

Visual Basic[®]

Projects for the Classroom



Introduction	3
Why we have done these books.....	4
How to Use This Book	4
Future Developments	6
Wingdings Instructor Notes	7
Wingdings Student Project	9
Thermometer Program Instructor Notes	11
Thermometer Program Student Project.....	13
Dice Display Instructor Notes.....	15
Dice Display Student Project.....	17
Guessing Game 1 Instructor Notes	18
Guessing Game 1 Student Project	20
Guessing Game 2 Instructor Notes	21
Guessing Game 2 Student Project	23
Flip Book Program Instructor Notes.....	24
Flip Book Program Student Project	26
Simple House Drawing Instructor Notes.....	28
Simple House Drawing Student Project.....	30
Mice Invaders Instructor Notes	32
Mice Invaders Student Project.....	35
Tic Tac Toe Instructor Notes.....	38
Tic Tac Toe Student Project.....	40
Picture Display Program Instructor Notes.....	42
Picture Display Program Student Project.....	44
Multiple Choice Grading Program Instructor Notes	46
Multiple Choice Grading Program Student Project	48
Multiple Choice Grading Program with User Defined Types Instructor Notes..	50
Multiple Choice Grading Program with User Defined Types Student Project ..	52
Check Book Balancing Program Instructor Notes.....	54
Check Book Balancing Program Student Project	57
Drawing Program Instructor Notes	59
Drawing Program Student Project	62
Roman Numeral Conversion Instructor Notes	64

Roman Numeral Conversion	Student Project.....	66
Palindromes	Instructor Notes.....	68
Palindromes	Student Project.....	70
Conway's Game of Life	Instructor Notes	72
Conway's Game of Life	Student Project.....	75
Checkers Game Program	Instructor Notes.....	77
Checkers Game Program	Student Project.....	80

Introduction

Why we have done these books.

This book includes a set of Visual Basic projects available through Mainfunction.com. This book is designed to supplement and enhance existing and developing curriculum at the secondary and post secondary level.

There is a growing need for trained programmers in the work force. Visual Basic is one of the programming languages in highest demand. Some companies are paying bonuses for trained Visual Basic programmers. More and more schools recognize the need for Visual Basic training for School to Work programs and are developing courses to fill that need. College preparatory schools are finding that Visual Basic is an ideal first programming language for their students.

In any programming course, there is a need to projects that both develop necessary skills and hold student interest. The projects in this book are designed to supply additional projects for instructors to use.

Who are we

The Mainfunction offers news, curriculum, grants and resources for post-secondary and secondary computer science, engineering and information technology educators. It's about using technology in innovative ways to further computer science and information technology instruction. Visit us on the World Wide Web at <http://www.mainfunction.com> or direct to the teacher section at <http://educators.mainfunction.com>.

How to Use This Book

Target Audience

This project book has been written to be used in the context of a first programming course using Visual Basic. It assumes no previous programming knowledge on the part of a student. It is designed for instructors to use as a supplement to their primary instructional resources. As such, it assumes that the student has available to them an instructor and a textbook for use as reference.

Projects

All projects have two sections. The first section for teacher use and the second for student use. The instructor section includes the following sections:

- ABILITY LEVEL – Required ability level for students attempting project.
- APPROXIMATE COMPLETION TIME – An estimate of how long students will require completing the project.
- OBJECTIVES – What skills and information are being reinforced by the project.
- SKILLS NEEDED – A list of prerequisite knowledge and skills for students who undertake the project.

- **MATERIALS NEEDED** – What resources and materials the project requires supporting student work on the project.
- **TEACHING SUGGESTIONS** – Suggestions on ways to introduce the project, common problems encountered by students, and other information related to the project.
- **RESOURCES** – Any additional resources involved in the project.
- **SUGGESTED EVALUATION** – Indications of what to look for and grade in student projects.
- **SUGGESTED SOLUTION** – A narrative programming solution to the project. Coded and commented sample solutions in Visual Basic are available by sending email to editor@mainfunction.com.

The student sections may be reproduced and distributed to students. These sections include:

- **ABILITY LEVEL** – Required ability level for students attempting project.
- **APPROXIMATE COMPLETION TIME** – An estimate of how long students will require completing the project.
- **OBJECTIVES** – What skills and information are being reinforced by the project.
- **OVERVIEW OF PROJECT** – A summary of what the project involves.
- **PROJECT INSTRUCTIONS** – A list of general instructions for completing the project.
- **ADDITIONAL RESOURCES** - Any additional resources involved in the project.
- **SUGGESTED SOLUTION** – A suggestion of one possible solution. This usually comes in the form of a screen capture of a completed solution form. Students should be encouraged to develop alternative solutions. Some instructors may choose not to distribute a solution so as not to limit student creativity. Others may wish to insist on a specific form appearance so that students do not waste time on the appearance of the form over the code solution.
- **PROJECT EXTRAS** – A list of optional additions to the project. These suggestions will be used by students desiring to do more than the minimum requirements of the project.

Ability Levels

All projects in this book have a suggested ability level. The levels, beginner, intermediate and advanced are rather broad. The explanations below are intended to help the instructor select the projects that are appropriate for their students.

Beginner

Assumes little or no previous experience with programming or Visual Basic before the instructor's introduction of the project. May understand basic concepts but be unsure of implementation details.

Intermediate

Understands basic concepts including objects, events, properties of objects, and form design. Understands assignment statements, loops, and decision statements. Understands simple text file input and output. Understands variable and control arrays.

Advanced

Understands record types and advanced data structures. Understands file input and output. Uses multiple forms, subroutines and functions. Is able to design solutions to complex problems.

Future Developments

Mainfunction intends to add projects to the resource database on a regular basis. We encourage you to submit your favorite projects. All projects will be credited to the submitter. Please visit <http://educators.mainfunction.com> and register as a teacher to contribute.

Wingdings Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 20 Minutes

OBJECTIVES:

- Use an assignment statement to modify object properties

SKILLS NEEDED:

- Basic understanding of object properties

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

The purpose of this project is to give students a chance to see objects respond to events. It relies on the most basic of object properties (font, caption and text) and the event (click) that students are most used to using.

Explain the difference between design time and run time changes to properties. Tell them that, while the user can not directly change captions, the program can be programmed to allow the user to indirectly change many properties.

Pay careful attention to the explanation of assignment statements. Some students will have trouble grasping that the copy moves from right to left. They are used to thinking from left to right.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should respond to a click of a command button and copy a string from one box to another. Consider giving that as the C level and ask students to add other features for more credit. See the student section for suggestions of additional features.

SUGGESTED SOLUTION:

Have the students create a label and a text box. Have them change the font of the label box to Wingdings. Wingdings are a standard font included with Windows. If this font is not available, use any font that includes something other than normal English characters.

The command click routine will include a simple assignment statement to copy the contents of the text box into the caption of the label box.

Wingdings Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 20 Minutes

OBJECTIVES:

- Use an assignment statement to modify object properties

OVERVIEW OF PROJECT:

Create a program that shows a user what characters they enter look like in a different font.

PROJECT INSTRUCTIONS:

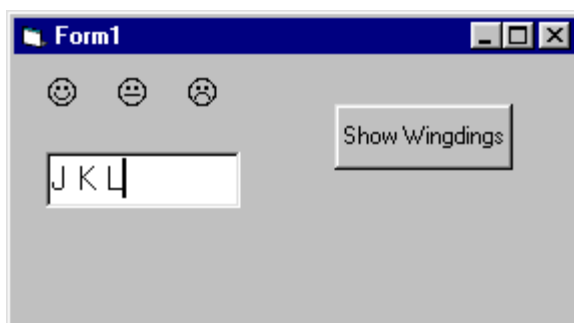
1. On a new form, create a label box, a text box and a command button.
2. Change the font property in the label box to Wingdings, or to a font assigned by your teacher. You may want to increase the size of the font as well.
3. Clear the caption property in the label box and the text property in the text box.
4. Change the caption in the command button to something descriptive.
5. Write an assignment statement in the command click routine to copy the contents of the text box into the label box.
6. Test your program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

The End command is used to shutdown a program. Add an exit command button.

Copying nothing in to a text or caption property empties it. Using a pair of double quotes with no space between them indicates nothing. Use that to create a Clear button that empties both the text and label boxes.

Create a number of label boxes with different fonts and font sizes. Copy the text string in to all the label boxes.

Copy the text box into the label box in response to some other event. For example, on mouse move over the label box.

Thermometer Program Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use mathematical formulas in a program
- Use a scroll bar object

SKILLS NEEDED:

- Understanding of object properties
- Understanding of mathematical operations

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Most introductory programming texts use temperature conversion as an early project or example. Asking a student to write a simple input a number, convert the value and display the result program is that it does not really use the power of the computer. This project uses scroll bars as an input device to make the example more visually interesting and to introduce a powerful user interface tool.

The student may set the initial value of the bar either in the property box at design time or in the form load routine with an assignment statement. Unless the student includes program code at form load, the label boxes will not display the results of a conversion until the scroll bar is moved. This can be a useful introduction to concepts of initialization in general.

Scroll bars must have their minimum and maximum values set to work properly with this project. The minimum value is set at the top of the bar. The maximum is at the bottom. This is the opposite of what one sees with a thermometer. For this project, the student will want to set minimum to the highest value and the maximum to the lowest value. Visual Basic does not care that minimum is higher than maximum. This gives the teacher the opportunity to explain that variables, or properties, are just names that the computer uses but does not really understand.

Note: When a user drags a scroll bar the bars value does not register a change until the mouse button is released. This may confuse some students initially.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete thermometer program should:

- Change the values in the Fahrenheit and centigrade labels with the movement of a scroll bar
- Exit the program cleanly

SUGGESTED SOLUTION:

```
Private Sub VScroll11_Change()  
' Display the current (new) value of the scroll bar  
Label1.Caption = Format (VScroll11.Value)  
' Convert the value of the scroll bar to centigrade  
Label2.Caption = Format (Int ((VScroll11.Value - 32) * (5 / 9)))  
End Sub
```

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a scroll bar object

OVERVIEW OF PROJECT:

Create a scroll bar to represent a thermometer. As the slider on the scroll bar is moved display temperature in both Fahrenheit and centigrade in boxes on a form.

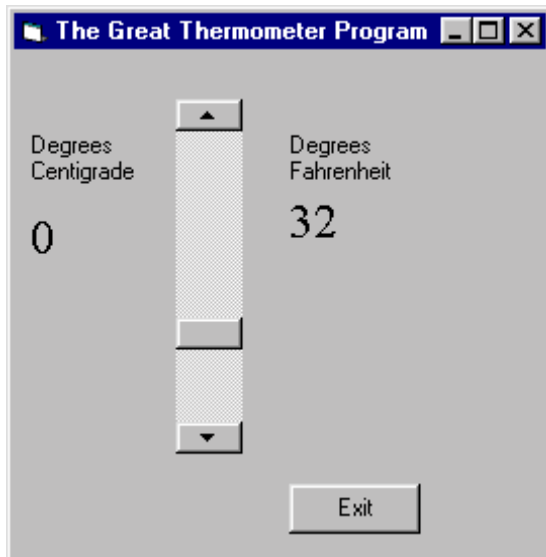
PROJECT INSTRUCTIONS:

1. On a new form, create a scroll bar.
2. Set the minimum and maximum properties for the scroll bar.
3. Create label boxes to display Fahrenheit and centigrade temperatures.
4. Create label boxes to label the temperature boxes.
5. Copy the value of the scroll bar into the Fahrenheit label box in the routine reacting to changes to the scroll bar.
6. Change the value in the centigrade box by using the formula for converting Fahrenheit to centigrade when the Fahrenheit value changes.
7. Create an exit button with the appropriate code to end the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:



The formula for converting Fahrenheit to centigrade is: $C = (F - 32) * (5 / 9)$

PROJECT EXTRAS:

Add a background to the form. Perhaps a weather related picture.

