BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

AUTONOMOUS INSTITUTE UNDER VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI 590018

DEPARTMENT OF

ELECTRONICS & COMMUNICATION ENGINEERING

Internship Report on

"Stadium Seating Management"

For the course: Python

Name	**Usn no**
Shankramma.K	**3BR23EC150**
T.Sahana	**3BR23EC163**
U.Preethi Dharani	**3BR23EC173**
Vandana.K	**3BR23EC179**
V.Hema Nandini	**3BR23EC180**

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited institution*

(Recognised by Govt.of karnataka,approved by AICTE,NewDelhi & Affiliated to Visvesvaraya Technological University, Belgavi)

"Jnana Gangotri" Campus,No.873/2,Ballari-Hospet Road,Allipur,Ballari-583104

Karnataka,India.

Ph: 08392-237100/23719, Fax:08392-237197

2024-2025

STADIUM SEATING MANAGEMENT

1. Introduction:

The **Stadium Seating Management System** is a Python-based system designed to manage seat reservations in a stadium-like environment. The system uses a simple grid structure to represent seating arrangements, allowing users to perform operations such as reserving seats, cancelling reservations, checking availability, and displaying the current seating layout.

This report provides an overview of the code structure, its functionality, and potential areas for improvement.

Problem Statement:

1. Difficulty in balancing customer preferences, such as group seating and proximity, with maximizing revenue.

2. Inability to handle last-minute seat changes, cancellations, or upgrades smoothly.

3. Compliance with safety regulations, including accessibility and social distancing protocols.

4. The risk of booking errors, such as double-booking or assigning unavailable seats.

Objectives:

- **Maximizing Occupancy:** Ensure that the maximum number of seats are filled, taking into account demand, seat availability, and any restrictions on seating.

---

2. Code Structure and Components:

The code is organized into a single class called Stadium Seating. The class handles all the major operations of the system, such as:

1. **Displaying the Seating Layout**: The system displays the seating arrangement, marking available seats as 0 and reserved seats as 1.
2. **Reserving a Seat**: It allows the reservation of a seat if it is available.
3. **Cancelling a Reservation**: It enables users to cancel an existing reservation.
4. **Checking Seat Availability**: This feature allows users to check if a seat is available or already reserved.

The seating arrangement is represented as a 2D list (array), with rows and columns reflecting the stadium layout.

---

3. Detailed Analysis of Code Components

3.1 __init__(self, rows, cols)

- **Purpose**: This is the constructor method, responsible for initializing the stadium seating system. It takes two parameters, rows and cols, which represent the number of rows and seats in each row.
- **Functionality**:
    - It creates a 2D list self.seats, where each element is initialized to 0 (indicating the seat is available).
    - **Improvements**: Consider adding input validation to check whether rows and cols are positive integers.

3.2 display_seating(self)

Purpose: This method displays the current seating arrangement.

Functionality: It prints seats in a matrix format, using 0 for available seats and 1 for reserved seats.

Strengths: Easy to read and interpret. The method effectively shows the state of the seats

Improvements: Instead of 0 and 1, you can use more user-friendly symbols like A for available and R for reserved.

3.3 Reserve seat (self, row, col)

- **Purpose**: This method reserves a seat if it is available.
- **Functionality**:
    - Checks whether the provided row and col values are within valid bounds.
    - If the seat is already reserved (1), the system notifies the user that the seat is unavailable.
    - If the seat is available (0), it marks the seat as reserved by setting it to 1.
    - **Improvements**:

        - It could return a status (like True or False) to indicate whether the reservation was successful, which would be useful in larger systems.
        - Handle cases where multiple seats need to be reserved at once.

3.4 Cancel reservation (self, row, col)

- **Purpose**: Cancels an existing reservation.
- **Functionality**:
    - Like the reserve_seat method, it checks the validity of the seat position.
    - If the seat is reserved (1), it cancels the reservation by setting the value back to 0.
    - If the seat is not reserved, the system notifies the user.
    - **Improvements**:
        - It could return a confirmation message (True/False) to the calling function, which would improve its use in larger systems.

3.5 Check availability (self, row, col)

- **Purpose**: Checks if a specific seat is available.
- **Functionality**:
    - This method verifies the seat position and then checks whether the seat is available (0) or reserved (1).
    - **Strengths**: Useful in allowing users to query the seat's status before attempting a reservation.
    - **Improvements**: It could be extended to handle queries for multiple seats or even entire rows.

