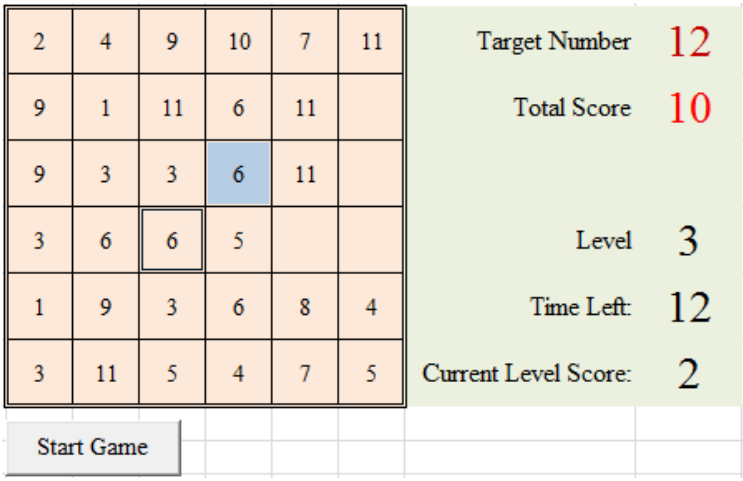
COMP1022Q VBA Adding Game

Contents

1. Brief Description of the Game

In this game, the player needs to select pairs of numbers from a 6 x 6 board very quickly. The selected numbers must add up to a given target number. Select the first number by clicking on it. Select the second number by holding down 'Ctrl' and clicking on the second number, so that both are selected together.

At the beginning, the target number is 10, and the player has 10 seconds. The player moves up one level after each four points earned. With each level up, the target number will increase by one, and a further 10 seconds will be given.



This is a screen dump of the game during play. The player is at the third level, the target number is 12, two cells have been selected, and one point is being earned.

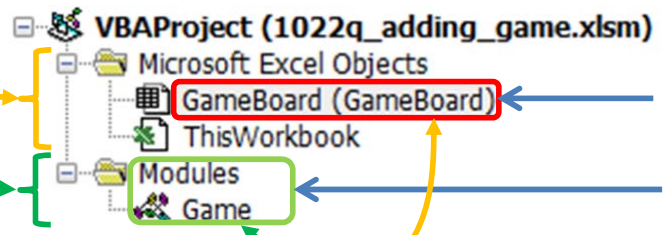
2. Code Organization of this Game

The image below shows the code organization in this Excel file.

As you can see, there are two sections under one VBAProject. They are: 'Microsoft Excel Objects' and 'Modules'.

Under 'Microsoft Excel Objects', an item with a icon is a worksheet; an item with a icon is a workbook. There is only one workbook for each Excel file. In this game we only use one worksheet, which is called 'GameBoard'.

Under 'Modules', an item with a icon is a module. In this example, we choose to put the majority of the code related to the game logic in the module 'Game'. We can use those functions/subroutines in other parts of the Excel file by writing `Game.`*`function_or_subroutine_name`*.

