Step 10: States and State Machines

This assigment is done in class on (monday) 03-24-14 AND (friday)03-21-14.

This assignment is due for monday class 03-24-14 AND for the firday class 03-21-14.

Resources

Please refer to this section for resources to help complete the step.

- The GarageLib Library Documentation

Getting Started

Today we will be working with a *garage door simulator*. A garage door is a simple thing. It has a motor than can be set to off, up, or down. When set to up, the door opens. When set to down, the door closes. There is a button inside the garage and in the car. If pressed when the door is closed, it causes it to open. If pressed when the door is open, it causes it to close. If pressed when the door is moving, it reverses the direction of the door. There is an infrared beam at the base of the door. If broken when the door is moving down, it reverses the door, raising it so it does not close on someone. If you try to close the door and the beam is broken, it will not close.

It will be your responsibility to write the control software for the garage door. You need to control the motor to open and close the door. This is an example of a *controls application*, where you will be responsible for making a hardware system work correctly through code you write.

Control systems like this control real-world systems with real-world people involved. If your code does not work correctly, you may kill or injure someone.

This project must be compiled and executed on a machine with an X-Server and wxWidgets. This is true for the Linux lab machines. I make no guarantee about personal machines and it is unlikely that you can work on this project remotely given the animation it uses. It is best to work on this project in the lab.

First, create a directory named step10 in your Linux account. You can put this under cse251 or under the root directory, wherever you want to put it. Remember, you use the mkdir command to create a directory:

```
mkdir step10
cd step10
```

Then download the file Garage.tar.gz into that directory. You can right click on the link in the browser and choose Save Link As... to save the file.

Then, using a terminal window in the directory where you put Garage.tar.gz, do this command:

```
tar xvzf Garage.tar.gz
```

This will create a directory called **Garage**. Change into that directory using cd.

Please edit the file garage.c and add your name to it in a comment.

Compiling and Running the Project

This project uses a *make file* to automate the building process. To compile the project, do not use the gcc command directly. Instead, use this command:

```
make garage32
```

for 32 bit machines or

```
make garage64
```

for 64 bit machines (See Project 2 for details)

You should see output something like this:

```
cbowen@ubuntu:~/cse251/Garage$ make
gcc -c -o garage.o garage.c
g++ -o garage garage.o -lm libGarageLib.a `wx-config --libs`
```

This is building your application for you. Whenever you need to recompile, just type: "make". We will be covering make files later.

To run the program for the first time, type:

```
./garage
```

It won't do much of anything yet. To see how it should work, type this command:

```
./garage-solution
```

Or, if you are on a 64-bit maching

```
./garage-solution64
```

This is an example solution. Play with it a bit to see how it works. The button can be clicked with the mouse. If you click near the bottom of the door, a cat appears who blocks the infrared beam. Try clicking while the door is closing. Double clicking in that area make the cat stay there. Double click again to make it go away.

There is online documentation for the GarageLib simulator available.

1. Creating the State Machine

We are first going to reproduce the process from the lecture. Using gedit, edit the file: garage.c. This is the C program you will be working on today. First, add these lines to the beginning of your program, before the main function:

```
/* Our possible garage door states */
#define DoorClosed 1
#define DoorOpening 2
#define DoorOpen 3
#define DoorClosing 4
```

Remember that **#define** is just a way to saying a number using a name instead. When I put DoorClosed in my program, it is the same thing as putting in a 1. But, this allows our states to have names, making our program easier to read and understand.

Now, we need a variable to store the state and we need to set it to an initial state. Add this variable to the main function:

```
    int state = DoorClosed;
```

This simply says that when your system starts up the door is closed.

We will have a function associated with each state. This function tells what to do when in that state. Add this state function for the DoorClosed state:

#define statements should go at the beginning of your .c file, but after any #includes. Here's what mine looks like:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "GarageLib.h"

/* Our possible garage door states */
#define DoorClosed 1
#define DoorOpening 2
#define DoorOpen 3
#define DoorClosing 4
```

```
void StateDoorClosed(int *state)
{

}
```

Be sure to add the *declaration* for this function. Be sure you are compiling often to be certain you did not create an error.

We need to add a switch statement to the *control loop* to call our state function. Add this code inside the *control loop*:

```
        switch(state)
        {
        case DoorClosed:
            StateDoorClosed(&state);
            break;
        }
```

In an embedded application, a control loop is a continuous loop that runs as long as the system does. It is in this loop that we make decisions about what the system should do.

Your control loop should look like this after you add this code:

```
    while(IsGarageRunning())
    {
        switch(state)
        {
        case DoorClosed:
            StateDoorClosed(&state);
            break;
        }

    }
```

This won't do anything, yet. Note how I am passing the state *by reference* to the function, so it can change it if necessary. Remember the class period on pointers? This is an important use for them.

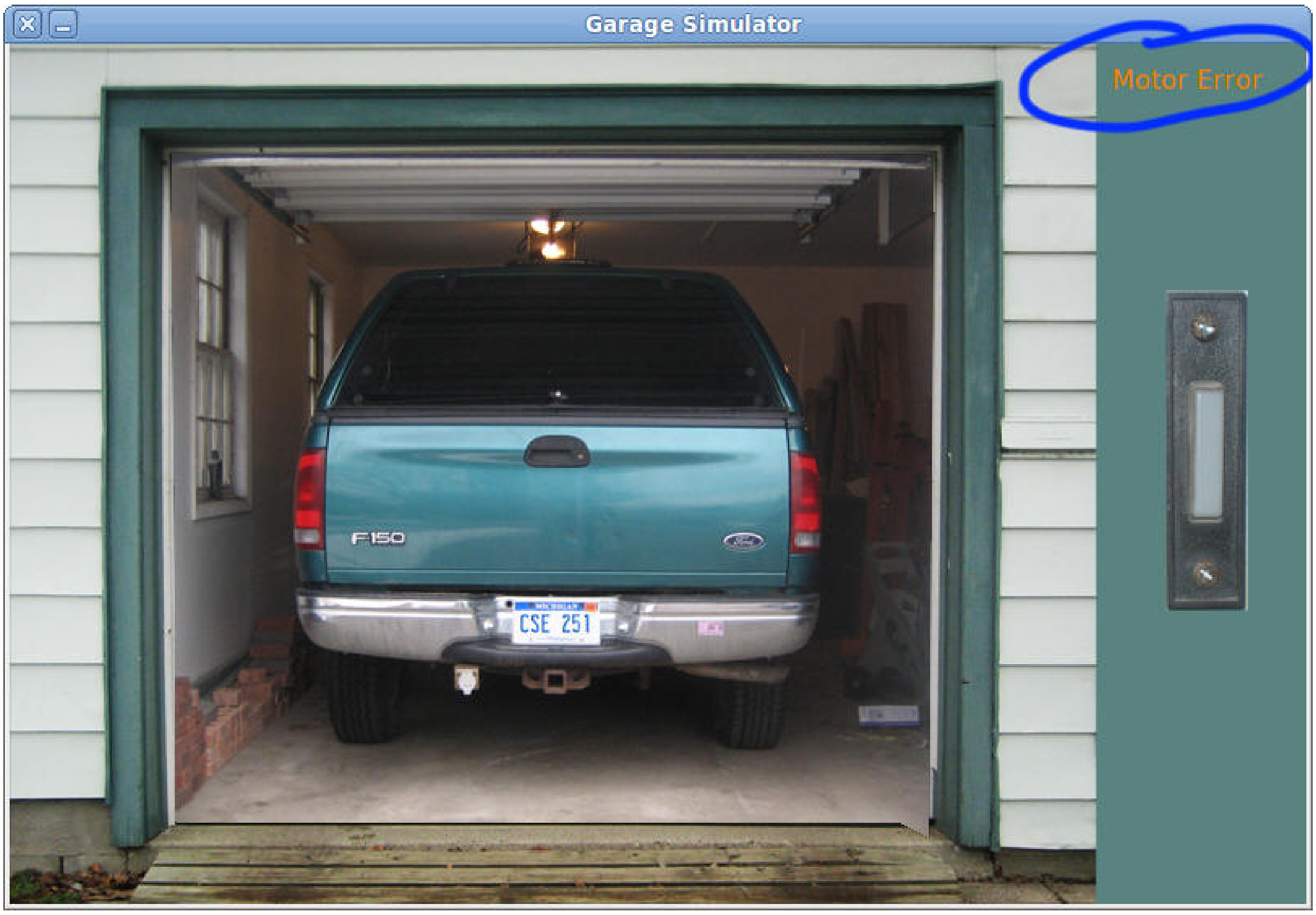
So, what do we do in this state? If the button has not been pressed, we do nothing. But, if it has been pressed, we need to turn on the motor and change the state to indicate that the door is opening. We can determine if the button has been pressed by calling WasButtonPressed, a function in the GarageLib library. To turn on the motor, we call SetMotorPower with a value of 1 for up. And, of course, we change the state. Add this code to the StateDoorClosed function:

```
    if(WasButtonPressed())
    {
        SetMotorPower(1);
        *state = DoorOpening;
    }
```

Your StateDoorClosed function should look like this:

```
void StateDoorClosed(int *state)
{
    if(WasButtonPressed())
    {
        SetMotorPower(1);
        *state = DoorOpening;
    }
}
```

Run the program. When you click on the button, the door should start to rise. Wait until it gets to the top and see what happens:



The important thing to understand here is that the function StateDoorClosed() is being called continuously as long as the door is closed. In fact, it is called by the program 1000 times per second just like the control loop in the Elevator simulator in Project 2. Each time it is called it is responsible for making a decision about what to do when in that state. Often that is nothing, but if some *event* occurs, like the button is pressed or the infrared beam is broken, it has to do something. In this case it starts the motor and goes to a different state.

The motor ran until it reached the top, but you never turned it off, so it just kept on running until it failed. We get a motor error. What we want to do is turn off the motor when it gets all of the way open. This decision will be made while we are in the DoorOpening state. So, let's create a function for the DoorOpening state:

```
void StateDoorOpening(int *state)
{

}
```

And, add code to the control loop to call this function when the state is DoorOpening:

```
    while(IsGarageRunning())
    {
        switch(state)
        {
        case DoorClosed:
            StateDoorClosed(&state);
            break;

        case DoorOpening:
            StateDoorOpening(&state);
            break;
        }

    }
```

See how we are adding a new **case** to the switch statement for the new state. We will eventually do this for every state.

Okay, what should we do in this state? We will detect that the door has finished opening by comparing the current value of GetDoorPosition with the maximum amount we should open the door. If we are done opening the door, we will go to the DoorOpen state. The amount we want to open is (DoorHeight - DoorTolerance). DoorHeight is the maximum height we can open the door. The problem is, moving things like doors don't stop instantaneously, so we have to turn off the motor a bit short of the maximum height. The value DoorTolerance (2cm in case you are wondering) has been defined by our bosses as the point where we turn the motor off. Add this code to the StateDoorOpening function:

```
    if(GetDoorPosition() >= (DoorHeight - DoorTolerance))
    {
        SetMotorPower(0);
        *state = DoorOpen;
    }
```

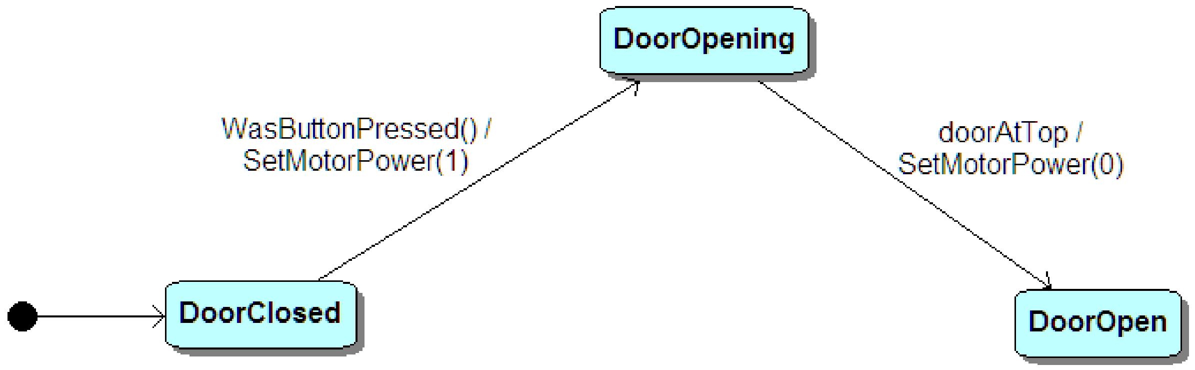
This should run now and stop in time to avoid the motor error. It's stuck when it gets to the top, though. The button does nothing now.

The simulation is meant to be fairly realistic and the door moves at a fairly normal pace of 0.25 meters per second. But, if you are impatient, you can change the speed the door moves in the program by adding this line of code right **after** the call to GarageStartup:

```
        SetDoorSpeed(1);
```

This switches the speed to 1 meter per second. Be careful making this too fast or the simulation will not work right.

Here is a state diagram that illustrates what we have now:


DoorOpening
WasButtonPressed() / SetMotorPower(1)
doorAtTop / SetMotorPower(0)
DoorClosed
DoorOpen

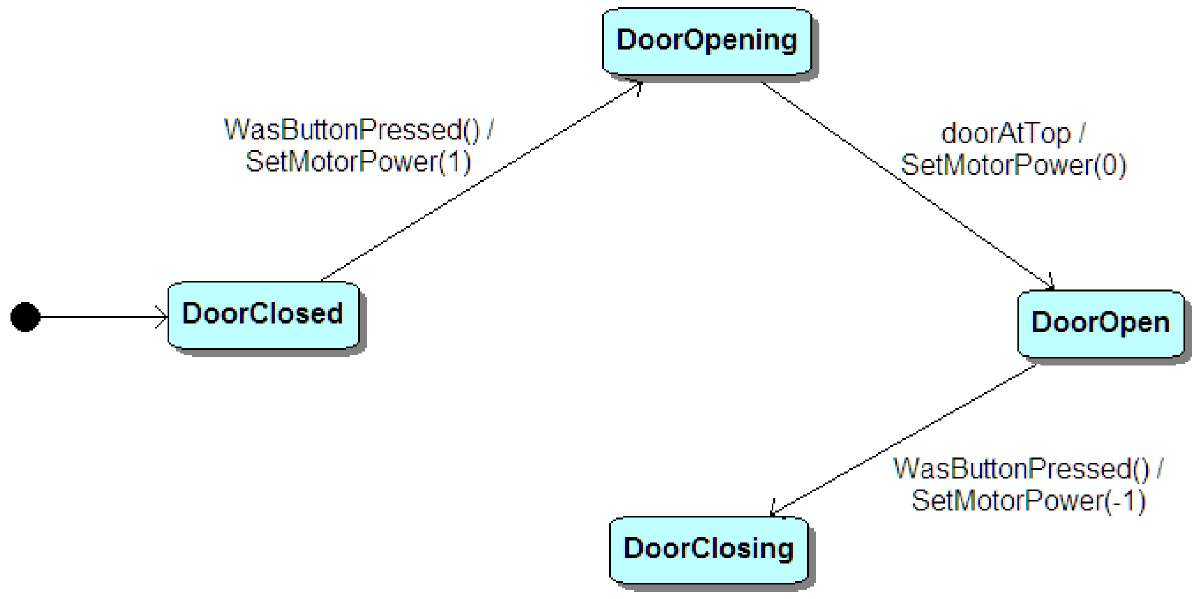
Note how I sometimes use simple names for the "trigger" event instead of trying to write some code.

So, let's handle the DoorOpen state. Here's the state function we will need:

```
void StateDoorOpen(int *state)
{
    if(WasButtonPressed())
    {
        SetMotorPower(-1);
        *state = DoorClosing;
    }
}
```

Add the code to the switch statement in the control loop to call this. When you run the program is should open the door correctly and you should be able to also close it, though when it closes all of the way you will get a motor error and the program becomes non-functional.

Right now our state diagram looks like this:


DoorOpening
WasButtonPressed() / SetMotorPower(1)
doorAtTop / SetMotorPower(0)
DoorClosed
DoorOpen
WasButtonPressed() / SetMotorPower(-1)
DoorClosing

This brings us to a couple important observations about state diagrams and state machines:

1. Don't dead end in any state unless you expect your system to enter some dead state.
2. Handle every state.

We are not handling the state DoorClosing, yet. So add this function for that state:

```
void StateDoorClosing(int *state)
{
    if(GetDoorPosition() < DoorTolerance)
    {
        SetMotorPower(0);
        *state = DoorClosed;
    }
}
```

Add the code to the control loop switch statement to call this and you should have a working garage door.

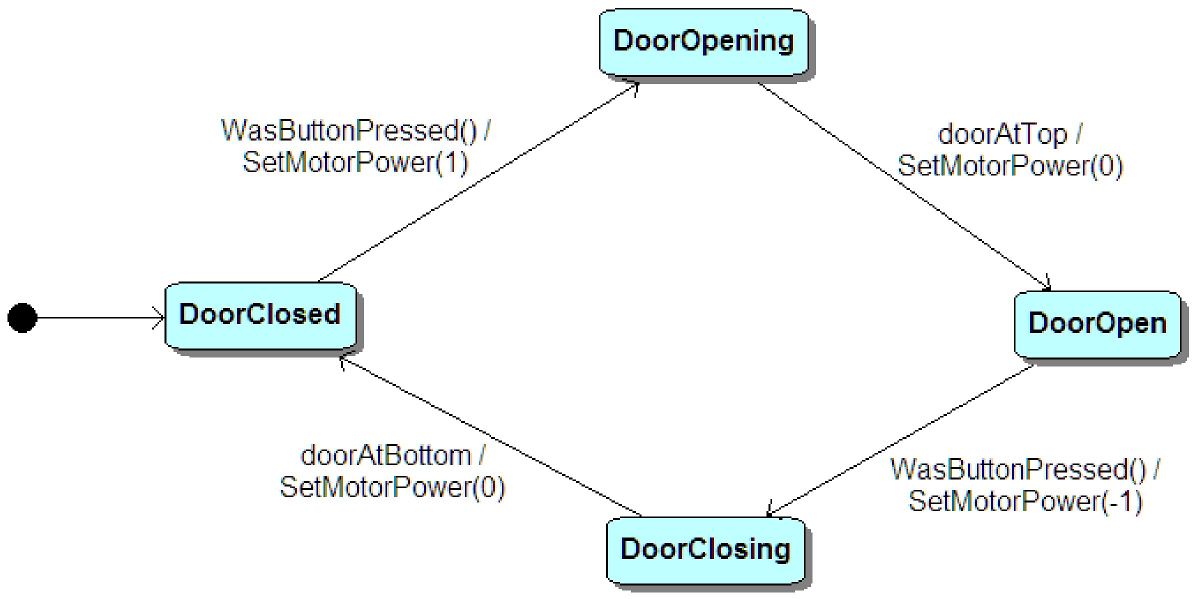
Just to be sure, your control loop should look like this:

```
    while(IsGarageRunning())
    {
        switch(state)
        {
        case DoorClosed:
            StateDoorClosed(&state);
            break;

        case DoorOpening:
            StateDoorOpening(&state);
            break;

        case DoorOpen:
            StateDoorOpen(&state);
            break;

        case DoorClosing:
            StateDoorClosing(&state);
            break;
        }

    }
```

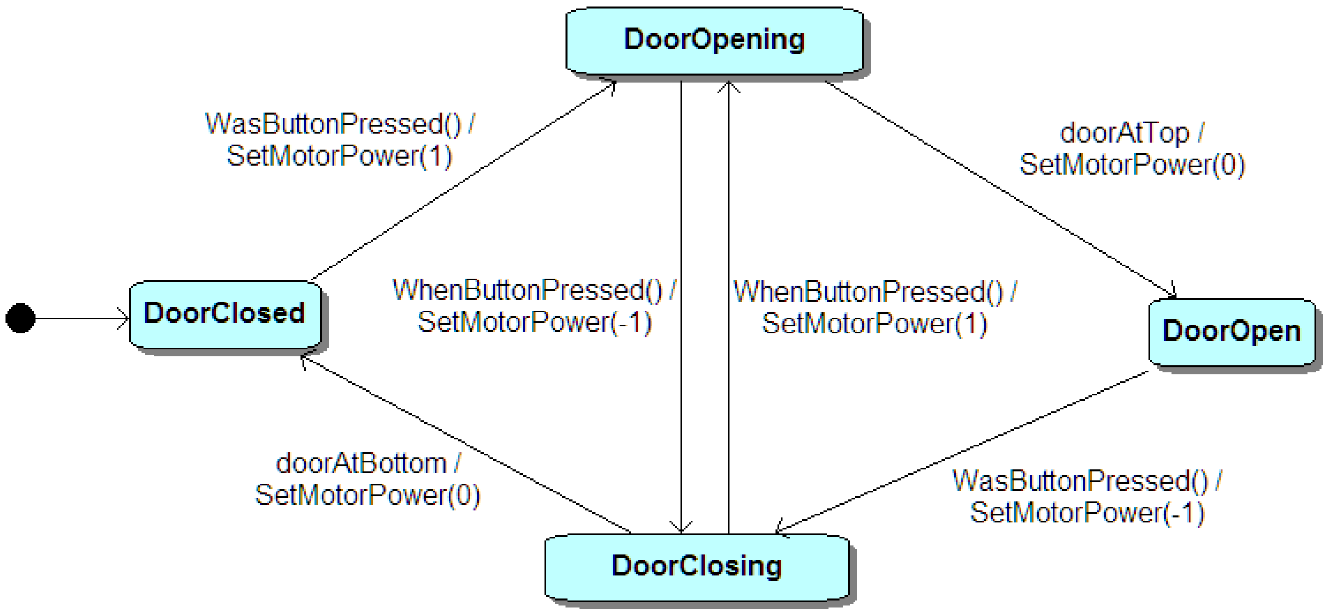
The four state functions look like this:

```
void StateDoorClosed(int *state)
{
    if(WasButtonPressed())
    {
        SetMotorPower(1);
        *state = DoorOpening;
    }
}


void StateDoorOpening(int *state)
{
    if(GetDoorPosition() >= DoorHeight - DoorTolerance)
    {
        SetMotorPower(0);
        *state = DoorOpen;
    }
}

void StateDoorOpen(int *state)
{
    if(WasButtonPressed())
    {
        SetMotorPower(-1);
        *state = DoorClosing;
    }
}

void StateDoorClosing(int *state)
{
    if(GetDoorPosition() < DoorTolerance)
    {
        SetMotorPower(0);
        *state = DoorClosed;
    }
}
```

The state diagram now looks like this:


DoorOpening
WasButtonPressed() / SetMotorPower(1)
doorAtTop / SetMotorPower(0)
DoorClosed
DoorOpen
doorAtBottom / SetMotorPower(0)
WasButtonPressed() / SetMotorPower(-1)
DoorClosing

In reality, most garage doors work more like this:


DoorOpening
DoorClosed
DoorOpen
DoorClosing
WasButtonPressed() / SetMotorPower(1)
doorAtTop / SetMotorPower(0)
WhenButtonPressed() / SetMotorPower(-1)
WhenButtonPressed() / SetMotorPower(1)
doorAtBottom / SetMotorPower(0)
WasButtonPressed() / SetMotorPower(-1)

The difference is simple: If you press the button when the door is opening, it changes to closing. If closing, it changes to opening. We now have two possible things that could happen when we are in the DoorOpening state. Right now the DoorOpening state function looks like this:

```
void StateDoorOpening(int *state)
{
    if(GetDoorPosition() >= DoorHeight - DoorTolerance)
    {
        SetMotorPower(0);
        *state = DoorOpen;
    }
}
```

We make one decision here as the code currently appears. Now we will make another decision. Add this code to this function to handle the other decision:

```
void StateDoorOpening(int *state)
{
    if(GetDoorPosition() >= DoorHeight - DoorTolerance)
    {
        SetMotorPower(0);
        *state = DoorOpen;
    }
    else if(WasButtonPressed())
    {
        SetMotorPower(-1);
        *state = DoorClosing;
    }
}
```

This is how we make two decisions.

Now I would like you to implement the case of the button pressed while the door is closing.

Final Tasks

Some final tasks to do and this program will be done. Please read the tasks first before proceeding.

1. There is a function called IsBeamBroken. You can call it like this:

```
    if(IsBeamBroken())
    {

    }
```

If this function returns true, the infrared beam has been broken. If you click near the bottom of the door, you will see a cat appear. We don't want to crush the cat, so change your program in these two ways:

A. If the door is closing and the beam is broken, reverse the motor to open the door instead.
B. If the door is open, the beam is broken, and the button is pressed, ignore the button.

If you double-click where the cat appears, it will stay there. Double-click on the cat again to make it go away.

2. I am concerned that I may forget to close my garage door. Add a feature to your program that will automatically close the door after 10 seconds, provided the beam is not broken, of course.

To solve this part, you will need to use a timer. I have provided these two functions:

ResetTimer() : The function resets a timer you can use. Think of this as pressing the start button on a stopwatch. It starts timing at time zero. An example of how to use this function:

```
    /* Reset the timer */
    ResetTimer();
```

GetTimer() : This function returns the number of seconds since ResetTimer() was called. This function reads the stopwatch.

```
    if(GetTimer() > 10)
    {
        /* 10 seconds have elapsed */
    }
```

Turn in garage.c via http://secure.cse.msu.edu/handin/

CSE 251