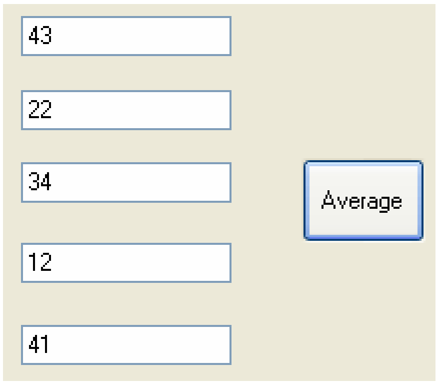
Arrays Part 2

We've covered enough material so far that we can write very sophisticated programs. Let's cover a few more examples that use arrays.

First, once in a while it may be useful to be able to access controls on your forms through an array. Consider a form with five textboxes, and a button. We might want to calculate the average of all the numbers in the textboxes:



If the textboxes are named txtNum1, txtNum2, txtNum3, etc. then we could write code in the Average button to the tune of:

```
Dim intAve As Integer

intAve = CInt(txtNum1.Text) + CInt(txtNum2.Text) + _
         CInt(txtNum3.text) + _
         CInt(txtNum4.Text) + CInt(txtNum5.Text)
intAve = intAve \ 5
MessageBox.Show("The average is " & intAve)
```

This works fine, but it is somewhat tedious. Besides, what if we had 50 textboxes? It would be much nicer if there was some way to make an array of textboxes and then we could loop over the array:

```
sum = 0
For i = 0 to aryTextBoxes.Length – 1
        sum += CInt(aryTextBoxes(i).Text)
Next
intAve = sum \ aryTextBoxes.Length
```

If we could set up this array, then the loop will sum up all the entries in the textboxes and divide by the number of entries to get the average.

There are a couple of ways to perform the above; the simplest way is a bit tedious to set up, but once set up we can use the arrays for all our references.

In this simplest method we do the following:

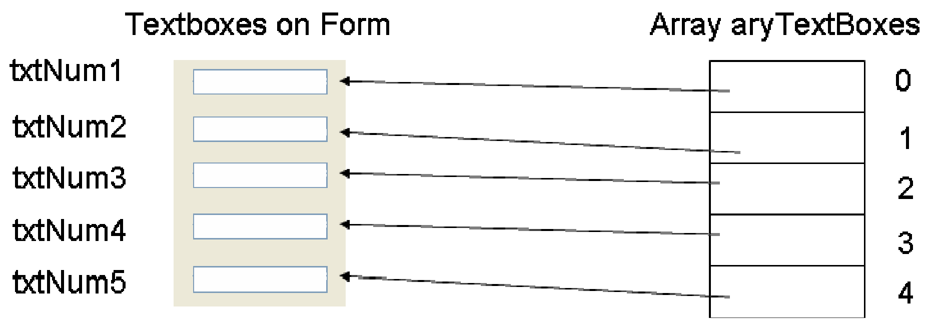
1) Design our form with all the textboxes, labels, etc. For example, we might make a form with 5 textboxes.
2) Make an array of with the same data type as the control added to the form. The array should be the same size as the number of controls on the form. For example, we might make an array of TextBox that can hold five textboxes (0-4):

    Dim aryTextBoxes(4) as TextBox

3) Assign each entry in the array to one of the textboxes on the form. You have to do this in code, so normally it would go in someplace like the form load event, so it is executed once when the program first starts up:

    aryTextBoxes(0) = txtNum1
    aryTextBoxes(1) = txtNum2
    aryTextBoxes(2) = txtNum3
    aryTextBoxes(3) = txtNum4
    aryTextBoxes(4) = txtNum5

   This is the part that can be tedious, but it only has to be done once. This sets a reference or pointer to the textbox on the form:



4) You can now access the array and it will be referencing one of the textboxes; e.g. aryTextBoxes(3).Text will access the same thing as txtNum4.Text.

We could now programmatically compute the average of all textbox entries as:

```
Dim intAve As Integer
Dim sum As Integer = 0
Dim i As Integer

For i = 0 To aryTextBoxes.Length - 1
    sum += CInt(aryTextBoxes(i).Text)
Next
intAve = sum \ aryTextBoxes.Length

MessageBox.Show("The average is " & intAve)
```

Grade Calculator

Here is an example that may actually be of practical use to you! Blackboard's gradebook stores grades for you, but doesn't let you run any "what-if" scenarios to see what your final grade will be depending upon what you get on assignments that have not yet been graded. For example, you might want to know what grade you need to get on the Final in order to get at least 90% in the class. Let's write a program that calculates your grade (as a percentage) based on values you can type into a form.

