**Assignment 1: Control an Elevator - A C# Project**

**ID:1928162**

**Name: Kautilya Reddy Chappidi**

**Aim:**

To develop an application in C# using .NET Core that can simulate the movement of an elevator.

**Design:**

The Application would be used predominantly on a desktop, as such the user should have the flexibility to move the windows to any position on their screen/monitor. The added benefit of such a design is that it would automatically create a Thread for the processes in each form.

As such the application should have three forms. One for the actual Elevator Simulation, the Control Box and Logs respectively.

The Application should also be able to generate the number of floors in the Elevator Simulation window and also generate their corresponding buttons in the Control Box Form.

The Control Box Form should also have two additional buttons to Open and Close the doors.

As in Most Elevators, there is also an audio cue for Opening the door, Closing the door, Going UP and Going Down.

**Technologies Used:**

**Software :**

* Microsoft Visual Studio 2019 (.NET Core)
* DB Browser for SQLite

NuGet Packages :

* System.Data.SQLite.Core
* System.Speech

**Implementation:**

The User should enter the number of floors he/she requires on their building.

A TextBox has been created which can take the input from the user and a button (Submit Button) is used to capture the text within the field. This capture text is a string and has to converted into an int datatype.

There is also another button on the form which prompts the User for the location of the SQLite Database File. This would store the file’s path as a string datatype variable.

Once the Submit Button is clicked, the captured text is converted into an int by using the

*Int.Parse()* method.

This would create a new instance of the Elevator Simulation window, The Control box Window and the Logs Window and hide the existing StartScreen Window,as seen in Code\_Snippet1.

private void btnSubmitNumberofFloors\_Click(object sender, EventArgs e)

        {

            floors = txtBxNumberofFloors.Text;

            try

            {

                if (int.Parse(floors) > 0 && int.Parse(floors) < 126)

                {

                    this.Hide();

                    Logs logForm = new Logs();

                    logForm.Show();

                    ButtonsForm buttonsForm = new ButtonsForm();

                    buttonsForm.numberOfFloors(floors);

                    buttonsForm.SetConnection(conexonString);

                    buttonsForm.Show();

                    Elevator elevatorForm = new Elevator();

                    elevatorForm.numberofFloorArts(floors);

                    elevatorForm.SetConnection(conexonString);

                    elevatorForm.Show();

                }

                else

                {

                    MessageBox.Show("Please Follow the Instructions");

                }

            }

            catch

            {

                MessageBox.Show("Please Follow the Instructions");

            }

        }

Code\_Snippet1

The textbox can take the input of not only numbers but also alphanumerical inputs, which is not desirable. As such a method has been created which forces the TextBox to be blank if there is an undesirable input as seen in Code\_Snippet2

        private void txtBxNumberofFloors\_TextChanged(object sender, EventArgs e)

        {

            double parsedValue;

            if (!double.TryParse(txtBxNumberofFloors.Text,out parsedValue))

            {

                txtBxNumberofFloors.Text = "";

            }

        }

Code\_Snippet2

When The Close Button on the Window is Clicked, it hides the window, but does not close the window.So the Close button’s click method kills the application, As seen in Code\_Snippet3.

private void Startscreen\_FormClosing(object sender, FormClosingEventArgs e)

        {

            Application.Exit();

        }

By Default, the Submit Button will be set to disabled, as the application would run without an connection to the database. Hence the user is forced to clicked the “Connect” button to connect to the database.

Once the Connection with the database is established, it would then enable the Submit Button.

In the Elevator Simulation Screen, the basic template for a floor is to have :

* An Elevator Frame (PictureBox)
* An Elevator Door (PictureBox)
* An Elevator Box (PictureBox)
* A Floor Button (Button)
* A Floor Label (Label)

Based on the input from the number of floors, new instances of the above-mentioned entities are created and their positions are set to an offset which would result in a background which looks very similar to the inner cross section of a building. As seen in [img1]

The images set to the PictureBoxes were made in Adobe Illustrator with a size of 100 pixels x 100 pixels. But the cumulative size of all the above mentioned entities can be directed to the size of the Elevator Frame, which is 260x260pixels. As such the offset is 260 pixels per floor in the Y axis per floor.

The generated buttons for each floor would then be assigned their very own Eventhandler, to connect to their function.

