Visualizing Graph Algorithms Developer Guide

Welcome:

I would first like to thank you for taking an interest in my project, and for having the interest to expand it. I hope this developer guide helps you better understand my code and makes it easier to add on or enhance it.

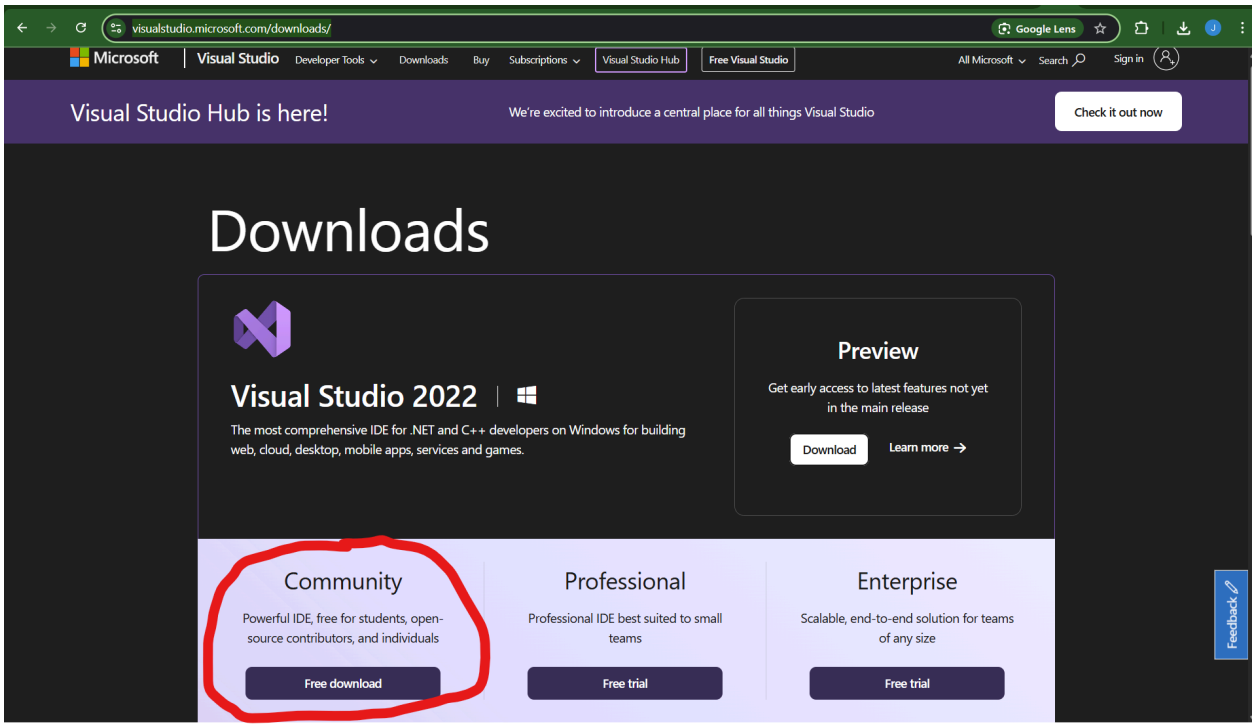
Note: For some of you, a lot of the below information may not be as helpful as it would be for others. However, I do ask everyone to read the section: **Get Up To Date Program** found on page 7. If you don't read this section, you may run into an issue with the file. Also if you already have Visual Studio 2022, make sure you have the .NET Desktop Development workload extension downloaded for it. Hope you enjoy my program.

What You Need:

- Computer or Laptop
- Visual Studio 2022 *(older versions may work, but unsure)*
- Updated project zip file (instructions below).
- Understanding of what a graph is (Please see user manual if you don't know)

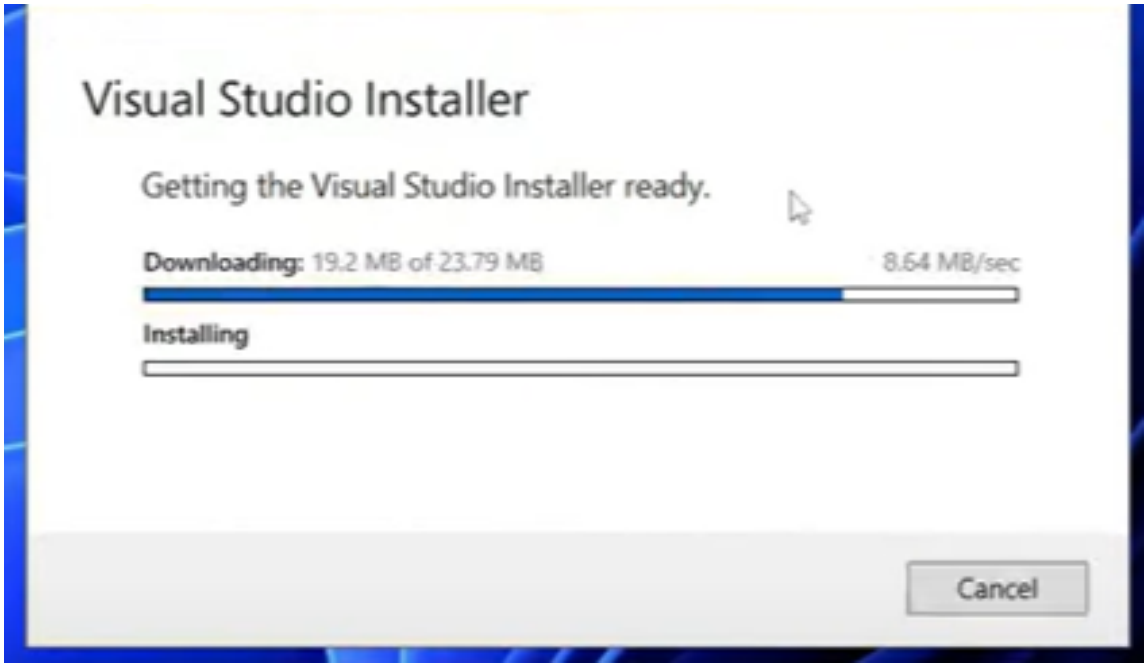
If You Don't Have Visual Studio:

- To reiterate, you will need a computer with Visual Studio installed. (I used version 2022, you may be able to use 2019 or 2017, it may prompt you to update the file, which it will handle)
- To download Visual Studio 2022, you can search for Visual Studio 2022 download, and it should be the first link. Or use this URL: https://visualstudio.microsoft.com/downloads/
- Next, you want to download the Community Free Version shown below:

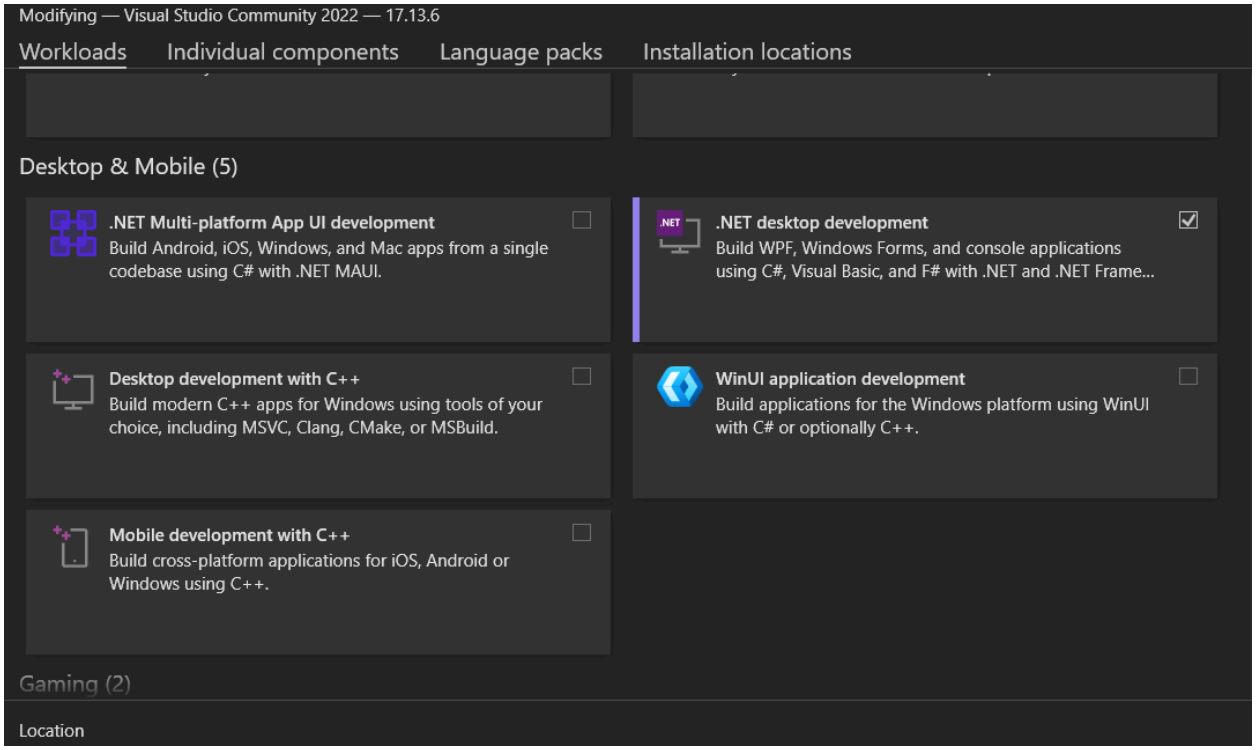


- You can download it anywhere on your computer; it doesn't really matter.
- Once downloaded, you need to open the file. You may be prompted with, Would you like Visual Studios to make changes, Click allow.
    - Clicking "Allow" is safe, and this doesn't give Visual Studio control of your system. It will simply allow Visual Studio to use the resources it needs to execute algorithms and animations.
    - Without this permission, the application won't be able to run.
- Next, we need to follow the prompts as needed to set up. The first is shown below, and you want to click **Continue**.
    - Please note this could take several minutes, maybe even longer depending on network speed to download.

