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

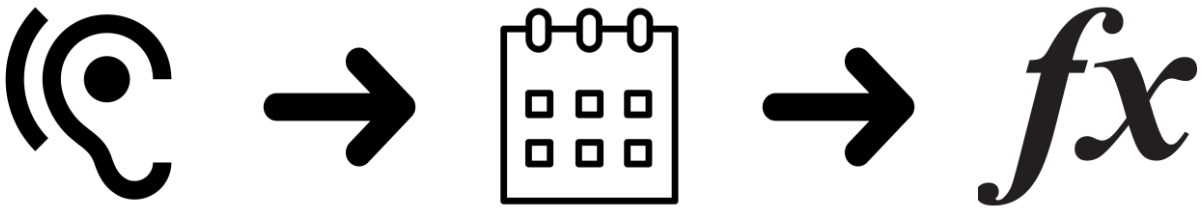
T. Elphick

S. Cunningham

INTRODUCTION TO EVENT-DRIVEN PROGRAMMING

Last session we started building a program for playing Uno, but we only got so far as to displaying cards on a form. On the Slack channel you will find a template for this session (UNO.zip) where we will start looking at event driven programming and filling in some of the functionality for our Uno game.

What is event-driven programming?



Listener *hears an* *Event* *triggers a* *Function*

In our previous sessions we have looked at what are called *procedural* or *sequential* programs. These programs are like a set of instructions which run from line 1 to the last line of the program.

Event-driven programming takes a slightly different approach, in that some event triggers some code to be executed. This approach to programming is particularly useful in GUI programs, as they respond to user actions, but we have no way of knowing when these actions will occur. Therefore, we create a "listener" which waits for a user action to occur, triggering an "event" which executes some code often referred to as the callback function.

So what types of events can we have in a GUI program? Examples include a user clicking on a button, scrolling through a list, hovering over certain widgets and so on. Today we will look at how C# handles events and how we can use these to provide essential functionality to our Uno game.

Creating Events for Uno

Make sure you download the Uno.zip template from the Slack channel. Unzip the folder and then open the solution in Visual Studio (if you have issues ask for help). Let's have a look at what has changed in our template.

