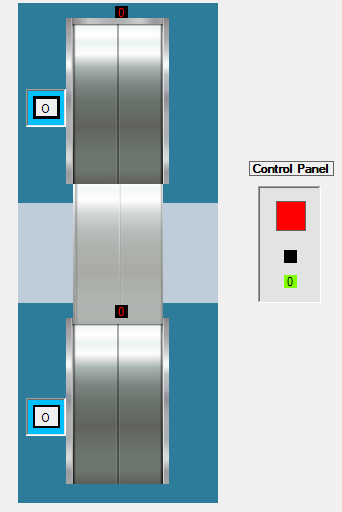
Elevator System

­­

Aim

This project was made for the subject CIS016-2, assignment one in the course of Bsc Computer Science Year 2 for the University of Bedfordshire.

The goal was to create a fully operable two-floor elevator system for a company using object-oriented software control application, with a control panel that monitors the elevator status, request buttons at each floor followed by a light system for a better visual input.

The doors have to be closed during trips to prevent passengers being injured by brushing against the wall of the elevator shaft. It can only open as it reaches the designated floor.

Also, the doors have to give enough time for the passengers to hop on and off before it starts its journey. If a button is pressed before the journey starts the doors should open again to accommodate more people.

Task description

**Task 1:** To create a GUI which contains

* Two request buttons corresponding to the two floors, respectively
* One control panel with two buttons and a display window
* Two display areas that display the status of the elevator, i.e. which floor the elevator currently stays, one is of each floor
* A log button

**Task 2:** To create a control program that processes the events published by the GUI. That is,

* When any request button is pressed,
  + the elevator appears in the corresponding floor
  + then the display areas and the display window on the control panel show the corresponding floor number at the same time
* When the floor number buttons on the control panel are pressed,
  + the elevator appears to the corresponding floor
  + then the display areas and the display window on the control panel show the corresponding floor number at the same time

**Task 3:** To create a log which record all operations. That is,

* The corresponding status of the elevator and the time information is stored in a database (it can be MS Access or other equivalent databases)
* To display the information which has been stored in the database on the GUI

**Task 4:**

* To animate the events described in Task 2 using delegation and timer

**Task 5:** To integrate and optimise the programs developed in Tasks 1 to 4. That is,

* Optimise the potability by using relative path instead of absolute path
* Optimise the maintainability by avoiding any duplication among the event handlers over the database related functions
* Optimise the robustness by eliminating logical errors and handling exception
* Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker

**Task 6:**

* To write a report which should have the following sections: aim, task description, design, programs and comments, and testing results.

Design

I chose a very simple design for testing purposes, easy to understand and operate, which leaves room for improvement.

I have also added a few extra features in order to simulate a real case scenario and some unreal ones for perception purposes.

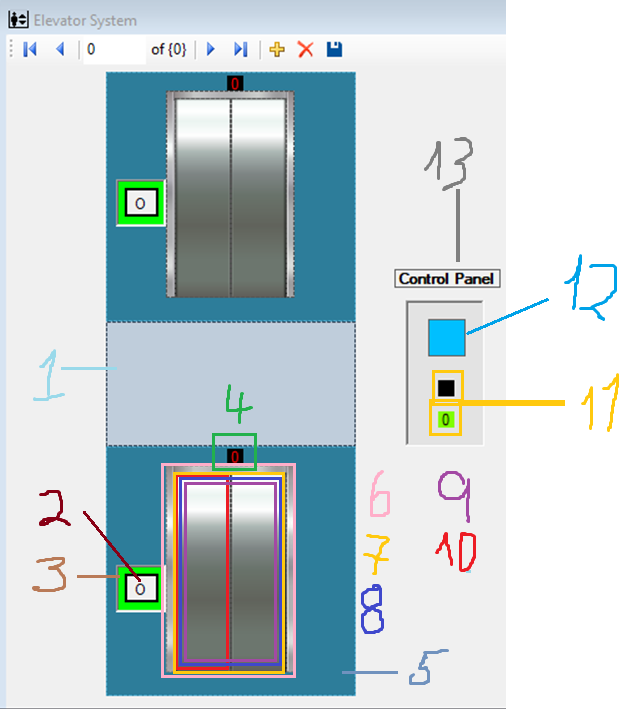


Fig 1 – Basic Design

As shown in the Fig 1, its possible to observe the graphic disparity between real world graphics and Computer-Graphic Imagery. In my opinion it was a good choice, because we can focus on the important stuff without losing too much appeal for its beauty and purpose.

