**Assignment 3**

**Write a function which took at input a list of lists and return an updated list of lists representing the updated board state. Ensure that this function does not perform any visualization and rename this function to conway\_assignment\_two**Attached is a .py file named Question1.py. which runs on command line by going into the directory of the file and typing:  
  
*python Question1.py  
  
Python Code:  
  
import argparse*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import matplotlib.animation as animation*

*ON = 255*

*OFF = 0*

*vals = [ON, OFF]*

*def conway\_assignment\_two(N):*

*return np.random.choice(vals, N\*N, p=[0.2, 0.8]).reshape(N, N)*

*def addGlider(i, j, grid):*

*glider = np.array([[0, 0, 255],*

*[255, 0, 255],*

*[0, 255, 255]])*

*grid[i:i+3, j:j+3] = glider*

*def addGosperGliderGun(i, j, grid):*

*gun = np.zeros(11\*38).reshape(11, 38)*

*gun[5][1] = gun[5][2] = 255*

*gun[6][1] = gun[6][2] = 255*

*gun[3][13] = gun[3][14] = 255*

*gun[4][12] = gun[4][16] = 255*

*gun[5][11] = gun[5][17] = 255*

*gun[6][11] = gun[6][15] = gun[6][17] = gun[6][18] = 255*

*gun[7][11] = gun[7][17] = 255*

*gun[8][12] = gun[8][16] = 255*

*gun[9][13] = gun[9][14] = 255*

*gun[1][25] = 255*

*gun[2][23] = gun[2][25] = 255*

*gun[3][21] = gun[3][22] = 255*

*gun[4][21] = gun[4][22] = 255*

*gun[5][21] = gun[5][22] = 255*

*gun[6][23] = gun[6][25] = 255*

*gun[7][25] = 255*

*gun[3][35] = gun[3][36] = 255*

*gun[4][35] = gun[4][36] = 255*

*grid[i:i+11, j:j+38] = gun*

*def update(frameNum, img, grid, N):*

*newGrid = grid.copy()*

*for i in range(N):*

*for j in range(N):*

*total = int((grid[i, (j-1)%N] + grid[i, (j+1)%N] +*

*grid[(i-1)%N, j] + grid[(i+1)%N, j] +*

*grid[(i-1)%N, (j-1)%N] + grid[(i-1)%N, (j+1)%N] +*

*grid[(i+1)%N, (j-1)%N] + grid[(i+1)%N, (j+1)%N])/255)*

*if grid[i, j] == ON:*

*if (total < 2) or (total > 3):*

*newGrid[i, j] = OFF*

*else:*

*if total == 3:*

*newGrid[i, j] = ON*

*img.set\_data(newGrid)*

*grid[:] = newGrid[:]*

*return img,*

*def main():*

*parser = argparse.ArgumentParser(description="Runs Conway's Game of Life simulation.")*

*parser.add\_argument('--grid-size', dest='N', required=False)*

*parser.add\_argument('--mov-file', dest='movfile', required=False)*

*parser.add\_argument('--interval', dest='interval', required=False)*

*parser.add\_argument('--glider', action='store\_true', required=False)*

*parser.add\_argument('--gosper', action='store\_true', required=False)*

*args = parser.parse\_args()*

*N = 100*

*if args.N and int(args.N) > 8:*

*N = int(args.N)*

*updateInterval = 50*

*if args.interval:*

*updateInterval = int(args.interval)*

*grid = np.array([])*

*if args.glider:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGlider(1, 1, grid)*

*elif args.gosper:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGosperGliderGun(10, 10, grid)*

*else:*

*grid = conway\_assignment\_two(N)*

*fig, ax = plt.subplots()*

*img = ax.imshow(grid, interpolation='nearest')*

*ani = animation.FuncAnimation(fig, update, fargs=(img, grid, N, ),*

*frames = 10,*

*interval=updateInterval,*

*save\_count=50)*

*if args.movfile:*

*ani.save(args.movfile, fps=30, extra\_args=['-vcodec', 'libx264'])*

*plt.show()*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*

**Write a function which calls conway\_assignment\_two. This function will contain a loop. Each iteration through this loop will represent a single time-step of our simulation. This function will take a parameter which causes it to save the board state for each time-step and return this board state history as a list.**Attached is a .py file named Question2.py. which runs on command line by going into the directory of the file and typing:  
  
