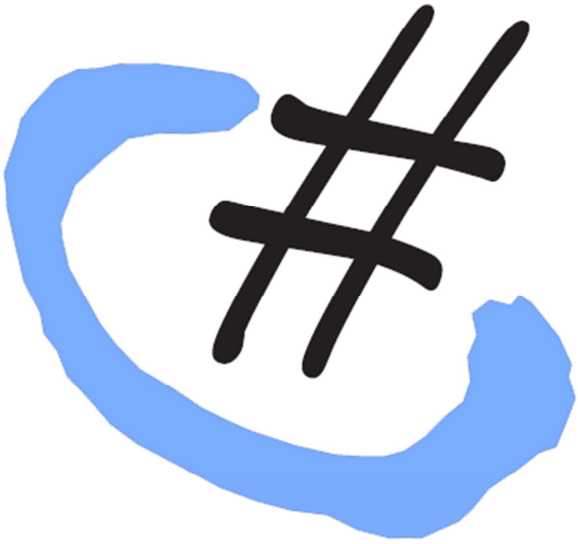
UNIVERSITY OF BIRMINGHAM

Department of Electronic, Electrical and Computer Engineering



Assignment

Name: Konstantinos Tsimpoukas

I.D.: 1130191

Tutor: Costas Constantinou

Course: MSc Embedded Systems

CONTENTS

INTRODUCTION

Object-Oriented Programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships. (Grady Booch, Robert A.Maksimchuk, Michael W.Engle, Boobi J.Young, Jim Conallen, Kelli A.Houston, 2007, p.41)

This definition combines the main concepts of the object-oriented programming. Firstly, OOP uses objects and not algorithms. Objects are the basic logical building blocks. Another important point is that each object is an instance of a class and that the classes are connected via inherited relationships.

In our project we have to design and implement a lift simulator. The solution constitutes of three units of software: Lift Control System, Graphical Animation and the External Event Generator (Figure 1).

So, these three units will have to co-operate in order to achieve the desired behaviour in our system. The Graphical User Interface will have all the appropriate components (labels, panels, buttons) in order to communicate with the current user. The user will request an action and the GUI has to interpret this request to a specific behaviour. The behaviour of the lift will be determined from the Controller. The Controller will have all the logic that is needed to model the lift.

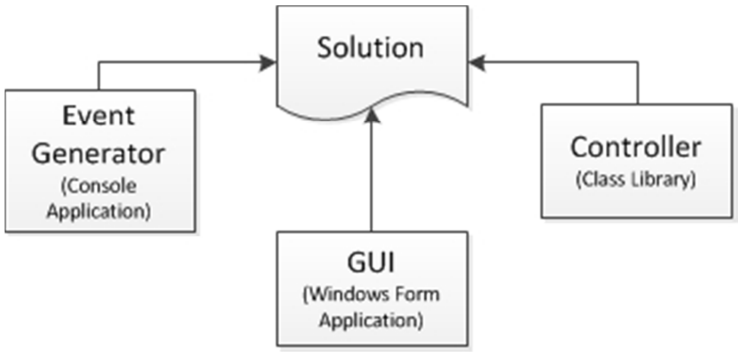


Figure 1 Components of Solution

The waterfall model was used to develop this software project. The waterfall model consists of five major phases: requirements analysis, functional specifications, design, implementation and testing. The first phase is the requirements analysis in which I was concerned with what was *needed*. In the next phase, functional specification, I had to set precisely the objective in order to meet the general requirements. The next stage, design, required a lot of thinking and solving the problems that were gradually revealed. The process of designing is accomplished in several iteration stages. Other problems you have to overcome when you are starting the project and others when you are close to the end. Finally, this stage is more time consuming compared to the other stages. The next phase is the implementation. This phase is about coding our design. Finally, in the testing stage we can verify if our design-implementation complies with the specifications.

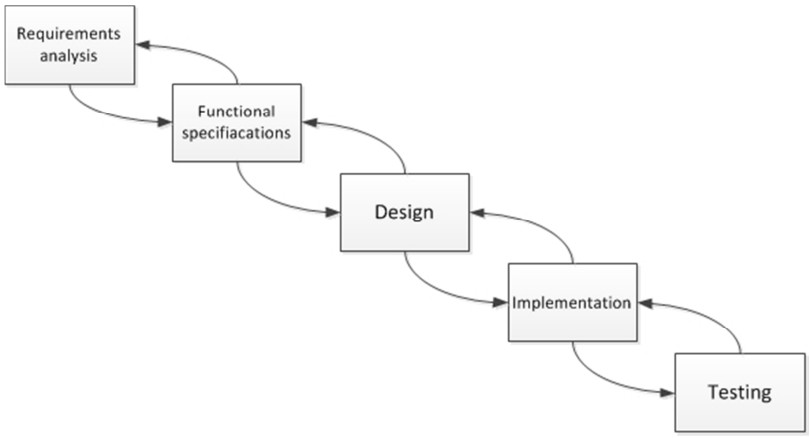


Figure 2 Waterfall Model describing the process of software development (David Budgen, 1994, p.15)

1 REQUIREMENTS & SPECIFICATIONS

The requirements and the specifications of the lift simulator project will be briefed upon in this section. In the Evaluating Specifications in the Conclusion Section, I will compare these original specifications with the accomplished ones.

1.1 GRAPHICAL USER INTERFACE

In the suggested layout, there are buttons for the floor requests in each floor and buttons inside the lift panel for the user requests. The image buttons for the floor requests are updated when a request is made to show from where the request has been generated. Also, the buttons in the lift panel are changing colours. Inside the lift there is a label and two pictures. The pictures keep updating according to the direction of the lift and the label displays the current floor. Apart from these, there are two buttons which can control the lift's speed. There is the Simulator button (Play/Pause) and the increase/decrease Mean Event Time 2 buttons. When the Simulator button is on, we produce some random requests (users and floors) and the lift is following them. With the mean event time buttons we can increase/decrease the time that the requests are being generated. Finally, we have a label which displays the number of the events that the lift is serving.

1.2 CONTROLLER

The controller must be programmed to serve on board user requests or to pick someone whose request is in the same direction with the current direction of the lift. For example, if a user enters the lift in the ground floor and requests to go to the third floor and at the same time, another user requests from the first floor to go upwards, then the lift must stop to the first floor, pick the new user, accept his new request (maybe for the second) serve it, and then continue in the same direction to the third floor. In the same scenario, while the lift is going up, if someone requests to go downwards, the lift has to serve the first request and then return to serve the other user.

The controller must be implemented as an independent thread of execution and in a Class Library project.

Another specification concerning the lift controller is that it must be implemented as an independent unit of software. The controller will give specific behaviour to the lift (GUI), therefore we have to design-implement the controller in such a way that we can alter this behaviour. For example, we can program a controller which never interrupts the current request-movement or program a controller that gives priority to the 2nd floor requests due to increased demand from this floor.

1.3 Event Generator

The event generator has to generate user and floor requests according to a simple random process.

It must also be implemented as an independent thread of executing and in a Console Application project.

2 Design & Implementation

In this section, I will describe how I designed and implemented the lift simulator. I used Visual Studio 2008 (.NET framework 3).

In Appendix II, I include my full commented code.

2.1 Graphical User Interface

Figure 1 and 2 in Appendix I, show the GUI of the lift simulator. It is like the suggested one with some add-ins that will be explained in this section.

2.1.1 GUI Description

In the right side, there are buttons for the floor requests. There are six buttons (One for the ground, one for the third floor, two for the first (up and down) and two for the second floor (up and down)). I designed some images (light blue arrows) for these buttons, for the normal condition. We have a normal condition when the buttons are not pushed. When a floor button is pushed then its image is updated to a new yellow arrow. This indicates that a request from this floor is made. The direction of the arrows shows that the user, who is waiting in this floor wants to go in this direction. Also, there are labels which display the floors (Floor 3, Floor 2, Floor 1, Ground). Next to these labels, there are yellow panels which inform the users that they are waiting in the floors for the current position of the lift.

The lift panel is placed in the centre of the form. Inside the lift panel, we can see the direction indicator which uses the same colour arrows as the floor request buttons, and the four floor buttons (I call them user requests buttons). When someone presses these buttons, the number inside the button turns to dark orange and when the request is served the number inside the button turns black again. Comparing the two figures in Appendix I, we can notice that when the lift is moving, the label inside the lift that informs the current floor is off. But when the lift stops somewhere, then this label and the other four small panels display the current position of the floor using numbers (0, 1, 2, 3).

In the left side of our form, we can see two panels, the first is for the External Event Generator and the other is for the Speed Control. In the speed panel, we notice two buttons for reducing/increasing the lift's speed and also another label which displays the current speed. This is a characteristic which was not in the suggested form. In the Event Generator panel, we can see the main play/pause button, the increase/decrease mean event time buttons and two other labels, the first displays the mean event time in milliseconds (also characteristic which was not in the suggested form) and the other displays the events that are being generated.

2.1.2 VOICE MESSAGES

Another feature that I have implemented and it wasn't in the requirements is the welcome voice message (.wav) which plays when you run the application. Also the voice messages (.wav) inform the users where the lift is currently at. Apart from these, when a request is generated, a wav file plays which informs for the direction of the lift. The voice features are not supported while the Event Generator is on. These messages are only playing during the manual control and this is because when the event generator is on, many requests are generated simultaneously, so there is a lack of time for each message to be played. There will be some videos (.avi) in the DVD which will test these capabilities. I recorded these wav files using a simple Text-to-Voice application.

2.1.3 DESIGN PROCEDURE

Inside the GUI all the code for the buttons, the labels and all the other graphical components is generated automatically. Therefore, we have to write only the code for the event handlers. Here we will focus on the code that is written in the Windows Form Application.

The first thing that I wanted to do was to generate a request that the lift followed. With a while loop checking the position of the lift's panel was not difficult. The next problem was that the panel moved very quickly. So I had to introduce delay to this movement. My first implementation used the Thread.Sleep(milliseconds) method but this created many problems. Mainly because the maximum duration of running time was passing in sleeping mode. For example, when the lift was moving (while the while loop with the Thread.Sleep() Method was executing) it wasn't possible to interact with the GUI. So the program was slow and not friendly to the user. In the next implementation, I used the Timer Class which solved many problems.

Finally, I used two timers. One is handling all the upward movements of the lift and the other is handling the downward movements. The logic of my design is the following: when a request is raised, the event-handler checks the current position of the lift and calls the appropriate timer (timer1 for going up and timer2 for going down), at this point, the form thread is interacting with the controller thread and gives the correct permission, the form reads the permission and chooses a procedure for this permission. This procedure moves the lift smoothly and sets some flags. It also updates the appropriate button images and direction images. An analytic example of a user second floor request is given using UML (activity diagram) in Figure 4.

In Figure 3, we can see a sequence diagram which shows the major flow of events when a request is produced from a user.

Figure 5 shows the procedure when we have two or more requests.

The logic sequence that is going to be followed depends mainly on the current position of the lift. As a result, there are many if statements and while loops which check for the lift's position. That was the reason that I declared four public constants (FLOOR3, FLOOR2, FLOOR1, GROUND) which represent the position of the four floors in the beginning of the form application.

For the control of the event generator I declared a public string which holds the current state of the generator (PLAY/PAUSE). This variable is initialized as PAUSE because when the program is starting I want the generator to be inactive. So when we push the PLAY button the image turns to the PAUSE image and the state to PLAY, when the generator is playing and PAUSE is clicked, the image is updating to PLAY and the state to PAUSE.

