = notSaturated(allCraters);

    craterLists = [] # Create a 3D list containing both allCraters and numCraters to be accessed by main()
    craterLists.append(allCraters) # First  element of list is 2D list of all crater information
    craterLists.append(numCraters) # Second element of list is 1D list containing the number of visible craters every 1000 years

    return craterLists

#-----------------------------------------------------------------------------------------------------------------------------------

def drawAllCraters(allC):

    # Utilize variable values saved in the beginning to create a light grey 500 x 500 km window
    turtle.setup(width, height)
    turtle.bgcolor('white')

    # Loop through masterCraters list and draw each one
    for n in range(len(allC[0])):
        drawCrater(allC[0][n][0], allC[0][n][1], allC[0][n][2]) # [0]: allCraters, [n]: crater index, [0,1,2]: r, x, y respectively

    # Create a message for the end of the simulation that includes the length of time that has "passed"
    endMessage = "The area is saturated with impact craters.  " + str(len(allC[1])) + ",000 years have passed."

    # Display the end message in the middle of the simulation window.
    turtle.write(endMessage, move=False, align="center", font=("Arial", 25, "bold"))

    turtle.hideturtle() # Remove turtle cursor

    turtle.done() # End turtle drawing session. (May take a while)

#-----------------------------------------------------------------------------------------------------------------------------------

# Creates a plot which features the total number of visible impact craters with respect to time
# numC - list of total craters at each point in time
def plotCraterData(numC):

    time = range(0, len(numC)) # Create a list of times in kiloyears

    plt.plot(time, numC) # Plot number of visible craters with respect to time
    plt.title('Number of Visible Impact Craters on the Surface Every 1000 Years') # Plot title
    plt.xlabel('Time [Thousands of Years]') # x-axis label
    plt.ylabel('Number of Visible Craters') # y-axis label|
    plt.show() # show plot

    print('There are', numC[len(numC) - 1], 'total visible craters after', len(time), 'thousand years.') # Print plot summary

#-----------------------------------------------------------------------------------------------------------------------------------

# Main function that utilizes the methods defined above
# Disclaimer: the graphics window must be closed in order to view the plotted data.
def main():

    # Create a 3D list of all imformation relevant the the simulation using the createCraters function
    craterLists = createCraters()

    # Create list of total craters at each point in time
    totalCraters = craterLists[1]

    # Draw craters in master list
    drawAllCraters(craterLists)

    # Plot number of craters wirh respect to time.
    plotCraterData(totalCraters)

#-----------------------------------------------------------------------------------------------------------------------------------

# Run the simulation
main()
```

```
# Course Project: Impact Cratering Saturation Simulation
# ASTR 3750: Planets, Moons, & Rings
# Professor Nick Schneider
# Programmed by Michael Carter
# November 16th, 2017
#-------------------------------------------------------------------------------------------------------------------------

# Import necessary packages
from matplotlib import pyplot as plt
import turtle
import random
import math

#-------------------------------------------------------------------------------------------------------------------------

# Define important variables
width   = 1000 # window width  (1 kilometer = 2 pixels)
height  = 1000 # window height
satPrct = 0.8  # A fraction denoting the percentage of the surface that must be cratered to be considered "saturated"
rMin    = 20   # minimum crater radius given in outline
rMax    = 300  # maximum crater radius based on first assumption in part 1 (the largest crater on Earth)
#rMax   = 60   # max crater radius based on changed assumption in part 2 (smaller craters that are more likely to form at such a rate)


#-------------------------------------------------------------------------------------------------------------------------

# Draws light grey circles that represent impact craters
# r - radius of the crater
# x - x coordinate in window of crater
# y - y coordinate in window of crater
def drawCrater(r, x, y):

    t = turtle.Pen()  # create pen "t"
    t.speed(1000)     # set pen to ludicrous speed
    t.penup()       #  lift cursor
    t.goto(x,y)     #  move cursor
    t.pendown()     # place cursor
    t.begin_fill()          # fill end
    t.circle(r)             # draw circle
    t.color('light grey')  # fill color
    t.end_fill()            # fill end
    t.hideturtle()  # remove cursor
```

```
#-----------------------------------------------------------------------------------------------------------------------

# Generates a random impact location and crater radius
def getCraterInfo():

    i = [] # A three element list containing crater information. Many of these will be stored in a 2D list allCraters

    i.append(random.randrange(rMin, rMax, 1))  # Generate random radii between bounds defined up top
    i.append(random.randrange(- width/2,   width/2, 1)) # Generate a random x value within the window
    i.append(random.randrange(-height/2,  height/2, 1)) # Generate a random y value within the window

    return i

#-----------------------------------------------------------------------------------------------------------------------

# Determines if new crater covers any old craters, and removes covered craters from allCraters
# newC - the 3 value list of information for the new crater
# allC - the 2D list containing the information of all craters so far
def removeCovered(newC, allC):

    i = 0 # Initial value indicating current index of allC being accessed for comparison

    # Compare all old craters in allC to newC and determine if old craters should be removed from allC
    while i < len(allC):

        # Old crater epicenter distance from new crater epicenter
        d = ((newC[1] - allC[i][1])**2 + (newC[2] - allC[i][2])**2)**0.5

