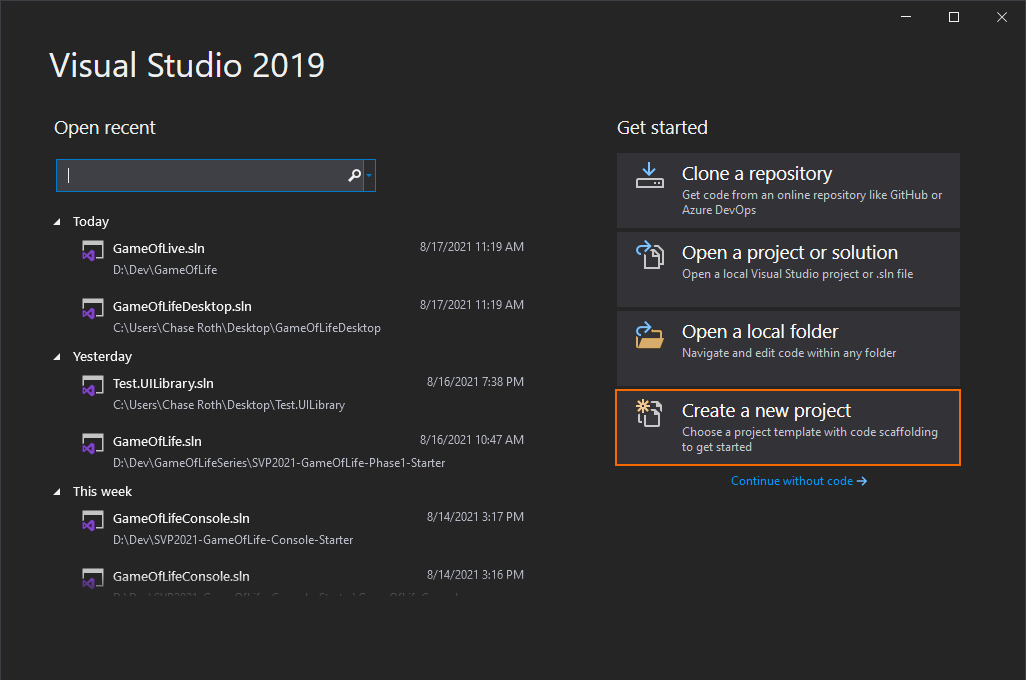
Phase #2

**Building a Desktop Application**

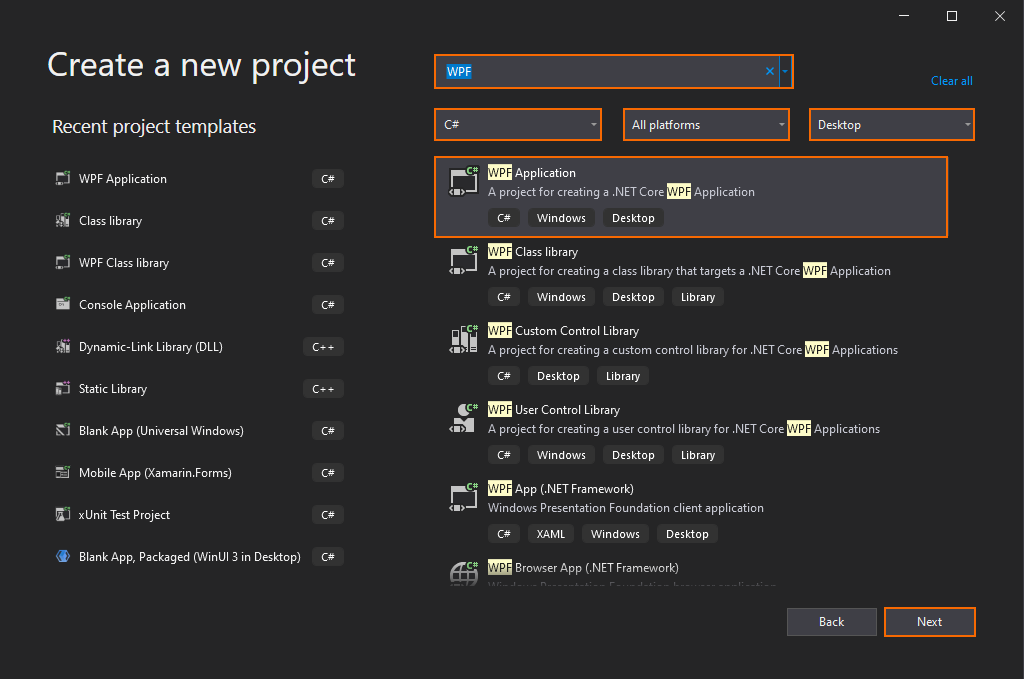
# Step 1 | Creating a Default Desktop App

