Object-oriented programming and software engineering

CIS016-2, Assignment 1: Control an Elevator – A C# Project

**University of Bedfordshire, 2017-2018**

Faisal Hoque – 1618196

2017/2018

# Aim

According to the assignment brief: “This assignment aims to develop a students’ capabilities and skills of solving real-world problems with C# programming language.”

To my understanding the assignment is designed to make me think critically about the task description and design a creative solution to the task described; this is aimed to be achieved with my current skills and with any extra skills I may have gained during the practical sessions.

# Task Description

The task to my understanding was to build an elevator for a company consisting of two floors, a ground and first floor, ground being floor 0 and first being floor 1. The company wanted the elevator to consist of an indoor car, which is what moves up and down, and a control panel consisting of two buttons referring to the ground floor and the first floor, the car would also be controllable from the outside with a request button on each corresponding floor.

Each floor also required floor doors, so users could not fall down the elevator shaft when there is no elevator on the corresponding floor. The doors would also need to open as the elevator car reaches the required floor. All buttons required a built-in light that would highlight when a request or control panel button is pressed. If a button on the control panel is pushed while the doors are open, then they will start to close instantly rather than waiting on a delay and the car shall then move to the floor requested.

The assignment brief scheduled a few tasks to help demonstrate the functionality of the elevator. The first task included creating a Guided User Interface (GUI) that contains: Two request buttons, one control panel with two extra buttons and a display window, two display areas for the status of the elevator and a button that logs information about each button pushed.

The second task was to process the events published by each button from the GUI, so every time the request buttons or control panel buttons are pushed then a list or table will be updated with the corresponding events, alongside making the elevator move up or down according to the buttons requested. The third task requires me to animate the process of the elevator moving up/down in task two.

The fourth and fifth task were to log all the data processed by the buttons into a Microsoft Access database and show the log on to the GUI. Task six was to integrate all the tasks from 1-5 and task seven is to write a report about the aim, task description, design, programs and comments and testing results.

# Design

The design for the elevator program was quite an interesting one, as even though I have been in a lot of elevators and used them I don’t accurately remember the critical parts of the elevator, except the fact that I can push buttons and it will take me to the floor I requested.

To get a good understanding of what I wanted to create, I first did a little research on google to see how an elevator may look etc. and watched a few videos on it working. I then planned to visit the town mall and universities elevator to get a better understanding however, beforehand I was a little eager, so I coded a rough scratch on how I wanted the elevator to look like.

## Elevator Prototype Code

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Windows**.**Forms**;**

**using** System**.**Data**.**OleDb**;**

**namespace** DB\_Forms\_Prac\_4

**{**

**public** **partial** class Elevator **:** Form

**{**

**private** const int DOOR\_OPEN **=** 5**;**

// Start y = 65;

**private** double x**,** y**,** top1Door**,** top2Door**,** bot1Door**,** bot2Door**;**

**private** bool moveUp**,** moveDown**,** botOpen**,** topOpen**,** opening**,** botRqst**,** topRqst**,** error**;**

**private** int timeOpen**;**

**public** Elevator**()**

**{**

InitializeComponent**();**

**}**

**private** void Elevator\_Load**(object** sender**,** EventArgs e**)**

**{**

x **=** 140**;** y **=** 65**;**

top1Door **=** bot1Door **=** 140**;**

top2Door **=** bot2Door **=** 290**;**

moveUp **=** moveDown **=** botOpen **=** topOpen **=** botRqst **=** topRqst **=** error **=** **false;**

timeOpen **=** 0**;**

label1**.**Text **=** "Top: " **+** moveUp**.**ToString**()** **+** ", Bot: " **+** moveDown**.**ToString**();**

retrieveLog**();**

**}**

**private** void btnAlarm\_Click**(object** sender**,** EventArgs e**)**

**{**

error **=** **true;**

lblDisplayCP**.**Text **=** lblDisplayBot**.**Text **=** lblDisplayTop**.**Text **=** "!!!!!"**;**

timeOpen **=** 505**;**

eleTimer**.**Enabled **=** **true;**

string datetime **=** DateTime**.**Today**.**ToString**(**"d"**)** **+** " " **+** DateTime**.**Now**.**ToString**(**"HH:mm:ss tt"**);**

**if(**error**)**

**{**

submitLog**(**datetime**,** "Emergency Alarm button was pressed. !EMERGENCY MODE!"**);**

**}**

**else**

**{**

submitLog**(**datetime**,** "Emergency Alarm button was pressed."**);**

**}**

**}**

**private** void btnOpen\_Click**(object** sender**,** EventArgs e**)**

**{**

**if(**y **>=** 445**)**

**{**

botRqst **=** **true;**

eleTimer**.**Enabled **=** **true;**

**}**

**if(**y **<=** 65**)**

**{**

topRqst **=** **true;**

eleTimer**.**Enabled **=** **true;**

**}**

string datetime **=** DateTime**.**Today**.**ToString**(**"d"**)** **+** " " **+** DateTime**.**Now**.**ToString**(**"HH:mm:ss tt"**);**

**if(**error**)**

**{**

submitLog**(**datetime**,** "The button to open the doors was pressed. !EMERGENCY MODE!"**);**

**}**

**else**

**{**

submitLog**(**datetime**,** "The button to open the doors was pressed."**);**

**}**

**}**

**private** void btnClose\_Click**(object** sender**,** EventArgs e**)**

**{**

**if(**botOpen **||** topOpen**)**

**{**

timeOpen **=** 5**;**

eleTimer**.**Enabled **=** **true;**

**}**

string datetime **=** DateTime**.**Today**.**ToString**(**"d"**)** **+** " " **+** DateTime**.**Now**.**ToString**(**"HH:mm:ss tt"**);**

**if(**error**)**

**{**

submitLog**(**datetime**,** "The button to close the doors was pressed. !EMERGENCY MODE!"**);**

**}**

**else**

**{**

submitLog**(**datetime**,** "The button to close the doors was pressed."**);**

**}**

**}**

**private** void rqstUp\_Click**(object** sender**,** EventArgs e**)**

**{**

**if((**y **<=** 65 **||** y **>=** 445**)** **&&** **!**botRqst**)**

**{**

moveUp **=** topRqst **=** **true;**

eleTimer**.**Enabled **=** **true;**

**}**

label1**.**Text **=** "Top: " **+** moveUp**.**ToString**()** **+** ", Bot: " **+** moveDown**.**ToString**();**

string datetime **=** DateTime**.**Today**.**ToString**(**"d"**)** **+** " " **+** DateTime**.**Now**.**ToString**(**"HH:mm:ss tt"**);**

**if(**error**)**

**{**

submitLog**(**datetime**,** "Request button to first floor was pressed. !EMERGENCY MODE!"**);**

**}**

**else**

**{**

submitLog**(**datetime**,** "Request button to first floor was pressed."**);**

**}**

**}**

**private** void rqstDown\_Click**(object** sender**,** EventArgs e**)**

**{**

**if((**y **<=** 65 **||** y **>=** 445**)** **&&** **!**topRqst**)**

**{**

moveDown **=** botRqst **=** **true;**

eleTimer**.**Enabled **=** **true;**

**}**

label1**.**Text **=** "Top: " **+** moveUp**.**ToString**()** **+** ", Bot: " **+** moveDown**.**ToString**();**

string datetime **=** DateTime**.**Today**.**ToString**(**"d"**)** **+** " " **+** DateTime**.**Now**.**ToString**(**"HH:mm:ss tt"**);**

**if(**error**)**

**{**

submitLog**(**datetime**,** "Request button to ground floor was pressed. !EMERGENCY MODE!"**);**

**}**

**else**

**{**

submitLog**(**datetime**,** "Request button to ground floor was pressed."**);**

**}**

**}**

**private** void doorTimer\_Tick**(object** sender**,** EventArgs e**)**

**{**

**if** **(**botOpen **||** topOpen**)**

**{**

**if** **(**timeOpen **<** 5**)**

**{**

timeOpen**++;**

**}**

**}**

**}**

**private** void eleTimer\_Tick**(object** sender**,** EventArgs e**)**

**{**

**if** **(**moveUp **&** moveDown**)** moveUp **=** moveDown **=** **false;**

**if** **(**y **<=** 65**)** moveUp **=** **false;**

**if** **(**y **>=** 445**)** moveDown **=** **false;**

**if(!**error**)**

**{**

topDoor**();**

bottomDoor**();**

**}**

**else**

**{**

// Bottom

**if** **(**bot1Door **>=** 0**)** bot1Door**--;**

**if** **(**bot2Door **<=** 430**)** bot2Door**++;**

// Top

**if** **(**top1Door **>=** 0**)** top1Door**--;**

**if** **(**top2Door **<=** 430**)** top2Door**++;**

**}**

Invalidate**();**

**}**

**private** void Elevator\_Paint**(object** sender**,** PaintEventArgs e**)**

**{**

e**.**Graphics**.**FillRectangle**(**Brushes**.**Black**,** 100**,** 25**,** 380**,** 740**);**

// Stop at y = 445.

e**.**Graphics**.**FillRectangle**(**Brushes**.**Gray**,** **(**float**)**x**,** **(**float**)**y**,** 300**,** 300**);**

//e.Graphics.FillRectangle(Brushes.Gray, 140, 445, 300, 300);

// Top floor elevator doors.

e**.**Graphics**.**FillRectangle**(**Brushes**.**Silver**,** **(**float**)**top1Door**,** 65**,** 150**,** 300**);**

e**.**Graphics**.**FillRectangle**(**Brushes**.**Silver**,** **(**float**)**top2Door**,** 65**,** 150**,** 300**);**

// Bottom floor elevator doors.

e**.**Graphics**.**FillRectangle**(**Brushes**.**Silver**,** **(**float**)**bot1Door**,** 445**,** 150**,** 300**);**

e**.**Graphics**.**FillRectangle**(**Brushes**.**Silver**,** **(**float**)**bot2Door**,** 445**,** 150**,** 300**);**

**}**

**private** void bottomDoor**()**

**{**

// If the down/up button has pressed the elevator will move accordingly.

**if** **((**moveDown **&&** y **<=** 445**)** **&&** **!**topOpen**)** y**++;**

// Once the elevator arrives, the movement stops.

**if** **(**y **==** 445**)**

**{**

moveDown **=** **false;**

lblDisplayBot**.**Text **=** "G"**;**

lblDisplayTop**.**Text **=** "G"**;**

lblDisplayCP**.**Text **=** "G"**;**

**}**

// Controls how the doors move, once the elevator reaches the bottom floor

// and doors are closed, the doors then open

**if** **((**y **>=** 445 **&&** bot1Door **>=** 0**)** **&&** **(!**botOpen **&&** botRqst**))** bot1Door**--;**

**if** **((**y **>=** 445 **&&** bot2Door **<=** 430**)** **&&** **(!**botOpen **&&** botRqst**))** bot2Door**++;**

// Once the doors reach the specific opening length, the variale is declared true.

**if** **(**bot1Door **<=** 0 **&&** bot2Door **>=** 430**)** botOpen **=** **true;**

// Doors stay open for 5 seconds, then start to close.

**if** **(**timeOpen **>=** 5 **&&** botOpen**)**

**{**

**if** **(**bot1Door **<=** 140 **&&** botOpen**)** bot1Door**++;**

**if** **(**bot2Door **>=** 290 **&&** botOpen**)** bot2Door**--;**

// Once the doors are closed the variables are declared false.

**if** **(**bot1Door **>=** 140 **&&** bot2Door **<=** 290**)**

**{**

botOpen **=** botRqst **=** **false;**

timeOpen **=** 0**;**

eleTimer**.**Enabled **=** **false;**

**}**

label1**.**Text **=** "Top: " **+** moveUp**.**ToString**()** **+** ", Bot: " **+** moveDown**.**ToString**();**

**}**

**}**

**private** void topDoor**()**

**{**

// If the down/up button has pressed the elevator will move accordingly.

**if** **((**moveUp **&&** y **>=** 65**)** **&&** **!**botOpen**)** y**--;**

// Once the elevator arrives, the movement stops.

**if** **(**y **==** 65**)**

**{**

moveUp **=** **false;**

lblDisplayBot**.**Text **=** "1"**;**

lblDisplayTop**.**Text **=** "1"**;**

lblDisplayCP**.**Text **=** "1"**;**

**}**

// Controls how the doors move, once the elevator reaches the bottom floor

// and doors are closed, the doors then open

**if** **((**y **<=** 65 **&&** top1Door **>=** 0**)** **&&** **(!**topOpen **&&** topRqst**))** top1Door**--;**

**if** **((**y **<=** 65 **&&** top2Door **<=** 430**)** **&&** **(!**topOpen **&&** topRqst**))** top2Door**++;**

// Once the doors reach the specific opening length, the variable is declared true.