        # If the old crater's center is within the new crater and the old crater has a smaller radius, remove the old crater from allC
        if (d <= newC[0] and allC[i][0] < newC[0]):

            # Create a temporary list to indicate which crater must be removed from list
            oldC = []; oldC = allC[i]

            # Remove old crater info from allCraters
            allC.remove(oldC) # (No need to increment i, as the next crater will now assume the position of the removed crater)
        else:
            i += 1 # increment i, check next crater

    return allC
```

```
#-----------------------------------------------------------------------------------------------------------------------

# Determines if the section of the planets surface is "saturated" to end cratering simulation period
# Assumes "saturated" to mean that the total area of visible craters is 80% of the window's area
# allC - a 2D list containing all data for craters so far
def notSaturated(allC):

    thresholdA = width * height * satPrct # Threshold for what is or is not considered saturated [pixels^2]

    crateredA = 0 # Initial cratered area to be recalculated with every new crater [pixels^2]

    # Add up the area of all visible craters drawn [pixels^2]
    for n in range(len(allC)):
        crateredA += (allC[n][0]**2 * math.pi)

    # Compare total area of drawn craters to area of surface section
    if (crateredA <= thresholdA):
        return True  # returns True if area is not saturated
    else:
        return False #  returns False if area is saturated

    #- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    # Ideal method would involve examining the colors of all the pixels after drawing every crater
    # Method would count the white pixels at the end of the simulation and calculate ratio of white pixels to total pixels
    # This ratio would represent the proportion of the surface unaffected by impacts
```

```
#-------------------------------------------------------------------------------------------------------------------

# A method that creates a 3D list containing the final values for all craters in index [0] and total crater amounts in [1]
def createCraters():

    allCraters = []  # a 2D list that will contain the information of all craters created, to be utilized by removeCovered()
    numCraters = []  # a 1D list containing the number of craters for every iteration of the loop, or every 1000 years

    notSat  = True # a boolean variable that will represent the area's status as saturated or not

    # Iterate a loop for every new crater formed, ultimately creating a final allCraters containing only visible craters
    while notSat:

        # Retrieve new crater information. cc = current crater: refers to the crater being created this loop iteration
        cc = getCraterInfo()

        # Remove old craters covered by cc from allCraters
        updatedCraters = removeCovered(cc, allCraters)

        # Replace allCraters with new version that is missing covered craters
        allCraters = updatedCraters

        # Store new crater info in the next index of allCraters
        allCraters.append(cc)

        # Store the total number of visible craters as of this millenium in an array for plotting.
        numCraters.append(len(allCraters)) # (The length of numCraters will equal the number of kiloyears passed in the simulation)

        # Check if area is saturated and if so, end loop
        notSat = notSaturated(allCraters);

    craterLists = [] # Create a 3D list containing both allCraters and numCraters to be accessed by main()
    craterLists.append(allCraters) # First  element of list is 2D list of all crater information
    craterLists.append(numCraters) # Second element of list is 1D list containing the number of visible craters every 1000 years

    return craterLists
```

```
#-----------------------------------------------------------------------------------------------------------------------

def drawAllCraters(allC):

    # Utilize variable values saved in the beginning to create a light grey 500 x 500 km window
    turtle.setup(width, height)
    turtle.bgcolor('white')

    # Loop through masterCraters list and draw each one
    for n in range(len(allC[0])):
        drawCrater(allC[0][n][0], allC[0][n][1], allC[0][n][2]) # [0]: allCraters, [n]: crater index, [0,1,2]: r, x, y respectively

    # Create a message for the end of the simulation that includes the length of time that has "passed"
    endMessage = "The area is saturated with impact craters.  " + str(len(allC[1])) + ",000 years have passed."

    # Display the end message in the middle of the simulation window.
    turtle.write(endMessage, move=False, align="center", font=("Arial", 25, "bold"))

    turtle.hideturtle() # Remove turtle cursor

    turtle.done() # End turtle drawing session. (May take a while)

#-----------------------------------------------------------------------------------------------------------------------

# Creates a plot which features the total number of visible impact craters with respect to time
# numC - list of total craters at each point in time
def plotCraterData(numC):

    time = range(0, len(numC)) # Create a list of times in kiloyears

    plt.plot(time, numC) # Plot number of visible craters with respect to time
    plt.title('Number of Visible Impact Craters on the Surface Every 1000 Years') # Plot title
    plt.xlabel('Time [Thousands of Years]') # x-axis label
    plt.ylabel('Number of Visible Craters') # y-axis label
    plt.show() # show plot

    print('There are', numC[len(numC) - 1], 'total visible craters after', len(time), 'thousand years.') # Print plot summary
```

```
#-----------------------------------------------------------------------------------------------------------------------

# Main function that utilizes the methods defined above
# Disclaimer: the graphics window must be closed in order to view the plotted data.
def main():

    # Create a 3D list of all imformation relevant the the simulation using the createCraters function
    craterLists = createCraters()

    # Create list of total craters at each point in time
    totalCraters = craterLists[1]

    # Draw craters in master list
    drawAllCraters(craterLists)

    # Plot number of craters wirh respect to time.
    plotCraterData(totalCraters)

#-----------------------------------------------------------------------------------------------------------------------

# Run the simulation
main()
```