THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2024

Contributors

J. Turner

R. Mercado

V. Moxham-Bettridge

A. Hinze

C. Pilbrow

N. Kanji

T. Elphick

S. Cunningham

INTRODUCTION TO GRAPHICAL USER INTERFACES

Last session we looked at console programming and the Command Line Interface (CLI) to introduce you to common programming structures. This week we will introduce you to Graphical User Interfaces (GUIs), which is the type of interface you are more likely to be familiar with.

A Brief History of Computers

To understand where GUIs fit in the larger context of programming a brief history of computers and their evolution is useful. The first "computer" was built in 1822 by Charles Babbage, it was called the Analytical Engine (see figure 6) which was capable of performing automatic calculations. Ada Lovelace, the world's first computer programmer, worked with Babbage on creating a programmable language for the engine.

Figure 6: Babbage's Analytical Engine

Retrieved from: https://www.computerhistory.org/babbage/engines/

In 1890, Herman Hollerith designed a punch card system (based on Joseph Marie Jacquard's original punch cards created in 1801) to calculate the 1880 census; this saved the US government $5 million. Early computers would adopt this punch card system.

Alan Turing presents the universal machine, later called the Turing machine, that is capable of computing anything that is computable in 1936. This continues to be the central concept of the modern computer today.

John Mauchley and J. Presper designed the Electronic Numerical Integrator and Calculator (ENIAC) in 1943-1944, which is considered to be the grandfather of digital computers. It took up a 20 by 40-foot room and had 18,000 vacuum tubes. They then received funding to build the UNIVAC which is the first commercial computer for business and government applications.

With the invention of the resistor in 1947, vacuum tubes were no longer needed, reducing the size of the computer. In 1953 Grace Hopper invented the first computer programming language, COBOL (see figure 7).

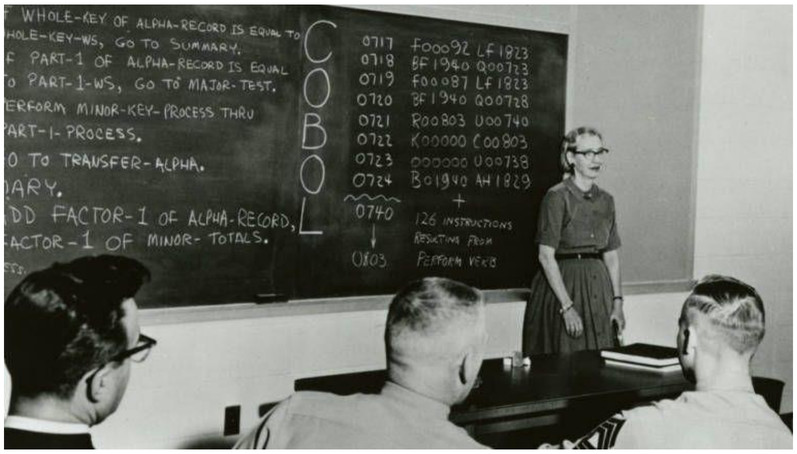


Figure 7: Grace Hopper teaching COBOL

Retrieved from: https://digital.hagley.org/islandora/object/islandora:2514824/datastream/MEDIUM/view

At this point in time computers are still quite large and do not have a standard output. It is not until the mid 1960s that the first glass teletypes entered the market. Note the terminal-like display they used (see figure 8).

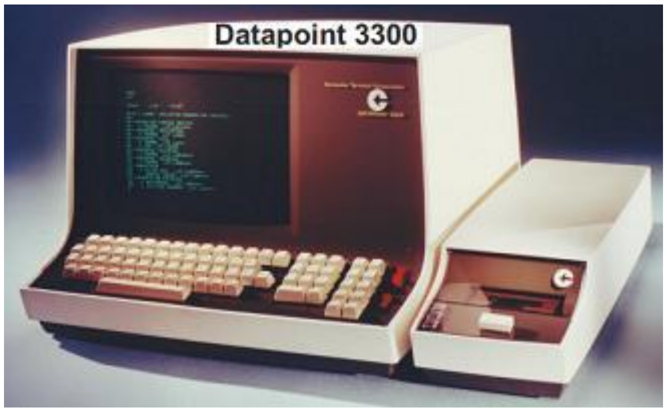


Figure 8: Datapoint 3300

Retrieved from: https://www.scaruffi.com/science/images/datapoint3.jpg

In the late 1960s one of the first operating systems (UNIX) was produced at Bell labs, written in the C programming language. Modern computing operating systems, such as Mac OSX or Linux are built upon the principles (and in some instances code) of this early operating system.