The method of implementing this is detailed below in Code\_Snippet3.

        public void numberofFloorArts(string floornumber)

        {

            /\*elevatorYellowBox.Visible = true\*/;

            numFloors = floornumber;

            for (int i = 1; i < Int32.Parse(numFloors) + 1; i++)

            {

                Label floornum = new Label();

                floornum.Location = new Point(64, 26+i\*260);

                floornum.Size = new Size(131, 45);

                floornum.Font = new Font("Segoe UI", 24F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);

                floornum.Padding = new System.Windows.Forms.Padding(0);

                floornum.Margin = new System.Windows.Forms.Padding(0);

                floornum.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(64)))), ((int)(((byte)(64)))), ((int)(((byte)(64)))));

                floornum.ForeColor = System.Drawing.SystemColors.Control;

                floornum.AutoSize = false;

                floornum.TextAlign = ContentAlignment.MiddleCenter;

                floornum.Text = i.ToString();

                Button btnCall = new Button();

                btnCall.Location = new Point(12, 150+i\*260);

                btnCall.Height = 23;

                btnCall.Width = 35;

                btnCall.BackColor = System.Drawing.Color.Maroon;

                btnCall.ForeColor = System.Drawing.Color.Maroon;

                btnCall.Text = i.ToString();

                btnCall.Click += new EventHandler(floorCall);

                PictureBox floorbg = new PictureBox();

                floorbg.Height = 260;

                floorbg.Width = 260;

                floorbg.Image = Properties.Resources.elevator\_front;

                floorbg.Margin = new System.Windows.Forms.Padding(0);

                floorbg.Padding = new System.Windows.Forms.Padding(0);

                floorbg.SizeMode = PictureBoxSizeMode.StretchImage;

                floorbg.Location = new Point(0, i\*260);

                PictureBox elevatordoor = new PictureBox();

                elevatordoor.Height = 156;

                elevatordoor.Width = 131;

                elevatorDoor.Margin = new System.Windows.Forms.Padding(0);

                elevatorDoor.Padding = new System.Windows.Forms.Padding(0);

                elevatordoor.Location = new Point(64, 104+(i\*260));

                elevatordoor.SizeMode = PictureBoxSizeMode.StretchImage;

                elevatordoor.Image = Properties.Resources.elevator\_left\_door;

                scrollingPanel.Controls.Add(btnCall);

                scrollingPanel.Controls.Add(floornum);

                scrollingPanel.Controls.Add(floorbg);

            }

        }

Code\_Snippet3

The maximum desktop size handled by Windows 10 is 32768 pixels. As such the number of floors that can be generated are limited to 125 floors.

The Elevator Door and the Elevator Box are placed at the same location duding Instantiation. i.e 104 pixels from the top. As such the position of the Elevator Box can written as a function of 104 + 260\*i. Where “i” is number of each floor for ever floor.

The current position of the elevator is saved as a variable current\_postion, which is the Y Co-ordinate of the Elevator Box/Door. From this, the target position is calculated. The target floor is retrieved from the Text Property of the button.

If the difference between the target floor and the current floor is a positive integer, then the elevator is supposed to go down, due to design of the elevator and if the difference is a negative integer then it is supposed to move up. The animations are initiated by Starting the respective timers as seen in Code\_Snippet4

public void floorCall(object sender, EventArgs e)

        {

            if(isboxdoorOpen == false)

            {

                if (isboxMoving == false)

                {

                    Button btn = sender as Button;

                    isboxMoving = true;

                    target\_floor = int.Parse(btn.Text);

                    new\_position = 104 + (target\_floor \* 260);

                    current\_position = Elevator.ElevatorInstance.elevBox.Location.Y;

                    current\_floor = (current\_position - 104) / 260;

                    dummy = current\_position;

                    var timeString = DateTime.Now.ToString("t");

                    string date = DateTime.UtcNow.ToString("MM-dd-yyyy");

                    ButtonsForm.ButtonsFormInstance.floor\_indicator.Text = btn.Text;

                    ButtonsForm.ButtonsFormInstance.openButton.Enabled = true;

                    ButtonsForm.ButtonsFormInstance.closeButton.Enabled = true;

                    string txtQuery = "INSERT INTO elevator\_log (Date,Time,CurrentFloor, TargetFloor, IsDoorOpen, CurrentFloorPosition, TargetFloorPosition) VALUES ('" + date + "','" + timeString + "','" + current\_floor + "','" + target\_floor + "','" + isboxdoorOpen + "','" + current\_position + "','" + new\_position + "')";

                    ExecuteQuery(txtQuery);

                    LoadData();

                    //string query = "INSERT INTO"

                    if ((new\_position - current\_position) > 0)

                    {

                        SpeechSynthesizer say = new SpeechSynthesizer();

                        say.Volume = 100;

                        say.Rate = 2;

                        say.Speak("Going Down");

                        ButtonsForm.ButtonsFormInstance.down.Visible = true;

                        ButtonsForm.ButtonsFormInstance.up.Visible = false;

                        ButtonsForm.ButtonsFormInstance.openButton.Enabled = false;

                        ButtonsForm.ButtonsFormInstance.closeButton.Enabled = false;

                        Elevator.ElevatorInstance.tmr3.Start();