**if** **(**top1Door **<=** 0 **&&** top2Door **>=** 430**)** topOpen **=** **true;**

// Doors stay open for 5 seconds, then start to close.

**if** **(**timeOpen **==** 5 **&&** topOpen**)**

**{**

**if** **(**top1Door **<** 140 **&&** topOpen**)** top1Door**++;**

**if** **(**top2Door **>** 290 **&&** topOpen**)** top2Door**--;**

// Once the doors are closed the variables are declared false.

**if** **(**top1Door **>=** 140 **&&** top2Door **<=** 290**)**

**{**

topOpen **=** topRqst **=** **false;**

timeOpen **=** 0**;**

eleTimer**.**Enabled **=** **false;**

**}**

label1**.**Text **=** "Top: " **+** moveUp**.**ToString**()** **+** ", Bot: " **+** moveDown**.**ToString**();**

**}**

**}**

**private** void submitLog**(**string datetime**,** string entry**)**

**{**

OleDbConnection conn **=** **new** OleDbConnection**(**"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Elevator Log.accdb"**);**

OleDbCommand command **=** conn**.**CreateCommand**();**

conn**.**Open**();**

command**.**CommandText **=** "INSERT INTO `Logs` (`EventDate`, `EventEntry`) VALUES('" **+** datetime **+**

"', '" **+** entry **+** "');"**;**

command**.**Connection **=** conn**;**

command**.**ExecuteNonQuery**();**

conn**.**Close**();**

retrieveLog**();**

**}**

**private** void retrieveLog**()**

**{**

lstBox**.**Items**.**Clear**();**

OleDbConnection conn **=** **new** OleDbConnection**(**"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Elevator Log.accdb"**);**

conn**.**Open**();**

OleDbCommand command **=** **new** OleDbCommand**(**"SELECT \* FROM `Logs`"**,** conn**);**

OleDbDataReader reader **=** command**.**ExecuteReader**();**

**while** **(**reader**.**Read**())**

**{**

lstBox**.**Items**.**Add**(**reader**[**"EventDate"**].**ToString**()** **+** " : " **+** reader**[**"EventEntry"**].**ToString**());**

**}**

conn**.**Close**();**

**}**

**}**

**}**

## Brief Prototype Testing

Figure 1: Main screen of the entire elevator prototype, contains the elevator animation, two request buttons, two display windows, a control panel with 5 buttons and 2 display windows and a list box to show the data.

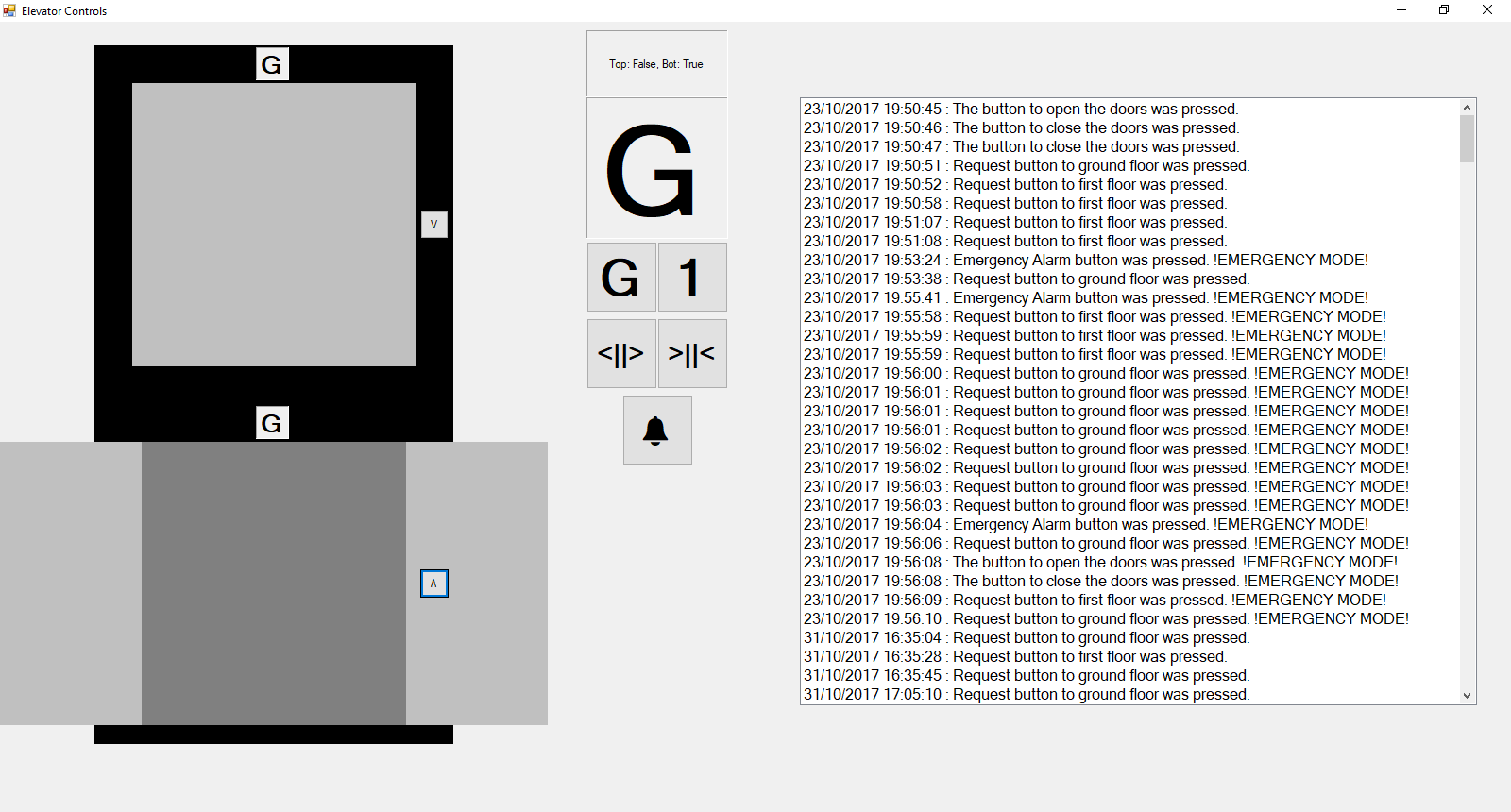
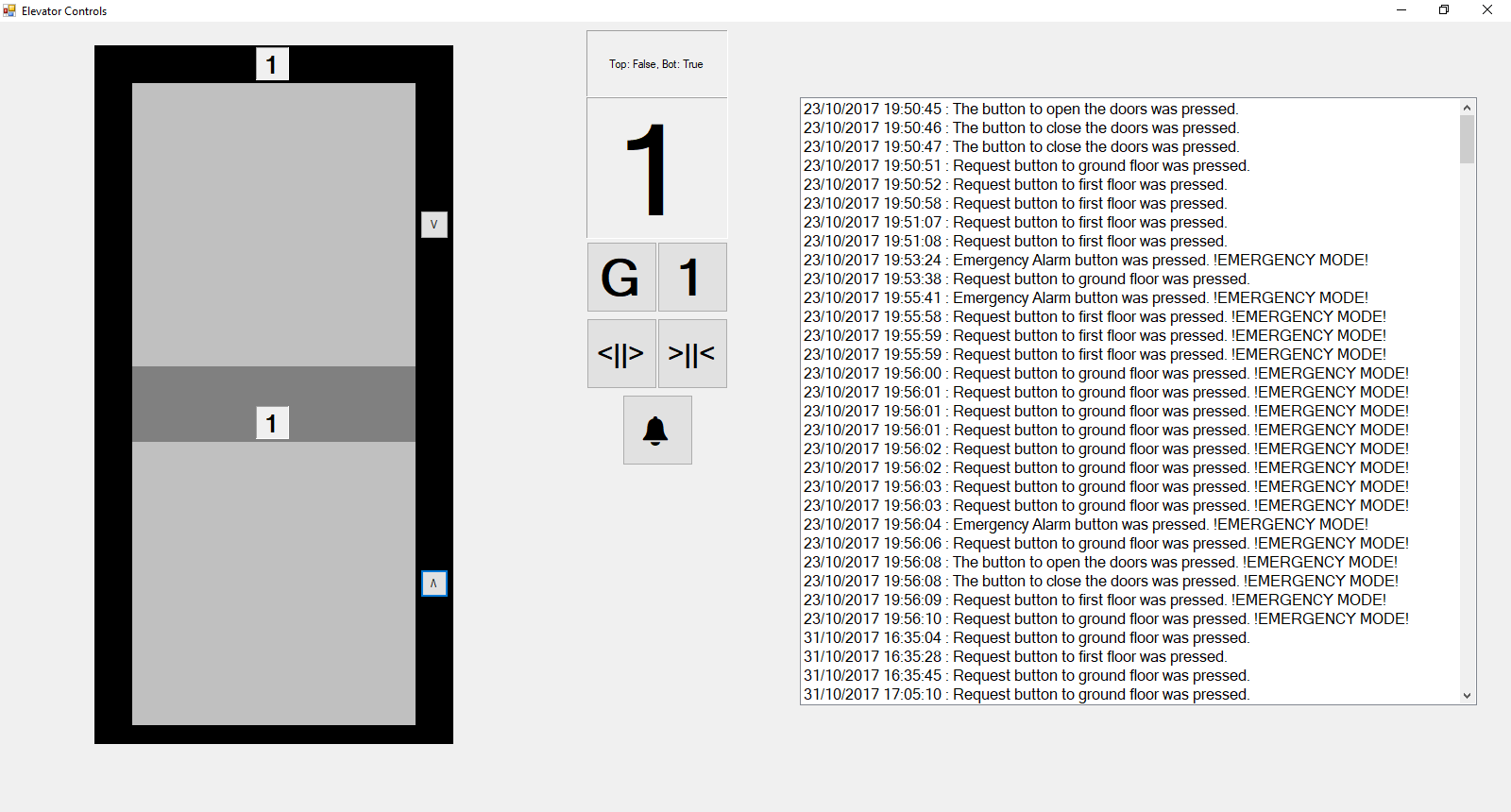
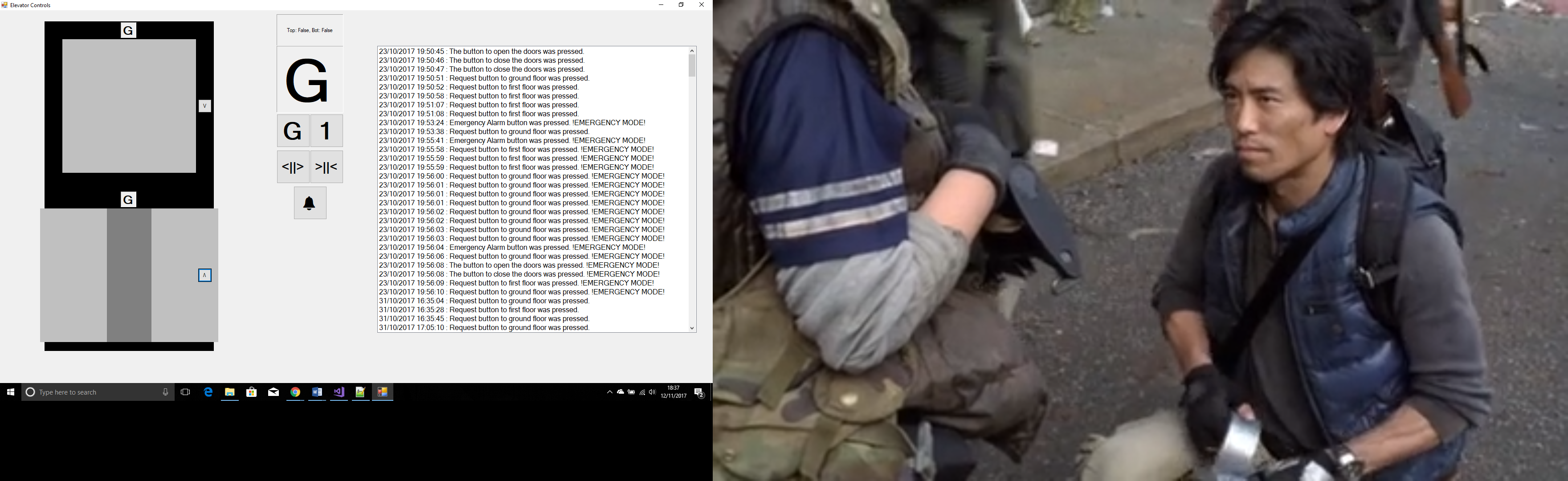


Figure : Shows the elevator doors opening as the elevator moves to ground floor and the displays changing accordingly.