It was not until 1973, 51 years ago, that computers had their first GUI monitors with the introduction of the Xerox Alto computer (figure 9).

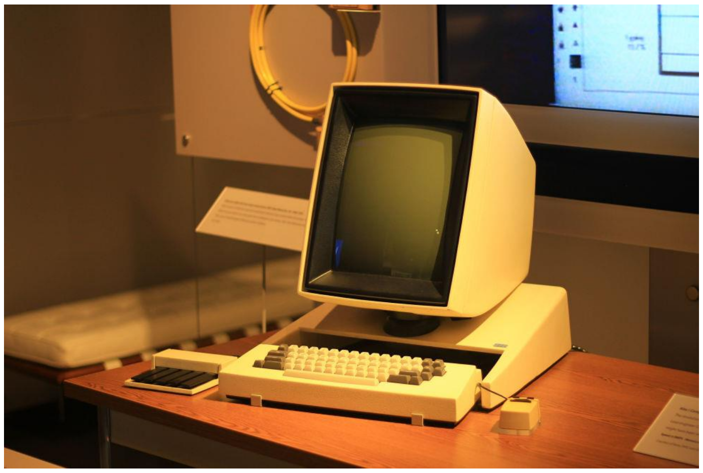


Figure 9: Xerox Alto GUI Computer
Retrieved from: https://cdn.arstechnica.net/wp-content/uploads/2016/06/11400989606_a7964d27bb_k-980x653.jpg

This then led onto IBM personal computers, Apple, Microsoft and so on, finishing with the smartphone in present day, the most widely used computer on the market.

Why learn to program the CLI and GUI?

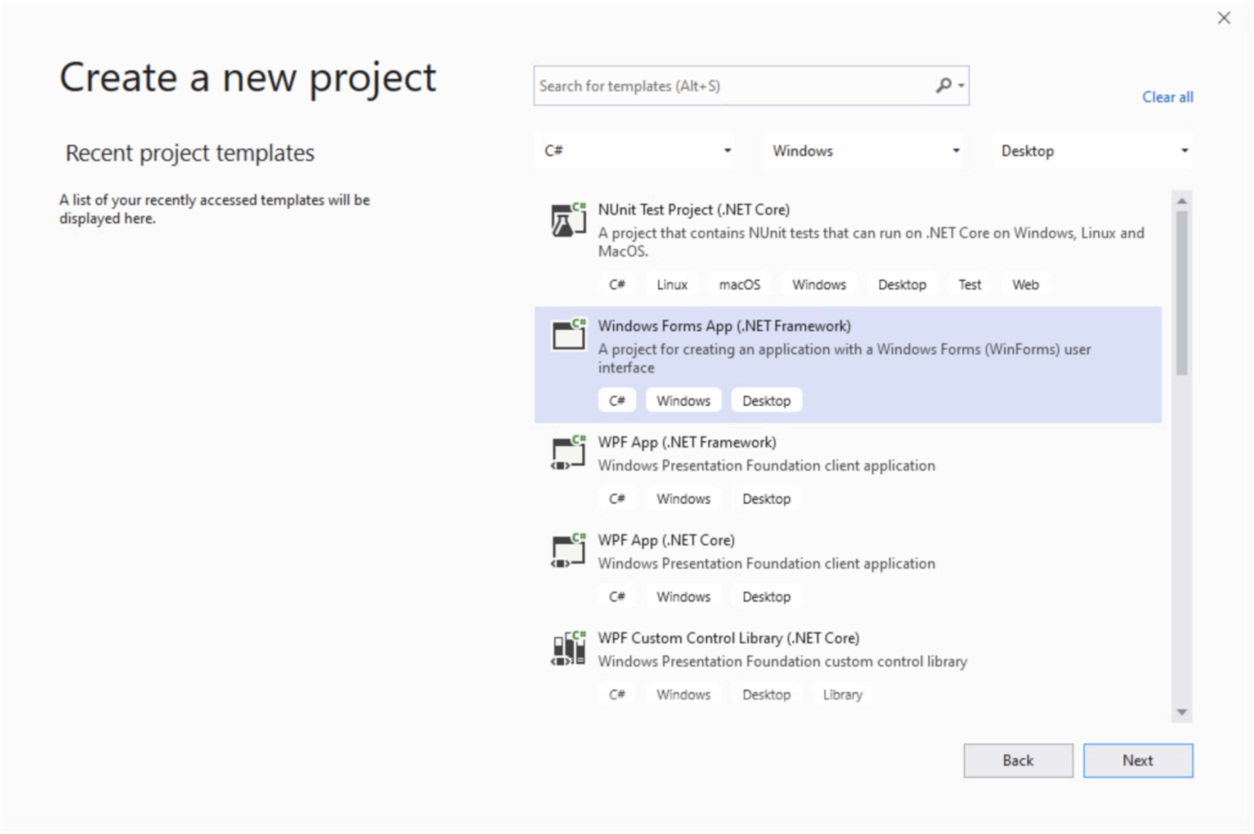
The invention of the GUI was great for making computers more accessible to the general public. Before this, users had to remember commands for the CLI, which was great for programmers but not so great for others.