The elevator main structure is composed by:

1. PictureBox component which represents a glass/transparent view and separates both floors.
2. Button to execute the call.
3. PictureBox component which represents the floor status light.
4. Label which tells us the current floor.
5. PictureBox component which represents the floor wall.
6. PictureBox component which represents the elevator façade for a better visual experience.
7. Area where both doors will be closed or open and initially has 8, 9 and 10.
8. PictureBox component which represents the inside of the elevator (not visible).
9. PictureBox component which represents the elevator closed with less opacity, to achieve the X- ray effect and being able to see its movements through the wall.
10. PictureBox component which represents a single door. Both doors are Singletons\*.
11. Labels that indicate the current floor. One of them will black out to light up the other one.
12. PictureBox component which represents the current elevator status.
13. Label named ‘Control Panel’.

**The Lights System and Current Floor**

|  |  |  |
| --- | --- | --- |
| Fig 2 – Basic Design | Fig 3 – Basic Design | Fig 4 – Basic Design |
| In the Fig 2 the elevator is in the ground floor, and the button’s light is green indicating that the switch can be pressed to open the door. The blue light indicates that the system is in standby, so the first floor’s switch is blue and the control panel’s light is blue as well. | In the Fig 3 the elevator is on the move so the control panel’s light is red indicating that the system is busy. The blue light indicates that the system is in standby, if any floor’s switch is pressed a call to that floor is added. | In the Fig 4 the elevator just arrived the first floor, and the button’s light is green indicating that the switch can be pressed to add another call, which means that the doors will shut and then reopen. The control panel’s light is also green indicating the door is open. The blue light indicates that the system is in standby, so the ground floor’s switch is blue. |

The Labels on top of each floor’s elevator façade are updated as soon as a floor is reached and the elevator has stopped. This will also take place in the Control Panel’s Label system, which is controlled by changing the background color to black and the other floor and setting the current floor’s to green.

Programs and comments

For this project I used a version control system, GitHub, which has the benefits of making it easy to perform rollbacks, integrate in large teams and see the progresses made. Also, it is open source for anyone who wants to contribute.

It is also a plus for building a course portfolio, which I can show to future employers.

The repository is the following: <https://github.com/brunoboto96/csElevatorSystem.git>