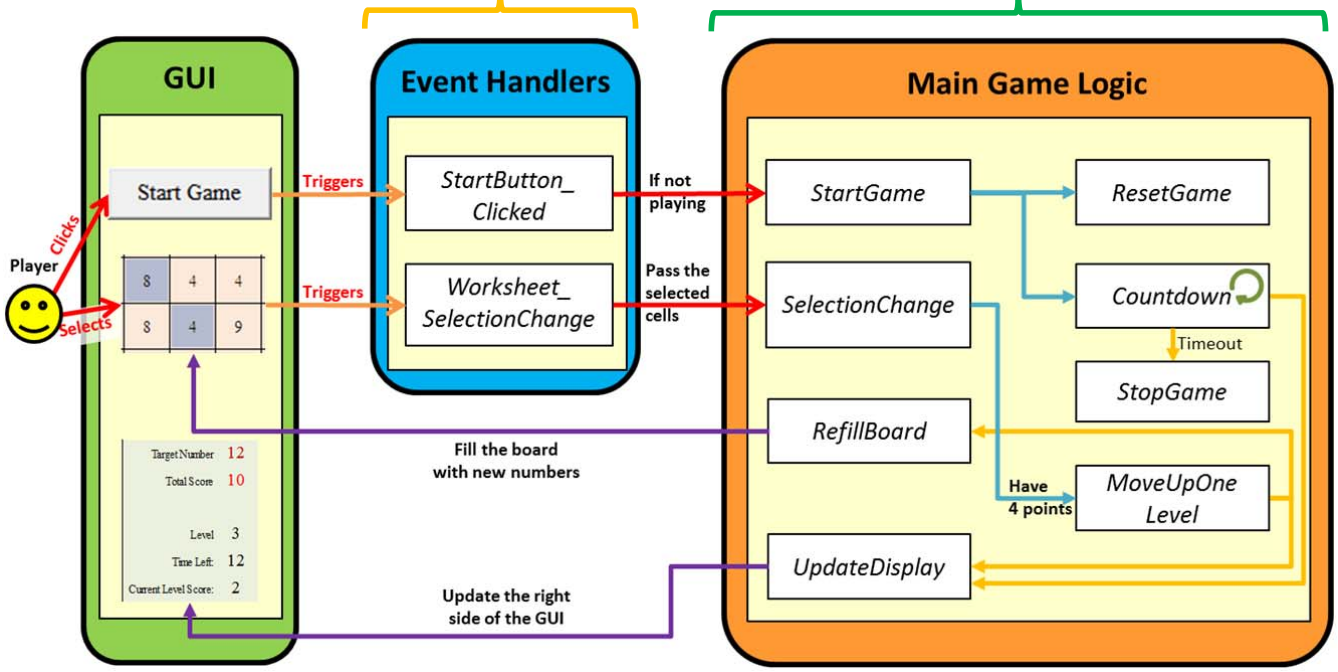


The event handler code for this worksheet ('GameBoard') goes here. In this game, there are two event handlers, StartButton_Click() and Worksheet_SelectionChange().

In this game, the majority of the code goes here

The code organization of this game

3. Structure of the Game



4. Understanding the Code

The game code uses three things that we have not discussed in the course so far: the `Intersect()` function, the `Nothing` keyword and the `For Each` loop.

4.1. The Intersect() Function and the Nothing Keyword

The `Intersect()` function takes Range objects as inputs and then returns the intersection of these ranges, i.e. the cells which appear in all of these ranges. For example, the following code puts the intersection of the ranges A1:A10 and A5:A15 into the variable `MyCells`:

```
Set CommonCells = Intersect( Range("A1:A10"), Range("A5:A15") )
```

After running the above line of code the `CommonCells` variable stores the intersection, i.e. A5:A10, of the given ranges.

If the given ranges do not overlap each other the `Intersect()` function returns the `Nothing` keyword, meaning that the intersection is empty. For example, the code shown below returns `Nothing` (because there are no cells in both inputs) and puts `Nothing` into the variable `MyCells`:

```
Set MyCells = Intersect( Range("A1:A5"), Range("A10:A15") )
```

To check if a variable contains `Nothing`, you can use the command `Is Nothing`, like this:

```
If MyCells Is Nothing Then

    ... do this if MyCells is empty ...

End If
```

4.2. The For Each Loop

The code uses the `For Each` loop in several places.

Like the `For` loop, the `For Each` loop repeatedly runs the loop content. Although you need to use a counter in a `For` loop, you do not need to use one in a `For Each` loop. The `For Each` loop goes through a collection of items, such as the cells in a Range object, by running the loop content for each of the items once. It is useful when you want to repeatedly do something with each item.

For example, the following code goes through each cell in A1:A10 and fills each cell with a random number:

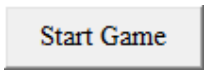
```
For Each Cell In Range("A1:A10")
    Cell.Value = WorksheetFunction.RandBetween(1, 10)
Next Cell
```

A similar `For Each` loop is used in the game to fill the numbers in the game board.