The CLI is still used today by programmers as it allows us to use less system resources to run programs than with a GUI. It is used amongst software developers and engineers to run and test "hidden" code essential to functionality that the users do not interact with.

In the next section we will get started with building a GUI for a simple program and then look at how to program each different element of the interface.

GUIs in Visual Studio

To get started with creating GUIs in Visual studio you will need to create a Windows Form app (.NET Framework) project (see figure 10).



A list of your recently accessed templates will be displayed here.

NUnit Test Project (.NET Core)
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop Test Web

Windows Forms App (.NET Framework)
A project for creating an application with a Windows Forms (WinForms) user interface
C# Windows Desktop

WPF App (.NET Framework)
Windows Presentation Foundation client application
C# Windows Desktop

WPF App (.NET Core)
Windows Presentation Foundation client application
C# Windows Desktop

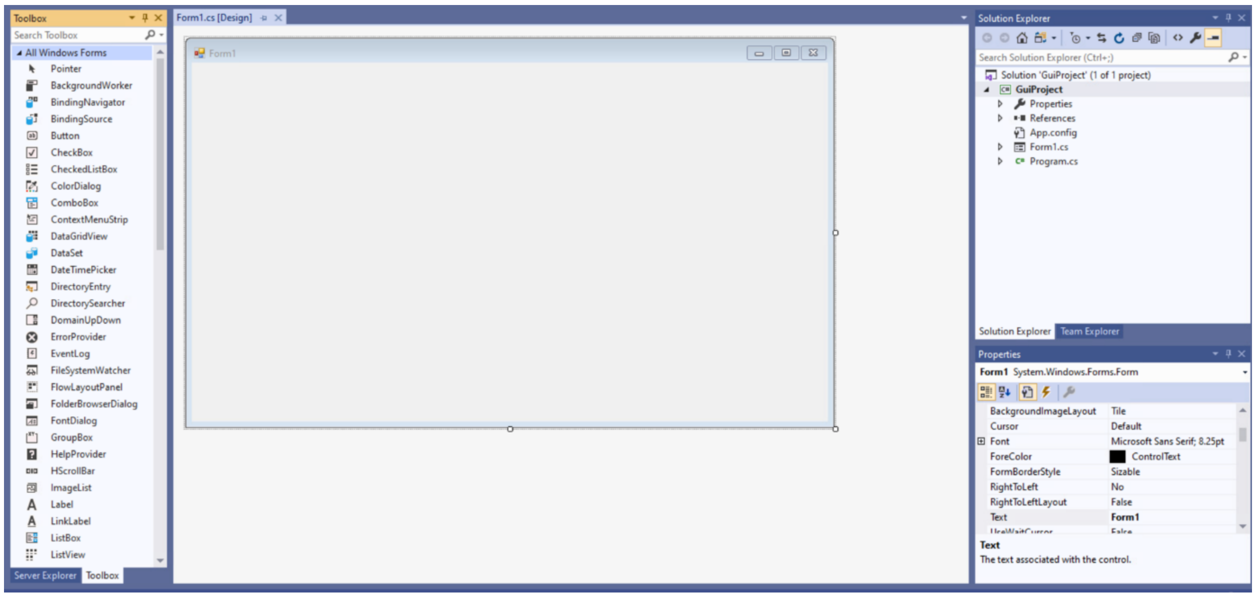
WPF Custom Control Library (.NET Core)
Windows Presentation Foundation custom control library
C# Windows Desktop Library

