**DSAL ASSIGNMENT**

Swaathi Lakshmanan

Admission No. : p2227171

Module Name : Data Structures and Algorithms

Module Code : EM0303

Class : DISM/FT/2B/21

Lecturer’s Name : Mr. Tan Hu-Shien

Topic/Titles : Data Structures and Algorithms Assignment 1 report

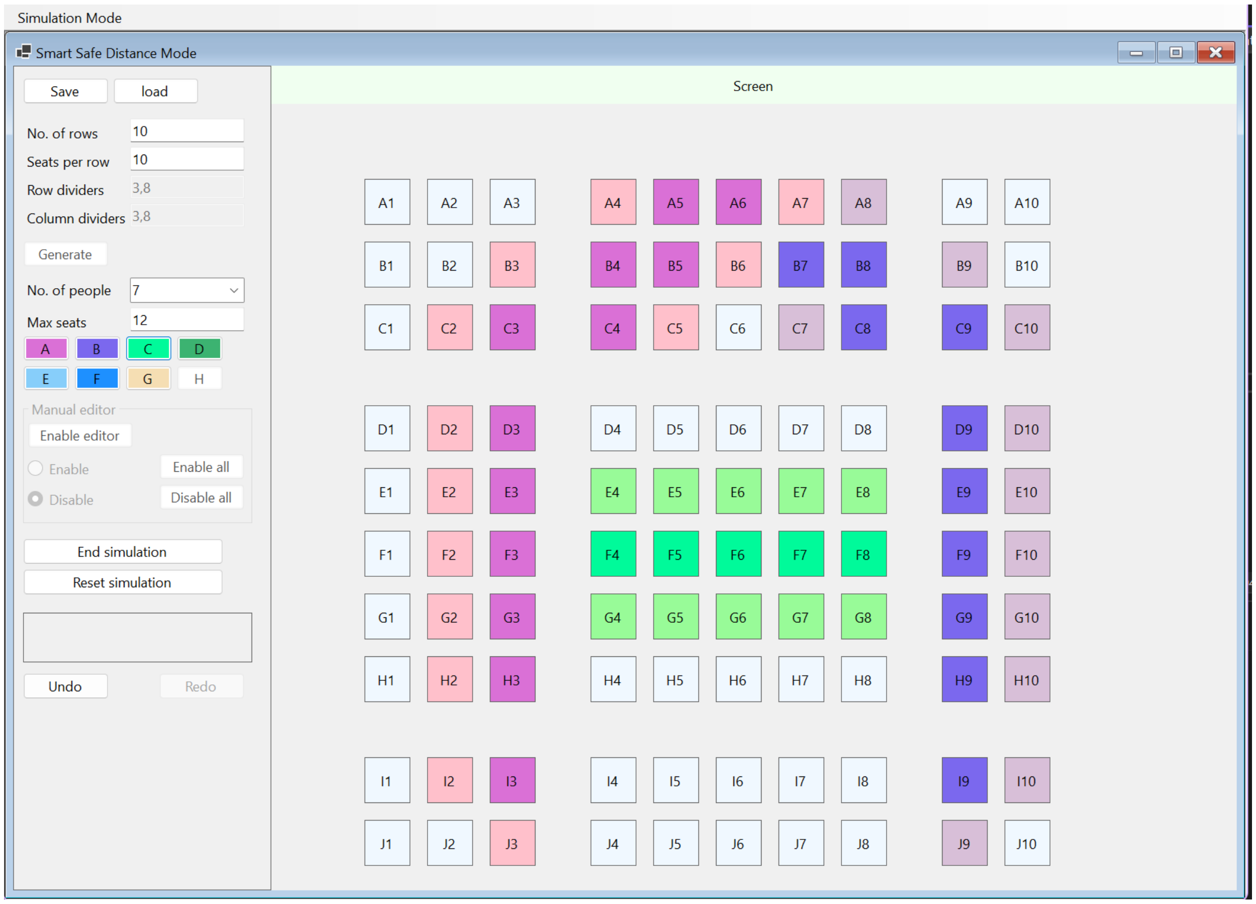
Assignment No. (i.e. 1, 2,3) : 1

Date submitted : 28/5/2023

# Table of Contents

# Report Introduction

The assignment requires us to code a seat booking simulation system for a movie theatre, where a user can generate a desired number of seats and columns and dividers, enable/ disable seats, simulate eight different people booking and three different modes. The entire simulation can be saved and loaded from a binary file.


Simulation Mode
Smart Safe Distance Mode
Save
load
Generate
A
B
C
D
E
F
G
H
Manual editor
Enable editor
Enable
Disable
Enable all
Disable all
End simulation
Reset simulation
Undo
Redo
Screen
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10
E1 E2 E3 E4 E5 E6 E7 E8 E9 E10
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
G1 G2 G3 G4 G5 G6 G7 G8 G9 G10
H1 H2 H3 H4 H5 H6 H7 H8 H9 H10
I1 I2 I3 I4 I5 I6 I7 I8 I9 I10
J1 J2 J3 J4 J5 J6 J7 J8 J9 J10

| | |
|---|---|
| No. of rows | 10 |
| Seats per row | 10 |
| Row dividers | 3,8 |
| Column dividers | 3,8 |
| No. of people | 7 |
| Max seats | 12 |

Figure 0. Assignment screen shot

This report will state what was achieved, talk about the different data structures and algorithms used in this program and the problems encountered.

# Program Design and Functionalities

## Brief explanation on all my classes

1. bin_functionalitites
    a. Contains the code to serialise data and deserialize data and populate the form.
2. form_functions
    a. Contains the code for the form to function.
3. Node
    a. Contains the code for nodes in the double linked list.
4. Person
    a. Contains the data for one person.
5. Persons
    a. Contains a list of all the persons.
6. saveandload
    a. Contains the data that will be serialised.
7. Seat
    a. Contains the data for one seat.
8. SeatDoubleLinkedList
    a. Contains the code for the seat double linked list to be created.
9. Seats
    a. Contains the code for generating and pressing seat labels.
10. userActions
    a. Contains the code for the undo and redo buttons.

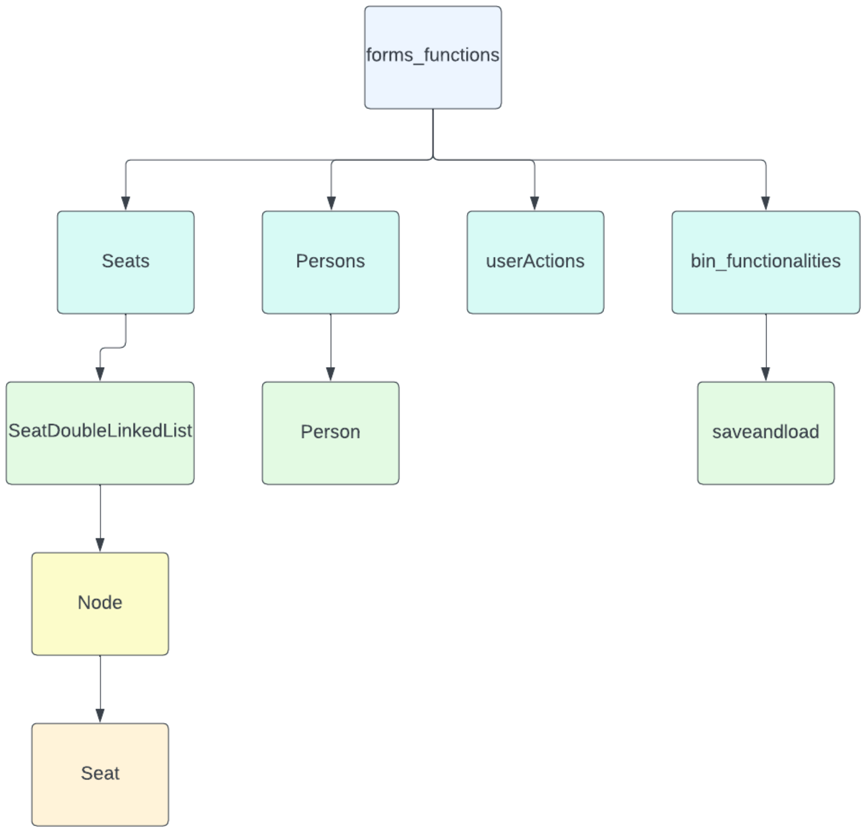
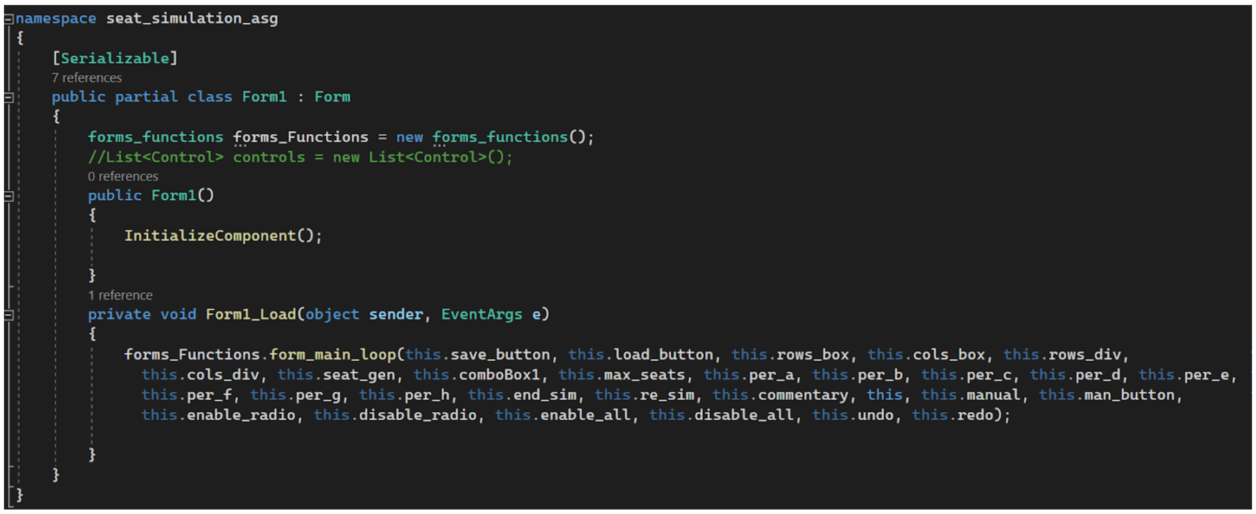