5. Code – The Event Handlers for GUI

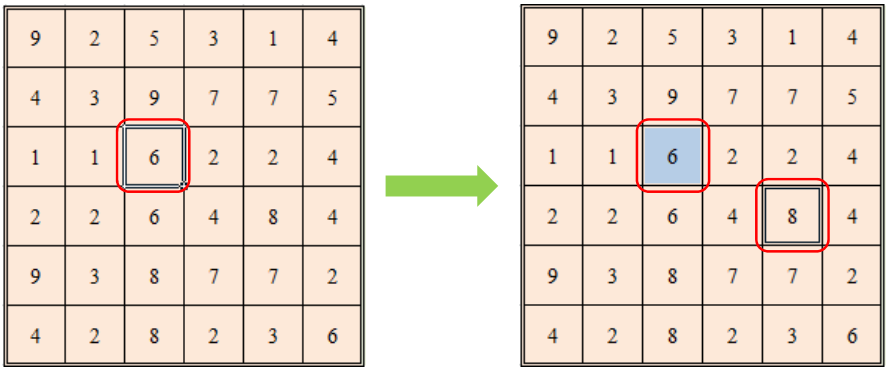
5.1. StartButton_Click()

```
' These subroutines are responsible for event handling.
' They call other subroutines in the 'Game' module using 'Game.function_name'.

Option Explicit

' This subroutine is called when the button is clicked
Private Sub StartButton_Click()
    ' If game is not being played, start the game
    If Game.CurrentlyPlaying = False Then

        ' Start the game
        Game.StartGame
    End If
End Sub
```



5.2. Worksheet_SelectionChange()

```
' If the selection of the cells is changed, e.g., by pressing one of the arrow
' keys, it will trigger a 'SelectionChange' event, which runs this subroutine.
' 'Target' points to the selected cell(s).
```



This shows a second selection being made

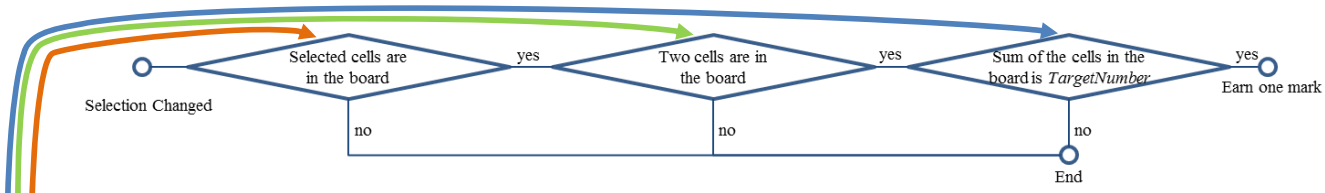
```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    ' Do something only when the game is being played
    If Game.CurrentlyPlaying Then

        ' Call the subroutine defined in the 'Game' module to further
        ' process this event, passing the selected cell(s)
        Game.SelectionChange Target
    End If
End Sub
```

