```python
# Anthony Tesoriero, Joseph Salemo, Joshua Lunsk, October 17 2019, Concurrency
# Status: Mostly Completed

# Works for n = 3 and n = 4

# Idea Notes
# Size = n (3 for example); CurrentPosition = pos
# North = -n; South = +n; East = +1; West = -1

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

import random


# ---------- Initialization ---------- #
class Concurrency:

        def __init__(self):

                # Size of grid (n x n)
                # self.n = int(input("What is the size of your grid? (n x n) "))
                self.n = 3
                print("Set to (", self.n, "x", self.n, ") grid.")
                print()
                print("1. Spiral Run: Agents start in opposite corners, and move cou
                print("2. Slither Run: Agents start in opposite corners, and move in
                self.version = int(input("Would you like to run 1 or 2? "))

                # self.version = 2

                self.moves = 0

                # Initialize agents [Name, space, current direction]
                self.agent1 = ["Agent 1", 1, 3]
                self.agent2 = ["Agent 2", self.n**2, 1]
                self.foundGems = 0

                # Gets 4 unique random nums
                rands = []
                for i in range(15):
                        rand = random.randint(1, self.n ** 2)
                        if rand not in rands:
                                rands.append(rand)

                # List of gems [name, position]
                self.gems = [["an emerald", rands[0]], ["a crown", rands[1]], ["a co

                self.setType()

        # ---------- Starting Functions ---------- #

        def setType(self):
                # Runs spiralRun when verion 1 is chosen
                if self.version is 1:
                        self.printHeader("Spiral Run")
                        self.spiralRun()
                # Runs slitherRun when version 2 is chosen
                else:
                        self.printHeader("Slither Run")
                        self.slitherRun()

        def printHeader(self, runName):
                print()
                print("-----------------------------")
                print("----- Running", runName, "-----")
                print("-----------------------------")
                print()

        # ---------- Gem Functions ---------- #

        # Finds gem with agent
        def findGem(self, gem, agent):
                self.foundGems += 1
                print(agent[0], "found", gem[0], "in room", gem[1])
                print(4 - self.foundGems, "gems left.")
                gem[1] = -1

        # Find if agent position equals gems
        def gemCheck(self, agent):
                for gem in self.gems:
                        if agent[1] is gem[1]:
                                self.findGem(gem, agent)

        # ---------- Movement Functions ---------- #

        # Checks if direction attempt is valid
        # dir: 1 = north, 2 = east, 3 = south, 4 = west
        def checkDir(self, agent, dir):
                # North
                if dir is 1 and agent[1] - self.n < 1:
                        return False
                # East
                if dir is 2 and agent[1] % self.n is 0:
                        return False
                # South
                if dir is 3 and agent[1] + self.n > self.n ** 2:
                        return False
                # West
                if dir is 4 and (agent[1] - 1) % self.n is 0:
                        return False
                # No failures
                return True

        # Turn agent left
        def turnLeft(self, agent):
                # North turn west
                if agent[2] is 1:
                        agent[2] = 4
                # East turn north
                elif agent[2] is 2:
                        agent[2] = 1
                # South turn east
                elif agent[2] is 3:
                        agent[2] = 2
                # West turn south
                else:
                        agent[2] = 3

        # Turn agent right
        def turnRight(self, agent):
                # North turn east
                if agent[2] is 1:
                        agent[2] = 2
                # East turn south
                elif agent[2] is 2:
                        agent[2] = 3
```

```python
		# South turn west
		elif agent[2] is 3:
			agent[2] = 4
		# West turn north
		else:
			agent[2] = 1

	# Moves agent
	# dir: 1 = north, 2 = east, 3 = south, 4 = west
	def moveForward(self, agent):
		self.gemCheck(agent)
		if agent[2] is 1 and self.checkDir(agent, agent[2]):
			agent[1] -= self.n
		elif agent[2] is 2 and self.checkDir(agent, agent[2]):
			agent[1] += 1
		elif agent[2] is 3 and self.checkDir(agent, agent[2]):
			agent[1] += self.n
		elif agent[2] is 4 and self.checkDir(agent, agent[2]):
			agent[1] -= 1

	# Moves both agent1 and agent 2
	def moveBoth(self):
		self.moveForward(self.agent1)
		self.moveForward(self.agent2)
		self.moves += 1

	# Turns both agent1 and agent2
	# dir: 0 for left, 1 for right
	def turnBoth(self, dir):
		if dir is 0:
			self.turnLeft(self.agent1)
			self.turnLeft(self.agent2)
		elif dir is 1:
			self.turnRight(self.agent1)
			self.turnRight(self.agent2)

	# ---------- Spiral Moving for Version 1 ---------- #

	# Run in a counter-clockwise spiral
	def spiralRun(self):

		dec = 0

		while dec != self.n-1:
			for i in range(self.n-1 - dec):
				self.moveBoth()
			dec += 1
			self.turnBoth(0)

		self.moveBoth()
		self.gemCheck(self.agent1)
		self.gemCheck(self.agent2)

		if self.foundGems is 4:
			print("All gems found in", self.moves, "moves!")
		else:
			print("FAILED")
			print(self.gems)

	# ---------- Slither Moving for Version 2 ---------- #

	# Run in a slither snaking pattern
	def slitherRun(self):

		dir = 0

		while self.foundGems is not 4: # and runSafety < 20:
			# Loop for each column
			for i in range(self.n):
				# More forward until wall
				for i in range(self.n-1):
					self.moveBoth()
				# Mod to return 0 or 1, meaning turn left or right.
				self.turnBoth(dir % 2)
				self.moveBoth()
				self.gemCheck(self.agent1)
				self.gemCheck(self.agent2)
				self.turnBoth(dir % 2)
				dir += 1

			self.turnBoth(1)
			self.turnBoth(1)

		if self.foundGems is 4:
			print("All gems found in", self.moves, "moves!")
		else:
			print("FAILED")
			print(self.gems)

# ---------- Driver ---------- #

# Run Concurrency
game = Concurrency()
```