Figure : Shows the elevator moving to the ground floor as the request button was pushed



Figures 2, 3 & 4 all show the grey elevator car moving upon the request of the button, in this case the bottom request button was pushed, the button with the upside-down V. The tests show the elevator does more and correspond accordingly to the buttons.

The bit of code that moves the elevator is quite simple, a variable with the value of the floor you want the elevator to be on is declared such as: **private** double y = 65**;** the line of code here is like what I used to declare the starting point of my elevator. I then declared a method that would contain two lines of code, one that would increase/decrease the value of ‘y’ according to the floor it is moving to. The first method included: **if** **((**moveDown **&&** y **<=** 445**)** **&&** **!**topOpen**)** y**++;** this bit of code states if the request button has been pushed and the elevator is currently below 445 then start increasing the value till you reach 445, which moves it to the ground floor. The second line those the same but for the top floor: **if** **((**moveUp **&&** y **>=** 65**)** **&&** **!**botOpen**)** y**--;** both lines of code will only work if the doors are closed.

Figure : Shows the doors closing after five seconds

## Brief Prototype Discussion

The objective of this prototype was to understand the technologies I would use to develop the elevator program, as I was new to the C# programming in general. I developed the prototype in week 3, and the practicals at the time did not touch on the technologies required for the elevator program so I used this programme as an experimental session.

I was able to successfully connect my prototype to a Microsoft Access database, utilising two methods: one that would insert into the database and the other that would retrieve from the database.

**private** void submitLog**(**string datetime**,** string entry**)**

**{**

OleDbConnection conn **=** **new** OleDbConnection**(**"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Elevator Log.accdb"**);**

OleDbCommand command **=** conn**.**CreateCommand**();**

conn**.**Open**();**

command**.**CommandText **=** "INSERT INTO `Logs` (`EventDate`, `EventEntry`) VALUES('" **+** datetime **+**

"', '" **+** entry **+** "');"**;**

command**.**Connection **=** conn**;**

command**.**ExecuteNonQuery**();**

conn**.**Close**();**

retrieveLog**();**

**}**

**private** void retrieveLog**()**

**{**

lstBox**.**Items**.**Clear**();**

OleDbConnection conn **=** **new** OleDbConnection**(**"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Elevator Log.accdb"**);**

conn**.**Open**();**

OleDbCommand command **=** **new** OleDbCommand**(**"SELECT \* FROM `Logs`"**,** conn**);**

OleDbDataReader reader **=** command**.**ExecuteReader**();**

**while** **(**reader**.**Read**())**

**{**

lstBox**.**Items**.**Add**(**reader**[**"EventDate"**].**ToString**()** **+** " : " **+** reader**[**"EventEntry"**].**ToString**());**

**}**

conn**.**Close**();**

**}**

The two methods are essentially similar, though the difference is submitLog**(**string datetime**,** string entry**)** inserts the data passed into the methods parameters into the Access database, the retrieveLog**()** loops through the entire database and inserts that information into the table to be presented to the user. The methods are very inefficient as it creates multiple objects of the OleDbConnection and commands rather than creating one instance and rewriting the data within the CommandText. As messy as the code is, the code helped me understand the connection to an Access database.

## Next steps and UML design

The prototype helped me understand how I wanted to organise my final elevator programme, with the types of classes I needed and the sketch of how the final design would look like. To help me out I created two UML Diagrams, one being a class diagram and the other being an activity diagram.

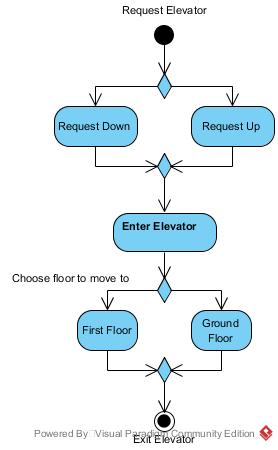
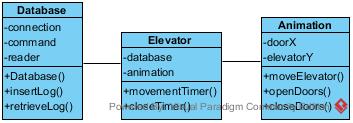


Figure : UML Activity Diagram

Figure : UML Class Diagram

The activity diagram and class diagram were both designed right after the prototype. The activity diagram was designed to see the actions and steps a user of the elevator may take while using the elevator.

The class diagram helped me better understand the way I may object-orient the elevator programme and create the basic variables/methods that was needed to start the basis of the classes.

The steps I took next was to visit some real-life elevators, such as the University and town mall elevators. I had a good look on the outside of the elevators and primarily prioritised on the displays on the outside and some extra features maybe for the inside. Though for some reason I did not actually investigate the inside of the elevator as thoroughly as I probably should have had, which most certainly would have reminded me of a few more features to focus on.

The design of the final elevator was simple, two floors, as the elevator car was 346 pixels in height, I made the length of both floors 692 pixels in height, so the first floor is at location (0, 0) and the ground floor is at location (0, 346). For the prototype I used the paint and draw methods to create the elevators animations however that was very glitchy, lagging and inefficient. For the new one I wanted to change the location of a label instead of a drawing, to do this I planned to utilise the label**.**Location **=** **new** Point**(**x**,** 0**);** methods that labels have built in with them, this method will change the position of the label according to the new value specified.

The part where when the elevator is requested to the first floor and then requested to the ground floor, the elevator had to move automatically without the user pressing the request button again, was probably the trickiest part. As in the prototype I had not actually implemented that feature, so when a request button is pushed the elevator will go to the corresponding floor, however if a second request button was pushed while the elevator was moving it would not go to that floor after the elevator had met its destination. The way I implemented this feature was utilising an array and simply cleaning up the code, since the prototype code was extremely messy in the final programme I had to organise the code a lot more which allowed me to utilise less features and be a lot more efficient in CPU usage.

# Programme Code

## Elevator.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Windows**.**Forms**;**

**namespace** Elevator

**{**

**public** **partial** class Elevator **:** Form

**{**

// Objects of the Database and Movement class.

**private** Database database**;**

**private** Movement movement**;**

// Integer variable to delay the closing of doors.

**private** int count**;**

// Constructor to initialise the GUI.

**public** Elevator**()**

**{**

InitializeComponent**();**

**}**

// Method to initialise variables/objects when the GUI loads.

**private** void Elevator\_Load**(object** sender**,** EventArgs e**)**

**{**

// Initialise the objects and provides this specific class as a parameter, to be using within the other classes.

database **=** **new** Database**(this);**

movement **=** **new** Movement**(this);**

// Sets the initial value of the delay.

count **=** 3**;**

// Retrieves all the content of the database and uploads them to the table.

database**.**databaseEvent**(**"Retrieve"**,** **null);**

**}**

// Method that handles the click of the button that requests to the ground floor.

**private** void rqstGround\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "Request to ground floor button pushed."**);**

// Inserting 'Down' into the list of 'direction' to make the elevator move down.

movement**.**direct**(**"Down"**);**

// Change the colour of the buttons text to notify the user that it's been pressed.

rqstGround**.**ForeColor **=** Color**.**LightGreen**;**

**}**

// Method that handles the request button for the first floor.

**private** void rqstFirst\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "Request to first floor button pushed."**);**

// Inserting 'Up' into the list of 'direction' to make the elevator move up.

movement**.**direct**(**"Up"**);**

// Change the colour of the buttons text to notify the user that it's been pressed.

rqstFirst**.**ForeColor **=** Color**.**LightGreen**;**

**}**

// Method that handles the control panel button for the ground floor.

**private** void btnGround\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "Ground button has been pushed."**);**

// Inserting 'Up' into the list of 'direction' to make the elevator move down.

movement**.**direct**(**"Down"**);**

// Checks if the door is open, if so then forces the doors to close.

**if** **((**topLeft**.**Location**.**X **<=** **-**1 **&&** topRight**.**Location**.**X **>=** 122**)** **||** **(**botLeft**.**Location**.**X **<=** **-**1 **&&** botRight**.**Location**.**X **>=** 122**))**

**{**

// Changes the delay of the doors to 0, so the doors close instantly.

count **=** 0**;**

**}**

// Change the colour of the buttons text to notify the user that it's been pressed.

btnGround**.**ForeColor **=** Color**.**LightGreen**;**

**}**

// Method that handles the control panel button for the first floor.

**private** void btnFirst\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "First floor button has been pushed."**);**

// Inserting 'Up' into the list of 'direction' to make the elevator move up.

movement**.**direct**(**"Up"**);**

// Checks if the door is open, if so then forces the doors to close.

**if((**topLeft**.**Location**.**X **<=** **-**1 **&&** topRight**.**Location**.**X **>=** 122**)** **||** **(**botLeft**.**Location**.**X **<=** **-**1 **&&** botRight**.**Location**.**X **>=** 122**))**

**{**

// Changes the delay of the doors to 0, so the doors close instantly.

count **=** 0**;**

**}**

// Change the colour of the buttons text to notify the user that it's been pressed.

btnFirst**.**ForeColor **=** Color**.**LightGreen**;**

**}**

// Method that handles the control panel button to open the doors, depending on the current floor of the elevator.

**private** void btnOpen\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "Request to open elevator doors button has been pushed."**);**

// If the elevator is on the first floor, then open the first floor doors.

**if(**insideEle**.**Location**.**Y **<=** 0**)**

**{**

// Inserting 'CloseTop' into the list of 'direction' to make the first floor doors open.

movement**.**direct**(**"OpenTop"**);**

**}**

// Else if the elevator is located on the ground floor, then open the ground floor doors.

**else** **if(**insideEle**.**Location**.**Y **>=** 346**)**

**{**

// Inserting 'CloseBot' into the list of 'direction' to make the ground floor doors open.

movement**.**direct**(**"OpenBot"**);**

**}**

// Let's the user know that the button has been requested, by being changed to a different colour.

btnOpen**.**ForeColor **=** Color**.**LightGreen**;**

**}**

// The method to close the doors, according to the floor.

**private** void btnClose\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "Request to close elevator doors button has been pushed."**);**

// Checks to see if the first floor door or the ground floor door is open, then closes accordingly.

**if** **((**topLeft**.**Location**.**X **<=** **-**1 **&&** topRight**.**Location**.**X **>=** 122**)** **||** **(**botLeft**.**Location**.**X **<=** **-**1 **&&** botRight**.**Location**.**X **>=** 122**))**

**{**

// Sets the delay counter to 0, so the doors start to close instantly.

count **=** 0**;**

// Changes the text of the buttons colour to green to let the user know the button as been requested.

btnClose**.**ForeColor **=** Color**.**LightGreen**;**

**}**

**}**

// Method that handles the emergency alert button.

**private** void btnAlert\_Click**(object** sender**,** EventArgs e**)**

**{**

// Utilising the database delegate to insert data to the database.

database**.**databaseEvent**(**"Insert"**,** "Emergency Alert button has been pushed."**);**

// Inserts the value of 'Emergency' to the 'direction' list as an index of 0, to prioritise over other elements of the list.

movement**.**direct**(**"Emergency"**,** 0**);**

// The colour of the button's text is permanently changed to red.

btnAlert**.**ForeColor **=** Color**.**Red**;**

**}**

// Timer to run the methods of allowing the elevator to move and the doors to open/close.

**private** void eleTimer\_Tick**(object** sender**,** EventArgs e**)**

**{**

// Calls the methods from the Movement class.

movement**.**moveElevator**();**

movement**.**openDoor**();**

// If the counter is less than or equal to 0 then start to close the doors.

**if** **(**count **<=** 0**)**

**{**

movement**.**closeDoor**();**

**}**

// If the location of the elevator is on the first floor then reset the request buttons.

**if** **(**insideEle**.**Location**.**Y **<=** 0**)**

**{**

// Changes the text colour of the request to the first floor buttons.

rqstFirst**.**ForeColor **=** Color**.**Black**;**

btnFirst**.**ForeColor **=** Color**.**Black**;**

**}**

// If the location of the elevator is on the ground floor then reset the request buttons.

**if** **(**insideEle**.**Location**.**Y **>=** 346**)**

**{**

// Changes the text colour of the request to the ground floor buttons.

rqstGround**.**ForeColor **=** Color**.**Black**;**

btnGround**.**ForeColor **=** Color**.**Black**;**

**}**

**}**

// Timer that counts down the 'count' variable every second.