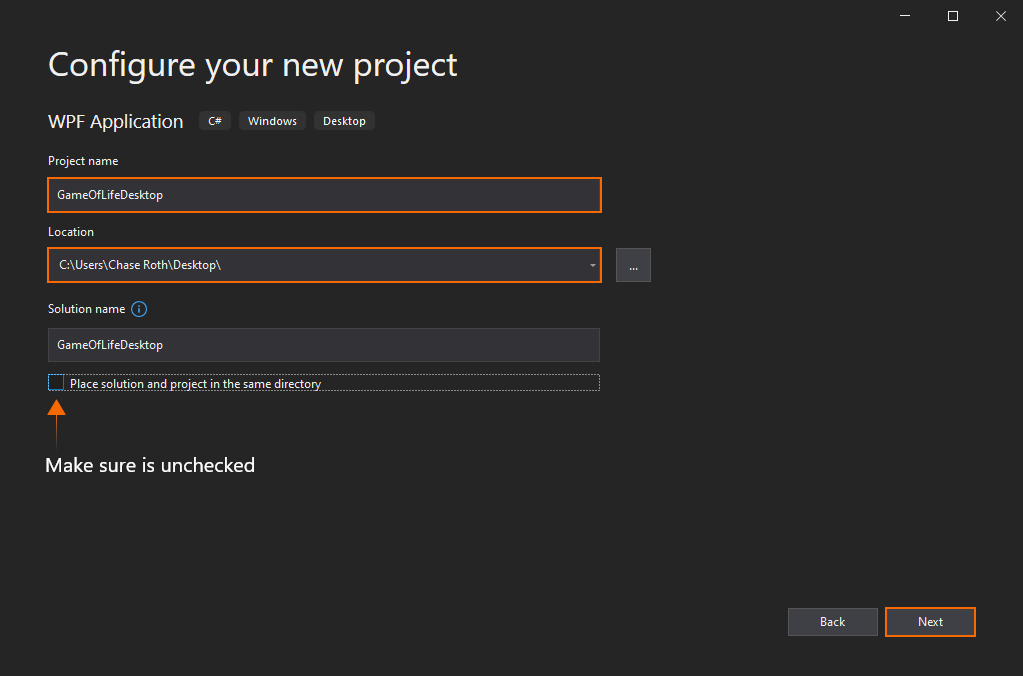
In your first step you will be creating a new Visual Studio 2019 solution with a single desktop project in it. Make sure you follow along closely because these steps are important!

First, you need to open Visual Studio 2019, once opened you should see this:



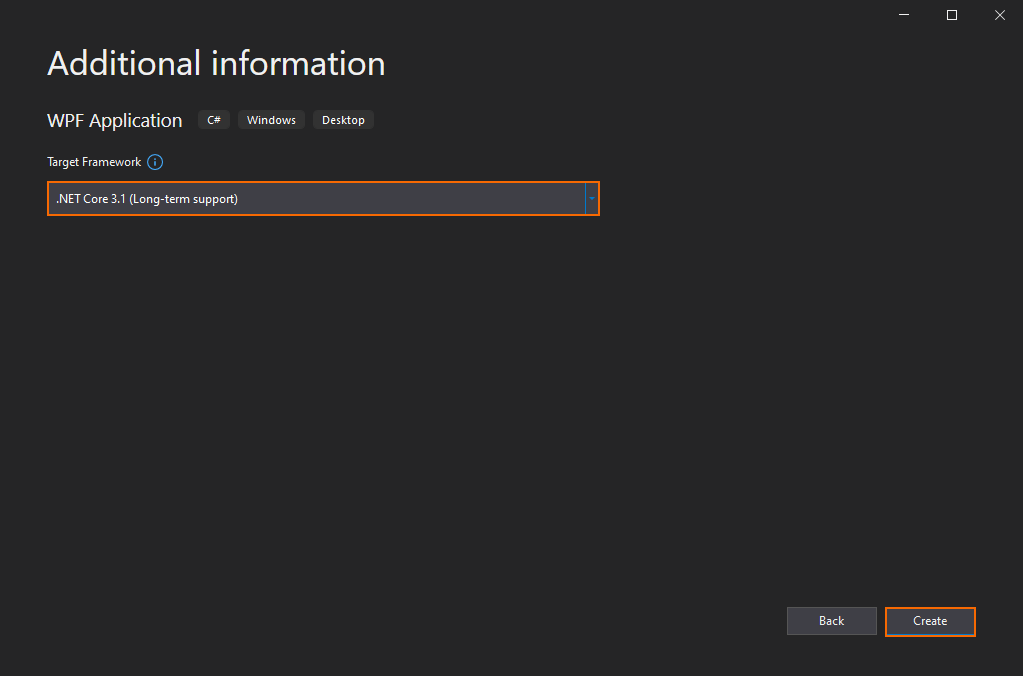
You will want to click on the *Create a new project* button which is highlighted orange in the screenshot. Once you have done that you should then see the following:



Start off by making sure that you type *WPF* into that top search bar and that all three of those boxes in the image match yours. Your boxes will probably be different so click on the dropdown to select the correct one to match the above image. Once you have copied the setup above, you should see the box that matches the *WPF Application* button. Once you select the *WPF Application* highlight, go down to the *Next* button in the bottom right and click that to proceed. You should then be presented with the configuration page:

It is recommended that the location you use for your solution be your desktop *(for the sake of this session)*. In later steps you will need to find your project files and drag n’ drop some other files in, therefore create it somewhere easily accessible. Lastly, make sure you change the appropriate settings so that your window matches the image above. Once they match, click the *Next* button to be presented the following:

|  |  |
| --- | --- |
| **Warning** | If you are un-able to create the solution on your desktop, simply create a new folder on your desktop and tell Visual Studio 2019 to create it inside there instead. Name of the folder like *svp\_ics\_the\_best* or something similar. |



This is the last step for creating the solution. Make sure you select *.NET Core 3.1 (Long-term support)* for the target framework; once done, click on the *Create* button. Now you are ready to continue to *Step 2*!

# Step 2 | Modifying the Default Project

In this step you will configure your existing project.

* + Download two libraries from Github and add them your project (2.1)
  + Remove the default *MainWindow* (2.2)
  + Create a new *MyMainWindow* (2.3)
  + Set *MyMainWindow* as the app’s main window (2.4)
  + Remove old *MainWindow* from startup (2.5)

### 2.1 | Download and Reference Two Libraries

In this step you will be downloading two libraries of code from Github. These two libraries were created to assist you in your app’s creation. You can get the libraries from this link: <https://github.com/MAD-NTID/SVP2021-GameOfLife-Phase2-Starter>. If you go there you should see two libraries; *GameOfLife.dll* and *GameOfLifeDesktop.UILibrary.dll*.

A screenshot of a computer

Description automatically generated

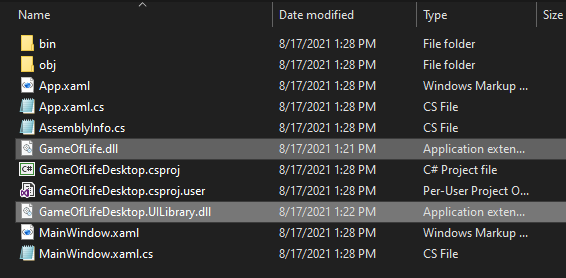
|  |  |
| --- | --- |
| **Tip** | If you click the green *Code* button you can download via a .zip and then unzip where you need once downloaded. This might be easier for some of you so feel free to do it that way. |

Once you have completed the download, open *File Explorer* so that you see the two *.dll* files. After that open another *File Explorer* that shows you all the files inside your desktop app. Remember where you created your project in the previous *Step 1*?

A screenshot of a computer

Description automatically generated with medium confidence

If you are in the right place, you should see files very similar to ones above. Now with both *File Explorers* open, you must drag the two *.dll* files into your project’s folder. After this, you should see them inside the project’s *File Explorer* as follows:



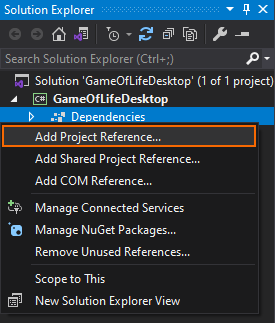
Once you have done that you can close out of both *File Explorers* and go back into *Visual Studio 2019*. Now inside your *Solution Explorer* you should see the two *.dll* files there. A screenshot of a computer

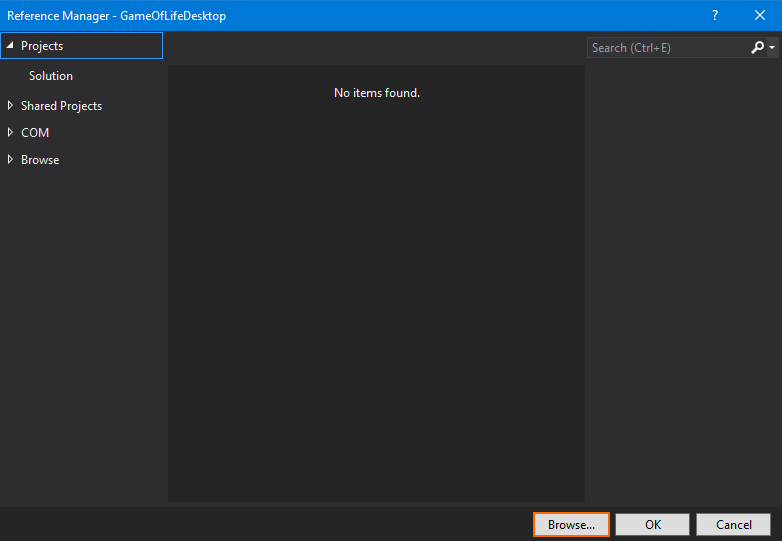
Description automatically generated with low confidence

With that completed, now you need to tell your project to use them, therefore right click on the *Dependencies* text.