*python Question2.py  
  
Python Code:**def conway\_assignment\_two(grid):*

*p = len(grid)*

*q = len(grid[0])*

*for i in range(p):*

*for j in range(q):*

*print(grid[i][j], end='')*

*print()*

*def save(grid, row, col):*

*return (len(grid) > row and len(grid[0]) > col and row >= 0 and col >= 0)*

*def solve(grid):*

*p = len(grid)*

*q = len(grid[0])*

*u = [1, -1, 0, 1, -1, 0, 1, -1]*

*v = [0, 0, -1, -1, -1, 1, 1, 1]*

*for i in range(p):*

*for j in range(q):*

*if (grid[i][j] > 0):*

*for k in range(8):*

*if (save(grid, i + u[k], j + v[k]) and grid[i + u[k]][j + v[k]] > 0):*

*grid[i][j] += 1*

*else:*

*for k in range(8):*

*if (save(grid, i + u[k], j + v[k]) and grid[i + u[k]][j + v[k]] > 0):*

*grid[i][j] -= 1*

*for i in range(p):*

*for j in range(q):*

*if (grid[i][j] > 0):*

*if (grid[i][j] < 3):*

*grid[i][j] = 0*

*# Since Any live cell with*

*# 2 or 3 live neighbors live*

*elif (grid[i][j] <= 4):*

*grid[i][j] = 1*

*# Since Any live cell with*

*# > 3 live neighbors dies*

*elif (grid[i][j] > 4):*

*grid[i][j] = 0*

*else:*

*# Since Any dead cell with*

*# exactly 3 live neighbors*

*# becomes a live cell*

*if (grid[i][j] == -3):*

*grid[i][j] = 1*

*else:*

*grid[i][j] = 0*

*if \_\_name\_\_ == '\_\_main\_\_':*

*grid = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],*

*[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, ],*

*[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ],*

*[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],*

*[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ], ]*

*solve(grid)*

*conway\_assignment\_two(grid)* **A script which runs your simulation for a number of board sizes (starting at 1000x1000) and increasing in increments of 250x250 up to the largest size you can simulate for 100 time steps. This script will run both conway\_assignment\_two and conway\_assignment\_three functions. This script will output a graph of the time steps per millisecond for each board size you attempt for both the assignment three and assignment two implementations.**   
  
Attached is a .py file named Question3.py. which runs on command line by going into the directory of the file and typing:  
  
*python Question3.py  
  
Python Code:*  
  
*import argparse*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import matplotlib.animation as animation*

*ON = 255*

*OFF = 0*

*vals = [ON, OFF]*

*def conway\_assignment\_three(N):*

*return np.random.choice(vals, N\*N, p=[0.2, 0.8]).reshape(N, N)*

*def addGlider(i, j, grid):*

*glider = np.array([[0, 0, 255],*

*[255, 0, 255],*

*[0, 255, 255]])*

*grid[i:i+3, j:j+3] = glider*

*def addGosperGliderGun(i, j, grid):*

*gun = np.zeros(11\*38).reshape(11, 38)*

*gun[5][1] = gun[5][2] = 255*

*gun[6][1] = gun[6][2] = 255*

*gun[3][13] = gun[3][14] = 255*

*gun[4][12] = gun[4][16] = 255*

*gun[5][11] = gun[5][17] = 255*

*gun[6][11] = gun[6][15] = gun[6][17] = gun[6][18] = 255*

*gun[7][11] = gun[7][17] = 255*

*gun[8][12] = gun[8][16] = 255*

*gun[9][13] = gun[9][14] = 255*

*gun[1][25] = 255*

*gun[2][23] = gun[2][25] = 255*

*gun[3][21] = gun[3][22] = 255*

*gun[4][21] = gun[4][22] = 255*

*gun[5][21] = gun[5][22] = 255*

*gun[6][23] = gun[6][25] = 255*

*gun[7][25] = 255*

*gun[3][35] = gun[3][36] = 255*

*gun[4][35] = gun[4][36] = 255*

*grid[i:i+11, j:j+38] = gun*

*def update(frameNum, img, grid, N):*

*newGrid = grid.copy()*

*for i in range(N):*

*for j in range(N):*

*total = int((grid[i, (j-1)%N] + grid[i, (j+1)%N] +*

*grid[(i-1)%N, j] + grid[(i+1)%N, j] +*

*grid[(i-1)%N, (j-1)%N] + grid[(i-1)%N, (j+1)%N] +*

*grid[(i+1)%N, (j-1)%N] + grid[(i+1)%N, (j+1)%N])/255)*

*if grid[i, j] == ON:*

*if (total < 2) or (total > 3):*

*newGrid[i, j] = OFF*

*else:*

*if total == 3:*

*newGrid[i, j] = ON*

*img.set\_data(newGrid)*

*grid[:] = newGrid[:]*

*return img,*

*def main():*

*parser = argparse.ArgumentParser(description="Runs Conway's Game of Life simulation.")*