6. Code – The Main Game Logic

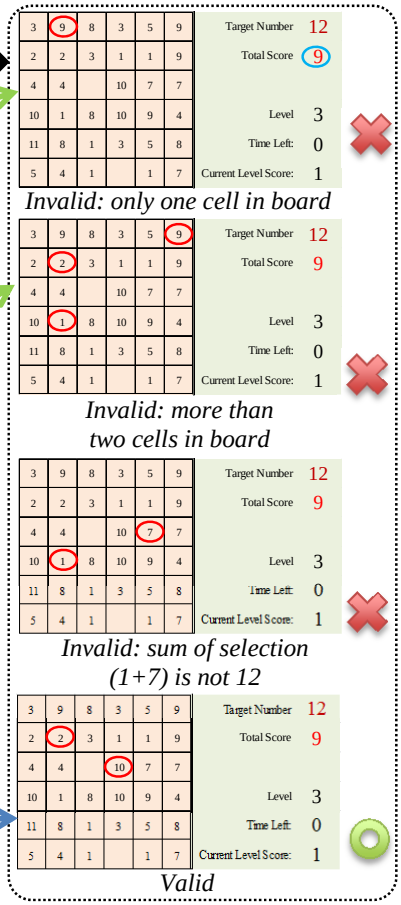
```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' Global variables for the main game logic are declared here.
'
' (A global variable is a variable which can be accessed and changed
' by any function. Global variables were mentioned in labs 6 and 7).
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

' We make the following variable 'Public' to make sure it can be accessed
' in other places (such as the worksheet), as well as this place (the module).

' The following variable is used to indicate whether the game is currently being played.
' 'True' means the game is being played; 'False' means the game is not being played
Public CurrentlyPlaying As Boolean


' The following variables are global variables for this place
' (the module). They can't be accessed outside this place
' because they don't have 'Public' added to them.

' TargetNumber: the target number the player needs to add up. It is increased by
' one for each successive level
' CurrentLevelScore: the score earned only in the current level
' TotalScore: the score earned throughout the whole game
' TimeLeft: how many seconds left to play
' Level: the current level
Dim TargetNumber As Integer, CurrentLevelScore As Integer, TotalScore As Integer
Dim TimeLeft As Integer, Level As Integer

' These variables store references to the Cells used for display
Dim Board As Range, ScoreDisplay As Range, TotalScoreDisplay As Range
Dim LevelDisplay As Range, TargetNumberDisplay As Range, TimeDisplay As Range


' The following variables are constants, 'Const'.
' That means their value is permanently fixed and cannot be changed.

' This constant specifies the target number to start with in level one
Const LevelOneTarget As Integer = 10

' This constant defines how many points should be earned to move up one level
Const PointsToMoveUpOneLevel As Integer = 4
```

6.1. SelectionChange()

```
' This subroutine is called when the player selects one or more cells.
' It checks whether the selected numbers add up to the target number.
' If they do, the score will be increased.
```



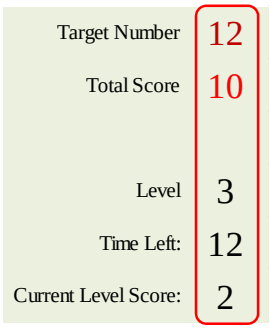
Flowchart showing selection checking




```
Sub SelectionChange(ByVal SelectedRange As Range)

    ' Find the intersection of the selection with the game board.
    ' The purpose is to reject any selections outside the game board.
    Dim SelectedCellsWithinTheBoard As Range
    Set SelectedCellsWithinTheBoard = Intersect(SelectedRange, Board)

    ' If there is at least one cell selected in the board
    If Not SelectedCellsWithinTheBoard Is Nothing Then

        ' Further check if both selected cells are in the board.
        ' If not, exit the subroutine.
        If SelectedCellsWithinTheBoard.Count <> 2 Then
            Exit Sub
        End If

        ' Prepare to calculate the sum of selected cells
        Dim Sum As Integer
        Dim Cell As Range

        Sum = 0

        ' We have to use a loop to access the two cells in the
        ' selection. The For Each loop goes through each of the
        ' two cells once inside SelectedCellsWithinTheBoard.
        For Each Cell In SelectedCellsWithinTheBoard.Cells
            Sum = Sum + Cell.Value
        Next Cell

        ' If the sum matches the target, update the score
        ' and the display
        If Sum = TargetNumber Then

            ' Clear the contents of the selected cells inside the
            ' board without changing the formatting, such as the
            ' background colors and font sizes. ('Range.Clear' will
            ' clear the contents as well as the formatting)
            SelectedCellsWithinTheBoard.ClearContents

            ' Increase the score for this level and the total score by one
            CurrentLevelScore = CurrentLevelScore + 1
            TotalScore = TotalScore + 1

            ' If the player has earned enough points to move up one level, call the
            ' 'MoveUpOneLevel' subroutine to handle everything related to moving up
            ' one level
            If CurrentLevelScore = PointsToMoveUpOneLevel Then
                MoveUpOneLevel
            End If

            ' Update the display to show the score change
            UpdateDisplay
        End If
    End If
End Sub
```