Set different background colors for the Fahrenheit and centigrade labels.

Label the slider bar with degree markings on either side.

Add buttons to move the slider to "freezing" and/or "boiling."

Dice Display Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use string functions
- Understand and use mathematical functions

SKILLS NEEDED:

- Understanding of basic variable types – double, integer and string

MATERIALS NEEDED:

- Visual Basic
- Paint Program

TEACHING SUGGESTIONS:

The two critical parts of this project are picking the random number and loading the appropriate picture into each picture box.

The RND function returns a real number (Visual Basic type *Single*) that is greater than zero but less than 1. To produce the random integers required by this project, students may use this formula:

$$\text{Dice1} = \text{Int} ((6 * \text{Rnd}) + 1)$$

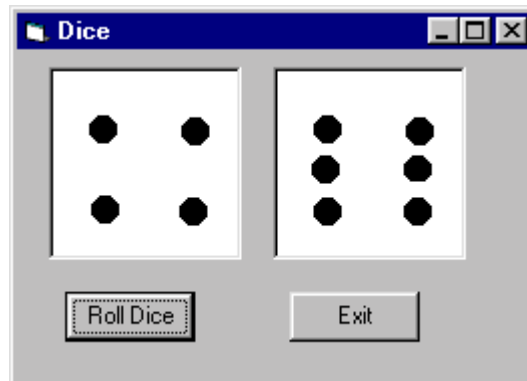
Multiplying the value returned by Rnd results in a value greater than zero but less than six because Rnd will never be equal to one. The Int function truncates the Single precision result of the multiplication and provides an integer between one and five. Since the range we need is from 1 to 6 the programmer adds 1. Explain to the class that using variables in place of the numeric constants 1 and 6 creates a more general formula.

SUGGESTED SOLUTION:

Six 1 inch by 1 inch die face images should be prepared for students to use. Black and white images will take up the least room and load quickest. Place the images where students can either use them directly or copy them to their own workspaces. Optionally, students can create their own dice images but they must be careful to create them same size as the picture.

The easiest way to load these images is to create the images with names that are identical except for an identifying number. For example, DIE1.BMP, DIE2.BMP... DIE6.BMP. Use the FORMAT function and a random number in

range to concatenate a file name into a string variable for the LoadPicture function.



The image above is one possible solution.

```
Private Sub Command1_Click()  
Dim Dice1 As Single  
Dim Dice2 As Single  
  
Dice1 = Int((6 * Rnd) + 1) ' Pick a random number between 1 and 6  
Dice2 = Int((6 * Rnd) + 1) ' Pick a second random number between 1 and 6  
  
' Build file names and load images into each picture box  
Picture1.Picture = LoadPicture("c:\d" + Format(Dice1) + ".bmp")  
Picture2.Picture = LoadPicture("c:\d" + Format(Dice2) + ".bmp")  
  
End Sub
```

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete dice display program will:

- Display at least two die images
- Die images will change randomly in response to button clicks
- Die images will generally represent different values
- The program will exit in response to a button click

Dice Display Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use string functions
- Understand and use mathematical functions

OVERVIEW OF PROJECT:

Many games depend on the roll of a pair of dice. The object of this program is to simulate the roll of dice. The program will display dice images showing the value of dice values. The project assumes standard six sided dice but can be expanded to other sizes if desired.

PROJECT INSTRUCTIONS:

1. Create a form with two picture boxes and two command buttons.
2. Set the height and width of both picture boxes to 1440 twips by 1440 twips.
3. Set the caption of one button to Exit and write code so that the program terminates when the button is pushed.
4. Set the caption of the second button to "Roll Dice" and write code so that pictures of dice in each picture box

ADDITIONAL RESOURCES:

- Textbook
- Paint program

SUGGESTED SOLUTION:

Use the Rnd function and a formula to pick a random number between one and six. Use that number to build the name of a bit map image (BMP) file and use LoadPicture to display that image in a picture box. Do this for each picture box in the form when the display dice button is clicked.

PROJECT EXTRAS:

Display more than two dice.

Display the total of the dice in a label box.

Create and use your own die images.

Guessing Game 1 Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use global variables

SKILLS NEEDED:

- Understanding of basic events and object properties

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Scope of variables is one of the more confusing computer science concepts to be explained in an introductory programming course. The values stored in local variables are lost whenever a subroutine is existed. Variables used by several routines or that must be saved for different calls to the same routine must be created differently.

Visual Basic has several options for extending the scope of variables. Declaring a variable in a form's General Declarations section makes a variable and its value available to all subroutines. We can say that the scope of the variable is global to all routines in the form. A name used for a global variable may only be used for one unique variable.

If we just need a variable in one routine but want to have it keep its value between runs of the routine we want it to be static. Declare a static variable in the subroutine that uses it using the **static** reserved word rather than the **Dim** reserved word. Names used as static variables may be used in any number of routines but changes to any of those variables will have no effect on the variables of that name in any other subroutine. This makes programs more difficult to understand and to debug. You will want to discourage students from using duplicate names for static variables.

SUGGESTED SOLUTION:

The guessing game where a player is told that their guess is too high or too low lends itself to simulating a binary search. Using a binary search, any number in the range of 1 to 100 may be determined in no more than seven guesses. Use global variables to keep track of the highest and lowest possible numbers, as well as the current guess.

Set a guess that is too high as the new top of range. Set a guess that is too low as the new bottom of range. The next guess should be in the middle of the new range of possibilities.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Test the full range of the program and insure that it will properly guess both one and one hundred. Also, try 49 and 51. The program should properly reach any number.

A good program will not duplicate guesses or guess numbers higher or lower than have already been rejected.

A new game should reset all settings so that counts and ranges are set the same for all games.

Guessing Game 1 Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use global variables

OVERVIEW OF PROJECT:

The object of this project is to write a computer program that will allow the computer to guess a number that you have selected. The computer will make a guess and you, the player, will tell the computer if it guessed too high, too low, or that it guessed the number.

The program will also allow the player to start a new game or to exit the program.

PROJECT INSTRUCTIONS:

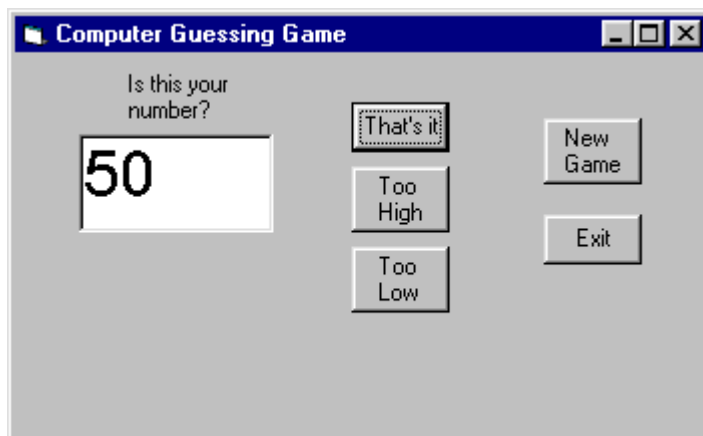
1. Create a form with a labeled box to display the computers guess.
2. Create a new game and an exit button.
3. Create objects to indicate if the guess was too high, too low, or right on target.
4. Create program code to respond to the player's indication by selecting and displaying a new computer guess.
5. When the computer guesses the player's number, reset the so that a new game may begin.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

This solution uses buttons to indicate the success of the computer's guess. The new game button will reset the guess to 50 and set any counters back to zero.



PROJECT EXTRAS:

Display the number of guesses the computer required to find the players number.

Declare victory automatically when the last possible guess is made.

Keep and report counts of how many guesses were too high and too low.

Guessing Game 2 Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 45 minutes

OBJECTIVES:

- Understand and use **IF** blocks
- Understand and use **Elseif**

SKILLS NEEDED:

- Understanding of basic events and object properties
- Familiarity with the RND function and generating random numbers in a range.

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Nested **IF** statements and the use of the **Elseif** in **IF** blocks allow a programmer to evaluate a number of possibilities. In this program, there are three possibilities for the program to consider. The number the player guesses may be too high, too low or the number the computer is looking for. The programmer has several options available to evaluate these possibilities. One is to have three totally independent **IF** blocks, one for each case.

Three independent **IF** blocks are easy to set up but are not as efficient as a program could be. The program must evaluate all three statements even if the first one satisfies the problem. Using nested blocks and the **Elseif** statement is more efficient because comparisons are only made until one statement evaluates to true.

If the programmer knows the likely probability of the various options, they may order the **IF** checks to have the most probably check made first and the least likely check performed last.

Block **IF** statements require **End If** statements. Forgetting an **End If** statement is one of the most common errors made with **IF** blocks. A second source of confusion is the difference between **Elseif** (as one word) and having an **If** as the first line of the **Else** clause. In the second case, there must be a matching **End If**

for the **IF** statement. A number of **Elseif** clauses may be part of the initial **IF** statement.

Suggest that students use indents to identify the sections of block IF statements. This will make it easier to match **End If** to **If**. Students often just add **End If** statements at the end of a section to silence error messages. This more often results in logic errors than in a correctly working program.

SUGGESTED SOLUTION:

Use the **RND** function to select a random number, for the computer. For example:

```
MyNumber = Int (RND * 100) + 1
```

The following section of code compares the player's guess (in Text1.Text) to the computer's number (in MyNumber) with the results reported to the user via message boxes.

```
If Val (Text1.Text) > MyNumber Then
    MsgBox "Your guess is too high",, "Guessing Game"
ElseIf Val (Text1.Text) < MyNumber Then
    MsgBox "Your guess is too low",, "Guessing Game"
Else
    MsgBox "You guessed my number in " & Format (GuessCnt) & _
        " guesses.",, "Guessing Game"
End If
```

This example assumes a static counter variable, called GuessCnt, which is incremented at each guess.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

The program must correctly identify a numbers relationship to the number "guessed" by the computer. It must not give several conflicting messages for the same guess.

Guessing Game 2 Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 45 minutes

OBJECTIVES:

- Understand and use **IF** blocks
- Understand and use **Elseif**

OVERVIEW OF PROJECT:

Create a program to have the computer pick a random number and allow a player to guess the number. The program will tell the user if their guess is correct or, if incorrect, if the guess is too high or too low.

The program should allow the player to start new games or exit the game completely.

PROJECT INSTRUCTIONS:

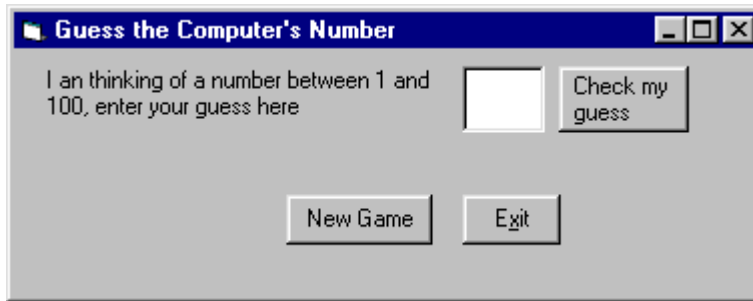
1. In the Form Load routine, use the RND function to pick a random number between one and a hundred.
2. Create a text box for the player to enter their guess and a button for the user to tell the computer the guess is ready to evaluate.
3. Evaluate the player's guess and display a message to tell the user if their guess is correct or, if incorrect, if the guess is too high or too low.
4. In the subroutine for the "New Game" button, clear the text box and assign a new value to the computer's number. Reset any counters in use.
5. Add an exit button with code to shutdown the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

One possible game board is displayed below. Display messages to the player using label captions, picture boxes or message boxes.



