VBA Flash Card System

Britton Kowalk

Executive Summary

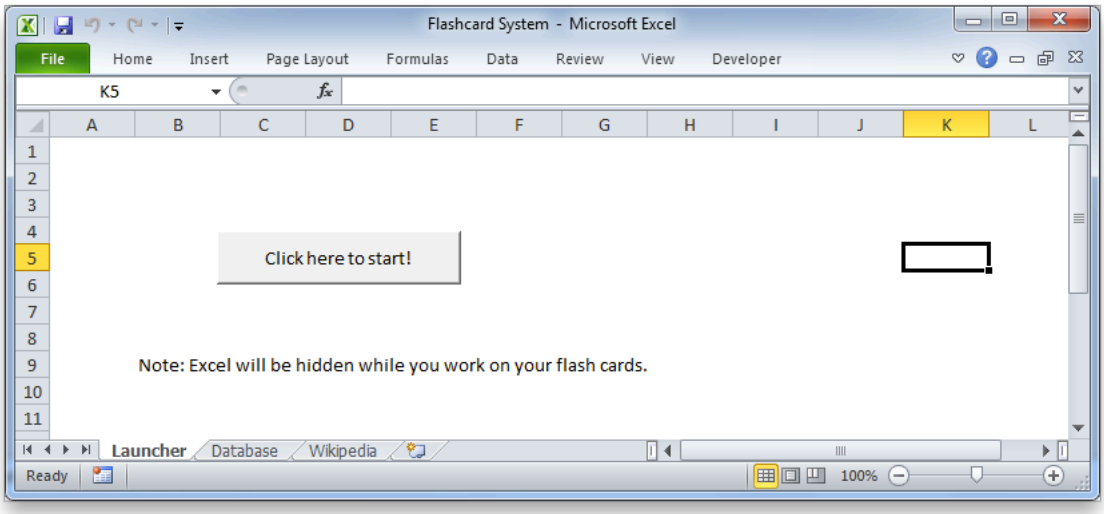
In my Business Law class this semester I've been amazed at the many nuances and intricacies of the legal system. So amazed, in fact, that I've started getting nervous about the final test. I've realized that it's going to be a bit of a challenge without some help. To prepare, then, I decided to build a flash card system that I can use to study.

Similar flash card systems have been developed by students in the past (see http://vbaprojects.blogspot.com/2010/12/flash-card-machine-brian-baugh.html for one example), but these projects lacked a few functions that I really hoped for like a true GUI interface, a simple way to browse and edit cards, and ways to hide cards that have been amply learned.

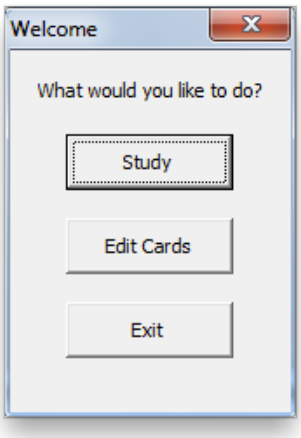
To fill this need, I have designed a fully GUI-based flashcard system. The management side of the application allows adding, removing, and editing cards. The study side features simple controls to move forward and backward through cards, to mark cards as learned (which hides them from study), to reset all cards that have been hidden, and to display hints the user has input. As an additional challenge, I added the ability to display brief snippets retrieved from Wikipedia on any given term.

Implementation

The application is launched by opening its excel document:



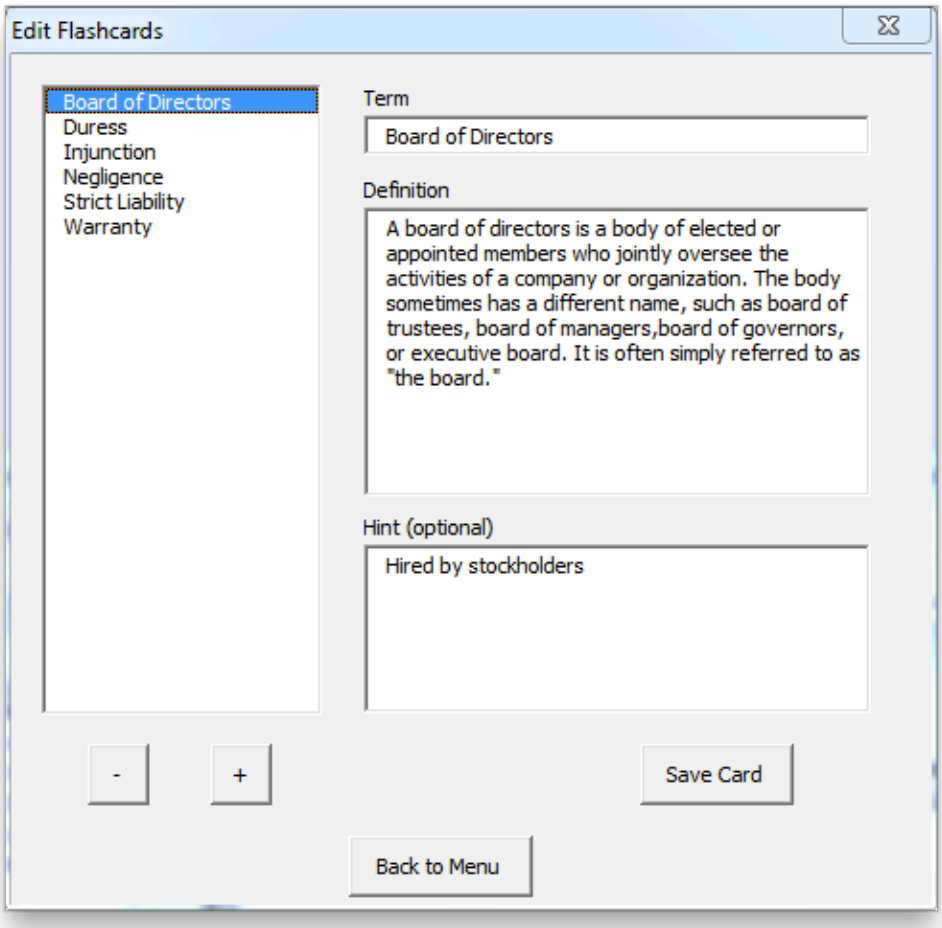
A second tab ("Database") can be accessed if a user feels the need to input data directly into the app's back end database. Otherwise, clicking the start button launches the welcome screen:



After this point, the programming logic is of greater interest. We'll now focus on that. The implementation of the flash card system can be broken down into two sections: card management and study.