Figure 1. How classes are connected

## forms_functions

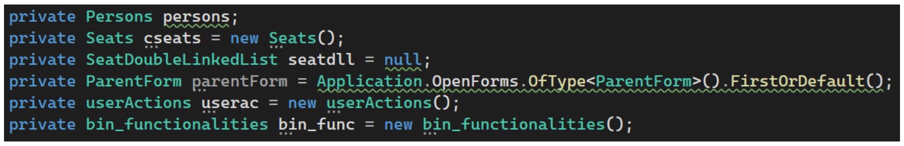


```
namespace seat_simulation_asg
{
    [Serializable]
    7 references
    public partial class Form1 : Form
    {
        forms_functions forms_Functions = new forms_functions();
        //List<Control> controls = new List<Control>();
        0 references
        public Form1()
        {
            InitializeComponent();

        }
        1 reference
        private void Form1_Load(object sender, EventArgs e)
        {
            forms_Functions.form_main_loop(this.save_button, this.load_button, this.rows_box, this.cols_box, this.rows_div,
              this.cols_div, this.seat_gen, this.comboBox1, this.max_seats, this.per_a, this.per_b, this.per_c, this.per_d, this.per_e,
              this.per_f, this.per_g, this.per_h, this.end_sim, this.re_sim, this.commentary, this, this.manual, this.man_button,
              this.enable_radio, this.disable_radio, this.enable_all, this.disable_all, this.undo, this.redo);

        }
    }
}
```

Figure 2. Code behind Form1, Normal Mode.

This is how the code behind all forms look like in my assignment. As you can see all the code behind the form is in this function called form_main_loop.

forms_functions begins by creating new objects of the majority of the classes.



```
private Persons persons;
private Seats cseats = new Seats();
private SeatDoubleLinkedList seatdll = null;
private ParentForm parentForm = Application.OpenForms.OfType<ParentForm>().FirstOrDefault();
private userActions userac = new userActions();
private bin_functionalities bin_func = new bin_functionalities();
```

Figure 3. Initialized variables at the start of forms_functions

After which, in the function, form_main_loop, it brings together these objects to make the form function.

form_main_loop sets up and manages the form's execution. It mostly comprises event handlers and control configurations. It starts by taking the form controls as parameters.

## SeatDoubleLinkedList

This is the class that creates the double linked list in the assignment. It uses IEnumerable interface to allow iteration over the nodes in the array. It starts by creating a two dimensional array of nodes called seats and eventually linking it with the nodes around it.

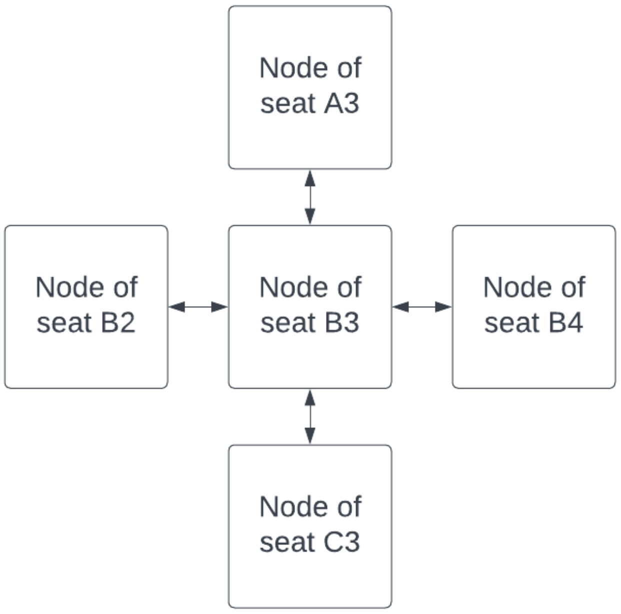
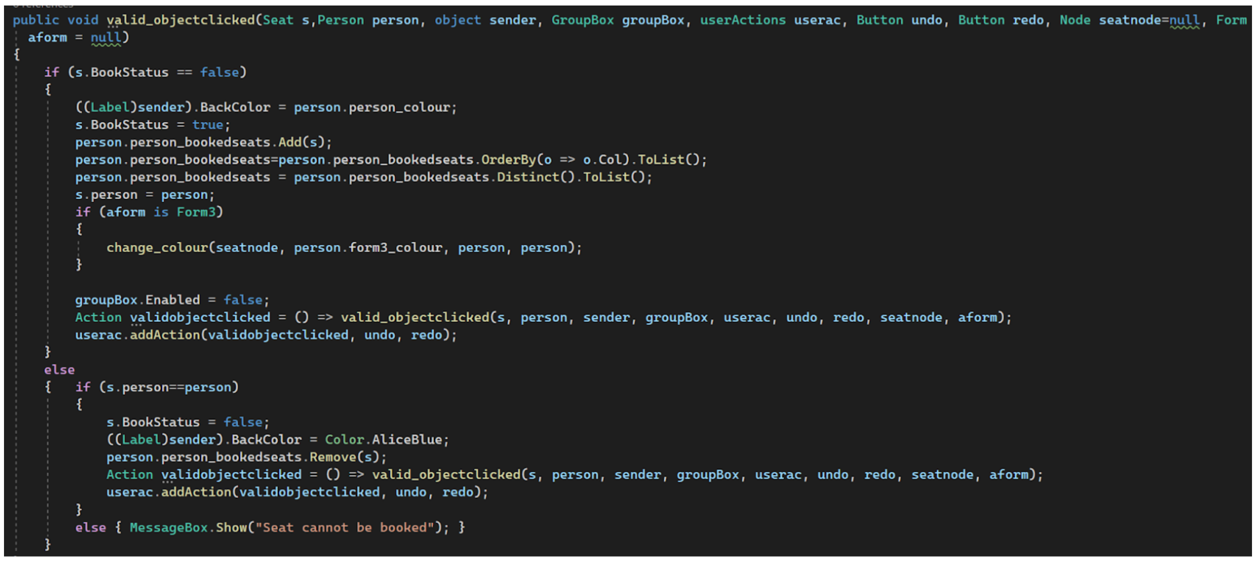


Figure 4.Graphical representation of the linking of nodes in SeatDoubleLinkedList

I decided to link all the seats around a seat because I figured it would be easier to implement the Safe Distance Mode and Safe Distance Smart Mode.

## Seats

The seats class contains the code behind what happens when a seat label is clicked.
Before explaining the main functions within the seats, I will briefly explain valid_objectclicked.

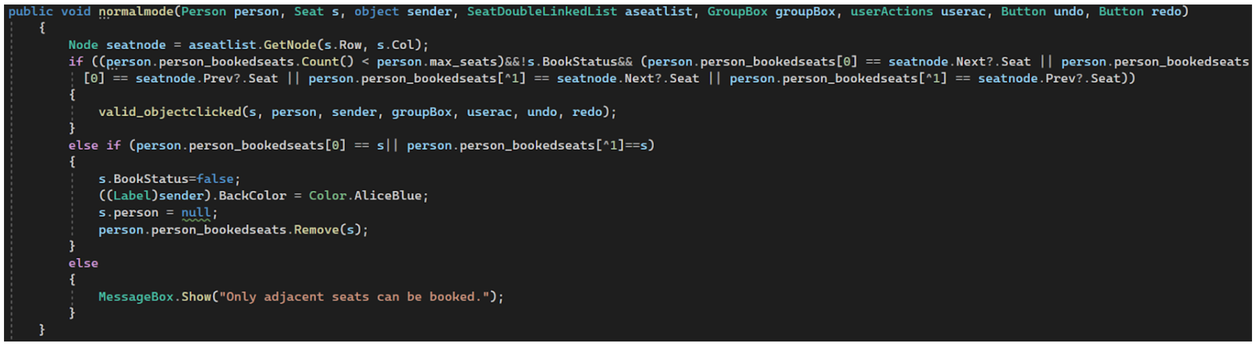


```
public void valid_objectclicked(Seat s,Person person, object sender, GroupBox groupBox, userActions userac, Button undo, Button redo, Node seatnode=null, Form
  aform = null)
{
    if (s.BookStatus == false)
    {
        ((Label)sender).BackColor = person.person_colour;
        s.BookStatus = true;
        person.person_bookedseats.Add(s);
        person.person_bookedseats=person.person_bookedseats.OrderBy(o => o.Col).ToList();
        person.person_bookedseats = person.person_bookedseats.Distinct().ToList();
        s.person = person;
        if (aform is Form3)
        {
            change_colour(seatnode, person.form3_colour, person, person);
        }

        groupBox.Enabled = false;
        Action validobjectclicked = () => valid_objectclicked(s, person, sender, groupBox, userac, undo, redo, seatnode, aform);
        userac.addAction(validobjectclicked, undo, redo);
    }
    else
    {   if (s.person==person)
        {
            s.BookStatus = false;
            ((Label)sender).BackColor = Color.AliceBlue;
            person.person_bookedseats.Remove(s);
            Action validobjectclicked = () => valid_objectclicked(s, person, sender, groupBox, userac, undo, redo, seatnode, aform);
            userac.addAction(validobjectclicked, undo, redo);
        }
        else { MessageBox.Show("Seat cannot be booked"); }
    }
```

Figure 5. valid_objectclicked method in Seats

It is the function that eventually changes the label colour, changes the seat's properties to represent it being booked or unbooked and and changes the person's properties to represent them booking the seat. It's present in most of the methods that this section will talk about.

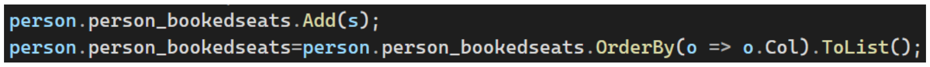
### Normal Mode function



```
public void normalmode(Person person, Seat s, object sender, SeatDoubleLinkedList aseatlist, GroupBox groupBox, userActions userac, Button undo, Button redo)
    {
        Node seatnode = aseatlist.GetNode(s.Row, s.Col);
        if ((person.person_bookedseats.Count() < person.max_seats)&&!s.BookStatus&& (person.person_bookedseats[0] == seatnode.Next?.Seat || person.person_bookedseats
          [0] == seatnode.Prev?.Seat || person.person_bookedseats[^1] == seatnode.Next?.Seat || person.person_bookedseats[^1] == seatnode.Prev?.Seat))
        {
            valid_objectclicked(s, person, sender, groupBox, userac, undo, redo);
        }
        else if (person.person_bookedseats[0] == s|| person.person_bookedseats[^1]==s)
        {
            s.BookStatus=false;
            ((Label)sender).BackColor = Color.AliceBlue;
            s.person = null;
            person.person_bookedseats.Remove(s);
        }
        else
        {
            MessageBox.Show("Only adjacent seats can be booked.");
        }
    }
```

Figure 6. normalmode method in Seats

When a seat is clicked, it is added into a list of the users booked seats after which it is ordered by column number.