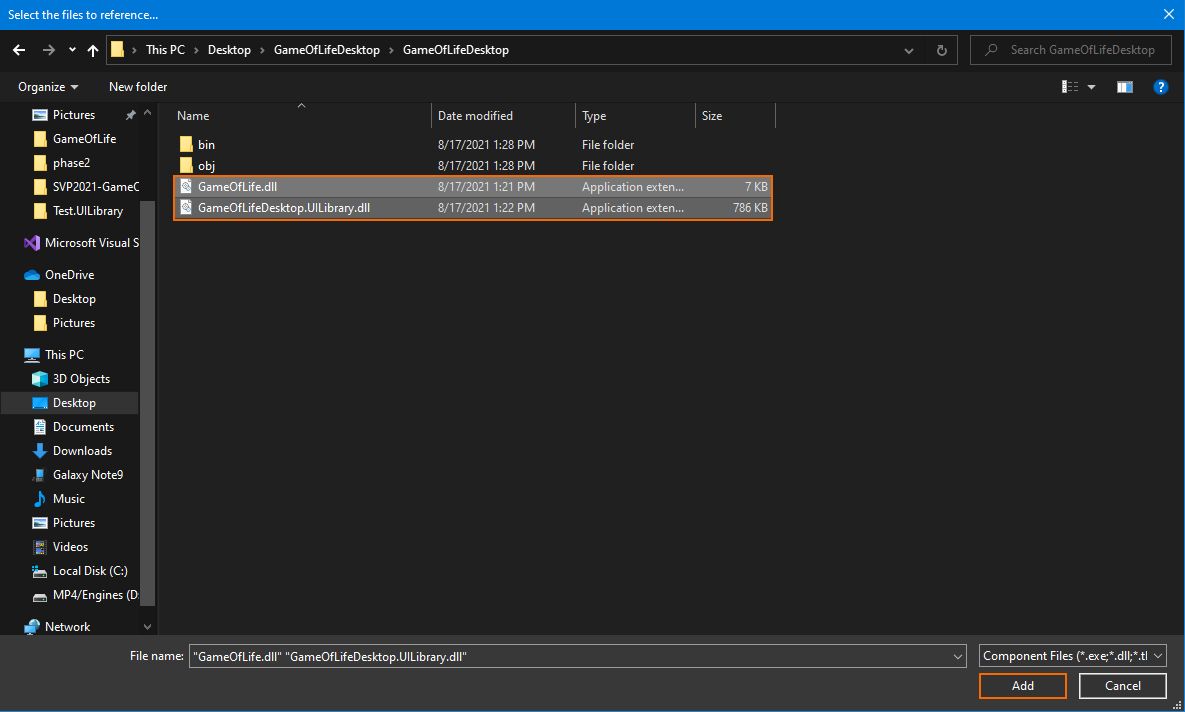
A screenshot of a computer

Description automatically generated with medium confidence  
After right clicking you should see the following:

  
Click on the *Add Project Reference…* button and then you should see:

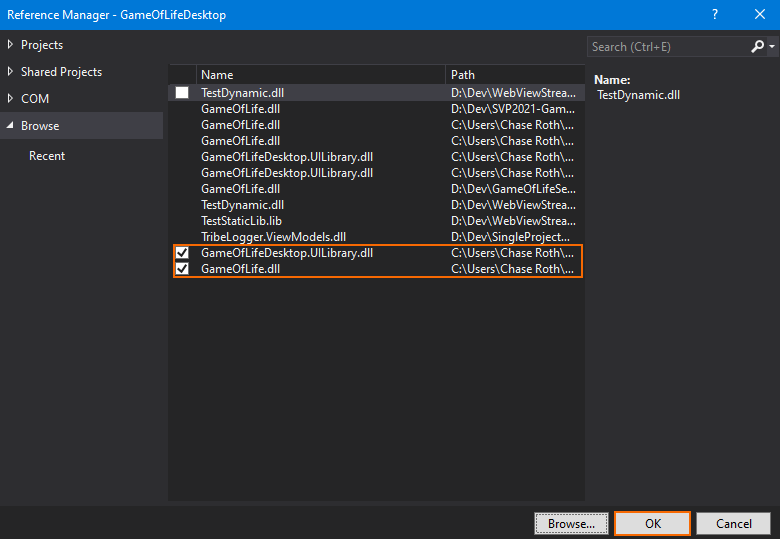


Click on the bottom right *Browse…* button because you are going to need to search for those two *.dll* files.