Figure 10: Creating a GUI project

You will then be able to configure your new project, don't change anything except the Project Name. We are going to call our project "GuiProject".

When the project opens you will see a blank form which you can add widgets to (see figure 11), widgets are things like buttons or text boxes that allow the user to interact with the interface. On the left you should see a panel called "Toolbox" (if you don't you can add it from the View menu), this lists all the widgets that you can include in your form. You can drag and drop different widgets onto the form and place them in different locations to alter what the form looks like.

The other panel which is relevant to us is the "Properties" panel on the right bottom corner of the screen. When we add widgets and other GUI elements to our project we can change their properties from this area. This can include things like what the name of a particular widget is in the code as well as what text is displayed on the widget itself.



Form1 System.Windows.Forms.Form

BackgroundImageLayout	Tile
Cursor	Default
Font	Microsoft Sans Serif; 8.25pt
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**Form1**
UseWaitCursor	False

Text
The text associated with the control.

Figure 11: Visual Studio GUI project

Let's have a go at creating a simple form for the game of UNO using our GUI project (it won't be functional yet!). If you haven't played UNO before you can find the rules to the game at the following link: https://service.mattel.com/instruction_sheets/42001pr.pdf.

Exercises:

1. First, we will need to display a deck and hand for the user. To do this we will add the following elements:

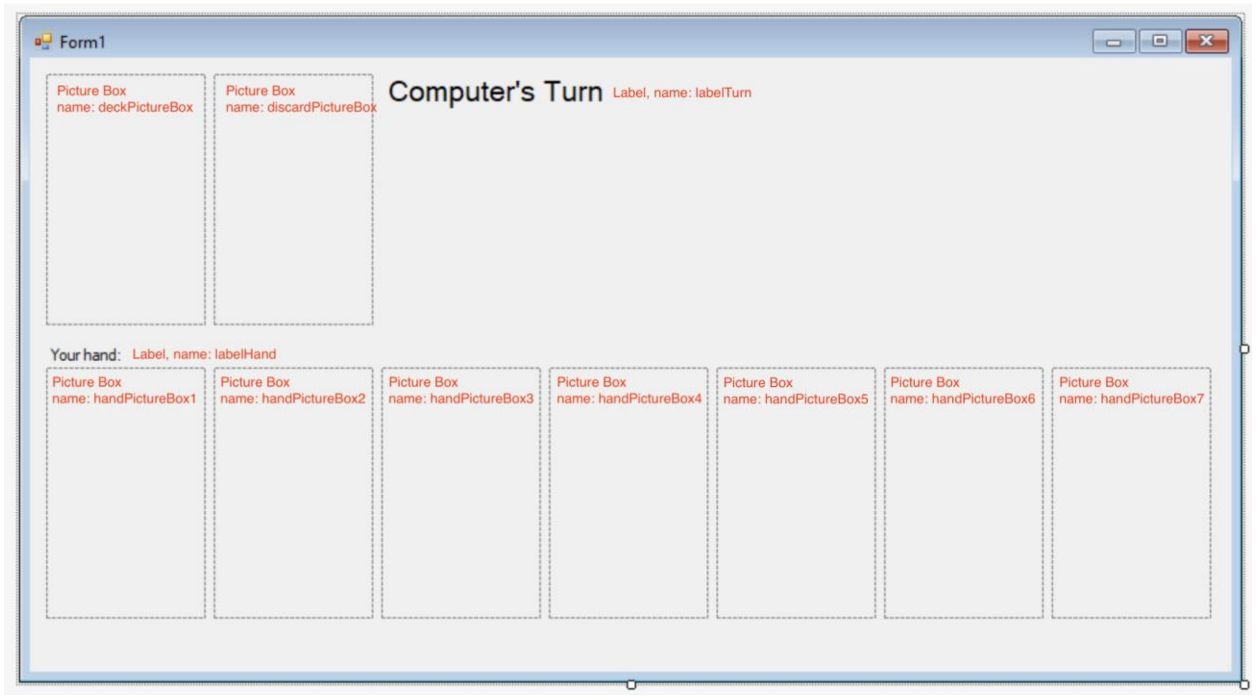


Figure 12: Form Elements

2. Let's view the code for the form we created by selecting the form and pressing the view code button:

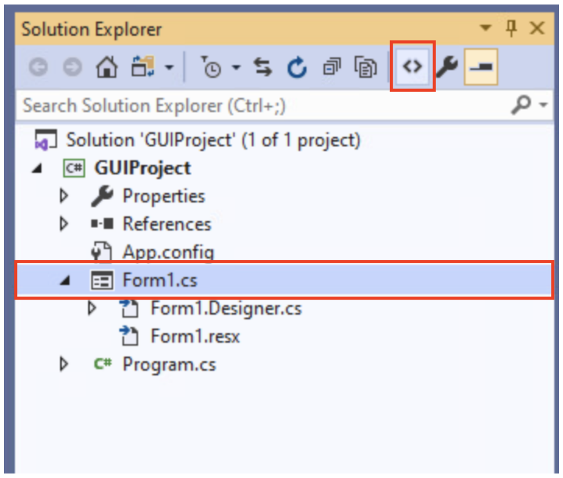
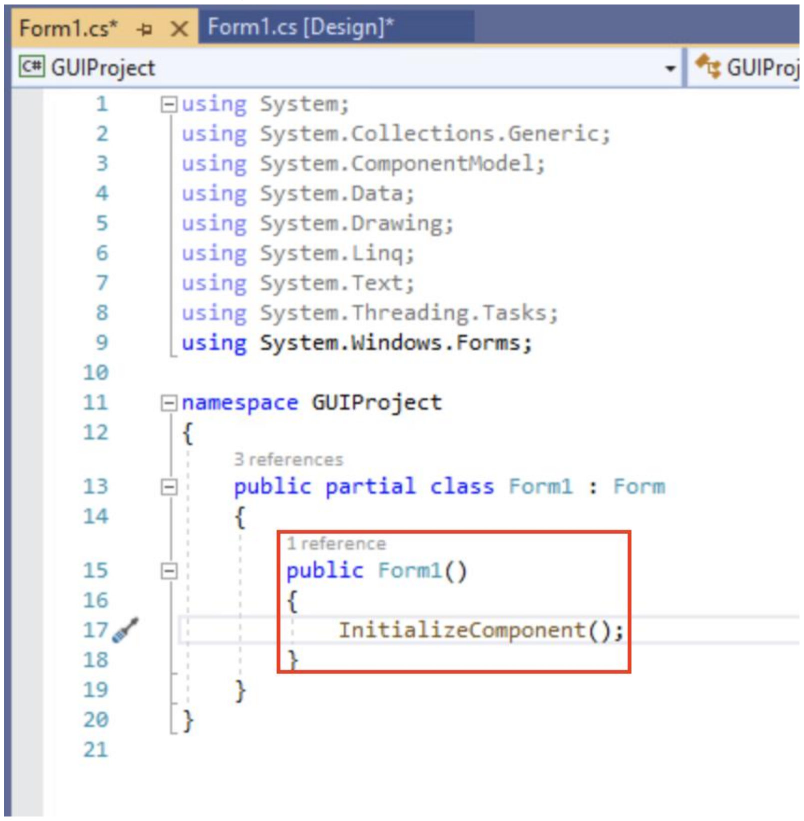


Figure 13: Solution Explorer

3. You should now be able to view the code for the form. We are going to modify the constructor so that we display UNO cards in our picture boxes.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GUIProject
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Figure 14: The Constructor function of Form1

On the Slack channel, we have added some images for UNO cards, let's look at how we can load these in a picture box.

