class Seat:

    def \_\_init\_\_(self, row, number, is\_occupied=False):

        self.row = row

        self.number = number

        self.is\_occupied = is\_occupied

    def \_\_repr\_\_(self):

        return f"{self.row}-{self.number}{'\*' if self.is\_occupied else ''}"

class SeatingSection:

    def \_\_init\_\_(self, section\_id, rows, seats\_per\_row):

        self.section\_id = section\_id

        self.rows = {row: [Seat(row, i) for i in range(1, seats\_per\_row + 1)] for row in range(1, rows + 1)}

    def get\_seat(self, row, number):

        return self.rows[row][number - 1]

    def update\_seat(self, row, number, is\_occupied):

        self.rows[row][number - 1].is\_occupied = is\_occupied

    def \_\_repr\_\_(self):

        return f"Section {self.section\_id}: "+", ".join([str(seat) for sublist in self.rows.values() for seat in sublist])

class Stadium:

    def \_\_init\_\_(self):

        self.sections = {}

    def add\_section(self, section\_id, rows, seats\_per\_row):

        self.sections[section\_id] = SeatingSection(section\_id, rows, seats\_per\_row)

    def remove\_section(self, section\_id):

        if section\_id in self.sections:

            del self.sections[section\_id]

    def update\_seating(self, section\_id, row, number, is\_occupied):

        if section\_id in self.sections:

            self.sections[section\_id].update\_seat(row, number, is\_occupied)

    def get\_section(self, section\_id):

        return self.sections.get(section\_id, None)

    def \_\_repr\_\_(self):

        return "\n".join([str(section) for section in self.sections.values()])

def optimize\_stadium\_seating(stadium):

    # Example simple optimization: fill seats from front rows to back

    for section in stadium.sections.values():

        for row in sorted(section.rows):

            for seat in section.rows[row]:

                if not seat.is\_occupied:

                    seat.is\_occupied = True

def manage\_seat\_allocations(stadium, allocations):

    for allocation in allocations:

        section\_id, row, seat\_num = allocation

        if section\_id in stadium.sections:

            stadium.update\_seating(section\_id, row, seat\_num, True)

import unittest

class TestStadiumSeating(unittest.TestCase):

    def setUp(self):

        self.stadium = Stadium()

        self.stadium.add\_section(101, 10, 10)

        self.stadium.add\_section(102, 5, 20)

    def test\_crud\_seats(self):

        # Test Create/Read

        section = self.stadium.get\_section(101)

        self.assertFalse(section.get\_seat(1, 1).is\_occupied)

        # Test Update

        self.stadium.update\_seating(101, 1, 1, True)

        self.assertTrue(section.get\_seat(1, 1).is\_occupied)

        # Test Delete

        self.stadium.remove\_section(102)

        self.assertIsNone(self.stadium.get\_section(102))

    def test\_optimization(self):

        optimize\_stadium\_seating(self.stadium)

        section = self.stadium.get\_section(101)

        self.assertTrue(all(seat.is\_occupied for row in section.rows.values() for seat in row))

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()