Remember you just moved the *.dll* files into your project’s folder so you can find them there. Once you find them, select both and then click the *Add* button in the bottom right.

|  |  |
| --- | --- |
| **Tip** | To select multiple files, hold *ctrl* while left clicking. |

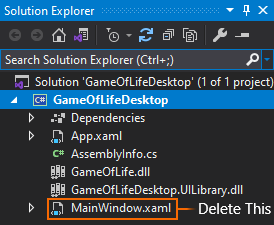


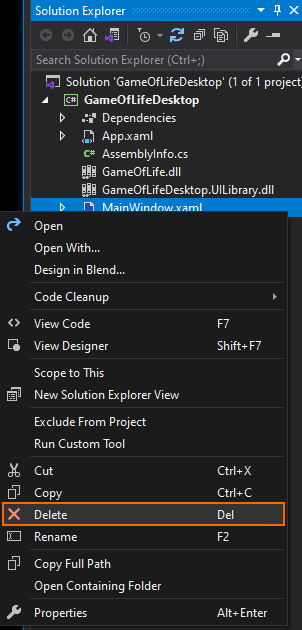
Above you can see the two libraries with the checkmarks next to them *(ignore the other .dll files you see in the example)*. Those two checked marked *.dll* files are the ones you just selected and asked to be added. Now press the *OK* button in the bottom right to continue. After that you should see Visual Studio 2019 again and if you click the *Dependencies’* dropdown button and then *Assemblies’* dropdown you will see the two *.dll* files there.

A screenshot of a computer

Description automatically generated with medium confidence

### 2.2 | Remove the Default *MainWindow*

For this section you will need to remove the *MainWindow* from your project like follows:  
  
  
If you are seeing a *MainWindow.xaml* and *a MainWindow.xaml.cs*, click the dropdown arrow next to *MainWindow.xaml* and then you will have what is above.  
  
Now right click on the file and you should see the image below:

  
Go ahead and click the *delete* button to proceed. With that, you probably received one of these:

Graphical user interface, application

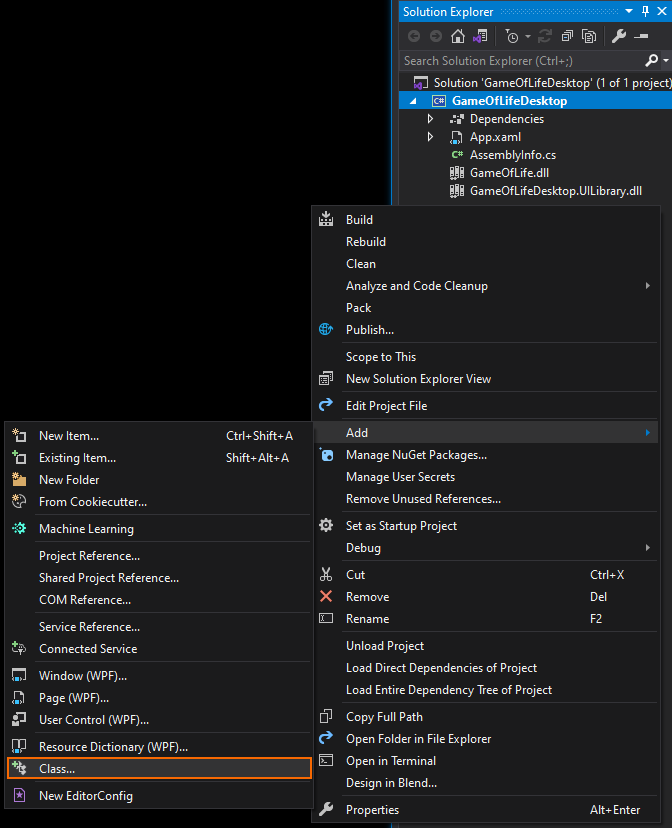
Description automatically generated  
Click on *OK* and you will have successfully deleted the default *MainWindow* provided.

### 2.3 | Create your Own MainWindow

In this section you will create your own *MainWindow* called *MyMainWindow*, but before you do that, make sure your *Solution Explorer* looks as follows:

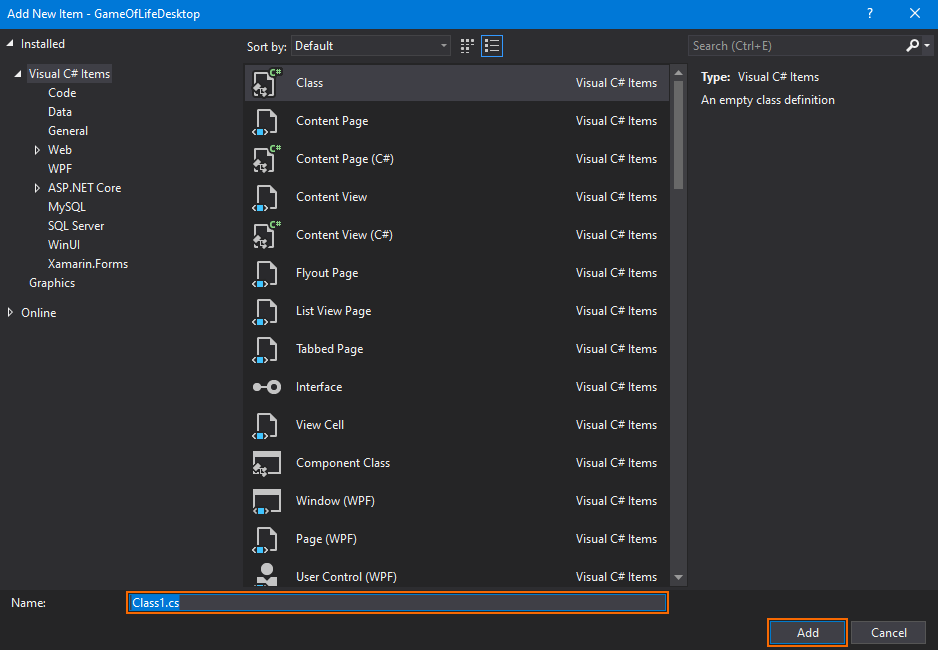
Graphical user interface, text, application

