```
# !/usr/bin/env python3
# (c) Dana Hughes 2021
#
# constructing a working game of life
#

#import the necessary modules
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys
import os
import random
import math
import time

xmax = 70
ymax = 30
world = []
xx = chr(0x2588) # prints a black rectange

def createworld():
  '''
  Initializes the map using xmax rows and ymax columns. The list 2D where
  1. Create a 2D array that represents the playing field or map (or world)
  '''
  b = [] # the whole map
  for i in range(ymax):
    t = []
    for j in range(xmax):
      t.append(' ')
    b.append(t)
  return b

def glider():
  '''
  2. Create a start configuration and place it on the map
  '''
```

```
    global world
    xi = random.randint(3, xmax-3)
    yi = random.randint(3, ymax-3)
    ...
    world[yi][xi+1] = xx
    world[yi+1][xi+2] = xx
    ...


def counter(xi,yi,xmaxsize,ymaxsize):
    """
    checks whether the specified xi/yi coordinate is alive or dead
    taking the boundary into account
    """
    # if then statement
    if xi >= 0 and yi >= 0 and xi < xmaxsize:
      if yi >=0 and xi >= 0 and yi < ymaxsize:
        if world[yi][xi] != ' ':
          return 1 # alive cell
    return 0 # dead cell

def countneighbors(xi,yi):
    """
    evaluates all 8 neighbors of a cell xi/yi
    123
    4 5
    678
    """
    # use counter function inside countneighbors function
    global world
    count = 0
    count += counter(xi - 1, yi + 1, xmax, ymax) #1
    count += counter(xi, yi + 1, xmax, ymax) #2
    count += counter(xi + 1, yi + 1, xmax,ymax) #3
    count += counter(xi - 1, yi, xmax, ymax) #4
    count += counter(xi + 1, yi, xmax, ymax) #5
    count += counter(xi - 1, yi - 1, xmax, ymax) #6
    count += counter(xi, yi - 1, xmax, ymax) #7
    count += counter(xi + 1, yi - 1, xmax, ymax) #8
```

```
    return count

  def evolve():
    """
    check the number of neighbors and decide whether the cell
    stays alive or dead in the next generation, fills the next
    into newworld that then replaces the old world variable, returns
    number of alive cells.
    RULES:
    1. if alive and 2 or 3 neighbors --> alive
    2. if dead and 3 neighbors       --> alive
    3. otherwise                     --> dead
    """
    global world
    alive = 0
    newworld = createworld()
    for xi in range(xmax):
      for yi in range(ymax):
        count = countneighbors(xi,yi)
        newworld[yi][xi] = ' '
        if count == 3 and count == 2 and world[yi][xi] != ' ':
          newworld[yi][xi] = 1
        elif count == 3 or world[yi][xi] == ' ':
          newworld[yi][xi] = 1
        else:
          newworld[yi][xi] = 0
          alive += 1
    for yi in range(ymax):
      world[yi] = newworld[yi][:]
    return alive



  def showworld(timetosleep):
    '''
    5. Display the map and use a sleep function so that the map stays up for
    '''
    global world
    os.system('clear')
    print('\n'.join(map(''.join, world)))
```

```
    time.sleep(timetosleep)

if __name__ == '__main__':
  # create map
  world = createworld()
  glider()
  showworld(1.0)
```

✓ 1s completed at 1:40 AM