Matt Blodgett VBA Project (Nibbles) Write-up

Contents

Executive Summary

For my project, I decided to address the significant lack of Excel games for my young son to play. I created an excel version of the game 'Nibbles' or 'Snake'.

Gameplay

In this classic game, the user controls a snake on a game board made entirely of squares. The head of the snake occupies one space on the square, with a long tail trailing behind it. The goal of the game is for the snake to eat 'apples', or red squares with a number embedded in it. Once a certain number of apples (9 in my version of the game) have been eaten, the level is complete and the snake moves on to another level. If the snake runs into its own body or the walls of the level, the level must be restarted. After the snake runs out of lives, the game is over. Scoring is awarded based on the number of the apple being eaten, a multiplier defined for each level, and a reward for the speed of the snake.

Creating user-defined levels

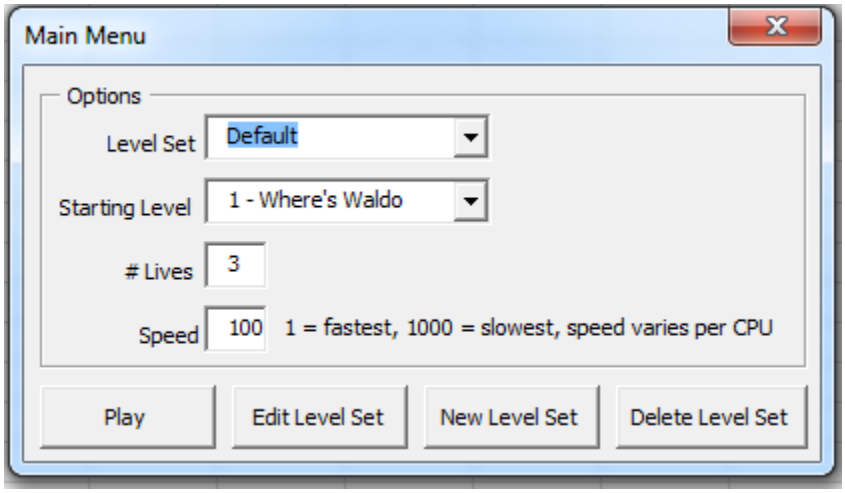
In order to keep the game fresh, new and entertaining, I added the ability of the user to create new levels to play. Users can create any number of level sets, with any number of levels. Each level can be customized in size, starting position, scoring options, and with user drawn walls.

Implementation Documentation

User Manual

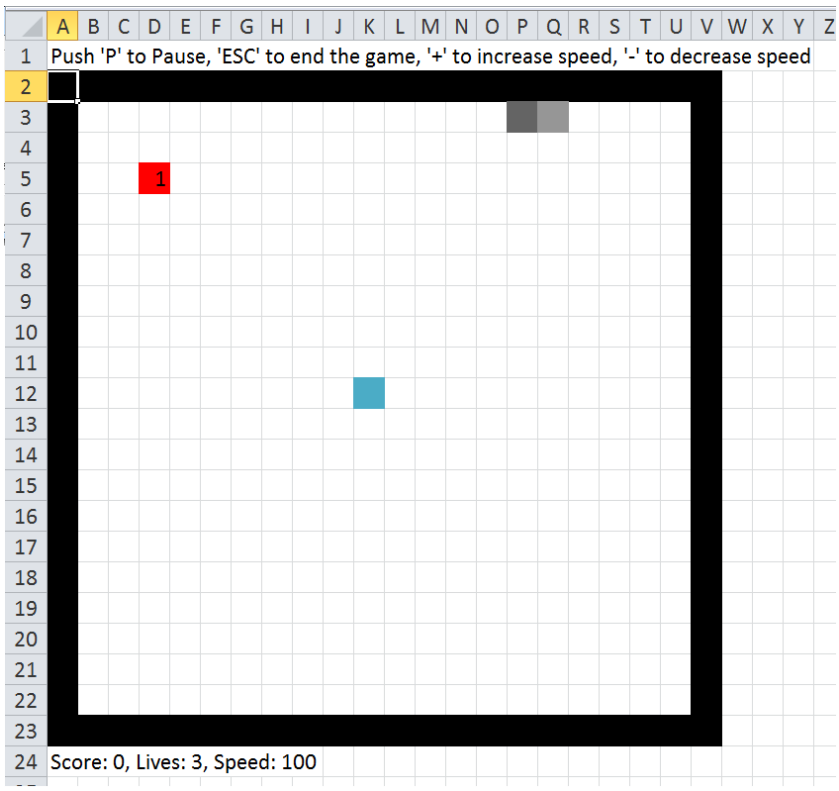
How to play

To play Nibbles (snake), simply click on the 'Play Nibbles' button. This will bring up the main menu:



Choose a level set, the starting level, enter the number of lives you with to start with and the starting speed of the game. Note that speed varies between computers, so you may have to try it out to get used to it. You can also adjust the speed while playing.

To play, click the 'play' button. This will create the level and prompt you to start. The playing field will look different for each level, but will look something like this:

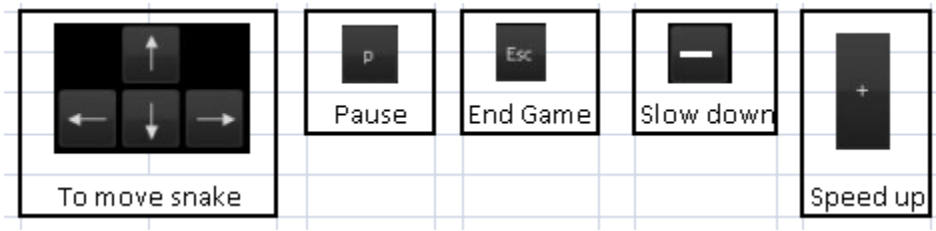


The black outline represents the boundaries of the game, and each square is one spot for the snake to travel in. The light gray spot is the head of the snake, and the body of the snake trails behind it in dark gray.

User defined walls are shown inside the box in various colors. If the head of the snake hits any of the walls, or the body of the snake itself, you lose a life, and must restart the current level.

The objective of the game is to collect the 'apples' (red squares with numbers) on each level. The numbers appear consecutively 1 - 9. When you hit the number 9, the level is immediately terminated, and game play moves to the next level.

To play, use the arrow keys to change the direction of the snake. You can make the snake move forward more quickly by holding down the arrow key in the current snake direction.



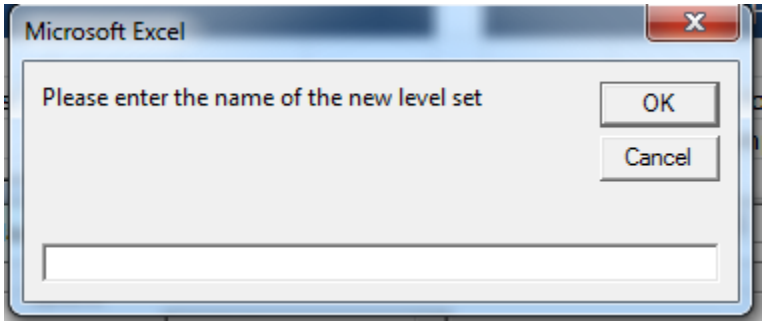
To pause play, push 'P'. To end play, push 'ESC'. Below the playing field you can see your score, the number of lives remaining, and the current speed of the game. To increase speed, push the '+' button on your keypad. Conversely, push the '-' button on the keypad to decrease speed.

Points are awarded based on a multiplier for each level, the number that the snake is 'eating', and a multiplier based on the speed of the game. The formula is as follows:

