import random  
import time  
from colorama import init  
from colorama import Fore, Back, Style  
  
  
*## Functions*def printMaze(maze):  
 for i in range(0, height):  
 for j in range(0, width):  
 if (maze[i][j] == 'u'):  
 print(Fore.WHITE + str(maze[i][j]), end=" ")  
 elif (maze[i][j] == '1'):  
 print(Fore.GREEN + str(maze[i][j]), end=" ")  
 else:  
 print(Fore.RED + str(maze[i][j]), end=" ")  
  
 print('\n')  
  
  
*# Find number of surrounding cells*def surroundingCells(rand\_wall):  
 s\_cells = 0  
 if (maze[rand\_wall[0] - 1][rand\_wall[1]] == '1'):  
 s\_cells += 1  
 if (maze[rand\_wall[0] + 1][rand\_wall[1]] == '1'):  
 s\_cells += 1  
 if (maze[rand\_wall[0]][rand\_wall[1] - 1] == '1'):  
 s\_cells += 1  
 if (maze[rand\_wall[0]][rand\_wall[1] + 1] == '1'):  
 s\_cells += 1  
  
 return s\_cells  
  
  
*## Main code  
# Init variables*wall = '0'  
cell = '1'  
unvisited = 'u'  
height = 20  
width = 20  
maze = []  
  
*# Initialize colorama*init()  
  
*# Denote all cells as unvisited*for i in range(0, height):  
 line = []  
 for j in range(0, width):  
 line.append(unvisited)  
 maze.append(line)  
  
*# Randomize starting point and set it a cell*starting\_height = int(random.random() \* height)  
starting\_width = int(random.random() \* width)  
if (starting\_height == 0):  
 starting\_height += 1  
if (starting\_height == height - 1):  
 starting\_height -= 1  
if (starting\_width == 0):  
 starting\_width += 1  
if (starting\_width == width - 1):  
 starting\_width -= 1  
  
*# Mark it as cell and add surrounding walls to the list*maze[starting\_height][starting\_width] = cell  
walls = []  
walls.append([starting\_height - 1, starting\_width])  
walls.append([starting\_height, starting\_width - 1])  
walls.append([starting\_height, starting\_width + 1])  
walls.append([starting\_height + 1, starting\_width])  
  
*# Denote walls in maze*maze[starting\_height - 1][starting\_width] = '0'  
maze[starting\_height][starting\_width - 1] = '0'  
maze[starting\_height][starting\_width + 1] = '0'  
maze[starting\_height + 1][starting\_width] = '0'  
  
while (walls):  
 *# Pick a random wall* rand\_wall = walls[int(random.random() \* len(walls)) - 1]  
  
 *# Check if it is a left wall* if (rand\_wall[1] != 0):  
 if (maze[rand\_wall[0]][rand\_wall[1] - 1] == 'u' and maze[rand\_wall[0]][rand\_wall[1] + 1] == '1'):  
 *# Find the number of surrounding cells* s\_cells = surroundingCells(rand\_wall)  
  
 if (s\_cells < 2):  
 *# Denote the new path* maze[rand\_wall[0]][rand\_wall[1]] = '1'  
  
 *# Mark the new walls  
 # Upper cell* if (rand\_wall[0] != 0):  
 if (maze[rand\_wall[0] - 1][rand\_wall[1]] != '1'):  
 maze[rand\_wall[0] - 1][rand\_wall[1]] = '0'  
 if ([rand\_wall[0] - 1, rand\_wall[1]] not in walls):  
 walls.append([rand\_wall[0] - 1, rand\_wall[1]])  
  
 *# Bottom cell* if (rand\_wall[0] != height - 1):  
 if (maze[rand\_wall[0] + 1][rand\_wall[1]] != '1'):  
 maze[rand\_wall[0] + 1][rand\_wall[1]] = '0'  
 if ([rand\_wall[0] + 1, rand\_wall[1]] not in walls):  
 walls.append([rand\_wall[0] + 1, rand\_wall[1]])  
  
 *# Leftmost cell* if (rand\_wall[1] != 0):  
 if (maze[rand\_wall[0]][rand\_wall[1] - 1] != '1'):  
 maze[rand\_wall[0]][rand\_wall[1] - 1] = '0'  
 if ([rand\_wall[0], rand\_wall[1] - 1] not in walls):  
 walls.append([rand\_wall[0], rand\_wall[1] - 1])  
  
 *# Delete wall* for wall in walls:  
 if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):  
 walls.remove(wall)  
  
 continue  
  
 *# Check if it is an upper wall* if (rand\_wall[0] != 0):  
 if (maze[rand\_wall[0] - 1][rand\_wall[1]] == 'u' and maze[rand\_wall[0] + 1][rand\_wall[1]] == '1'):  
  
 s\_cells = surroundingCells(rand\_wall)  
 if (s\_cells < 2):  
 *# Denote the new path* maze[rand\_wall[0]][rand\_wall[1]] = '1'  
  
 *# Mark the new walls  
 # Upper cell* if (rand\_wall[0] != 0):  
 if (maze[rand\_wall[0] - 1][rand\_wall[1]] != '1'):  
 maze[rand\_wall[0] - 1][rand\_wall[1]] = '0'  
 if ([rand\_wall[0] - 1, rand\_wall[1]] not in walls):  
 walls.append([rand\_wall[0] - 1, rand\_wall[1]])  
  