*parser.add\_argument('--grid-size', dest='N', required=False)*

*parser.add\_argument('--mov-file', dest='movfile', required=False)*

*parser.add\_argument('--interval', dest='interval', required=False)*

*parser.add\_argument('--glider', action='store\_true', required=False)*

*parser.add\_argument('--gosper', action='store\_true', required=False)*

*args = parser.parse\_args()*

*N = 1000*

*if args.N and int(args.N) > 250:*

*N = int(args.N)*

*updateInterval = 100*

*if args.interval:*

*updateInterval = int(args.interval)*

*grid = np.array([])*

*if args.glider:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGlider(1, 1, grid)*

*elif args.gosper:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGosperGliderGun(10, 10, grid)*

*else:*

*grid = conway\_assignment\_three(N)*

*fig, ax = plt.subplots()*

*img = ax.imshow(grid, interpolation='nearest')*

*ani = animation.FuncAnimation(fig, update, fargs=(img, grid, N, ),*

*frames = 10,*

*interval=updateInterval,*

*save\_count=50)*

*if args.movfile:*

*ani.save(args.movfile, fps=30, extra\_args=['-vcodec', 'libx264'])*

*plt.show()*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*  
  
**A script which runs your simulation using conway\_assignment\_three function for 500 time steps using a board of size 1000x1000 with a random initial board state where the cells are chosen to be alive with 50% probability. This script will plot the number of alive and dead cells (they should sum to the number of cells in your board). This plot will be dynamic and will update after each time step. This graph will be created and updated using the Matplotlib library.**  
  
Attached is a .py file named Question4.py. which runs on command line by going into the directory of the file and typing:  
  
*python Question4.py  
  
Python Code:*

*import argparse*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import matplotlib.animation as animation*

*ON = 255*

*OFF = 0*

*vals = [ON, OFF]*

*def conway\_assignment\_three(N):*

*return np.random.choice(vals, N\*N, p=[0.5, 0.5]).reshape(N, N)*

*def addGlider(i, j, grid):*

*glider = np.array([[0, 0, 255],*

*[255, 0, 255],*

*[0, 255, 255]])*

*grid[i:i+3, j:j+3] = glider*

*def addGosperGliderGun(i, j, grid):*

*gun = np.zeros(11\*38).reshape(11, 38)*

*gun[5][1] = gun[5][2] = 255*

*gun[6][1] = gun[6][2] = 255*

*gun[3][13] = gun[3][14] = 255*

*gun[4][12] = gun[4][16] = 255*

*gun[5][11] = gun[5][17] = 255*

*gun[6][11] = gun[6][15] = gun[6][17] = gun[6][18] = 255*

*gun[7][11] = gun[7][17] = 255*

*gun[8][12] = gun[8][16] = 255*

*gun[9][13] = gun[9][14] = 255*

*gun[1][25] = 255*

*gun[2][23] = gun[2][25] = 255*

*gun[3][21] = gun[3][22] = 255*

*gun[4][21] = gun[4][22] = 255*

*gun[5][21] = gun[5][22] = 255*

*gun[6][23] = gun[6][25] = 255*

*gun[7][25] = 255*

*gun[3][35] = gun[3][36] = 255*

*gun[4][35] = gun[4][36] = 255*

*grid[i:i+11, j:j+38] = gun*

*def update(frameNum, img, grid, N):*

*newGrid = grid.copy()*

*for i in range(N):*

*for j in range(N):*

*total = int((grid[i, (j-1)%N] + grid[i, (j+1)%N] +*

*grid[(i-1)%N, j] + grid[(i+1)%N, j] +*

*grid[(i-1)%N, (j-1)%N] + grid[(i-1)%N, (j+1)%N] +*

*grid[(i+1)%N, (j-1)%N] + grid[(i+1)%N, (j+1)%N])/255)*

*if grid[i, j] == ON:*

*if (total < 2) or (total > 3):*

*newGrid[i, j] = OFF*

*else:*

*if total == 3:*

*newGrid[i, j] = ON*

*img.set\_data(newGrid)*

*grid[:] = newGrid[:]*

*return img,*

*def main():*

*parser = argparse.ArgumentParser(description="Runs Conway's Game of Life simulation.")*

*parser.add\_argument('--grid-size', dest='N', required=False)*

*parser.add\_argument('--mov-file', dest='movfile', required=False)*

*parser.add\_argument('--interval', dest='interval', required=False)*

*parser.add\_argument('--glider', action='store\_true', required=False)*

*parser.add\_argument('--gosper', action='store\_true', required=False)*

*args = parser.parse\_args()*

*N = 1000*

*if args.N and int(args.N) > 250:*

*N = int(args.N)*

*updateInterval = 500*

*if args.interval:*

*updateInterval = int(args.interval)*

*grid = np.array([])*

*if args.glider:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGlider(1, 1, grid)*