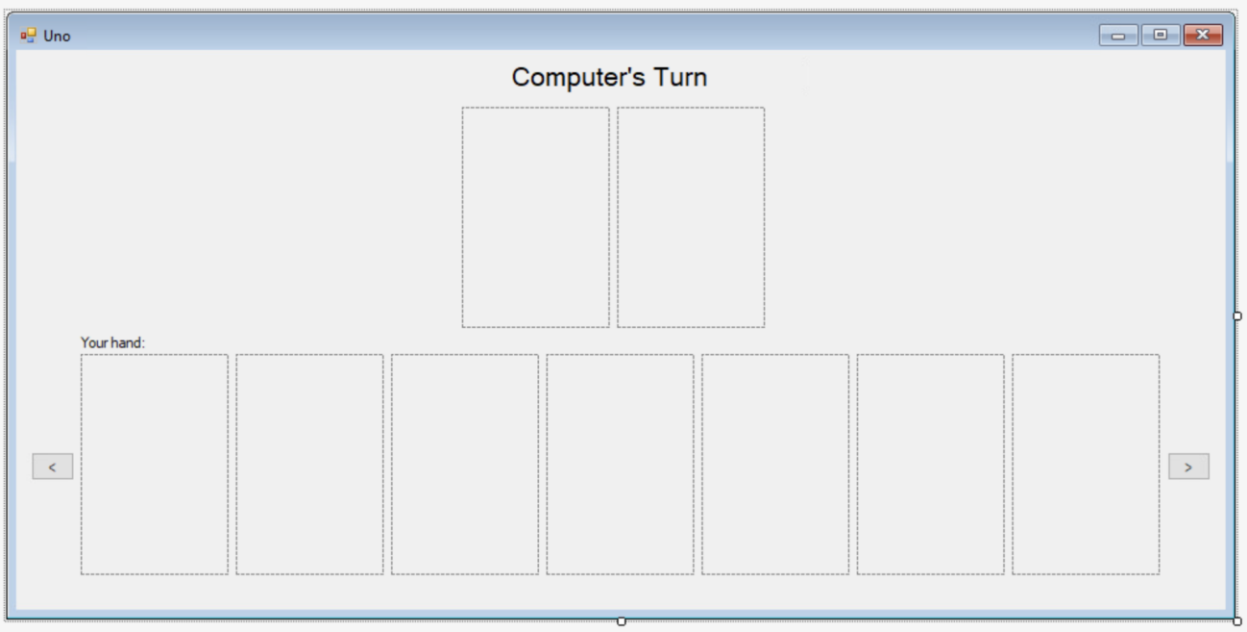


Figure 17: Updated Uno Template

In our new template we can see that there have been buttons added to scroll through the player's hand (either to the left or right). This will allow us to handle when a player has more than seven cards. We have also changed the layout of the form slightly to better reflect the layout of an actual game. Next, let's see what's changed in the code.

We now have two different relevant C# files, Card.cs and Form1.cs. Card.cs contains the class for a Card object, this keeps the important details and attributes of a card contained within the class. Form1.cs looks a lot different than the last time we saw it, the documentation provides details on what each method does, familiarise yourself with this before moving onto the next set of exercises.

When a player's turn occurs they have a few options, they can play a card from their hand or pick up a card from the pile. Starting with the easy option, let's handle picking up a card from the pile. To pick up a card on their turn, the user should click on the deck pile, the game should then add that to their hand and allow them to scroll through all the cards they can see.

Exercises:

1. In the form window, double click on the picture box for the deck. You should see a new function called "deckPictureBox_Click" created. This is the function which will be executed when the user clicks on the picture box.
2. First we need to check if it is the player's turn, if it is we can add the next card from the deck to their hand.

```
private void deckPictureBox_Click(object sender, EventArgs e){
      if(turn==true){
            Card pickup = getCardFromDeck();
            hand.Insert(0, pickup);
            displayPlayer();
      }
}
```

In the above example we get a random card from the deck and insert it at position 0 into the player's hand (i.e. the first card). We then refresh the cards displayed to the user so we can see the difference.

3. Double click on the left button to create a click event for the left button. If a user clicks on this button the easiest way to update the display is to remove the card at position 0 and add it to the end of the list, giving the impression of shuffling through the cards. Add the following code to the method:

```
private void buttonLeft_Click(object sender, EventArgs e) {
      Card c = hand[0];
      hand.RemoveAt(0);
      hand.Add(c);
      displayPlayer();
}
```

Note that the user can shuffle through their cards regardless of whether or not it is their turn, that is they should be able to work out what cards they have in their hand. Next, we will look at the right button and follow a similar approach.

4. Double click on the right button to create a click event for the right button. If a user clicks on this button the easiest way to update the display is to remove the last card and add it to the front of the list, giving the impression of shuffling through the cards. Add the following code to the method:

```
private void buttonRight_Click(object sender, EventArgs e) {
      Card c = hand[hand.Count - 1];
```

```
        hand.RemoveAt(hand.Count - 1);
        hand.Insert(0,c);
        displayPlayer();
    }
```

Again, note that we do not care whether or not it is the player's turn. The last event we need to implement is the tricky part, in when a player plays a card. This is complex as we need to ensure that the rules of the game are adhered to, i.e. we need to know the colour of the card on the discard pile and the colour of the card that the user has selected.

5. First, let's add a variable at the top of the code to store the selected card:

```
...

public const int HAND_SIZE = 7;

bool turn = true;

Card cardToPlay = null;

...
```

6. Now, let's create a Click event handler for each of the picture boxes. This time instead of double clicking on the box we are going to create one separate function for all the picture boxes. Enter the following code:

```
private void HandPictureBoxClick(object sender, EventArgs e) {
        PictureBox pictureBox = ((PictureBox) sender);
}
```

7. At the moment the code is not attached to the picture boxes. We can fix this in the Design view of the form. Select one of your picture boxes and click on the events icon.