In our case, we have three categories:

Exams	50%
Assignments	30%
Labs	20%

Each item per category is worth the same amount. To compute the percentage of your final grade based on the exams we would use the formula:

$$ExamPercent = (0.5) \times \frac{\left(\frac{GradeExam1}{TotalPo\,\mathrm{int}\,sExam1} + \frac{GradeExam2}{TotalPo\,\mathrm{int}\,sExam2} + ... + \frac{GradeExam_n}{TotalPo\,\mathrm{int}\,sExam_n} \right)}{n}$$

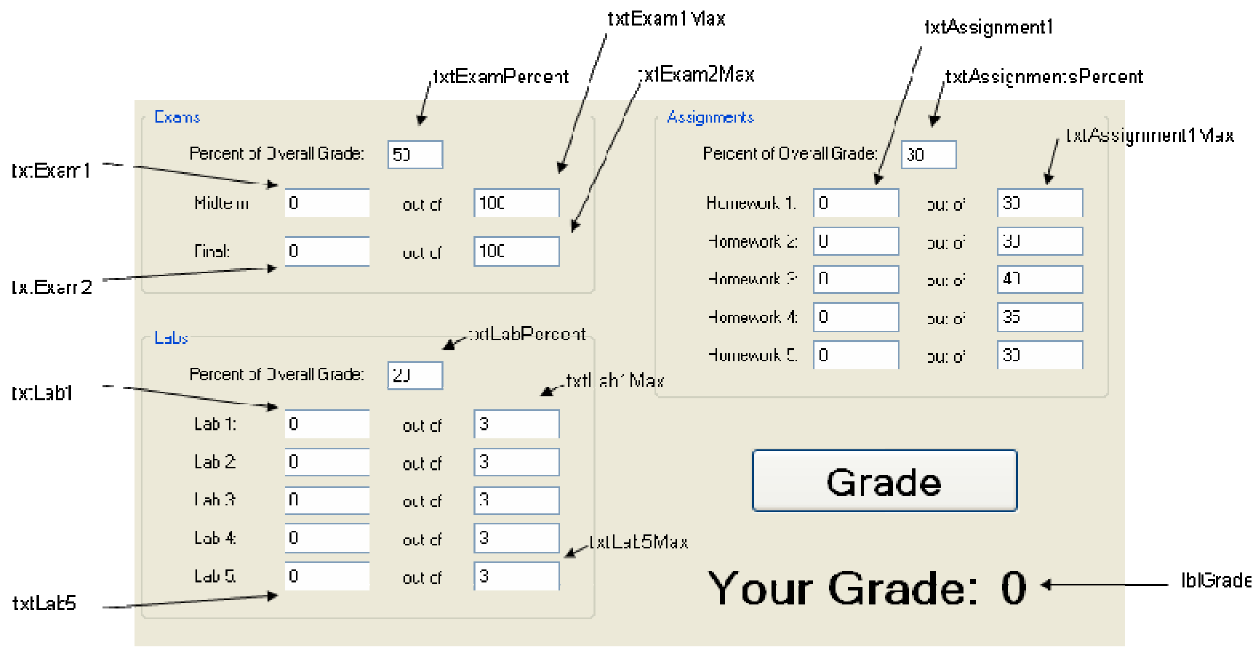
Similarly, we can compute the percentage from assignments and labs:

$$AssignmentPercent = (0.3) \times \frac{\left(\frac{GradeAssign1}{TotalPo\,\mathrm{int}\,sAssign1} + \frac{GradeAssign2}{TotalPo\,\mathrm{int}\,sAssign2} + ... + \frac{GradeAssign_n}{TotalPo\,\mathrm{int}\,sAssign_n} \right)}{n}$$

$$LabPercent = (0.2) \times \frac{\left(\frac{GradeLab1}{TotalPo\,\mathrm{int}\,sLab1} + \frac{GradeLab2}{TotalPo\,\mathrm{int}\,sLab2} + ... + \frac{GradeLab_n}{TotalPo\,\mathrm{int}\,sLab_n} \right)}{n}$$

Our total percentage is then *ExamPercent* + *AssignmentPercent* + *LabPercent*

In our specific class, we have two exams, five labs, and five assignments. Let's design our form as follows:



Exams

Percent of Overall Grade: 50

Midterm 0 out of 100

Final: 0 out of 100

Labs

Percent of Overall Grade: 20

Lab 1: 0 out of 3

Lab 2: 0 out of 3

Lab 3: 0 out of 3

Lab 4: 0 out of 3

Lab 5: 0 out of 3

Assignments

Percent of Overall Grade: 30

Homework 1: 0 out of 30

Homework 2: 0 out of 30

Homework 3: 0 out of 40

Homework 4: 0 out of 35

Homework 5: 0 out of 30

Your Grade: 0

The exams textboxes are named txtExam1 and txtExam2, with their max scores in txtExam1Max and txtExam2Max. I set default values for all the textboxes.

The same naming scheme applies to the labs and the assignments. The labs are named txtLab1, txtLab2, etc. and their max scores in txtLab1Max, txtLab2Max, etc. The assignments are txtAssignment1, txtAssignment2, … txtAssignment1Max, txtAssignment2Max, … finally, the overall grade will be displayed in lblGrade.