Description automatically generated  
If yours matches the above image, then you are all set to proceed!

Now right click on an empty space in the solution explorer *(the darker gray)* and then hover the *Add* button to see the following:

|  |  |
| --- | --- |
| **Warning** | If you are hovering over the *Add* button yet seeing different results; make sure your *GameOfLifeDesktop* project is selected (highlighted in blue in the above screenshot), and not the *Solution ‘GameOfLifeDesktop’* above it. |

Click on the *Class* button which yields the following:

  
First replace the highlighted text *Class1.cs* with *MyMainWindow* and once done click the *Add* button in the bottom right. Whew, I know this is quite the hassle, but you are almost there! You should now see the *MyMainWindow* showing up in your Solution Explorer as follows:

Graphical user interface, text, application, chat or text message

Description automatically generated  
The bottom file is the one you just created! Visual Studio 2019 should have also opened the code in the text editor as well, revealing:

Text

Description automatically generated  
To begin editing this file you should do two things for starters. First you need to add a reference to the *GameOfLifeDesktop.UILibrary* which you added in previous steps. Secondly, you will need to extend *MyMainWindow* so that it gets functionalities from *GameOfLifeMainWindow* which is provided via the *GameOfLifeDesktop.UILibrary*.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to reference the *GameOfLifeDesktop.UILibrary*? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to the *MyMainWindow* to make it get functionalities from *GameOfLifeMainWindow*? |  |

### 2.4 | Set MyMainWindow as the App’s MainWindow

Now it is time to make your desktop app use the *MyMainWindow* you just created. To start, in your *Solution Explorer* click the dropdown arrow to expose the *App.cs* like the following shows:

A screenshot of a computer

Description automatically generated with low confidence

Double click on *App.xaml.cs* and you should see it open in your text editor. It should contain something like the following:

Text

Description automatically generated

Here with your instructor’s guidance, you are going to expose a built-in procedure that is executed by the *App* on startup. You will need to add code into that procedure responsible for setting *MyMainWindow* to be used as the main window.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you type to expose the startup procedure? |  |
|  |  |  |
| 🡺 | What code did you add to mark the beginning of the procedure’s code? |  |
|  |  |  |
| 🡺 | What command did you add to set the *App.MainWindow* to your *MyMainWindow*? |  |
|  |  |  |
| 🡺 | What command did you usher to show the window? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to mark the end of the procedure’s code? |  |

Only more one step is needed before the app is ready to use *MyMainWindow*!

### 2.5 | Remove the Old MainWindow from Startup

For this brief section you will be removing one line of code from the *App.xaml* file. Essentially since you are using *MyMainWindow* you want to remove the old code that used the one you deleted.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you remove so the app no longer uses the old *MainWindow*? |  |

Now run you Desktop app to make sure it can start. You should see some of the buttons and such which won’t do much right now.

# Step 3 | Starting the Simulation

In this step you will add a procedure that contains code to start the simulation. Before you start though, it is important to note that the infrastructure for this app has been setup for you behind the scenes. Therefore, know that when you can access for say, the *Game* variable without creating it yourself; it has been setup for you already. This is the case for all steps here forth.

* + Expose an existing procedure and add a library reference (3.1)
  + Disabling and capturing provided input (3.2)
  + Create a grid and set its number of rows & columns (3.3)
  + Create a loop that visits all rows (3.4)
  + Create a loop that visits all columns (3.5)
  + Setup the images displaying the instructors in the simulation (3.6)
  + Add *cycleGrid* to the window and start the simulation (3.7)

### 3.1 | Exposing the OnStartSimulation Procedure

In this section you will expose the OnStartSimulation procedure as the title suggest. Don’t worry about the protected override void when exposing this procedure, for those words are beyond today’s session. This goes for all similar sections where you are exposing a procedure. This procedure executes when the *Start* button is clicked. You have already seen the *Start* button when you ran your app in the previous step.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you type to expose the start procedure? |  |
|  |  |  |
| 🡺 | What code did you write to mark the start of the procedure? |  |
|  |  |  |
| 🡺 | What code did you write to mark the end of the procedure? |  |