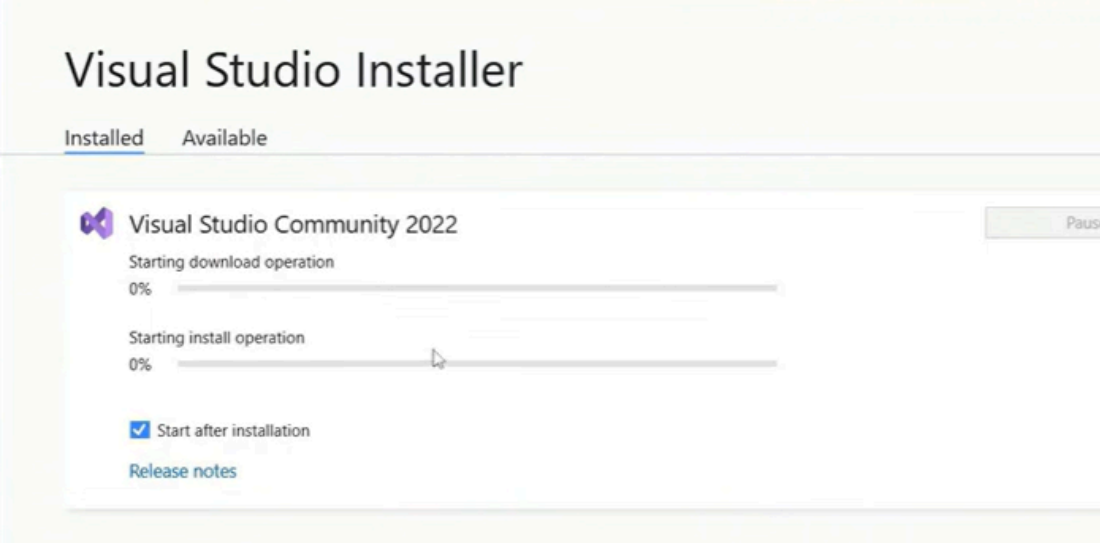
- After that, you should see the screen below:



- The screen below will then show up. You need to make sure you select **.NET desktop development** as selected and shown below.



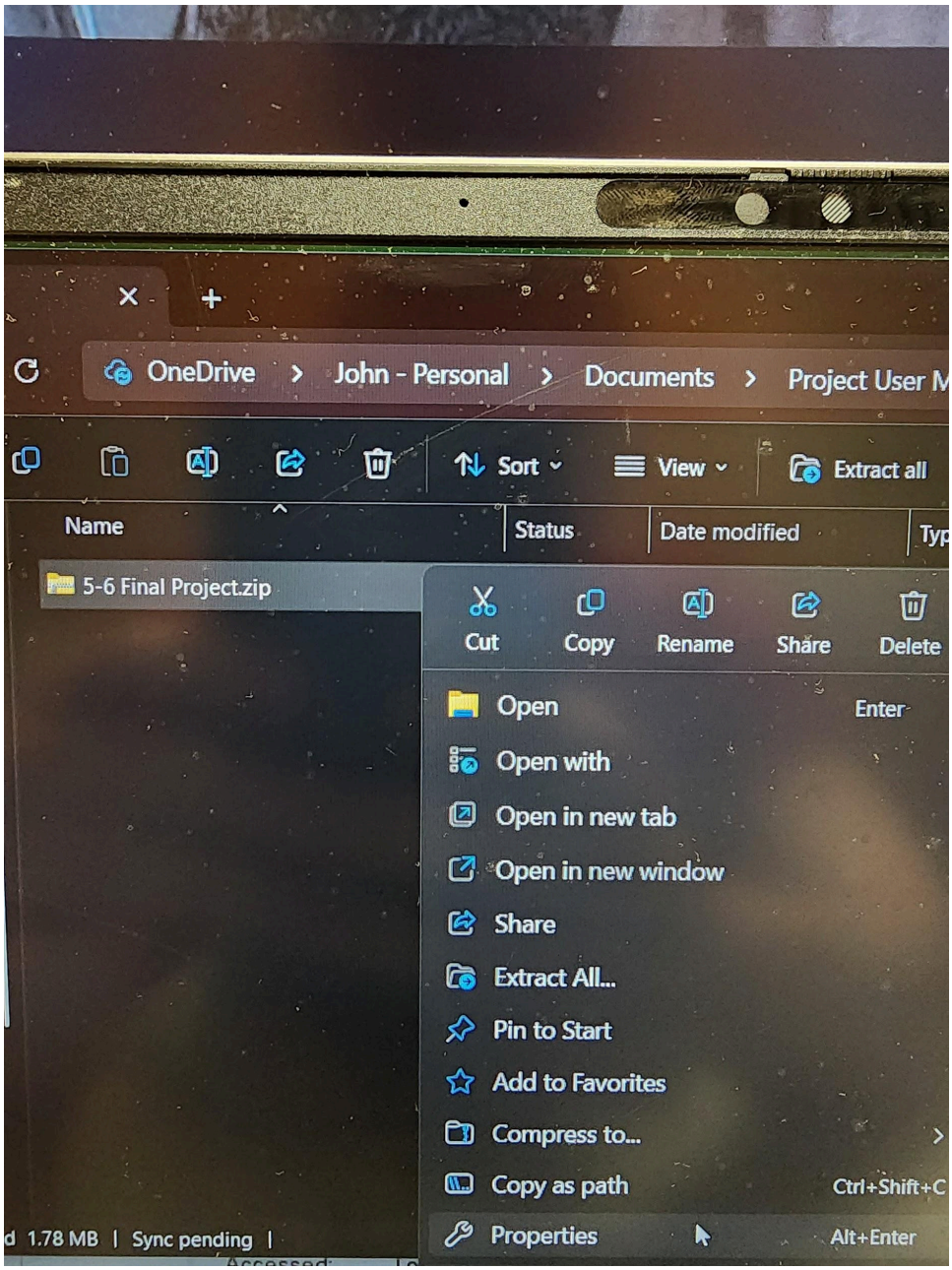
- Once you do that, you want to click install in the bottom right corner. This should then bring you to the screen below:



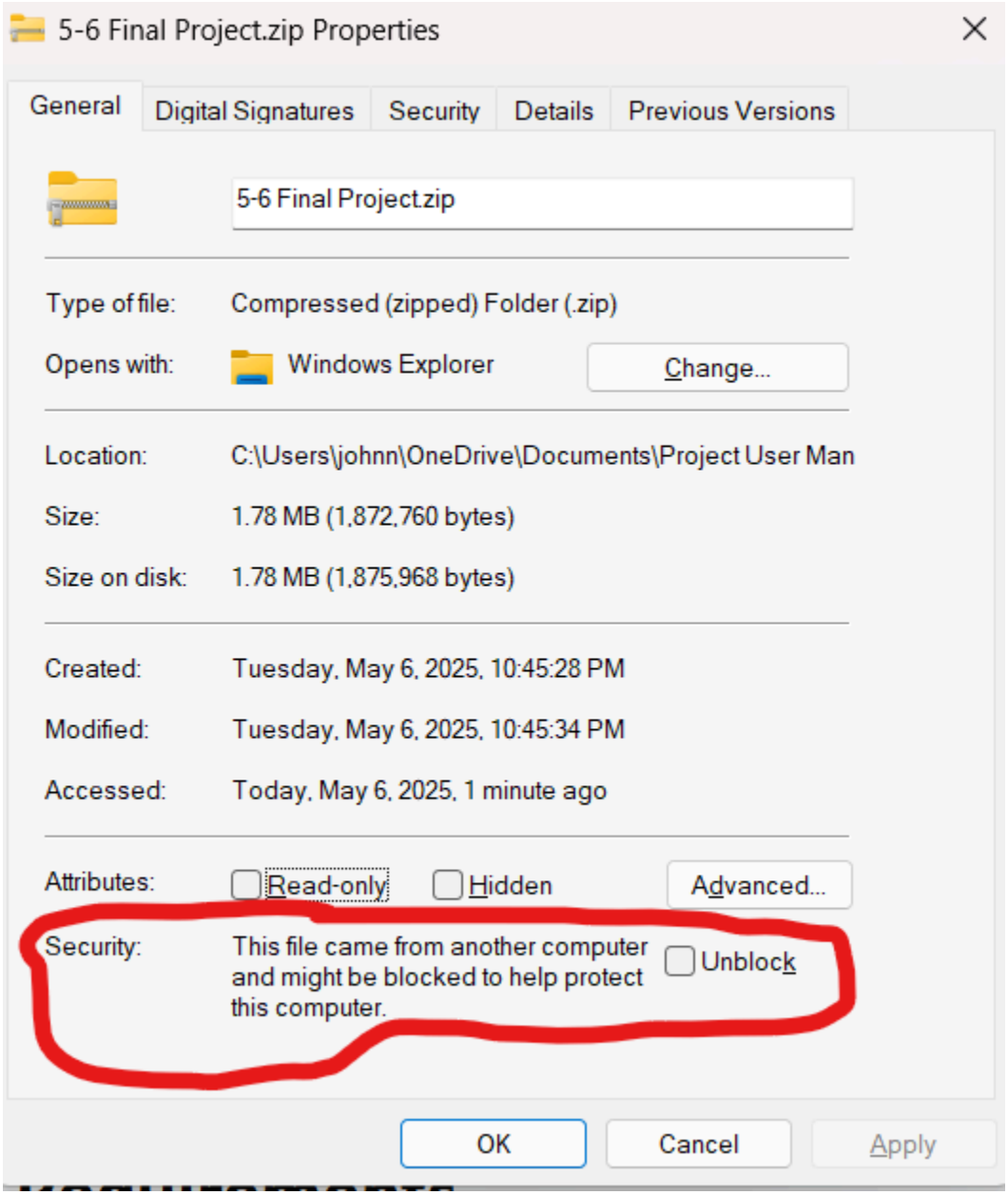
- Once you are done installing, you should have the following screen. **NOTE: Nothing more needs to be done after this step in regards to downloading Visual Studio; the rest is optional for now.** From here, you can choose to launch, log in, and create an account, or skip. You can also choose what theme you'd like, such as light or dark mode (Note you may need to do this later on as well, but I am not 100% sure).