First, we can allocate space for all of the arrays. These are created as class variables:

```
Dim aryLabGrades(4) As TextBox              ' Five lab grades
Dim aryLabGradesMax(4) As TextBox           ' Max score for each lab
Dim aryExamGrades(1) As TextBox             ' Two exam grades
Dim aryExamGradesMax(1) As TextBox
Dim aryAssignmentsGrades(4) As TextBox      ' Five homework grades
Dim aryAssignmentsGradesMax(4) As TextBox
```

Next, in the Form_Load event, we can manually set each array entry to the corresponding textbox on the form:

```
        ' Assign textboxes on the form to slots in the arrays
        aryLabGrades(0) = Me.txtLab1
        aryLabGrades(1) = Me.txtLab2
        aryLabGrades(2) = Me.txtLab3
        aryLabGrades(3) = Me.txtLab4
        aryLabGrades(4) = Me.txtLab5
        aryLabGradesMax(0) = Me.txtLab1Max
        aryLabGradesMax(1) = Me.txtLab2Max
        aryLabGradesMax(2) = Me.txtLab3Max
        aryLabGradesMax(3) = Me.txtLab4Max
        aryLabGradesMax(4) = Me.txtLab5Max
        aryExamGrades(0) = Me.txtExam1
        aryExamGrades(1) = Me.txtExam2
        aryExamGradesMax(0) = Me.txtExam1Max
        aryExamGradesMax(1) = Me.txtExam2Max

        aryAssignmentsGrades(0) = Me.txtAssignment1
        aryAssignmentsGrades(1) = Me.txtAssignment2
        aryAssignmentsGrades(2) = Me.txtAssignment3
        aryAssignmentsGrades(3) = Me.txtAssignment4
        aryAssignmentsGrades(4) = Me.txtAssignment5
        aryAssignmentsGradesMax(0) = Me.txtAssignment1Max
        aryAssignmentsGradesMax(1) = Me.txtAssignment2Max
        aryAssignmentsGradesMax(2) = Me.txtAssignment3Max
        aryAssignmentsGradesMax(3) = Me.txtAssignment4Max
        aryAssignmentsGradesMax(4) = Me.txtAssignment5Max
```

Finally, we will need to add the code in the Button Click event to compute the final grade based upon what the user enters into the textboxes. Here is how we would compute just the Exam percentage, which was algebraically specified as:

$$ExamPercent = (0.5) \times \frac{\left(\frac{GradeExam1}{TotalPo\operatorname{int} sExam1} + \frac{GradeExam2}{TotalPo\operatorname{int} sExam2} + \ldots + \frac{GradeExam_n}{TotalPo\operatorname{int} sExam_n}\right)}{n}$$

In this case we need a loop to calculate the numerator of the equation, divide by the number of entries in the array, then multiply by 0.5:

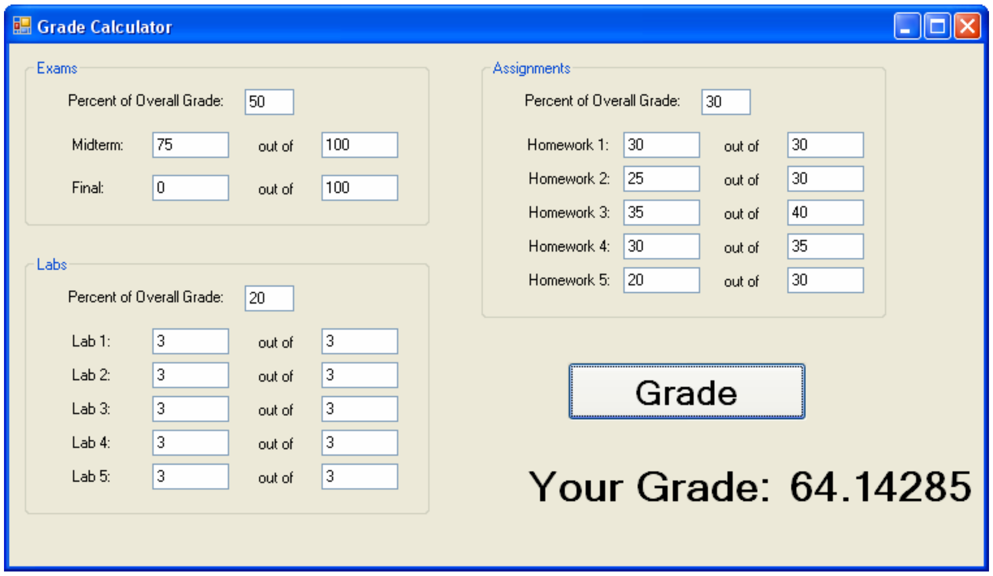
```
    Dim sngExamComponent As Single = 0
    Dim i As Integer

    For i = 0 To aryExamGrades.Length - 1
        sngExamComponent += CSng(aryExamGrades(i).Text) /      _
                            CSng(aryExamGradesMax(i).Text)
    Next
    sngExamComponent = sngExamComponent / aryExamGrades.Length
    sngExamComponent = sngExamComponent * CSng(Me.txtExamPercent.Text)
```

The loop goes through each exam grades and adds together the grade divided by the max. The whole thing is then divided by the number of grades and finally multiplied by the weight for exams (in this case 0.50).