PROJECT EXTRAS:

Count and display how many attempts the player takes to guess the computer's number.

Allow the player a limited number of guesses. After the number of guesses has been exceeded, display the computer's number.

Flip Book Program Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a timer object

SKILLS NEEDED:

- Basic understanding of object properties
- Understanding of static variables

MATERIALS NEEDED:

- Visual Basic
- Graphic editor such as Paintbrush

TEACHING SUGGESTIONS:

Animation attracts the interest of most people. This project uses a simple form of animation, the flipbook. Flipbooks use a series of pictures. Each picture is slightly different from its predecessor. The rapid changing of pictures results in the appearance of movement. This is a natural application for a timer.

Two properties are most important in timer objects: enabled and interval. Timer routines must be enabled to execute. Once they are enabled, timer routines execute after the passage of time indicated by the interval time. The time is set in milliseconds.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should start a timer that loads or displays a series of pictures. It should also exit cleanly.

SUGGESTED SOLUTION:

Create a series of picture files using a graphical editor. The names of these files should have names that are identical except for an identifying number. For example, Bsktbl0.bmp, Bsktbl1.bmp... Bsktbl10.bmp

Initially the timer object should be set to false. The timer interval should be set to some fraction of a second. Experimentation will help determine the optimum time that will be dependent on the size and loading time of the pictures used. A static counter variable should be incremented in the timer routine. Use the FORMAT function and the counter to concatenate a file name into a string variable for the LoadPicture function. Once the counter reaches its maximum value, and the last picture has been displayed, the counter should be reset to zero and the enabled property should be set to false to disable the timer.

Use a command button to enable the timer. This will cause the animation to start.

An alternate solution would involve a control array of picture boxes. Pictures can be associated with each picture box at design time. The visible property of each picture box would be set to false. The timer routine would use a counter similar to the first solution. In this case, the timer routine would set the visible property of the current picture box to false and the visible property of the next box to true.

This solution provides faster display of pictures at the cost of larger image size of executable programs and less flexibility. Using a single picture box and building file names as needed allows a program to accept partial file names and display a variety of animations.

Note: Visual Basic version 5.0 picture support includes bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF files, and JPEG (.jpg) files.

Visual Basic version 4.0 picture support includes bitmaps (.bmp), icons (.ico), run-length encoded files (.rle), and metafile (.wmf) files.

Flip Book Program Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a timer

OVERVIEW OF PROJECT:

Create a program that displays an animated flipbook.

PROJECT INSTRUCTIONS:

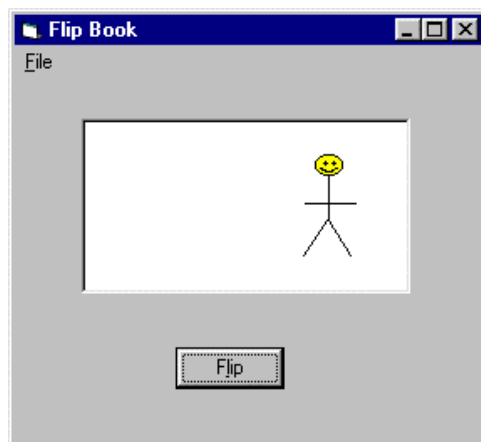
1. Create a picture box on a form
2. Create a series of pictures that vary slightly for each other.
3. Create a command button to enable the timer.
4. Write code in the timer event to load the display the picture series.
5. Program a way to cleanly exit the program.

ADDITIONAL RESOURCES:

- Textbook
- Graphic editor such as Paintbrush

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

Add the ability to accept the name of the first file in a picture series to be displayed.

Add the ability of the user to change the timer interval.

Add a second picture box and display two series at the same time. Alternatively, display the same series twice.

Simple House Drawing Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand objects and properties
- Understand events

SKILLS NEEDED:

- Understanding of a coordinate system

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Understanding the coordinate system used by Visual Basic is key to this project. The top left-hand corner is 0,0 and the X and Y values of the bottom right corner are positive integers whose value varies based on the size of the form. This means that adding to X moves to the right. Adding to Y moves down and subtracting from Y moves up. It is very common for students to confuse X and Y. Reversing X and Y often results in figures drawn sideways.

Using the BF parameters to the Line method fills rectangles. Triangles can not be filled automatically. An innovative student could fill the roof by drawing multiple lines. Fill style must be set to 0 (solid) or circles will not be filled. If a fill color is not specified the circle will be filled with the forms foreground color and appear unfilled.

Encourage students to plan their houses before writing the code. Graph paper can be useful for plotting out the locations of various objects to be drawn.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete house-drawing program should:

- Draw a house with a roof, a door and two windows.
- Draw a circle representing the sun in the sky.

- Add additional objects of the students choosing.
- Clear the form at the user's request.
- Exit cleanly at the user's request.

SUGGESTED SOLUTION:

```

Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As
Single, Y As Single)
' Respond to the press of the mouse button by drawing a house and warm
yellow sun

' Draw the main box for the house with the top left
' corner where the mouse was clicked (X, Y)
Line (X, Y)-(X + 1000, Y + 1000), QBColor(2), BF
' Draw the first line of the roof up to a peak
Line (X, Y)-(X + 500, Y - 300), QBColor(1)
' Draw a line from the peak of the roof to the corner of the house
Line (X + 500, Y - 300)-(X + 1000, Y), QBColor(1)
' Draw a rectangle for a door
Line (X + 425, Y + 600)-(X + 575, Y + 1000), , BF
' Draw two white windows on the house
Line (X + 200, Y + 200)-(X + 400, Y + 400), QBColor(15), BF
Line (X + 600, Y + 200)-(X + 800, Y + 400), QBColor(15), BF
' Set the FillStyle to solid
FillStyle = 0
' Set the fillcolor to Bright Yellow
FillColor = QBColor(14)
' Draw a bright yellow sun to the left and above the house
Circle (X - 500, Y - 500), 300, QBColor(14)
End Sub

Private Sub Command1_Click()
' Clear the form of any pictures drawn
Form1.Cls
End Sub

Private Sub Command2_Click()
End ' Exit the program
End Sub

```

Simple House Drawing Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand objects and properties
- Understand events

OVERVIEW OF PROJECT:

The object of this program is to draw a simple picture of a house on a form. Draw the house in a location indicated by the user clicking the mouse button.

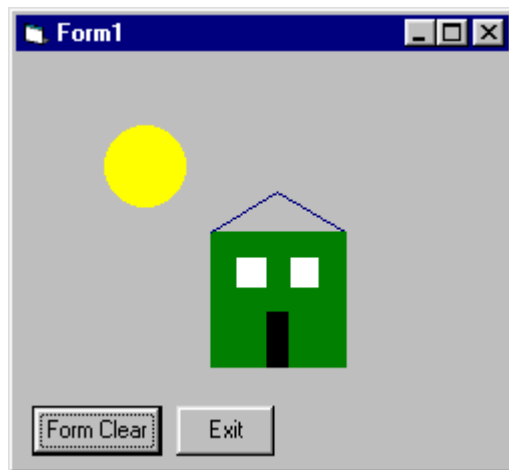
PROJECT INSTRUCTIONS:

1. On a new form, create two command buttons.
2. In the MouseDown routine for the form draw a filled box starting at the location indicated by the X and y valued returned to the routine.
3. Next, draw a triangle as a roof.
4. Draw a tall rectangle in the center or the bottom of the house.
5. Draw two white rectangles to represent windows on the house.
6. Somewhere above the house, draw a filled yellow circle for the sun.
7. If the exit button is pressed, shut down the program.

ADDITIONAL RESOURCES:

- Textbook
- Graph paper

SUGGESTED SOLUTION:



PROJECT EXTRAS:

Draw other objects around the house. For example, a chimney, ornate windows, clouds in the sky, step stones up to the door, or a smile on the sun.

Change the caption in the forms title bar.

Make the bottom of the form green and the top part of the form blue.

Fill in the roof.

Mice Invaders Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 3-4 hours

OBJECTIVES:

Use timers

Use the message box function

SKILLS NEEDED:

- Understanding of looping constructs
- Understanding of variable and object arrays

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Two properties are most important in timer objects: enabled and interval. Timer routines must be enabled to execute. Once they are enabled, timer routines execute after the passage of time indicated by the interval time. The time is set in milliseconds. The maximum value for a timer interval is 65,535 milliseconds. This is equivalent to just over 1 minute. Multiple timer objects may be created.

Students will occasionally ask how they can create timers that cover greater intervals than a minute. A very simple way to do this is by using a static counter variable in the timer event routine. Set the timer interval for as large a value that is evenly divisible into the length of time desired. Every time the interval is completed, increment the counter. Once the counter indicates that enough intervals have passed, reset the counter and then perform the desired actions.

The difference between the message box statement and message box function often confuses students. The function returns a value that allows the program to make a decision based on a response from the user. Make it clear to students that the message box statement only displays a message. The function form must be used if a response is required. Students often forget either the parentheses or the response variable required by the message box function.

This program uses a very simple form of animation. Changing the Top property moves picture boxes. Once this simple program is completed, encourage students to develop their own original games using the same concepts.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Picture boxes should move at a uniform rate down the form. Clicking of a box should result in its immediately being placed at the top of the form. The player should have the option of at least three different speeds. Displays should accurately report the number of boxes clicked and missed. The game should have a clear end. Restarting the game should reset all counters and start with the boxes at the top of the screen.

SUGGESTED SOLUTION:

Create a frame for the boxes to move on. While boxes may be created directly on the form, creating the boxes on top of a frame will make defining the range of the box movement easier. Create the playing boxes as a control array. Create the zero element in the array and completely define all properties before creating additional objects. Be sure to set the back color and picture properties.

Create label boxes for the status display information. Declare form level variables to hold the number of boxes hit and escaped. Also creating a form level variable to control the speed of box movement will facilitate changing that rate through program action.

Two variables will control the movement of the boxes. The first is the timer interval. The distance the box moves each interval is the second variable. The larger the distance moved each interval or the shorter the time between movements the faster the boxes will move. Encourage students to experiment with both settings, once the program works, to find optimum settings. Selecting only one of those variables for use in changing speed during a program run will simplify the program.

The timer routine is the workhorse of this program. In the timer routine, write a loop that incrementally moves each box by adding to its top value. After moving the box, compare it's location to the bottom of the frame. If the top value of the box is larger than the height of the frame added to the top of the frame then that box has dropped below the frame and "escaped." If this is the case, increment the count of escaped boxes and update the display. Also, move the box back to the top of the frame (set the boxes top to the value of the top of the frame).

After each escape, check the escape counter to determine if enough boxes have escaped to end the game. If enough boxes have escaped, display a message box displaying only the yes and no buttons and asking the player if they want to continue with a new game. If the user selects the no button, the End statement will exit the game. If the player selects the yes button, call the new game routine to initialize the display and counter variables.

The picture box click routine moved the box to the top of the frame, increments the hit counter and updates the counter display.

Allow the user to select different box speeds by using a group of menu options. Label them slow, medium and fast. In the click routine for each option set a specific value to the distance to move each box on each timer event. A move

involved and flexible method would be a scroll bar. Use a scroll bar by having the change event place the scroll bar value into the from level variable controlling box speed. Be sure the set reasonable minimum and maximum values for the scroll bar.

Mice Invaders Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 3-4 hours

OBJECTIVES:

- Use timers
- Use the message box function

SKILLS NEEDED:

- Understanding of looping constructs
- Understanding of variable and object arrays

OVERVIEW OF PROJECT:

Have a number of picture boxes move down a form. If the user clicks on a picture box, move that box back to the top of the screen. Count and display the number of times a box is clicked. When a box moves below the bottom of the screen, count that box as having 'gotten away.' After a specific number of boxes getting away, stop the game and allow the user to continue playing or exit the program. Allow the user to quit the game and exit the program.