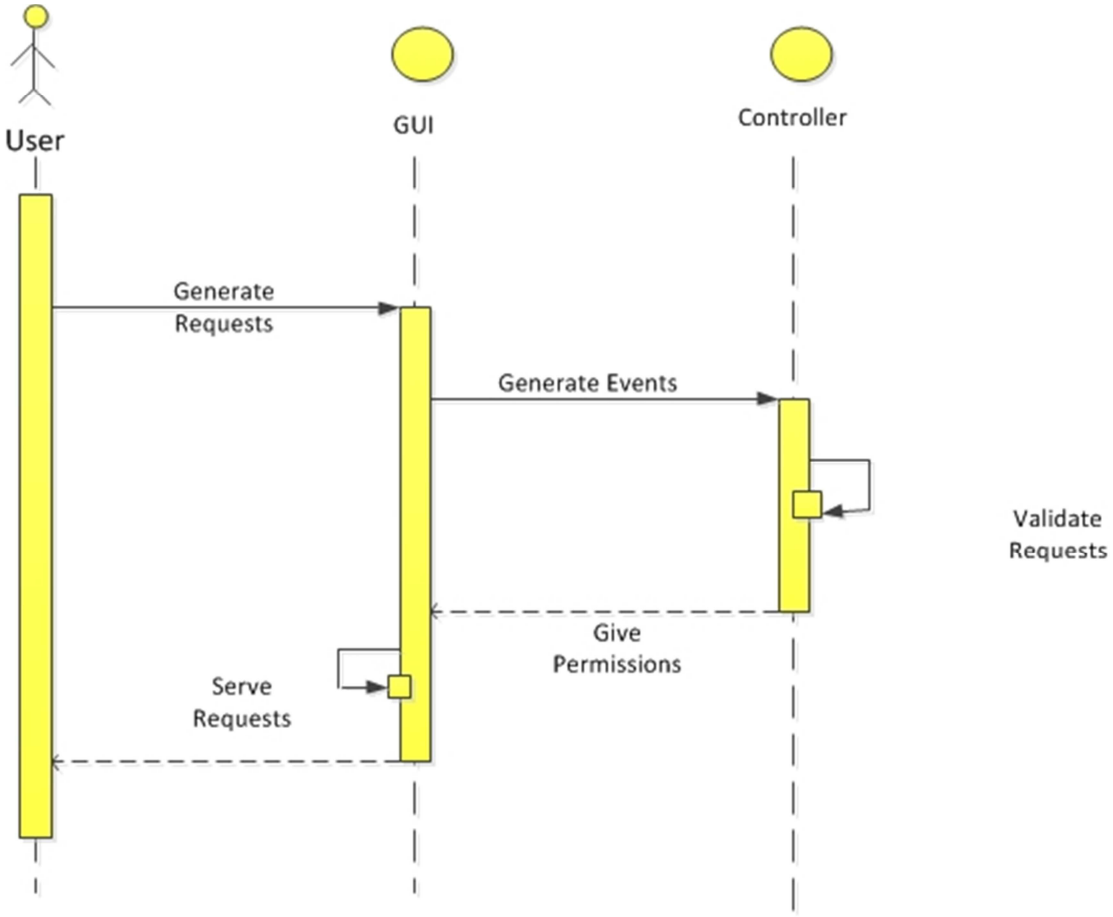


Figure 3 Basic object interaction for serving requests generated by Users (Sequence Diagram)

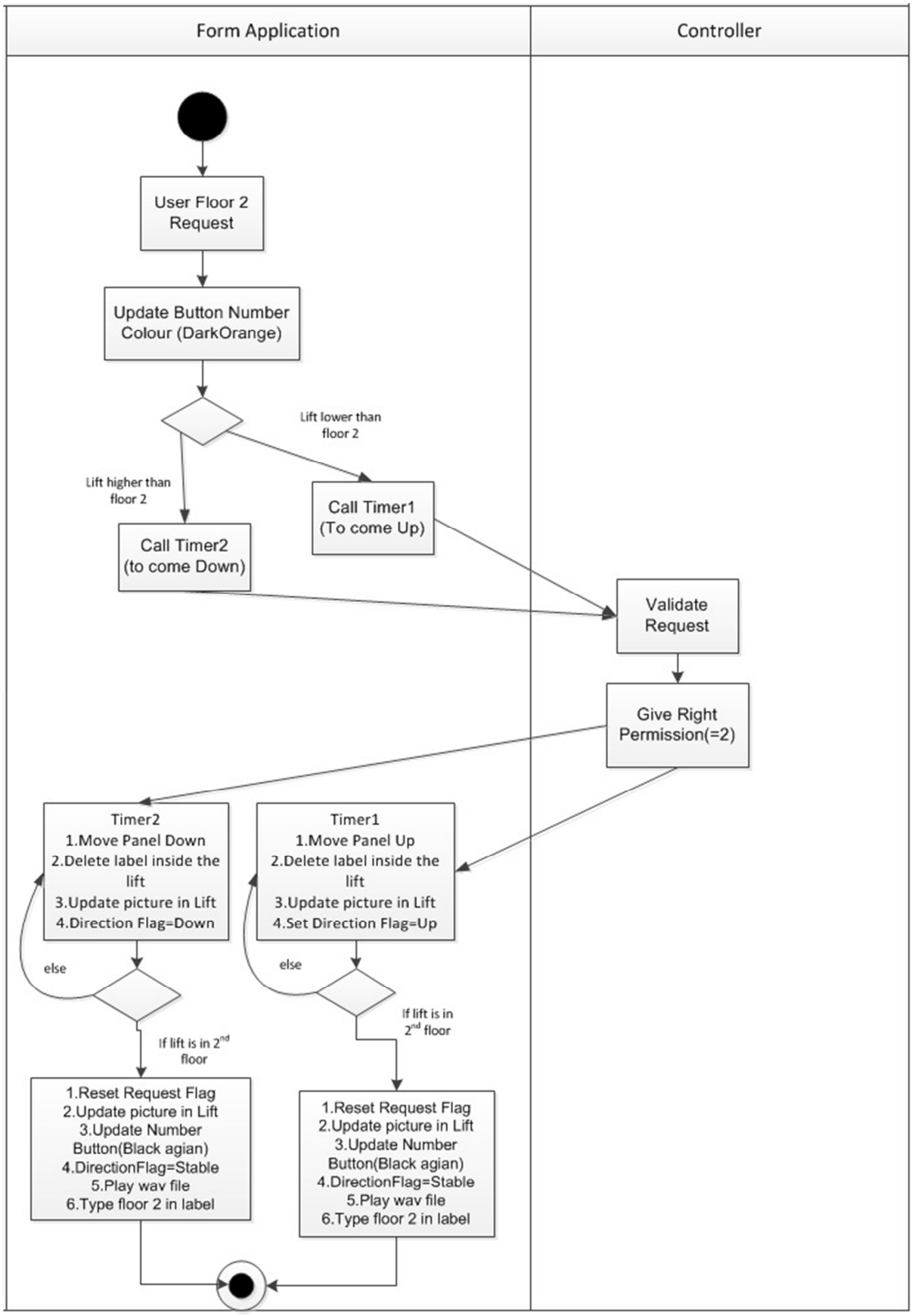


Figure 4 Single User Request (Activity Diagram)

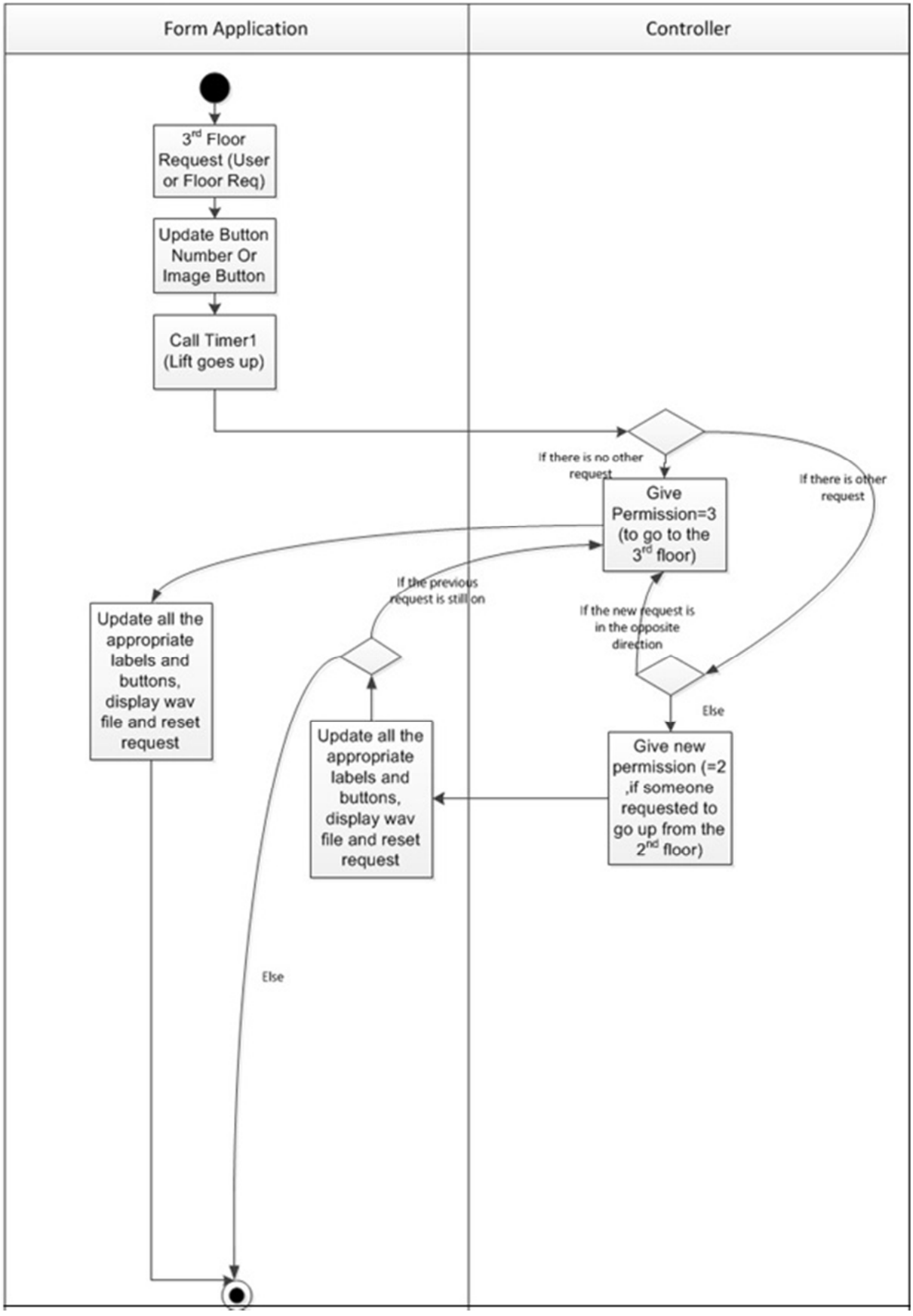


Figure 5 Serving two requests (Activity Diagram)

2.2 CONTROLLER

The controller is the most important component of our software project. This object is communicating with the Windows Form Application and the Event Generator, in order to succeed the desired behaviour according to the specifications.