We can repeat this process for the lab and assignment component, then add them together to get the final grade. Here is the whole piece of code:

```
Dim sngExamComponent As Single = 0
Dim sngAssignmentComponent As Single = 0
Dim sngLabComponent As Single = 0
Dim sngTotal As Single = 0
Dim i As Integer

' Calculate the exam component
For i = 0 To aryExamGrades.Length - 1
    sngExamComponent += CSng(aryExamGrades(i).Text) / _
                        CSng(aryExamGradesMax(i).Text)
Next
sngExamComponent = sngExamComponent / aryExamGrades.Length
sngExamComponent = sngExamComponent * CSng(Me.txtExamPercent.Text)

' Calculate the lab component
For i = 0 To aryLabGrades.Length - 1
    sngLabComponent += CSng(aryLabGrades(i).Text) / _
                        CSng(aryLabGradesMax(i).Text)
Next
sngLabComponent = sngLabComponent / aryLabGrades.Length
sngLabComponent = sngLabComponent * CSng(Me.txtLabPercent.Text)

' Calculate the assignments component
For i = 0 To aryAssignmentsGrades.Length - 1
    sngAssignmentComponent += CSng(aryAssignmentsGrades(i).Text) / _
                CSng(aryAssignmentsGradesMax(i).Text)
Next
sngAssignmentComponent = sngAssignmentComponent / _
                        aryAssignmentsGrades.Length
sngAssignmentComponent = sngAssignmentComponent * _
                        CSng(Me.txtAssignmentsPercent.Text)

' Calculate overall by adding together each component
sngTotal = sngAssignmentComponent + sngLabComponent + sngExamComponent
Me.lblGrade.Text = CStr(sngTotal)
```

Here is a screenshot of the application in action. If we had the grades entered and blew off the final (grade of 0) we'd still end up with 64%, a D overall.


Grade Calculator

Exams

Percent of Overall Grade: 50

Midterm:	75	out of	100
Final:	0	out of	100

Labs

Percent of Overall Grade: 20

Lab 1:	3	out of	3
Lab 2:	3	out of	3
Lab 3:	3	out of	3
Lab 4:	3	out of	3
Lab 5:	3	out of	3

Assignments

Percent of Overall Grade: 30

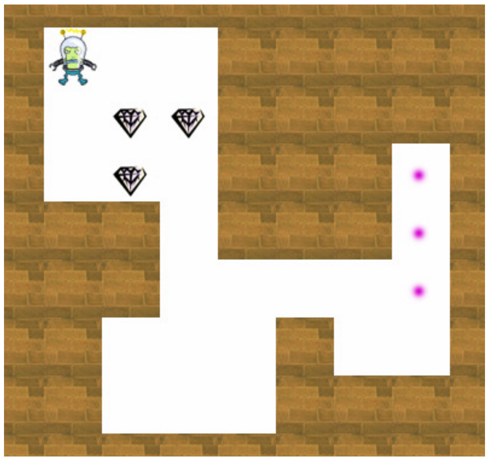
Homework 1:	30	out of	30
Homework 2:	25	out of	30
Homework 3:	35	out of	40
Homework 4:	30	out of	35
Homework 5:	20	out of	30

Your Grade: 64.14285

Sokoban Game

As a much more complex example, let’s build a program to play the game of Sokoban. Since this will be a fairly complex program, we’ll break it up into lots of smaller milestones.

First, the object of Sokoban is to push all of crates onto their destinations. In our case, we’ll use gems instead of crates. Here is a screenshot:



The alien is the player. He can only push the gems and the goal is to push all of them to their destinations . The challenge becomes that gems may get stuck against the wall or in a corner and can no longer be pushed, at which point the player must restart the level from the beginning. A gem cannot be pushed if there is something blocking it on the other side. If you would like to learn more about Sokoban, there is a lot of information on wikipedia: http://en.wikipedia.org/wiki/Sokoban

Representing the Game Board

Our version of Sokoban will always be on a 10 cell tall by 10 cell high game board. To represent the game board we can use a 2D array.

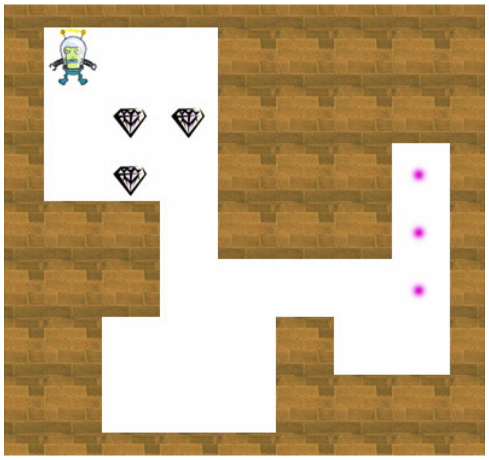
Let’s make it a 2D array of Strings. Each string will represent the item(s) that are in the respective cell. We can use the following codes:

" " (space) -	Blank space
"X" -	Wall
"A" -	Alien
"G" -	Gem
"F" -	Final destination for gem, empty
"P" -	Final destination for gem, with a gem
"a" -	Alien over an empty final destination for a gem

For example, we could use the following 2D array of strings to represent our level:

```
XXXXXXXXXX
XA  XXXXXX
X GGXXXXXX
X G XXXFXX
XXX XXXFXX
XXX    FXX
XX   X  XX
XX   XXXXX
XXXXXXXXXX
XXXXXXXXXX
```

This corresponds graphically with:



Why use these letter codes for items on the game board? One advantage is we can put the game board in a text file and then load that file in our program. This lets us easily create levels by simply editing a text file and no programming expertise is required.