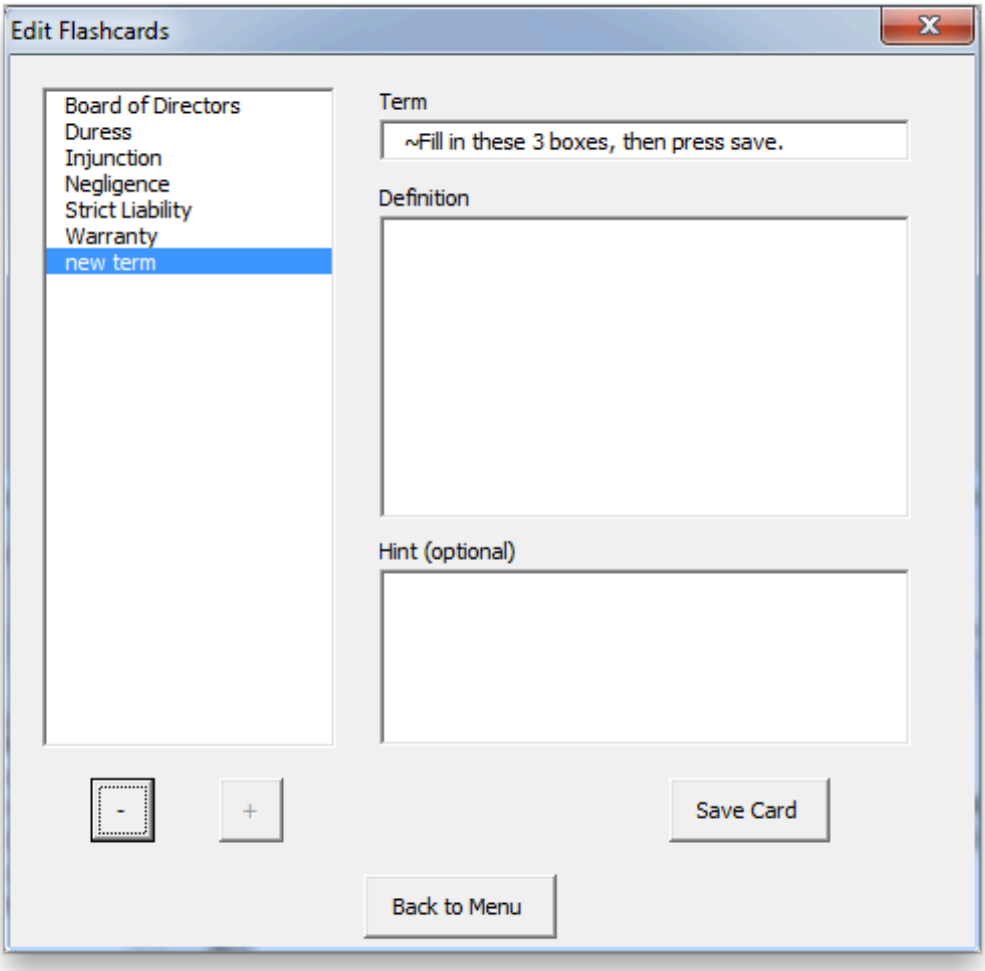
Card Management

If a user clicks "Edit Cards," they are presented with the following interface:



The interface's list of cards is populated on launch by scanning through the (now hidden) database spreadsheet. Edit fields (on the right) are populated whenever a user selects a term to view. Any changes made are written out to the database spreadsheet when the "Save Card" button is pressed—all fields are read and written to the row on the spreadsheet that corresponds with the selected row in the term list on the left.

Pressing the add button creates a new row in the database, displays it in the list on the left, then allows the user to input information and save the card. The add card button, further, is disabled until the user finishes inputting data, thus preventing the accidental creation of multiple, blank cards.:



Removing a card acts similarly—the row is removed from the database, then removed from the list on the left. While simple, the interface is very robust and stops the user from several different errors. Note how in the save code below, the system ensures that a proper card is selected before saving, then ensures that the user has changed the default text after creating a card.

```
Private Sub BtnSave_Click()
    If ListCards.ListIndex <> -1 Then
        If BoxTerm.Text = "~Fill in these 3 boxes, then press save." Then
            MsgBox ("You haven't entered any information for this new card yet.")
        Else
            Cells(ListCards.ListIndex + 2, 1).Value = BoxTerm.Text
            Cells(ListCards.ListIndex + 2, 2).Value = BoxDefinition.Text
            Cells(ListCards.ListIndex + 2, 3).Value = BoxHint.Text
            RefreshData
            BtnAdd.Enabled = True
            MsgBox ("Card saved.")
        End If
    Else
        MsgBox ("Please create a new card or select a card to edit.")
    End If
End Sub
```

To ensure that the interface always shows accurate data and shows the cards in an easily searchable, alphabetical order, a visual refresh subroutine was designed. After every change, this subroutine runs and (1) alphabetizes the database followed by (2) refreshing the data displayed on the GUI.