6.2. UpdateDisplay()

```
' This subroutine handles the updates of the right side of the display
Sub UpdateDisplay()

    ' Update the display with the values of corresponding variables
    TargetNumberDisplay.Value = TargetNumber
    TotalScoreDisplay.Value = TotalScore
    LevelDisplay.Value = Level
    TimeDisplay.Value = TimeLeft
    ScoreDisplay.Value = CurrentLevelScore

End Sub
```



The numbers in the area shown above are updated by the subroutine UpdateDisplay()

6.3. RefillBoard()

```
' This subroutine assigns a random number between 1 and (TargetNumber-1)
' to each cell on the 6 x 6 board.
Sub RefillBoard()
    Dim BoardCell As Range

    ' Loop through all cells on the 6 x 6 board.
    For Each BoardCell In Board

        ' This line generates and assigns a random number to each cell. The
        ' generated number is between 1 and (TargetNumber-1) inclusive.
        BoardCell.Value = WorksheetFunction.RandBetween(1, TargetNumber - 1)
    Next BoardCell
End Sub
```

8	9	5			
1	4	8	1	7	
1	8	7	1	7	
2	1	4	7	6	2
1	4	5	8	7	1
	7	1	3	6	1

Level 1: target number is 10

2	9	10	2	8	7
7	2	1	9	10	5
9	9	3	6	4	7
1	9	9	6	6	2
1	4	8	8	2	10
4	3	1	6	4	2

Level 2: target number is 11

6.4. StartGame()

```
' This subroutine is called when the game begins
Sub StartGame()

    ' Reset the game
    ResetGame

    ' Set the variable so that the program knows the game is being played
    CurrentlyPlaying = True

    ' Start to countdown as soon as the game begins
    Countdown
End Sub
```

6.5. ResetGame()

```
' This subroutine will always be called before calling the 'StartGame' subroutine
' to clear any data left from the previous gameplay.
Sub ResetGame()

    ' These lines set the variables with references to the cells used by the game
    Set Board = Range("Cells_Board")
    Set TotalScoreDisplay = Range("Cell_TotalScore")
    Set LevelDisplay = Range("Cell_Level")
    Set ScoreDisplay = Range("Cell_CurrentLevelScore")
    Set TargetNumberDisplay = Range("Cell_TargetNumber")
    Set TimeDisplay = Range("Cell_TimeLeft")

    ' Reset two variables for keeping scores to zero.
    CurrentLevelScore = 0
    TotalScore = 0

    ' Reset level to 1
    Level = 1

    ' Reset the target number to the constant defined at the top
    TargetNumber = LevelOneTarget

    ' Player has 10 seconds of time at the beginning
    TimeLeft = 10

    ' Fill the board with new random numbers
    RefillBoard

    ' Update the right side of the display
    UpdateDisplay

End Sub
```

3	9	8	3	5	9	Target Number	12
2	2	3	1	1	9	Total Score	9
4	4		10	7	7		
10	1	8	10	9	4	Level	3
11	8	1	3	5	8	Time Left:	0
5	4	1		1	7	Current Level Score:	1

Here the game board displays a previously finished game

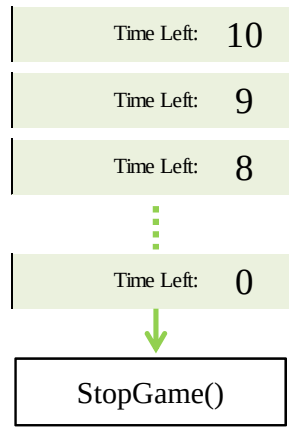
ResetGame

1	1	1	5	2	6	Target Number	10
1	3	8	5	7	1	Total Score	0
7	8	6	1	5	7		
5	6	4	6	4	7	Level	1
8	9	9	8	9	5	Time Left:	10
3	9	6	9	5	3	Current Level Score:	0