The Controller was implemented in a Class Library project which was referenced to the GUI. Inside this project, I built the controller class. An object of this class is being instantiated in the Form Application.

2.2.1 BASIC ARCHITECTURE

The logic of the controller is included in a single method (ControllerMethod()) which is always executing due to a while(true) loop. This is the basic idea. So the code inside the method is always executing in another thread (controllerThread) that is set up in the Windows Form Application.

```
public class Controller

{

	Initialization/declaration of the basic flags/sensor signals

	public void ControllerMethod()

	{

		while(true)

		{

			Code that controls the GUI

			Checking requests and giving the right permissions speed control

		}

	}

}
```

2.2.2 SIGNALS/FLAGS

In the controller class, the flags/sensor signals that communicate and interact with the GUI are being declared. In this section, I will analyse these signals/flags and their purpose.

2.2.2.1 SPEED VARIABLE AND FLAGS

Initially, we notice the public (integer) speed and two public (boolean) flags (increaseSpeedFlag and decreaseSpeedFlag). The controller is responsible for the speed at which the lift will move from floor to floor. So, the speed of the lift can be manipulated by clicking the plus/minus button as per requirement. Then the increaseSpeedFlag/decreaseSpeedFlag will be on respectively and then the control will move to the controller. The controller will read this request and it will give the permission to change the variable speed. The initial value for the speed is set on 5 and it ranges from 1 (slowest)

to 10 (fastest). Figure 6 illustrates the algorithmic procedure that is followed when a reduce speed request is generated.

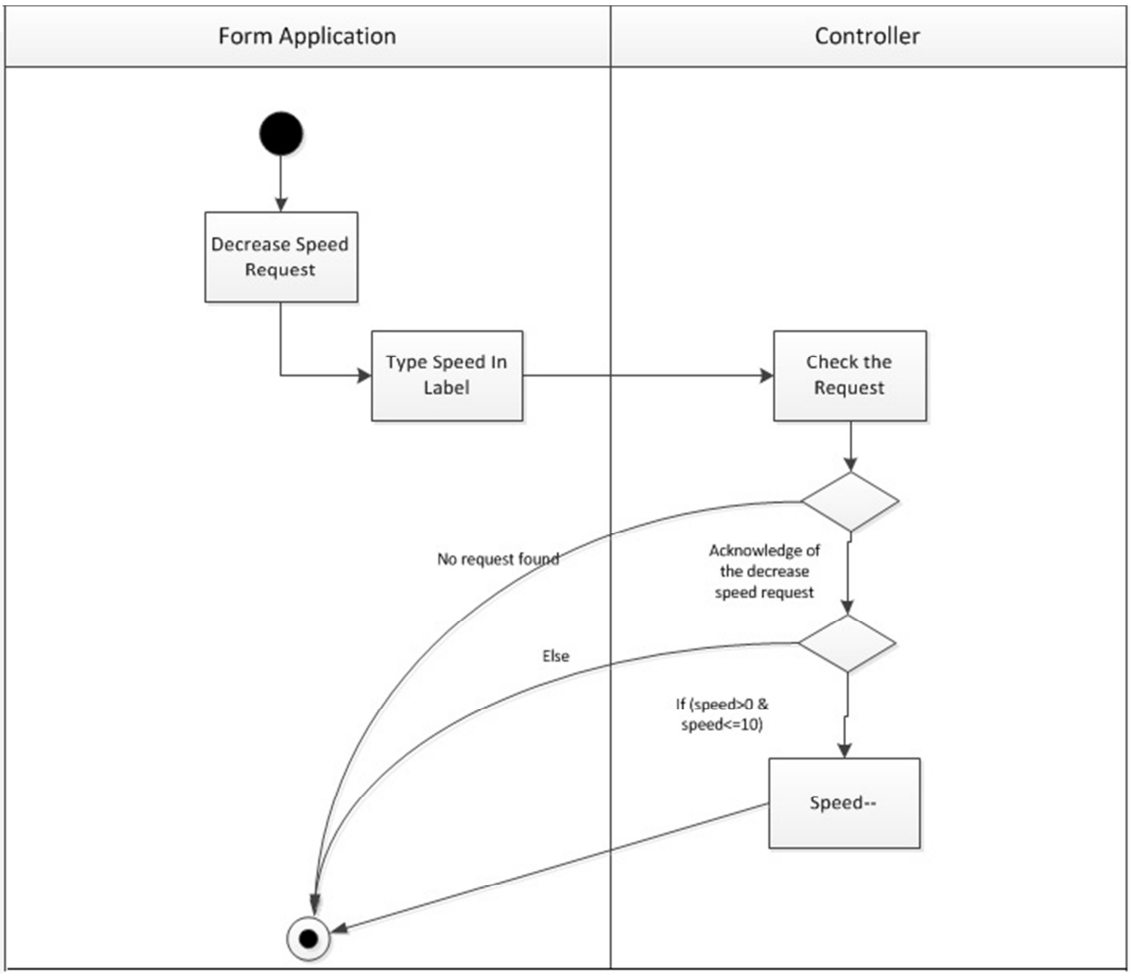


Figure 6 Decrease Speed Request (Activity Diagram)

2.2.2.2 User and Floor Request flags, Permission & Floor Flag

Again, in the declaration/initialization field in the beginning of the controller class, the request flags are following. We have ten request flags, the same number with the request buttons. So there are four user requests and six floor requests. Also, there is the direction flag (string), which takes three values: "DOWN","UP" and "STABLE", the floor flag (also string), which takes four values: "Ground","1st Floor","2nd Floor" and "3rd Floor" and the permission variable (integer). The permission takes 5 values: 0,1,2,3 and 5. The meaning of values 0, 1, 2 and 3 is that the GUI (the timers) takes the permission to move the lift to the ground floor, first floor, second floor and third floor respectively. When permission is 5 there is no action taken by the lift.

User requests flags	Floor requests flags
1. userGroundRequestFlag 2. userFirstFloorRequestFlag 3. userSecondFloorRequestFlag 4. userThirdFloorRequestFlag	1. floorGroundRequestFlag 2. floorFirstFloorRequestFlagDown 3. floorFirstFloorRequestFlagUp 4. floorSecondFloorRequestFlagDown 5. floorSecondFloorRequestFlagUp 6. floorThirdFloorRequestFlag

In Figure 3, we presented the main idea of communication between the GUI and the Controller. The requests are enabled from the button events and the controller has the logic to serve them giving the correct priorities. Figure 7 illustrates the mechanism with which the controller returns the right permissions to the Form. These permissions are checked from the timers which are responsible for the lift's movement.

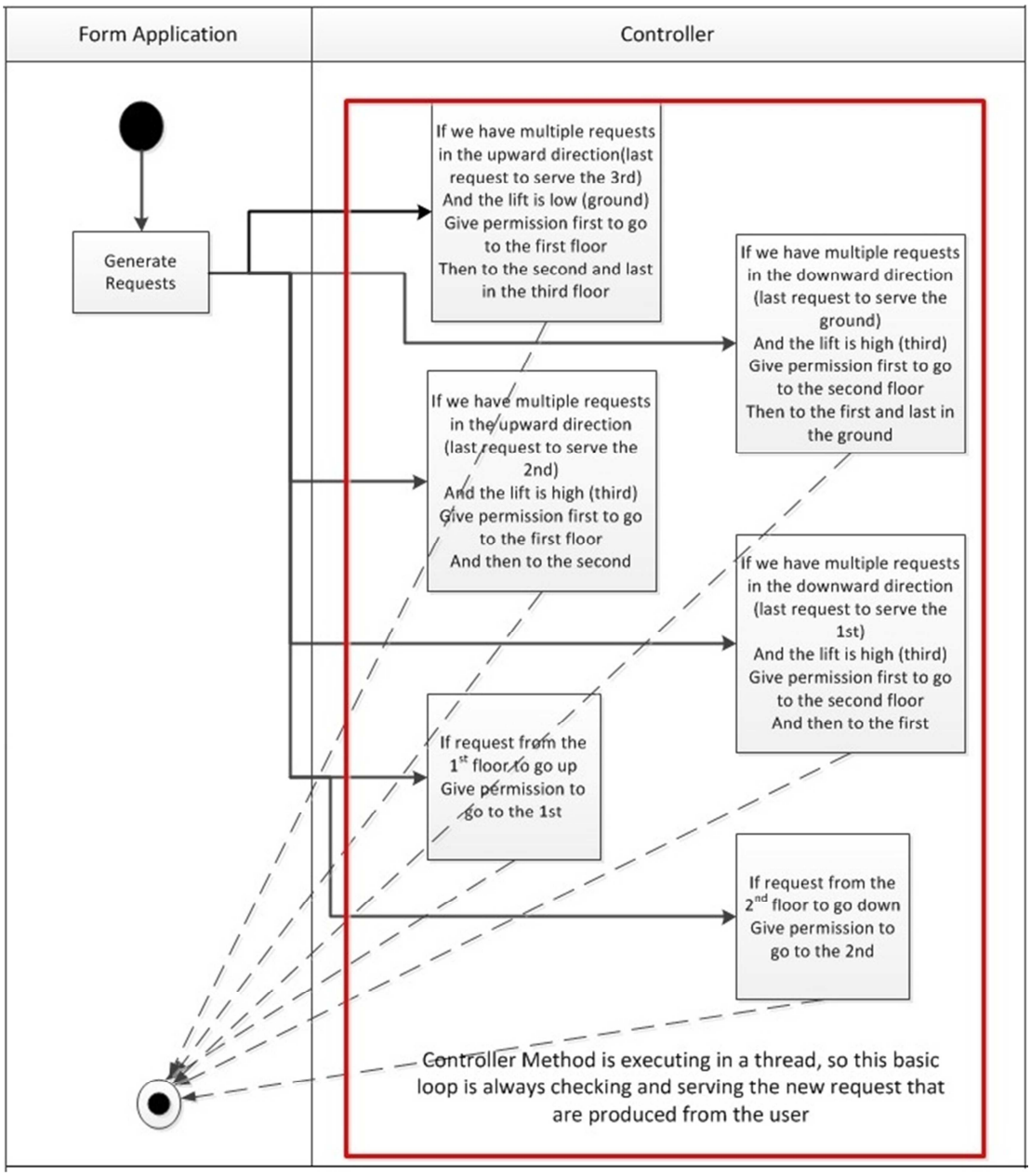


Figure 7 Controller Logic

2.2.3 Request Memorization

After implementing the above, the major problem was the request memorization.

To solve this problem, I placed if statements that were checking if there are other requests activated and not served. So when the lift stops in each floor, it checks if there are any other requests. If there are un-served requests, the program calls the appropriate timer. This part of the code is implemented

inside the Form Application, but it has to do with the general logic of implementing the controller. This is the reason why I present it here. In the next lines, I present how I implemented request memorization. Below we can see the way that this checking is happening when the lift reaches the first floor, inside timer1. The same logic is implemented for the other floors.

```
timer1()  //timer for upward movements
{
      If (permission is on to go to 1st floor)
      {
            If (lift's position lower than the floor 1 level)
            {
            Move the lift upwards, Reset labels, Update direction image, Set
            directionFlag="UP"
            }
            If (lift reached 1st floor)
            {
            Stop the current movement, Reset Request, Update direction image,
            Update button images and colours, Display labels and play Wav files,
            Wait for a while in this floor
                  If (there are requests from above waiting to be served)
                  {
                        Call timer1 to continue in the same direction, fetch
                        permissions from the controller thread
                  }
                  Else if (there are requests waiting to be served from below)
                  {
                        Call timer2, change direction of the lift, fetch
                        permissions from the controller thread
                  }
            }
      }
}
```

2.2.4 Debugging/Secondary Flags

The last flag/signal that was presented was the permission signal. After this, some flags named Secondary-Debugging follow (flag, flagy1, flagy2, flagy3, flagy4 which are integers). I used these flags to correct the behaviour of the lift.