**Methods used (by order of logic):**

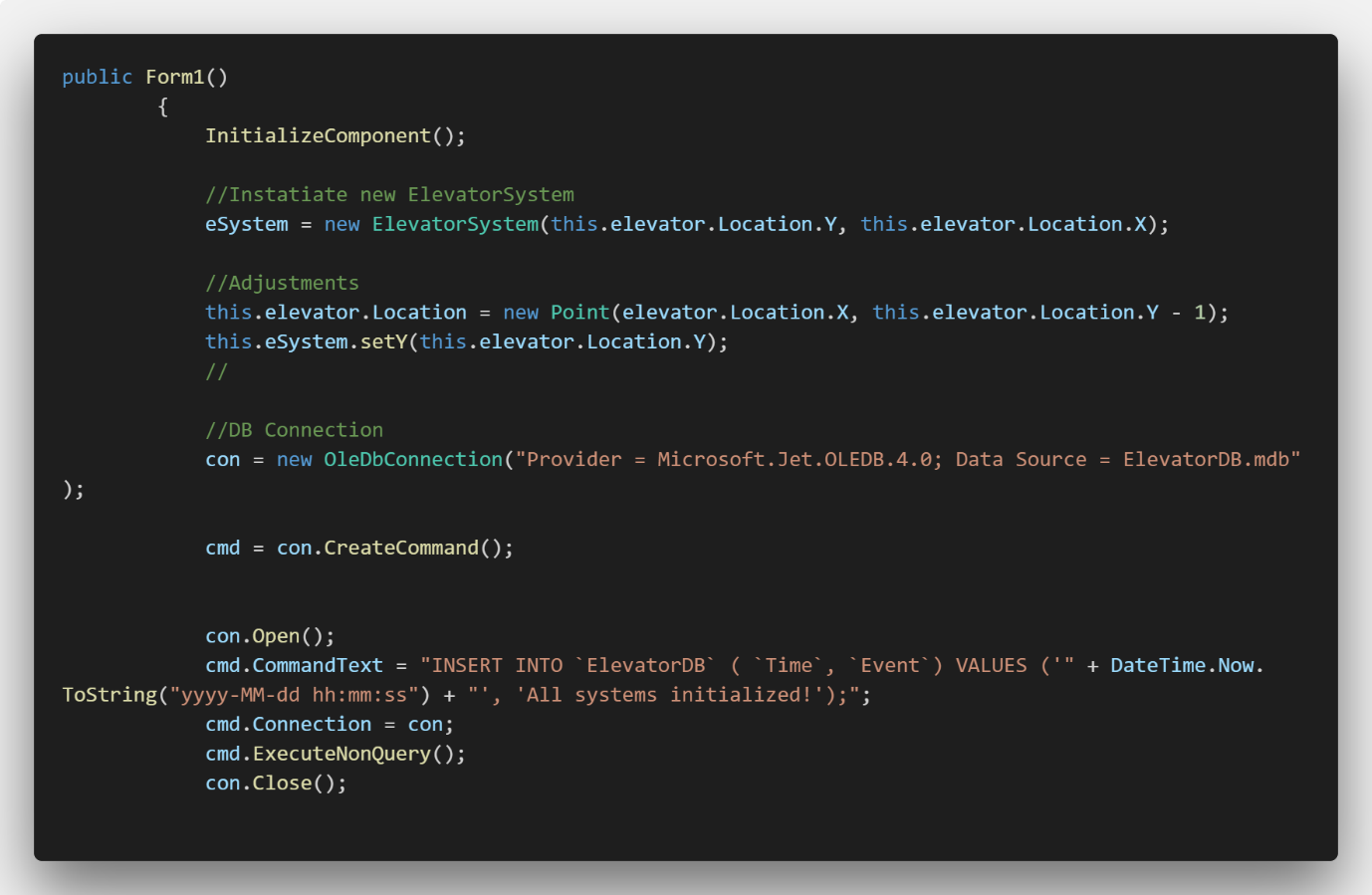


Fig 5 – Form constructor

This is a class constructor, it initializes all the components and draws it to the screen.

* Instantiate new object of type ElevatorSystem as eSystem
* Adjusts the elevator location, because of the threshold used to stop animations, preventing out of bounds
* Opens a Database connection
* Logs to the Database the initialization of the system using NonQuery

Fig 6 – ElevatorSystem object + getters and setters

* Initializes ElevatorSystem through the constructor accepting 2 parameters: x, y (elevator coordinates)
* Getters and Setters



Fig 7 – ElevatorSystem logic methods (part 2)

* Stores current floor
* Iterates calls
* Returns calls
* Clears all floor calls



Fig 8 – Switch calls

When the user presses the switch on any of the floors it executes the \_Click method.

* Runs a callFloorX() method of ElevatorSystem
* Logs the request to the Database



Fig 6 – Main Elevator System Timer

This timer has a short interval and is always ticking in order to perform actions by calling the necessary methods.

* Checks if system is busy
* Sets the necessary adjustments to perform the main actions
* Knows what floor to act



Fig 9 – Start Transition

Small method that starts the moving process.

* Starts timer
* Inform the system the elevator is moving



Fig 10 – Timer Transition

This method takes care of taking the passengers from one floor to the other.

* Sets boundaries, movement direction.
* Controls labels and lights
* Starts the door animation
* Logs to the Database, that the elevator has reached a floor.



Fig 11 – Open Doors Animation

This method draws opening of the door’s animation on the screen.