After the subroutine ResetGame(), the board is filled with the numbers for a completely new game

6.6. Countdown()

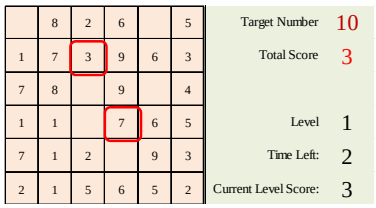
```
' This is a subroutine which updates and checks the time left for the game
Sub Countdown()

    ' The next two lines record the system time before the countdown starts.
    ' 'DateTime.Timer' is a floating point number representing the seconds
    ' passed since midnight.
    Dim StartTime As Single
    StartTime = DateTime.Timer

    ' The code runs only when game is being played
    While CurrentlyPlaying = True

        ' If one second has passed, run the following code
        If DateTime.Timer - StartTime >= 1 Then

            ' Update the 'StartTime' to the current time
            StartTime = DateTime.Timer

            ' Decrease the 'TimeLeft' by one every second
            TimeLeft = TimeLeft - 1

            ' Update the right side of the display
            UpdateDisplay

            ' Stop the game when time runs out
            If TimeLeft <= 0 Then
                StopGame
            End If
        End If
        ' The following instruction prevents Excel from not responding
        ' which can happen because it is too busy with the above loop
        DoEvents
    Wend
End Sub
```



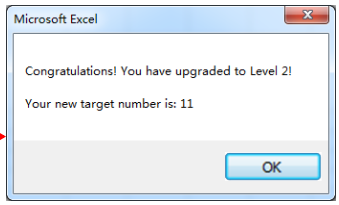
6.7. MoveUpOneLevel()

```
' This subroutine is called when the player earns enough
' points to move up one level
Sub MoveUpOneLevel()

    ' Increment the level by one
    Level = Level + 1

    ' Reset the score for the current level to zero
    CurrentLevelScore = 0

    ' Increment the target number by one
    TargetNumber = TargetNumber + 1

    ' Give a further 10 seconds
    TimeLeft = TimeLeft + 10

    ' Because time passes as the player reads the message, we
    ' need to store the current time before displaying the
    ' message, and restore it after the player  finishes reading
    Dim TimeLeftBeforeDisplayingMessage As Integer
    TimeLeftBeforeDisplayingMessage = TimeLeft

    ' Tell the player what's happening
    MsgBox ("Congratulations! You have upgraded to Level " & Level _
                & "!" & vbNewLine & vbNewLine & _
                 "Your new target number is: " & TargetNumber)

    ' Restore the time left for the player
    TimeLeft = TimeLeftBeforeDisplayingMessage

    ' Update the display to show the new level, time remaining, etc.
    UpdateDisplay

    ' Assign new numbers to the board.
    ' We do this after the showing the message box to prevent the
    ' player seeing the numbers while the message box is being shown.
    RefillBoard
End Sub
```

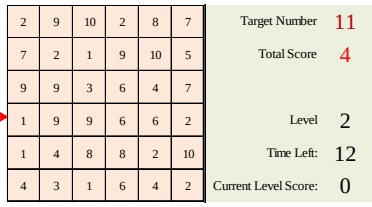
Earning the fourth point will trigger the subroutine MoveUpOneLevel()



This shows the fourth point being earned



Notification of the new level



Game display for the new level

6.8. StopGame()

```
' This subroutine is called when the game finishes.
' It stops the game and shows the player the final score.
Sub StopGame()

    ' This line displays a message to show the player the final score.
    ' The '&' sign is the string concatenation operator.
    ' The 'vbNewLine' means start a new line.
    MsgBox ("Game Over!" & vbNewLine & vbNewLine & "Your Total Score is: " _
            & TotalScore)

    ' Set the variable so that the main VBA code knows the game is not
    ' being played
    CurrentlyPlaying = False
End Sub
```