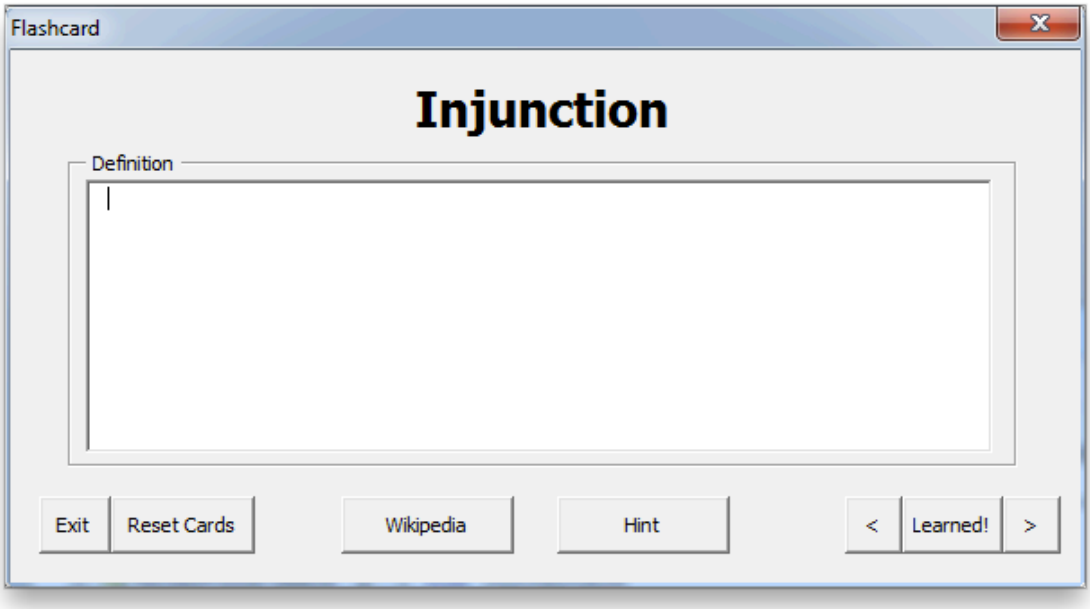
```
Private Sub RefreshData()
    ListCards.Clear
    Range("A2:D3000").Select
    ActiveWorkbook.Worksheets("Database").Sort.SortFields.Clear
    ActiveWorkbook.Worksheets("Database").Sort.SortFields.Add Key:=Range("A2:A3000") _
        , SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    With ActiveWorkbook.Worksheets("Database").Sort
        .SetRange Range("A1:D3000")
        .Header = xlYes
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With
    Range("a1").Select

    For rowCounter = 2 To Range("a3000").End(xlUp).Row
        ListCards.AddItem (Cells(rowCounter, 1).Value)
    Next
End Sub
```

The end result is a simple, robust interface for creating and editing large sets of cards.

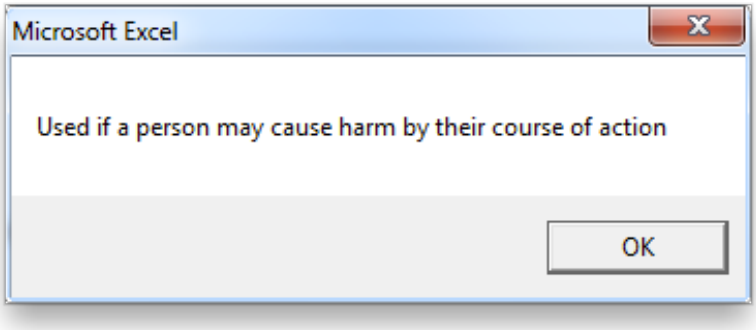
Study

Entering the study portion of the interface brings the user to the following screen:

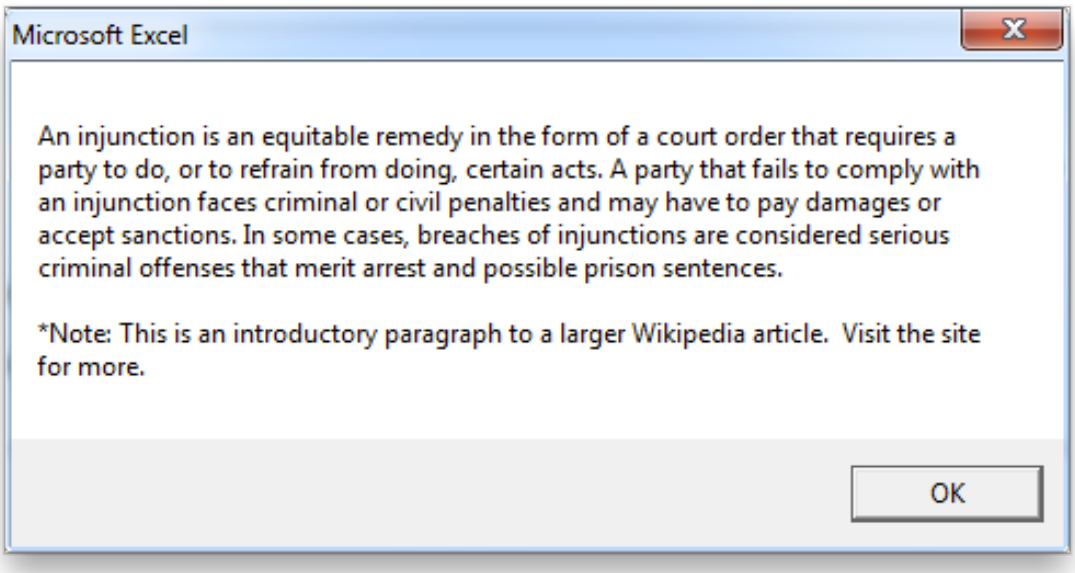


Upon launch, a random card is selected and its term displayed (the card selection function will be described below). The user can then use the controls on the bottom right to move forward (first revealing the definition, then moving to the next card selected by the card selection function), mark the card as learned (which will write a value to the "hide" column and prevent the next card function from selecting the card until the "reset cards" button on the left is pressed) or go back to the previous term studied.

In the event that the user needs a hint, they can simply click on the "hint" button. A message box is displayed with the hint the user attached to the card, which is read from the database spreadsheet:

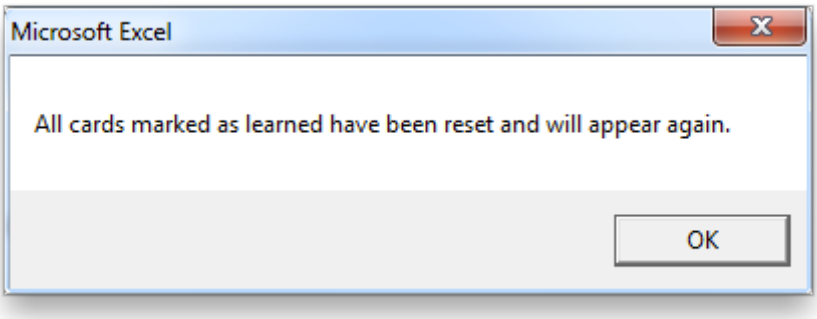


If the user wants further information on the given term, they can click the Wikipedia button to retrieve a paragraph from the introduction to the term's Wikipedia article:

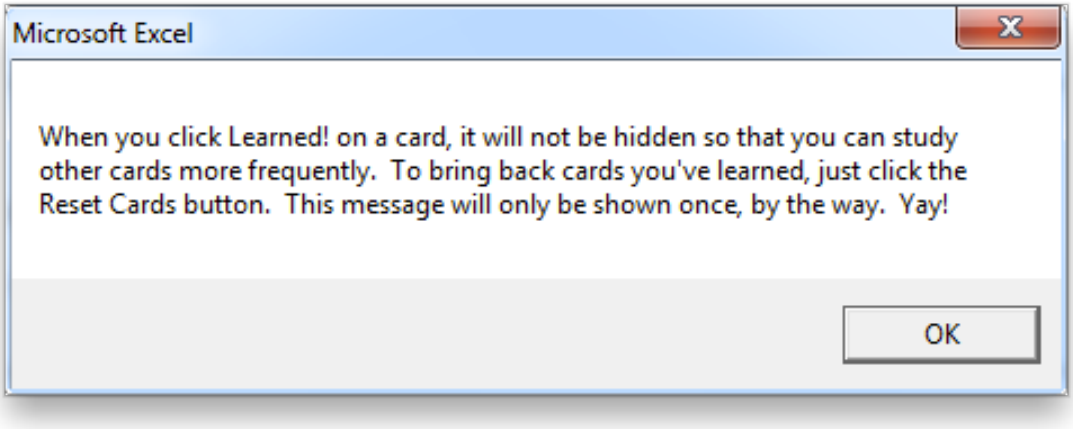


This term is gathered by taking the given term, transforming it to lowercase, changing spaces into underscores, and attempting to retrieve a Wikipedia article from the page http://en.wikipedia.org/wiki/[term_name]/. If the page is found, the system then seeks to find the table of contents on the page. The paragraph directly above the table of contents is retrieved as an introduction. Due to wide variation between Wikipedia pages, this paragraph before the contents section was found to be the most reliable method of retrieval.