*elif args.gosper:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGosperGliderGun(10, 10, grid)*

*else:*

*grid = conway\_assignment\_three(N)*

*fig, ax = plt.subplots()*

*img = ax.imshow(grid, interpolation='nearest')*

*ani = animation.FuncAnimation(fig, update, fargs=(img, grid, N, ),*

*frames = 10,*

*interval=updateInterval,*

*save\_count=50)*

*if args.movfile:*

*ani.save(args.movfile, fps=30, extra\_args=['-vcodec', 'libx264'])*

*plt.show()*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*  
**A script that runs your simulation using the conway\_assignment\_three function. It will take command-line parameters specifying the number of time steps, the size of the board, and the starting configuration (blinker, glider, and random). This ought to be the function you use to test your code. There will be an additional command-line parameter which animates the board states.**Attached is a .py file named Question5.py. which runs on command line by going into the directory of the file and typing:  
  
*python Question5.py  
  
Python Code:  
  
import argparse*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import matplotlib.animation as animation*

*ON = 255*

*OFF = 0*

*vals = [ON, OFF]*

*def conway\_assignment\_three(N):*

*return np.random.choice(vals, N\*N, p=[0.5, 0.5]).reshape(N, N)*

*def addGlider(i, j, grid):*

*glider = np.array([[0, 0, 255],*

*[255, 0, 255],*

*[0, 255, 255]])*

*grid[i:i+3, j:j+3] = glider*

*def addGosperGliderGun(i, j, grid):*

*gun = np.zeros(11\*38).reshape(11, 38)*

*gun[5][1] = gun[5][2] = 255*

*gun[6][1] = gun[6][2] = 255*

*gun[3][13] = gun[3][14] = 255*

*gun[4][12] = gun[4][16] = 255*

*gun[5][11] = gun[5][17] = 255*

*gun[6][11] = gun[6][15] = gun[6][17] = gun[6][18] = 255*

*gun[7][11] = gun[7][17] = 255*

*gun[8][12] = gun[8][16] = 255*

*gun[9][13] = gun[9][14] = 255*

*gun[1][25] = 255*

*gun[2][23] = gun[2][25] = 255*

*gun[3][21] = gun[3][22] = 255*

*gun[4][21] = gun[4][22] = 255*

*gun[5][21] = gun[5][22] = 255*

*gun[6][23] = gun[6][25] = 255*

*gun[7][25] = 255*

*gun[3][35] = gun[3][36] = 255*

*gun[4][35] = gun[4][36] = 255*

*grid[i:i+11, j:j+38] = gun*

*def update(frameNum, img, grid, N):*

*newGrid = grid.copy()*

*for i in range(N):*

*for j in range(N):*

*total = int((grid[i, (j-1)%N] + grid[i, (j+1)%N] +*

*grid[(i-1)%N, j] + grid[(i+1)%N, j] +*

*grid[(i-1)%N, (j-1)%N] + grid[(i-1)%N, (j+1)%N] +*

*grid[(i+1)%N, (j-1)%N] + grid[(i+1)%N, (j+1)%N])/255)*

*if grid[i, j] == ON:*

*if (total < 2) or (total > 3):*

*newGrid[i, j] = OFF*

*else:*

*if total == 3:*

*newGrid[i, j] = ON*

*img.set\_data(newGrid)*

*grid[:] = newGrid[:]*

*return img,*

*def main():*

*parser = argparse.ArgumentParser(description="Runs Conway's Game of Life simulation.")*

*parser.add\_argument('--grid-size', dest='N', required=False)*

*parser.add\_argument('--mov-file', dest='movfile', required=False)*

*parser.add\_argument('--interval', dest='interval', required=False)*

*parser.add\_argument('--glider', action='store\_true', required=False)*

*parser.add\_argument('--gosper', action='store\_true', required=False)*

*args = parser.parse\_args()*

*N = 10*

*if args.N and int(args.N) > 3:*

*N = int(args.N)*

*updateInterval = 1*

*if args.interval:*

*updateInterval = int(args.interval)*

*grid = np.array([])*

*if args.glider:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGlider(1, 1, grid)*

*elif args.gosper:*

*grid = np.zeros(N\*N).reshape(N, N)*

*addGosperGliderGun(10, 10, grid)*

*else:*

*grid = conway\_assignment\_three(N)*

*fig, ax = plt.subplots()*

*img = ax.imshow(grid, interpolation='nearest')*

*ani = animation.FuncAnimation(fig, update, fargs=(img, grid, N, ),*

*frames = 10,*

*interval=updateInterval,*

*save\_count=50)*

*if args.movfile:*

*ani.save(args.movfile, fps=30, extra\_args=['-vcodec', 'libx264'])*

*plt.show()*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*