4. First, ensure that you have copied the images into your project folder. You can do this by creating a new folder in your project and adding existing files to that folder. To do this, right click on the project and select Add > New Folder.

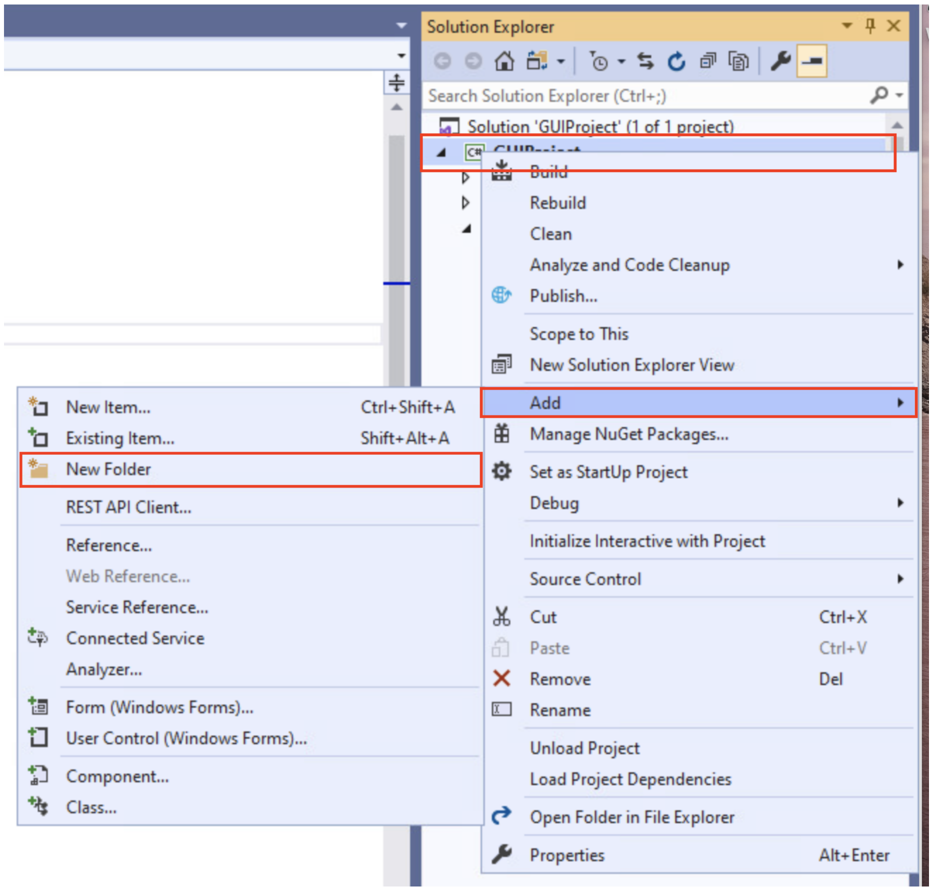


Figure 15: Add a New Folder

5. Once the folder is created, right click on the folder and add an existing item.

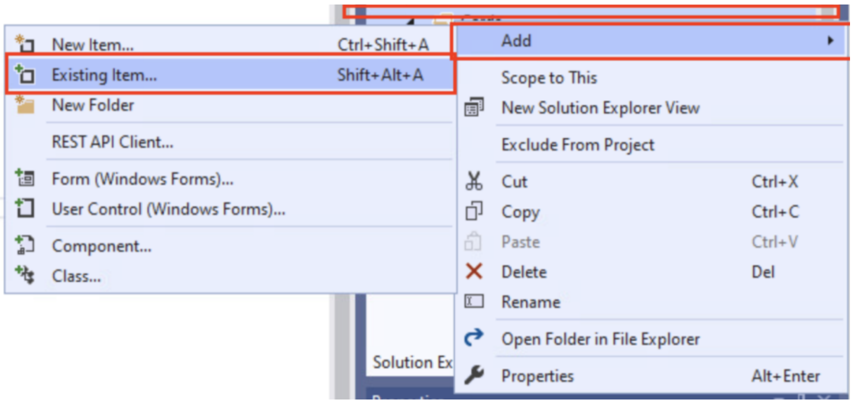


Figure 16: Add Existing Items

6. Then add all the card images using the add window. If you can’t see the images, ensure that you have the “All Files” option selected.

7. In your form1 code change the constructor function to the following:

```
15    public Form1() {
16          InitialiseComponent();
17          deckPictureBox.Image =
            Image.FromFile(“../../Cards/card_back_large.png”);
18    }
```

Now try running the code, you should notice that the image is too large for the picture box. We can fix this by adding the following code:

```
deckPictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
```

In order to display the rest of the deck and a hand we need to generate a random hand for the player and a random deck. For simplicity, we will assume we have exactly one of each card (54 total), as opposed to the real Uno game which has 108 cards.

8. Let’s create a new function which will generate the card to display for the discard pile.

```
6     using System.IO;
16    public void Form1() { … }
23    public void generateDeck() {
24          string dir = “../../Cards”;
25          FileInfo[] files = new DirectoryInfo(dir).GetFiles();
26          Random rand = new Random();
27          int index = rand.Next(files.Length);
28          string card_path = dir + “/” + files[index].Name;
29          discardPictureBox.Image = Image.FromFile(card_path);
30          discardPictureBox.SizeMode =
            PictureBoxSizeMode.StretchImage;
31    }
```

On line 23 we declare the name of the function. Line 24 specifies the path to the directory where we want to get our random cards from. Next, we get all the files from that particular folder and store them in a FileInfo array. FileInfo is simply another form of data type that we haven’t encountered yet, all you need to know for this program is that it stores lots of information on the files in a directory.

