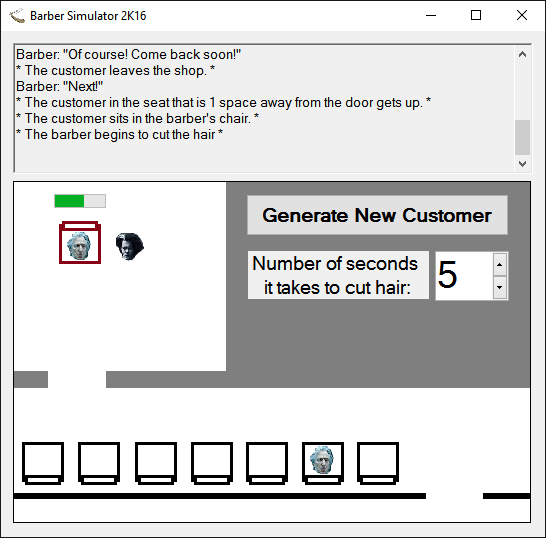
Barber Simulator 2k16 Documentation

# Program Design

This application simulates that of a barber cutting hair inside of a barber shop. The basic logic of the application is as follows. A barber is in a shop with a barber’s chair and 7 other waiting chairs. If there is no one in the store, the barber sleeps as they wait for a customer. If a customer enters the barber begins to work right away. If the barber is busy the customer will sit in the waiting char. If all the chairs are full the customer will just leave. The barber will then call the next customer to be server from the waiting chairs. This is determined based on the order they enter, not the order they are sitting in. If there are no more customers the barber will go back to sleep.

This application uses a GUI to represent and simulate this. The Image bellow outlines the different parts of the GUI and how they are intended to be used. This GUI was created using Microsoft Visual Studios standard windows forms application C# assembly.

This shows the history of what is happening from the bottom up.

Click this for a new customer to enter.

Controls the number of seconds a haircut takes. The range is from 1 to 300 seconds.

The Barber’s Chair will be occupied by the current customer or the barber if now customers are present

A progress bar showing the status of the current haircut

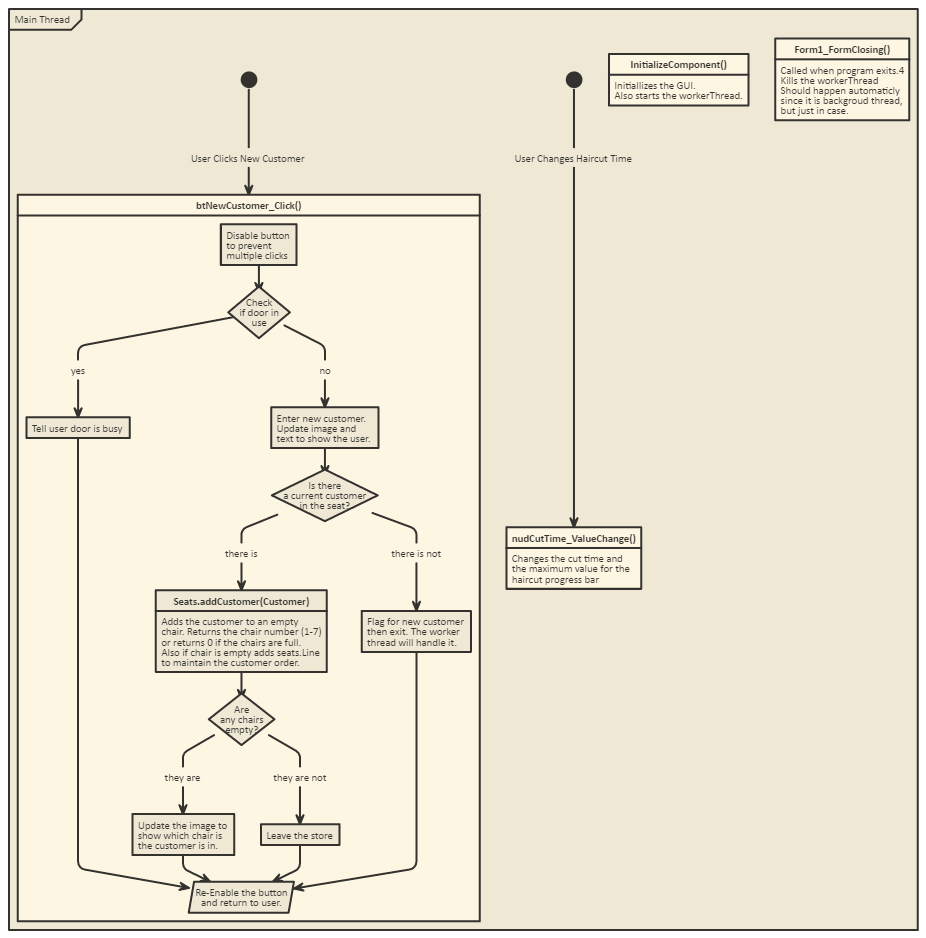
This is a Customer.