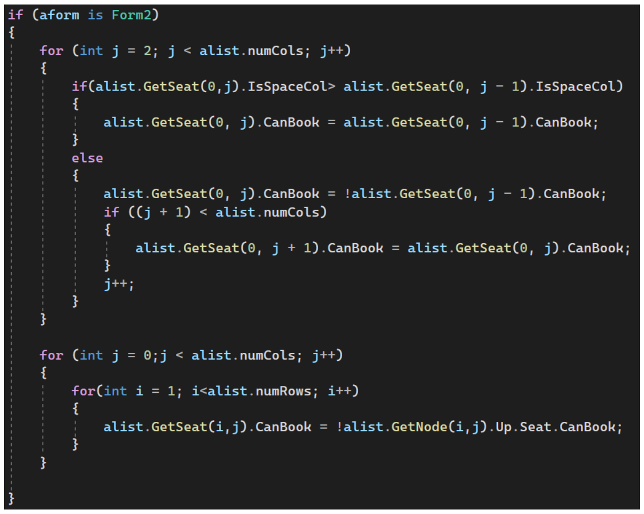


```
person.person_bookedseats.Add(s);
person.person_bookedseats=person.person_bookedseats.OrderBy(o => o.Col).ToList();
```

Figure 7.Ordering seats by column in valid_objectclicked

The code behind normal mode is fairly simple, it just makes sure that the seat being booked is beside a previously booked seat by checking beside the first and the last seat of the aforementioned list. If the seat is already booked, the code moves on the else if statement and checks if the seat being unbooked is the first or last seat in the list.

## Disabling seats for Safe Distance Mode



```
if (aform is Form2)
{
    for (int j = 2; j < alist.numCols; j++)
    {
        if(alist.GetSeat(0,j).IsSpaceCol> alist.GetSeat(0, j - 1).IsSpaceCol)
        {
            alist.GetSeat(0, j).CanBook = alist.GetSeat(0, j - 1).CanBook;
        }
        else
        {
            alist.GetSeat(0, j).CanBook = !alist.GetSeat(0, j - 1).CanBook;
            if ((j + 1) < alist.numCols)
            {
                alist.GetSeat(0, j + 1).CanBook = alist.GetSeat(0, j).CanBook;
            }
            j++;
        }
    }

    for (int j = 0;j < alist.numCols; j++)
    {
        for(int i = 1; i<alist.numRows; i++)
        {
            alist.GetSeat(i,j).CanBook = !alist.GetNode(i,j).Up.Seat.CanBook;
        }
    }

}
```

Figure 8. Disabling seats in genseats

After the seats are generated, the first loop in the above figure loops through the seats in the first row and disabled every other two seats unless there is a column during which it follows the CanBook status of the previous status.

The second loop looks at the seat above it and sets the opposite canbook status for that seat.

## Safe Distance Smart Mode function

The smart Distance Mode function starts by checking if the seats around a particular seat is being booked, and if it is, allowing the user to book the seat. If the seat is already booked by the same person, under the assumption that said seat is not booked, it starts from a seat beside the said seat and recursively checks for seats that are booked that are connected to it.

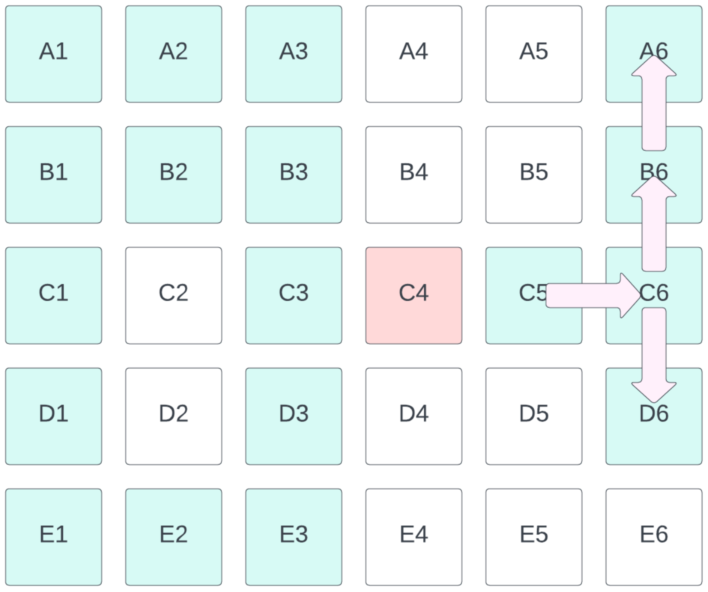
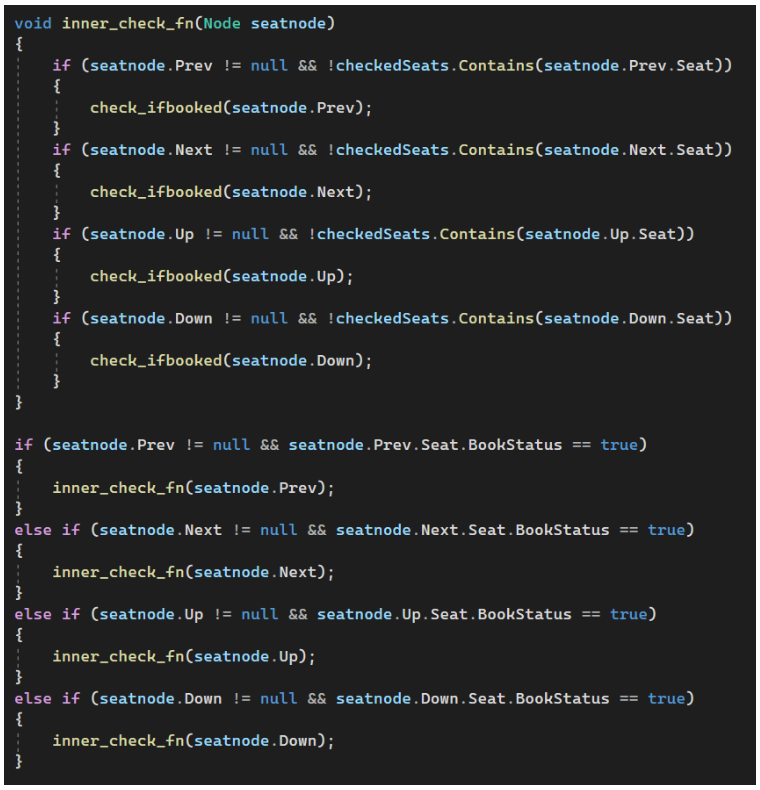


Figure 9. Example of when an unbooking of a seat is not allowed in Safe Distance Smart Mode

The concept of the smart mode is that every seat that is booked is connected to all the other seats that are booked. Thus as seen in the diagram above, the code will check for the seats beside C5 that are booked and add it to an array.



```
void inner_check_fn(Node seatnode)
{
    if (seatnode.Prev != null && !checkedSeats.Contains(seatnode.Prev.Seat))
    {
        check_ifbooked(seatnode.Prev);
    }
    if (seatnode.Next != null && !checkedSeats.Contains(seatnode.Next.Seat))
    {
        check_ifbooked(seatnode.Next);
    }
    if (seatnode.Up != null && !checkedSeats.Contains(seatnode.Up.Seat))
    {
        check_ifbooked(seatnode.Up);
    }
    if (seatnode.Down != null && !checkedSeats.Contains(seatnode.Down.Seat))
    {
        check_ifbooked(seatnode.Down);
    }
}

if (seatnode.Prev != null && seatnode.Prev.Seat.BookStatus == true)
{
    inner_check_fn(seatnode.Prev);
}
else if (seatnode.Next != null && seatnode.Next.Seat.BookStatus == true)
{
    inner_check_fn(seatnode.Next);
}
else if (seatnode.Up != null && seatnode.Up.Seat.BookStatus == true)
{
    inner_check_fn(seatnode.Up);
}
else if (seatnode.Down != null && seatnode.Down.Seat.BookStatus == true)
{
    inner_check_fn(seatnode.Down);
}
```

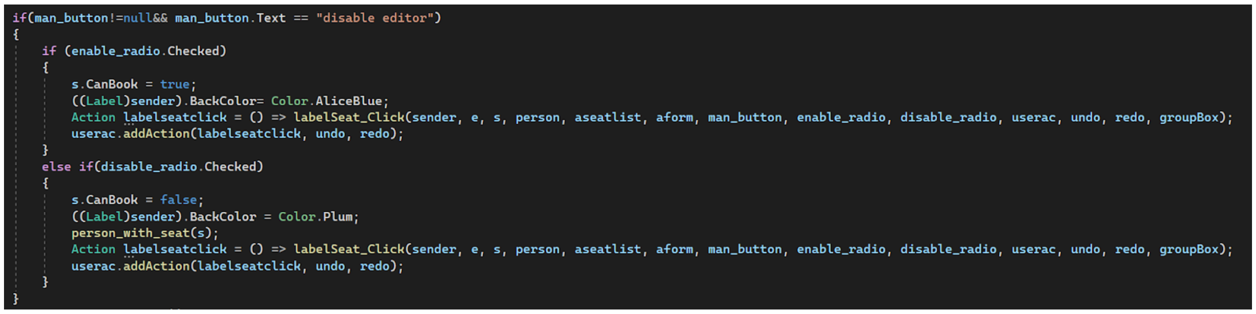
Figure 10.Checking for connected booked seats in smartmode method

After which it will check if the count of the array is the same as 1 less of the count of the booked seats of the person. If it is lesser, the function will alert the user that C4 cannot be unbooked.

To ensure the code does not run infinitely and to make it more efficient, I used a Hashset to keep track of the seats that are already checked.