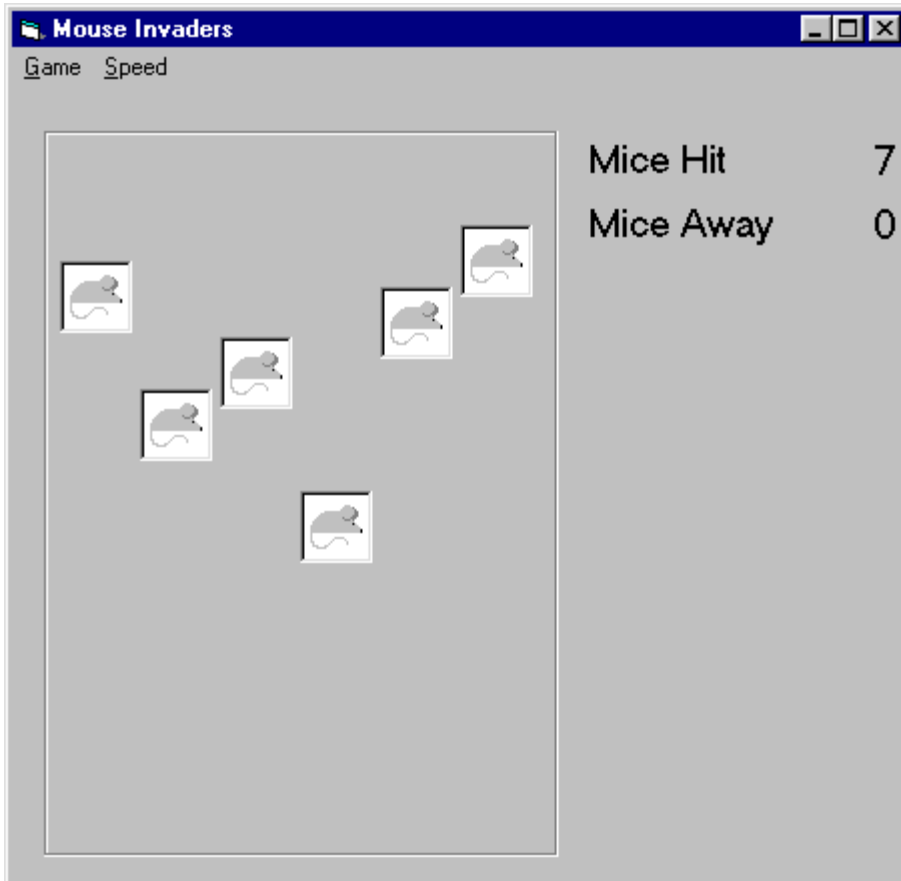
PROJECT INSTRUCTIONS:

1. Create an object array of picture boxes across the screen.
2. Add a picture or icon file to the picture property of each picture box.
3. Create a timer object. Each time the timer event occurs, move each box down by adding to its Top property value.
4. In the timer event, check to see if a box moves below the bottom of the screen. Increment and display a counter variable if the box escapes. Move escaped boxes back to the top of the screen.
5. If the number of escaped boxes exceeds a specific number, for example ten, call the message box function to ask the user if they wish to play a new game.
6. If the user wishes to play a new game, reset all counters and reposition the boxes at the top of the form. If they do not want to continue, exit the program.
7. Program the picture click routine to move any picture clicked on back to the top of the screen by resetting its Top property value.
8. Add an option, either by buttons, by menu items or by some other way, to allow the user to select the speed the boxes move.
9. Allow the user to exit the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:



The mice in this example are icon files distributed with Visual Basic (mouse04.ico). These icons are an optional part of the installation and may or may not be available on your computer.

PROJECT EXTRAS:

Add more mice.

Allow the user to select different icons for the picture boxes.

Create your own characters for the picture boxes.

Put a different picture in each box.

Make the pictures move horizontally across the screen rather than down the screen.

Make the pictures speed up the longer the game runs.

Use a scroll bar to change the speed of the boxes.

Add a pause command.

Tic Tac Toe Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 1 hour 30 minutes

OBJECTIVES:

- Understand control arrays and their use in IF statements
- Understand and use nested IF statements
- Use Boolean variables as flags

SKILLS NEEDED:

- Understanding of Boolean operators
- Understanding of control properties and control arrays
- Understanding of multi line and nested IF blocks

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Control arrays are very similar to variable arrays. Unlike simple variables, control arrays have a number of properties with their own values. This makes them somewhat like arrays of records. That connection can be used when records are introduced. For this project, the program will access values using the Tag property. The Tag property may have any type of value. For this project, integer values are easiest to use. Suggest that students use a 0 for an empty box, a 1 for a box with an X in it and a 2 for an O marked box.

This project will use a Boolean flag to keep track of whose turn it is. The flag can be easily changed using the assignment flag = **NOT** flag. Explain that students do not need to write an IF statement to first check the current value and that the NOT statement will reverse the value what ever it is.

Students make a number of common mistakes with this project.

If a student forgets to set the Tag property to 0 they will get a error when they compare a tag whose value has been set to 1 or 2 with a tag whose value has not been set.

If a program reports that there are three moves in a row incorrectly the student may have arranged the labels in an order other then what they think they did. Check the index properties and rearrange as necessary.

When pasting new copies of a label, a student will occasionally paste a label inside an existing label. This will cause indexing problems. Using the drop down menu in the property box to move to specific boxes will help identify these cases.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete Tic Tac Toe program should handle the following cases correctly:

- moves may not be made on occupied squares
- a winning game should be recognized
- no additional moves should be permitted after a game has been won
- the winner (X or O) should be properly identified
- tie games should be recognized
- the board should be cleared either automatically after a game is over or manually by user request

SUGGESTED SOLUTION:

One solution would be to use captions in label boxes. A second alternative would be to use Loadpicture to load appropriate pictures into picture boxes. A .BMP image of 1 inch by 1 inch created with PaintBrush will fit neatly in a 1440 by 1440 twip picture box. Remember that CLS will not clear a picture. Using Loadpicture to load a null picture will restore a picture box to blank. Alternatively, a "blank" image may be loaded.

When an object is clicked, first verify that the object is empty. Display a message box or sound the beep if the box is not empty. If the box is empty, display the symbol of the current player and set the objects tag to indicate who has moved into it. Check to see if there is now a winner. Declare a winner when three boxes in a row, column or diagonal are occupied by the same non-zero code. Declare the player who last moved the winner.

If there is no winner, check to see if all the boxes are occupied. Declare a draw if all boxes are occupied and there is no winner.

In case of a draw or a winner being declared, clear the board after the user acknowledges the message. Create a subroutine to clear the board. Call this subroutine from the declare winner code, declare draw code and the clear board or new game button or menu item. Clear the board by resetting the display and setting all the object tags back to zero.

Change the player indicator variable after each move has been made and evaluated.

Tic Tac Toe Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour 30 minutes

OBJECTIVES:

- Understand control arrays and how they differ from individual controls
- Understand IF statements using arrays
- Use Boolean variables as flags

OVERVIEW OF PROJECT:

Create a version of Tic Tac Toe that allows two players to play against each other. The game should report winners, losers, and draws. The program must not allow illegal moves or additional moves after a game has been won.

PROJECT INSTRUCTIONS:

1. On a new form, create a label box.
2. Set the label box Tag property to 0.
3. Copy the label box and paste eight new copies on the form. Answer Yes when Visual Basic asks if you want to create a control array.
4. Add two buttons: One for exiting the program and one for clearing the board. Give each button an appropriate caption.
5. In the label click subroutine, write program code to verify the box is available. Hint: An available box has a tag value of zero.
6. If an open box is clicked on, display an X or O depending on which player made the move.
7. Set the labels tag to a new value to indicate which player (1 or 2) moved into that box.
8. After each move, check to see if there is a winner or a drawn game. Report that the game is over if there is a winner or a draw. If there is a winner, no additional moves should be allowed.
9. If the clear game button is pressed reset all the tags to zero and the label captions to blank.
10. If the exit game is pressed, shut down the game.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

Assuming the label box control array is named Label1 and the boxes in the array are arranged from 0 in the top left to 8 in the bottom right corners the following IF statement will identify a win across the top of the board.

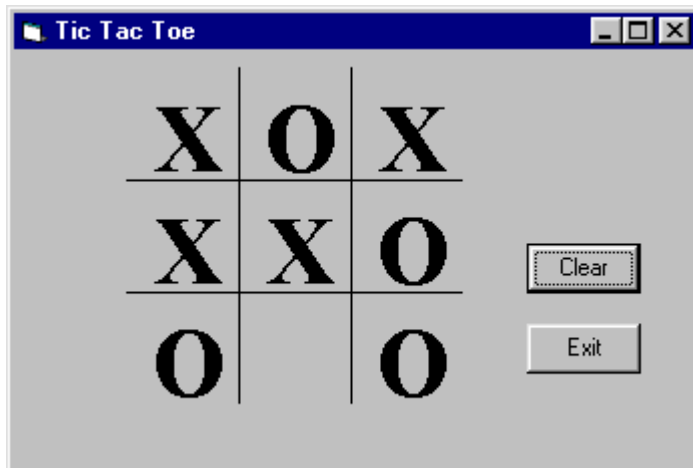
```
IF Label1(0).Tag <> 0 and Label1(0).Tag = Label1(1).Tag and Label1(1).Tag = Label1(2).Tag then Winner = True
```

This will set the Boolean variable Winner to True if the top three boxes have the same **non zero** value. The value of Winner should have been set to False before the checks for a winner are made. Once all eight possible winning combinations are checked, the program can take appropriate action if Winner is True.

The value of a Player flag indicates which player to report as the winner and must be switched after each move.

Checking the tags in all nine label boxes can identify a draw. If all the tags are non zero the game is a draw.

A completed game form may look like the following:



PROJECT EXTRAS:

Use the Beep statement to notify a user when they click on an occupied box.

Display a count of the number of draws and the number of times each player wins a game.

Have the game board clear automatically once the game is over.

Picture Display Program Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use the common dialog box

SKILLS NEEDED:

- Basic understanding of object properties

MATERIALS NEEDED:

- Visual Basic
- Graphic editor such as Paintbrush
- Picture files

TEACHING SUGGESTIONS:

The common dialog box is one of the more useful objects provided with Visual Basic. The common dialog box may be used for opening and saving files, setting print options, and selecting colors and fonts.

Setting the filter for the open dialog box will probably give students the most trouble. Go over several examples and point them to the help file for reference. Students will have a tendency to define more dialog properties than necessary.

If students are using several versions of Visual Basic, be aware that different versions invoke the common dialog box in different ways. Versions 3 and 4 use the Action property. Version 5 and version 6 use the ShowOpen method. The help files in all versions have useful examples.

While the picture box has an auto size property, the form size does not automatically resize to hold objects in it. The size of the form may be changed under program control. The objects on a form may also be moved to accommodate changes in the size or number of other objects on a form. Objects on a form should have some distance between them. Explain that students need to make room for borders when they move objects around the form.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should open the common dialog box with the proper filter for picture files enabled. The file requested by the user should be displayed full size. The

form should be resized to include the whole picture. Objects on the form should be relocated as needed so that the picture does not cover them.

SUGGESTED SOLUTION:

Create a picture box in the top left-hand corner of the form. Be sure to set the auto size property to true so that the box will expand or collapse to fit the picture loaded. Add a common dialog object. Create boxes to display the height and width of the picture with appropriate labels for each box. Create an Open file menu option under a file menu. In the open menu click routine, assign a filter string to the filter property of the common dialog box. Once a file has been selected, load the file into the picture box.