**private** void counter\_Tick**(object** sender**,** EventArgs e**)**

**{**

// As the value of 'count' starts at 3, the 'count' variable will decrease to 0 in 3 seconds of the timer start.

count**--;**

**}**

// Sets the value for all displays.

**public** void setDisplay**(**string floor**)**

**{**

// Changes the text of all three labels according to the value of the string variable.

groundDisplay**.**Text **=** floor**;**

firstDisplay**.**Text **=** floor**;**

controlDisplay**.**Text **=** floor**;**

**}**

// Sets the delay counter for closing the doors.

**public** void setCounter**(**bool val**)**

**{**

// Enables the timer, to start the delay, according to the value of val.

counter**.**Enabled **=** val**;**

// If the value is false, the reset the delay to 3.

**if** **(!**val**)**

**{**

// The count variable is changed to 3.

count **=** 3**;**

**}**

**}**

// Returns the DataGridView of the table.

**public** DataGridView getLogTable**()**

**{**

**return** logTable**;**

**}**

// Returns the Label of the elevator label.

**public** Label getElevator**()**

**{**

**return** insideEle**;**

**}**

// Returns the Label of the first floor left door.

**public** Label getTopLeft**()**

**{**

**return** topLeft**;**

**}**

// Returns the Label of the first floor right door.

**public** Label getTopRight**()**

**{**

**return** topRight**;**

**}**

// Returns the Label of the ground floor left door.

**public** Label getBotLeft**()**

**{**

**return** botLeft**;**

**}**

// Returns the Label of the ground floor right door.

**public** Label getBotRight**()**

**{**

**return** botRight**;**

**}**

// Returns the Button of the first floor request button.

**public** Button getRqstFirst**()**

**{**

**return** rqstFirst**;**

**}**

// Returns the Button of the ground floor request button.

**public** Button getRqstGround**()**

**{**

**return** rqstGround**;**

**}**

// Returns the Button of the control panel request to first floor.

**public** Button getBtnFirst**()**

**{**

**return** btnFirst**;**

**}**

// Returns the Button of the control panel request to ground floor.

**public** Button getBtnGround**()**

**{**

**return** btnGround**;**

**}**

// Returns the Button of the control panel to open the doors.

**public** Button getBtnOpen**()**

**{**

**return** btnOpen**;**

**}**

// Returns the Button of the control panel to close the doors.

**public** Button getBtnClose**()**

**{**

**return** btnClose**;**

**}**

// Returns the Button of the Emergency Alert on the control panel.

**public** Button GetBtnAlert**()**

**{**

**return** btnAlert**;**

**}**

**}**

**}**

## Movement.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Drawing**;**

**namespace** Elevator

**{**

class Movement

**{**

// Variables and objects.

**private** Elevator elevator**;**

**private** Sound sound**;**

**private** List**<**string**>** direction**;**

**private** int elePos**,** leftX**,** rightX**;**

// Constructor, to initialise the variables/objects.

**public** Movement**(**Elevator e**)**

**{**

// Instantiate the object.

elevator **=** e**;**

sound **=** **new** Sound**();**

// Instantiate a list of the string data type, to contain the direction of the elevator.

direction **=** **new** List**<**string**>();**

// Stores the integer values of the elevator and the doors.

elePos **=** 346**;**

leftX **=** 0**;**

rightX **=** 122**;**

**}**

// Method to control the movement of the elevator itself.

**public** void moveElevator**()**

**{**

// If the list of direction has content, then run the next statements.

**if(**direction**.**Any**())**

**{**

// If the first item of direction is 'Up' then run the next statements.

**if(**direction**[**0**]** **==** "Up"**)**

**{**

// If the elevators location is about 0 then move the elevator up.

**if** **(**elevator**.**getElevator**().**Location**.**Y **>=** 0**)**

**{**

// Minus the value of 'elePos' each time.

elePos**--;**

// Change the value accordingly.

elevator**.**getElevator**().**Location **=** **new** Point**(**0**,** elePos**);**

**}**

// Else remove the 'direction' of 'Up' and start opening the first floor doors.

**else**

**{**

// Plays a sound, notifying the users that they have reached the first floor.

sound**.**soundEvent**(**"First"**);**

// This will delete the 'Up' value within the direction list.

direction**.**RemoveAt**(**0**);**

// This will then insert a value to open the first floor doors.

direction**.**Insert**(**0**,** "OpenTop"**);**

// Set the displays of every display window to the first floor.

elevator**.**setDisplay**(**"1"**);**

// Change the colour of the buttons, to indicate the request has been fulfilled.

elevator**.**getRqstFirst**().**ForeColor **=** Color**.**Black**;**

elevator**.**getBtnFirst**().**ForeColor **=** Color**.**Black**;**

**}**

**}**

// Else if the direction is 'Down' then do the opposite of 'Up'.

**else** **if(**direction**[**0**]** **==** "Down"**)**

**{**

// If the position of the elevator is below 346 then move the elevator down.

**if(**elevator**.**getElevator**().**Location**.**Y **<=** 346**)**

**{**

// Increases the value of elePos each time.

elePos**++;**

// Relocates the position of the elevator.

elevator**.**getElevator**().**Location **=** **new** Point**(**0**,** elePos**);**

**}**

// Else start opening the door and reset values accordingly.

**else**

**{**

// Plays a sound, notifying the users that they have reached the ground floor.

sound**.**soundEvent**(**"Ground"**);**

// Removes the value of 'Down' within the list of 'direction'.

direction**.**RemoveAt**(**0**);**

// Insert 'OpenBot' within 'direction', so the bottom doors can start opening.

direction**.**Insert**(**0**,** "OpenBot"**);**

// Set the displays to show the current floor of Ground.

elevator**.**setDisplay**(**"G"**);**

// Resets the requests of the buttons.

elevator**.**getRqstGround**().**ForeColor **=** Color**.**Black**;**

elevator**.**getBtnGround**().**ForeColor **=** Color**.**Black**;**

**}**

**}**

// Else if the first value of 'direction' is 'Emergency' then run the following lines of code.

**else** **if(**direction**[**0**]** **==** "Emergency"**)**

**{**

// If the top door of the elevators are closed or open, then open them.

**if** **(**elevator**.**getTopLeft**().**Location**.**X **>=** **-**122 **&&** elevator**.**getTopRight**().**Location**.**X **<=** 244**)**

**{**

// Starts to decrease the value of leftX.

leftX**--;**

// Changes the position of the left door according to the value of leftX.

elevator**.**getTopLeft**().**Location **=** **new** Point**(**leftX**,** 0**);**

// Increases the value of rightX.

rightX**++;**

// Changes the position of the right door according to the value of rightX.

elevator**.**getTopRight**().**Location **=** **new** Point**(**rightX**,** 0**);**

**}**

**if** **(**elevator**.**getBotLeft**().**Location**.**X **>=** **-**122 **&&** elevator**.**getBotRight**().**Location**.**X **<=** 244**)**

**{**

// Starts to decrease the value of leftX.

leftX**--;**

// Changes the position of the left door according to the value of leftX.

elevator**.**getBotLeft**().**Location **=** **new** Point**(**leftX**,** 346**);**

// Increases the value of rightX.

rightX**++;**

// Changes the position of the right door according to the value of rightX.

elevator**.**getBotRight**().**Location **=** **new** Point**(**rightX**,** 346**);**

**}**

**}**

**}**

**}**

// A method to control the opening of doors, first or ground floor.

**public** void openDoor**()**

**{**

// Checks to see if the 'direction' list contains anything.

**if** **(**direction**.**Any**())**

**{**

// If the 'direction' list contains 'OpenTop' then start opening the first floor doors.

**if** **(**direction**[**0**]** **==** "OpenTop"**)**

**{**

// Checks to see if the doors are currently open.

**if** **(**elevator**.**getTopLeft**().**Location**.**X **>=** **-**122 **&&** elevator**.**getTopRight**().**Location**.**X **<=** 244**)**

**{**

// Decreases the value of leftX.

leftX**--;**

// Assigns the value of leftX to the first floor left door of the elevator.

elevator**.**getTopLeft**().**Location **=** **new** Point**(**leftX**,** 0**);**

// Increases the value of rightX.

rightX**++;**

// Assigns the value of rightX to the first floor right door of the elevator.

elevator**.**getTopRight**().**Location **=** **new** Point**(**rightX**,** 0**);**

**}**

**else**

**{**

// Remove the value of 'OpenTop' from the first index of 'direction'.

direction**.**RemoveAt**(**0**);**

// Insert the value of 'CloseTop' into 'direction' so the doors can start closing.

direction**.**Insert**(**0**,** "CloseTop"**);**

// Enable the delay for the doors to start closing.

elevator**.**setCounter**(true);**

// Change the request to open the doors button back to normal.

elevator**.**getBtnOpen**().**ForeColor **=** Color**.**Black**;**

**}**

**}**

// Else if the 'direction' list contains 'OpenBot' then open the ground floor doors.

**else** **if(**direction**[**0**]** **==** "OpenBot"**)**

**{**

// If the ground floor doors were closed then start opening them.

**if(**elevator**.**getBotLeft**().**Location**.**X **>=** **-**122 **&&** elevator**.**getBotRight**().**Location**.**X **<=** 244**)**

**{**

// Decreases the value of leftX.

leftX**--;**

// Changes the location of the ground floor left door according to the value of leftX.

elevator**.**getBotLeft**().**Location **=** **new** Point**(**leftX**,** 346**);**

// Increase the value of rightX.

rightX**++;**

// Changes the location of the ground floor right door according to the value of rightx.

elevator**.**getBotRight**().**Location **=** **new** Point**(**rightX**,** 346**);**

**}**

// Else if the doors are fully open, start the process of closing the doors.

**else**

**{**

// Remove the value of 'OpenBot' from the 'direction' list.

direction**.**RemoveAt**(**0**);**

// Insert into the 'direction' list with the value of 'CloseBot' to run the closeDoor() method.

direction**.**Insert**(**0**,** "CloseBot"**);**

// Starts the delay of when the doors should start closing.

elevator**.**setCounter**(true);**

// Changes the color of the button that requested to open the doors, if it was pressed.

elevator**.**getBtnOpen**().**ForeColor **=** Color**.**Black**;**

**}**

**}**

**}**

**}**

// Method that closes the doors according to the location of the elevator.

**public** void closeDoor**()**

**{**

// If the 'direction' list has any elements within it, then the content within the if statement will run.

**if** **(**direction**.**Any**())**

**{**

// If the 'direction' list contains 'CloseTop' then the doors will start to close.

**if** **(**direction**[**0**]** **==** "CloseTop"**)**

**{**

// If the first floor elevator doors are open, then start the process of closing them.

**if** **(**elevator**.**getTopLeft**().**Location**.**X **<=** **-**1 **&&** elevator**.**getTopRight**().**Location**.**X **>=** 122**)**

**{**

// Increase the value of leftX.

leftX**++;**

// Change the position of the first floor left door, with the value of leftX.

elevator**.**getTopLeft**().**Location **=** **new** Point**(**leftX**,** 0**);**

// Decrease the value of rightX.

rightX**--;**

// Change the position of the first floor right door, with the value of rightX.

elevator**.**getTopRight**().**Location **=** **new** Point**(**rightX**,** 0**);**

**}**

// Else once the elevator is closed, reset values accordingly.

**else**

**{**

// Remove the value of 'CloseTop' from the 'direction' list.

direction**.**RemoveAt**(**0**);**

// Reset the delay of doors closing.

elevator**.**setCounter**(false);**

// Reset the request of the close button to the default colour.

elevator**.**getBtnClose**().**ForeColor **=** Color**.**Black**;**

**}**

**}**

// Else if the first element of 'direction' is equal to 'CloseBot' then close the ground floor doors.

**else** **if** **(**direction**[**0**]** **==** "CloseBot"**)**

**{**

// If the doors are currently open then close them.

**if** **(**elevator**.**getBotLeft**().**Location**.**X **<=** **-**1 **&&** elevator**.**getBotRight**().**Location**.**X **>=** 122**)**

**{**

// Increase the value of leftX.

leftX**++;**

// Change the position of the ground floor left door according to the value of leftX.

elevator**.**getBotLeft**().**Location **=** **new** Point**(**leftX**,** 346**);**

// Decrease the value of rightX.

rightX**--;**

// Change the position of the ground floor right door according to the value of rightX.

elevator**.**getBotRight**().**Location **=** **new** Point**(**rightX**,** 346**);**

**}**

// Once the doors are close then reset values accordingly.