2.2.4.1 Flag For Circle Requests

I introduced flag (flag) in order to hold some requests that from the existing code were wrongly reset. The problem was that when we had the two requests active from the 2nd floor (both the floor requests) and a request for the 3rd floor, the lift was first approaching the 2nd floor resetting both of the requests from the 2nd floor and then continuing to the second floor. The ideal behaviour is to reach the 2nd floor, reset the up request, continue to the third and then return to serve the down floor request (see in Figure 8). Therefore, I had to preserve active the down request and the image to indicate that this request is un-served.

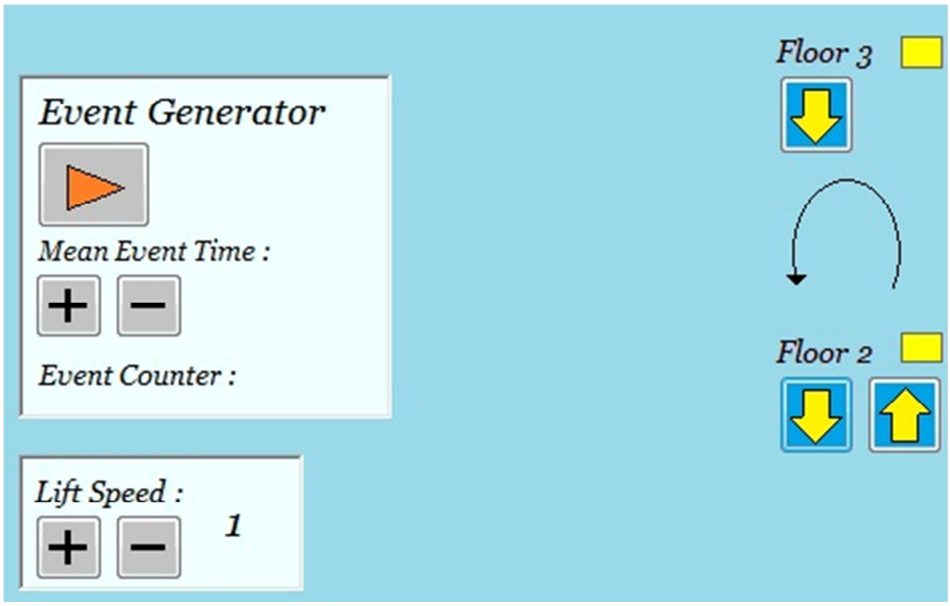


Figure 8 Circle Requests

So, when the button for the down request (in this occasion in the 2nd floor) is clicked, the flag equals to 2. Now inside timer1, when the lift stops on the second floor, we check if there is request to continue to the third floor, if there is such a request and the flag=2, this means that we have a circle request (as I described it earlier, Figure 8), so we keep the down request on and the button image as it was. After exiting this check (it is an if statement) we reset the flag, so the next time that we will have a request like that, the behaviour will be the same. The same logic followed to debug the other circle requests, using the same flag with different values. Circle requests are tested and presented in the videos that are included in the DVD.

2.2.4.2 Flagy flags

The four last flags (flagy1, flagy2, flagy3, flagy4) are used because the lift in some occasions changed direction while other requests, in the same direction, were on. So I added another condition in the while loops of the controller to check these new flags.

The previous while loops in the controller were like this: (while loop for up requests, with the last in the 3rd floor)

```
if (there are requests in the up direction, and the last is from the 3rd floor)

{

	while (requests from the 3rd floor are on)

	{

	Give permission for the 3rd floor

		while (requests from the 2nd floor are on, up direction requests)

		//Here was the second problem that I mentioned earlier, due to this
		while loop, when the lift served the 2nd floor and then moved towards
		the 3rd floor, if a request from the 2nd was coming, the lift changed
		its direction in the middle of the 2nd and 3rd floor and served this
		new request first and then continued to the 3rd floor, that was the
		reason that I introduced the flagy flags.

		{

			Give permission for the 2nd floor

			//the same problem here in this while loop

				while (requests from the 1st floor are on, up direction
				requests)

				{

					Give permission for the 1st floor

				}

		}

			//the same problem here in this while loop

		while (requests from the 1st floor are on, up direction requests)

		{

			Give permission for the 1st floor

		}

	}

}
```

So the implementation of these flags (flagys) inside these while loops statements (above), is as follows:

```
if (there are requests in the up direction, and the last is from the 3rd floor)

{

	while (requests from the 3rd floor are on)
```

```
        {
        Give permission for the 3rd floor
                while (requests from the 2nd floor are on, up direction requests and
                flagy==4 )
                {
                        Give permission for the 2nd floor
                                while (requests from the 1st floor are on, up direction
                                requests and flagy==1)
                                {
                                        Give permission for the 1st floor
                                }
                }
                while (requests from the 1st floor are on, up direction requests and
                flagy==1)
                {
                        Give permission for the 1st floor
                }
        }
}
```

The changes in the logic are underlined. With these changes in the conditions in the if statements, permission will only be granted if the right flag (flagy) has the proper value. So when the button in the 2nd floor is clicked (or the button from inside the lift for the 2nd floor) for going up, the flagy4 is becoming equal to 4 and when the timer1 is moving the lift upwards, toward the 3rd floor, this flag (flagy4) is becoming 0 again (initial value, false value). The same methodology followed for the other flagy flags.

2.3 EVENT GENERATOR

For the event generator, I used a third timer in the Form Application and a Console Application projects. The timer has an adjustable interval period from the plus/minus mean event time buttons. The simple random class is producing random numbers inside the Console Application. So when we click the Event Generator button, the timer and the generator thread starts, when we push the same button for the second time, the timer and the generator thread stop/suspend respectively. Figure 8 illustrates what happens when the Play button of the Event Generator is on. The loop that the Figure illustrates is executing until the Pause button is clicked. When the Pause button is clicked, the thread is suspended, timer3 stops, counter is zeroed and the image in the button is again updated to the Play image.

Inside the event generator (Console Application project) we declare and integer variable to hold the next generated integer from the random object. Timer3 is reading this value and makes a basic filtering. We trigger an event, only if the new choice is different from the previous one and also if the

new choice refers to a different floor from the previous one. I implemented this filter because I wanted a better distribution of the generated events among the floors.

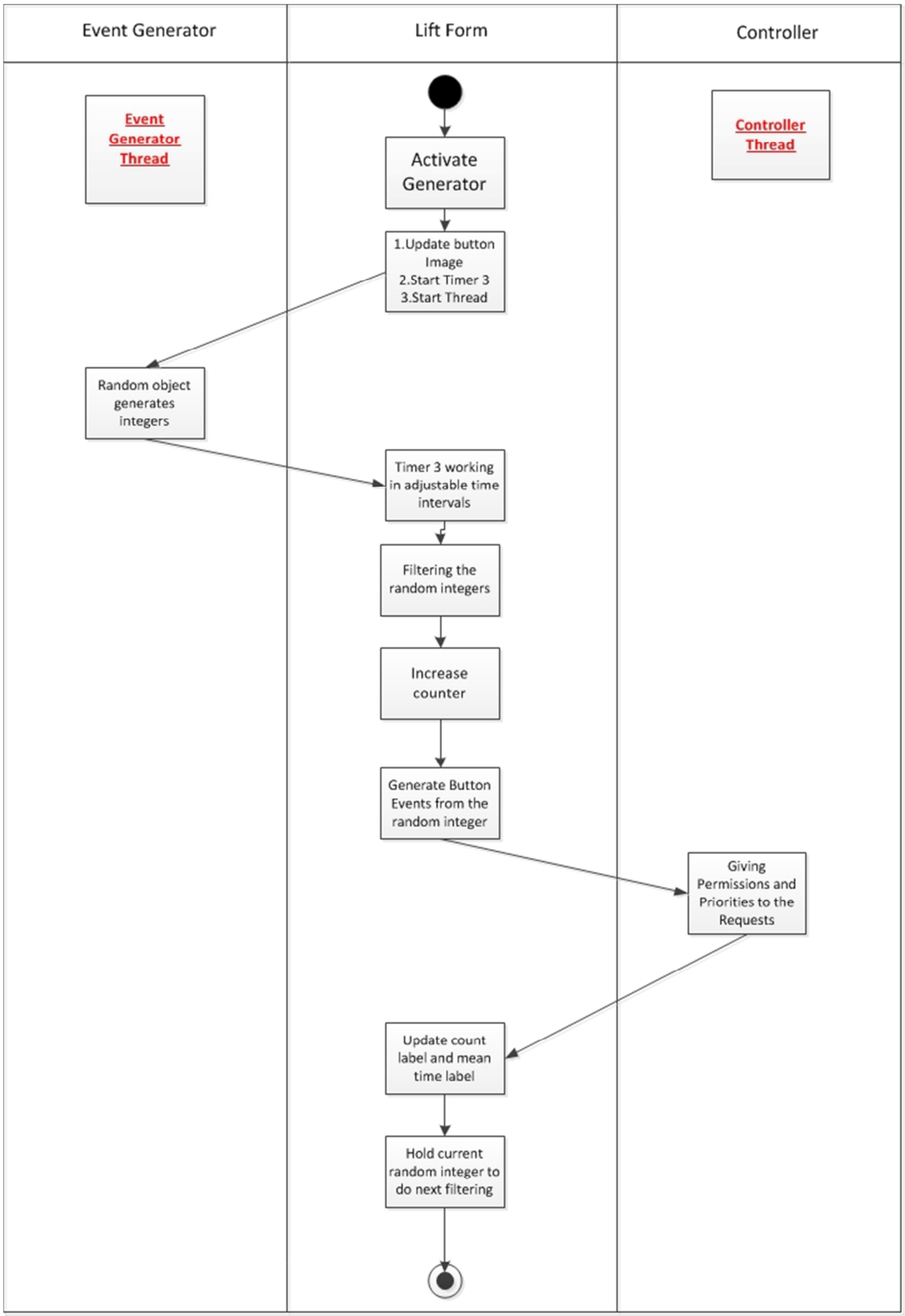


Figure 8 Event Generator-GUI-Controller Interaction (Activity Diagram)

3 TESTING DESIGN

The testing of the project is made through the videos that I have in the DVD. I used the CamStudio application to video record (.avi) the lift's GUI.

In our software project, there are two basic modes/functionalities. The first is when the GUI is interacting only with the controller (Manual mode) and the second is when the simulator is on and all the objects and threads are running together (Simulator mode).

In this point, I have to notice that the video recorder application is introducing some delay in the movement of the lift and a distortion of the real GUI. Only by running the form project we can see the actual response and behaviour of the system. However, the videos test the response of the system in all the possible modes and combinations of requests.

3.1 MANUAL MODE

For testing this mode I have included five videos in the DVD:

1.singleRequestsChangingLiftSpeed

This video tests the system's response for single requests while we change the speed of the lift.