Call a routine to adjust the size of the form to hold the picture. Also, move the display boxes as required so that the picture box does not obscure them. Lastly, copy the pictures height and width into the appropriate boxes.

Note: Visual Basic version 5.0 and 6.0 picture support includes bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF files, and JPEG (.jpg) files.

Visual Basic version 4.0 picture support includes bitmaps (.bmp), icons (.ico), run-length encoded files (.rle), and metafile (.wmf) files.

Picture Display Program Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use the common dialog box

OVERVIEW OF PROJECT:

Create a program that opens and displays a picture. The program will also display size information about the picture. This program may be used to determine the size of pictures in twips so that picture boxes may be created for them in other programs.

PROJECT INSTRUCTIONS:

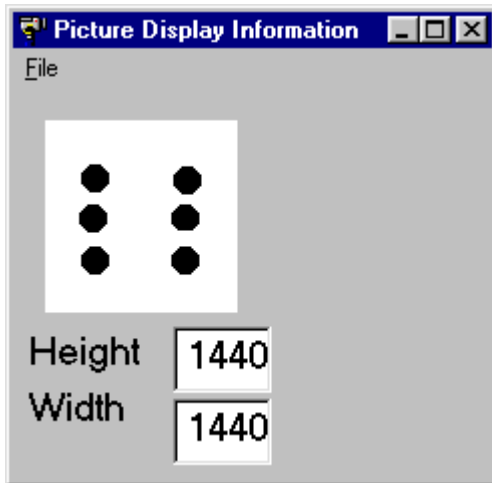
1. Create a picture box with the auto resize property set to true.
2. Set the filter and initial directory properties for a common dialog box.
3. Load the picture file selected by the open dialog box into the picture box.
4. Adjust the size of the form, if necessary, to hold the whole picture. Move any objects covered by the picture.
5. Copy the picture height and width information into properly labeled objects.
6. Program a way to cleanly exit the program.

ADDITIONAL RESOURCES:

- Textbook
- Graphic editor such as Paintbrush
- Picture files

SUGGESTED SOLUTION:

One possible solution would look like this.



PROJECT EXTRAS:

Display the full name and path of the file opened.

Convert the size in twips into inches for a second display.

Define different filters so that a user will see just one file type at a time.

Multiple Choice Grading Program

Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Read and parse a sequential data file

SKILLS NEEDED:

- Understanding of loops
- Understanding of string manipulation functions
- Understanding of decision making statements

MATERIALS NEEDED:

- Visual Basic
- Data file with names and answers

TEACHING SUGGESTIONS:

Reading data in from a sequential file has many benefits for both students and instructors. Tops among them may be easy reproducibility for testing purposes. Once a set of data is developed and tested, it can be used to prove other programs by comparing results with what is expected. For this reason, as well as to avoid problems caused by faulty data sets, having all students use a common data file is generally desirable.

Do While loops are a natural tool for reading sequential data files. Introduce the concept of end of file markers and the EOF built-in function. Make students aware that if they want to know how many records they have read, they have to keep track of record numbers themselves. Some students will be made aware of how many records are in a test set and write programs that use the FOR-NEXT loop which they may find easier. Their lives will be complicated if they their program will be evaluated using a second, different, file then the file they use for development. This has the advantage of more accurately simulating real word conditions. In the real world, programmers can rarely count on knowing the size of the data set their programs will use.

Students should have been exposed to the notion that a lower case letter is not equal to an uppercase letter in Visual basic by this time. They may need a reminder at this time. Mixing the case of answers in the data set or using uppercase in the main data set and providing a lower case answer key will show them the need to match cases. The Lcase and Ucase built-in functions will prove useful here.

Be aware that some students will use parallel picture boxes to create columns. If the font attributes are not the same, those boxes may have alignment problems. They must also be sure to clear the right picture boxes at the right times.

RESOURCES:

- Textbook
- Text editor for creating a data file

SUGGESTED EVALUATION:

A complete program should:

- Read and display all names in the data file
- Properly score each students set of answers
- Display the proper grade for each student
- Calculate and display the average number of correct answers for the whole data file.
- Exit the program cleanly

SUGGESTED SOLUTION:

Prepare an ASCII text data file for students to use. The file should have at least 10 entries.

The program should use a Do While loop checking on the state of End of File (EOF) to read the data file. The program should count each student as it reads its record. A variable should be used to add the total correct answers for each student. Use this variable with the student counter variable to calculate the average number of correct answers. Displaying the average as part of the same routine that counts students and adds totals avoids the need for form level variables.

Care must be taken when determining when to open and close the data file. Trying to open a file that is not closed, or trying to read a file that is not yet open or has been opened and closed, are common errors. For this reason, it is recommended that the open and close be done in the same routine. Alternately, the file should be opened once in Form Load and not closed until the exit program routine is called.

Multiple Choice Grading Program

Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Read and parse a sequential data file

OVERVIEW OF PROJECT:

Create a program to correct and grade a set of multiple choice test results. Read a set of correct answers, from a data file or a text box depending on what your instructor has specified, and compare it with the answers in a data file. The data file will have two fields for each student. The first field will contain the student's name. Second field will contain a set of answers to a multiple-choice test. For each student, compare the answer key answers to the answers from the student's record. Keep a count of the number of answers from the student's record that match the answers in the answer key.

For each student, display the student's name, the number of correct answers, and a letter grade. Calculate the letter grade scale based on information provided by the instructor.

Calculate the average number of correct answers for the class and report it in a separate picture box.

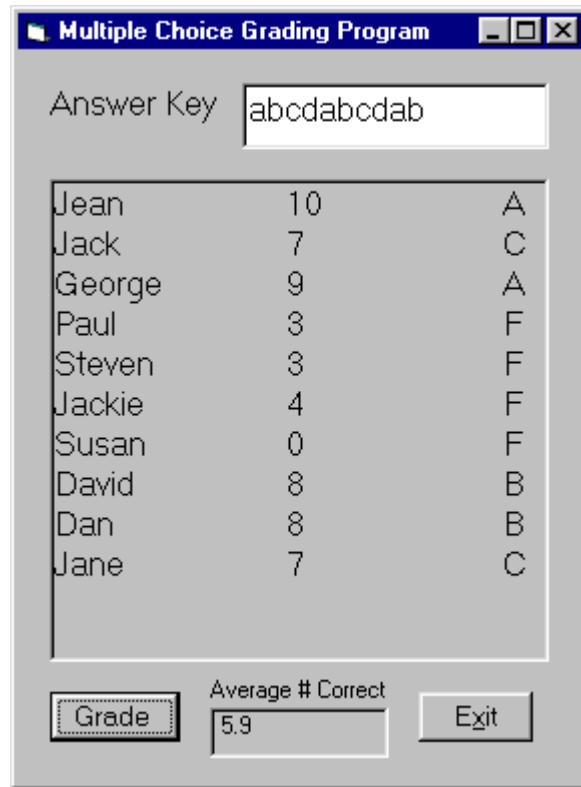
PROJECT INSTRUCTIONS:

1. Create a picture box large enough to display the names, correct count and letter grade of each student.
2. Create code behind a command button to read the data file and report results
 1. Open the data file
 2. Read the answer key
 3. Read a students record
 4. Compare student's answers to answer key and count correct answers
 5. Display student's name, number correct and letter grade
 6. Count number of students and build running total of correct answers
 7. Loop until all student records have been read and processed
3. Calculate and report average number of correct answers
4. Create an exit button with the appropriate code to end the program

ADDITIONAL RESOURCES:

- Textbook
- Instructor supplied data file

SUGGESTED SOLUTION:



PROJECT EXTRAS:

Report the average letter grade as well as the average number of correct answers.

Keep track of the highest and lowest scoring students and report them in a separate box.

Multiple Choice Grading Program with User Defined Types

Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Use a User Defined type

SKILLS NEEDED:

- Understanding of sequential files
- Understanding of arrays
- Understanding of string manipulation functions
- Understanding of decision making statements

MATERIALS NEEDED:

- Visual Basic
- Data file with names and answers

TEACHING SUGGESTIONS:

User defined types allow the programmer to group related data elements and manipulate them as a single item. They are very often used to define record types for files but may be used in other ways as well. In this case we want students to group names, as read from a data file, and number of correct answers, as calculated by the program.

User defined types may only be declared, or described, in a separate module file. Students will have to be instructed in the proper way to insert a new module for their version of Visual Basic. Be sure to remind students to save this file. It will be a third type of file, after a project and form file. User defined types are global program wide and not just available at the form level. Students who wish to create programs with multiple forms will want to know that variables declared in the module file will be available between forms as form level variables are not.

We will also want the program to rate each student in the data file based on information that it will not have until all data records have been read. Creating an array of a user defined type allows us to manipulate the data without having to read the file a second time.

While the rating of students as average, above average, or below average may be done in the same routine as the initial reading of the data file, having students code a separate routine reinforces the use of local as opposed to form level or global variables.

RESOURCES:

- Textbook
- Text editor for creating a data file

SUGGESTED EVALUATION:

A complete program should:

- Read and display all names in the data file
- Properly score each students set of answers
- Display the proper grade for each student
- Calculate and display the average number of correct answers for the whole data file.
- Rate all students as having an average, above average, or below average number of correct answers.
- Exit the program cleanly.
- Make proper use of form level and local variables.

SUGGESTED SOLUTION:

Prepare an ASCII text data file for students to use. The file should have at least 10 entries.

The student record array should be defined at the form level to allow its use across a number of routines. A form level variable should also be defined to hold the count of elements in the array that is being used. An alternative solution would be for the name after the last used to be set to some known flag value.

The program should use a Do While loop checking on the state of End of File (EOF) to read the data file. The program should count each student as it reads its record. This counter variable should also be used as an index into the array of student records. A variable should be used to add the total correct answers for each student. This variable will be used with the student counter variable to calculate the average number of correct answers. This count should be entered in a field of the student record array.

Care must be taken when determining when to open and close the data file. Trying to open a file that is not closed or trying to read a file that is not yet open or has been opened and closed are common errors. For this reason, it is recommended that the open and close be done in the same routine. Alternately, the file should be opened once in Form Load and not closed until the exit program routine is called.

Multiple Choice Grading Program with User Defined Types Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Use a User Defined type

OVERVIEW OF PROJECT:

Create a program to correct and grade a set of multiple choice test results. Read a set of correct answers, from a data file or a text box depending on what your instructor has specified, and compare it with the answers in a data file. The data file will have two fields for each student. The first field will contain the student's name. Second field will contain a set of answers to a multiple-choice test. For each student, compare the answer key answers to the answers from the student's record. Keep a count of the number of answers from the student's record that match the answers in the answer key.

For each student, display the student's name, the number of correct answers, and a letter grade. Calculate the letter grade scale based on information provided by the instructor. Store the student's name and the number of correct answers in an array of a user defined type.

Calculate the average number of correct answers for the class and report it in a separate picture box.

Using the array of student records, display a list for all students. Report if a student had an average, above average or below average number of correct answers.