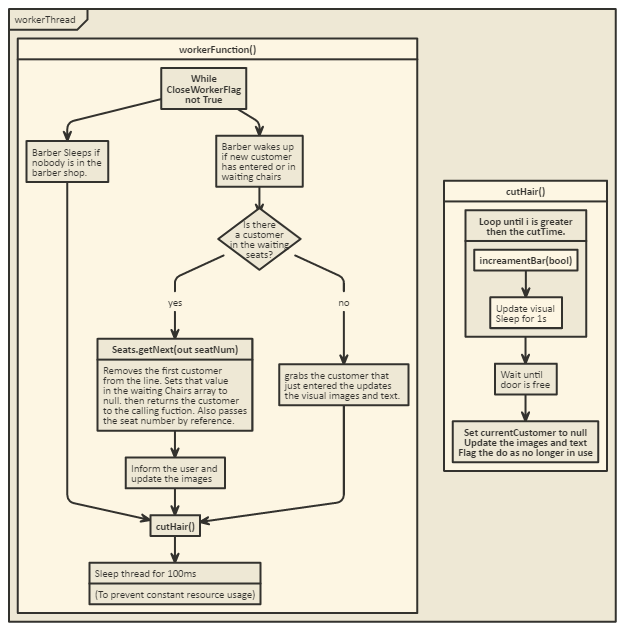
This is the entrance. It will only allow for one person leaving or entering at a time.

This is the barber.

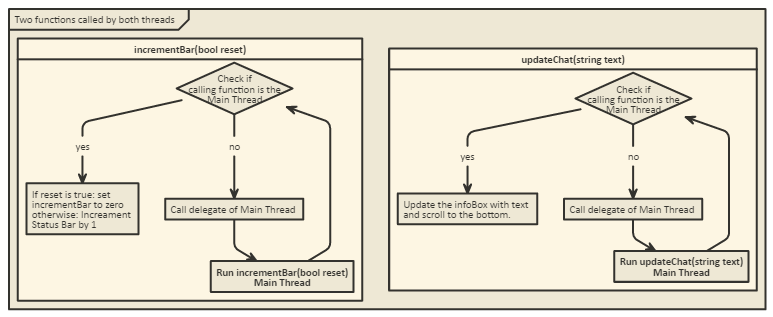
These are the waiting chairs. There are 7.

This isn’t too difficult to implement. It uses 2 threads to accomplish all of this. One is the main thread that the user interacts with and generates more customers. The next is the worker thread, which could be seen as the barber thread. It does all off the actually work and customer handling once they are waiting in the store.

This program really only uses 3 custom classes: a Customer class, a Seats class, and the windows form class. The windows form class is divided across two files: form1.cs and form1.designer.cs. The designer file just contains creation of all the visual objects and they respective locations. Form1.cs contains the actual functions for the application to work as well as the other two classes. Due to lack of relevance, complexity and the large number of visual objects in the GUI, this documentation will not contain a flowchart for any GUI objects or the form itself. For documentation on those classes please see Microsoft’s [documentation](https://msdn.microsoft.com/en-us/library/w0x726c2(v=vs.110).aspx) on it. Instead this documentation will focus on the 2 separate threads and outline them in the flow chart. Please note that everything with a black circle connected means that it is driven by an interaction from the user.

The main thread waits for the user to interact with it. It waits for an event to be trigger from some sort of button click. It does prep for the customers but never actually pulls them to the barber’s chair. That’s the workerThread’s purpose.

This is the worker thread. It is always run in a continuous loop checking for any updates. If the new customer enters or there are customers in the chairs it will wake up the barber (Unless the barber was already awake) and start to work on the customers. If there are no customers, the barber will sleep(if he wasn’t already sleeping).