## Code for Editor Mode

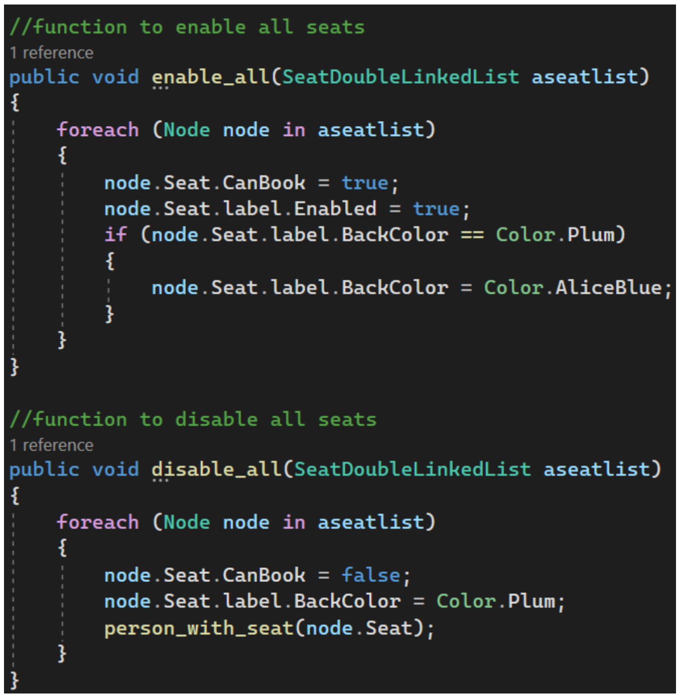
Similar to the normal mode function, the code behind editor mode is also pretty simple.



```
if(man_button!=null&& man_button.Text == "disable editor")
{
    if (enable_radio.Checked)
    {
        s.CanBook = true;
        ((Label)sender).BackColor= Color.AliceBlue;
        Action labelseatclick = () => labelSeat_Click(sender, e, s, person, aseatlist, aform, man_button, enable_radio, disable_radio, userac, undo, redo, groupBox);
        userac.addAction(labelseatclick, undo, redo);
    }
    else if(disable_radio.Checked)
    {
        s.CanBook = false;
        ((Label)sender).BackColor = Color.Plum;
        person_with_seat(s);
        Action labelseatclick = () => labelSeat_Click(sender, e, s, person, aseatlist, aform, man_button, enable_radio, disable_radio, userac, undo, redo, groupBox);
        userac.addAction(labelseatclick, undo, redo);
    }
}
```

Figure 11.Code for when the radio buttons in the manual editor are pressed

If a seat label is clicked, the code checks if the manual editor button was pressed, and if it was which radio button was checked and disables or enables the seat accordingly.



```
//function to enable all seats
1 reference
public void enable_all(SeatDoubleLinkedList aseatlist)
{
    foreach (Node node in aseatlist)
    {
        node.Seat.CanBook = true;
        node.Seat.label.Enabled = true;
        if (node.Seat.label.BackColor == Color.Plum)
        {
            node.Seat.label.BackColor = Color.AliceBlue;
        }
    }
}

//function to disable all seats
1 reference
public void disable_all(SeatDoubleLinkedList aseatlist)
{
    foreach (Node node in aseatlist)
    {
        node.Seat.CanBook = false;
        node.Seat.label.BackColor = Color.Plum;
        person_with_seat(node.Seat);
    }
}
```

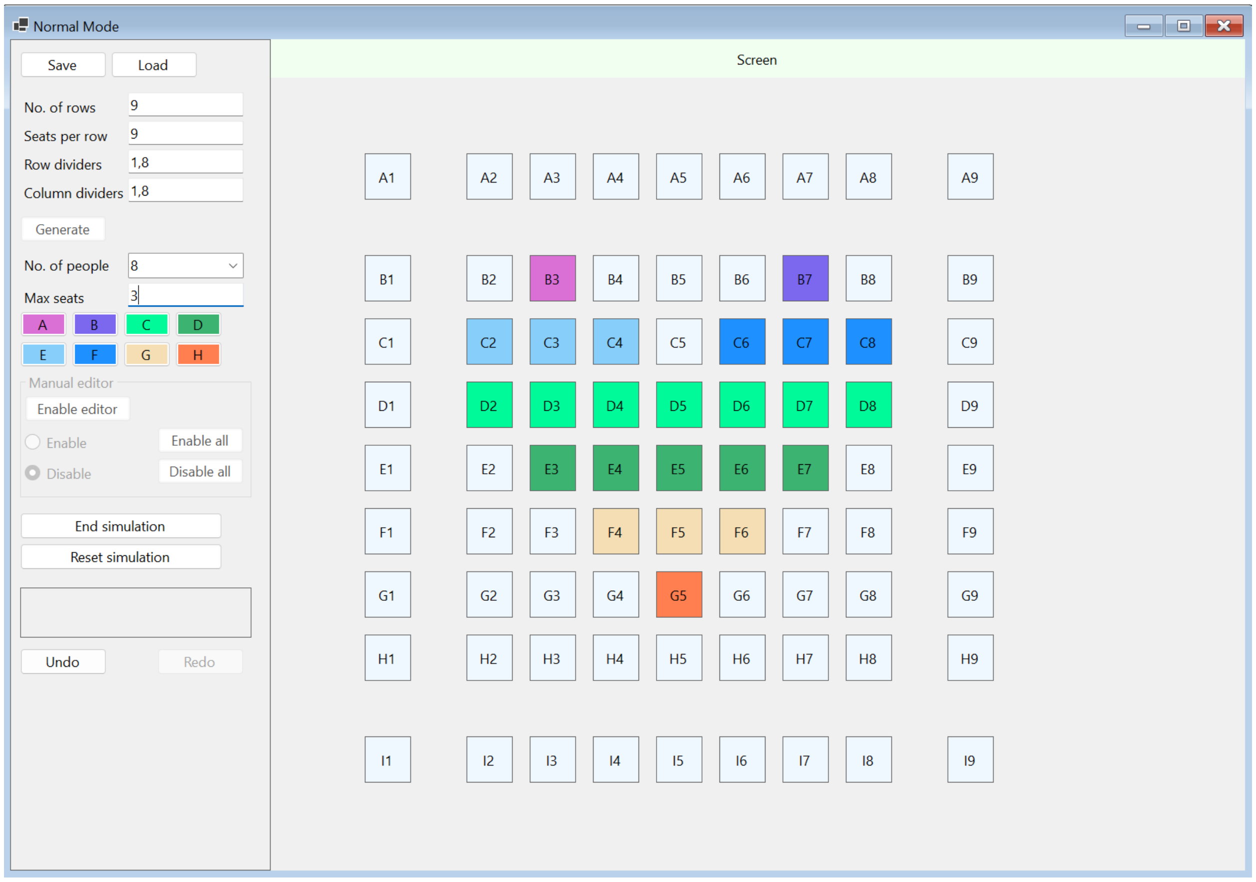
Figure 12.Code for when enable all and disable all buttons

If the enable all or disable all buttons are pressed, the code iterates through every seat, enabling or disabling the seat.

## bin_functionalities

This is the class that serialises and deserialises the simulation. Gen_serialize and gen_deserilaize are the methods that serialise and deserialize when the form data when the generate and reset simulation buttons are pressed respectively. Serialise and Deserialise the methods that serialise and deserialize the form data when the save and load buttons are pressed respectively. The populate_form function is the main function that gathers the deserialized data and puts it back into the form.

# Normal Mode



No. of rows: 9
Seats per row: 9
Row dividers: 1,8
Column dividers: 1,8
No. of people: 8
Max seats: 3

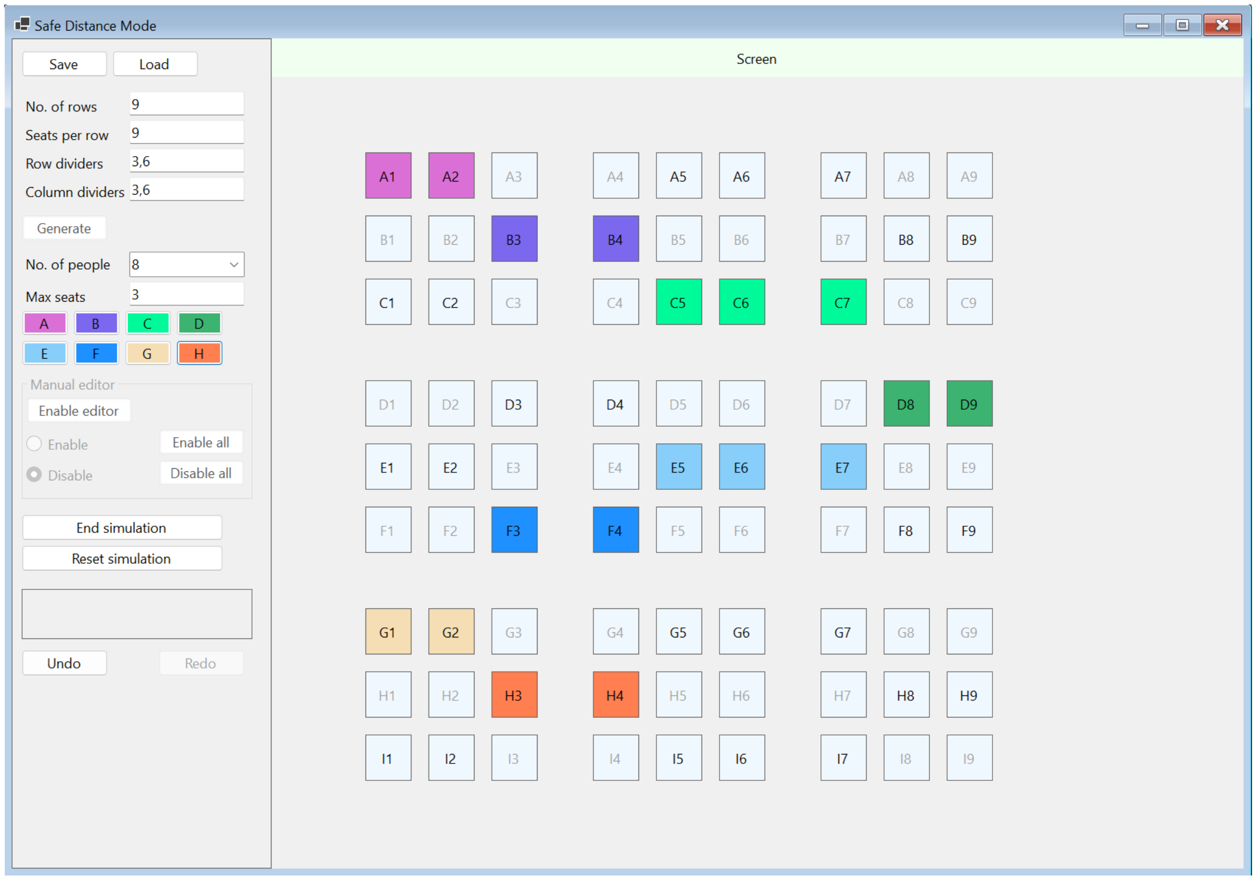
Figure 13. Normal mode screen shot

| | **Normal mode functionality** | **done/half-done/not done/have error** |
|---|---|---|
| 1 | Able to simulate the select/unselect seat for each person (A,B,C,D,E,F,G,H) | done |
| 2 | Able to ensure good user follow when the user changes the number of person during the process of booking/unbooking seats. | done |
| 3 | Able to ensure adjacent seat booking. When you use a person to simulate booking, the | done |