Get Up-To-Date Program File:

- Now you will need the most recent version of my project, which can be downloaded from my website under the project section, and my last blog post as well. Here is the URL: https://compsci04.snc.edu/cs460/2025/johnolson/website/project.html
- Then you want to click the most recent uploaded file (which will be shown) and click the download button. You may put this file anywhere. **NOTE: DO NOT Extract the file yet, just download and follow further steps.**
- Once downloaded, you will need to **right-click** on the file, navigate to **properties** as shown below, or click on the file and hit **ALT+Enter**:



- Then you will be brought to a similar screen shown below. If you look down at the bottom under securities (shown in the red circle), if you have a similar message as I do below, you're going to want to **select Unblock**, then **Ok or Apply.**

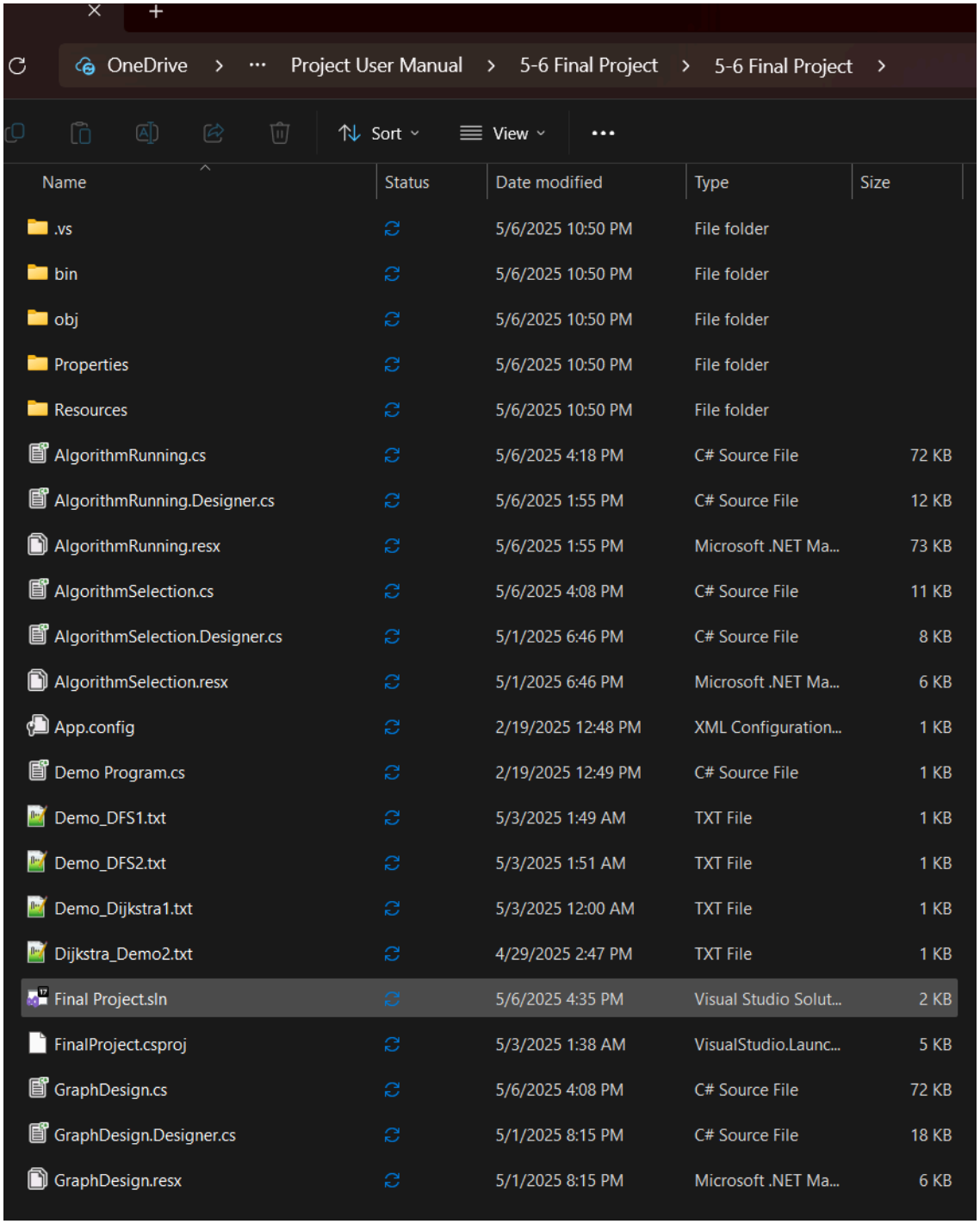


- Now you can unzip or extract the file. If you click on the file, it should say in the bar to extract all, or when you right-clicked on it earlier, it was right under the share.
- Now you are ready for the next step

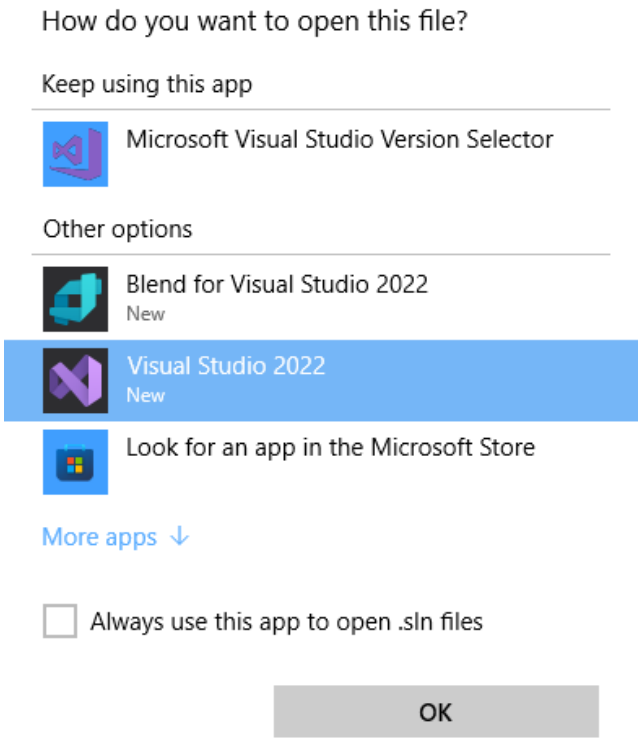
Running / Launching a program

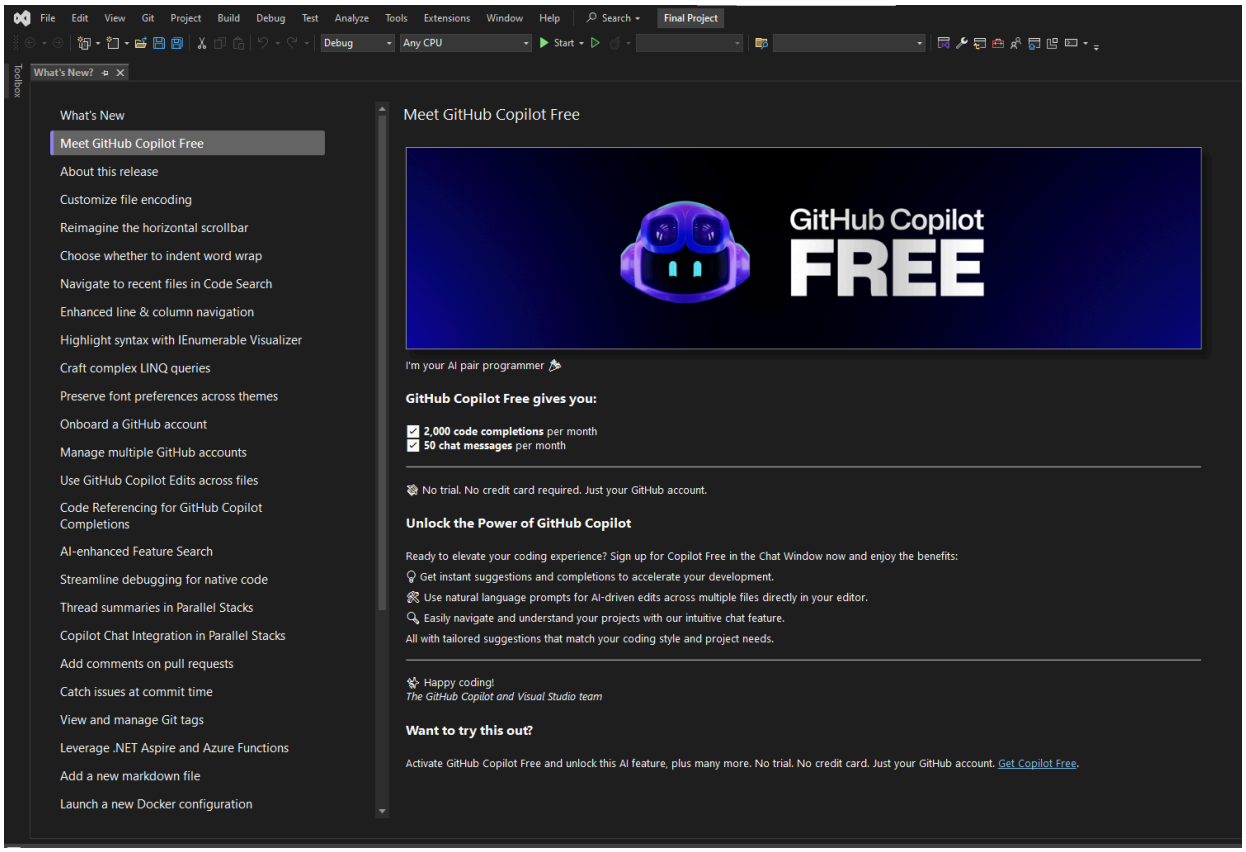
- **DISCLAIMER:** The design, graph form, flickers, or may look glitchy. This is because it is redrawing everything to the screen after any change. (I apologize, though if I had more time, I would have a more efficient and less flickering method.)