Normally when uses objects and editing them across multiple threads, one uses the volatile keyword during the objects declaration. Although in the case of windows forms, this will not work. Windows forms objects need to be modified by the same thread that they were created by, which is the Main Thread in this case. To have the workerThread still be able to update the status bar and the infoBox it must call the Main Thread to do this for it. This is done by checking if the object requires to be invoke by the calling function. If it does we use a delegate to recall this same function. This delegate runs in the Main Thread it was created in. It then loops back through and sees the invoke status is no longer required and continuous with the update.

Also the increment bar function is never called by the main thread directly just by the delegate. It could be though. The update chat is directly called by them main during the new customer event.

# Files/Database Section

There are a few images linked in the project folder to this project. If one just runs this application from the .exe the root of the project folder, linking the images from a file location should not be needed. This is because the images are stored as a resource to the program. When compiled this cause the images used to be compile in the .exe file itself.

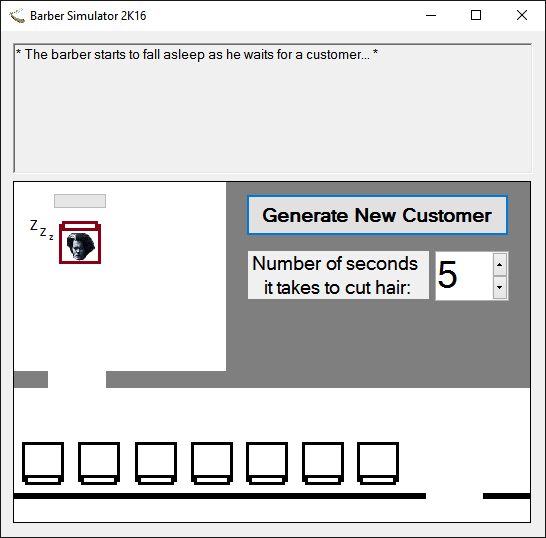
# Compile/Execution Section

This program was created in C# using .net framework 4.5.2. You will need the .net framework installed on your system to run it. The .exe needed to run the program is in the root of the project directory. Also the entire solution can be open in visual studio. (May need to add images back to resource file. I tested on 2 different computers and neither needed to do this but it is hard to tell with the range of available systems if the behavior will be the same).

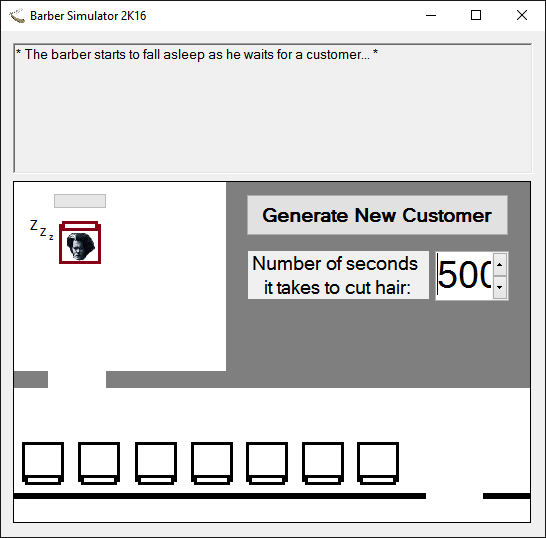
# Test Cases Section

Since this is a GUI Application that uses a button and a numeric up and down box, there isn’t really any invalid data that can be sent to the application. The numeric box will accept any value from 1 to 300 and if the value entered is less than 1 or greater than 300 it will round itself back up to 1 or down to 300 respectively

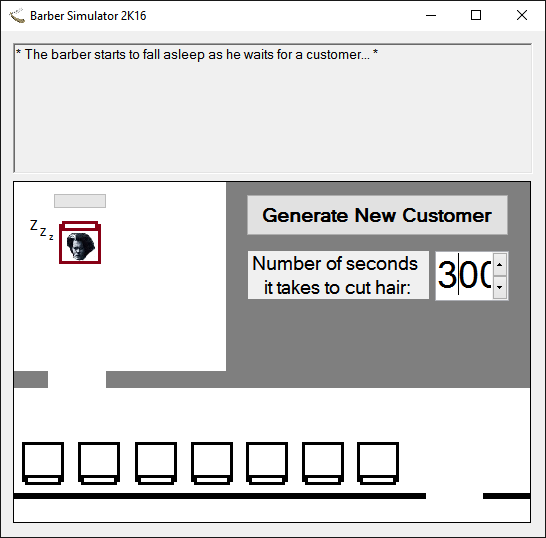
Now the following is a demonstration of all the operations working as intend. First Lets start the application.