* Sets boundaries
* Knows which door to interact
* Reduces doors size to show inside of the elevator
* The right door moves proportionally with the size reduction

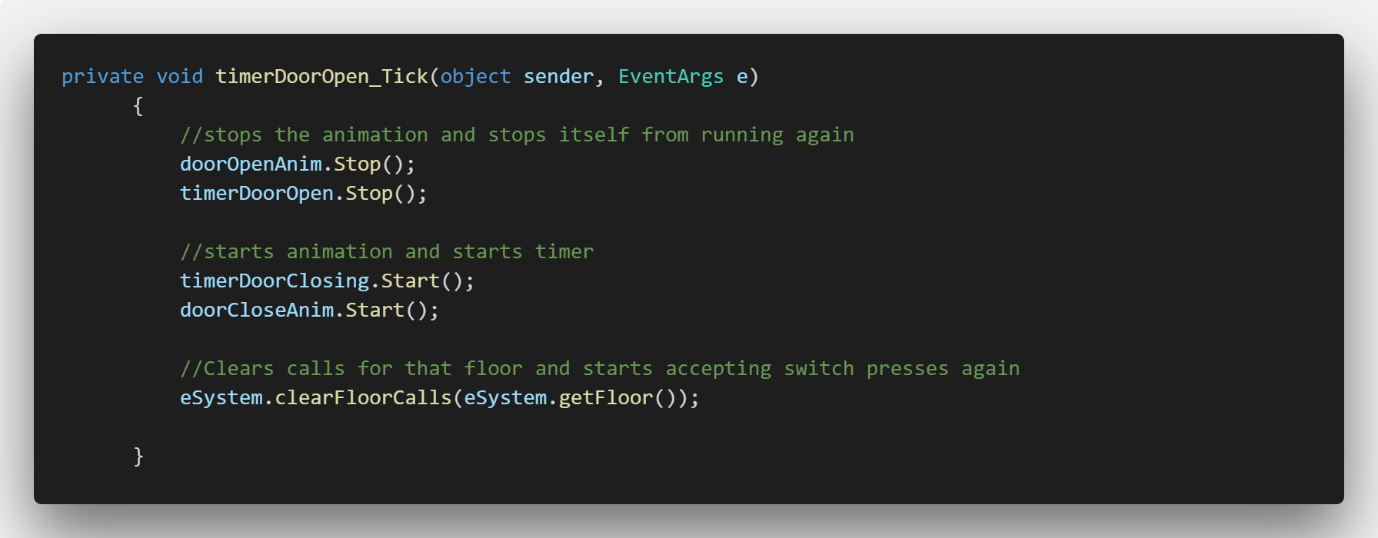


Fig 12 – Doors Opened

As soon as the doors are opened, this method will fire.

* Starts close doors animation and timer
* Reduces doors size to show inside of the elevator
* The right door had to be moved proportionally towards the right



Fig 13 – Close Doors Animation

This method draws the closing of the door’s animation on the screen.

* Sets boundaries, knows which door to interact
* Increases doors size to show inside of the elevator
* The right door moves proportionally with the size reduction



Fig 14 – Doors Closed

As soon as the doors are opened, this method will fire.

* Starts close doors animation and timer
* Reduces doors size to show inside of the elevator
* The right door moves proportionally with the size reduction



Fig 15 – Updating the Database

These methods log system events to the database.

* Executes Database Insert queries.