**else**

**{**

// Removes the value of 'CloseBot' by index.

direction**.**RemoveAt**(**0**);**

// Resets the delay counter.

elevator**.**setCounter**(false);**

// Changes the close buttons colour back to default.

elevator**.**getBtnClose**().**ForeColor **=** Color**.**Black**;**

**}**

**}**

**}**

**}**

// Inserts values into the 'direction' list according to the parameters.

**public** void direct**(**string move**)**

**{**

// If the value is not already inside of the list then insert the contents of 'move'.

**if** **(!**direction**.**Contains**(**move**))** direction**.**Add**(**move**);**

**}**

// Inserts the value by index.

**public** void direct**(**string move**,** int index**)**

**{**

// If the value is not already inside of the list then insert the contents of 'move' by 'index'.

**if** **(!**direction**.**Contains**(**move**))** direction**.**Insert**(**index**,** move**);**

**}**

// Method to get the list variable 'direction'

**public** List**<**string**>** getDirection**()**

**{**

// Returns the variable 'direction'

**return** direction**;**

**}**

**}**

**}**

## Database.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Data**.**OleDb**;**

**namespace** Elevator

**{**

class Database

**{**

// Object of the Elevator class.

**private** Elevator elevator**;**

// Variables to establish a connection with the database and create queries.

**private** OleDbConnection connection**;**

**private** OleDbCommand command**;**

**private** OleDbDataReader reader**;**

// Delegates to establish the type of connection to the database (Insert/Retrieve).

**public** **delegate** void DatabaseDelegate**(**string eventType**,** string details**);**

**public** DatabaseDelegate databaseEvent**;**

// Constructor to initialise the objects/variables.

**public** Database**(**Elevator e**)**

**{**

// Instantiates the elevator object.

elevator **=** e**;**

// Establishes a connection to the access database.

connection **=** **new** OleDbConnection**(**"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Elevator Log.accdb"**);**

// Instantiates the command object.

command **=** connection**.**CreateCommand**();**

// Instantiates the delegate object, with the EventHandler method.

databaseEvent **+=** **new** DatabaseDelegate**(**EventHandler**);**

**}**

// Method to help instantiate the database delegate.

**private** void EventHandler**(**string eventType**,** string details**)**

**{**

// Retrieves the current date and concatenates it with the time of the current day.

string dateTime **=** DateTime**.**Today**.**ToString**(**"d"**)** **+** " " **+** DateTime**.**Now**.**ToString**(**"HH:mm:ss tt"**);**

// Switch statement to establish whether the variable of eventType contains Insert or Retrieve.

**switch** **(**eventType**)**

**{**

// If 'Insert' return the method of logEvent.

**case** "Insert"**:** logEvent**(**dateTime**,** details**);**

**break;**

// If 'Retrieve' return the method of retrieveEvent.

**case** "Retrieve"**:** retrieveEvent**();**

**break;**

**}**

**}**

// A method that stores information of the date the button was pushed and the event type.

**private** void logEvent**(**string datetime**,** string details**)**

**{**

// Opens a connection with the database.

connection**.**Open**();**

// Places a SQL query within the CommandText of the OleDbCommand called command.

command**.**CommandText **=** "INSERT INTO `Logs`(`DateTime`, `Entry`) VALUES('" **+** datetime **+** "', '" **+** details **+** "');"**;**

// Executes the SQL query.

command**.**ExecuteNonQuery**();**

// Closes the connection with the database.

connection**.**Close**();**

// Calls the method that retrieves the data from the database and places it within the table.

retrieveEvent**();**

**}**

// A method that gets the content of the database one by one and stores it within the table.

**private** void retrieveEvent**()**

**{**

// Clears the current data of the table.

elevator**.**getLogTable**().**Rows**.**Clear**();**

// Stores a new query within the CommandText of command.

command**.**CommandText **=** "SELECT \* FROM `Logs`;"**;**

// Opens a connection with the database.

connection**.**Open**();**

// Executes the SQL query, then stores it within the object of reader.

reader **=** command**.**ExecuteReader**();**

// A while loop that runs as long as the contents of the object 'reader' runs true.

**while(**reader**.**Read**())**

**{**

// A string array with the contents that is retrieved from the database.

string**[]** data **=** **new** string**[]** **{** reader**[**"ID"**].**ToString**(),** reader**[**"DateTime"**].**ToString**(),** reader**[**"Entry"**].**ToString**()** **};**

// Stores the arrays as a row within the table.

elevator**.**getLogTable**().**Rows**.**Add**(**data**);**

**}**

// Closes the connection.

connection**.**Close**();**

**}**

**}**

**}**

## Sound.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**IO**;**

**using** System**.**Media**;**

**namespace** Elevator

**{**

class Sound

**{**

// Initiates objects.

**private** SoundPlayer soundPlayer**;**

// Declares a delegate to play sound types.

**public** **delegate** void SoundDelegate**(**string soundType**);**

**public** SoundDelegate soundEvent**;**

// Constructor of the class to instantiate variables/objects.

**public** Sound**()**

**{**

// Instantiate objects.

soundPlayer **=** **new** SoundPlayer**();**

soundEvent **+=** **new** SoundDelegate**(**EventHandler**);**

**}**

// Method to handle the sound types.

**private** void EventHandler**(**string soundType**)**

**{**

// Switch that sorts out the sound types.

**switch(**soundType**)**

**{**

// If 'First', then run the FirstFloorSound.

**case** "First"**:** FirstFloorSound**();**

**break;**

// If 'Ground', then run the GroundFloorSound.

**case** "Ground"**:** GroundFloorSound**();**

**break;**

**}**

**}**

// Methods to start playing sounds.

**private** void FirstFloorSound**()**

**{**

// Declare the stream, of the audio file.

soundPlayer**.**Stream **=** Properties**.**Resources**.**First\_floor**;**

// Play the audio file.

soundPlayer**.**Play**();**

**}**

**private** void GroundFloorSound**()**

**{**

// Declare the stream.

soundPlayer**.**Stream **=** Properties**.**Resources**.**Ground\_floor**;**

// Play the audio file.

soundPlayer**.**Play**();**

**}**

**}**

**}**

## Programme Code Discussion

To create the programme, I created/manipulated in total 4 classes and integrated the code with the windows forms components I designed in the Elevator.cs[Design] mode. The Elevator.cs file is the code that connects all the other .cs files. The Movement.cs file handles all the movements the animation itself may make, such as the elevator car moving up/down. The Database.cs file handles the event handlers and method for inserting/retrieving data from the database. The Sound.cs file handles any sounds that may play according to the elevators current floor.

# Testing

Figure : The main elevator program, with the elevator animation, control panel and GUI log.

The main Guided User Interface (GUI) is split up in three different sections, one: the animation demonstrating the movement of the elevator, two: the control panel with 5 buttons and a display window, three: a table that shows the information held in the database.

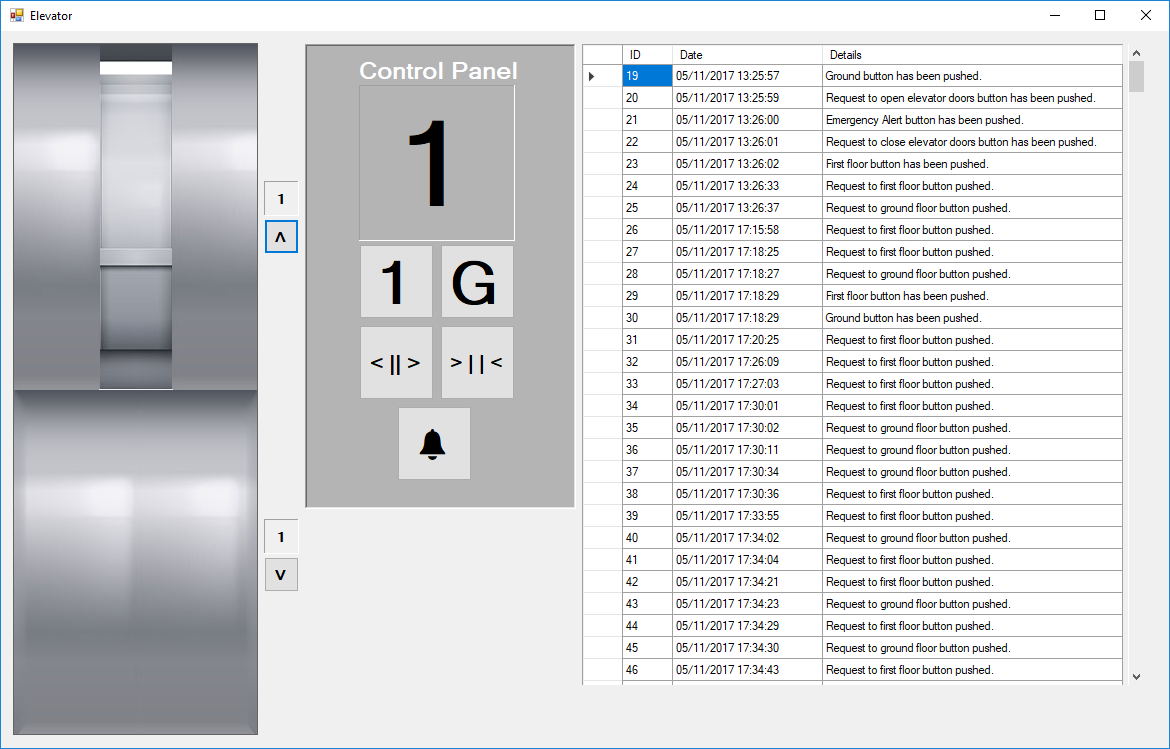
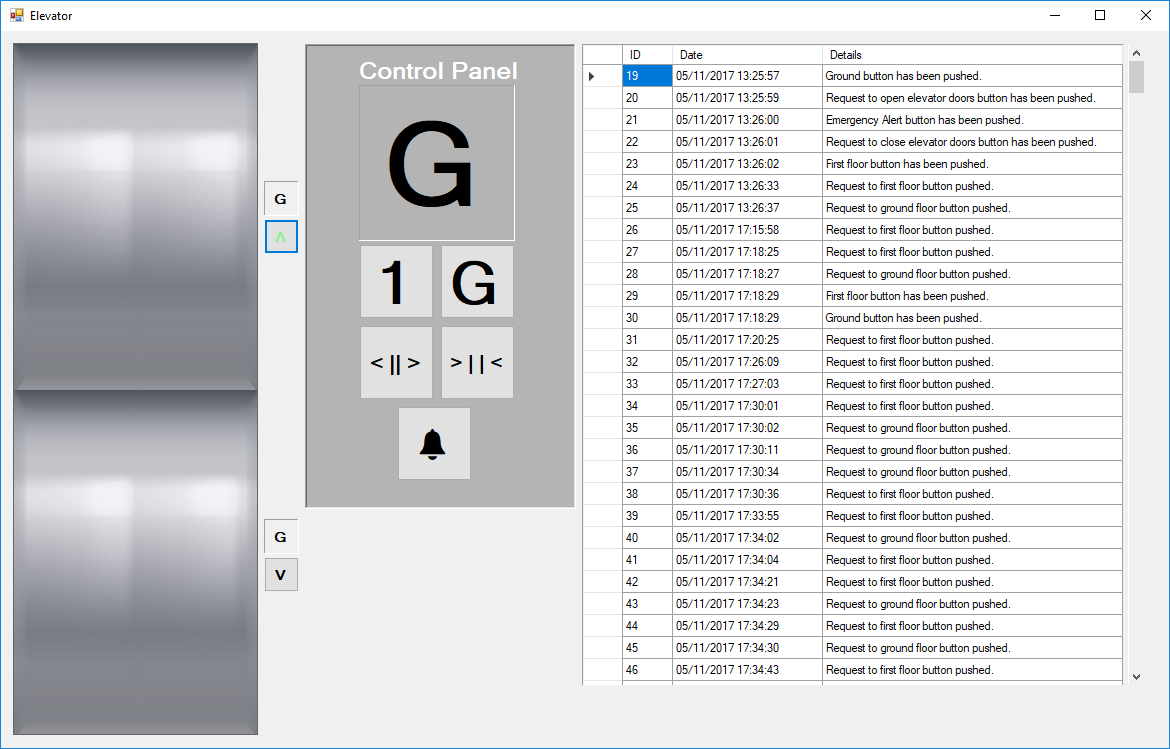


Figure : Elevator moving while top floor has been requested.

Figure : Elevator arriving at top floor, displays changed and request button back to normal

Figure 8 shows the elevator while the elevator is being requested, as you can see the elevator button goes green as soon as it has been pressed, indicating it has been requested. The elevator car will start to move towards the requested floor (in this case first floor). Figure 9 shows what happens once the elevator has arrived at its destination, the display windows all change accordingly and puts the requested buttons colour back to normal.