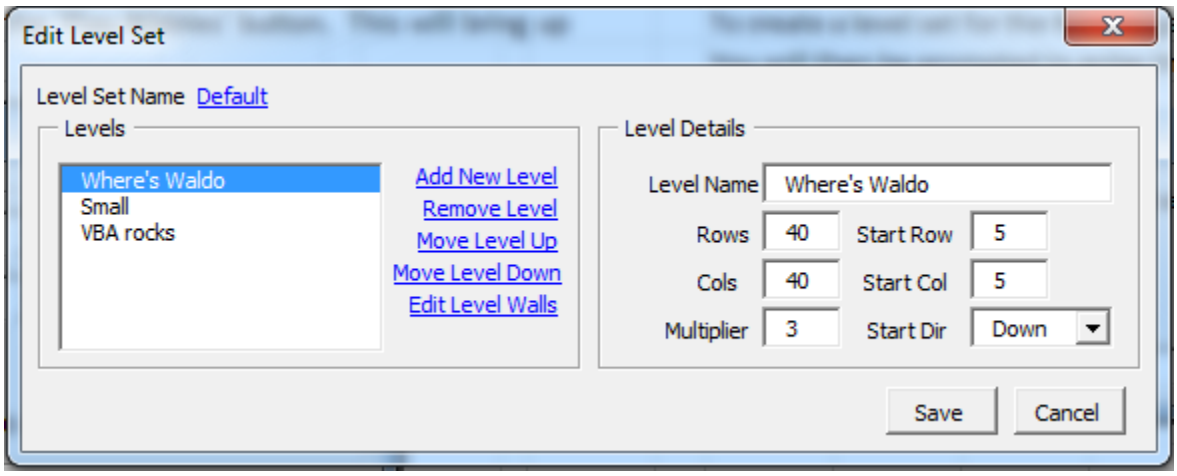
Score = Score + (Multiplier * NumberEaten * max(200 – Speed, 1))

Editing levels

To create a level set for the Nibbles game, click 'New Level Set' on the main menu You will then be prompted to enter the name of the new level set:



After entering the new name, you will then be taken to the edit screen. To delete an existing level set, simply choose the level set and hit 'Delete Level Set' to edit an existing level set, simply choose that level set, and hit 'Edit Level Set'.



To edit a level, choose the level from the levels box on the left.

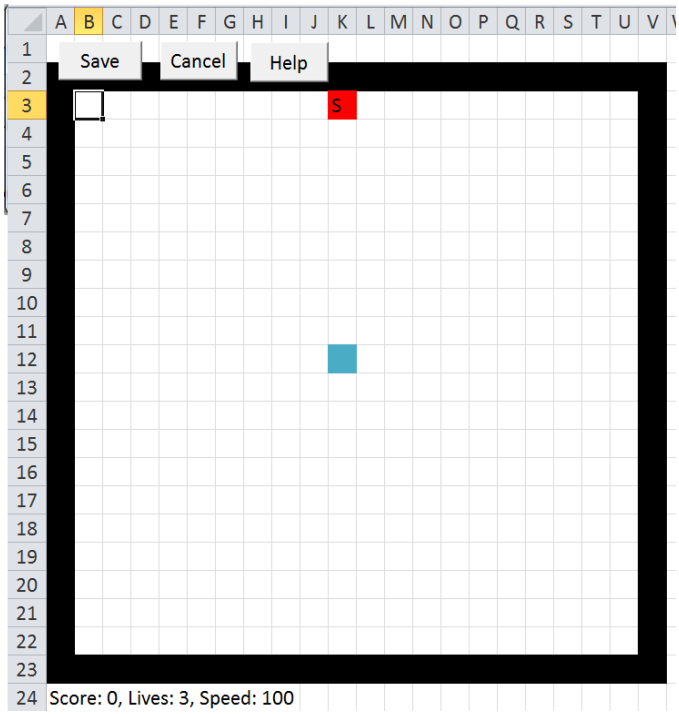
You can then edit the options for that level - Name, size, Score multiplier, Starting position and starting direction.

Use the hyperlink buttons to add new levels, remove levels, and move levels up and down.

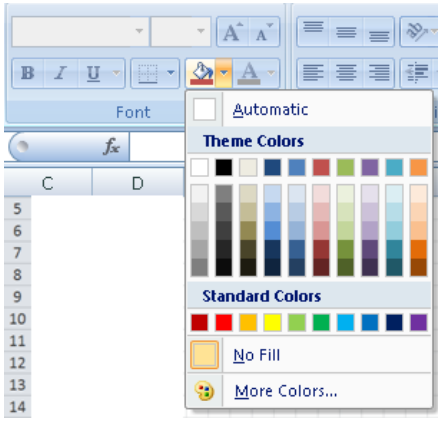
If you want to rename a level set, click the hyperlinked name of the level set.