2.doubleRequestsChangingLiftSpeed

Here we test the response of the system when we have two requests to serve. Also, here we increase/decrease the lift's speed.

3.tripleRequests

Here the requests are more demanding. The lift has to serve three requests. So, here we can see if the lift is behaving in the right way. The controller plays a critical role in serving the requests in the right order.

4.multipleRequests

Here things are getting more demanding. The lift has to serve the requests that are in the same direction and then alter its direction to serve the requests that are waiting to be served in the opposite direction.

5.circleRequests

I have discussed these requests in the Design Section. For example, if the lift stops on the 3rd floor and we have requests from a user inside the lift to go to the ground and in the same time two more requests, one from the second floor to go downwards and another also from the 2nd floor to go upwards, the lift has to stop in the second, serve the request with the down direction, continue to the ground and then return to serve the other request from the 2nd floor (upward direction). So in this video, I test the response of the system in such requests.

3.2 SIMULATOR MODE

For the Simulator mode, there is one video in the video folder in the DVD. In this video, we can see the system's response for various mean event times, and for various speeds. Furthermore, we test the functionality of the PLAY/PAUSE button by stopping the simulator and turning it on again. We notice that the counter is zeroed. So, when we click the PLAY button again, the counter is beginning from zero. Also in Figure 2 in Appendix I, we can see (by noticing the event counter) that our system is stable and can serve a great amount of requests.

Also, I have already mentioned that the voice messages are only playing in the manual mode and this because in the Simulator mode we have the option to increase speed and in the same time decrease the mean event time, so there is not enough time space for the voice messages to be played. That's the reason that I decided to have them active only in the Manual Mode.

4 CONCLUSIONS

4.1 IMPROVEMENTS AND EXTENSIONS

As I already presented in the Design Section I have introduced some extensions in the functionality of the lift simulator. These extensions/improvements are as follows:

- Voice message which informs the users for the current floor. (only in the manual mode)
- Voice message which informs the direction of the lift. (only in the manual mode)
- Welcome voice message.
- Label displaying the current speed of the lift.
- Label displaying the current mean event time.
- Panels displaying the current position of the lift in each floor.

We could also introduce more voice messages like "doors are opening" and "doors are closing". But in order to implement them, we have to decrease the lift's speed and also increase the time that the lift waits in each floor.

Furthermore, a nice implementation would be if we could represent users in our GUI that they will make their requests either from the floors or inside the lift. For example, we can have human figures waiting on the floors to be served and human figures that are already being served in the lift (inside the lift's panel). Each figure could correspond to a specific request. This implementation could be very close to reality. As a result, we can easily improve the functionality of our controller because we would have a graphical representation of the users.

Finally, another idea would be if we can model the behaviour of two or three lifts from a single controller.

4.2 EVALUATING SPECIFICATIONS

The general specifications of the software project are met. The lift is moving smoothly from floor to floor. The lift is obeying the requests and also is putting them in the right priority, serving them in the

way that was described in the Requirements & Specification Section. The behaviour of the simulator is also good. However, during the Simulator Mode we can notice some behaviour that is not complied with the requirements. This is because the time at which the requests are generated is very crucial.

4.3 SUMMING UP

The most valuable achievement that I gained was the experience in OOP. My previous experience with OOP was only with Java but not at such a level. During this project, I realised the usefulness of OOP in designing more complex systems. Also, it made me understand the advantages and the disadvantages that OOP has in comparison with the classical programming approach (from languages like C and Fortran). For example, a great characteristic of OOP is the re-usability of software components called objects (like the new object that I created in the Event Generator rnd, from the Random Class).

In addition, I conceived the concepts of classes and objects clearly. I understood how you can give characteristics and functionalities to a new class and how you can use it in your implementation by creating new objects (like I've done with the controller).

In this project, I was also introduced to multithreading programming. The designer has to understand how multithreading works because he has to adjust his design/implementation to this programming approach. Multithreading can enable more features in an application. So, the application can be more user-friendly and can include more unique characteristics.

Manipulating and setting graphical user interfaces was also a new experience for me. In this project, I gained experience in using customised buttons, labels which inform us for current values, moving panels and inserting/updating images and voice messages when an event is triggered. Moreover, this project was event-based. So, I learned how to handle multiple events and requests.

5 APPENDIX I

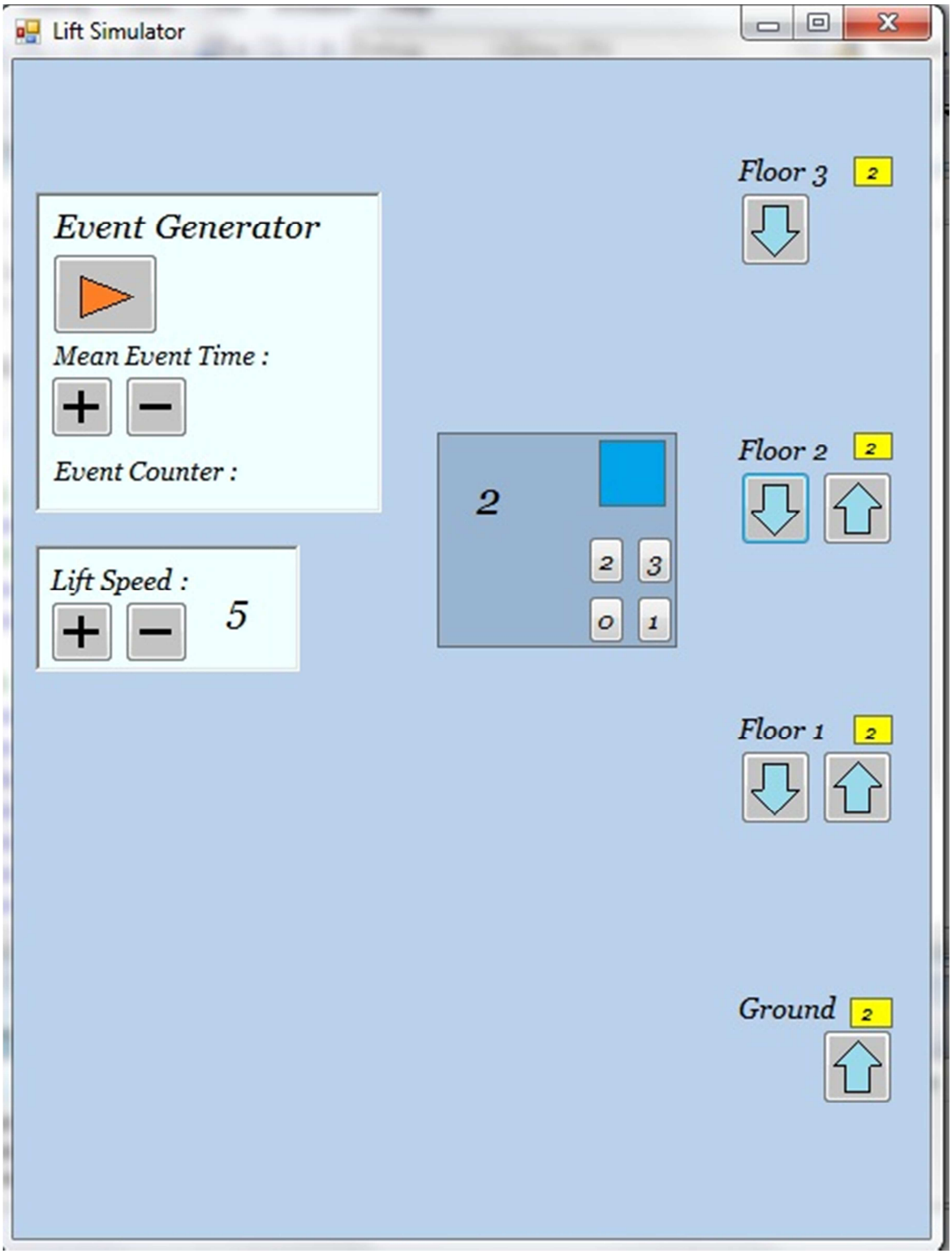


Figure 1 Lift Simulator Form (Steady)

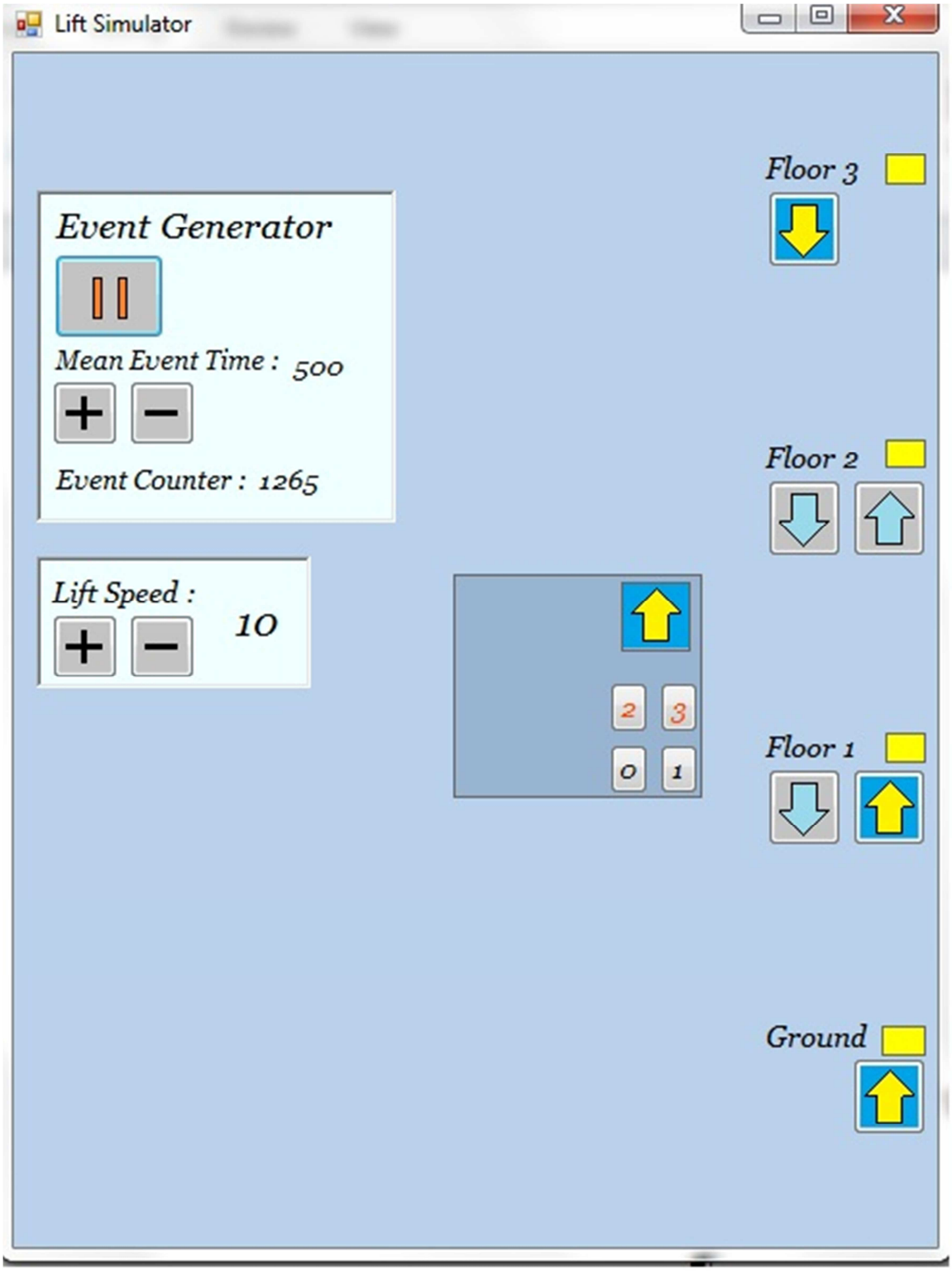


Figure 2 Generator On (Moving)