                    }

                    else if ((new\_position - current\_position) < 0)

                    {

                        SpeechSynthesizer say = new SpeechSynthesizer();

                        say.Volume = 100;

                        say.Rate = 2;

                        say.Speak("Going Up");

                        ButtonsForm.ButtonsFormInstance.down.Visible = false;

                        ButtonsForm.ButtonsFormInstance.up.Visible = true;

                        ButtonsForm.ButtonsFormInstance.openButton.Enabled = false;

                        ButtonsForm.ButtonsFormInstance.closeButton.Enabled = false;

                        Elevator.ElevatorInstance.tmr4.Start();

                    }

                    else

                    {

                        MessageBox.Show("Already at destination");

                    }

                    current\_position = new\_position;

                }

                else

                {

                    MessageBox.Show("Please wait for the elevator to reach destination");

                }

            }

            else

            {

                ButtonsForm.ButtonsFormInstance.openButton.Enabled = true;

                ButtonsForm.ButtonsFormInstance.closeButton.Enabled = true;

                MessageBox.Show("This is Health Hazard\nPlease Close the door");

            }

The animations for handling the Movements of the Elevator Box are handled by timers. There 4 total timers incorporated as seen below in Code\_Snippet4.

The respective Ticks of the timers include how they are supposed to move. The animation can be stopped by using the Stop() method which is integrated into the timers function as seen in Code\_Snippet3. The speed of the animation can be set by changing the time interval in the property menu.

        private void tmrElevatorUp\_Tick(object sender, EventArgs e)

        {

            Elevator.ElevatorInstance.elevBox.Location = new Point(Elevator.ElevatorInstance.elevBox.Location.X, dummy++);

            if (Elevator.ElevatorInstance.elevBox.Location.Y >= new\_position)

            {

                Elevator.ElevatorInstance.tmr3.Stop();

                Elevator.ElevatorInstance.elevBox.Location = new Point(Elevator.ElevatorInstance.elevBox.Location.X, new\_position);

                Elevator.ElevatorInstance.elevDoor.Location = new Point(Elevator.ElevatorInstance.elevDoor.Location.X, new\_position);

                ButtonsForm.ButtonsFormInstance.openButton.Enabled = true;

                ButtonsForm.ButtonsFormInstance.closeButton.Enabled = true;

                ButtonsForm.ButtonsFormInstance.down.Visible = false;

                ButtonsForm.ButtonsFormInstance.up.Visible = false;

                isboxMoving = false;

            }

        }

        private void tmrElevatorDown\_Tick(object sender, EventArgs e)

        {

            Elevator.ElevatorInstance.elevBox.Location = new Point(Elevator.ElevatorInstance.elevBox.Location.X, dummy--);

            //System.Diagnostics.Debug.WriteLine(Elevator.ElevatorInstance.elevBox.Location);

            if (Elevator.ElevatorInstance.elevBox.Location.Y <= new\_position)

            {

                Elevator.ElevatorInstance.tmr4.Stop();

                Elevator.ElevatorInstance.elevBox.Location = new Point(Elevator.ElevatorInstance.elevBox.Location.X, new\_position);

                Elevator.ElevatorInstance.elevDoor.Location = new Point(Elevator.ElevatorInstance.elevDoor.Location.X, new\_position);

                ButtonsForm.ButtonsFormInstance.openButton.Enabled = true;

                ButtonsForm.ButtonsFormInstance.closeButton.Enabled = true;

                ButtonsForm.ButtonsFormInstance.down.Visible = false;

                ButtonsForm.ButtonsFormInstance.up.Visible = false;

                isboxMoving = false;

            }

        }

public void doorCloseAnimation(object sender, EventArgs e)

        {

            if (isboxdoorOpen == true)

            {

                j--;

                Elevator.ElevatorInstance.pBox1.Location = new Point(j, Elevator.ElevatorInstance.pBox1.Location.Y);

                if (Elevator.ElevatorInstance.pBox1.Location.X == initialDoorPosOpen)

                {

                    Elevator.ElevatorInstance.tmr2.Stop();

                    j = initialDoorPosClose;

                    ButtonsForm.ButtonsFormInstance.closeButton.Enabled = true;

                    ButtonsForm.ButtonsFormInstance.openButton.Enabled = true;

                    ButtonsForm.ButtonsFormInstance.flower.Enabled = true;

                    isboxdoorOpen = false;

                }

            }

            else

            {

                Elevator.ElevatorInstance.tmr2.Stop();

                MessageBox.Show("The Door is already Open\n CLOSE the door first");

            }

        }

        public void doorOpenAnimation(object sender, EventArgs e)

        {

            if (isboxdoorOpen == false)

            {

                i++;