Let's say that we have a text file named `level0.txt` and it contains:

```
XXXXXXXXXX
XA  XXXXXX
X GGXXXXXX
X G XXXFXX
XXX XXXFXX
XXX    FXX
XX   X  XX
XX   XXXXX
XXXXXXXXXX
XXXXXXXXXX
```

Here is code to load it into a 2D array:

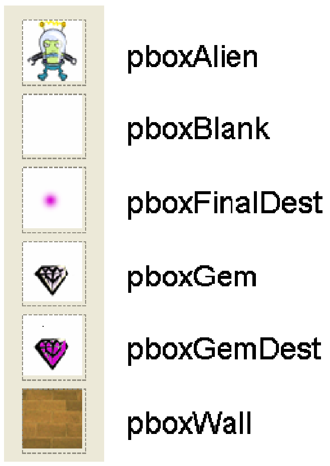
```
Public Class frmSokoban
    Dim aryBoard(9, 9) As String        ' Game board

    Private Sub frmSokoban_Load(. . .) Handles MyBase.Load
        ' Load in level0.txt
        Dim fileLevel As IO.StreamReader = _
                             IO.File.OpenText("../../level0.txt")
        Dim x, y As Integer

        ' The levels are exactly 10 lines of 10 characters per line.
        ' Read in each line then loop and put each char into the array
        For y = 0 To 9
            Dim sLine As String
            sLine = fileLevel.ReadLine        ' Read one whole line
            ' Put each character into the array
            For x = 0 To 9
                aryBoard(x, y) = sLine.Substring(x, 1)
            Next        Next
        fileLevel.Close()
    End Sub
End Class
```

To test it out, we could put a breakpoint at the end and inspect aryBoard to see if it has loaded the information.

Displaying The Game Board

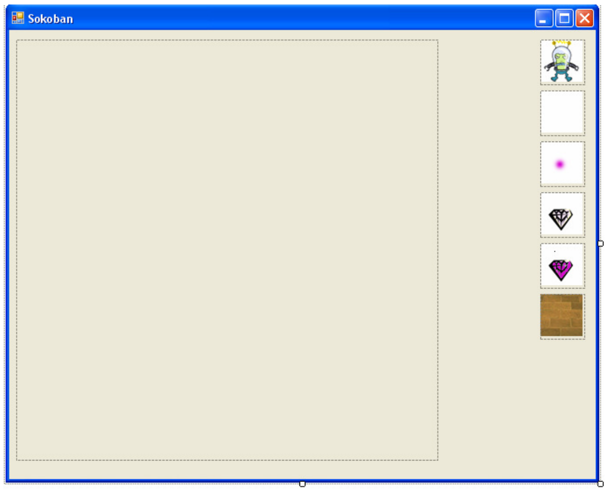
We could display the game board by outputting it as text to a textbox or the console, but that would be somewhat unsatisfying. Let’s write some code to draw the game board graphically by repeatedly drawing images for each cell.

To do this, let’s add a picturebox to the form for each graphical element that could go into a cell:



These images were just created from clipart and a paint program. Each one is 48 pixels wide by 48 pixels tall. We are placing the pictureboxes on the form so they can serve as source images to copy to our main game board. Since we don't want the user to see them, we can make each one **invisible** by setting its **visible property to false**.

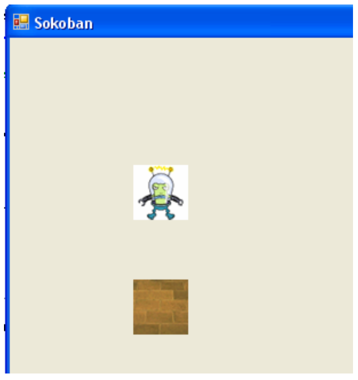
Finally, let's place a big picturebox called pboxBoard in the middle left of the form. This picturebox will serve to display our game board. Here is a snapshot of the whole form:



To draw the little icon images into the game board, we can use the DrawImage function. We put it inside the Paint event for the picturebox, just like you did when drawing lines and circles. Here is a small example:

```
    Private Sub pboxBoard_Paint(. . .) Handles pboxBoard.Paint
        Dim g As Graphics = e.Graphics

        g.DrawImage(pboxAlien.Image, 100, 100)
        g.DrawImage(pboxWall.Image, 100, 200)
    End Sub
```

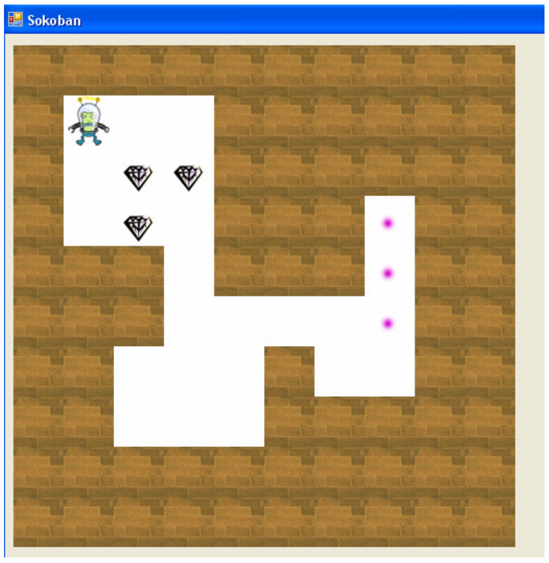
This draws the Alien image into pboxBoard with the upper left coordinate at 100,100. It draws the wall image into pboxBoard with the upper left coordinate at 100,200:



To draw our entire game board we just need to make a loop that goes over every element in the 2D array and then draw it in the corresponding location. Here is the whole code:

```
    Private Sub pboxBoard_Paint(. . .) Handles pboxBoard.Paint
        Dim g As Graphics = e.Graphics

        Dim x, y As Integer

        For y = 0 To 9
            For x = 0 To 9
                Select Case aryBoard(x, y)
                    Case "A", "a"
                        g.DrawImage(pboxAlien.Image, x * 48, y * 48)
                    Case " "
                        g.DrawImage(pboxBlank.Image, x * 48, y * 48)
                    Case "X"
                        g.DrawImage(pboxWall.Image, x * 48, y * 48)
                    Case "G"
                        g.DrawImage(pboxGem.Image, x * 48, y * 48)
                    Case "F"
                        g.DrawImage(pboxFinalDest.Image, x *48, y * 48)
                    Case "P"
                        g.DrawImage(pboxGemDest.Image, x * 48, y * 48)
                End Select
            Next
        Next
    End Sub
```

The result looks like this when the program is run:



If we ever change the game board we can force the display to refresh by calling `pboxBoard.Invalidate()`. For example, the following code would “move” the alien down one cell by putting a blank in the old location (1,1) and putting the alien in (1,2):

```
    Private Sub Button1_Click(. . .) Handles Button1.Click
        aryBoard(1, 1) = " "
        aryBoard(1, 2) = "A"
        pboxBoard.Invalidate()
    End Sub
```

Moving The Player

To move the player (i.e. the Alien) around on the screen, first the computer needs to know what the coordinates are for the cell it is in. Here is a subroutine that searches through the array until it finds the "A" and then it returns the X and Y location as ByRef parameters.

```
    Public Sub FindAlien(ByRef xCoord As Integer, _
                         ByRef yCoord As Integer)
        Dim x, y As Integer
        For y = 0 To 9
            For x = 0 To 9
                If aryBoard(x, y) = "A" Then
                    xCoord = x
                    yCoord = y
                    Return
                End If
            Next
        Next
    End Sub
```

We would use this subroutine with a call like:

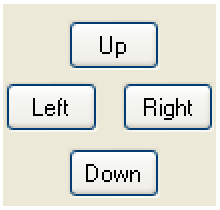
```
        FindAlien(intX, intY)
```

After the call, intX will be set to the X coordinate of the alien and intY will be set to the Y coordinate.

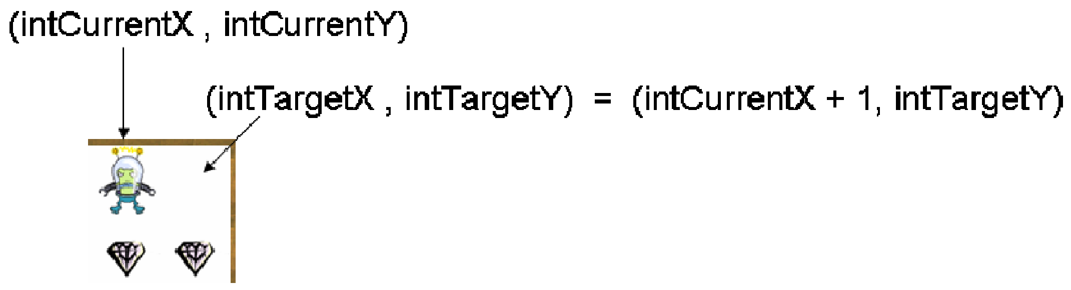
To move the alien let's add two new class level variables to track the alien's cell coordinates and then initialize them in the form load event using our subroutine:

```
Public Class frmSokoban
    Dim aryBoard(9, 9) As String       ' Game board
    Dim intCurrentX, intCurrentY As Integer

    Private Sub frmSokoban_Load(. . .) Handles MyBase.Load
        ' Previous code to load in level0.txt

        ...

        FindAlien(intCurrentX, intCurrentY)
    End Sub
…
```

Now that we have the alien's location we can try to move it. Add four buttons that correspond to the directions we might want to move (no diagonal moves allowed).



We need to determine what cell the player is trying to move to. We'll use the variables intTargetX and intTargetY to depict the target:


(intCurrentX , intCurrentY)
(intTargetX , intTargetY) = (intCurrentX + 1, intTargetY)

If Pushing Right

If the cell that the player is trying to move to is BLANK then allow the move by erasing the "A" from aryBoard at the old location, put an "A" in the new location, update the intCurrentX and intCurrentY variables, and then redraw the board. Here is code for clicking on the RIGHT button:

```
Private Sub btnRight_Click(. . .) Handles btnRight.Click
    Dim intTargetX, intTargetY As Integer

    intTargetX = intCurrentX + 1     ' Right is +1 in the X
    intTargetY = intCurrentY
    If aryBoard(intTargetX, intTargetY) = " " Then
        ' Erase player from current location
        aryBoard(intCurrentX, intCurrentY) = " "
        ' Put in new location
        intCurrentY = intTargetY
        intCurrentX = intTargetX
        aryBoard(intCurrentX, intCurrentY) = "A"
        ' Redraw
        pboxBoard.Invalidate()
    End If
End Sub
```