6 APPENDIX II

6.1 FORM APPLICATION CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.Diagnostics;
using System.Media;
using System.Threading;
using Controller1;
using Generator;


namespace FormLiftSimulator
{
    public partial class Form1 : Form
    {
        //Second Basic Controller Thread
        Thread controllerThread;
        Thread generatorThread;

        //If i don't put System.Windows.Forms I ll have an error, because
there are 2 Timer Classes (Threading & Forms)
        //Timer Objects
        System.Windows.Forms.Timer myTimer1;
        System.Windows.Forms.Timer myTimer2;
        System.Windows.Forms.Timer myTimer3;

        //Controller and EventGenerator Objects
        Controller myController;
        EventGenerator myEventGenerator;

        //Constants of the Location.Y's of the floors
        private const int FLOOR3 = 50;
        private const int FLOOR2 = 200;
        private const int FLOOR1 = 350;
        private const int GROUND = 500;

//--variables for the Event Generator--------------//

        //Generator State
        public string generatorState;  //{"PLAY","PAUSE"} 2-States

        //Generator Meant Event Time &
        public int meantTime = 1000;   //initial mean time value
        public int  counter= 0;
        public int previousbuttonNumber = 10;  //here this initialization ,
and 10 because is a ground request, so i don't want a request like this
because the lift starts from the ground floor

//-------------End-of-Declarations----------------------//
```

```
//--------------------Start-----------------------------------//


        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //Initializing Pictures and Buttons
            pictureBox1.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-v.1\FormLiftSimulator\Images\blue.png");
            button10.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
            button9.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
            button7.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
            button5.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
            button6.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
            button8.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");

            //play welcome wav "welcome to the simulator"
            playWelcome();

            //initializing the labels in the lift and on the floors
            label5.Text = "0";
            label13.Text = "0";
            label14.Text = "0";
            label15.Text = "0";
            label16.Text = "0";

            //Initializing Generator's State
            generatorState = "PAUSE";

            //The two other basic objects of the project(Controller and
EventGenerator)
            myController = new Controller();
            myEventGenerator = new EventGenerator();

            //Setting up the timers
            myTimer1 = new System.Windows.Forms.Timer();
            myTimer1.Interval = 25;  //steady delay for timer 1
            myTimer1.Tick += timer1_Tick;
            myTimer2 = new System.Windows.Forms.Timer();
            myTimer2.Interval = 25;   //steady delay for timer 2
            myTimer2.Tick += timer2_Tick;
            myTimer3 = new System.Windows.Forms.Timer();
            myTimer3.Interval = meantTime;   //adjustable delay for timer 3
            myTimer3.Tick += timer3_Tick;
```

```
            //Type First given speed value
            label7.Text = Convert.ToString(myController.speed);

            //Controller Thread is executing in parallel with the Form Code
            controllerThread = new Thread(new
ThreadStart(myController.ControllerMethod));
            controllerThread.Start();

            //initialize the directionFlag
            myController.directionFlag = "STABLE";
        }

//-----------------------------Timers----------------Timers----------------
------------------------//
    //-----------------------------Timers-------------------Timers---------
-------------------------//


        //------------------------Timer1--------------------------//
        //------------------------Timer1--------------------------//

                            //----Going-Up----//

        private void timer1_Tick(object sender, EventArgs e)
        {

            if (myController.permission == 3)
            {

                if (panel1.Top > FLOOR3)
                {
                        panel1.Top -= myController.speed;
                        resetLabels();
                        pictureBox1.Image = Image.FromFile(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                        myController.directionFlag = "UP";
                        myController.flagy4 = 0;  //reset this flag so it
can be inside this loop only once
                        myController.flagy1 = 0;
                }
                if(panel1.Top == FLOOR3)
                {
                    myTimer1.Stop();
                    myController.userThirdFloorRequestFlag = false;
                    myController.floorThirdFloorRequestFlag = false;
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\blue.png");
                    button4.ForeColor = Color.Black;

                    myController.directionFlag = "STABLE";
                    button10.Image = Image.FromFile(@"H:\Visual Basic 2008
- PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
                    SetLabelsAndPlayWav();
                    Application.DoEvents(); //better response of the form
application for displaying everuthing and especially the labels that are
displaying the current floor
                    Thread.Sleep(700); //delay in each station
```

```
                    //so when the lift stops in the 3rd check which
requests are still on (all for down movement) and call timer2
                    //implementing memory with these last if statements
                    if (myController.floorSecondFloorRequestFlagDown ||
myController.floorSecondFloorRequestFlagUp ||
myController.userSecondFloorRequestFlag
                        || myController.userFirstFloorRequestFlag ||
myController.floorFirstFloorRequestFlagDown ||
myController.floorFirstFloorRequestFlagUp
                        || myController.userGroundRequestFlag ||
myController.floorGroundRequestFlag)
                    {
                      myTimer2.Start();
                    }

                }
            }
            if (myController.permission == 1)

            {
                 if (panel1.Top > FLOOR1)
                {

                    panel1.Top -= myController.speed;
                    resetLabels();
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                    myController.directionFlag = "UP";
                }
                 if (panel1.Top == FLOOR1)
                {
                    myTimer1.Stop();
                    myController.userFirstFloorRequestFlag = false;
                    myController.floorFirstFloorRequestFlagDown = false;
                    myController.floorFirstFloorRequestFlagUp = false;
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\blue.png");
                    button3.ForeColor = Color.Black;

                    myController.directionFlag = "STABLE";
                    button7.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
                    button6.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
                    if ((myController.userThirdFloorRequestFlag ||
myController.floorThirdFloorRequestFlag
                        || myController.userSecondFloorRequestFlag ||
myController.floorSecondFloorRequestFlagUp)
                        && (myController.flag == 3))
                    {
                        myController.floorFirstFloorRequestFlagDown = true;
                        button7.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                    }
                    myController.flag = 0;
```

```
                    SetLabelsAndPlayWav();
                    Application.DoEvents(); //better response of the form
application for displaying everuthing and especially the labels that are
displaying the current floor
                    Thread.Sleep(700);

                     //when the lift stops in first check if there are
requests in the same direction to continue, else change direction
                    if (myController.userThirdFloorRequestFlag ||
myController.floorThirdFloorRequestFlag
                        || myController.userSecondFloorRequestFlag ||
myController.floorSecondFloorRequestFlagUp ||
myController.floorSecondFloorRequestFlagDown)
                    {
                       myTimer1.Start();
                    }
                    else if (myController.userGroundRequestFlag ||
myController.floorGroundRequestFlag)
                    {
                        myTimer2.Start();
                    }

                }
            }
            if (myController.permission==2)
            {
                if (panel1.Top > FLOOR2)
                {
                    panel1.Top -= myController.speed;
                    resetLabels();
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                    myController.directionFlag = "UP";
                    myController.flagy1 = 0;  //reset this flag so it can
be inside this loop only once
                }
                if (panel1.Top == FLOOR2)
                {
                    myTimer1.Stop();
                    myController.userSecondFloorRequestFlag = false;
                    myController.floorSecondFloorRequestFlagDown = false;
                    myController.floorSecondFloorRequestFlagUp = false;
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\blue.png");
                    button2.ForeColor = Color.Black;
                    myController.directionFlag = "STABLE";
                    button9.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
                    button8.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
                    if ((myController.floorThirdFloorRequestFlag ||
myController.userThirdFloorRequestFlag)
                        && (myController.flag == 2))
                    {
                        myController.floorSecondFloorRequestFlagDown =
true;
```

```
                        button9.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                    }
                    myController.flag = 0;
                    SetLabelsAndPlayWav();
                    Application.DoEvents(); //better response of the form
application for displaying everything and especially the labels that are
displaying the current floor
                    Thread.Sleep(700); //pause to the Current floor


                    //when the lift stops in 2nd check if there are
requests in the same direction to continue, else change direction
                    if (myController.userThirdFloorRequestFlag ||
myController.floorThirdFloorRequestFlag)
                    {
                       myTimer1.Start();
                    }
                    else if (myController.userFirstFloorRequestFlag ||
myController.floorFirstFloorRequestFlagDown ||
myController.floorFirstFloorRequestFlagUp
                        || myController.userGroundRequestFlag ||
myController.floorGroundRequestFlag)
                    {
                        myTimer2.Start();
                    }
                }
            }

        }

        //------------------------Timer2--------------------------//
        //------------------------Timer2--------------------------//

                            //--Going-Down--//

        private void timer2_Tick(object sender, EventArgs e)
        {

            if (myController.permission==0)
            {
                if (panel1.Top < GROUND)
                {
                    panel1.Top += myController.speed;
                    resetLabels();
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                    myController.directionFlag = "DOWN";
                    myController.flagy3 = 0;
                    myController.flagy2 = 0;
                }
                if (panel1.Top == GROUND)
                {
                    myTimer2.Stop();
                    myController.userGroundRequestFlag = false;
                    myController.floorGroundRequestFlag = false;
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\blue.png");
```

```
                    button1.ForeColor = Color.Black;
                    myController.directionFlag = "STABLE";
                    button5.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
                    SetLabelsAndPlayWav();
                    Application.DoEvents(); //better response of the form
application for displaying everuthing and especially the labels that are
displaying the current floor
                    Thread.Sleep(700);//delay

                    //ground floor now, so check if other requests are
active to move upwards
                    if(myController.userThirdFloorRequestFlag ||
myController.floorThirdFloorRequestFlag
                        || myController.userSecondFloorRequestFlag ||
myController.floorSecondFloorRequestFlagUp ||
myController.floorSecondFloorRequestFlagDown
                        || myController.userFirstFloorRequestFlag ||
myController.floorFirstFloorRequestFlagUp ||
myController.floorFirstFloorRequestFlagDown)
                    {
                        myTimer1.Start();
                    }

                }
            }


            if (myController.permission==1)
            {
                if (panel1.Top < FLOOR1)
                {
                    panel1.Top += myController.speed;
                    resetLabels();
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                    myController.directionFlag = "DOWN";
                    myController.flagy2 = 0;
                }
                if (panel1.Top == FLOOR1)
                {
                    myTimer2.Stop();
                    myController.userFirstFloorRequestFlag = false;
                    myController.floorFirstFloorRequestFlagDown = false;
                    myController.floorFirstFloorRequestFlagUp = false;
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\blue.png");
                    button3.ForeColor = Color.Black;
                    myController.directionFlag = "STABLE";
                    button7.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
                    button6.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
                    if ((myController.floorGroundRequestFlag ||
myController.userGroundRequestFlag)
                       && (myController.flag == 1))
```

```
                    {
                        myController.floorFirstFloorRequestFlagUp = true;
                        button6.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                    }
                    myController.flag = 0;
                    SetLabelsAndPlayWav();
                    Application.DoEvents();//better response of the form
application for displaying everuthing and especially the labels that are
displaying the current floor
                    Thread.Sleep(700);  //pause in each floor for 0.5 sec