### 3.2 | Disable & Capture User Input

Here you will be disabling the input controls and then capturing the values the user entered in them. We are disabling them so the user cannot change their values when the simulation is running. Just like the console app you will need three values to be given to the *Game* variable: rows, columns, and the cycle time.

|  |  |  |
| --- | --- | --- |
| 🡺 | What command did you write to disable user input? |  |
|  |  |  |
| 🡺 | What code did you write to capture the number of rows? |  |
|  |  |  |
| 🡺 | What code did you add to capture the number of columns? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to get the cycle time? |  |

### 3.3 | Create a Grid and Set its Number of Rows & Columns

For this section you will add a library and then create a *Grid* (imported from the library you just added) and store it inside of the *cycleGrid* variable. After that you will need to execute a procedure that already exist for you. The procedure name is *SetCycleGridRowsAndColumns*, following how other procedures are told to execute see if you can figure out how to tell it to execute.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to use another library in your *MyMainWindow*? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you write to create a new *Grid* and store it in the *cycleGrid* variable? |  |
|  |  |  |
| 🡺 | What command did you put to set the rows and columns of the *cycleGrid* variable? |  |

### 3.4 | Create a Loop that Visits Each Row

In this section you are writing a loop that visits each row *(sound familiar?)*. This loop will use the *Game* variable and will look essentially identical to the console app’s loop for rows.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add for the first line of the loop? |  |
|  |  |  |
| 🡺 | What code did you add to mark the start of the loop? |  |
|  |  |  |
| 🡺 | What code did you add to mark the end of the loop? |  |

### 3.5 | Create a Loop that Visits Each Column

You guessed it; in this section you will need to add another loop that visits each column just like the console app!

### 3.6 | Setup the Images for Instructors Within the Simulation

In this section you will be setting up the *Image* needed to display a picture in your app. You are doing this inside the loops because just like the console app, you need to present each cell in its column and row. First you need to create a *cell* variable that is basically a cell in your grid. Then you will need to set the row and column the *cell* is to occupy, followed by what image it should display. Lastly, you will need to add it to the *cycleGrid* variable.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you enter to create the *cell* variable? |  |
|  |  |  |
| 🡺 | What command did you use to set the row and column of the cell? |  |
|  |  |  |
| 🡺 | What code did you add to set the image it should display? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What command did you use to add the cell to the *cycleGrid* variable? |  |

### 3.7 | Add cycleGrid to the Window and Start the Simulation

This is the last section of step 3 so rejoice! Although, before you celebrate too early, you need to add two more lines of code! The first line will add the *cycleGrid* to the window’s *mainGrid* variable which will allow you to see your work *(it will show up in the window)*! Secondly, you will need to execute an existing procedure named *Start* that belongs to the *Game* variable.

|  |  |  |
| --- | --- | --- |
| 🡺 | What command did you use to add the *cycleGrid* to the window’s *mainGrid* variable? |  |
|  |  |  |
| 🡺 | What code did you enter to execute the *Start* command which belongs to the *Game* variable? |  |

Now run the app and you should be able to enter values for the text boxes and click start! **(Make sure to select an instructor from the list).** That said, note that things won’t update properly because you are yet to the responsible code.

Graphical user interface

Description automatically generated

Now that is a lot of Bubies!

# Step 4 | Adding the Update Procedure

The previous step resulted in some cool effects, but it could be a lot better. Therefore, lets improve them! For this step you will be adding the update procedure which will allow you to see the instructor population fluctuate.

* + Expose an existing procedure and add a library reference (4.1)
  + Create a loop that visits each row (4.2)
  + Create a loop that visits each column (4.3)
  + Get the cell’s status from the *nextCycle* (4.4)
  + Setup update procedure (4.5)
  + Update image to show if the cell is dead or alive (4.6)
  + Execute update statistics procedure (4.7)

### 4.1 | Expose the Existing *OnNextCycle* Procedure

In this section you expose the *OnNextCycle* procedure that will house all the update code. This procedure will provide you with the *nextCycle* grid went it is executed. Lastly, you will also need to add another reference to the *GameOfLife* library.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to expose the existing *OnNextCycle* procedure? |  |
|  |  |  |
| 🡺 | What code did you add to mark the beginning of the procedure? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to mark the end of the procedure? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you write to add a reference to the *GameOfLife* library? |  |

### 4.2 | Adding a Loop that Visits all Rows

In this section you will add a loop that visits all the rows of the game. You have already done this in the *OnStartSimulation* procedure so you should be able to copy it into the *OnNextCycle* procedure.

### 4.3 | Adding a Loop that Visits all Columns

Like the previous, you will be adding a loop, but this time it is to visit each column. Go head and copy the loop from the *OnStartSimulation* and then you should have the same loop setup there as here.