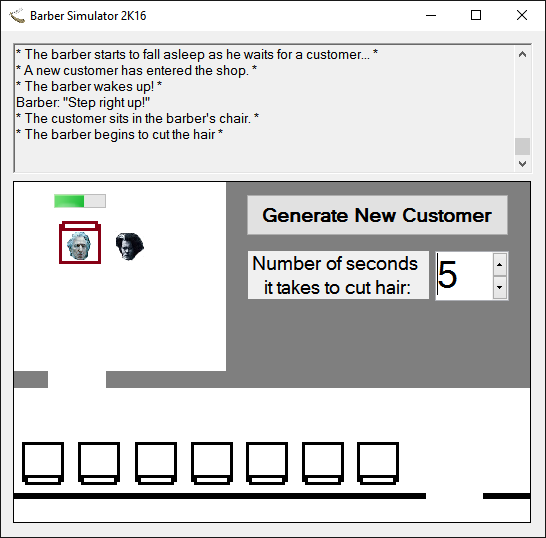
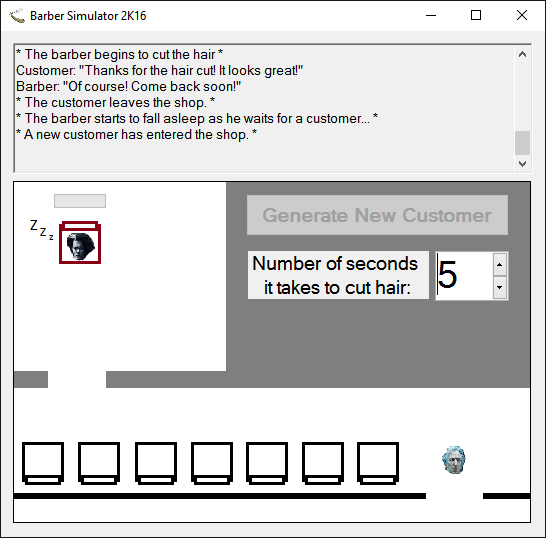
Notice that we have a message up top saying that “the barber starts to fall asleep as he waits for a customer”. This will happen any time there is an update. Also there’s the number box for the length of time for a haircut. Let’s put in a value that is greater than the maximum 300: 500.



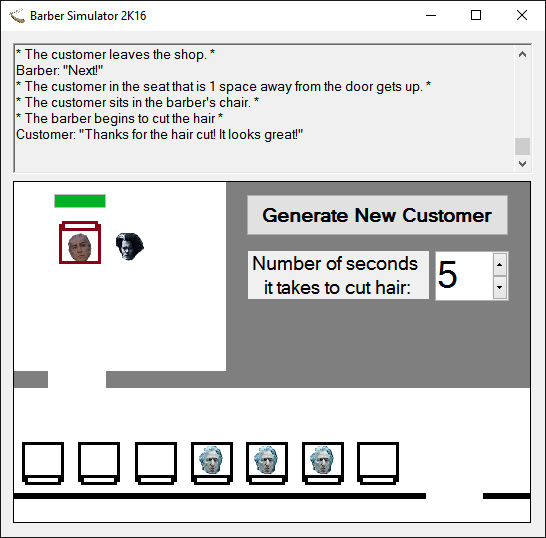
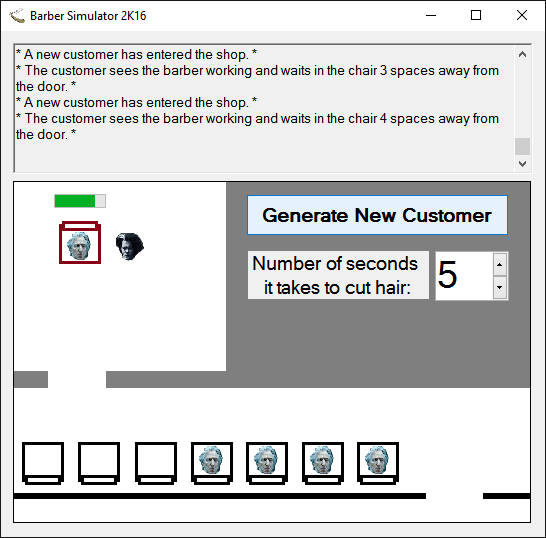
Hit enter or press on anything else…