- Now you want to navigate to where you saved and extracted your file and open it. You should see a screen similar to what is below:

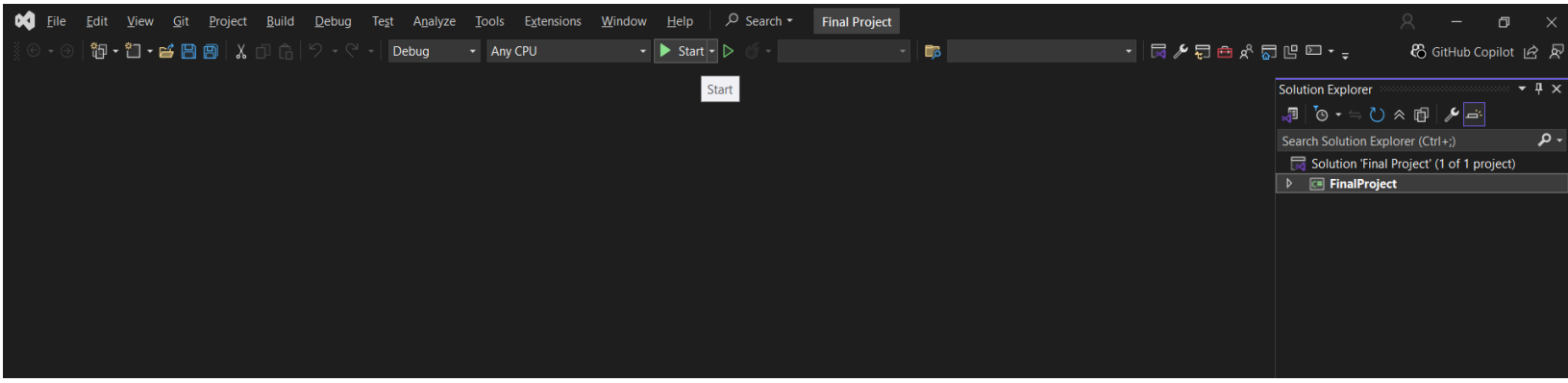


- Now you need to open up the Final Project.sln (.sin may not display; it will be Microsoft Visual Studio Solution (under Type)), which is highlighted on the above screen.
- Below screen may appear. Please select Visual Studio 2022

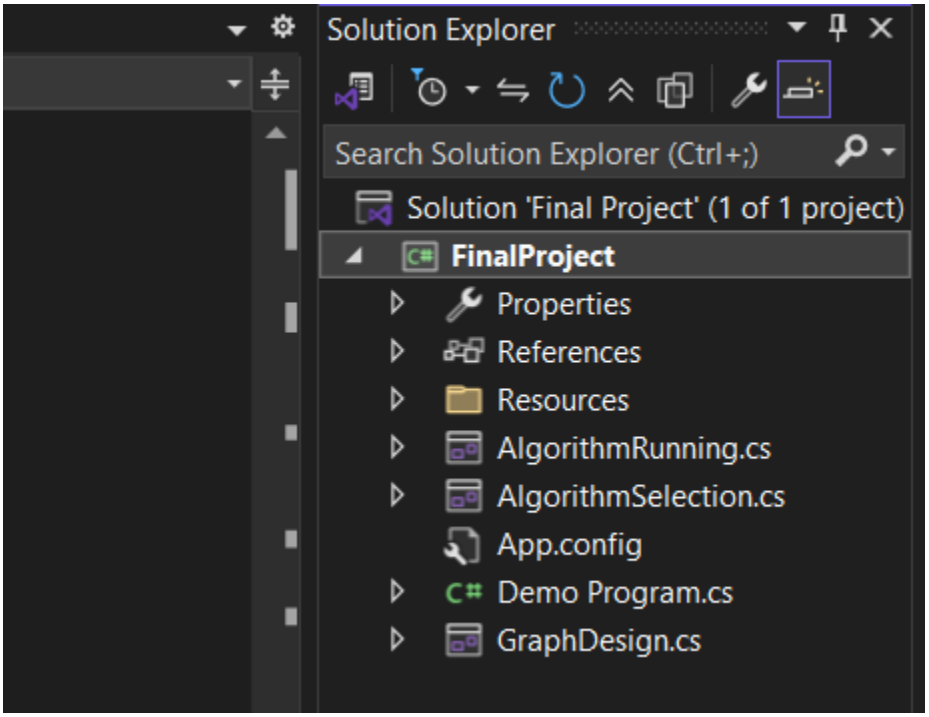

How do you want to open this file?
Keep using this app
Microsoft Visual Studio Version Selector
Other options
Blend for Visual Studio 2022
New
Visual Studio 2022
New
Look for an app in the Microsoft Store
More apps
Always use this app to open .sln files
OK


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Final Project
Debug
Any CPU
Start
What's New?
What's New
Meet GitHub Copilot Free
About this release
Customize file encoding
Reimagine the horizontal scrollbar
Choose whether to indent word wrap
Navigate to recent files in Code Search
Enhanced line & column navigation
Highlight syntax with IEnumerable Visualizer
Craft complex LINQ queries
Preserve font preferences across themes
Onboard a GitHub account
Manage multiple GitHub accounts
Use GitHub Copilot Edits across files
Code Referencing for GitHub Copilot Completions
AI-enhanced Feature Search
Streamline debugging for native code
Thread summaries in Parallel Stacks
Copilot Chat Integration in Parallel Stacks
Add comments on pull requests
Catch issues at commit time
View and manage Git tags
Leverage .NET Aspire and Azure Functions
Add a new markdown file
Launch a new Docker configuration
Meet GitHub Copilot Free
GitHub Copilot
FREE
I'm your AI pair programmer
GitHub Copilot Free gives you:
2,000 code completions per month
50 chat messages per month
No trial. No credit card required. Just your GitHub account.
Unlock the Power of GitHub Copilot
Ready to elevate your coding experience? Sign up for Copilot Free in the Chat Window now and enjoy the benefits:
Get instant suggestions and completions to accelerate your development.
Use natural language prompts for AI-driven edits across multiple files directly in your editor.
Easily navigate and understand your projects with our intuitive chat feature.
All with tailored suggestions that match your coding style and project needs.
Happy coding!
The GitHub Copilot and Visual Studio team
Want to try this out?
Activate GitHub Copilot Free and unlock this AI feature, plus many more. No trial. No credit card. Just your GitHub account. Get Copilot Free.

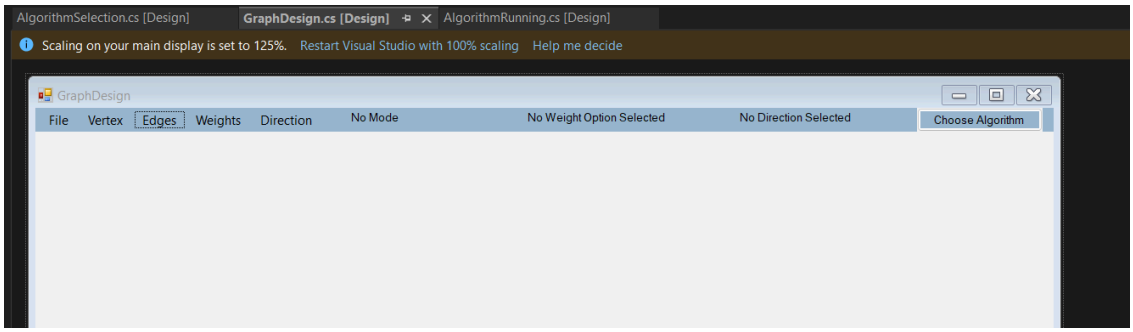
- You should come to a screen similar shown either above or below. On the far right-hand side of the screen, you should see the FinalProject section under the Solution Explorer.



- Once you do that you should be met with the below screen. The only files relevant are the AlgorithmRunning.cs, AlgorithmSelection.cs, and GraphDesign.cs.



- If you double-click on them, you should see the design screens pop up. Similar to the image below. From there, click control + alt + 0 to open the code window. Now you can begin!!!



Variable References:

- Below are the important variables you need to know.

```
/************** Vertex State Representaion ****************/
private const int UNVISITED = 0; // These are how verticies start off
private const int VISITED = 1; // These are how some verticies progress from UNVISITED to DONE like in DFS
private const int DONE = 2; // This is the completed state

/********************* Play / Pause and fastforward variables ****************/
private ManualResetEvent pauseEvent = new ManualResetEvent(true);  // Controls pause/play state
private bool isPaused = false;  // Tracks if algorithum is paused
private bool fastForward = false; // Track if Fast Forward was clicked
// This tracks the current sleep time for when we are paused and trying to fast forward,
int tempSleeptime; // that way we still are using the same value when we are done.
private int sleepTime = 5000; // Default value is 5000, but this controls how many seconds the algorithum delays / sleeps

/***************** Graph Variables *********************************/

// These are variables needed for the graph from form 1
private int[,] adjacencyMatrix; // Adjacency matrix representing the graph
private int vertexCount; // Number of vertices
private int vertexWidth; // Width of each vertex (for visualization)
private List<Point> vertexPositions; // Positions of each vertex on the form
```

```
/************ Algorithum Specific Variables **************/
// This will determin what algorithum we are using
private int functionCall;

// State tracking arrays
private int[] vertexStatus; // Keeps track of visited nodes
private int[] parent; // Stores the parent of each vertex in
private int[] cost; // This stores the minimum path or cost of each vertex for Dijkstra's

private bool isDFSRunning = false; // Ensures DFS does not restart while running
private bool isDijkstasRunning = false; // Ensures Dijkstas does not restart while running
bool isDirected; // This will tell us if the graph is directed or not
private bool isWeighted = false;  // Change to true for weighted graph

// For tracking weights and label positions
List<PointF> edgeWeightPositions; // This stores the location of where the weight is supposed to be on the screen
List<string> edgeWeightText; // This stores the string value of what the user wanted their value to be
```

