

Further introduction to Small Basic introducing the development of a graphical user interface.

Preparation required:

Small Basic available on all computers, all sample programs available in a shared repository.

Graphic Shuffle Investigation sheet per person.

Optional: Investigating A Perfect Shuffle sheet and cards per person.

A Graphical User Interface

The final coding challenge returns to our shuffling theme and introduces a GUI (pronounced gooey). Learning the particulars of any language carries a significant overhead. There is scope for debate about whether a school should focus on 1 language or provide exposure to many.

The purpose of these materials is to suggest possible environments, but inevitably we can't, and shouldn't provide a crash course in any. Once pupils gain some familiarity with Small Basic you can encourage them to investigate the Microsoft Small Basic curriculum. This is a series of 1 hour lessons introducing different features. Free to download from goo.gl/BcVI5T they are very useful for introducing specific functionality, such as the graphics window (lesson 2.1). To give a sense of how powerful Small Basic is, we'll take our shuffle routine and produce a graphical user interface (GUI) for the program.



It is easier to start with reference to an existing program. Studying and amending code is another effective learning technique and allows us to focus on the structure of the code. The code for this is provided GraphicShuffle.sb. The challenges also need the card images included in the program resources. The investigation can be delivered as a taught activity or independently. An activity sheet is included to support the latter.

The slides allow the code to be introduced. Creating a graphics window is straightforward. Note the use of the FilePath variable, which allows the code to create a string pointing to a subdirectory of card images (included in the resources). Line 43 demonstrates how the FilePath variable is concatenated with the filename of the card back image.


Make sure you are clear how the 52 card backs are displayed in a row. Can you find the code to display the other graphical elements? It is in the Clear subroutine, so the canvas is redrawn each time that is called.

Currently the code doesn't work. Notice the buttons added at the end of Clear. They are examples of control objects. These allow a GUI to be controlled via button clicks and other common features. Ours don't work because the relevant code on Line 27 is commented out. When it is activated, a button click will 'raise an event' causing the Click subroutine to be called.

It is worth emphasising how this type of program leads towards a different programming 'paradigm'. Event driven programs sit in a main loop, displaying a set of objects waiting for an event to be raised. Code routines are then attached to the event handlers. Thinking of problems in this way can lead to a very different way of developing software. For now though, once working, a series of challenges are posed.

Algorithms and Data Structures
Investigating A Card Shuffle App

You are provided with a Card Shuffler program. Even if you aren't familiar with coding a GUI, studying this program carefully, and amending it should make it clear how they work. Open the program Graphic_Shuffle_investigation.sb. Make sure you have the folder Cards folder in the same directory as the program. Run the program – it should look like the window below, but nothing works!



Creating a graphics window, like that shown is straightforward. Can you see it in your code? It is shown right (lines 7 – 11). Note also the use of a FilePath variable (line 19), which will allow you later in the program to create a string pointing to the subdirectory of the card images. Line 43 (shown left) demonstrates how the FilePath variable is concatenated (joined) with the filename of the card back image (53.png) in the string CardBack.

Explain how 52 card back images are displayed in a row.

Which subroutine contains the commands to create the other graphic elements (titles, buttons etc.)?

The buttons are examples of control objects. These allow a GUI to be controlled via button clicks and other common features. Ours don't work, because the relevant code on line 27 is commented out. When it is activated, a button click will 'raise an event' causing the Click subroutine to be called.

Explain how the Click subroutine works.

Challenge 1 asks to refactor the code, so it displays the original Deck in order. Once the buttons are activated we are able to display the cards in shuffled order but not their initial state. This is a simple task that involves recognising that Shuffle is called in the main program, not just on a button click. Commenting this out, or removing it will solve the problem.

Challenge 2 encourages further study of the code, by tidying up the shuffle subroutine. It currently implements a Knuth shuffle. This should be replaced with their own shuffle algorithm, developed earlier. There is no swap subroutine here, so they will have to include that as well, if they used one. The shuffle subroutine also includes the code to display the cards face down. This has nothing to do with shuffling. It is a common mistake and needs emphasising. Subroutines should be developed for separate tasks, so should be factored out into its own subroutine.

Challenge 3 asks to sort the Deck back into order? Implementing any sorting algorithm is well beyond most KS3 pupils, but it is included as a challenge for teachers or able students. Encapsulate this in a subroutine **CardSort**. A simpler way to produce the cards in order is by recreating the array. Simple refactoring, calling ManyIntegers will achieve this. Whichever approach is adopted, the real challenge is then to work out how to create and display a fourth button which calls CardSort when it is clicked.

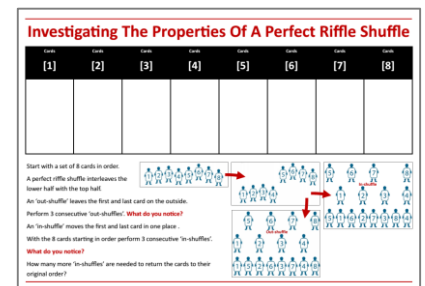
Challenge 4 is difficult. We represent a deck of cards by the numbers 1 to 52. Write the code to allow the input of a **CardNumber**. The program, should print the description of the card e.g. "King of Hearts". Encapsulate this in a subroutine **CardName**. Incorporating this in the GUI could be an extra extension for very able students. A slide provides a hint and teaching aid. The numbers of the card image files are listed; 1 to 13 represent Hearts, 14-26 Diamonds and so on. So cards 1, 14, 27 and 40 are all two's of a particular suit. Using floor division and remainders, think of a way of extracting the suit and value from the number.



As previously mentioned, the Small Basic Introductory Guide is very good. Chapter 10 focuses on arrays. The later examples introduce a two dimensional array to represent a grid in a graphics window. Finally, it uses a counter controlled loop to introduce some very effective animation. As a homework for teachers it is well worth working through and could provide the basis for several coding challenges. How about creating a chess board, for example?

A Perfect Riffle Shuffle

Riffle shuffles interleave the top and bottom halves of a deck. A final activity is designed to reveal a surprising insight. It is also used in the introduction to Tenderfoot Unit 2: Clever Stuff For Common Problems, so use here is optional, if time allows. It is best tackled as a hands on activity using 8 cards and the investigation template. An explanation is required of an out-shuffle, which leaves the top and bottom cards in place. Using 8 cards, starting with them in order, perform 3 out-shuffles. What do you notice? The cards return to their original positions.



An in-shuffle, interleaves the cards but moves the top and bottom cards 'in'. Starting with the cards in order, try 3 consecutive in-shuffles. The order hasn't returned after 3 shuffles (it has reversed), so encourage further investigation. It should take another 3 in-shuffles to reinstate the original order.

A final challenge might be to develop each in and out-shuffle as a separate subroutine. The GUI Shuffle application could be refactored to accommodate them and a series of investigations embarked on. How many consecutive in or out shuffles are now needed to return the deck to its original state? As well as returning to the original order, experiments could be undertaken to see how to control the position of the top card. What sequence of moves would be required to move it down the pack to the 2nd, 3rd, 4th or any other position? Such perfect shuffles lie at the heart of some magician's routines for manipulating cards. This is an activity we will return to in Tenderfoot Unit2.