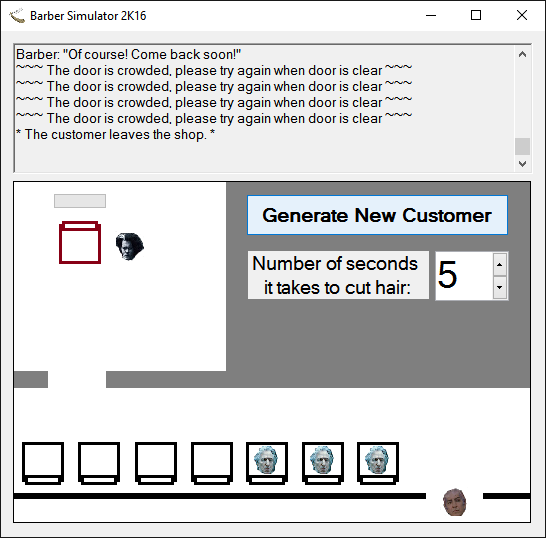
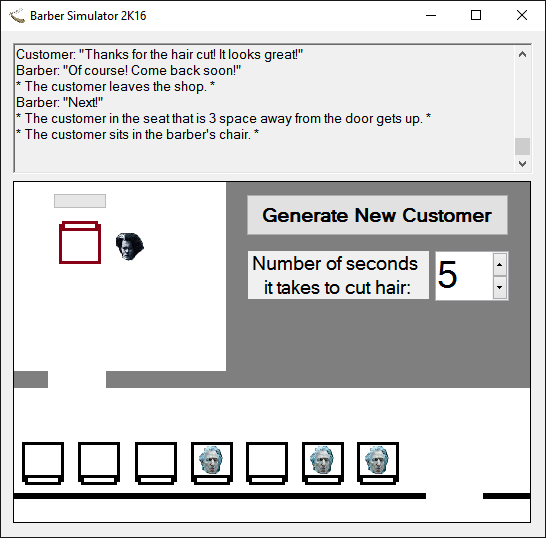
It goes back down to 300. This can be done at any time and the barber’s progress will not reset but just adjust to the change. Now let’s set that back to a reasonable time like 5 seconds, and click the generate customer button.



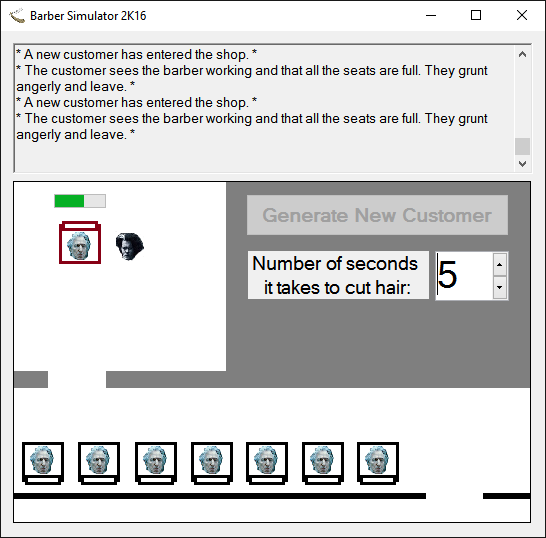
What we will see is that the customer walks in from the entrance and immediately goes to the barber for a haircut. The status bar then start to update in increments that will total to the number of seconds currently listed. Ok time for more customers. Hit the button 5 times. It will have you wait until the current customer leaves the doorway.



The wait seats start to fill in and the customers are then worked on as the next finishes. When 2 chairs are empty lets add 2 more.



What one will see now is that the older customers, in terms of when they walked in, are handled first still. Also try entering a new customer as on leaves. It won’t be possible as the door is full. The leaving customer must fully exit before a new one can come in. Next try filling the room and seeing what happens to a customer when the room is full.



The customer will see that all the seats are full and just leave. You will need to wait tell a seat opens up before then next customer can enter. That is really all the possibilities of this application.