Graph Design Form Enhancements:

- I would really like to see the drawing to be enhanced in this form, as it is currently terribly inefficient. Right now this form uses the below painting method:

```
/********************* Painting Function  *********************************/

// OnPaint method is overridden to repaint the image each time the form is painted;
// otherwise the image would only persist until the next repainting
// Sources Used:
// https://stackoverflow.com/questions/952906/how-do-i-call-paint-event
// https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.control.onpaint?view=windowsdesktop-9.0
0 references
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    UpdateAdjacencyMatrix(); // This will updtate the matrix, if there was a vertex added it will expand it
    DrawEdges(e.Graphics); // We need to redraw edges, as there may have been one added, moved, or deleted
    DrawVertices(e.Graphics); // We need to redraw all verticies, as they may have been added, moved, or deleted

    // This will handle drawing the edge weight onto the screen by looping through all th edge weights stored
    for (int i = 0; i < edgeWeightPositions.Count; i++)
        e.Graphics.DrawString(edgeWeightText[i], font, Brushes.Black, edgeWeightPositions[i], vertexLabelFormat);
}
```

- This is how the painting is called. Anytime you add a vertex, edge, or delete / move something, it triggers a change and an Invalidate(). Everytime Invalidate() is called, and a change occurs, it calls this OnPaint function which redraws everything all at once, regardless if it was changed or not.
- I want this form to paint, like in the Algorithm Running form. Below I have 3 screen shots and I will do my best to explain this.
- So how I paint in form 3 is using what's called a bitmap. A bitmap you can think of as a canvas. You put what you want onto the canvas, then once you are ready you blast it to the screen. Therefore, the bit map only gets updated if there's a change, and it only adds what's changed.
- In order to use the bitmap you must do what is shown below. You need to use the lock variable, because using, is a protected action. If two processes try to use it at the same time, the program will break. To prevent this, you want to lock it. This ensures, only one process can use it at a time.

```
// This will lock the bitmap to ensure this is the only process trying to draw
lock (bitmapLock)
{
    // Seeing bitmaps aren't thread safe, and using is a protected process, we can't have 2 processes attempting
    //to modify this bitmap at the same time. So in order to eliminate potential race conditions, or crashes
    // we use the above lock bitmap
    using (Graphics g = Graphics.FromImage(graphBitmap))
    {
```

- Once you are in the using property, you just need to do g. and whatever you'd want to paint. Weather that is an ellipse or circle, or edge etc. The graphics g is mapped to the bitmap. So anything added into it will reflect into the bitmap.

```
// Rehighlight from vertex as visited in yellow but not done
g.FillEllipse(visitedVertex, vertexPositions[from].X - vertexWidth / 2, vertexPositions[from].Y - vertexWidth / 2, vertexWidth, vertexWidth);
// Highlight to vertex as visited in yellow but not done
g.FillEllipse(visitedVertex, (vertexPositions[to].X - vertexWidth / 2), (vertexPositions[to].Y - vertexWidth / 2), vertexWidth, vertexWidth);
```

- This is the OnPaint function used in the Running Algorithms form. I also lock this as you don't want two things interacting with the bitmap at the same time. But the bitmap would be null if there wasn't a change, thus it wouldn't blast it to the screen.

```
/************************     Paint Functoin      ****************************************/
0 references
protected override void OnPaint(PaintEventArgs e)
{
    // This will lock the bitmap to ensure this is the only process trying to draw
    lock (bitmapLock)
    {
        // If the graphBitmap has been drawn already, ie not null, then we want to display the bitmap
        // starting at 0,0 or the top left corner on the form.
        if (graphBitmap != null)
            e.Graphics.DrawImage(graphBitmap, 0, 0);
    }
}
```

- This shouldn't need to be done much. You can look at Algorithm Running form to really see how the painting works, but all you'd need to do is call the functions that now paint, then put it into the bitmap vs the screen graphics, then update the OnPaint function, to just have the screen graphics draw the bitmap image onto the screen.

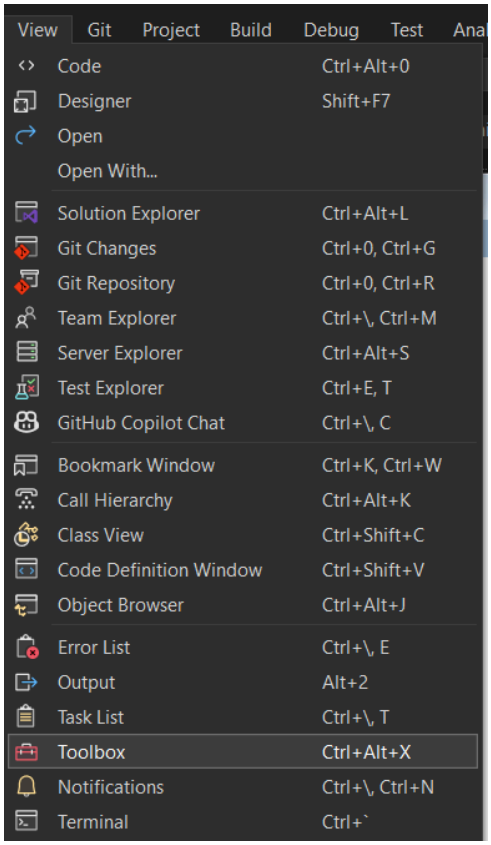
Adding An Algorithm:

- If you are looking to expand the program, by adding on an algorithm, I will detail the steps how to do so, to the best of my ability. You will need to make minor adjustments in the Algorithm Selection form, and Algorithm Running, is where the new algorithm will go.

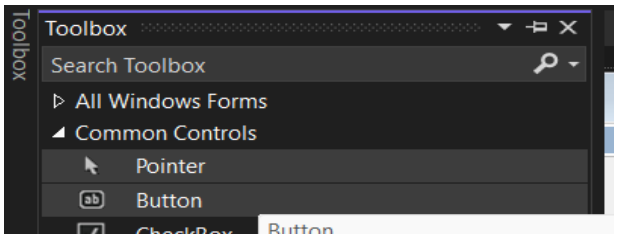
Algorithm Selection Form:

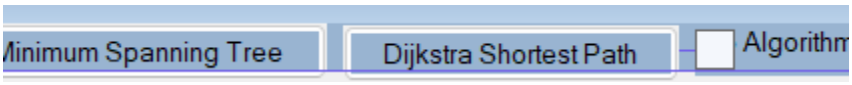
- You first need to add a button into the AlgorithmSelection.cs [Design] form. To do so, you need to make sure you are on the form design page shown below:

- Then you want to click view in the top left of the screen, and click on Toolbox shown below or do control + alt + x.

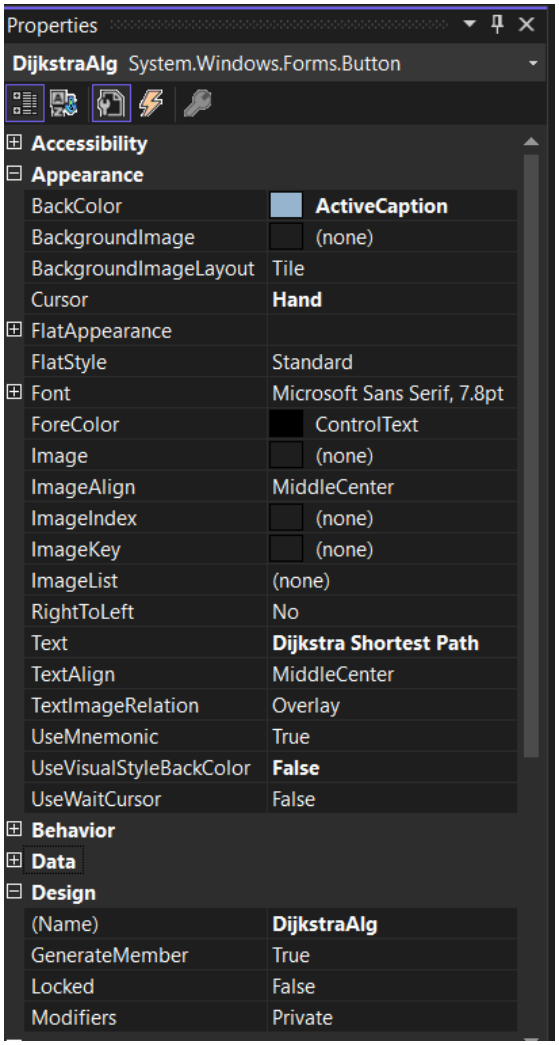


- Then you want to drag and drop a button from the Toolbox (shown below) and drag it to the menu bar / strip. (This is the blue line at the top of the form, where all the other buttons are.

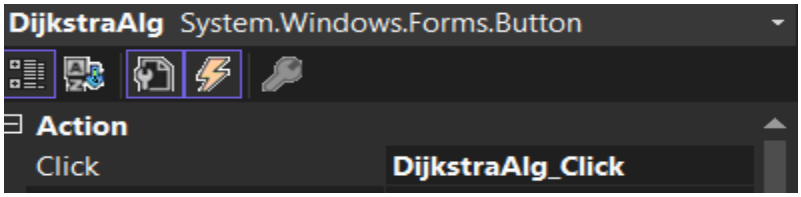




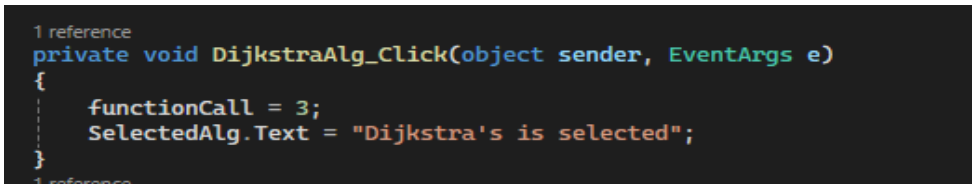
- Once you drop the button into the menu bar, then you might want to rearrange things on the bar to make it more visible. After that you want to click on the button, then on the bottom right of the screen there should be a properties window (shown below). You want to make sure you set the BackColor and Cursor to be the same as below. You also want to change the text to be the name of the algorithm, and then under Design Name, you should make this a useful name as this is the id for the button.