| | | |
|---|---|---|
| | application only allows you to select adjacent seats. | |
| 4 | Able to generate the seats with *custom* row, *custom* column. For example, user provide values to textboxes to layout seats with 10 rows having 10 seats per row. | done |
| 5 | Able to generate seats with *custom* row divider, *custom* column divider. | done |
| 6 | Proper validations are enforced for the textboxes. | done |
| 7 | Able to manually edit the seat properties, to enable/disable the seats *before* beginning the seat selection simulation (seat booking). Able to get the enable all and disable all to work. | done |
| 8 | Able to **disallow** a seat selection which is **next** to another seat selected by another person. E.g. When you simulate Person B booking, person B cannot select a seat next to a seat which has been booked by person A. | done |
| 9 | Able to save the *entire* simulation state into a binary file. | done |

| | | |
|---|---|---|
| 10 | After loading the binary file, the application can reproduce the simulation *state*. For example, the seat layout, the seats which have been booked, the current person which was used for simulating the booking "*before*" the state was saved etc.<br><br>The application also enables and disables the correct Button, Label controls based on the loaded data. | done |
| 11 | Able to **reset** the simulation. | done |
| 12 | Able to **undo-redo** seat selection. | done |

## Safe Distance Mode


Safe Distance Mode
Save
Load
Generate
A
B
C
D
E
F
G
H
Manual editor
Enable editor
Enable
Disable
Enable all
Disable all
End simulation
Reset simulation
Undo
Redo
Screen
A1 A2 A3 A4 A5 A6 A7 A8 A9
B1 B2 B3 B4 B5 B6 B7 B8 B9
C1 C2 C3 C4 C5 C6 C7 C8 C9
D1 D2 D3 D4 D5 D6 D7 D8 D9
E1 E2 E3 E4 E5 E6 E7 E8 E9
F1 F2 F3 F4 F5 F6 F7 F8 F9
G1 G2 G3 G4 G5 G6 G7 G8 G9
H1 H2 H3 H4 H5 H6 H7 H8 H9
I1 I2 I3 I4 I5 I6 I7 I8 I9

No. of rows 9
Seats per row 9
Row dividers 3,6
Column dividers 3,6
No. of people 8
Max seats 3

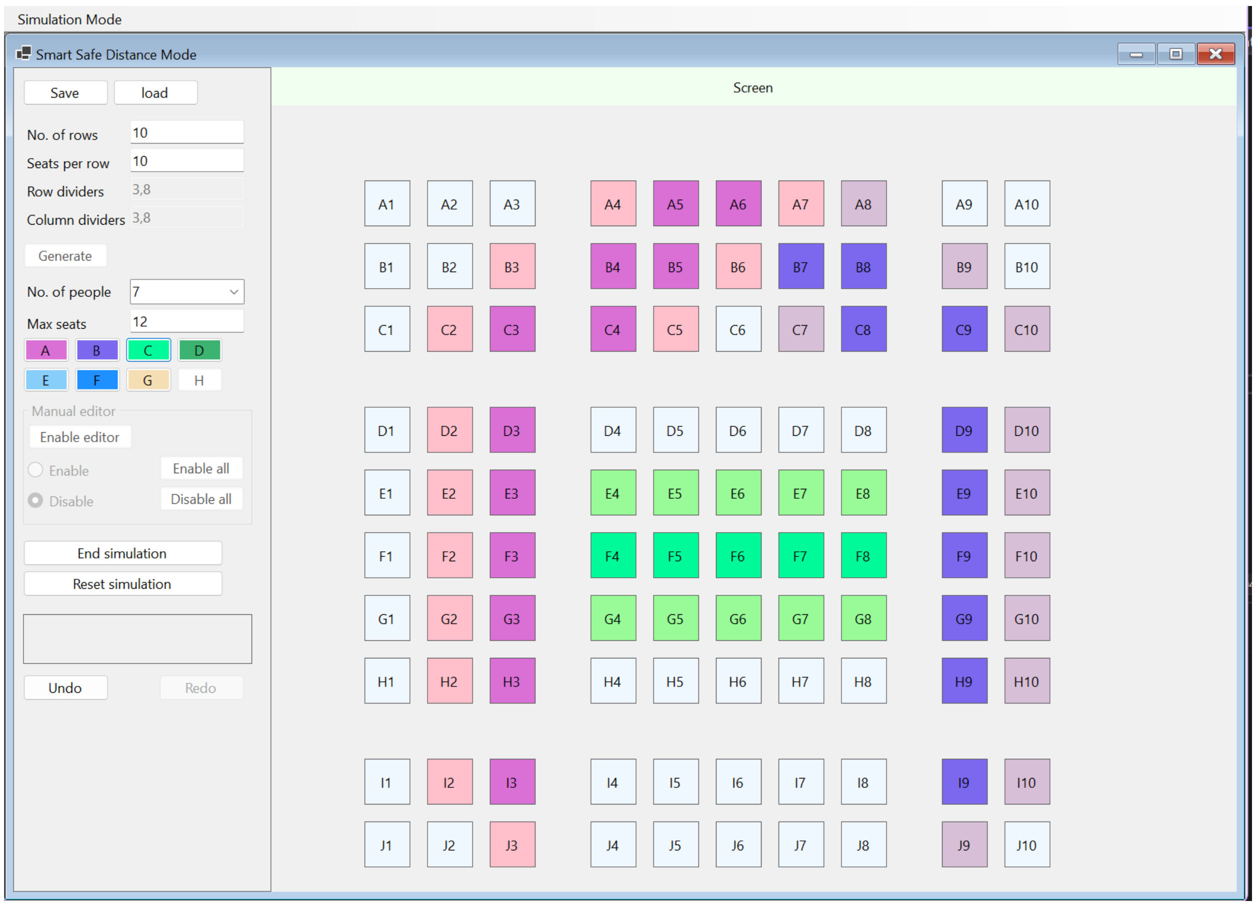
Figure 14.Safe Distance Mode Screen Shot

| | **Safe distancing functionality** | **done/half-done/not done/have error** |
|---|---|---|
| 1 | Able to simulate the select/unselect seat for each person (A,B,C,D,E,F,G,H) | done |
| 2 | Able to auto-disable Label controls to support safe-distancing when the "Setup Safe Distance Mode" button is clicked. | done (In my form, pressing the generate button will generate the seats and disable them at the same time.) |

| | | |
|---|---|---|
| 3 | Able to ensure adjacent seat booking. When you use a person to simulate booking, the application only allows you to select adjacent seats. | done |
| 4 | Able to allow the **unbooking** work correctly so that you don't end up with seat bookings which are not continuous. | done |
| 5 | Able to generate the seats with *custom* row, *custom* column. For example, users provide values to textboxes to layout seats with 10 rows having 10 seats per row. | done |
| 6 | Able to generate seats with *custom* row divider, *custom* column divider. | done |
| 7 | Proper validations are enforced for the input controls. | done |
| 8 | Able to manually edit the seat properties, to enable/disable the seats *before* beginning the seat selection simulation (seat booking).<br>Able to get the enable all and disable all to work. | done |
| 9 | Able to **disallow** a seat selection which is **next** to another seat selected by another person. E.g. When you simulate Person B booking, person B cannot select a seat next to a seat which has been booked by person A. | done |

| | | |
|---|---|---|
| 10 | Able to save the *entire* simulation state into a binary file. | done |
| 11 | After loading the binary file, the application can reproduce the simulation *state*. For example, the seat layout, the seats which have been booked, the current person which was used for simulating the booking "*before*" the state was saved etc.<br><br>The application also enables and disables the correct Button, Label controls based on the loaded data. | done |
| 12 | Able to **reset** the simulation. | done |

## Safe Distance Smart Mode


Simulation Mode
Smart Safe Distance Mode
Save
load
Screen
Generate
A
B
C
D
E
F
G
H
Manual editor
Enable editor
Enable
Enable all
Disable
Disable all
End simulation
Reset simulation
Undo
Redo
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10
E1 E2 E3 E4 E5 E6 E7 E8 E9 E10
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
G1 G2 G3 G4 G5 G6 G7 G8 G9 G10
H1 H2 H3 H4 H5 H6 H7 H8 H9 H10
I1 I2 I3 I4 I5 I6 I7 I8 I9 I10
J1 J2 J3 J4 J5 J6 J7 J8 J9 J10

No. of rows 10
Seats per row 10
Row dividers 3,8
Column dividers 3,8
No. of people 7
Max seats 12

Figure 15.Safe Distance Smart Mode Screen Shot

| | **Safe distancing and Smart mode functionality** | **done/half-done/not done/have error** |
|---|---|---|
| 1 | Able to simulate the select/unselect seat for each person | done |
| 2 | Able to ensure adjacent seat booking **(vertically and horizontally)**. When you use a person to simulate booking, the application only allows you to select adjacent seats. | done |

|  |  |  |
|---|---|---|
| 3 | Able to generate the seats with *custom* row, *custom* column. For example, users provide values to textboxes to layout seats with 10 rows having 10 seats per row. | done |
| 4 | Able to hardcode your own dividers because it is not necessary to have textbox input for custom divider information | done |
| 5 | Proper validations are enforced for the input controls. | done |
| 6 | Able to manually edit the seat properties, to enable/disable the seats *before* beginning the seat selection simulation (**seat booking**).<br>Able to get the "enable all seats" and "disable all seats" to work. | done |
| 7 | Able to **automatically disable the seats surrounding the booked seats** to ensure safe distancing rule. | done |
| 8 | Able to avoid disabling the seats which are separated by a row divider or column divider. | done |
| 9 | Able to save the *entire* simulation state into a binary file. | done |