In all instances of interaction with the flash card window, confirmations and instructive dialogs help the user to understand the interface better. For example, resetting the cards reveals the following dialog while the application clears the "hidden" marker from every card row in the database:



Terms that yield no Wikipedia article will display a message of "Sorry—no Wikipedia article found for this term". Terms with no hint will remind the user that they forgot to input a hint. Even the learned button helps the user understand its function after pressing it:



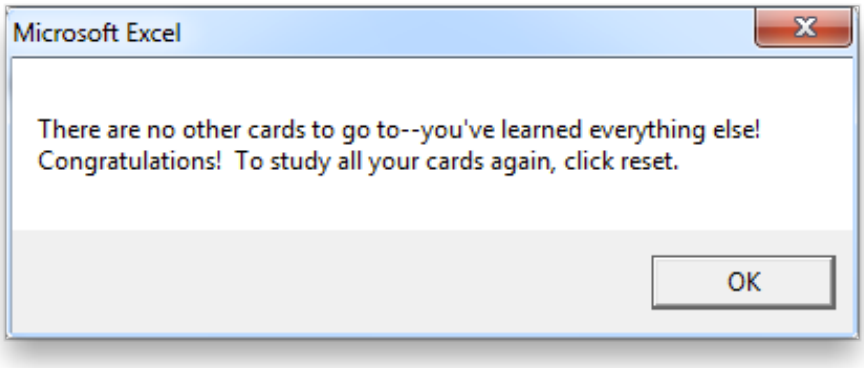
This message is only displayed once in order to streamline the study process.

The flash card section presented many challenges in coding. One of these (which continually presented problems) was the subroutine selecting the next card. After many iterations, the final code checks the number of rows in the database and selects a random row, checks if the row has a marker in the "hide" column (filled in when a user marks a card as learned), and checks if the potential next row is the same as the last row displayed. So long as neither of these throw red flags, the routine sets the new row and allows it to be displayed. If either causes problems, the system tries another row.

```
Do While foundTerm = False And tries < 1000
    possibleRow = Rnd() * (finalTermRow - 2) + 2

    If Cells(possibleRow, 4).Value = "" Then
        If possibleRow <> previousRow Then
            foundTerm = True
        End If
    End If
    tries = tries + 1
Loop
```

What if, however, the user has clicked "learned" on all of their cards? The system will catch this when it reaches its 1000th try to find a new row, realizing then that the deck is complete. The user is then given the following instructions:



Discussion of Learnings and Challenges

I've been programming for some time, so much of the back end code of this project was straightforward. Having not spent much time in VBA user forms, however, the greatest challenge I encountered was in creating a robust interface. Over and over, I found ways that I could break my system or bring about results that weren't desirable. I spent a lot of time looking into logic that would enable and disable buttons, check input fields, and ensure accurate reflections of information from the database. In short, my big lesson from this project was the importance of testing.

For example, I discovered that it would be simple for a user to create many unwanted rows in the card database that, once the system alphabetizes them back into order, would be hard to find in a large card set. To prevent users from creating lots of new cards without inputting their information, the add card button is now disabled until a new card is properly input and saved.

As one more example, I first programmed my Wikipedia functionality based on the structure of just a few pages that I had come across. I found patterns in them and used these to figure out where the introductory paragraphs that should be pulled in were found. However, I found over time that massive errors could occur in this method—the wide variation of Wikipedia pages made it necessary to pull input from only one paragraph rather than all intro paragraph (the best delimiters Wikipedia provides at present specify the beginning and end of the body of the article, but nothing about the structure beneath that is built by the wiki community). I determined that this functionality, even if imperfect, would still be quite useful in many instances, and completed and included it.