                    //lift in the 1st, check if there are requests from
below to continue in the same direction(down), else change direction
                    if (myController.userGroundRequestFlag ||
myController.floorGroundRequestFlag)
                    {
                        myTimer2.Start();
                    }
                    else if (myController.userSecondFloorRequestFlag ||
myController.floorSecondFloorRequestFlagDown ||
myController.floorSecondFloorRequestFlagUp
                        || myController.userThirdFloorRequestFlag ||
myController.floorThirdFloorRequestFlag)
                    {
                        myTimer1.Start();
                    }
                }
            }
            if (myController.permission==2)
            {
                if (panel1.Top < FLOOR2)
                {
                    panel1.Top += myController.speed;
                    resetLabels();
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                    myController.directionFlag = "DOWN";
                }
                if (panel1.Top == FLOOR2)
                {
                    myTimer2.Stop();
                    myController.userSecondFloorRequestFlag = false;
                    myController.floorSecondFloorRequestFlagDown = false;
                    myController.floorSecondFloorRequestFlagUp = false;
                    pictureBox1.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\blue.png");
                    button2.ForeColor = Color.Black;
                    myController.directionFlag = "STABLE";
                    button9.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downLightBlue.png");
                    button8.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upLightBlue.png");
                    if ((myController.userGroundRequestFlag ||
myController.floorGroundRequestFlag
```

```
                        || myController.userFirstFloorRequestFlag ||
myController.floorFirstFloorRequestFlagDown)
                        && (myController.flag == 4))
                    {
                        myController.floorSecondFloorRequestFlagUp = true;
                        button8.Image = Image.FromFile(@"H:\Visual Basic
2008 - PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                    }
                    myController.flag = 0;
                    SetLabelsAndPlayWav();
                    Application.DoEvents();
                    Thread.Sleep(700);//delay

                    //lift in the 2nd, check if there are requests from
below to continue in the same direction(down), else change direction
                    if (myController.userFirstFloorRequestFlag ||
myController.floorFirstFloorRequestFlagDown ||
myController.floorFirstFloorRequestFlagUp
                        || myController.userGroundRequestFlag ||
myController.floorGroundRequestFlag)
                    {
                        myTimer2.Start();
                    }
                    else if(myController.userThirdFloorRequestFlag ||
myController.floorThirdFloorRequestFlag)
                    {
                        myTimer1.Start();
                    }

                }
            }
        }

        //---------------timer3---------------------//
        //---------------timer3---------------------//

               //----For-the-Generator-------//

        //I came up with this solution (i mean using Timer) for the Event
Generator because
        //I couldn't use the while loop (while(generatorState=="PLAY"){//do
these;}
        //I couldn't follow the way that i program the event generator in
the DllLiftSimulator

        private void timer3_Tick(object sender, EventArgs e)
        {


                myTimer3.Interval = meantTime;  //updating interval when
the timer is executing
                //I want to program a filter for the random event generator
(to be not so random)

                if (myEventGenerator.buttonNumber != previousbuttonNumber)
//filter 2 same choices in a row
                {

                    switch (myEventGenerator.buttonNumber)
                    {
```

```
                        case 1:
                            if (previousbuttonNumber != 6 &&
previousbuttonNumber != 7)
                            {
                                button2_Click(this, EventArgs.Empty);
//2nd User Request
                                counter++;
                            }
                            break;
                        case 2:
                            if (previousbuttonNumber != 8 &&
previousbuttonNumber != 9)
                            {
                                button3_Click(this, EventArgs.Empty);
//1st User Request
                                counter++;
                            }
                            break;
                        case 3:
                            if (previousbuttonNumber != 5)
                            {
                                button4_Click(this, EventArgs.Empty);
//3rd User Request
                                counter++;
                            }
                            break;
                        case 4:
                            if (previousbuttonNumber != 10)
                            {
                                button1_Click(this, EventArgs.Empty);
//Ground User Request
                                counter++;
                            }
                            break;
                        case 5:
                            if (previousbuttonNumber != 3)
                            {
                                button10_Click(this, EventArgs.Empty);
//3rd Floor Request
                                counter++;
                            }
                            break;
                        case 6:
                            if (previousbuttonNumber != 7 &&
previousbuttonNumber != 1)
                            {
                                button8_Click(this, EventArgs.Empty);
//2nd Floor Request Up
                                counter++;
                            }
                            break;
                        case 7:
                            if (previousbuttonNumber != 7 &&
previousbuttonNumber != 1)
                            {
                                button9_Click(this, EventArgs.Empty);
//2nd Floor Request Down
                                counter++;
                            }
                            break;
                        case 8:
```

```
                            if (previousbuttonNumber != 9 &&
previousbuttonNumber != 2)
                            {
                                button7_Click(this, EventArgs.Empty);
//1st Floor Request Down
                                counter++;
                            }
                            break;
                        case 9:
                            if (previousbuttonNumber != 8 &&
previousbuttonNumber != 2)
                            {
                                button6_Click(this, EventArgs.Empty);
//1st Floor Request Up
                                counter++;
                            }
                            break;
                        case 10:
                            if (previousbuttonNumber != 4)
                            {
                                button5_Click(this, EventArgs.Empty);
//Ground Floor Request
                                counter++;
                            }
                            break;

                    }

                }

                //Update Labels in the Event Generator Panel
                label12.Text = Convert.ToString(counter);
                label10.Text = Convert.ToString(meantTime);

                //Store random choice to previousbuttonNumber to do basic
filtering of the random number generation
                previousbuttonNumber = myEventGenerator.buttonNumber;


        }

    //-------------------------User-Requests------------------------------
---//
        //-----------------------User-Requests----------------------//

        private void button1_Click(object sender, EventArgs e)
        {
            //User Ground Floor Request
            button1.ForeColor = Color.OrangeRed;
            myController.userGroundRequestFlag = true;
            if(generatorState=="PAUSE")
                playGoingDown();
            myTimer2.Start();
        }

        private void button4_Click(object sender, EventArgs e)
        {
            //User Floor 3 Request
            button4.ForeColor = Color.OrangeRed;
            myController.userThirdFloorRequestFlag = true;
            if (generatorState == "PAUSE")
```

```
            playGoingUp();
        myTimer1.Start();

    }

    private void button3_Click(object sender, EventArgs e)
    {
        //User Floor 1 Request
        button3.ForeColor = Color.OrangeRed;
        if (panel1.Top > FLOOR1)
        {
            myController.userFirstFloorRequestFlag = true;
            myController.flagy1 = 1;
            if (generatorState == "PAUSE")
                playGoingUp();
            myTimer1.Start();
        }
        else if (panel1.Top < FLOOR1)
        {
            myController.userFirstFloorRequestFlag = true;
            myController.flagy3 = 3;
            if (generatorState == "PAUSE")
                playGoingDown();
            myTimer2.Start();
        }
    }

    private void button2_Click(object sender, EventArgs e)
    {
        //User Floor 2 Request
        button2.ForeColor = Color.OrangeRed;
        if (panel1.Top > FLOOR2)
        {
            myController.userSecondFloorRequestFlag = true;
            myController.flagy4 = 4;
            if (generatorState == "PAUSE")
                playGoingUp();
            myTimer1.Start();
        }
        else if (panel1.Top < FLOOR2)
        {
            myController.userSecondFloorRequestFlag = true;
            myController.flagy2 = 2;
            if (generatorState == "PAUSE")
                 playGoingDown();
            myTimer2.Start();
        }
    }


    //----------------Floor-Requests-----------------------------//
            //-----------Floor-Requests------------//

    private void button5_Click(object sender, EventArgs e)
    {
        //Floor Ground Request
        myController.floorGroundRequestFlag = true;
        button5.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
        if (generatorState == "PAUSE")
```

```
                playGoingDown();
            myTimer2.Start();
        }

        private void button10_Click(object sender, EventArgs e)
        {
            //Floor Floor 3 Request
            myController.floorThirdFloorRequestFlag = true;
            button10.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
            if (generatorState == "PAUSE")
                 playGoingUp();
            myTimer1.Start();
        }

        private void button7_Click(object sender, EventArgs e)
        {
            //Floor Floor 1 Request Down
            if (panel1.Top > FLOOR1)
            {
                myController.floorFirstFloorRequestFlagDown = true;
                myController.flag = 3;
                button7.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                if (generatorState == "PAUSE")
                    playGoingUp();
                myTimer1.Start();
            }
            else if (panel1.Top < FLOOR1)
            {
                myController.floorFirstFloorRequestFlagDown = true;
                button7.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                myController.flagy3 = 3;
                if (generatorState == "PAUSE")
                      playGoingDown();
                myTimer2.Start();
            }
        }

        private void button6_Click(object sender, EventArgs e)
        {
            //Floor Floor 1 Request Up
            if (panel1.Top > FLOOR1)
            {
                myController.floorFirstFloorRequestFlagUp = true;
                button6.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                myController.flagy1 = 1;
                if (generatorState == "PAUSE")
                    playGoingUp();
                myTimer1.Start();
            }
            else if (panel1.Top < FLOOR1)
            {
                myController.floorFirstFloorRequestFlagUp = true;
```

```
                button6.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                myController.flag = 1;
                if (generatorState == "PAUSE")
                    playGoingDown();
                myTimer2.Start();
            }
        }

        private void button9_Click(object sender, EventArgs e)
        {
            //Floor Floor 2 Request Down
            if (panel1.Top > FLOOR2)
            {
                myController.floorSecondFloorRequestFlagDown = true;
                myController.flag = 2;
                button9.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                if (generatorState == "PAUSE")
                    playGoingUp();
                myTimer1.Start();
            }
            else if (panel1.Top < FLOOR2)
            {
                myController.floorSecondFloorRequestFlagDown = true;
                button9.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\downYellow.png");
                myController.flagy2 = 2;
                if (generatorState == "PAUSE")
                     playGoingDown();
                myTimer2.Start();
            }
        }

        private void button8_Click(object sender, EventArgs e)
        {
            //Floor Floor 2 Request Up
            if (panel1.Top > FLOOR2)
            {
                myController.floorSecondFloorRequestFlagUp = true;
                button8.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                myController.flagy4 = 4;
                if (generatorState == "PAUSE")
                    playGoingUp();
                myTimer1.Start();
            }
            else if (panel1.Top < FLOOR2)
            {
                myController.floorSecondFloorRequestFlagUp = true;
                button8.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\upYellow.png");
                myController.flag = 4;
                if (generatorState == "PAUSE")
                    playGoingDown();
                myTimer2.Start();
```

```
            }
        }

//----------------------------Play---Wav----Files---Methods---------------
----------------------//

        private void play3Floor()
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\third floor.wav"))
            {
                mySoundPlayer.Play();   //Quicker than PlaySync()
            }
        }

        private void play2Floor()
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\second floor.wav"))
            {
                mySoundPlayer.Play();       //Quicker than PlaySync()
                                            //Play() uses a new Thread
            }
        }

        private void play1Floor()
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\first floor.wav"))
            {
                mySoundPlayer.Play();       //Quicker than PlaySync()
                                            //Play() uses a new Thread
            }
        }


        private void play0Floor()
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\ground floor.wav"))
            {
                mySoundPlayer.Play();       //Quicker than PlaySync()
                                            //Play() uses a new Thread
            }
        }

        private void playGoingDown()
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\going down.wav"))
            {
                mySoundPlayer.Play();       //Quicker than PlaySync()
            }
        }

        private void playGoingUp()
```

```
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\going up.wav"))
            {
                mySoundPlayer.Play();       //Quicker than PlaySync()
            }
        }
        private void playWelcome()
        {
            using (SoundPlayer mySoundPlayer = new SoundPlayer(@"H:\Visual
Basic 2008 - PROJECTS C#\LiftSimulator-
v.Timer\FormLiftSimulator\Audio\welcome.wav"))
            {
                mySoundPlayer.Play();       //Quicker than PlaySync()
            }
        }