Once the doors are open wide they then stay open for three seconds, where they then begin to close after a few seconds awaiting any other requests to the elevator, if there are no requests then it will just sit on the last floor it was called to.

When any of the request buttons are pushed a method sends the information to the database with the specific actions, for example if the first-floor request button was pushed then an action would be sent to the database such as: “First floor request button was pushed”, alongside the date and time of the current day.

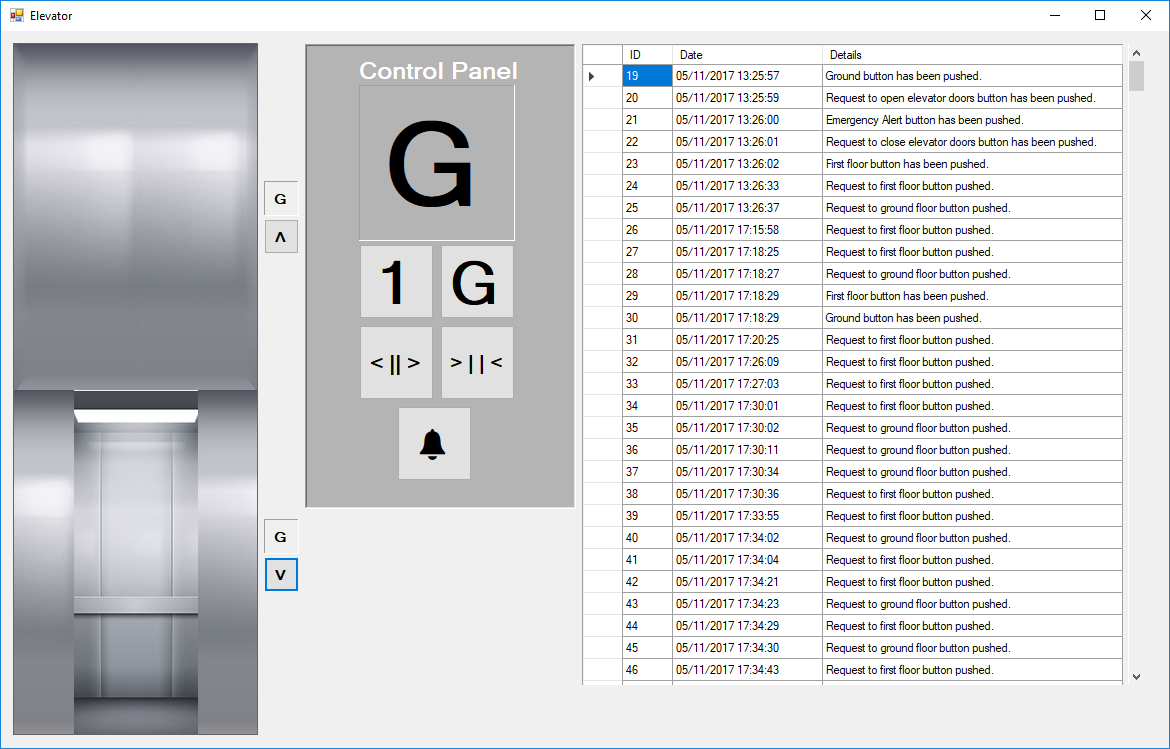
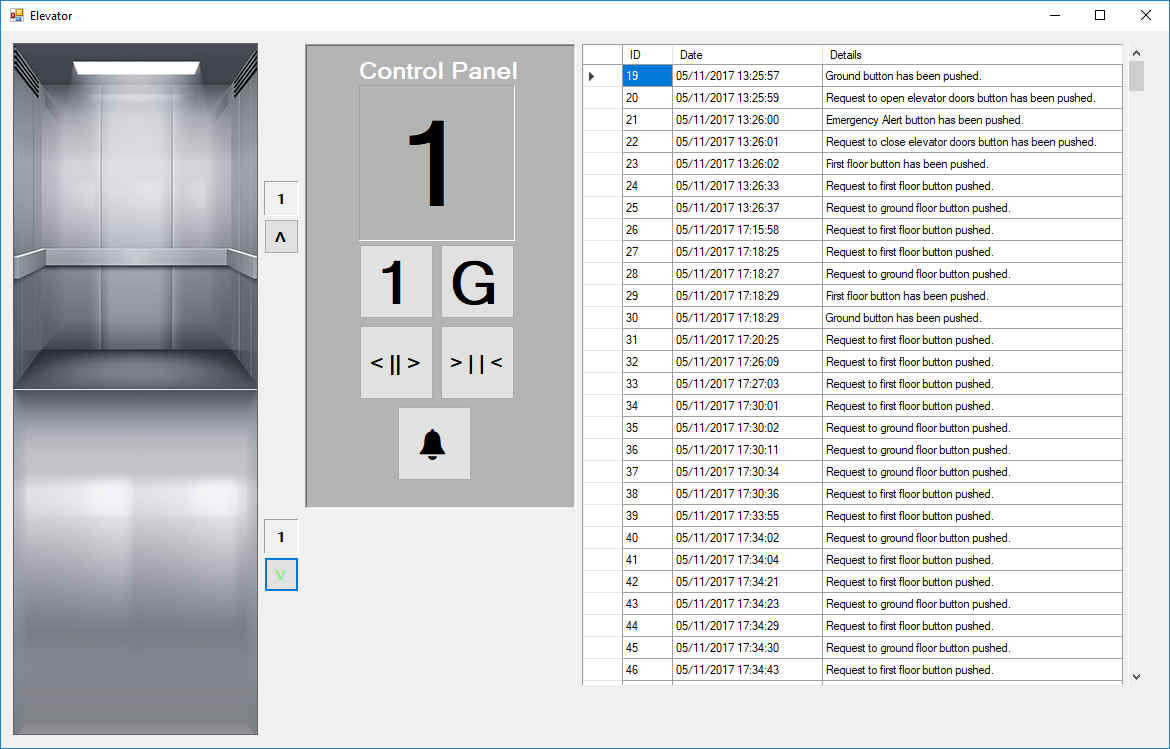
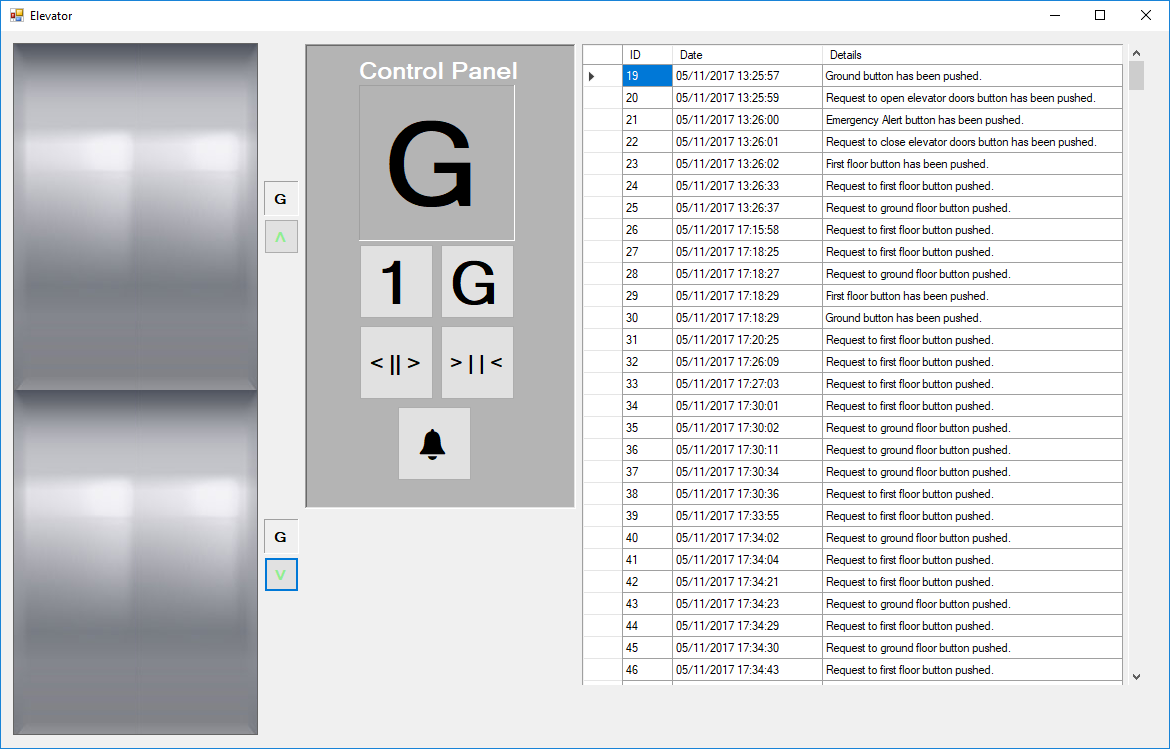


Figure : Shows the elevator being requested for both floors.

Figure : Shows the elevator arriving to the first floor and still requested to ground floor.

Figure : Shows the elevator moving to the ground floor and opening the doors.

Figures 10, 11 and 12 are tests to check if the elevator requests still work while the elevator is handling a request to another floor. In the testing, the request to first floor button was pushed first, then the down floor button, the elevator first moved to the first floor and after handling the opening/closing of doors on that floor. The elevator then started to handle the request for the ground floor, by moving to that floor and handling the opening/closing of doors.