- Once you have that done, you want to click on the lightning bolt, which will bring you to the events tab shown below. You want to scroll down to where you find Action: Click. Then you want to click once in the blank space to the right of it. Then you want to name the button click function. Once you type out the name, click enter and it will generate the function for you.

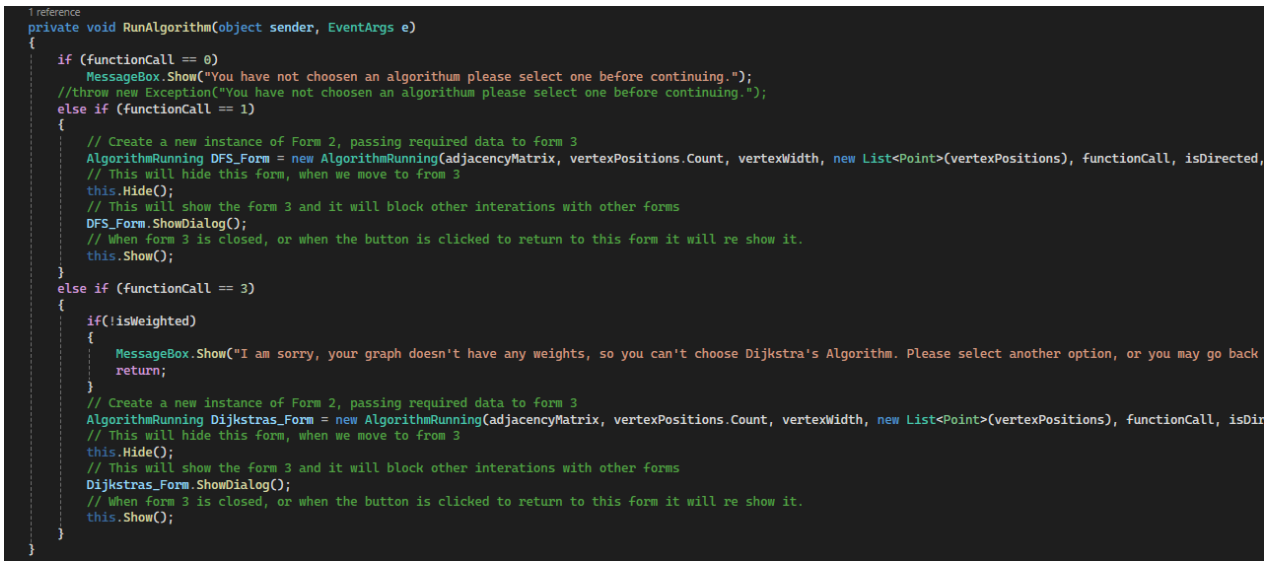


- Then you want to copy the below function, but obviously change it for your needs. The function call is a unique id integer for the function. DFS is 1, Minimum Spanning Tree is 2, and Dijkstra's is 3. You must set functionCall to an integer that isn't 1,2 or 3. From there you want to change the text from saying "Dijkstra's is selected" (in the below screenshot), to _____ algorithm is selected, so that the user knows which algorithm is being selected.



```
1 reference
private void DijkstraAlg_Click(object sender, EventArgs e)
{
    functionCall = 3;
    SelectedAlg.Text = "Dijkstra's is selected";
}
```

- After that, you need to modify the RunAlgorithm function shown below. I recommend you copy the else if (functionCall == 3) especially if the function requires it to be weighted. However, I would recommend altering this function even more. I would just do an if(functionCall ==0), then else be what is inside the else if (functionCall == 1). I would then check the requirements, like if it needs to be weighted in the on button click functions. Not in the Run Algorithm function. This would clean up this function.



```
1 reference
private void RunAlgorithm(object sender, EventArgs e)
{
    if (functionCall == 0)
        MessageBox.Show("You have not choosen an algorithum please select one before continuing.");
    //throw new Exception("You have not choosen an algorithum please select one before continuing.");
    else if (functionCall == 1)
    {
        // Create a new instance of Form 2, passing required data to form 3
        AlgorithmRunning DFS_Form = new AlgorithmRunning(adjacencyMatrix, vertexPositions.Count, vertexWidth, new List<Point>(vertexPositions), functionCall, isDirected,
        // This will hide this form, when we move to from 3
        this.Hide();
        // This will show the form 3 and it will block other interations with other forms
        DFS_Form.ShowDialog();
        // When form 3 is closed, or when the button is clicked to return to this form it will re show it.
        this.Show();
    }
    else if (functionCall == 3)
    {
        if(!isWeighted)
        {
            MessageBox.Show("I am sorry, your graph doesn't have any weights, so you can't choose Dijkstra's Algorithm. Please select another option, or you may go back
            return;
        }
        // Create a new instance of Form 2, passing required data to form 3
        AlgorithmRunning Dijkstras_Form = new AlgorithmRunning(adjacencyMatrix, vertexPositions.Count, vertexWidth, new List<Point>(vertexPositions), functionCall, isDir
        // This will hide this form, when we move to from 3
        this.Hide();
        // This will show the form 3 and it will block other interations with other forms
        Dijkstras_Form.ShowDialog();
        // When form 3 is closed, or when the button is clicked to return to this form it will re show it.
        this.Show();
    }
}
```

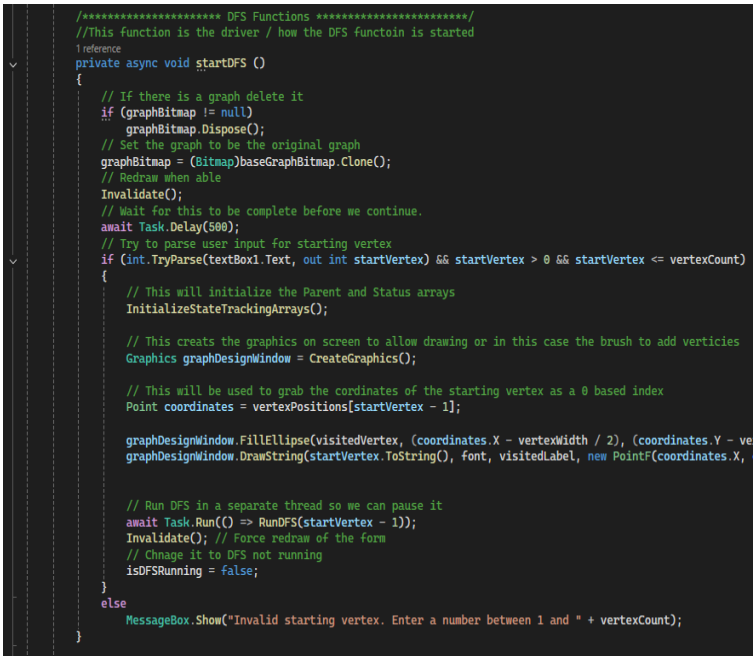
- After you add the button, create an on click function for that button, and set the functionCall variable to having its own number, like 4 for example, and you have altered the RunAlgorithm function, you are now ready to write the algorithm.

Algorithm Running Form:

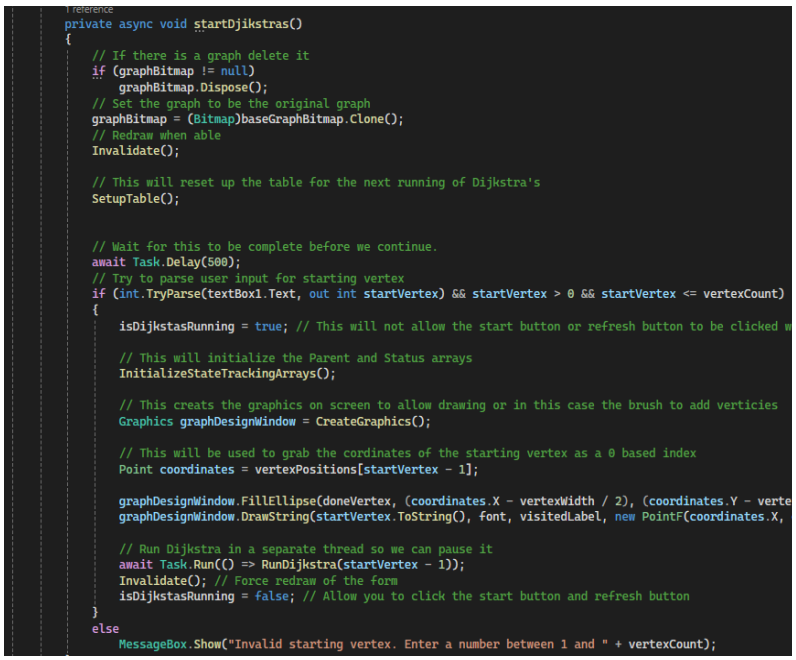
- Now it is time to start writing your desired algorithm. You can write it anywhere you want, but I tried my best to organize my code, so it is easier to find. I recommend either adding the new algorithm functions at the very bottom, or just after the Dijkstra's functions and before the drawing / animation functions.
- You must first create a global book for your algorithm, on line 88, like I have for Dijkstras and DFS. It should be defaulted to false.
- Then you **must** adjust the StartBTN_Click function found on line 249, and shown below.

```
// This is for the start button and will call the appropriate function
1 reference
private void StartBTN_Click(object sender, EventArgs e)
{
    //If function call was 1 then it means we are running DFS algorithm
    if (functionCall == 1)
    {
        if (!isDFSRunning)
            startDFS();
        else
            MessageBox.Show("Sorry can't restart, DFS is runnning. Please wait untill after the algorith, has finsihed.");
    }
    // If function call was 2 then it means we are running Dijkstra's
    else if (functionCall == 3)
    {
        if (!isDijkstasRunning)
            startDjikstras();
        else
            MessageBox.Show("Sorry can't restart, Dijkstra's is runnning. Please wait untill after the algorith, has finsihed.");

    }
}
```