                Elevator.ElevatorInstance.pBox1.Location = new Point(i, Elevator.ElevatorInstance.pBox1.Location.Y);

                if (Elevator.ElevatorInstance.pBox1.Location.X == initialDoorPosClose)

                {

                    Elevator.ElevatorInstance.tmr1.Stop();

                    i = initialDoorPosOpen;

                    ButtonsForm.ButtonsFormInstance.closeButton.Enabled = true;

                    ButtonsForm.ButtonsFormInstance.openButton.Enabled = true;

                    ButtonsForm.ButtonsFormInstance.flower.Enabled = true;

                    isboxdoorOpen = true;

                }

            }

            else

            {

                Elevator.ElevatorInstance.tmr1.Stop();

                MessageBox.Show("The Door is Already CLOSED\n OPEN the Door");

            }

        }

Code\_Snippet4

The Database whose connection string is retrieved from the start screen is then setup in 3 functions to make the code look as a simple instantiation as seen Code\_Snippet5.

private SQLiteConnection sql\_con;

        private SQLiteCommand sql\_cmd;

        private SQLiteDataAdapter DB;

        private DataSet DS = new DataSet();

        private DataTable DT = new DataTable();

        public string connectionstring;

        public void SetConnection(string s)

        {

            connector = s;

            sql\_con = new SQLiteConnection(@"Data Source=" + s);

        }

        private void ExecuteQuery(string txtQuery)

        {

            SetConnection(connector);

            sql\_con.Open();

            sql\_cmd = sql\_con.CreateCommand();

            sql\_cmd.CommandText = txtQuery;

            sql\_cmd.ExecuteNonQuery();

            sql\_con.Close();

        }

        private void LoadData()

        {

            SetConnection(connector);

            sql\_con.Open();

            sql\_cmd = sql\_con.CreateCommand();

            string CommandText = "SELECT \* FROM elevator\_log";

            DB = new SQLiteDataAdapter(CommandText, sql\_con);

            DS.Reset();

            DB.Fill(DS);

            DT = DS.Tables[0];

            Logs.LogsInstance.data.DataSource = DT;

            sql\_con.Close();

        }

Code\_Snippet5

The instantiation of these functions from Code\_Snippet5 can be seen in Code\_Snippet3.

The VoiceSynthesizer library is used to make the familiar sounds like going up, going down, opening the door, closing the door, etc. An example of its implementation is seen below in Code\_Snippet6.

                        SpeechSynthesizer say = new SpeechSynthesizer();

                        say.Volume = 100;

                        say.Rate = 2;

                        say.Speak("Going Up");

Code\_Snippet6.

**Known Issues:**

1. The timers of the Elevator moving up and down are unaffected by the Interval values.
2. In StartScreen, when the Connect button is clicked ,no file is selected, the Submit Button is Clicked. There are unhandled exceptions.
3. Certain Menustrip Items are not connected