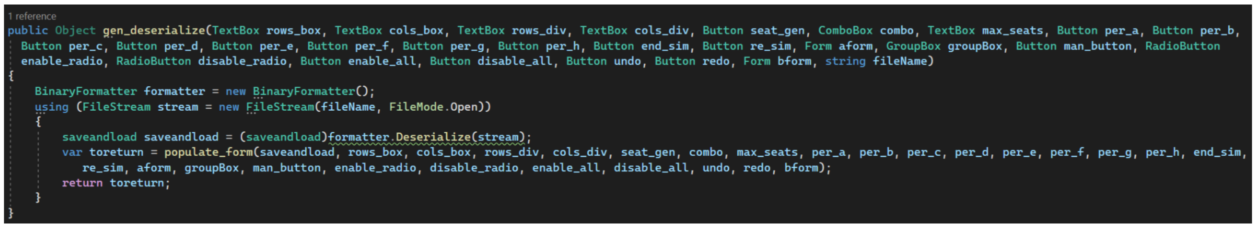
| | | |
|---|---|---|
| 10 | After loading the binary file, the application can reproduce the simulation *state*. For example, the seat layout, the seats which have been booked, the current person which was used for simulating the booking "*before*" the state was saved etc.<br><br>The application also enables and disables the correct Button, Label controls based on the loaded data. | done |
| 11 | Able to **reset** the simulation. | done |

## Challenges

I encountered various challenges during the process of this assignment. I will list down a few here.

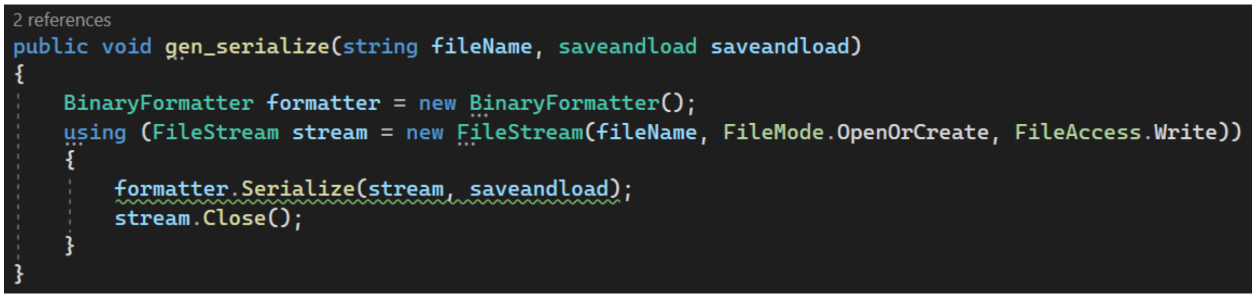
### The Reset button

Once I started coding the assignment, I missed to look at the requirements for the Reset button carefully and I initially coded the Reset button such that it deleted the current form and created a new one. However, I discovered this deviates from the requirements as per the video. Thus, I consulted my DSAL Lecturer who suggested I start coding the Load and Save buttons and simply save the state of the form after the Generate button is pressed in a binary file. Taking this advice, I coded bin_functionalities and created the methods gen_serialize and gen_deserialize.



```
1 reference
public Object gen_deserialize(TextBox rows_box, TextBox cols_box, TextBox rows_div, TextBox cols_div, Button seat_gen, ComboBox combo, TextBox max_seats, Button per_a, Button per_b,
  Button per_c, Button per_d, Button per_e, Button per_f, Button per_g, Button per_h, Button end_sim, Button re_sim, Form aform, GroupBox groupBox, Button man_button, RadioButton
  enable_radio, RadioButton disable_radio, Button enable_all, Button disable_all, Button undo, Button redo, Form bform, string fileName)
{
    BinaryFormatter formatter = new BinaryFormatter();
    using (FileStream stream = new FileStream(fileName, FileMode.Open))
    {
        saveandload saveandload = (saveandload)formatter.Deserialize(stream);
        var toreturn = populate_form(saveandload, rows_box, cols_box, rows_div, cols_div, seat_gen, combo, max_seats, per_a, per_b, per_c, per_d, per_e, per_f, per_g, per_h, end_sim,
           re_sim, aform, groupBox, man_button, enable_radio, disable_radio, enable_all, disable_all, undo, redo, bform);
        return toreturn;
    }
}
```

Figure 16. gen_deserialze method



```
2 references
public void gen_serialize(string fileName, saveandload saveandload)
{
    BinaryFormatter formatter = new BinaryFormatter();
    using (FileStream stream = new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.Write))
    {
        formatter.Serialize(stream, saveandload);
        stream.Close();
    }
}
```

Figure 17. gen_serialize method

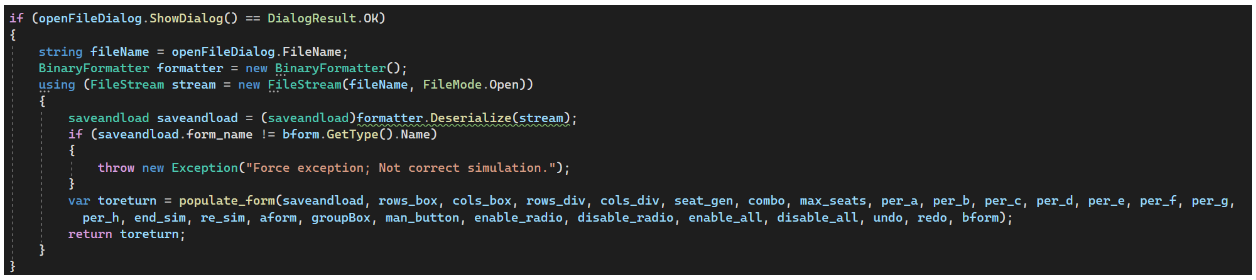
### Saving and Loading forms

I previously did not have any knowledge about saving a form state into a file. Thus I was pretty clueless about this. Initially I tried serialising form_main_loop and the various objects in it and quickly realised that was not possible. After doing extensive research, it became clear that a common approach was to store the data in each form control, save it in a binary file and rebuild the form controls with the data. This seemed very arduous given that I have many labels and other form controls. Thus I consulted my DSAL teacher about why my previous approach of serialising form_main_loop does not work. He told me that form controls cannot be serialised.

After this I took a break and focused on other parts of the form. While I was trying to code my Undo and Redo buttons, I discovered actions. I also realised actions can be serialised, thus I tried to simply save a list of actions and deserialize them and invoke every action until it reaches whichever state the form was in when the user pressed the save button. However, I could not quite figure out how to serialise actions. Thus, I created populate_form under bin_functionalities which collects the serialised data and populates the current form with the deserialized data. It is hardcoded to an extent but I could not find a way around this within the time constraint.

## populate_form method

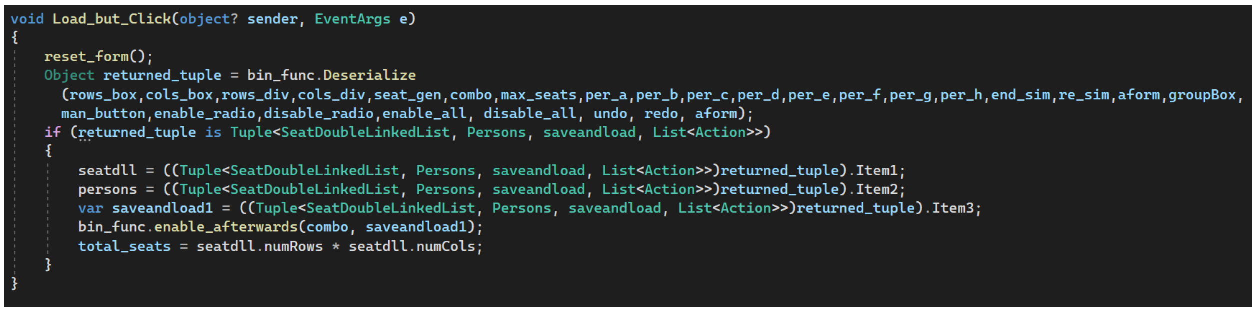
The populate_form method is parked under the Deserialize method.



```
if (openFileDialog.ShowDialog() == DialogResult.OK)
{
    string fileName = openFileDialog.FileName;
    BinaryFormatter formatter = new BinaryFormatter();
    using (FileStream stream = new FileStream(fileName, FileMode.Open))
    {
        saveandload saveandload = (saveandload)formatter.Deserialize(stream);
        if (saveandload.form_name != bform.GetType().Name)
        {
            throw new Exception("Force exception; Not correct simulation.");
        }
        var toreturn = populate_form(saveandload, rows_box, cols_box, rows_div, cols_div, seat_gen, combo, max_seats, per_a, per_b, per_c, per_d, per_e, per_f, per_g,
          per_h, end_sim, re_sim, aform, groupBox, man_button, enable_radio, disable_radio, enable_all, disable_all, undo, redo, bform);
        return toreturn;
    }
}
```