- I recommend copying the else if statement, and changing the inner if statement, to be !____ where blank is the bool name you gave. Then you need to change the message show box to indicate the appropriate algorithm. Lastly you need to name the starting function. It should start____ where blank is the algorithm name.
- Below are the two other starting functions. I will tell you the bare minimum you will need is the startDFS function. It must be an async void function, and you must have the algorithm call as await Task.Run(() => Run_____startVertex - 1)); where the blank is the algorithm. This way we keep consistency. Start Dijkstra's can be found on line 475, and startDFS can be found on line 395. The only difference between the two is that Dijkstra needed to have the table and feedback box. I will discuss this portion in a moment.



```
/********************* DFS Functions ***********************/
//This function is the driver / how the DFS functoin is started
1 reference
private async void startDFS ()
{
    // If there is a graph delete it
    if (graphBitmap != null)
        graphBitmap.Dispose();
    // Set the graph to be the original graph
    graphBitmap = (Bitmap)baseGraphBitmap.Clone();
    // Redraw when able
    Invalidate();
    // Wait for this to be complete before we continue.
    await Task.Delay(500);
    // Try to parse user input for starting vertex
    if (int.TryParse(textBox1.Text, out int startVertex) && startVertex > 0 && startVertex <= vertexCount)
    {
        // This will initialize the Parent and Status arrays
        InitializeStateTrackingArrays();

        // This creats the graphics on screen to allow drawing or in this case the brush to add verticies
        Graphics graphDesignWindow = CreateGraphics();

        // This will be used to grab the cordinates of the starting vertex as a 0 based index
        Point coordinates = vertexPositions[startVertex - 1];

        graphDesignWindow.FillEllipse(visitedVertex, (coordinates.X - vertexWidth / 2), (coordinates.Y - ve
        graphDesignWindow.DrawString(startVertex.ToString(), font, visitedLabel, new PointF(coordinates.X,


        // Run DFS in a separate thread so we can pause it
        await Task.Run(() => RunDFS(startVertex - 1));
        Invalidate(); // Force redraw of the form
        // Chnage it to DFS not running
        isDFSRunning = false;
    }
    else
        MessageBox.Show("Invalid starting vertex. Enter a number between 1 and " + vertexCount);
}
```

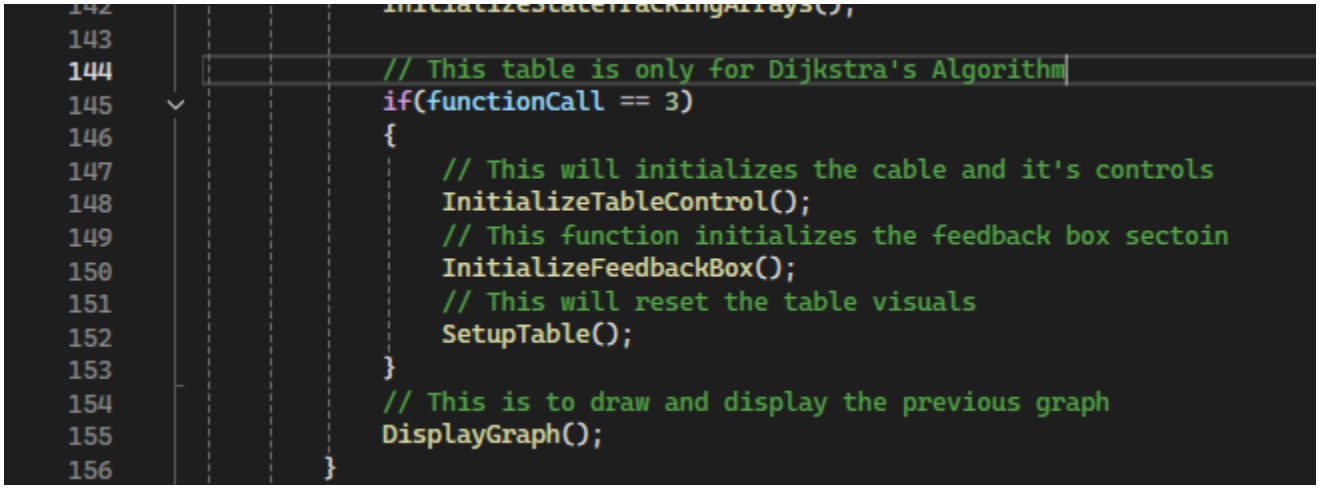


```
1 reference
private async void startDjikstras()
{
    // If there is a graph delete it
    if (graphBitmap != null)
        graphBitmap.Dispose();
    // Set the graph to be the original graph
    graphBitmap = (Bitmap)baseGraphBitmap.Clone();
    // Redraw when able
    Invalidate();

    // This will reset up the table for the next running of Dijkstra's
    SetupTable();


    // Wait for this to be complete before we continue.
    await Task.Delay(500);
    // Try to parse user input for starting vertex
    if (int.TryParse(textBox1.Text, out int startVertex) && startVertex > 0 && startVertex <= vertexCount)
    {
        isDijkstasRunning = true; // This will not allow the start button or refresh button to be clicked w

        // This will initialize the Parent and Status arrays
        InitializeStateTrackingArrays();

        // This creats the graphics on screen to allow drawing or in this case the brush to add verticies
        Graphics graphDesignWindow = CreateGraphics();

        // This will be used to grab the cordinates of the starting vertex as a 0 based index
        Point coordinates = vertexPositions[startVertex - 1];

        graphDesignWindow.FillEllipse(doneVertex, (coordinates.X - vertexWidth / 2), (coordinates.Y - verte
        graphDesignWindow.DrawString(startVertex.ToString(), font, visitedLabel, new PointF(coordinates.X,

        // Run Dijkstra in a separate thread so we can pause it
        await Task.Run(() => RunDijkstra(startVertex - 1));
        Invalidate(); // Force redraw of the form
        isDijkstasRunning = false; // Allow you to click the start button and refresh button
    }
    else
        MessageBox.Show("Invalid starting vertex. Enter a number between 1 and " + vertexCount);
```

- Now it's time to write your algorithm. If your algorithm needs a table and/or feedback box, then you need to follow these steps.

Using the Table:

- If you need to use the table, you need to make sure you edit the below if statement. Note if you don't need the feedback box, then make an inner loop for if(functionCall == 3) then InitializeFeedbackBox();. Once you do that, then you need to make sure you add the SetupTable() function call into the start function of your algorithm like I did in the above picture on the right.



```
143
144         // This table is only for Dijkstra's Algorithm
145         if(functionCall == 3)
146         {
147             // This will initializes the cable and it's controls
148             InitializeTableControl();
149             // This function initializes the feedback box sectoin
150             InitializeFeedbackBox();
151             // This will reset the table visuals
152             SetupTable();
153         }
154         // This is to draw and display the previous graph
155         DisplayGraph();
156     }
```

- After this, you may need to edit the SetupTable function below on line 841. If the Column names need to be changed, then this is where you need to do it. However, in the else statement, I'd ask you to make an if(functionCall ==3) then you do the exact things I have, else if (functionCall == __) where the black is the number you assigned the algorithm.

```
842             {
843                 // This will clear any existing columns and rows inorder to reset the table
844                 table.Columns.Clear();
845                 table.Rows.Clear();
846 
847                 // This will add al the nesicarry columns
848                 table.Columns.Add("Vertex", "Vertex");
849                 table.Columns.Add("Parent", "Parent");
850                 table.Columns.Add("Cost", "Cost");
851                 table.Columns.Add("Possible Parent", "Possible Parent");
852                 table.Columns.Add("Possible Cost", "Possible Cost");
853 
854                 // This will set a consistent width for every column
855                 foreach (DataGridViewColumn col in table.Columns)
856                     col.Width = 80;
857 
858                 // This will populate the table with the initial values
859                 for (int i = 0; i < vertexCount; i++)
860                 {
861                     // This will make sure a row exists. If it doesn't then we need to add one.
862                     EnsureRow(i);
863 
864                     table.Rows[i].Cells["Vertex"].Value = i + 1;
865                     // Defualt parent for all paths starts as -1
866                     table.Rows[i].Cells["Parent"].Value = "-1";
867                     // The defualt cost for all paths is infinate
868                     table.Rows[i].Cells["Cost"].Value = "Inf";   // Default cost is Inf
869                     table.Rows[i].Cells["Possible Parent"].Value = "";
870                     table.Rows[i].Cells["Possible Cost"].Value = "";
871                 }
872             }
```

- Now if you end up using the drawing functions please read the below information. If not you will need to write your own update functions. You can use the UpdateTableDone and UpdateTableUnvisited as a guide to writing your own. However, you need to use the below code. (Note: the ___ is your function name you decide on and the *** is any parameters you will need.)

```
C/C++
if (table.InvokeRequired)
       table.Invoke(new MethodInvoker(() => _____(*******)));
else
{
```

Using the Feedback Box:

- If you need to use the feedback box, then you need to follow the same first step and make sure you edit the below if statement in the constructor. Note if you don't need the table, then make an inner loop for if(functionCall == 3) then InitializeTableControl(); SetupTable();.

```
143
144         // This table is only for Dijkstra's Algorithm
145         if(functionCall == 3)
146         {
147             // This will initializes the cable and it's controls
148             InitializeTableControl();
149             // This function initializes the feedback box sectoin
150             InitializeFeedbackBox();
151             // This will reset the table visuals
152             SetupTable();
153         }
154         // This is to draw and display the previous graph
155         DisplayGraph();
156     }
```

- Now all you need to do is to start adding feedback wherever you want into your algorithm. To do so all you need to do is call AddFeedback() then whatever you want your feedback to be you put inside as shown below. If you want to use any variables in your feedback you need to use the $ before the string quotes. Then when you want to use a variable or do any math, you use {}. You can also print out the feedback as a string, and you just use the quotes “”. You may also call a function inside of it, so long as that function returns a string. **Note:** I would recommend putting this information when you are updating or checking anything. Think of this as you are in the room with the user, and needing to explain each step of the algorithm.

```
    AddFeedback($"The shortest path to vertex {i + 1} has a total cost of {cost[i]}. Path taken:");
    AddFeedback(GetPathString(i));
}
AddFeedback("---------- End of Dijkstra's Algorithm ----------");
```

Using Drawing Functions:

- If you are using the drawing functions, then it is important to note that you may need to adjust it to fit your function, like I did for Dijkstra’s. However, if you are using the drawing functions, and the table you need to update the section of code at the beginning. However, if you are using a different update table function then this is where you need to include it:

```
// This will only trigger if Dijkstra's is running, as we need to update the table for it
if (functionCall == 3)
    UpdateTableUnvisited(from, to, adjacencyMatrix[from, to]);
```

- For reference, this is at the top of the drawing functions (before you lock or use the using). If you are using these, then you need to add an || in the if statement to include your algorithm. If you are using your own, then you need to make sure you do another if(functionCall == ___) where the black is what number you assigned the function to be.