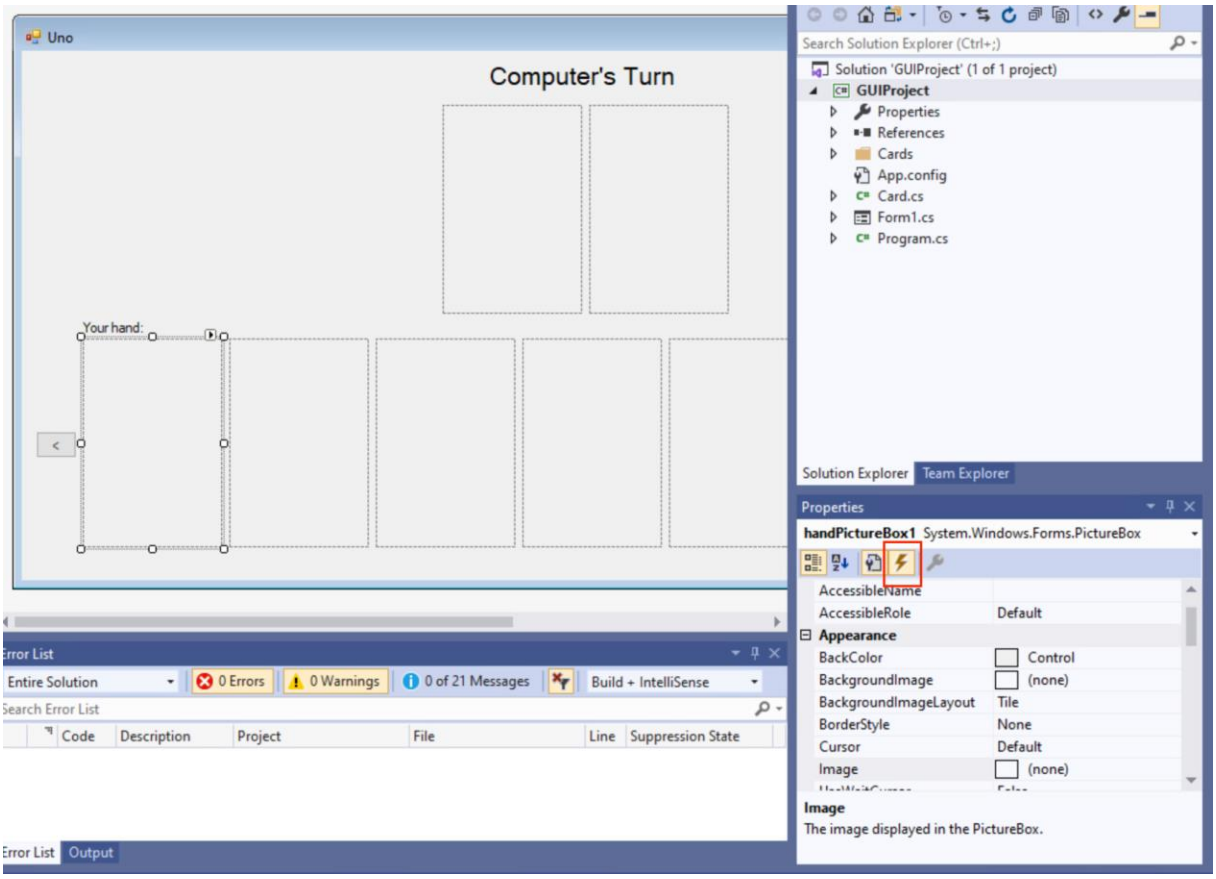


Figure 18: Event Properties

8. Change the Click action to HandPictureBox click. Then repeat this for the other 6 hand picture boxes.

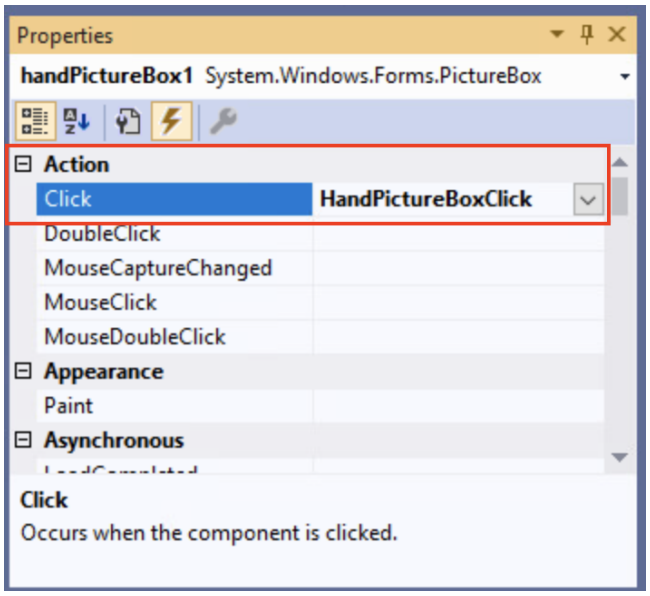


Figure 19: handPictureBox1 Click Event

9. Update the HandPictureBoxClick function with the following:

```
private void HandPictureBoxClick(object sender, EventArgs e) {
      PictureBox pictureBox = ((PictureBox)sender);
      pictureBox.BorderStyle = BorderStyle.Fixed3D;
      String name = pictureBox.Name;
      int num = Int32.Parse(name[name.Length - 1].ToString());
      int index = num - 1;
      cardToPlay = hand[index];
}
```

10. If you run the code you will see that you are able to select multiple cards, let's fix this by creating a new function called "clearSelection()" which resets the borders for all the picture boxes:

```
private void clearSelection() {
      cardToPlay = null;
      List<PictureBox> pictureBoxes = new List<PictureBox>()
{ handPictureBox1, handPictureBox2, handPictureBox3, handPictureBox4,
handPictureBox5, handPictureBox6, handPictureBox7 };
      for(int i = 0; i < HAND_SIZE; i++){
            pictureBoxes[i].BorderStyle = BorderStyle.None;
      }
}
```

11. Now we need to call this method, before we do anything else in our HandPictureBoxClick function.

```
private void HandPictureBoxClick(object sender, EventArgs e) {
      clearSelection();
      PictureBox pictureBox = ((PictureBox)sender);
      …
```

Try running the code again, now there should only be one box selected. Another issue is because the selection is associated with the picture box, if we scroll through the cards a different card appears selected. To solve this problem easily, ensure that you call the clearSelection function at the beginning of both the buttonRight_click and buttonLeft_click functions.

12. Add clearSelection to the following functions:

```
private void buttonLeft_Click(object sender, EventArgs e) {
      clearSelection();
```

```
        …
    private void buttonRight_Click(object sender, EventArgs e) {
        clearSelection();
        …
```

Once the player selects the card we need to allow them to play the card onto the discard pile. They will do this by clicking on the discard pile. Double click on the discard pile to create a click event handler for the discard pile. Here we will check if the card that the user has selected is playable.

13. In the click event we need to ensure that it is the user's turn and that they have selected a card to play:

```
    private void discardPictureBox_Click(object sender, EventArgs e)
{
        if(turn==true && cardToPlay != null){
            …
```

In this code we use "`&&`" to represent a logical and, this means both of the conditions in the if statement, `turn==true` and `cardToPlay!=null` must be true for the code inside the if statement to execute. We will talk about Boolean Algebra in a later session and explain these logical operators in more detail.

The null value here simply indicates whether or not something is "empty", that is, if something is null that variable has not stored anything. In this case we are checking that it is not empty, that a user has made a selection.

To determine whether a card is playable or not, we simply need to compare colours and types. In Uno a card is playable if it is the same colour or the same type, while wild cards are generally playable at any point in the game. The only time wild cards are not playable is if a draw card (+4 or +2) is in play, as this forces the other user to draw cards unless they can play the same card on top.

Let's break this problem down into different conditional statements, starting with colours. We will need to get the colour of the card on the top of the discard pile and the colour of the card that the user is trying to play.

But how do we know what colour the card is? Now that it is evident colour is important, the easiest way to compare this will be to keep track of the colour inside the Card class. This is one of the reasons objects are so useful, as we can change the class and then will have that property for every object.

14. In the card class add the attribute colour and modify the constructor as below:

```
…
public string colour;
…
public Card(string file, string c){
      card = Image.FromFile(file);
      file_name = file;
      colour = c;
}
…
```

However, when we modify a class constructor, we will have created errors in our form class. This is because we are still creating card objects without the colour stored. The good news is that we have a function which adds cards to the deck, so we only need to change this declaration in the addCard function.

15. Modify the addCard function in the Form class:

```
     private void addCard(string dir, string colour, string type,
int amount){
           Card new_card = new Card(dir + colour + "_" + type +
           "_large.png", colour);
           …
```

Now we can simply compare the colour strings to see if a card is playable by its colour.