This works but it won't allow us to walk over the empty final destination cells, only blank cells. We should be able to walk over the empty final destination cells. If we walked over a empty final destination cell (letter “F”) cell, we can replace this with a lowercase “a” to indicate the player is over one. Then when the player moves off we should replace the cell with a “F” again.

```
    Private Sub btnRight_Click(. . .) Handles btnRight.Click
        Dim intTargetX, intTargetY As Integer

        intTargetX = intCurrentX + 1     ' Right is +1 in the X
        intTargetY = intCurrentY
        If aryBoard(intTargetX, intTargetY) = " " Then
            ' Erase player from current location, consider if
            ' was over a final destination
            If aryBoard(intCurrentX, intCurrentY) = "a" Then
                aryBoard(intCurrentX, intCurrentY) = "F"
            Else
                aryBoard(intCurrentX, intCurrentY) = " "
            End If
            ' Put in new location
            intCurrentY = intTargetY
            intCurrentX = intTargetX
            aryBoard(intCurrentX, intCurrentY) = "A"
            ' Redraw
            pboxBoard.Invalidate()
        ElseIf aryBoard(intTargetX, intTargetY) = "F" Then
            aryBoard(intCurrentX, intCurrentY) = " "
            intCurrentY = intTargetY
            intCurrentX = intTargetX
            ' Lower case "a" is the alien over a final destination
            aryBoard(intCurrentX, intCurrentY) = "a"
            pboxBoard.Invalidate()
        End If
    End Sub
```

If we repeat this for every button click you might notice the code is identical except for the first three lines that set the values of X and Y. This makes the code a prime subject for a subroutine. Let's make a subroutine called MoveAlien to do the move, and the button clicks just set up the target:

```
    Private Sub btnUp_Click(. . .) Handles btnUp.Click
        Dim intTargetX, intTargetY As Integer

        intTargetX = intCurrentX
        intTargetY = intCurrentY - 1     ' Trying to move UP so -1 in Y
        MoveAlien(intTargetX, intTargetY)
    End Sub

    Private Sub btnRight_Click(. . .) Handles btnRight.Click
        Dim intTargetX, intTargetY As Integer

        intTargetX = intCurrentX + 1     ' Right is +1 in the X
        intTargetY = intCurrentY
        MoveAlien(intTargetX, intTargetY)
    End Sub

    Private Sub btnLeft_Click(. . .) Handles btnLeft.Click
        Dim intTargetX, intTargetY As Integer
        intTargetX = intCurrentX - 1     ' Left is +1 in the X
        intTargetY = intCurrentY
        MoveAlien(intTargetX, intTargetY)
    End Sub
    Private Sub btnDown_Click(. . .) Handles btnDown.Click
        Dim intTargetX, intTargetY As Integer

        intTargetX = intCurrentX
        intTargetY = intCurrentY + 1      ' Move positive for Y
        MoveAlien(intTargetX, intTargetY)
    End Sub

    Public Sub MoveAlien(ByVal intTargetX As Integer, _
                         ByVal intTargetY As Integer)
        If aryBoard(intTargetX, intTargetY) = " " Then
            ' Erase player from current location, consider if
            ' was over a final destination
            If aryBoard(intCurrentX, intCurrentY) = "a" Then
                aryBoard(intCurrentX, intCurrentY) = "F"
            Else
                aryBoard(intCurrentX, intCurrentY) = " "
            End If
            ' Put in new location
            intCurrentY = intTargetY
            intCurrentX = intTargetX
            aryBoard(intCurrentX, intCurrentY) = "A"
            ' Redraw
            pboxBoard.Invalidate()
        ElseIf aryBoard(intTargetX, intTargetY) = "F" Then
            If aryBoard(intCurrentX, intCurrentY) = "a" Then
                aryBoard(intCurrentX, intCurrentY) = "F"
            Else
                aryBoard(intCurrentX, intCurrentY) = " "
            End If
            intCurrentY = intTargetY
            intCurrentX = intTargetX
            ' Lower case "a" is the alien over a final destination
            aryBoard(intCurrentX, intCurrentY) = "a"
            pboxBoard.Invalidate()
        End If
    End Sub
```

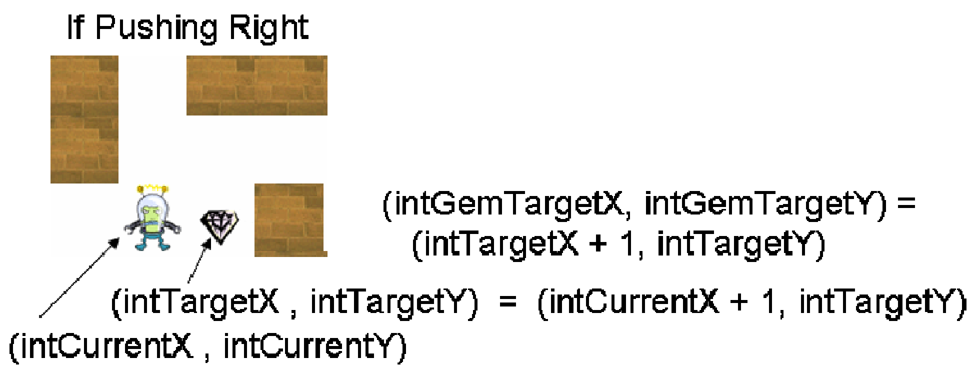
This allows the alien to move on blank spaces and over final destinations.