PROJECT INSTRUCTIONS:

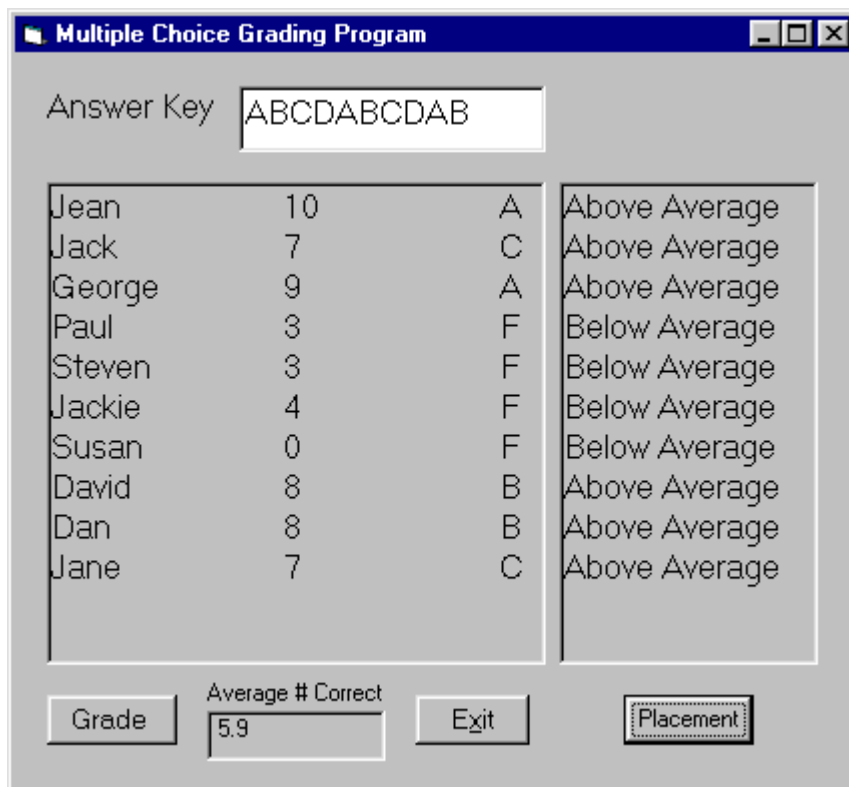
1. Create a user defined type to hold a string name field and a numeric count field.
2. Create a picture box large enough to display the names, correct count and letter grade of each student.
3. Create code behind a command button to read the data file and report results
 1. Open the data file
 2. Read the answer key
 3. Read a students record
 4. Compare student's answers to answer key and count correct answers
 5. Display student's name, number correct and letter grade
 6. Count number of students and build running total of correct answers

7. Loop until all student records have been read and processed
4. Calculate and report average number of correct answers
5. Create code behind a command button to loop through all student records and display if each student has an average, below average or above average number of answers correct.
6. Create an exit button with the appropriate code to end the program

ADDITIONAL RESOURCES:

- Textbook
- Instructor supplied data file

SUGGESTED SOLUTION:



PROJECT EXTRAS:

Report the average letter grade as well as the average number of correct answers.

Keep track of the highest and lowest scoring students and report them in a separate box.

Check Book Balancing Program Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Read and write to a random access file

SKILLS NEEDED:

- Understanding of object properties
- Understanding of looping constructs.

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Some of the vocabulary in this project may need explaining if some students are unfamiliar with checks and balancing checkbooks. Specifically, you will want to make sure they understand what a check clearing means.

If the check box object has not been introduced to students, this program has a natural use for it. Students often and incorrectly assume that the check box value property is a Boolean. Since this property has three options (checked, unchecked and grayed) they will have problems if they try to assign it directly to a Boolean variable.

Writing random access files require a user defined type. This type must be defined in a separate module (.BAS file). Be sure to explain how to add and save a module. Some students have trouble with the concept of a user-defined type. Often a student will attempt to use the type name as a variable rather than declaring a new variable of that record type.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should accept an unlimited number of checks. It should properly record the information for each check. The program should read the entire data file when balancing the checkbook. When balancing the checkbook, the program should properly recognize and handle both cleared and uncleared checks. The

program should display the correct new balance for the account. All uncleared, and only uncleared, checks should be displayed.

SUGGESTED SOLUTION:

Students have a number of options regarding where in their program to open and close the file. Some options are more efficient than others. Many will automatically open the file, write a record, and immediately close the file. While this is very inefficient, it is not necessarily wrong. It does have some advantages in a learning situation. This solution forces the student to be careful about when they open, read and close files. Attempting to read or close a closed file, or open an open file will cause errors.

The most efficient solution will open the file at form load and close it only at program exit. Using random files makes this very easy. Use a global variable to keep track of the current number of records in the data file.

The write check command, implemented in a command button for example, should build an output record by copying information from text boxes. Convert the check number and amount to appropriate number values. Numeric values take less storage space than string values. This will also avoid having to make these conversions whenever arithmetic operations are made. An If block should determine the status (cleared or uncleared) of the check and save this information as a Boolean value. Increment the record counter and use it to write the record to the end of the file.

If a solution involving repeated opens and closes of the file is selected, use the size of the record and the size of the file to determine the number of records in the file if needed. For example, assuming an integer variable called LastCheck is used as the record counter and the variable Check is of the record type defined for the file, the following code will return the current number of records in the file.

```
LastCheck = LOF(1) / Len(Check)
```

LOF is a built-in function that returns the size of a file in bytes. Len returns the length of the variable Check. There are several advantages of using the Len function. It spares the programmer the need to know exactly the storage requirements for each variable type. It also automatically keeps the program correct if the record length is changed at a future time.

Balancing the checkbook involves accepting and initial balance from the user. The beginning balance is often entered into a text box but an input box may also be used.

Once the initial balance has been determined, read the data file. Use a For loop, using the loops counter to indicate record number, to read each record. The count of records in the file is the loop terminator. Inside the loop, the cleared indicator for each record read is checked. Display uncleared checks, perhaps using a print method to a picture box. Subtract the amount of any cleared checks

from the current balance. Redisplay the current balance either every time it changes or after the read loop completes.

Close the data file on program exit, if it has not been previously cleared.

Check Book Balancing Program Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Read and write to a random access file

OVERVIEW OF PROJECT:

Create a program that allows a user to balance a checkbook. Accept information about checks from the user. For each check, accept the number of the check, the payee of the check (who is the check made out to), the amount of the check, and whether or not the check has cleared the bank.

Enter several checks and then balance the checkbook. Balancing the checkbook consists of accepting the initial balance of the checkbook and then calculating the current balance of the account. Reading the check information from the data file supplies the information needed to balance the checkbook. Subtract the amount of any check that has cleared from the beginning balance. Display the number, payee and amount of any check that has **not** cleared.

PROJECT INSTRUCTIONS:

1. Create a form to accept the number, payee, and amount of a check. Also, find out if the check has cleared.
2. Add a command button, or other object, to write the check information to a file when the user is ready.
3. Add a second form or second section to the main form to accept the starting balance, display the current balance, and display the number, payee and amount of checks outstanding.
4. Add a command button, or other object, to balance the checkbook using the information provided on the form and read from the data file.
5. Add a way to exit cleanly from the program, making sure the data file is closed properly first.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

Consider using a check box for the user to indicate if the check has cleared. Checking the numeric value of a check box is often easier than parsing a textual answer from a text box.

A completed program might look something like this:

The screenshot shows a window titled "Checkbook Balancing Program". It features a menu bar with "File". The main area is divided into sections. The top section has "Check #" (230) and "Payee" (Mr. John Doe) with a checked "Cleared" checkbox. Below is "Check Amount" (275.00) and a "Record" button. A horizontal line separates this from the next section, which has "Starting Balance" and "Current Balance" fields and a "Balance Checkbook" button. The bottom section is labeled "Checks Outstanding" and contains a large empty rectangular area.

PROJECT EXTRAS:

Use the Format function to format all money amounts as currency when they are displayed.

Add error checking to make sure that the user can not enter non-numeric values into text boxes accepting money amounts.

Use several forms for this project. Perhaps one form to select what operation to follow, one to enter checks and one to balance the checkbook.

Give the user the option to clear the data file or to use an existing file.

Give the user the option to select a data file to open or create.

Drawing Program Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Use the load command to add elements to a control array
- Use the circle and line methods to draw objects on a form

SKILLS NEEDED:

- Understanding of arrays
- Understanding of global variables
- Understanding of control structures (IF or SELECT CASE)

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Managing large numbers of elements in a control array can be a tedious chore. This is especially the case if objects are relocated or have important properties changed. The Load statement allows a program to add elements under program control. By using the size and location of the original object, the program may place the additional objects in appropriate relationship to each other. A loop controlling the loading of additional objects can also be used to set other properties guaranteeing that settings are made in the right order.

The value of the index variable indicates which member of a control array is selected. This value is normally lost once the item click routine is exited but a global variable may be used to save these values for use in other routines. This allows the programmer to use a control array as a means of selecting options.

Global variables may also be used to keep track of related pieces of information. For example, using global variables to keep the X and Y coordinates registered by a mouse down event allows the programmer to use them with the X and Y coordinates registered by a mouse up event. Having both sets of coordinates allows the program to set the beginning and ending of lines or boxes to draw.

The initial object of a control array must have its index value set for the Load statement to be able to add elements to the array. Forgetting to set an index value is a common programming error. Objects created by the Load statement are invisible and in the same location as the original object. The programmer must remember to set the object visible and move the object to an appropriate location after loading.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete drawing program should:

- Create a control array, loaded and initialized with different background colors inside a loop.
- Allow a user to select colors to use for drawing objects
- Allow a user to select one of at least 4 object types to draw
- Draw selected objects using the selected color at locations indicated by the user
- Clear the form on a users request
- Exit the program cleanly

SUGGESTED SOLUTION:

Use global variables to track the current color to use, the object to draw and the beginning location selected by a user.

Two control arrays created can indicate the object to be drawn and the color to use to draw it. An array of picture boxes may be used to show the objects to draw. This array should display an appropriate picture or drawn object as a label. A second array of objects may be used to display and select colors. Sixteen picture boxes, or other objects, are set to display the 16 QB colors. This array may be easily loaded and initialized in a loop as in the following example.

```
Picture1(0).BackColor = QBColor(0)      ' Set the first square to black

For I = 1 To 15                          ' Start the loop to add 15 color
squares
    Load Picture1(I)                      ' Load a new picture box
    Picture1(I).Visible = True            ' Set the box visible
    Picture1(I).BackColor = QBColor(I)    ' Set the color of the box
    If I Mod 2 = 1 Then                   ' Odd numbered boxes are moved right
        Picture1(I).Top = Picture1(I - 1).Top
        Picture1(I).Left = Picture1(I - 1).Left + Picture1(I - 1).Width
    Else                                   ' Even numbered boxes are moved down
        Picture1(I).Left = Picture1(0).Left
        Picture1(I).Top = Picture1(I - 1).Top + Picture1(I - 1).Height
    End If                                 ' End odd/even check
Next I                                     ' end the loop that builds the color chart
```

Students may use any number of object types in their programs. Check boxes or label boxes may also be used. Any object that may be selected and which may be used to display a color or an object may be used.

Lines and rectangles are easily drawn using the **Line** statement. Circles and other objects require a bit more creativity. Triangles will generally start with coordinates selected at mouse down and ending with mouse up. The type of

triangle and the location of the third corner may be selected by a selection on the form or indicated by the pressing of a control, alt or shift key when the mouse is released. Students should be encouraged to be creative in both the objects supported by their program and the implementation of those objects.

Drawing Program Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Use the load command to add elements to a control array
- Use the circle and line methods to draw objects on a form

OVERVIEW OF PROJECT:

The object of this program is to create a drawing program that allows a user to select colors to use and objects to draw. The user will select a colored object to indicate which color to use for drawing. A second set of objects will display objects the program will draw for the user. After selecting a color and an object, the user will use the mouse to show the program where to draw the object.

The user must have at least four objects to draw.

Create the first element of a control array to display colors at design time. Additional elements of the control array must be loaded at run time. The display properties of the array should also be set under program control.