 *# Leftmost cell* if (rand\_wall[1] != 0):  
 if (maze[rand\_wall[0]][rand\_wall[1] - 1] != '1'):  
 maze[rand\_wall[0]][rand\_wall[1] - 1] = '0'  
 if ([rand\_wall[0], rand\_wall[1] - 1] not in walls):  
 walls.append([rand\_wall[0], rand\_wall[1] - 1])  
  
 *# Rightmost cell* if (rand\_wall[1] != width - 1):  
 if (maze[rand\_wall[0]][rand\_wall[1] + 1] != '1'):  
 maze[rand\_wall[0]][rand\_wall[1] + 1] = '0'  
 if ([rand\_wall[0], rand\_wall[1] + 1] not in walls):  
 walls.append([rand\_wall[0], rand\_wall[1] + 1])  
  
 *# Delete wall* for wall in walls:  
 if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):  
 walls.remove(wall)  
  
 continue  
  
 *# Check the bottom wall* if (rand\_wall[0] != height - 1):  
 if (maze[rand\_wall[0] + 1][rand\_wall[1]] == 'u' and maze[rand\_wall[0] - 1][rand\_wall[1]] == '1'):  
  
 s\_cells = surroundingCells(rand\_wall)  
 if (s\_cells < 2):  
 *# Denote the new path* maze[rand\_wall[0]][rand\_wall[1]] = '1'  
  
 *# Mark the new walls* if (rand\_wall[0] != height - 1):  
 if (maze[rand\_wall[0] + 1][rand\_wall[1]] != '1'):  
 maze[rand\_wall[0] + 1][rand\_wall[1]] = '0'  
 if ([rand\_wall[0] + 1, rand\_wall[1]] not in walls):  
 walls.append([rand\_wall[0] + 1, rand\_wall[1]])  
 if (rand\_wall[1] != 0):  
 if (maze[rand\_wall[0]][rand\_wall[1] - 1] != '1'):  
 maze[rand\_wall[0]][rand\_wall[1] - 1] = '0'  
 if ([rand\_wall[0], rand\_wall[1] - 1] not in walls):  
 walls.append([rand\_wall[0], rand\_wall[1] - 1])  
 if (rand\_wall[1] != width - 1):  
 if (maze[rand\_wall[0]][rand\_wall[1] + 1] != '1'):  
 maze[rand\_wall[0]][rand\_wall[1] + 1] = '0'  
 if ([rand\_wall[0], rand\_wall[1] + 1] not in walls):  
 walls.append([rand\_wall[0], rand\_wall[1] + 1])  
  
 *# Delete wall* for wall in walls:  
 if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):  
 walls.remove(wall)  
  
 continue  
  
 *# Check the right wall* if (rand\_wall[1] != width - 1):  
 if (maze[rand\_wall[0]][rand\_wall[1] + 1] == 'u' and maze[rand\_wall[0]][rand\_wall[1] - 1] == '1'):  
  
 s\_cells = surroundingCells(rand\_wall)  
 if (s\_cells < 2):  
 *# Denote the new path* maze[rand\_wall[0]][rand\_wall[1]] = '1'  
  
 *# Mark the new walls* if (rand\_wall[1] != width - 1):  
 if (maze[rand\_wall[0]][rand\_wall[1] + 1] != '1'):  
 maze[rand\_wall[0]][rand\_wall[1] + 1] = '0'  
 if ([rand\_wall[0], rand\_wall[1] + 1] not in walls):  
 walls.append([rand\_wall[0], rand\_wall[1] + 1])  
 if (rand\_wall[0] != height - 1):  
 if (maze[rand\_wall[0] + 1][rand\_wall[1]] != '1'):  
 maze[rand\_wall[0] + 1][rand\_wall[1]] = '0'  
 if ([rand\_wall[0] + 1, rand\_wall[1]] not in walls):  
 walls.append([rand\_wall[0] + 1, rand\_wall[1]])  
 if (rand\_wall[0] != 0):  
 if (maze[rand\_wall[0] - 1][rand\_wall[1]] != '1'):  
 maze[rand\_wall[0] - 1][rand\_wall[1]] = '0'  
 if ([rand\_wall[0] - 1, rand\_wall[1]] not in walls):  
 walls.append([rand\_wall[0] - 1, rand\_wall[1]])  
  
 *# Delete wall* for wall in walls:  
 if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):  
 walls.remove(wall)  
  
 continue  
  
 *# Delete the wall from the list anyway* for wall in walls:  
 if (wall[0] == rand\_wall[0] and wall[1] == rand\_wall[1]):  
 walls.remove(wall)  
  
*# Mark the remaining unvisited cells as walls*for i in range(0, height):  
 for j in range(0, width):  
 if (maze[i][j] == 'u'):  
 maze[i][j] = '0'  
  
*# Set entrance and exit*for i in range(0, width):  
 if (maze[1][i] == '1'):  
 maze[0][i] = '1'  
 break  
  
for i in range(width - 1, 0, -1):  
 if (maze[height - 2][i] == '1'):  
 maze[height - 1][i] = '1'  
 break  
  
*# Print final maze*printMaze(maze)