        //----------Display--Lift's--Position-------------//
            //------Display--Lift's--Position--------//

        public void SetLabelsAndPlayWav()
        {
            if ((GROUND - 5 < panel1.Top) && (panel1.Top < GROUND + 5))
            {
                myController.floorFlag = "0";
                if (generatorState == "PAUSE")
                    play0Floor();
            }
            else if ((FLOOR1 - 5 < panel1.Top) && (panel1.Top < FLOOR1 +
5))
            {
                myController.floorFlag = "1";
                if (generatorState == "PAUSE")
                    play1Floor();
            }
            else if ((FLOOR2 - 5 < panel1.Top) && (panel1.Top < FLOOR2 +
5))
            {
                myController.floorFlag = "2";
                if (generatorState == "PAUSE")
                    play2Floor();
            }
            else if ((FLOOR3 - 5 < panel1.Top) && (panel1.Top < FLOOR3 +
5))
            {
                myController.floorFlag = "3";
                if (generatorState == "PAUSE")
                    play3Floor();
            }
            else
            {
                label5.Text = "";
                label13.Text = "";
                label14.Text = "";
                label15.Text = "";
                label16.Text = "";
            }

            label5.Text = myController.floorFlag;
```

```
            label13.Text = myController.floorFlag;
            label14.Text = myController.floorFlag;
            label15.Text = myController.floorFlag;
            label16.Text = myController.floorFlag;
        }

  //--------Reaset-Labels-Method------//

        public void resetLabels()
        {
            label5.Text = "";
            label13.Text = "";
            label14.Text = "";
            label15.Text = "";
            label16.Text = "";
        }

//-----------------------Lift--Speed--Button-Requests-------------//

        private void button11_Click(object sender, EventArgs e)
        {
            //Increase Speed Lift
            myController.increaseSpeedFlag = true;
            label7.Text = Convert.ToString(myController.speed);

        }

        private void button12_Click(object sender, EventArgs e)
        {
            //Decrease Speed Lift
            myController.decreaseSpeedFlag = true;
            label7.Text = Convert.ToString(myController.speed);
        }


        //-----------------Event-Generator-Buttons-----------------//
            //-------------Event-Generator-Buttons--------------//

        private void button13_Click(object sender, EventArgs e)
        {
            //Play/Pause Generator Button

            if (generatorState == "PAUSE")
                generatorState = "PLAY";
            else
                generatorState = "PAUSE";


            if (generatorState == "PLAY")
            {
                button13.Image = Image.FromFile(@"H:\Visual Basic 2008 -
PROJECTS C#\LiftSimulator-v.Timer-v.1\FormLiftSimulator\Images\pause.png");
                myTimer3.Start();
                generatorThread = new Thread(new
ThreadStart(myEventGenerator.produceRandomNumber));
                generatorThread.Start();

            }

            if (generatorState == "PAUSE")
                 {
```

```
                    button13.Image = Image.FromFile(@"H:\Visual Basic 2008
- PROJECTS C#\LiftSimulator-v.Timer-
v.1\FormLiftSimulator\Images\play.png");
                    myTimer3.Stop();
                    generatorThread.Suspend();
                    counter = 0;  //the  is zeroed here and in the
initialization field, when i declared this variable
                }

            }


        private void button14_Click(object sender, EventArgs e)
        {
            //increase mean time
            meantTime += 100;
        }

        private void button15_Click(object sender, EventArgs e)
        {
            //decrease mean time
            if (meantTime > 100 )
                meantTime -= 100;

        }



    }
}
```

6.2 CONTROLLER

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Controller1
{
    public class Controller
    {
        //General Variables
        public int speed = 5;
        //Speed Flags
        public bool increaseSpeedFlag = false;
        public bool decreaseSpeedFlag = false;

        //Request Flags ( FROM USERS AND FROM THE FLOORS )
        public bool userGroundRequestFlag = false;
        public bool userFirstFloorRequestFlag = false;
        public bool userSecondFloorRequestFlag = false;
        public bool userThirdFloorRequestFlag = false;
        public bool floorGroundRequestFlag = false;
        public bool floorFirstFloorRequestFlagDown = false;
        public bool floorFirstFloorRequestFlagUp = false;
```

```
        public bool floorSecondFloorRequestFlagDown = false;
        public bool floorSecondFloorRequestFlagUp = false;
        public bool floorThirdFloorRequestFlag = false;
        //Direction Flags
        public string directionFlag;  //{"DOWN","UP","STABLE"}
        //Floor Flags
        public string floorFlag;       //{"0","1","2","3"}

        //Permission
         public int permission = 5;     //permission to nowhere
{0,1,2,3,5=nowhere}

        //Secondary-Debugging Flags
         public int flag = 0;   //{0,1,2,3,4} flag values to hold some
requests (circle requests...,circle bugs...)

         public int flagy4 = 0;  // flags not to change direction of the
lift due to new requests from other direction
         public int flagy1 = 0;
         public int flagy3 = 0;
         public int flagy2 = 0;
//-------------------------Controller--Method-----------------------------
----------------------------//

        public void ControllerMethod()
        {
            while (true)
            {

                if (increaseSpeedFlag == true)
                {
                    if ((speed > 0) && (speed < 10))
                        speed++;

                    increaseSpeedFlag = false;
                }

                if (decreaseSpeedFlag == true)
                {

                    if ((speed > 1) && (speed <= 10))
                        speed--;

                    decreaseSpeedFlag = false;
                }


                if ((floorThirdFloorRequestFlag ||
userThirdFloorRequestFlag)
                    || (userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp)
                    || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp))
                {
                        while (floorThirdFloorRequestFlag ||
userThirdFloorRequestFlag)
                        {
                            permission = 3;
                            while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp) && (flagy4 == 4))
                            {
```

```
                                permission = 2;
                                while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp) && (flagy1 == 1))
                                {
                                    permission = 1;
                                }
                            }
                            while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp) && (flagy1 == 1))
                            {
                                permission = 1;
                            }

                        }
                    }


                    if ((userGroundRequestFlag || floorGroundRequestFlag)
                        || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown)
                        || (userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown))
                    {
                        while (floorGroundRequestFlag || userGroundRequestFlag)
                        {
                            permission = 0;
                            while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown) && (flagy3==3))
                            {
                                permission = 1;
                                while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown) && (flagy2 == 2))
                                {
                                    permission = 2;
                                }
                            }
                            while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown) && (flagy2 == 2))
                            {
                                permission = 2;
                            }
                        }
                    }

                    if ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp)
                       || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp))
                    {
                        while (userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp)
                        {
                            permission = 2;
                            while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp) && (flagy1 == 1))
                            {
                                permission = 1;
                            }
                        }
                    }
```

```
                if ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown)
                    || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown))
                {
                    while (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown)
                    {
                        permission = 1;
                        while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown) && (flagy2 == 2))
                        {
                            permission = 2;
                        }
                    }
                }


                    while (floorFirstFloorRequestFlagUp)
                    {
                        permission = 1;
                    }

                    while (floorSecondFloorRequestFlagDown)
                    {
                        permission = 2;
                    }

            }
        }


    }
}
```

6.3 Event Generator

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Controller1
{
    public class Controller
    {
        //General Variables
        public int speed = 5;
        //Speed Flags
        public bool increaseSpeedFlag = false;
        public bool decreaseSpeedFlag = false;

        //Request Flags ( FROM USERS AND FROM THE FLOORS )
        public bool userGroundRequestFlag = false;
        public bool userFirstFloorRequestFlag = false;
        public bool userSecondFloorRequestFlag = false;
        public bool userThirdFloorRequestFlag = false;
        public bool floorGroundRequestFlag = false;
```

```
        public bool floorFirstFloorRequestFlagDown = false;
        public bool floorFirstFloorRequestFlagUp = false;
        public bool floorSecondFloorRequestFlagDown = false;
        public bool floorSecondFloorRequestFlagUp = false;
        public bool floorThirdFloorRequestFlag = false;
        //Direction Flags
        public string directionFlag;  //{"DOWN","UP","STABLE"}
        //Floor Flags
        public string floorFlag;       //{"0","1","2","3"}

        //Permission
         public int permission = 5;     //permission to nowhere
{0,1,2,3,5=nowhere}

        //Secondary-Debugging Flags
         public int flag = 0;   //{0,1,2,3,4} flag values to hold some
requests (circle requests...,circle bugs...)

         public int flagy4 = 0;  // flags not to change direction of the
lift due to new requests from other direction
         public int flagy1 = 0;
         public int flagy3 = 0;
         public int flagy2 = 0;
//-------------------------Controller--Method-----------------------------
----------------------------//

        public void ControllerMethod()
        {
            while (true)
            {

                if (increaseSpeedFlag == true)
                {
                    if ((speed > 0) && (speed < 10))
                        speed++;

                    increaseSpeedFlag = false;
                }

                if (decreaseSpeedFlag == true)
                {

                    if ((speed > 1) && (speed <= 10))
                        speed--;

                    decreaseSpeedFlag = false;
                }


                if ((floorThirdFloorRequestFlag ||
userThirdFloorRequestFlag)
                    || (userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp)
                    || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp))
                {
                        while (floorThirdFloorRequestFlag ||
userThirdFloorRequestFlag)
                        {
                            permission = 3;
```

```
                            while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp) && (flagy4 == 4))
                            {
                                permission = 2;
                                while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp) && (flagy1 == 1))
                                {
                                    permission = 1;
                                }
                            }
                            while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp) && (flagy1 == 1))
                            {
                                permission = 1;
                            }

                    }
                }


                if ((userGroundRequestFlag || floorGroundRequestFlag)
                    || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown)
                    || (userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown))
                {
                    while (floorGroundRequestFlag || userGroundRequestFlag)
                    {
                        permission = 0;
                        while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown) && (flagy3==3))
                        {
                            permission = 1;
                            while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown) && (flagy2 == 2))
                            {
                                permission = 2;
                            }
                        }
                        while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown) && (flagy2 == 2))
                        {
                            permission = 2;
                        }
                    }
                }

                if ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp)
                   || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp))
                {
                    while (userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagUp)
                    {
                        permission = 2;
                        while ((userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagUp) && (flagy1 == 1))
                        {
                            permission = 1;
                        }
```

```
                    }
                }


                if ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown)
                    || (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown))
                {
                    while (userFirstFloorRequestFlag ||
floorFirstFloorRequestFlagDown)
                    {
                        permission = 1;
                        while ((userSecondFloorRequestFlag ||
floorSecondFloorRequestFlagDown) && (flagy2 == 2))
                        {
                            permission = 2;
                        }
                    }
                }


                    while (floorFirstFloorRequestFlagUp)
                    {
                        permission = 1;
                    }

                    while (floorSecondFloorRequestFlagDown)
                    {
                        permission = 2;
                    }

            }
        }


    }
}
```