PROJECT INSTRUCTIONS:

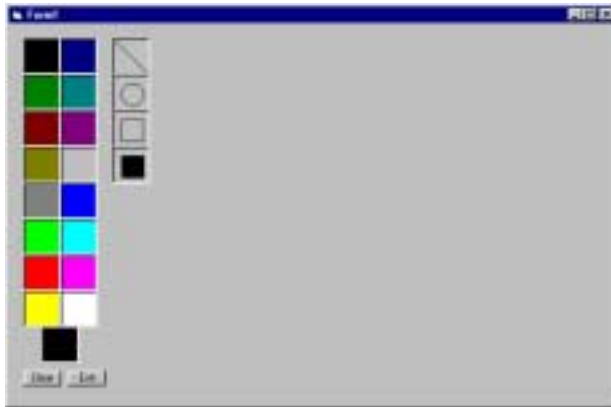
1. Create a small box in the top right hand corner of a form. Set its index value to 0 so that a control array may be created.
2. Program a loop to run at form load time to load additional elements of the control array.
3. Set the background colors and locations of the objects, as they are loaded.
4. Create a second set of objects to indicate shapes that the program will draw.
5. Create global variables to store the shapes and colors selected by the user. The user will click on an element in the appropriate control arrays to choose a shape or color.
6. Create program code on the form to draw objects of the selected shape and color in response to mouse actions.
7. Create an exit button or exit menu item with the appropriate code to end the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

In the solution below, the 17th square at the bottom of the color array shows the currently selected color. Change the background of that square when a new selection is made.



PROJECT EXTRAS:

- Create additional shape options for the user.
- Create the color and/or shape selection objects on their own forms.
- Change the color of the shapes to match the currently selected color

Roman Numeral Conversion Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3-4 Hours

OBJECTIVES:

Use complex decision making structures

SKILLS NEEDED:

- Understanding of decision making structures
- Understanding of string handling functions

MATERIALS NEEDED:

- Visual Basic
- Roman numeral conversion chart

TEACHING SUGGESTIONS:

This is as much logic puzzle as a programming problem. Occasionally a student will suggest a single If statement for each number. The prospect of 4,999 statements doesn't always daunt them. Reminding them that the program must convert both ways resulting in 9,998 lines will usually convince them that this is not a practical solution.

The number 4,999 was chosen for a practical reason. Roman numerals at five thousand and above involve characters with lines above them. Those characters are not in the ASCII character set. Four thousand nine hundred ninety nine is a number large enough to provide a challenge to most students.

Encourage students to think out how they convert these numbers by hand. When converting from Arabic to Roman, most people use a method involving subtracting the highest possible number associated with a symbol. That symbol is set aside and the next symbol is examined. This continues until no more symbols are needed and a Roman numeral has been built. This suggests several possible algorithms.

Converting from Roman to Arabic is even easier. Evaluate the string from left to right converting each symbol to its numeric value and adding it to a counter. The tricky part is handling cases of leading symbols whose value is subtracted from its larger neighbor. Compare each symbol with the symbol to its right to determine the appropriate number.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Verify that the program works for a wide range of numbers. The number 1994 and 4999 provide good tests as they involve a number of leading subtraction symbols. Verify that a number converted in one direction results in the same number when converted back.

SUGGESTED SOLUTION:

Create two text boxes to accept numbers to convert. Below each, create a command button labeled for the appropriate conversion.

Converting from Arabic to Roman

Use the Val function to copy the number in the text box into an integer variable. If the number is greater than or equal to 4,000, subtract 1000 from the number and append an M character to the string used to build the Roman numeral. Do the same for 3,000 and on down to 1,000. Compare the number to 900, if still greater then subtract 900 and append CM to the end of the string. Continue the same pattern to 500, 400, 300, 200, 100, 90, 50, 40, 30, 20, 10, 9, 5, 4, 3, 2, and 1. At this point, the whole number will be accounted for and the program should display the result.

Converting from Roman to Arabic

Initialize a counter variable. Also, initialize a string variable that will be used to hold the previous character for characters after the first. Start a loop to examine each character in the string entered into the text box. If the letter is an I add one to a counter. If it is a V, add 5 if the previous character was not an I. If the previous character was an I then that one should have been subtracted rather than added. Compensate for that addition by adding 3 to the counter rather than 5. Follow this format for the remaining characters up to M for a thousand. Remember that there is only one possible character to subtract before each symbol. The I comes before X (10) as well as V (5). The X comes before L (50) and C (100). The C comes before both D (500) and M (1000).

After evaluating all characters, display the resulting number in the appropriate box.

Roman Numeral Conversion Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3-4 Hours

OBJECTIVES:

- Use complex decision making structures

OVERVIEW OF PROJECT:

This program will convert regular numbers into roman numerals and convert Roman numerals into regular numbers. The program will read in a number or string of characters as appropriate. The user will indicate which conversion is to take place. The number will then be converted and displayed properly. The program must correctly convert numbers up to 4,999.

PROJECT INSTRUCTIONS:

1. Create two text boxes on a form. Label them.
2. Create two command buttons. One to convert an Arabic number into a Roman numeral. The second, to convert a Roman numeral into an Arabic number.
3. Place the converted number in the appropriate box.
4. Add an option to exit the program.

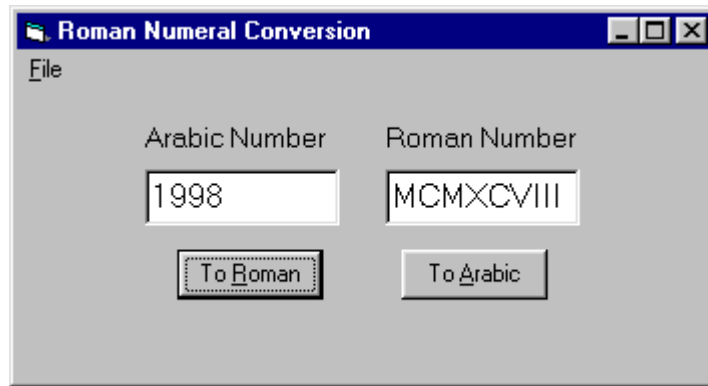
ADDITIONAL RESOURCES:

- Textbook
- The following Roman numeral conversion chart

1	I	40	XL
2	II	50	L
4	IV	90	XC
5	V	100	C
6	VI	500	D
9	IX	900	CM
10	X	1,000	M

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

Numbers over five thousand require symbols with lines above them. Create picture files that contain individual characters including the characters at 5,000 and above. Use these picture files to display numbers greater than 5,000.

If the user enters Roman numerals in lower case, redisplay them as upper case.

Palindromes Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 2-3 Hours

OBJECTIVES:

- Advanced string manipulation

SKILLS NEEDED:

- Understanding of loops
- Understanding of string manipulation functions

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

Extra characters, characters outside the range of characters we are interested in, and differing case of characters are the biggest obstacle in determining palindromes. Be sure that students are aware of what characters and characteristics of a character matter.

Review the `Mid$` and `Ucase$` functions before this project. Consider reviewing functions if necessary. Functions make this project much easier.

Valid test data is a necessity for any program. Consider supplying good test data to students to ensure that they are able to test their program effectively. If you do not supply test data, emphasize the importance of selecting good test data.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Test a number of strings with known solutions to ensure that all cases are found. The classic palindrome "Madam I'm Adam" makes a good test of a longer palindrome. In the sentence "Hello Madam, I'm Adam and this is my friend Bob", the following sub-strings of length 3 are palindromes:

ada - from the word "Madam"
mim - from the words "Madam, I'm"
ada - from the word "Adam"
ama - from the words "Adam and"

isi - from the words "this is"
sis - from the words "this is"
bob - from the word "bob"

SUGGESTED SOLUTION:

Removing extra characters is the first step. Use the Ucase\$ function to set the string in the input text box to all upper case. Use the Len function to determine the length of the string and set-up a loop that examines each character. Concatenate every valid character, between A and Z or 0 to 9, to a string variable. This will discard the characters outside the valid range.

Create a Boolean function to evaluate a string as a possible palindrome. Initialize one counter to one. Initialize a second counter to the length of the string. Use these counters to compare the first and last character. If they are the same, increment the first counter and decrement the second. If they are not the same, set a flag indicating that the string is not a palindrome. Continue comparing characters until the center of the string is reached or a mismatch is found. If the center is reached without a mismatch, return True as the value of the function. If a mismatch is found, return False as the function value.

If no sub-string length is entered, submit the whole input string to the check palindrome function. Display the appropriate message based on the return from the function.

If a sub-string length is entered, create a routine to select sub-strings and send them to the check palindrome function. The loop will start at the beginning of the string and increment up to the length of the string minus the one less than the length of the sub-string length. Use the Mid\$ function to select a series of sub-strings of the requested length. Send these sub-strings to the check palindrome function. If a sub-string returns True, display the sub-string in the picture box. If no palindromes are found, display an error message.

Palindromes Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 2-3 Hours

OBJECTIVES:

- Advanced string manipulation

OVERVIEW OF PROJECT:

A palindrome is a word or phrase that reads the same backwards as forwards. Some common examples are "Mom", "Dad", "Bob" or even the sentence "Madam, I'm Adam." When determining a palindrome, the case of letters is ignored, as is any spacing or punctuation if the phrase happens to be a sentence.

The goal of this program is to determine if a word or phrase is a palindrome and to determine if any sub-strings of words or phrases are palindromes. Case and punctuation are ignored. Only the 26 letters (a...z) and 10 numbers (0...9) are to be considered.

PROJECT INSTRUCTIONS:

1. Create a text box to accept the possible palindrome.
2. Create a second text box to accept the size of palindromes to find.
3. Create a picture box to display the palindromes found in the input text box.
4. Write code in a command button routine to check for palindromes.
5. If sub string palindromes were requested, display any palindromes found in a picture box.
6. Display a message box if not palindromes were found.
7. Create an end program option

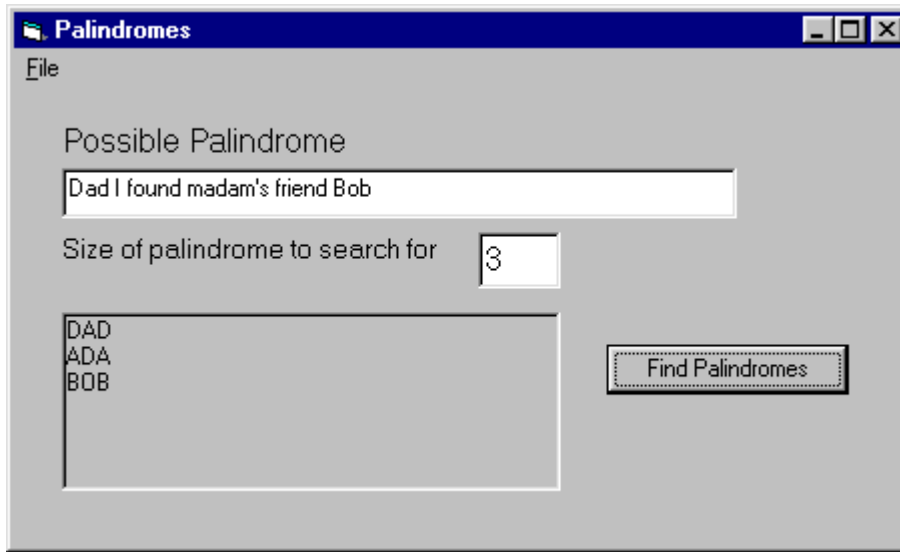
ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

Consider creating a Boolean function to examine a string and return True if the string is a palindrome. Examine the whole string by sending the contents of the text box to this function. Search for smaller palindromes by sending a number of sub-strings to the function.

A completed program might look something like this:



PROJECT EXTRAS:

Display a count of the number of palindromes found.

Conway's Game of Life Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 4 - 5 hours

OBJECTIVES:

- Loading and manipulating object arrays
- Evaluating complex conditions using nested IF blocks

SKILLS NEEDED:

- Understanding of object properties
- Understanding of Boolean operations

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

John Conway's Game of Life was one of the first "artificial life" programs. It remains one of the most popular life simulations for computer programmers. There are many sites on the World Wide Web devoted to it. You may want to suggest that interested students do some independent research on the topic. One of the things they may find are interesting patterns for use with the game. There are also a number of shareware and freeware versions of the game already written. These may serve as inspiration for more advanced or interested students.