Figure 18. populate_form in Deserialize method

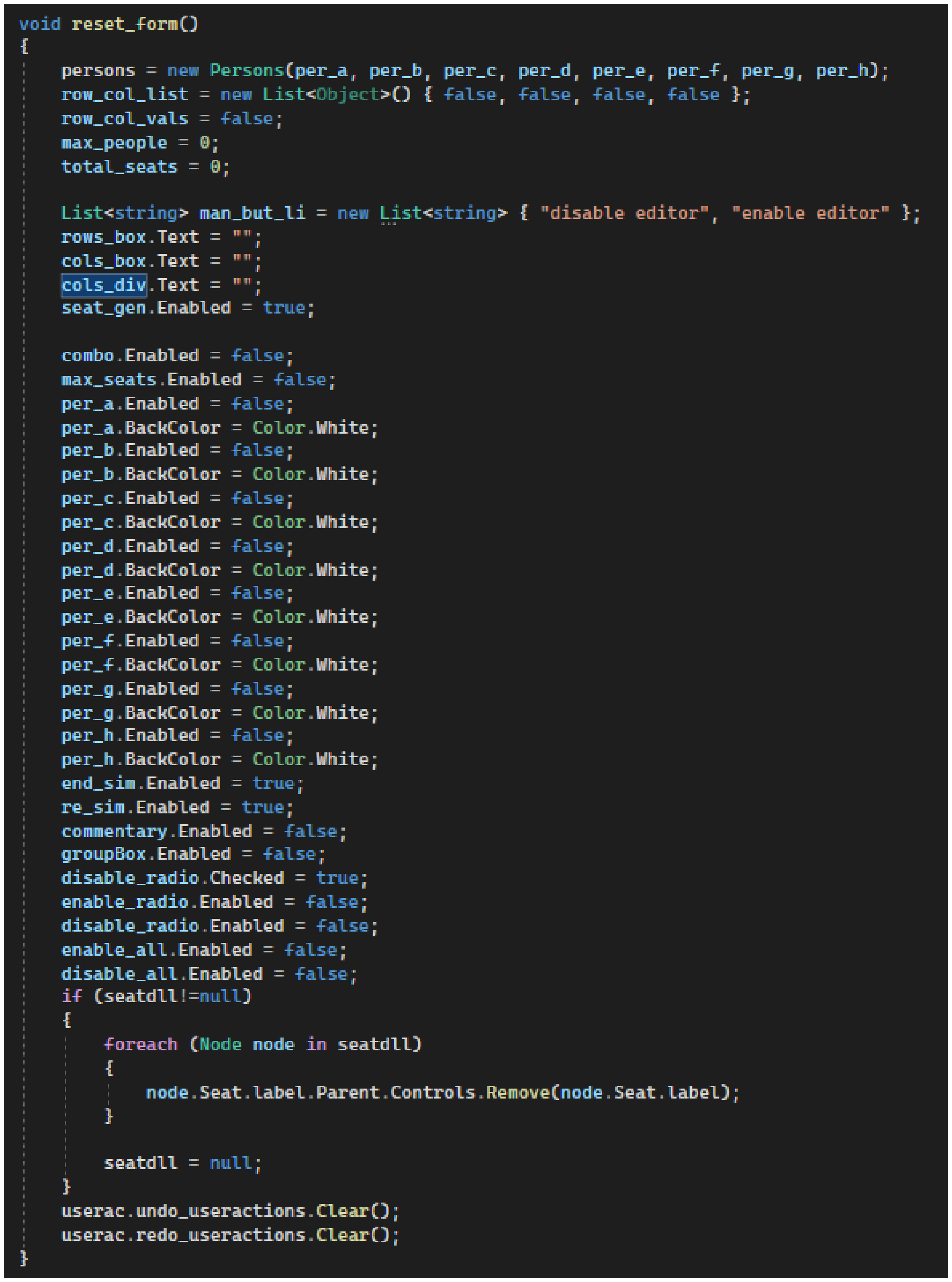
Initially before I invoked the deserialize method, under form_main_loop, I deleted the current form and created a new one of the same class because it took fewer lines of code to bring the form back to its initial state. However while doing this, I did not realise that this creates 2 threads of form_main_loop. While populate_form returned new values of objects to the first thread of form_main_loop, the second thread of form_main_loop still remained unchanged. All the changes from populate_form were occurring in the first_thread of form_main_loop. Since the form_main_loop takes the form controls as arguments, when I change a form control which will change other form controls, such as the combo box where you can choose the number of persons for the simulation, the code will stop abruptly, saying there was no reference for that control.



```
void Load_but_Click(object? sender, EventArgs e)
{
    reset_form();
    Object returned_tuple = bin_func.Deserialize
      (rows_box,cols_box,rows_div,cols_div,seat_gen,combo,max_seats,per_a,per_b,per_c,per_d,per_e,per_f,per_g,per_h,end_sim,re_sim,aform,groupBox,
      man_button,enable_radio,disable_radio,enable_all, disable_all, undo, redo, aform);
    if (returned_tuple is Tuple<SeatDoubleLinkedList, Persons, saveandload, List<Action>>)
    {
        seatdll = ((Tuple<SeatDoubleLinkedList, Persons, saveandload, List<Action>>)returned_tuple).Item1;
        persons = ((Tuple<SeatDoubleLinkedList, Persons, saveandload, List<Action>>)returned_tuple).Item2;
        var saveandload1 = ((Tuple<SeatDoubleLinkedList, Persons, saveandload, List<Action>>)returned_tuple).Item3;
        bin_func.enable_afterwards(combo, saveandload1);
        total_seats = seatdll.numRows * seatdll.numCols;
    }
}
```

Figure 19. Returned objects from populate_form.

I did not realise this in the beginning and was wondering what exactly the error was for about 3 days. Fortunately when I did realise where the error was stemming from, I overcame this with the reset_form method. It changes every single control on the initial form so that the same form can be used when a binary file is loaded. Doing this ensures there is only one thread of form_main_loop.



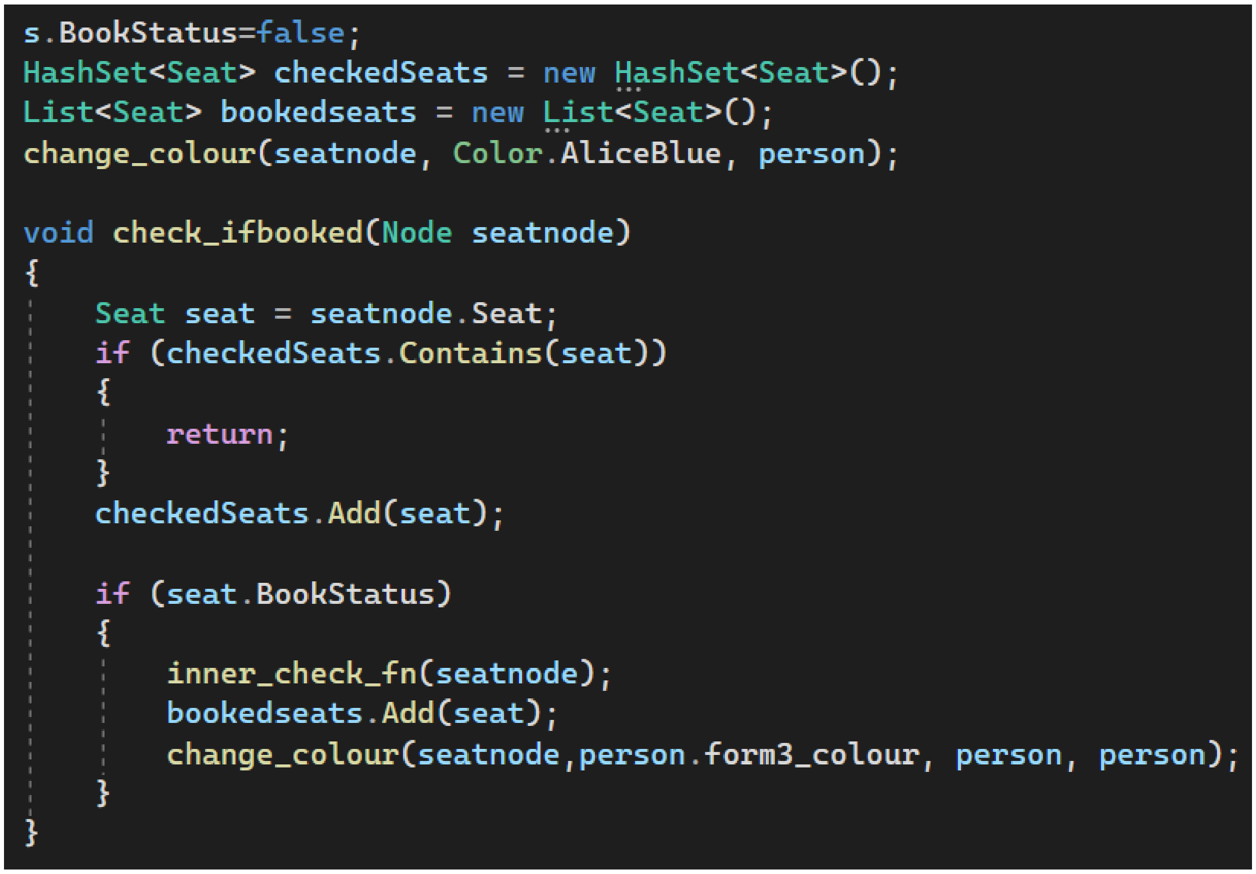
```
void reset_form()
{
    persons = new Persons(per_a, per_b, per_c, per_d, per_e, per_f, per_g, per_h);
    row_col_list = new List<Object>() { false, false, false, false };
    row_col_vals = false;
    max_people = 0;
    total_seats = 0;

    List<string> man_but_li = new List<string> { "disable editor", "enable editor" };
    rows_box.Text = "";
    cols_box.Text = "";
    cols_div.Text = "";
    seat_gen.Enabled = true;

    combo.Enabled = false;
    max_seats.Enabled = false;
    per_a.Enabled = false;
    per_a.BackColor = Color.White;
    per_b.Enabled = false;
    per_b.BackColor = Color.White;
    per_c.Enabled = false;
    per_c.BackColor = Color.White;
    per_d.Enabled = false;
    per_d.BackColor = Color.White;
    per_e.Enabled = false;
    per_e.BackColor = Color.White;
    per_f.Enabled = false;
    per_f.BackColor = Color.White;
    per_g.Enabled = false;
    per_g.BackColor = Color.White;
    per_h.Enabled = false;
    per_h.BackColor = Color.White;
    end_sim.Enabled = true;
    re_sim.Enabled = true;
    commentary.Enabled = false;
    groupBox.Enabled = false;
    disable_radio.Checked = true;
    enable_radio.Enabled = false;
    disable_radio.Enabled = false;
    enable_all.Enabled = false;
    disable_all.Enabled = false;
    if (seatdll!=null)
    {
        foreach (Node node in seatdll)
        {
            node.Seat.label.Parent.Controls.Remove(node.Seat.label);
        }

        seatdll = null;
    }
    userac.undo_useractions.Clear();
    userac.redo_useractions.Clear();
}
```

Figure 20. reset_form method

## Unbooking seats in Smart Safe Distance Mode

Since the booking seats in smart safe distance mode were pretty comprehensible, I thought that the logic behind unbooking them would be pretty comprehensible until I started coding. I did not anticipate how hard it would be to find the logic behind which seat can be unbooked. Initially I thought I could check the rows and columns and check whether there are gaps in either the rows or columns, but I couldn’t find a specific logic that worked as there were many exceptions.