- It is important to note that you may need to consider other types of edges, like cross edges and stuff, so here is information that is important to know when creating your own drawing functions.

Using Your Own Drawing Functions:

- So first and foremost, you **must** use the lock and using shown below:

```
// This will lock the bitmap to ensure this is the only process trying to draw
lock (bitmapLock)
{
    // Seeing bitmaps aren't thread safe, and using is a protected process, we can't have 2 processes attempting
    //to modify this bitmap at the same time. So in order to eliminate potential race conditions, or crashes
    // we use the above lock bitmap
    using (Graphics g = Graphics.FromImage(graphBitmap))
    {
```

- The green comments will explain exactly why you need it, but it is essential.
- You will need to consider drawing a directed edge, so you can literally copy and paste the below code. Note: then unvisitedEdge, is the pen variable at the top of the program.

```
// If the graph is directed, we need to draw differently for the edge
if (isDirected)
{
    // This will create the arrow
    unvisitedEdge.CustomEndCap = new AdjustableArrowCap(5, 5);

    // This creates a new Point called from and to that holds the current vertex position for the from and to vertex
    PointF f = vertexPositions[from]; // This is the position for the from vertex
    PointF t = vertexPositions[to]; // This is the posisiton for the to vertex


    // This code below has been used from this website below, just like in the is vertex function
    // https://www.expii.com/t/standard-form-of-circle-equation-5103

    // These will find the difference between the X and Y coordinates of the 'to' and 'from' points
    float x = t.X - f.X;
    float y = t.Y - f.Y;

    // Calculate the distance between the two points using the Pythagorean theorem
    float dist = (float)Math.Sqrt(x * x + y * y);

    // Find the direction by making the vector (x, y) a unit vector
    // This means making the direction have a length of 1
    float ux = x / dist; // This will find the direction in X
    float uy = y / dist; // This will find the directoin in Y

    // This will adjust the 'to' point so the arrow ends around the edge of the 'to' vertex, not the middle
    PointF adjustedTo = new PointF((t.X - ux), (t.Y - uy));
    // This will draw the directed edge
    g.DrawLine(unvisitedEdge, f, adjustedTo);
}
```

- It is very important to note that you must paint all edges, then both vertices because otherwise, the edge will be painted on top of the vertices and it looks very bad. If you don't need to change an edge, then you can just paint the specific vertex.
- If you do need to repaint a vertex make sure you use the below code. Note: the to or from is the specific vertex number you are repainting. The reason why we add one in the beginning, is because the vertices are 0 based, and we want them

to be visually accurate. () based meaning visually vertex one is 0 in code, hence we need to add one.

```
// Redraw the vertex numbers on the vertex
g.DrawString((from + 1).ToString(), font, visitedLabel, new PointF(vertexPositions[from].X, vertexPositions[from].Y), vertexLabelFormat);
g.DrawString((to + 1).ToString(), font, visitedLabel, new PointF(vertexPositions[to].X, vertexPositions[to].Y), vertexLabelFormat);
```

- You must also have the below code at the very bottom of your drawing function. This occurs after the lock. Invalidate will trigger the repaint, and the sleep, is to delay the visual for the next paint, based on the speed bar.

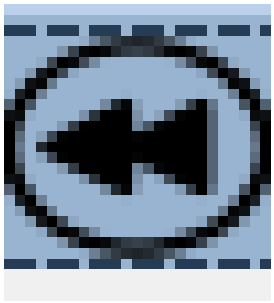
```
    }
    // Repaint when time allows
    Invalidate();
    // Sleep the thread for x amount of seconds where x is speedtime
    Thread.Sleep(sleepTime);
}
```

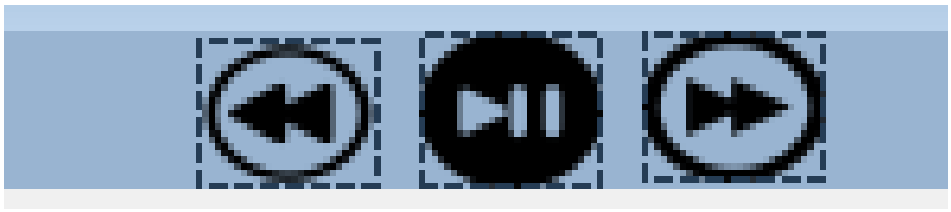
New Algorithm

- Now it's time to write your algorithm. There are a few key things you must include in your algorithm. Anytime you go to paint, (usually based on a status change), you must call the CheckPause() function after it before you do anything else. This is for the pause button, and the fast forward button, (if you implement it the rewind button too).
- The only time you don't do this, is in the case of Dijkstra's where I wanted to print all neighbors right away with no delays. Then I call the Tread.Sleep afterwards, then I call the CheckPause button. That is like the only exception. That way the user can step through every visual animation. Other than that you are all set!!!! (unless you want to implement the rewind button which is below).

Rewind Button Implementation:

- For reference, the below button is the rewind button.

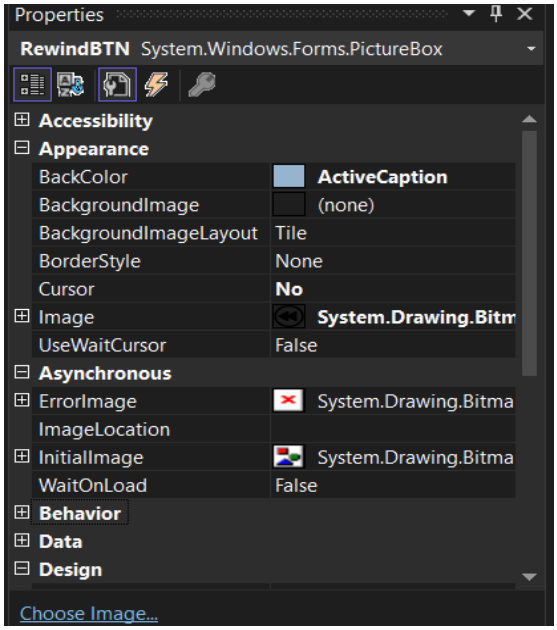




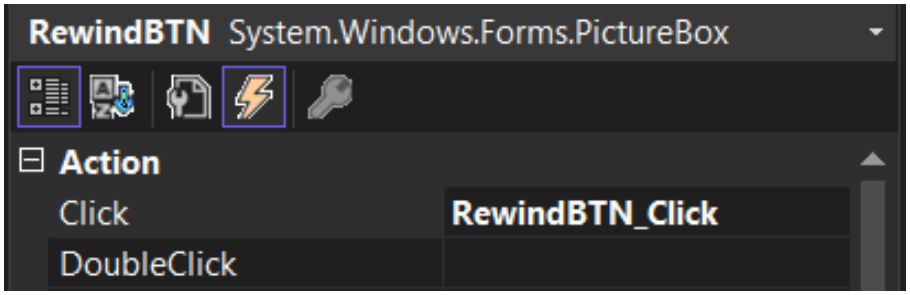
- Below are the two most important functions for this, however the most important is the CheckPause().

```
208     // This function deals with the pause and play button
        1 reference
209     private void PlayPause_Click(object sender, EventArgs e)
210     {
211         isPaused = !isPaused; // Set Paused flags
212
213         // If we are supposed to pause, we need to do so and remove the sleeptimer incase we want to fast forward, that way there is no delay
214         if (isPaused)
215         {
216             // This makes sure sleep is disabled temporarily
217             if(sleepTime > 0)
218                 tempSleeptime = sleepTime;
219             sleepTime = 0;
220             pauseEvent.Reset();  // Pause execution
221         }
222         // Otherwise we are no longer paused, so we need to resume program exectution and reset sleeptimer to the original time
223         else
224         {
225             sleepTime = tempSleeptime;
226             pauseEvent.Set();  // Resume execution
227         }
228
229     }
230
231     // This is for checking on the Play / Pause / Fast Forward / and Rewind buttons
        9 references
232     private void CheckPause()
233     {
234         while (isPaused)
235         {
236             pauseEvent.WaitOne(); // Wait until Play or Fast Forward is pressed
237
238             if (fastForward)
239             {
240                 // Reset after allowing one step
241                 fastForward = false;
242                 // Exit the while loop
243                 break;
244             }
245         }
246     }
```

- When you implement the rewind button, you must first change the Cursor below from no to Hand.



- Then you need to click the lighting bolt, which will take you to the events. And under Action, you should see a click shown below. The box next to the click, is where you name your function. You click in that box, then type out what you want the name to be like RewindBTN_Click or something, then you click enter. This will automatically generate the function for you.



- After that you are ready to start implementing. My first version would recall the whole algorithm from scratch, with 0 delay up to the previous step. So if you are on lets say step 9, you run the algorithm to step 8.
- One thing you can maybe do is create some kind of stack or list of all the steps, so you can just pop the most recent ones off the stack.
- **Key Note:** I would like to give you the biggest advice / information. You will need to keep track of the vertices, as you're going to need to repaint them. This for me was the hardest part as you might need to repaint 2 or 3 vertices and the edges. This is hard to do, and you need to remember, you must paint edges, then vertices.