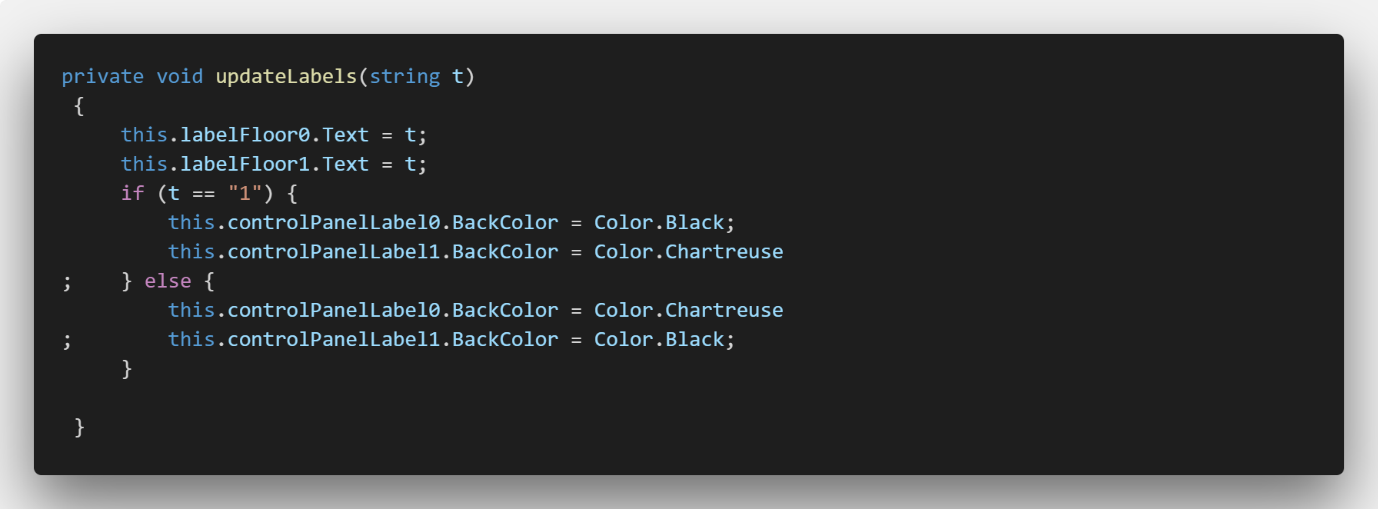


Fig 16 – Updating the Labels

Updates the labels to inform passengers of the current floor state.

* Changes text in the Labels
* Changes Background Color in Control Panel Label to either black out or show current floor



Fig 17 – Setting the Lights

These methods are changing the light system

* Sets switches’ lights
* Sets specific light with specific color
* Getter for the current floor switch light

Testing Results

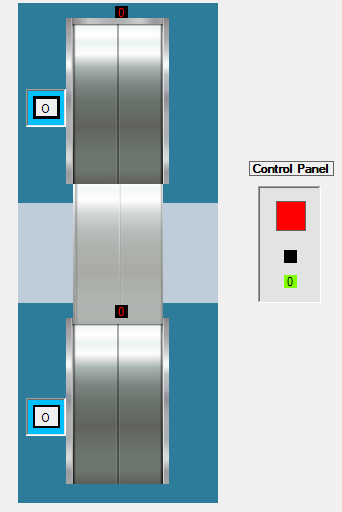


Fig 18 – Moving up!

Here we see the motion of the elevator moving up.

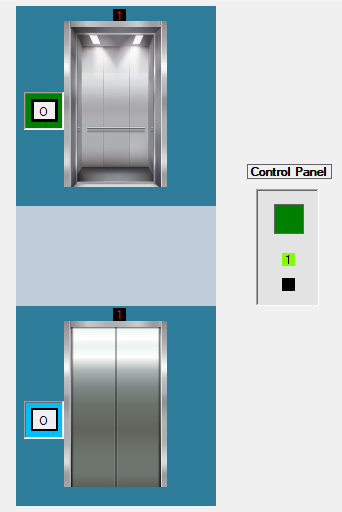


Fig 19 – Opening Doors

Here we see the doors opening.

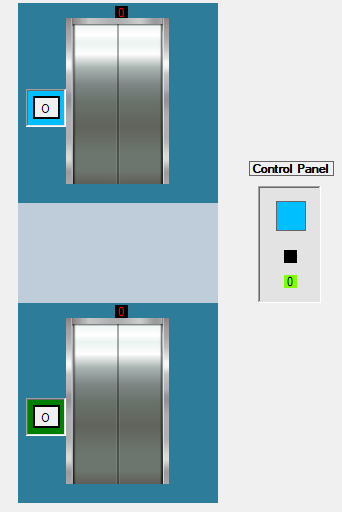


Fig 20 – Standing by

Here we see the system waiting for user input.