---

4. Key Features and Usability

1. **Simplicity**: The system is easy to use, with simple methods for reserving, cancelling, and checking seat availability. Each method prints messages that inform users of the action performed, making it user-friendly.

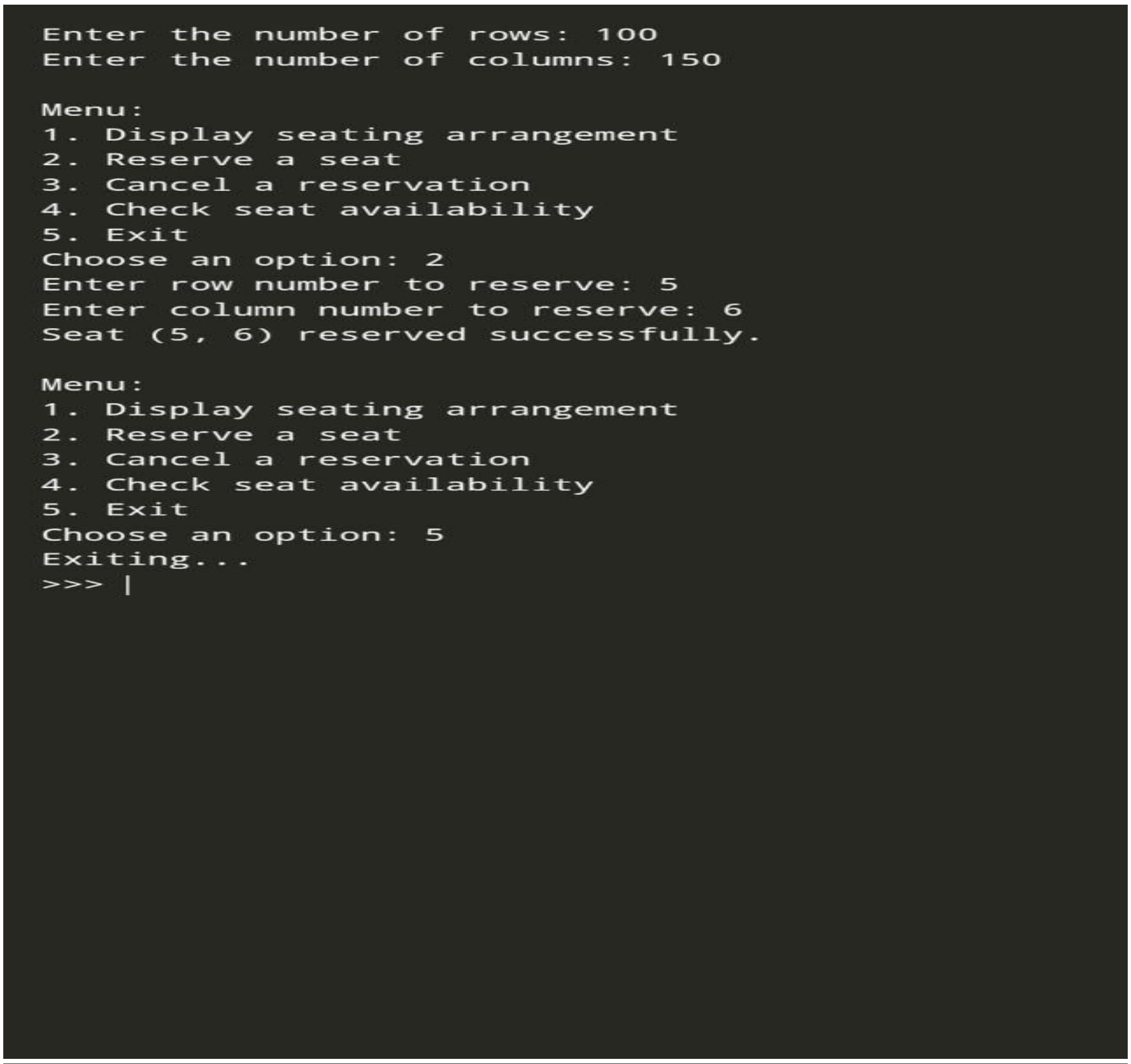
2. **Flexibility**: The system can easily adapt to different stadium sizes (number of rows and columns). This makes the code reusable in various seating management contexts, from small venues to large stadiums.
3. **User Feedback**: The system gives clear and immediate feedback for all actions (e.g., when trying to reserve an already booked seat or cancel an unreserved seat), which enhances user experience.