To draw or change the walls on a level, click the 'Edit Level Walls' hyperlink.

This will take you to the edit level screen:



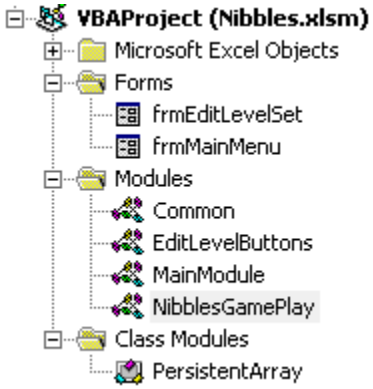
This is a normal Excel workbook with protected cells. Only the fill coloring will be saved. Use the normal Excel controls to draw and color walls:



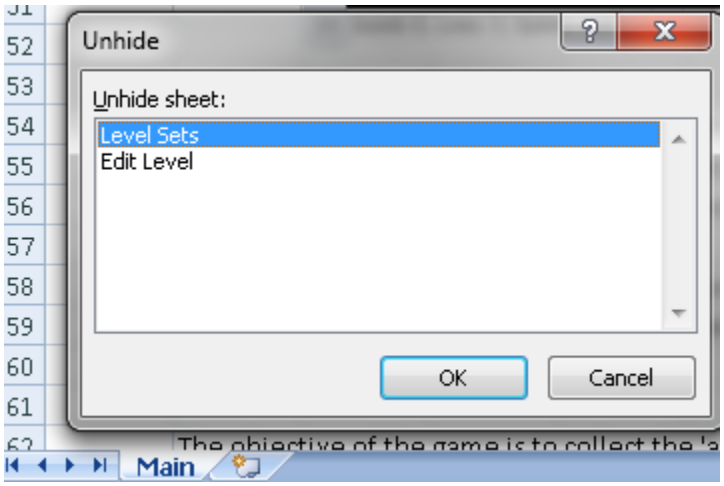
After you are done, hit 'Save'. If you want to go back without saving changes, hit cancel. For help, click 'Help'.

Note that the starting position is labeled 'S' and colored red. Any coloring on this cell will be ignored when the level is saved.

Technical Description of components

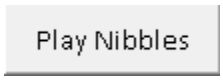


The VBA project consists of two forms, four modules, and one class module. There are also three Worksheets in the Workbook, with two of them hidden:



Main Worksheet

The first worksheet, “Main”, contains the instructions and the button to bring up the main menu. The “MainModule” module contains only a single method for this “Play Nibbles” button:




```
Sub PlayNibbles()
    frmMainMenu.Show
End Sub
```

Level Sets Worksheet

The second worksheet, “Level Sets” contains the saved data for each level:

	A	B	C	D	E	F	G	H	I	J	K
1	Level Set Name	Level #	Rows	Cols	StartX	StartY	StartXDir	StartYDir	Multiplier	Walls	LevelName
2	Default	1	20	20	10	10	0	-1	2		Small
3	Default	2	40	80	5	5	0	1	1	6,13,5066	VBA rocks
4	Default	3	40	40	5	5	0	1	3	2,6,255|3	Where's W
5	One Level Only	1	40	40	5	3	0	1	1	8,8,50665	Level A

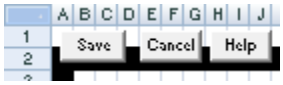
The ‘Walls’ data is a string with the following format, with each colored cell represented:

Row,Col,Color|Row,Col,Color|….|Row,Col,Color

This worksheet has no direct code associated with it, but is used as a database to store the level data.