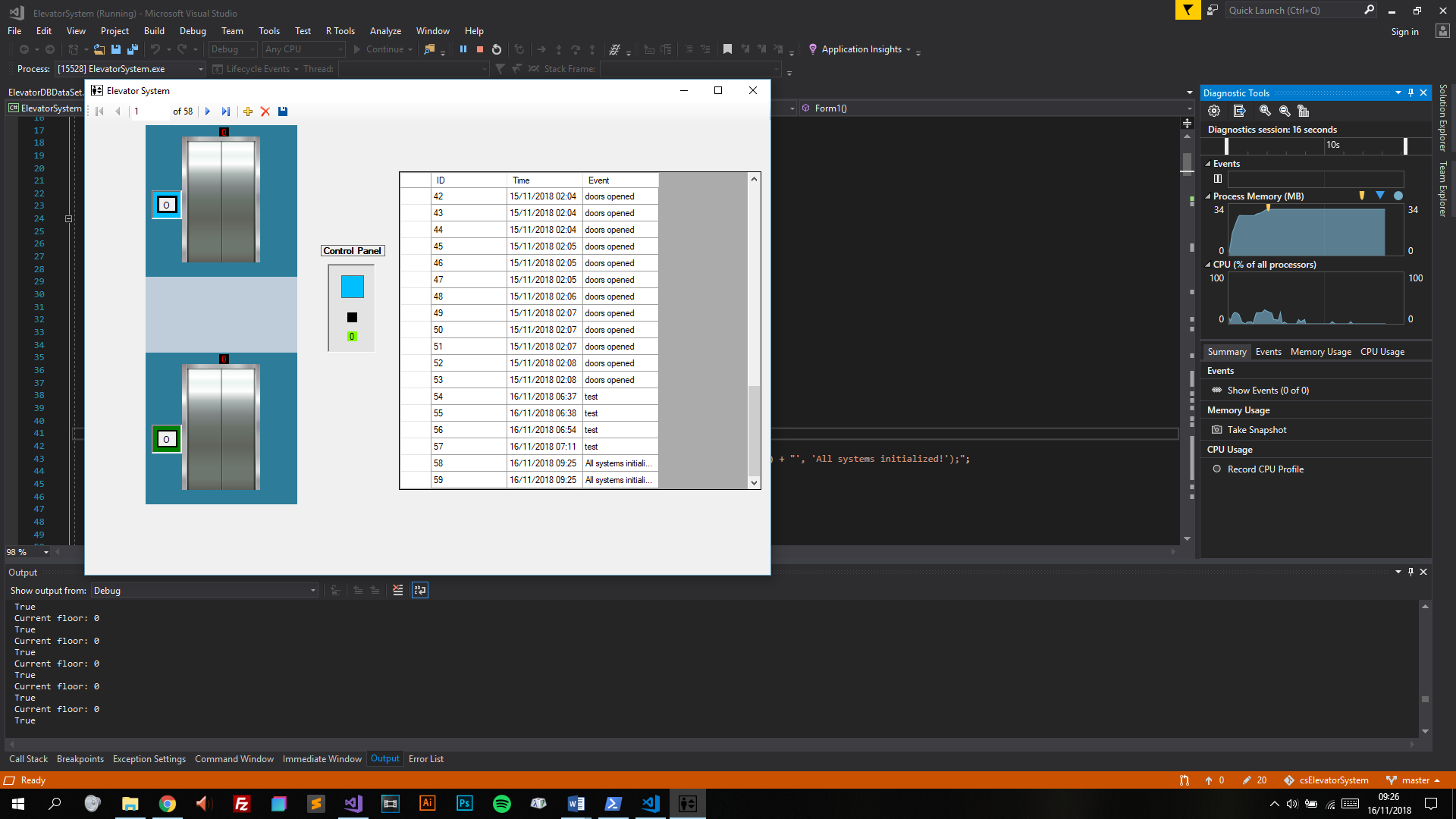


Fig 21 – Comple GUI fully operational

Here we see the entire program, with a DataGridView displaying the Database log.

Final Comments

This was a challenging and interesting project, it definitely helped me a lot learning the .NET Framework basics.

The idea is very basic, but also very logic oriented and can become a bit complex, which is what I like to design programmatically.

I had some issues trying to implement the TableAdapters and using them, so that part is still half functional. It works, but it doesn’t iterate to the next row and I tried several different ways: using the update method, insert method, try and catch update method, but nothing worked, so I settled with a mix between NonQuery SQL command and Update method. It was also renaming the DataSet and TableAdapters in the NameSpace, which was annoying, because I had to remove it every time, probably a misconfiguration.

I would also have liked to have used a different Timer library, but got a bit stuck in the implementation, so I abandoned that, because I didn’t want to ruin the main logic of the code too much.

I believe I have left room for improvement, but I am quite happy with the outcome. The design is quite good, the animations are smooth and functional, the UX is very simple and easy to use, the code is self-explanatory and is also fairly commented.

I also enjoyed having this project in a GitHub repository and making it public, I will definitely improve it and show it as a portfolio item for the .NET Framework in the future.