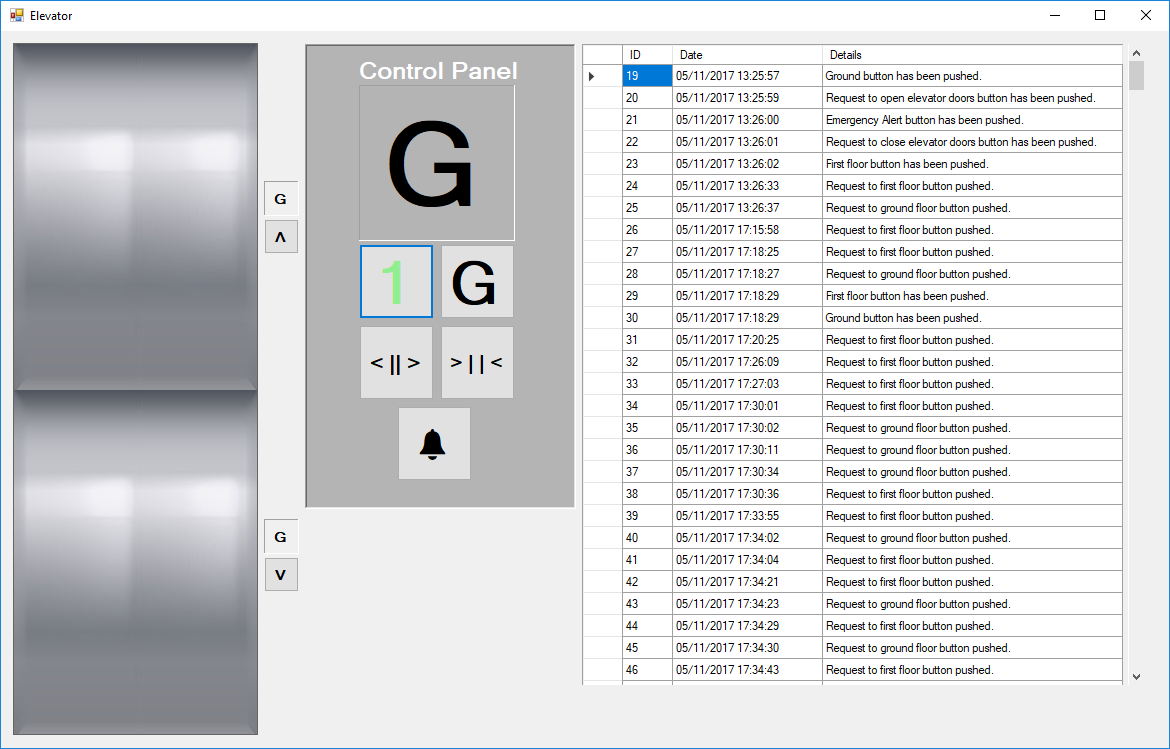
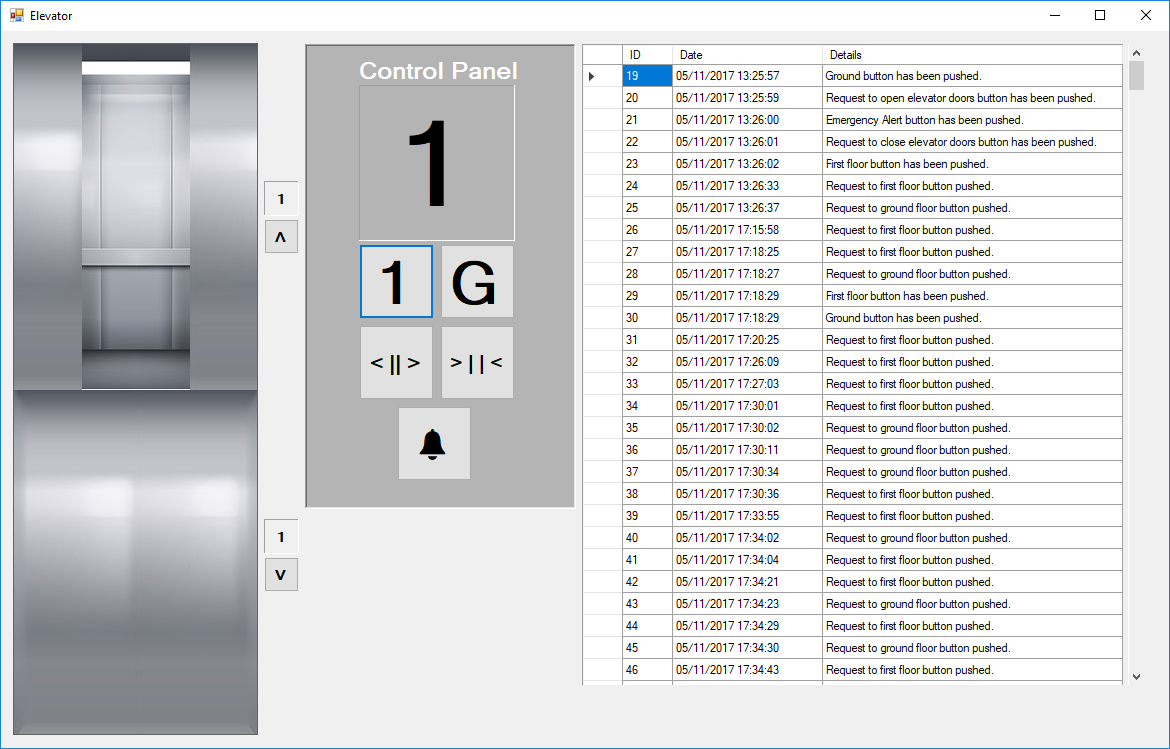
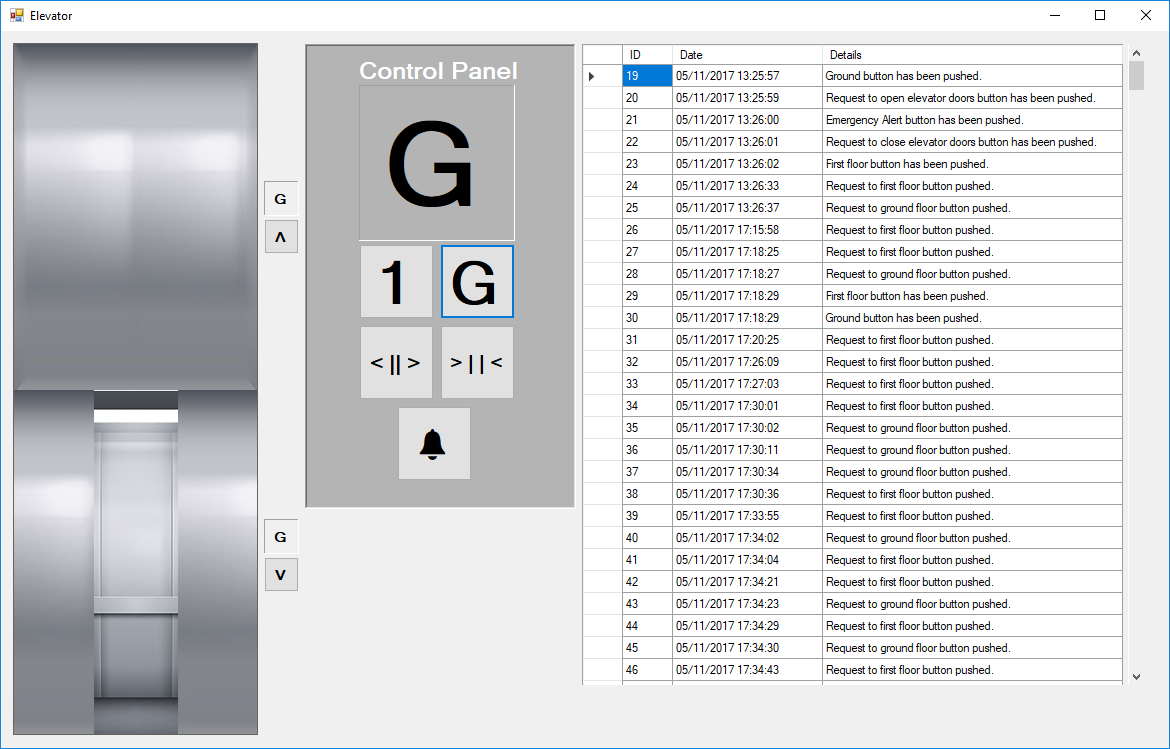
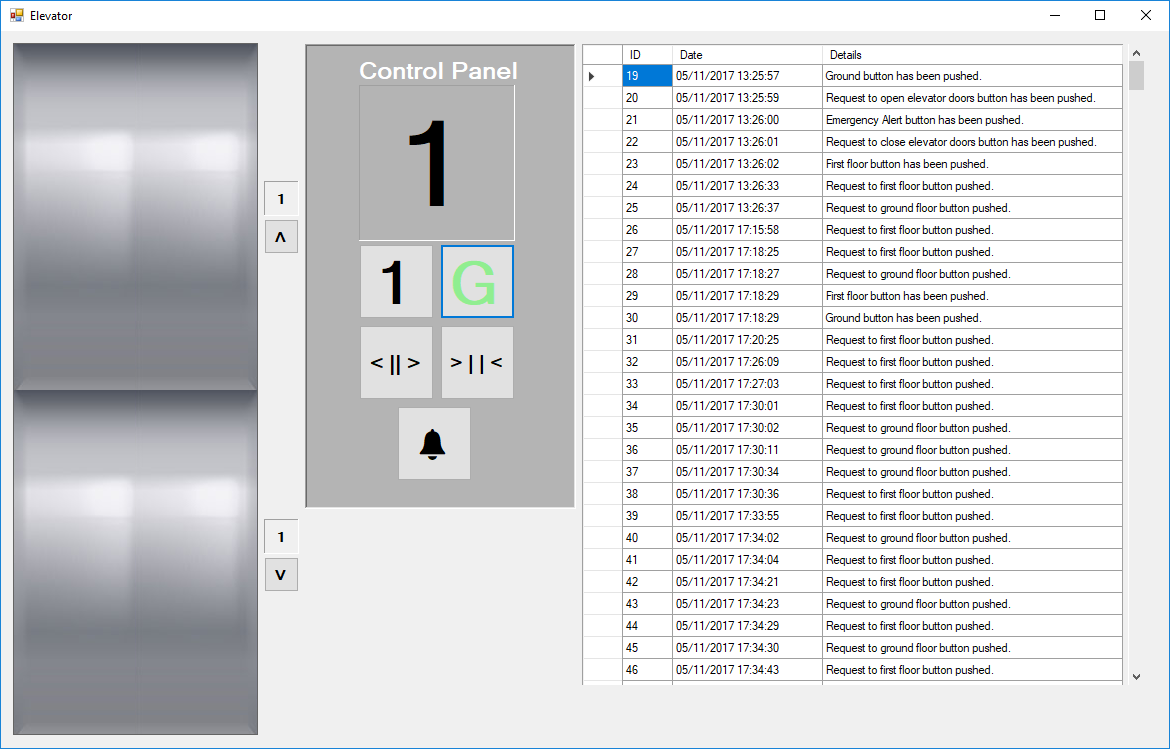


Figure : Shows the elevator arriving to the first floor.

Figure : Shows the request to first floor through the control panel.

Figure : Shows the elevator arriving to the ground floor.

Figure : Shows the elevator being requested to the ground floor.

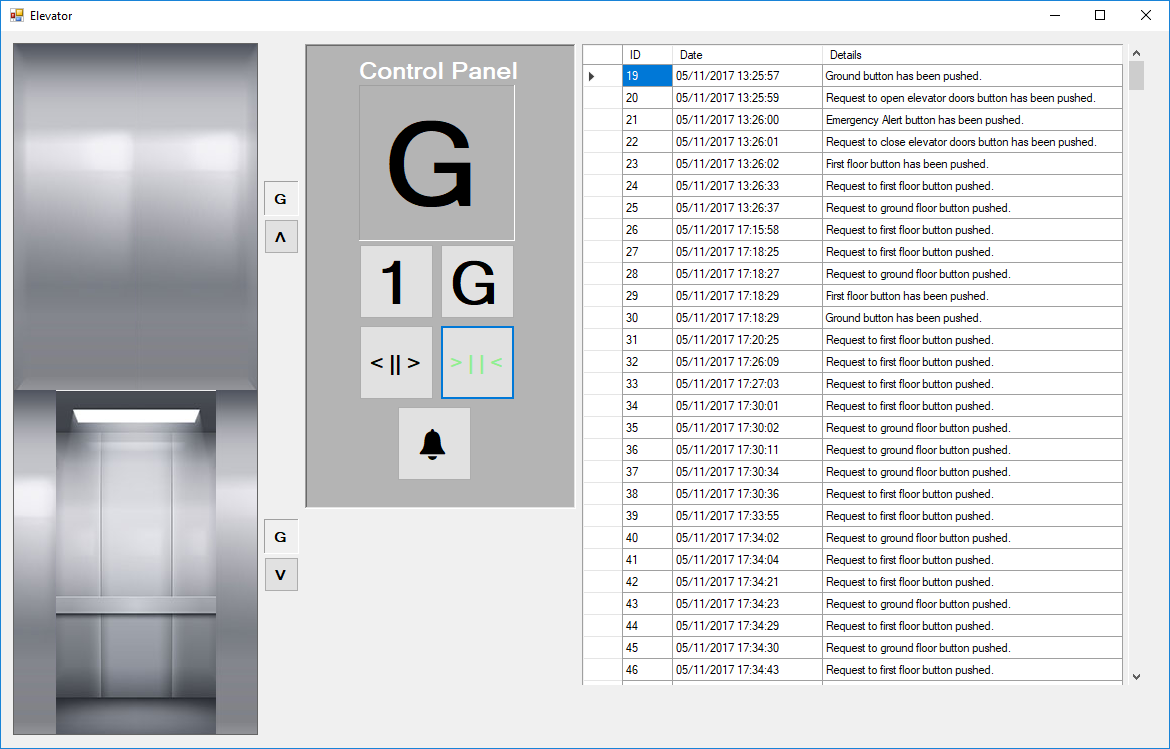
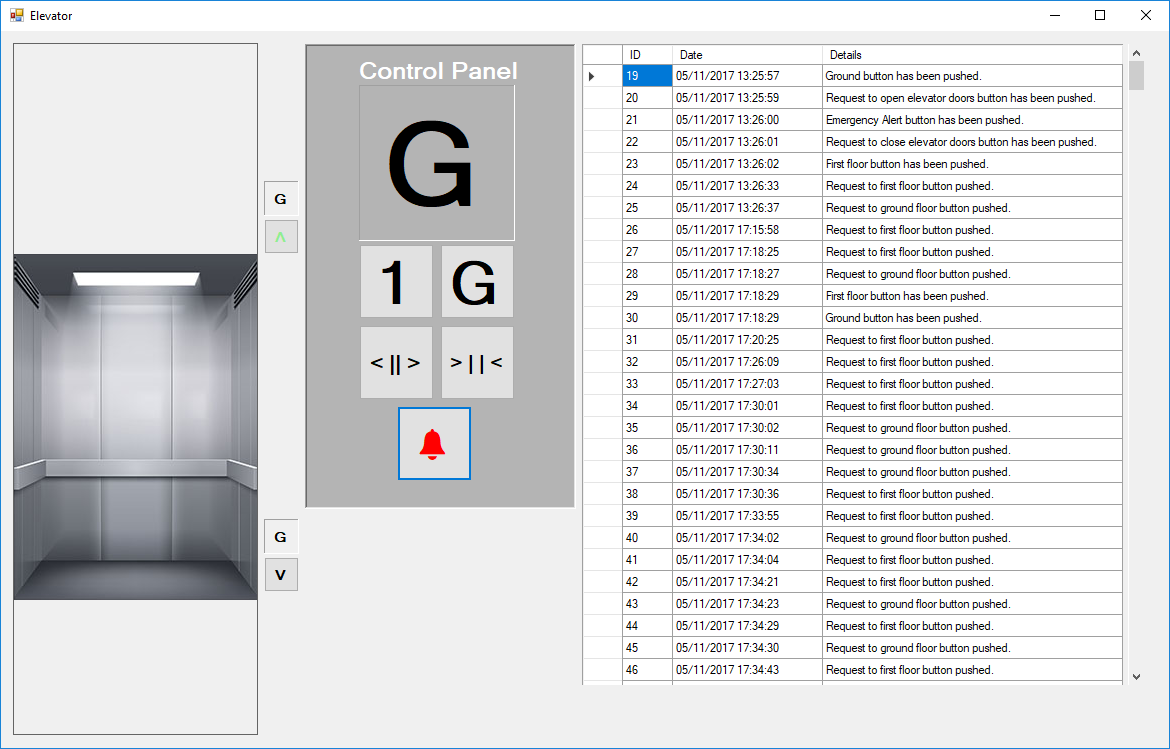
Figures 13, 14, 15 and 16 demonstrates the functionality of the control panel inside the elevator as the tasks require two specific buttons, one taking the elevator to the ground floor and the other to the first floor. Figures 13 and 14 shows the functionality of the first floor and how the elevator handles the specific requests, while figures 15 and 16 shows the request handling for the ground floor.