PROGRAM:

```
class StadiumSeating:
    def _init_(self, rows, cols):
        """
        Initializes the seating arrangement with a specified number of rows and columns.
        Each seat is initially available (marked with a 0).
        """
        self.rows = rows
        self.cols = cols
        self.seats = [[0 for _ in range(cols)] for _ in range(rows)]

    def display_seating(self):
        """
        Displays the current seating arrangement.
        0 indicates an available seat, and 1 indicates a reserved seat.
        """
        print("\nCurrent Seating Arrangement:")
        for row in self.seats:
            print(" ".join(str(seat) for seat in row))

    def reserve_seat(self, row, col):
        """
        Reserves a seat if it is available. Marks the seat with a 1.
        """
        if row >= self.rows or col >= self.cols:
            print("Invalid seat position!")
            return

        if self.seats[row][col] == 1:
            print(f"Seat ({row}, {col}) is already reserved.")
        else:
            self.seats[row][col] = 1
            print(f"Seat ({row}, {col}) reserved successfully.")

    def cancel_reservation(self, row, col):
        """
        Cancels a reservation. Marks the seat with a 0.
        """
        if row >= self.rows or col >= self.cols:
            print("Invalid seat position!")
            return

        if self.seats[row][col] == 0:
            print(f"Seat ({row}, {col}) is not reserved.")
        else:
            self.seats[row][col] = 0
            print(f"Reservation for seat ({row}, {col}) has been canceled.")

    def check_availability(self, row, col):
        """
        Checks if a specific seat is available.
        """
        if row >= self.rows or col >= self.cols:
            print("Invalid seat position!")
            return False

        if self.seats[row][col] == 0:
            print(f"Seat ({row}, {col}) is available.")
            return True
        else:
            print(f"Seat ({row}, {col}) is reserved.")
            return False


# Function to interact with the user
def main():
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))
    stadium = StadiumSeating(rows, cols)

    while True:
        print("\nMenu:")
        print("1. Display seating arrangement")
        print("2. Reserve a seat")
        print("3. Cancel a reservation")
        print("4. Check seat availability")
        print("5. Exit")
```

```
choice = input("Choose an option: ")


if choice == "1":
    stadium.display_seating()
elif choice == "2":
    row = int(input("Enter row number to reserve: "))
    col = int(input("Enter column number to reserve: "))
    stadium.reserve_seat(row, col)
elif choice == "3":
    row = int(input("Enter row number to cancel: "))
    col = int(input("Enter column number to cancel: "))
    stadium.cancel_reservation(row, col)
elif choice == "4":
    row = int(input("Enter row number to check: "))
    col = int(input("Enter column number to check: "))
    stadium.check_availability(row, col)
elif choice == "5":
    print("Exiting...")
    break
else:
    print("Invalid option, please try again.")
```

RESULT:



```
Enter the number of rows: 100
Enter the number of columns: 150

Menu:
1. Display seating arrangement
2. Reserve a seat
3. Cancel a reservation
4. Check seat availability
5. Exit
Choose an option: 2
Enter row number to reserve: 5
Enter column number to reserve: 6
Seat (5, 6) reserved successfully.

Menu:
1. Display seating arrangement
2. Reserve a seat
3. Cancel a reservation
4. Check seat availability
5. Exit
Choose an option: 5
Exiting...
>>> |
```

Conclusion:

This implementation serves as a solid foundation for a seating management system. It could be further enhanced with additional features such as tracking total reserved seats, summarizing all reservations, and improving input validation for a more robust user experience. Overall, it effectively meets the basic requirements for managing stadium seating.

Future Enhancement

Seat Block Reservation:

- Allow users to reserve a block of seats in a rectangular pattern by specifying a start seat and an end seat.
- Automatically ensure that all seats in the block are available before proceeding with the reservation.

Dynamic Pricing:

- Assign different prices to seats based on their location (e.g., front rows are more expensive). Add a method to calculate the total price for reserved seats.