Once I realised finding a logic to see which seats can be unbooked is extremely hard, I decided I will just check if the one seat beside the potentially unbooked seat is connected to every other seat. I chose a seat and then checked for a booked seat beside it and checked for a booked seat beside that one etc. Unfortunately this resulted in an infinite loop which finally caused the program to become unresponsive. To navigate this, I did some research and found Hash Set which ensures it only has unique elements and is easier to iterate through. I used a Hash Set to keep track of the already checked seats. This made my ‘checking from one seat’ idea possible.



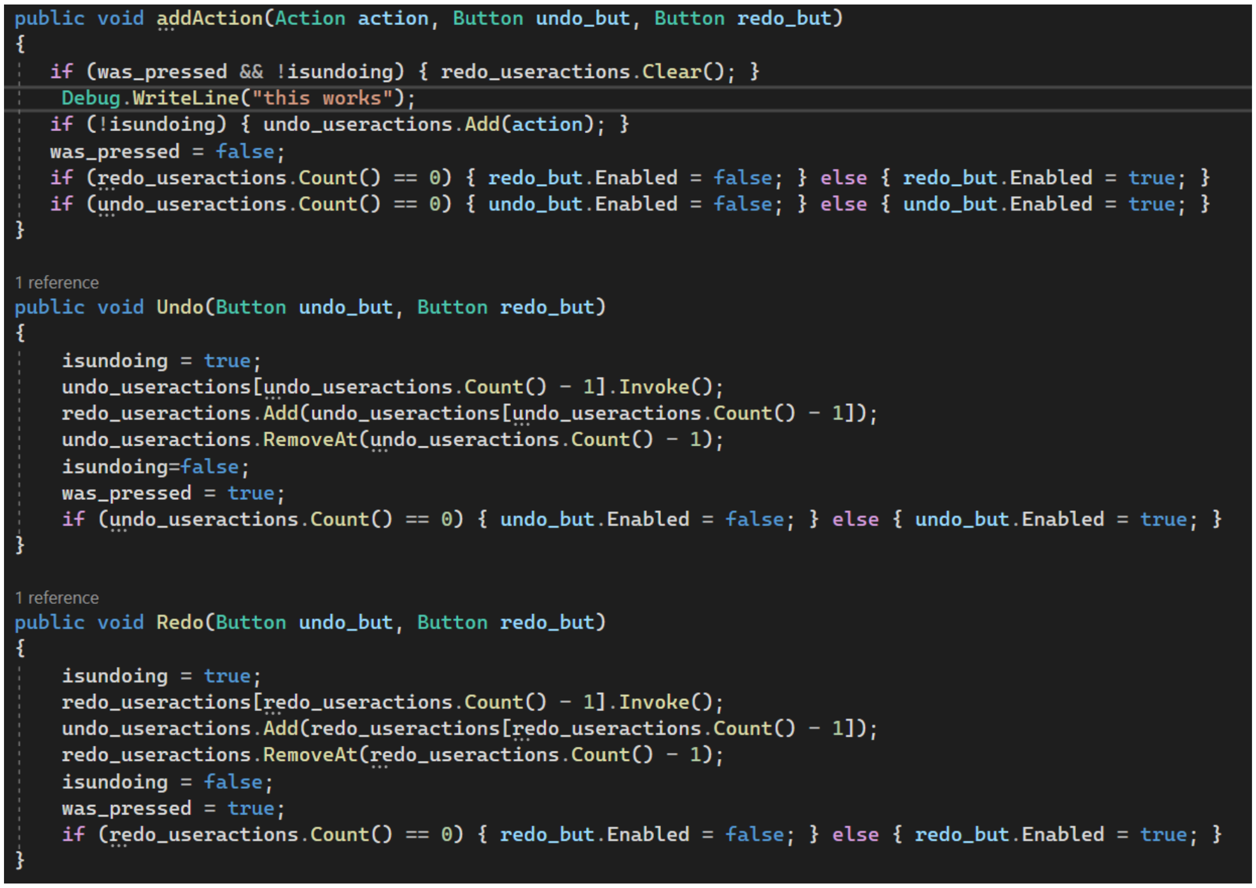
```
s.BookStatus=false;
HashSet<Seat> checkedSeats = new HashSet<Seat>();
List<Seat> bookedseats = new List<Seat>();
change_colour(seatnode, Color.AliceBlue, person);

void check_ifbooked(Node seatnode)
{
    Seat seat = seatnode.Seat;
    if (checkedSeats.Contains(seat))
    {
        return;
    }
    checkedSeats.Add(seat);

    if (seat.BookStatus)
    {
        inner_check_fn(seatnode);
        bookedseats.Add(seat);
        change_colour(seatnode,person.form3_colour, person, person);
    }
}
```

Figure 21. HashSet implementation

## The Undo and Redo buttons

In the beginning, I tried to do the undo redo method with a single list of actions, user_actions. I kept track of the number of times a user presses the undo or redo button with undo_count. If a user pressed the undo button, the code will do user_actions[^undo_count] and undo_count++. Similarly if the user presses the redo button, the code will do user_actions[^undo_count-1] and undo_count--. Unfortunately, even though this made sense to me, this did not work. I tried to use a single list until I realised I was strapped for time.

So I resolved this issue by using two lists of actions instead. I added an Action to the undo list every time an action was done. When the undo button was pressed, I invoked the action and popped it out of the undo list into the redo list and vice versa.



```
public void addAction(Action action, Button undo_but, Button redo_but)
{
   if (was_pressed && !isundoing) { redo_useractions.Clear(); }
    Debug.WriteLine("this works");
   if (!isundoing) { undo_useractions.Add(action); }
   was_pressed = false;
   if (redo_useractions.Count() == 0) { redo_but.Enabled = false; } else { redo_but.Enabled = true; }
   if (undo_useractions.Count() == 0) { undo_but.Enabled = false; } else { undo_but.Enabled = true; }
}

1 reference
public void Undo(Button undo_but, Button redo_but)
{
    isundoing = true;
    undo_useractions[undo_useractions.Count() - 1].Invoke();
    redo_useractions.Add(undo_useractions[undo_useractions.Count() - 1]);
    undo_useractions.RemoveAt(undo_useractions.Count() - 1);
    isundoing=false;
    was_pressed = true;
    if (undo_useractions.Count() == 0) { undo_but.Enabled = false; } else { undo_but.Enabled = true; }
}

1 reference
public void Redo(Button undo_but, Button redo_but)
{
    isundoing = true;
    redo_useractions[redo_useractions.Count() - 1].Invoke();
    undo_useractions.Add(redo_useractions[redo_useractions.Count() - 1]);
    redo_useractions.RemoveAt(redo_useractions.Count() - 1);
    isundoing = false;
    was_pressed = true;
    if (redo_useractions.Count() == 0) { redo_but.Enabled = false; } else { redo_but.Enabled = true; }
}
```

Figure 22. Undo and Redo buttons code

## A general challenge

I found this assignment generally challenging because of the complexity of the assignment and the various requirements that had to be fulfilled. It is a fairly huge assignment, bigger than any code I have written before. But this assignment definitely brought me out of my comfort zone and helped me explore many new concepts, such as a code having multiple threads, binary files, double linked list, Actions, winforms on the whole and more.

## Upon Reflection, what I would have done differently going forward.

I would have definitely taken into consideration how the form data could be condensed and read from a binary file before I started coding it. This way, I could have figured a way to reuse the functions in form_functions and Seats to finally populate the form. This would have decreased the lines of code and loops in the whole code in general and more specifically bin_functionalities. I would also have planned how to code the assignment such that I do not have to rely on Actions for my undo and redo functionality. I would have preferred to just know the method that was most recently used and reverse it. However, when I finally got around to coding the undo and redo, adding code that can be reversed would have been inefficient. Currently, the Seats class interacts with the Class objects, the linked list and the form controls. If I separated the data and double linked list from the form controls, I could have reused a lot of the code in the redo and undo code and the binary form code.

# References

GeeksforGeeks. (2020). *Construct a Doubly linked linked list from 2D Matrix*. [online] Available at: https://www.geeksforgeeks.org/construct-a-doubly-linked-linked-list-from-2d-matrix/ [Accessed 26 May 2023].

Siddiqui, T. (2022). *IEnumerable and IEnumerator in C# | CodeGuru.com*. [online] CodeGuru. Available at: https://www.codeguru.com/csharp/ienumerable-ienumerator-c-sharp/#:~:text=What%20is%20the%20IEnumerable%20Interface [Accessed 26 May 2023].

Maze, C. (2022). *IEnumerable in C#*. [online] Code Maze. Available at: https://code-maze.com/csharp-ienumerable/ [Accessed 26 May 2023].

Kanjilal, J. (2022). *Working with Serialization in C#*. [online] CodeGuru. Available at: https://www.codeguru.com/csharp/c-sharp-serialization/#:~:text=Binary%20Serialization%20in%20C%23 [Accessed 26 May 2023].

www.tutorialspoint.com. (n.d.). *What is Binary Serialization and Deserialization in C and how to achieve Binary Serialization in C*. [online] Available at: https://www.tutorialspoint.com/what-is-binary-serialization-and-deserialization-in-chash-and-how-to-achieve-binary-serialization-in-chash [Accessed 26 May 2023].

Thompson, B. (2020). *Serialization and Deserialization in C# with Example*. [online] www.guru99.com. Available at: https://www.guru99.com/c-sharp-serialization.html. [Accessed 26 May 2023].

DEV Community. (2021). *C# 9 - Delegate, Action and Func*. [online] Available at: https://dev.to/moe23/c-9-delegate-action-and-func-13d7 [Accessed 26 May 2023].

GeeksforGeeks. (2019). *How to set the Background color of the Button in C#?* [online] Available at: https://www.geeksforgeeks.org/how-to-set-the-background-color-of-the-button-in-c-sharp/ [Accessed 26 May 2023].

dotnet-bot (n.d.). *Control.Enabled Property (System.Windows.Forms)*. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.control.enabled?view=windowsdesktop-7.0 [Accessed 26 May 2023].

www.tutorialsteacher.com. (n.d.). *C# Class*. [online] Available at: https://www.tutorialsteacher.com/csharp/csharp-class [Accessed 26 May 2023].