Figure : Pressing the open doors button on the control panel.

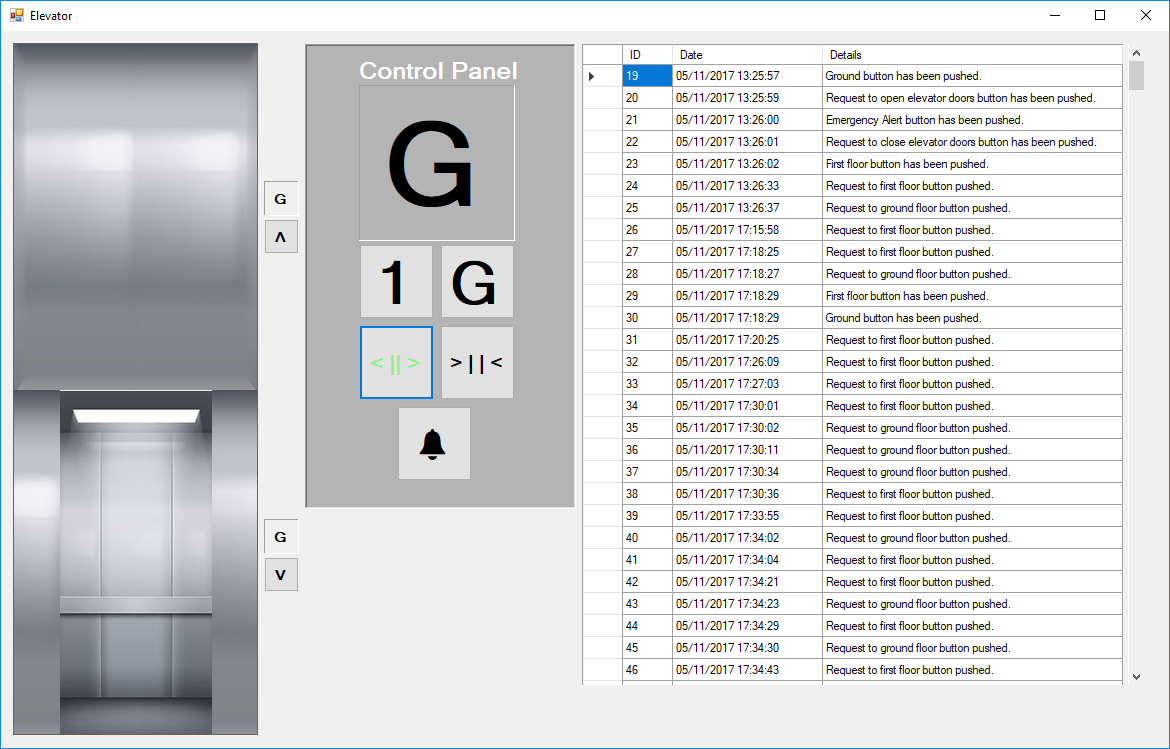
Figures 17 and 18 depict an extra feature implemented on the control panel to allow extra functionality and ease of access to the elevator specifically. The tests were to check if the doors would open or close according to the buttons pushed, figure 17 instantly starts to open the elevator doors if the elevator is on the first or second floor accordingly. Figure 18 starts to close the elevator doors according to which floor the doors are currently open and where the elevator is currently at.

Figure : Pressing the open doors button on the control panel.

Figure 19 is the final feature implemented to help user accessibility in case of emergencies the doors of both floors will open, and the elevator will stop immediately. This feature is primarily for emergencies though once activated it cannot be activated unless the program itself is terminated.

Figure : The emergency alert button.



Figure : Table on the Guided User Interface that shows information loaded from the database.

Figure 20 shows actions stored by the buttons of the control panel and the request buttons outside the elevator imported straight from the Microsoft Access Database linked to the programme. The information currently stored is the date the button was pushed and the action that was provided to identify the type of button.

Figure : Shows the additional information added by pushing specific buttons.

Figure 21 shows that the information does get stored directly to the database when buttons are pushed, this is new data added since figure 20.

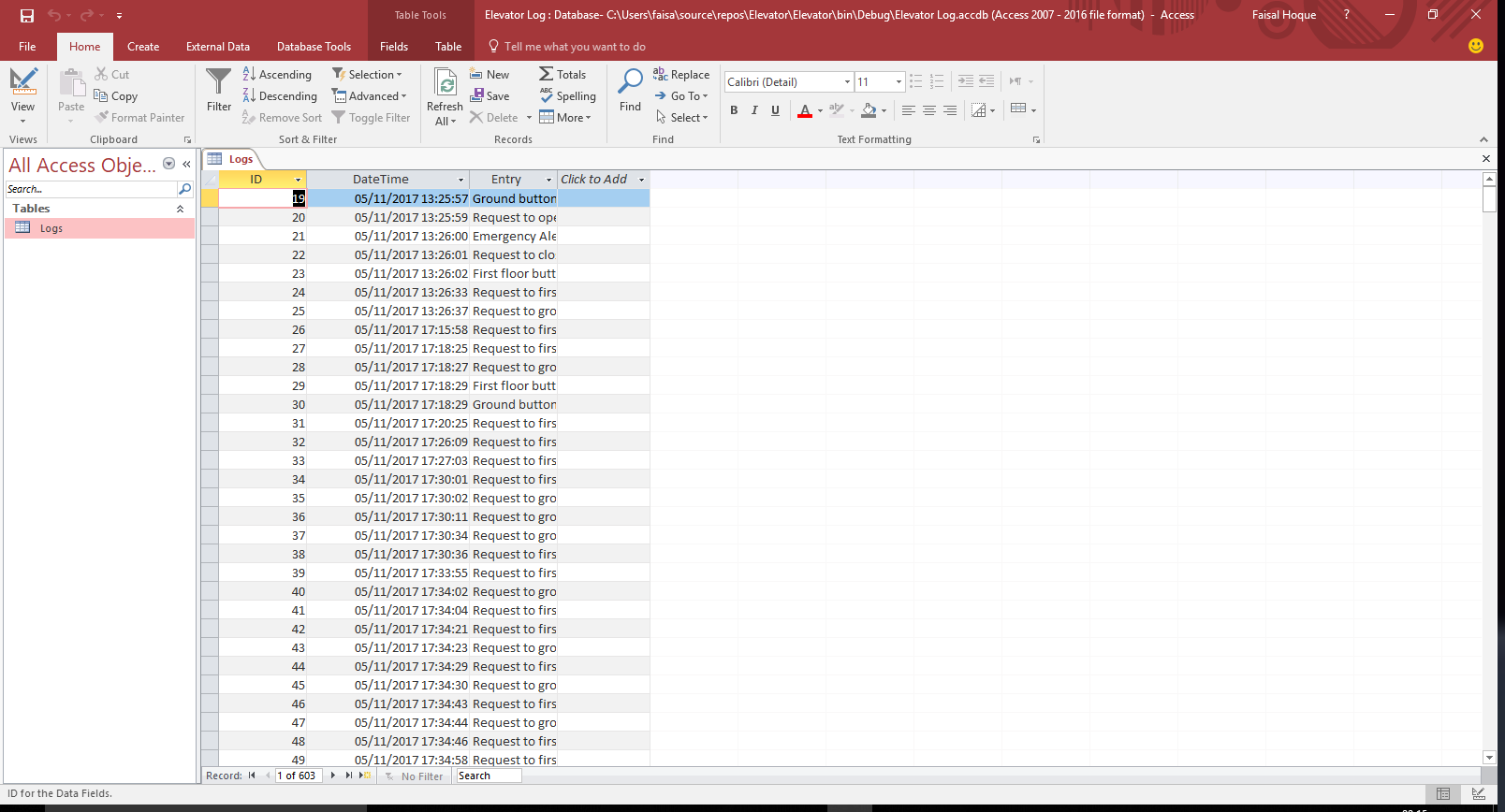
Figure 22 is the design of the Microsoft Access Database that is currently linked to the Elevator programme. It consists of a very basic structure, one table with three data fields, one data field is an auto increment number that increases every time data is inserted. The second is a Date/Time field that specifically includes the date and the time of when the buttons were pushed. The third is a Long Text data field that stores the actions of the buttons, such as whether request to first floor button was pushed etc.

Figure : The Microsoft Access Database that is linked to the Elevator programme.

# Discussion (812 words)

The elevator task was an interesting one to undertake as I was new to C# programming I was unsure of the correct syntaxes to use but, by week 3 I managed to make a very basic prototype that achieved around 70% functionality of the tasks set.

However, through creating a prototype I encountered many errors, more than I would have expected, not necessarily syntax errors but rather semantic errors such as while the elevator is moving down and then requested to the top floor the elevator stops and does not move anymore. The elevator moved up and down according to these variables: **private** bool moveUp**,** moveDown; which are both set to false by default, depending on the type of button pushed to request the elevator this code will take place: botRqst **=** **true;** and the timer that controls the elevator would be enabled with: eleTimer**.**Enabled **=** **true;** the problem with this is there is no control statement specifying when it should enable the movement of the elevator. That was fixed by wrapping those two lines of code with an if statement that would only run if the elevator was on a specific floor.

The elevator however, would only move when the elevator was on a specific floor, for example if the elevator was moving to the first floor and you requested it to the ground floor, the elevator would not handle that request even after it had arrived on the first floor. To fix this issue the utilisation of saving requests had to be done, the most efficient way to do this without handling data through a database or file would have been to use a List: **private** List**<**string**>** direction**;** over an Array. The list was utilised in two ways, temporarily holding recent requests and removing them after they had been dealt with. Whenever a request button is pushed the Elevator class utilises the method within the Movement class called ‘direct’: movement**.**direct**(**"Up"**);** the direct method first checked if the list already contained the specific direction, if it does not then insert it, if it does then do nothing. Within the Movement class under the moveElevator() there are control statements to check specific directions within the list and handle them accordingly. By utilising a list, the functionality of an elevator being requested from another floor while it is already handling a request, the elevator is able to handle that new request too without the user having to push the button again.

Another error discovered in the prototype is the animation of the elevator itself, that was created with the Graphics object that utilised the required shape method. e**.**Graphics**.**FillRectangle**(**Brushes**.**Black**,** 100**,** 25**,** 380**,** 740**);** this feature however was extremely inefficient as it created a glitch to the graphics as the screen would flash white and the colour of the graphics each time. The feature was resolved in the final elevator programme through utilising Labels and changing the image of the Label according to the required components. The Labels were moved by utilising the Location of the specific Label: elevator**.**getElevator**().**Location **=** **new** Point**(**0**,** elePos**);** and changing it each time to create an effect of motion.

The conclusive results can be tweaked by adding better graphics and more sound features with additional floors and a more useful emergency button with also an indoor elevator. However, the final programme achieves the tasks within the task description of the company, the elevator handles the requests for the first and ground floor and moves the elevator up/down accordingly with the elevator doors on each floor opening and closing accordingly too.