Edit Level Worksheet

The third worksheet, “Edit Level” is the worksheet that is brought up when the ‘Edit Level Walls’ button is clicked. It has three buttons, with the coloring and rows and cells set in the code for the different levels. The module “Edit Level Buttons” contains the code for these three buttons.




```
Option Explicit

Public Sub SaveLevelWalls()
    'Tell the form to save the walls
    frmEditLevelSet.SaveWalls

    'Hide the sheet
    Sheets("Edit Level").Visible = False

    'Show the form
    frmEditLevelSet.Show
End Sub

Public Sub CancelLevelWalls()
    'Hide the sheet and show the form again
    Sheets("Edit Level").Visible = False
    frmEditLevelSet.Show
End Sub

Public Sub Help()
    MsgBox _
        "Instructions on drawing walls for e
        "
```

Common module

This module contains several code subroutines and functions that are used during both game play and on the forms to edit the levels. The methods are listed below with descriptoins:

```
Function GetLevelSetVariable(VarName As String, LevelSetName As String, LevelNum As
Integer)
```

This function will return a cell value in the Level Sets worksheet. For the particular level set, level number, and variable name (column in the spreadsheet).

```
Function GetLevelSetNumberOfLevels(LevelSetName As String)
```

This function counts the number of levels in the Level Sets worksheet for a given level set.

```
Sub SortLevels()
```

This function sorts the levels in the Level Sets worksheet to ensure that level sets are on contiguous rows and in order by level number

```
Sub CreateLevel(LevelSetName As String, _
    LevelNum As Integer, _
    levelName As String, _
    Optional CreateNewWorksheet As Boolean = True, _
    Optional Rows As Integer = -1, _
    Optional Cols As Integer = -1, _
    Optional Walls As String = "NONE", _
    Optional ShowMessage As Boolean = True)
```

This function can create a new worksheet for a level, size the rows and column appropriately, draw the walls for a given level, and zoom the selection to the playing field. It can also do this for an existing worksheet (the Edit Level worksheet).

```
Sub DeleteLevelSet(LevelSetName As String)
```

This function deletes all rows in the Level Sets worksheet for a given Level Set name.

```
Function CheckTextNumeric(Text As String, Min As Integer,Optional Max As Integer = -1)
```

This function is used to make sure that valid values are entered into text boxes on the user forms

NibblesGamePlay module

This module has all of the methods for the actual game play. They are listed below with descriptions:

```
Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal nVirtKey As Long) As Integer
Const KeyPressed As Integer = -32767
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

These functions and constant are used to interact with the windows API and allow the user to interact during the game.

```
Sub IncrementNumberToFind(CurNumber As Integer, CurNumberX As Integer, _
    CurNumberY As Integer, Rows As Integer, Cols As Integer, Score As Integer, _
    Multiplier As Integer, Speed As Integer)
```

This function increments the number to find, chooses a new random location, erases the old number, and draws the new one.

```
Function PrintSnake(snake) As String
```

This is a debugging function to print out the array holding the snake positions.

```
Function MoveSnake(snake, xDir As Integer, yDir As Integer) As String
```

This function moves the snake in memory (switching positions in the array), and then erases the tail and draws the head on the actual gameboard.

```
Sub GrowSnake(snake, CurNumber As Integer, Multiplier As Integer)
```

This function grows the snake by the given amount (resizes the array and fills the new spots in the array with the tail value)

```
Public Sub UpdateScoreAndLives(Rows As Integer, Score As Integer, Lives As Integer,
    Speed As Integer)
```

This function updates the score on the gameboard.

```
Public Sub PlayGame(StartLives As Integer, Speed As Integer, LevelSetName As String,
    StartingLevel As Integer)
```

This function is the actual gameplay. It executes until the game is over, and checks for user input using the Win32 API. It then 'pauses' a given number of milliseconds before moving the snake to the next spot, checking to see if it was a valid move, etc.

PersistentArray class module

This class was created to persist an array in a form while it was hidden. It has only one variable (the array) and subroutines and functions to get and set and resize the array. It will be discussed in greater detail in the learning points.

Discussion of learning points

Split to store Wall data

One of the most useful functions that I had to learn about during this project was the *Split* function in VBA that takes a string, and splits it on a delimeter, creating an array. I used this twice on the Walls parameter that I save for each level:

```
'Parse the walls and draw them
'Our format is Row,Col,Color|Row,Col,Color|Row,Col,Color
'Row,Col of 1, 1 mean the topleft corner that they could draw in (B3 on our spreadsheet)
Dim WallArray, WallArrayItem, ItemArray
WallArray = Split(Walls, "|")