### 4.4 | Getting the Cell’s Status

In this small section you will need to get the cell’s status at a specified row and column. The *nextCycle* variable provided by the procedure is where you will get the cell from. You will need to store the value retrieved from *nextCycle* in a variable named *dataCell* which you must create.

|  |  |
| --- | --- |
| What line of code did you write to get & store the status of the cell at a specific row & column? | |
| 🡺 |  |

|  |  |
| --- | --- |
| **Tip** | The variable *nextCycle* is structured just like a grid. |

### 4.5 | Setup an Update Procedure

In this section you will setup an update procedure that will carry your message that you will define in *4.6* to the window. The procedure you are going to be executing is called *UpdateWindow* and you will be giving it your own procedure to run! Essentially you are giving a procedure called *UpdateWindow* a custom procedure you created to run. I know this is probably confusing as to why you would need to do this, but again, it is beyond the scope of this session.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you write expose the *UpdateWindow* procedure? |  |
|  |  |  |
| 🡺 | What code did you write to mark the start of the procedure’s contents? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you write to mark the end of the procedure’s contents? |  |

### 4.6 | Update the Image Cell

Here you will add the code that gives all the other code in *OnNextCycle* purpose! Inside of the procedure you just created that is executed by *UpdateWindow*; you will be adding the actual updates. First, you will need to get a reference to the *Image* cell at a specified row and column and then store it in a variable. Then using the *cellStatus* variable you setup in *4.4* you will determine whether the *Image* cell should display an image of the current instructor *(alive)* or nothing *(dead)*.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you write to get a reference to the *Image* cell at a specific row and column, followed by storing it in an *Image* variable? |  |
|  |  |  |
| 🡺 | What code did you write to check to see if the cell is alive? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you write to update the *Image* cell to display the instructor if it is alive? |  |

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to run if the previous *if* found the cell was not alive? |  |
|  |  |  |
| 🡺 | What code did you add to be executed when the cell is dead? |  |

### 4.7 | Execute Update Statistics Procedure

In this small section you will be adding one line of code that executes a procedure that will update the on screen cycleCount and cellsAlive text. The procedure is named *UpdateCycleStatistics,* and it should be executed right before the closing *}* of the *OnNextCycle* procedure.

|  |  |
| --- | --- |
| What line of code did you add to make this update happen? | |
| 🡺 |  |

Now you are ready to test your app and see the results of your work! Go ahead and start the app. You should now see the images disappear and appear when cells are dyeing and being revived once you start a game!

Graphical user interface

Description automatically generated

# Step 5 | Stopping the Simulation

In this step you will add a procedure that contains code to stop the simulation.

* + Expose an existing procedure (5.1)
  + Add code to stop the simulation (5.2)

### 5.1 | Exposing the OnStopSimulation Procedure

In this section you will expose the *OnStopSimulation* just as you exposed the *OnStartSimulation*, and besides the name the rest of the code to do so is the same.

### 5.2 | Stopping the Game

Inside of the *OnStopSimulation* procedure you will need to execute the *Stop* procedure that belongs to the *Game* variable.

After you have done this run your app and you should see that clicking the *Stop* button stops the simulation!

# Step 6 | Resuming the Simulation

In this step you will add a procedure that contains code to resume the simulation when stopped.

* + Exposing an existing procedure (6.1)
  + Add code to resume the simulation (6.2)

### 6.1 | Exposing the *OnResumeSimulation* Procedure

Here you will do the same thing you have done several times before, expose the *OnResumeSimulation* procedure.

### 6.2 | Resuming the Simulation

Here you will insert code into the *OnResumeSimulation* procedure that executes the *Resume* procedure that belongs to the *Game* variable.

Once you have done this, you should be able to now start, stop, and resume your simulation!

# Step 7 | Resetting the Simulation

In this step you will add a procedure that contains code to reset the simulation once stopped.

* + Exposing an existing procedure (7.1)
  + Create a new game and remove the old *cycleGrid* (7.2)
  + Clear user input & enable user input (7.3)

### 7.1 | Expose the OnResetSimulation Procedure

Here you will expose the last procedure needed for your desktop app! Go ahead and expose the *OnResetSimulation* like you have done so with the others.

### 7.2 | Create a New Game & Remove the Old *cycleGrid*

In this section you will need to create a new *GameOfLifeSession* and store it in the *Game* variable. Lastly, you will need to remove the old *cycleGrid* that contains all the *Images* from the window.

|  |  |  |
| --- | --- | --- |
| 🡺 | What code did you add to create a new *GameOfLifeSession* and store it in the *Game* variable? |  |
|  |  |  |
| 🡺 | What command did you enter to remove the *cycleGrid* from the window? |  |

### 7.3 | Clear User Input & Enable It

Welcome to the last section of the last step! In this section you will add two lines of code responsible for clearing the user input and enabling it. Now, after creating this rather lengthy document and looking back… I think you have done enough. Therefore, the code for this last step has been provided for you below 😊

ClearUserInput(); // Clear that user input!

ToggleInputEnabled(true); // Enable them to write in the text boxes!

After adding those last two lines run your app and you should be able to reset the simulation and then start a new one with different values! With that, you have completed the desktop app and feel free to play around with things!

🎉 Congratulations! 🎉