Pushing Gems

Next we need to be able to push the gems. This entails another subroutine similar to the "MoveAlien" subroutine above, except it should be for moving a gem.

Let's check for this scenario in the button click events. I decided to check for it here because we need to know what is two cells away from the player in the direction we are trying to push a gem, and that information is not available in the MoveAlien subroutine as it is written.

Here are the variables that describe the cells of interest:


If Pushing Right
(intGemTargetX, intGemTargetY) =
(intTargetX + 1, intTargetY)
(intTargetX , intTargetY) = (intCurrentX + 1, intTargetY)
(intCurrentX , intCurrentY)

Here is a modified version of clicking on the Right button:

```
Private Sub btnRight_Click(. . .) Handles btnRight.Click
    Dim intTargetX, intTargetY As Integer

    intTargetX = intCurrentX + 1    ' Right is +1 in the X
    intTargetY = intCurrentY
    If aryBoard(intTargetX, intTargetY) = "G" Or _
       aryBoard(intTargetX, intTargetY) = "P" Then
        ' Get coordinates of where we want to push a gem
        Dim intGemTargetX, intGemTargetY As Integer
        intGemTargetX = intTargetX + 1
        intGemTargetY = intTargetY
        ' Send in as arguments the gem's coordinates and
        ' coordinates of where the gem would be pushed
        PushGem(intTargetX,intTargetY,intGemTargetX,intGemTargetY)
    Else
        MoveAlien(intTargetX, intTargetY)
    End If
End Sub
```

We would place similar code in btnLeft_Click (except intGemTargetX = intTargetX -1) and so forth, updating the gem target for each direction. In the PushGem subroutine we need to check to see if the gem target is empty (either blank or a final destination). If so, move the gem there and move the player to where the gem was.

Here is the code for pushing a gem. It is somewhat messy because of checking for the two possible conditions that a gem or the alien might be over. Either might be standing over a final destination or a blank.

```
    Public Sub PushGem(ByVal intTargetX As Integer,
                       ByVal intTargetY As Integer, _
                       ByVal intGemTargetX As Integer, _
                       ByVal intGemTargetY As Integer)
        ' If gem target is blank or a final destination allow push
        If aryBoard(intGemTargetX, intGemTargetY) = " " Or _
           aryBoard(intGemTargetX, intGemTargetY) = "F" Then
            ' Move gem to target
            If aryBoard(intGemTargetX, intGemTargetY) = " " Then
                aryBoard(intGemTargetX, intGemTargetY) = "G"
            Else
                aryBoard(intGemTargetX, intGemTargetY) = "P"
            End If
            ' Move alien to where the gem was
            If aryBoard(intTargetX, intTargetY) = "G" Then
                ' Alien over blank
                aryBoard(intTargetX, intTargetY) = "A"
            ElseIf aryBoard(intTargetX, intTargetY) = "P" Then
                ' Alien over final destination
                aryBoard(intTargetX, intTargetY) = "a"
            End If
            ' Erase what the alien was over
            If aryBoard(intCurrentX, intCurrentY) = "A" Then
                aryBoard(intCurrentX, intCurrentY) = " "
            ElseIf aryBoard(intCurrentX, intCurrentY) = "a" Then
                aryBoard(intCurrentX, intCurrentY) = "F"
            End If
            ' Update position and redraw
            intCurrentX = intTargetX
            intCurrentY = intTargetY
            pboxBoard.Invalidate()
        End If
    End Sub
```

Checking for a Win

We're not quite done yet. The code lets us play but doesn't tell us when we've finished the level. One way to check for a win is after every move, scan through the entire board. If there are no "G"s then the level is complete. A "G" is a gem that is not on a final destination.

```
    Public Function LevelComplete() As Boolean
        Dim x, y As Integer
        For y = 0 To 9
            For x = 0 To 9
                If aryBoard(x, y) = "G" Then
                    ' Exit if we find any gems
                    Return False
                End If
            Next
        Next
        ' If we get here, we found no gems over blanks.
        ' So we must have won, return true
        Return True
    End Function
```

We can call this at the end of PushGem, since the only time we would win is when a gem was just pushed:

```
    Public Sub PushGem(ByVal intTargetX As Integer, _
                       ByVal intTargetY As Integer, _
                       ByVal intGemTargetX As Integer, _
                       ByVal intGemTargetY As Integer)
        ' If gem target is blank or a final destination allow push
        If aryBoard(intGemTargetX, intGemTargetY) = " " Or _
           aryBoard(intGemTargetX, intGemTargetY) = "F" Then

            . . .

            If LevelComplete() Then
                MessageBox.Show("You did it!  You finished the level!")
            End If
        End If
    End Sub
```

More Sokoban

There are a lot of other features that should be added; a “reset” button should be available to reset the board in case the player gets stuck. Multiple levels could also be added; after one level is complete the next, harder level would load. A score could be displayed that shows the number of moves made. High scores could be kept and saved in a text file.

These additions are left as an exercise to the reader, if desired ☺ These additions would also make a suitable homework #5 assignment.