    For Each WallArrayItem In WallArray
        ItemArray = Split(WallArrayItem, ",")
        newSheet.Cells(ItemArray(0) + 2, ItemArray(1) + 1).Interior.Color = ItemArray(2)
    Next
```

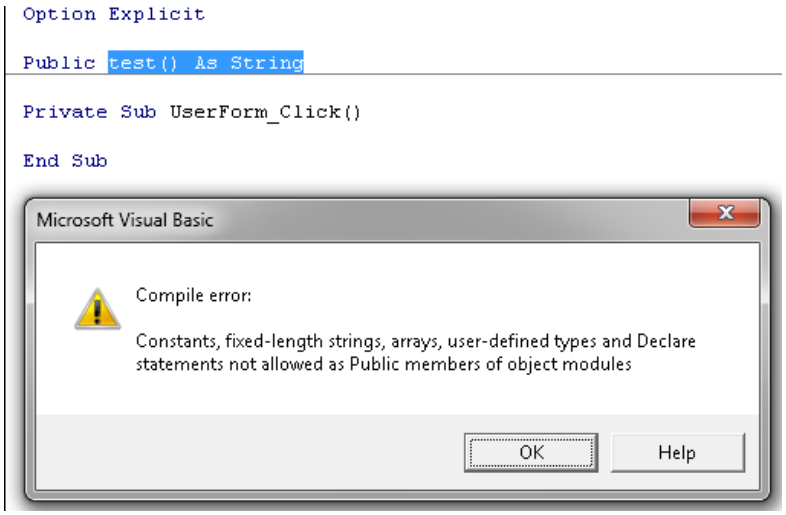
Calls to Win32 API

Perhaps the biggest challenge I faced was how to allow the user to interact with the game while the subroutine is still running. Initially, I thought I would use the Application.OnTime function in Excel, which would call the same routine at a point in the future. This would have worked, but the OnTime function will only execute one second later. I needed to 'pause' for less than a second.

After doing some research on the internet, I found that VBA can call external functions in the windows 32 API. I therefore use the Win32 'Sleep' function to 'pause' execution for several milliseconds. I also use the 'GetAsyncKeyState' Win32 function to check for keystroke input from the user. I also discovered the DoEvents function in Excel which will let the user interact with Excel during execution of a subroutine.

Persisting array when forms are hidden

One of the most interesting challenges I had was in trying to have the 'Edit Level Set' form be done completely in memory so that the user could cancel. Everything was fine because I loaded the level information into an array in memory. However, when I hid the form to show the 'Edit Level' worksheet and allow the user to interact and draw the walls, the array variable was being cleared. I tried to make it a Public variable so it would persist even when the form was hidden, and was given this error:



Therefore, in order to persist this

variable across the form, I created a new class module that had an array inside of it, and then used this object to access the array in my form. The new declaration is this:

```
Public LevelHolder As New PersistentArray
```

This then persists the object even when the form is hidden.

Protecting sheets

I was concerned that when the user was going to edit the level walls, they were going to mess up other things on the worksheet while editing (resize columns, delete rows, move the buttons, etc.) I discovered the Protect worksheet function in Excel, and used the macro recorder to discover how to use that in VBA.

Checking values in text boxes on Exit instead of Change

The last thing that I learned while doing this was when I wanted to put validations on the text boxes (for example, the lives must be numeric and greater than 0). As I did this, I initially had it in the Change event on the textbox. However, this fired on every keystroke. Therefore, I had to put it on the Exit event of the textbox. I also wrote a generic function to do these validations that I could call for each text box:

```
Function CheckTextNumeric(Text As String, Min As Integer, Optional Max As
Integer = -1)
    CheckTextNumeric = Text

    If Not IsNumeric(Text) Then
        CheckTextNumeric = Min
    ElseIf CInt(Text) < Min Then
        CheckTextNumeric = Min
    ElseIf CInt(Text) > Max And Max > -1 Then
        CheckTextNumeric = Max
    End If
End Function
```

This could then be called as follows from the text box Exit event:

```
Private Sub txtSpeed_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    txtSpeed.Text = CheckTextNumeric(txtSpeed.Text, 1)
End Sub
```