16. Modify the discardPictureBox_Click event handler as below:

```
     discardPictureBox_Click(object sender, EventArgs e) {
           if(turn==true && cardToPlay != null) {
                 Card c = discard[0];
                 if(c.colour == cardToPlay.colour){
                       hand.Remove(cardToPlay);
                       discard.Insert(0, cardToPlay);
                       displayDiscard();
                       displayPlayer();
     }
                 else {
                       MessageBox.Show("That card is not playable.");
                 }
```

```
            }
      }
```

In the above code we get the top card of the discard pile and then compare the colour of this card to the colour of the card to play. If they are the same then we remove the card from the player's hand and add it to the top of the discard pile. Next we refresh the display to the user by calling the displayDiscard and displayPlayer functions. If the card is unplayable we display a message box to the user to inform them that they cannot play that card.

Next, let's consider comparing by types. We will need to follow very similar actions to the above by adding a "type" attribute in the card class.

17. In the card class add the attribute type and modify the constructor as below:

```
      …
      public string colour;
      public string type;
      …
      public Card(string file, string c, string t){
            card = Image.FromFile(file);
            file_name = file;
            colour = c;
            type = t;
      }
      …
```

18. Modify the addCard function in the Form class:

```
      private void addCard(string dir, string colour, string type, int
amount){
            Card new_card = new Card(dir + colour + "_" + type +
"_large.png", colour, type);
            …
```

19. Modify the discardPictureBox_Click event handler as below:

```
      discardPictureBox_Click(object sender, EventArgs e) {
            if(turn==true && cardToPlay != null) {
                  Card c = discard[0];
                  if(c.colour == cardToPlay.colour || c.type ==
cardToPlay.type){
                        hand.Remove(cardToPlay);
                        discard.Insert(0, cardToPlay);
```

```
                displayDiscard();
                displayPlayer();
            else {
                MessageBox.Show("That card is not playable.");
            }
        }
    }
```

The "||" represents the logical or operator, this if statement is stating if the colours match OR the types match, then we can play the card, that is, provided one of the conditions is true then execute the code.

Note that at this point we haven't programmed the logic for the draw 2s and 4s or for the wild cards. In addition, we haven't programmed the AI player or how the game will take turns. We leave this for the advanced exercises section.

Summary

In this session we covered the benefits of using event-driven programming and demonstrated how to do this for a GUI program in C#. However, event-driven programming is used widely across different programming languages and not just for GUI programs. For example, when we send a request to a server we have to wait for that server to respond with an answer (the event) and write a function which handles the response correctly (a callback function). This is often done across different programming languages depending on the technologies used to interact with the server. For instance, if handled by a website we may use HTML, CSS, Javascript, PHP and SQL to interact with a database and gather some information that a user has requested and then display that to the user.

Advanced Exercises

Want to finish off the functionality for the Uno card game? Here are the next steps:

- Implement the functionality for the rest of the cards for the player (Draw 2s, Draw 4s, Wild cards, Skip, Reverse). You will need to determine how to allow the user to pick a colour when they play a draw 4 or wild card. How do you ensure that the skip or reverse works?
- Next, enforce ordering on the player's turn, ensure that if they draw a card that they cannot play a card and vice versa. *(Hint: you will need to "pause" the event clicks, an easy way to do this is to remove the click event)*.
- In Uno a player can play multiple cards of the same colour on one turn. How can you let the player select multiple cards to play?

- Now you will have the functionality for the player, you need to give them someone to play against. The AI already has a hand but you will need to determine how to implement this i.e. how will your AI play the game? You can make your AI "smart" depending on how you code their game strategy.
- It's not particularly entertaining to play against one player, can you add multiple AIs? How does this change some of the game play?

Useful Resources

- C# documentation How to handle an event: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/controls/how-to-add-an-event-handler?view=netdesktop-5.0
- C# documentation Tutorial 3: Create a matching game: https://docs.microsoft.com/en-us/visualstudio/ide/tutorial-3-create-a-matching-game?view=vs-2019
- C# Programming Beginner Tutorial: Rock! Paper! Scissors!: https://steemit.com/csharp-forbeginners/@dbzfan4awhile/c-sharp-programming-beginner-tutorial-rock-paper-scissors
- Best strategies to Win Uno: https://www.unorules.com/best-strategies-to-win-uno/