On line 26 we create a Random variable which allows us to get random numbers, we will use this to randomly select a card from the Cards folder. We do this by calling the Next function on line 27 and passing it the length of our files array. This will generate a random int variable between 0 and up to (but not including) the length of the array. On line 28 we create the valid path to the card, then on lines 29-30 we simply display this as the discard pile as we did for our deck pile.

At the moment while we have defined the functionality to generate the discard card, we haven't actually called our function, which means it won't be executed.

9. In the Form1 constructor function add a call to our generateDeck function by using the following:

   ```
   generateDeck();
   ```

   Now try running your code a few times, you should see the card of the discard pile randomly changing.
10. Can you get random cards for each of the player's hand cards? How could you use a loop to reduce the amount of duplicate code?
11. How can you ensure that you don't end up selecting the same card more than once (i.e. avoid duplicates)?

Summary

In this session we demonstrated the difference between programming the CLI and GUI. We introduced some simple concepts, such as how to display widgets in a form window and how to randomly select objects.

At this stage your Uno game is not functional, in that it doesn't allow a user to play a game against the computer. We will revisit the Uno game in later sessions when we introduce you to event driven programming so that you can make the game more interactive.

Advanced Exercises

Already know the basics of GUI programming? Start working on making the user interface more interactive, you will need to implement the following functionality:

- How does a user place a card on the discard pile?
- How does a user select a card from the deck?
- How can we display a hand which has more than seven cards?
- How do we create an "AI" for the player to play against?

Useful Resources

- GUI programming Introduction: https://www.youtube.com/watch?v=odjdWym0t4I
- C# documentation Windows Form Apps: https://docs.microsoft.com/en-us/visualstudio/designers/windows-forms-designer-overview?view=vs-2019
- C# Random Objects: https://docs.microsoft.com/en-us/dotnet/api/system.random?view=net-5.0#Instantiate
- Random Non-repeating Numbers: http://www.functionx.com/csharp1/topics/randomnumbers.htm
- C# documentation click events: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.control.click?view=net-5.0