**Testing:**

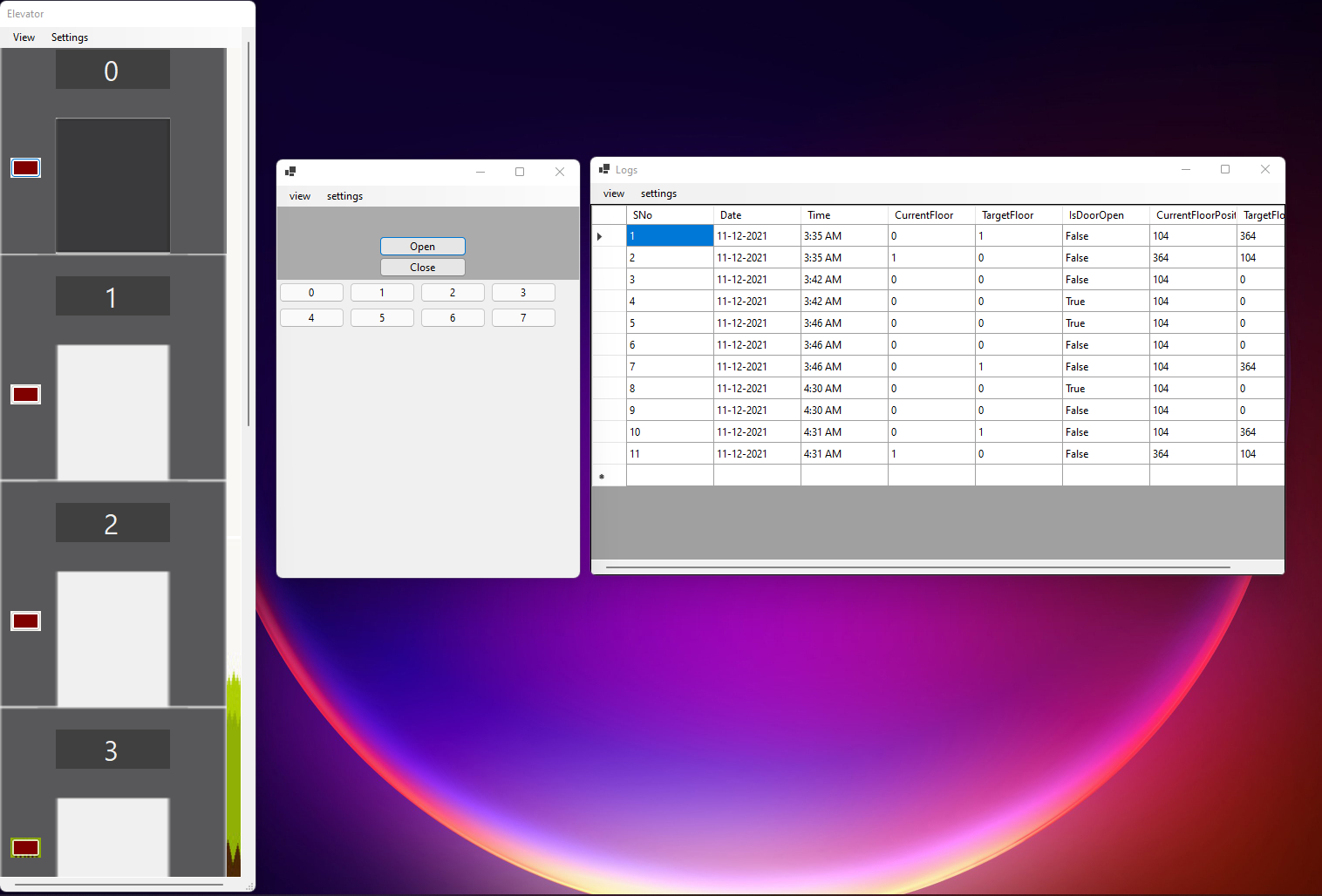
****

Figure : Generating 7 Floors

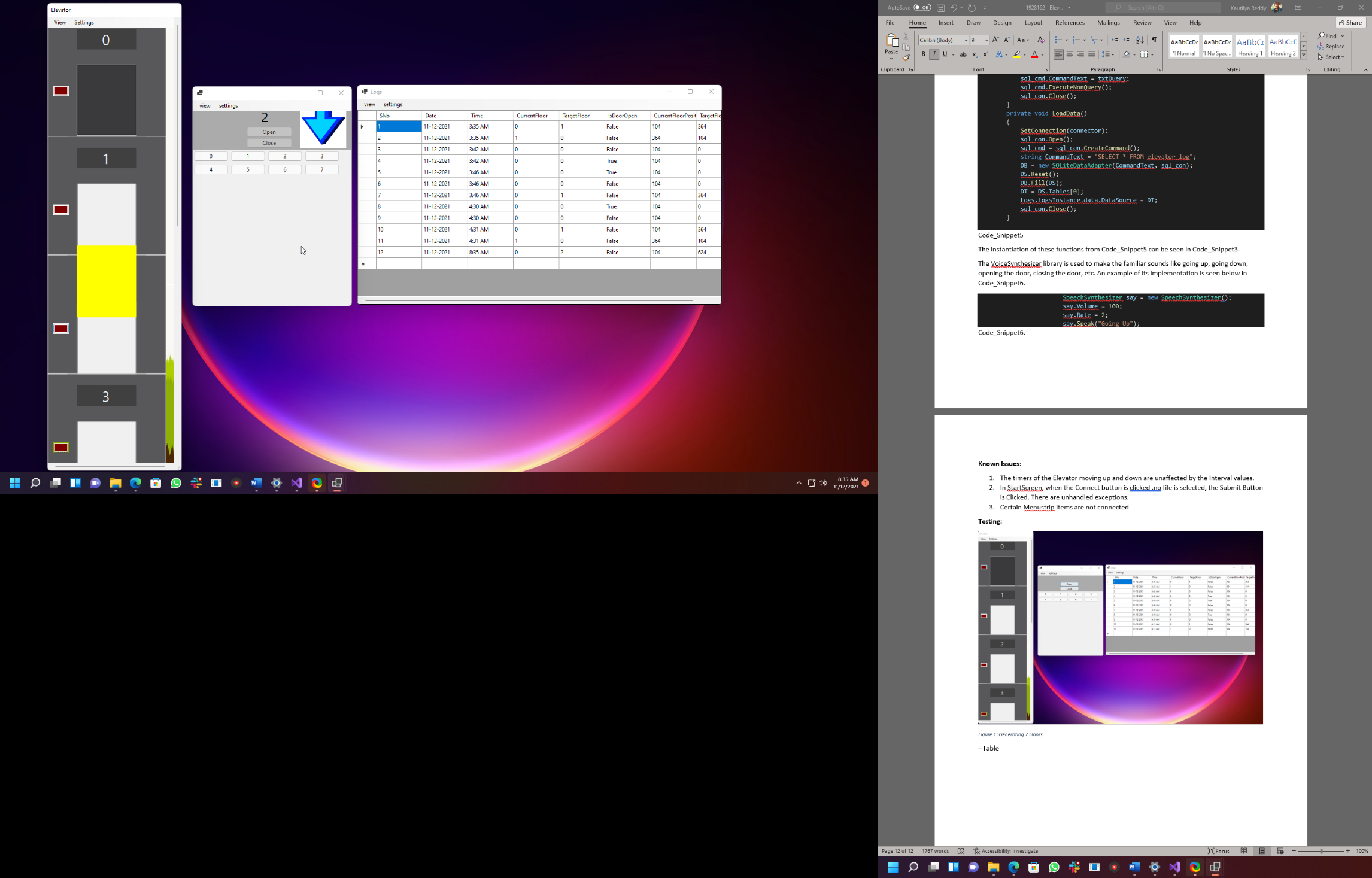


Figure Elevator Going Down and Entry Added to DataBase

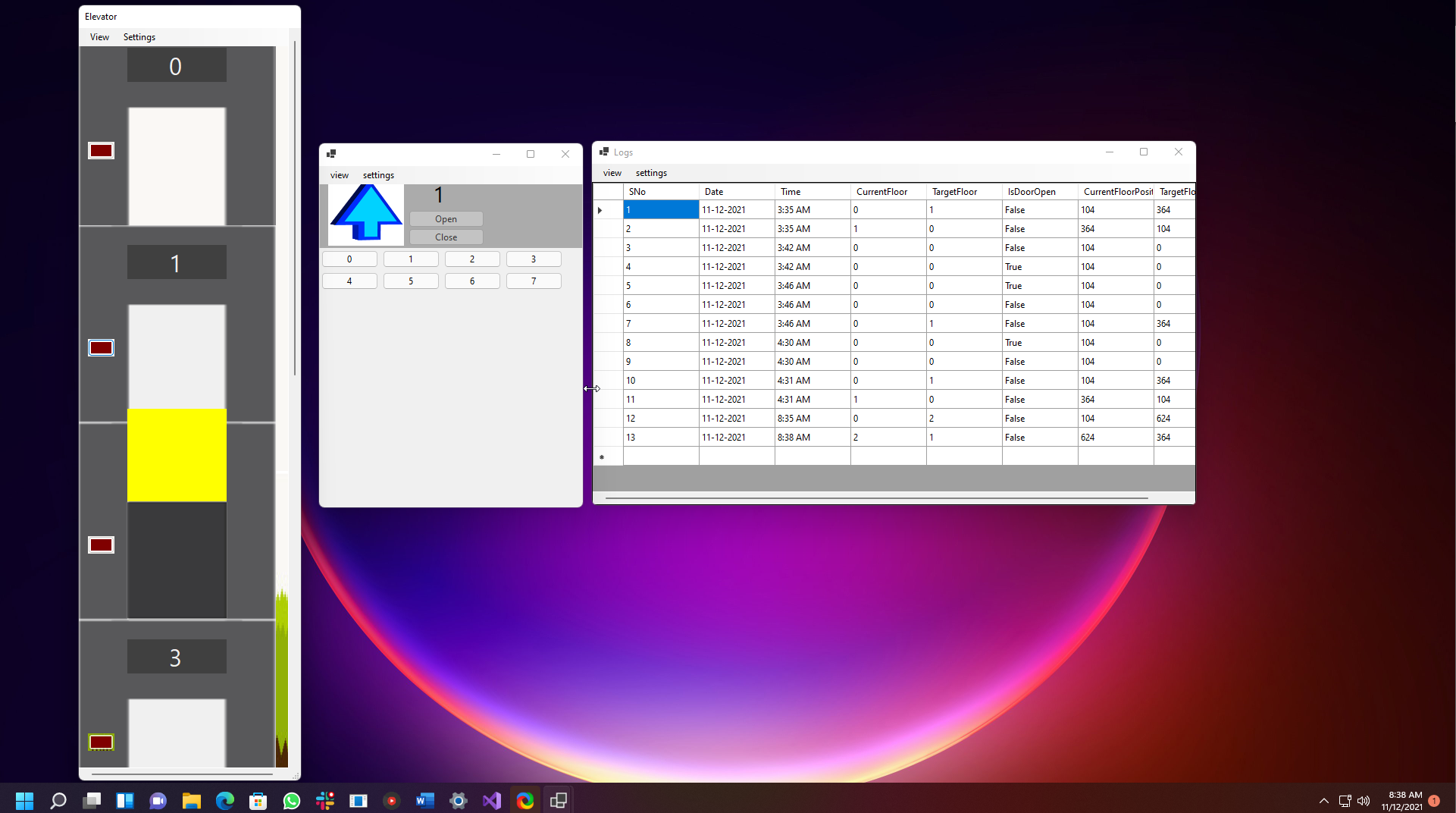


Figure Elevator Going Up and Entry Added to Database

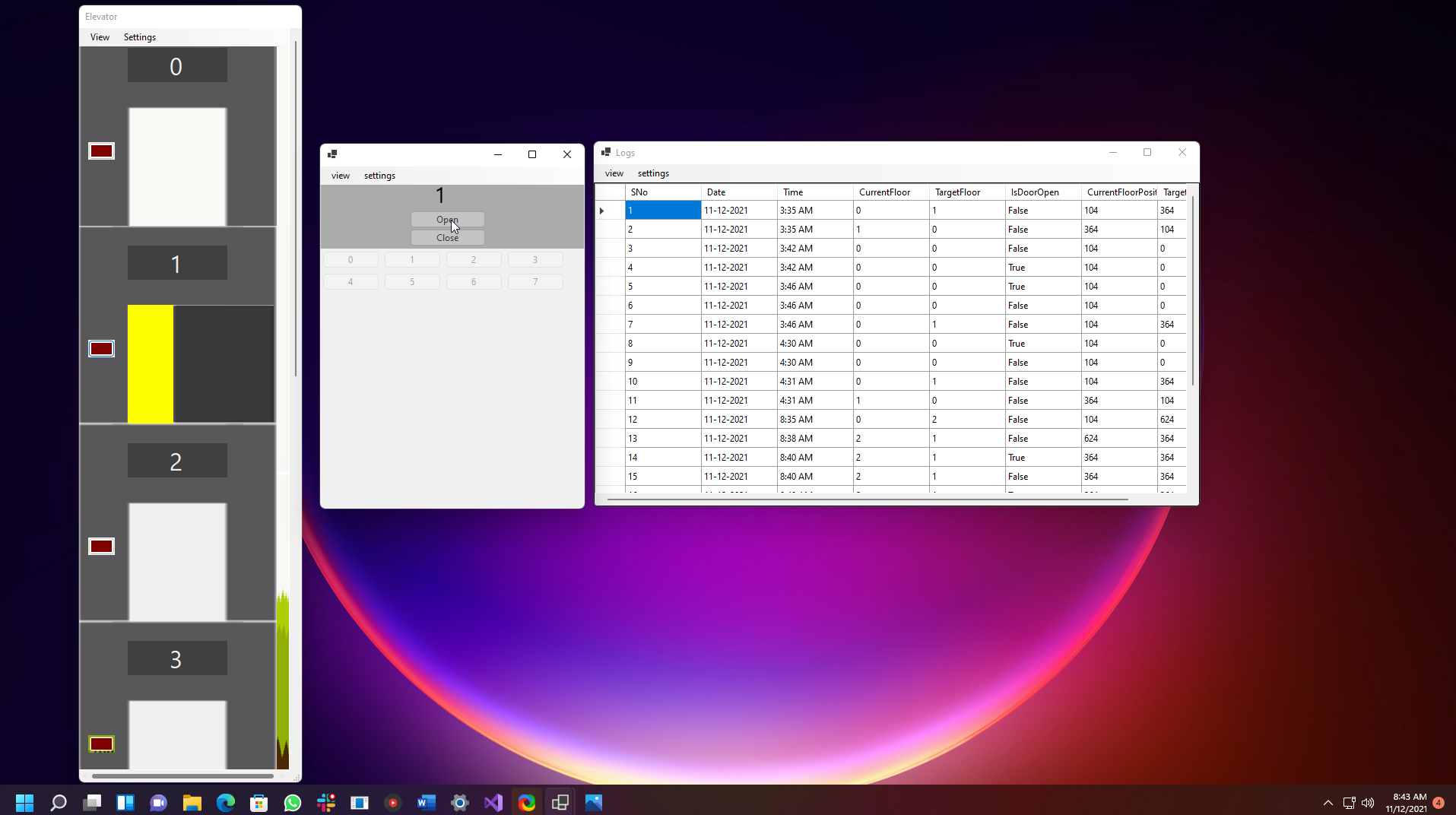


Figure Opening the Door and Entry Added to Database

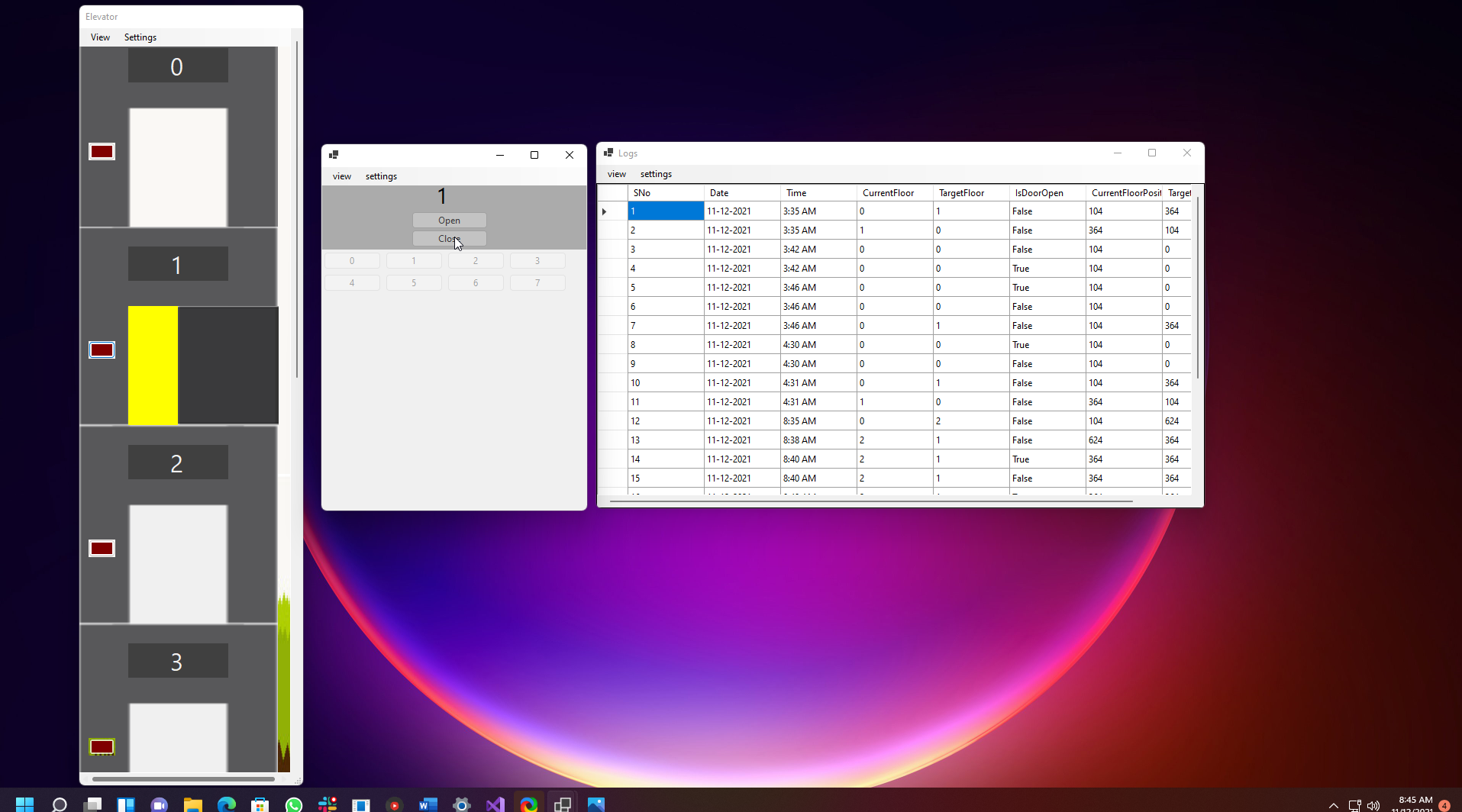


Figure Closing the Door and Entry Added to Database

**Matrix Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task Number** | **Sub-tasks** | **Possible Marks** | **Self-assessment (completed Yes/No)** |
| **Task 1** | Complete GUI for Task 1 | 10 | Yes |
| Skeleton of event handlers in place for all buttons | 10 | Yes |
| **Task 2** | All event handlers are functional | 10 | Yes |
| **Task 3** | Database (DB) is designed and can be connected | 5 | Yes |
| Log Information can be retrieved from DB and displayed in the GUI | 5 | Yes |
| When the log button is pressed, log information is sent to and stored in the DB | 5 | Yes |
| Use the disconnected model rather than connected model (Data source is updated via DataAdapters Update() method instead of ExecuteNonQuery() method) | 5 | Yes |
| **Task 4** | Events described in Task 2 animated using delegation and timer | 10 | Yes |
| **Task 5** | Using relative path instead of absolute path | 5 | Yes |
| Avoiding any duplication among the event handlers over the database related functions | 5 | No (Partial) |
| Eliminating logical errors and handling exceptions with try and catch | 5 | Yes |
| Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker | 5 | No(Partial) |
| Use state patterns instead of if-else statements to accommodate future changes of the requirement | 10 | No |
| **Task 6** | Test report | 10 |  |
| **Total** |  | 100 |  |