The rules of the game (outlined in the student's section) are simple. The implementation can get involved. Students may come up with overly complicated solutions so you should review them before they get too far along in program development.

Explain the benefits of using constants for important variables to allow program growth and change. For example, by using constants to indicate the number of columns and rows the board may be expanded and contracted just by changing those values in one place. These constants are throughout the program.

Students may attempt to create all the cells needed manually. This should be discouraged as it often leads to serious problems in debugging, severely limits program expandability, and misses an opportunity to let the computer do more of the work. It does require more up front planning but that is probably more an advantage than disadvantage.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete Life program should:

- Allow a user to select initial cells to contain “life.”
- Generate and display a new life generation.
- Allow the user to clear the board
- Allow the user to exit the game.

SUGGESTED SOLUTION:

Checks for neighbors may be implemented several ways. One way, often chosen by new programmers, is to treat corner cells, top and bottom rows, and left and right columns as special cases. This adds considerable complexity and creates many additional opportunities for errors.

Creating a sort of neutral zone around the edge of the board allows the programmer to treat all cells identically. The top and bottom rows and left and right columns are initialized as empty cells and rendered invisible (Visible property set false). This means those cells will never be changed or have life in them. The programmer can set up a simple set of If statements to check all cells.

In the code example below, MaxCol is a form level constant that indicates how many columns the playing board has. Using MaxCol lets the program indicate adjacent cells relative to the cell being checked. CellNo indicates the cell being checked. LifeCell is a control array that is used to display the game board and track life entities. Neighbor is the variable used to keep track of how many neighbors (objects with a tag value of one indicating life) the checked cell has.

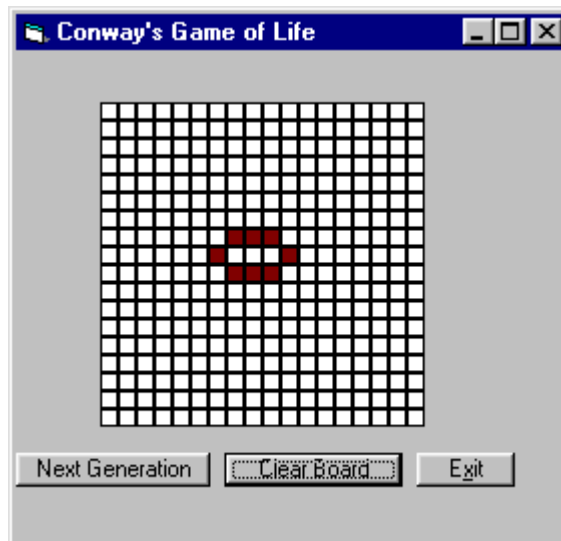
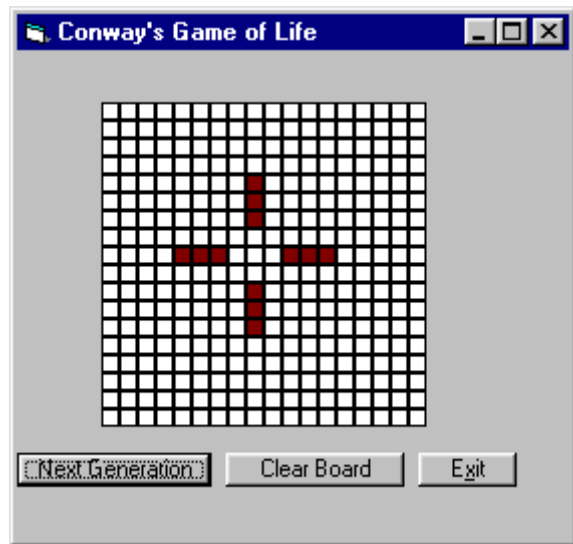
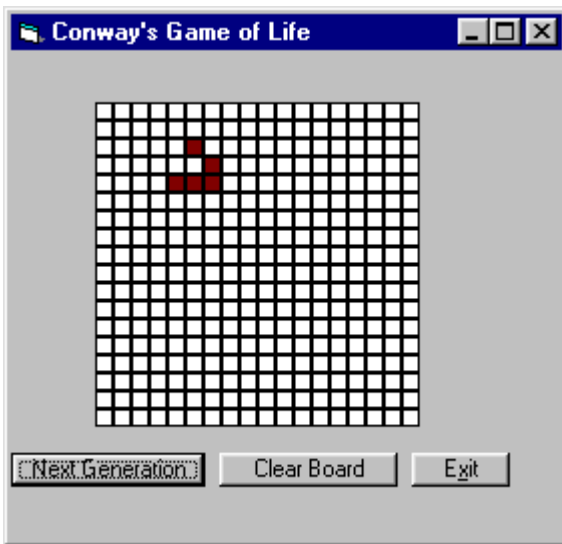
```
Neighbor = 0
If LifeCell(CellNo - 1).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo + 1).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo - MaxCol).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo - MaxCol + 1).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo - MaxCol - 1).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo + MaxCol).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo + MaxCol + 1).Tag = 1 Then Neighbor = Neighbor + 1
If LifeCell(CellNo + MaxCol - 1).Tag = 1 Then Neighbor = Neighbor + 1
```

Once the number of neighbors has been determined, the status of the current cell is easily determined. Students should consider writing a Boolean function to return the status of a cell. The function should accept the number of a cell to check. The function should return True if the cell should have a life in it for the next generation and False if it should not.

Students should also use an array separate from the array holding the current state of the board to hold the state for the next generation. Not using a separate array allows any changes made to affect the evaluation of other cells.

When a user sets or clears the life in a cell, the programmer must be careful to change both the visible indication of life (background color or picture) and the internal indication (object tag value, status array, etc.) to maintain a consistent state. Failure to do this results in undetermined results that may be hard to debug.

Here are a few interesting shapes to start. Students and instructor alike will easily discover others.



Conway's Game of Life Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 4 - 5 hours

OBJECTIVES:

- Loading and manipulating object arrays
- Evaluating complex conditions using nested IF blocks

OVERVIEW OF PROJECT:

John Conway's Game of Life was one of the first "artificial life" programs. It remains one of the most popular life simulations for computer programmers.

The game is played on a collection of cells. Each cell has eight neighboring cells. Each cell either is occupied by a "life" or is empty. The state of a cell for a new generation is determined by a few simple rules.

1. If a life has no neighbors or only one, it dies of loneliness.
2. If a life has four or more neighbors, it dies of overcrowding.
3. If a life has either two or three neighbors, it survives to the next generation.
4. If a cell has no life in it but has exactly three neighbors, a new life is born.

Create a program to allow a user to set up a game board with initial life values and generate new generations of the board.

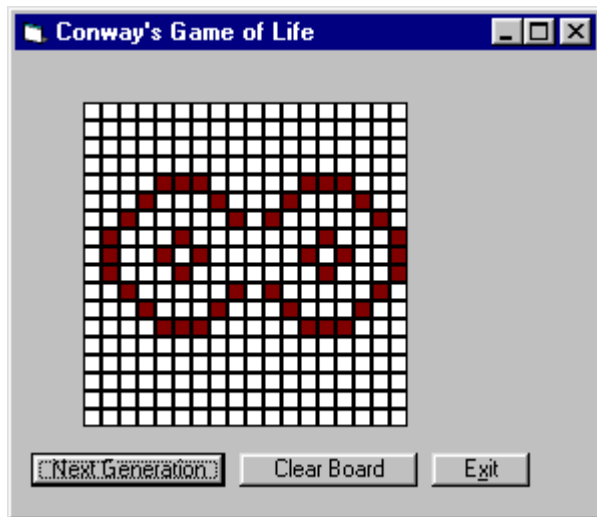
PROJECT INSTRUCTIONS:

1. Create an object to use to display a "life." Set the index property so additional elements of a control array may be loaded.
2. Create a board by loading new copies of the initial object in grid form.
3. Write code using the rules in the overview to determine which live cells "die" or "survive".
4. Write code using the rules in the overview to determine which cells have lives "born" into them.
5. Create a button to generate a new life generation using the code that determines life status.
6. Create a clear button with the appropriate code to empty all cells.
7. Create an exit button with the appropriate code to end the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:



PROJECT EXTRAS:

Add a button to have the program automatically generate and display several generations.

Use different colors for new life and life that has "survived" from the previous generation.

Allow the user to select a color for the life or background colors.

Use files to save, load, and edit interesting patterns.

Checkers Game Program Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 6-8 hours

OBJECTIVES:

- Use complex and nested decision constructs

SKILLS NEEDED:

- Understanding of looping constructs
- Understanding of variable and object arrays

MATERIALS NEEDED:

- Visual Basic

TEACHING SUGGESTIONS:

The game of checkers is familiar to most students. They will find programming the rules of checkers surprisingly complex. Have students write out the rules. Make sure they have them all listed before they start designing their program. A sample list follows.

1. Pieces at the top of the board may only move downward.
2. Pieces at the bottom of the board may only move upward.
3. Pieces may only move into empty squares.
4. Pieces may only move diagonally.
5. Pieces must move to an adjacent square or by jumping an opponent's piece to the next square past that piece on the same diagonal.
6. Remove jumped pieces from the board.

Encourage students to use the Load command to fill out a board by having the computer duplicate an original square object. The same loop that creates the board can be used to set the initial values for tracking piece locations. Students must give careful consideration to how they will check for valid moves. This is especially true for moves involving the top and bottom rows of the board. Encourage them to desk check their algorithms before implementation.

Review with students the advantages of nesting If statements. If they don't nest decision constructs, they will find that they need a more complex set of flags and indicators for this program.

Suggest to students that they provide some indication to the user of what piece they have selected for movement. They should also allow the user to deselect a piece if it has not been moved.

If students will be drawing discs on picture boxes using the Circle method, explain the paint event. The paint event occurs when part or all of an object is exposed after being moved or enlarged. A paint event also occurs after a window that was covering the object has been moved. If a Form_Paint routine is not written, students may find parts of their display missing if the form is covered or minimized at any time.

RESOURCES:

- Textbook
- Checker board and pieces to demonstrate the game and its rules

SUGGESTED EVALUATION:

Completed programs must follow all the rules listed in the teaching suggestions section. Be sure to test jumps and moves in a number of directions. Attempt to make moves into and out of the end rows.

SUGGESTED SOLUTION:

The first step is the creation of the playing board. Create one square using a picture box. Set the initial properties including an index value to allow loading new elements of a control array. Set the border style to none so that boxes will appear seamlessly next to one another. Set the Fill Style to solid so that objects drawn on the box will be filled.

To keep track of what boxes have what pieces on them, create an integer array. To facilitate using this array for determining valid moves at the ends of the board, this array will contain additional eighteen elements both below zero and above 63. In form load, initialize all elements of the array to -1, indicating that a player may not move into them. Create 63 more elements in the board square control array and move them into position. As squares are moved into position, change the BackColor of alternate squares in each row. Set the array holding contents of squares in odd numbered squares in the first 24 squares to indicate the first color. Set the odd numbered squares in the last 24 squares to hold the other color. Flag odd numbered squares in the middle of the board as empty.

Writing the code that colors the squares and initializes the board-tracking array as a function separate from the routine that loads the full control array. This allows that function to be called as a “New Game” routine.

Drawn objects in picture boxes may disappear if the board is hidden by another window. Create a form paint routine to redraw the board. This routine will examine the flag for each square and clear empty squares or draw the appropriate disk in occupied squares. Use this routine to draw the initial board by calling it from the board creation/initialization routine.

Draw disks on a picture box using the circle method. First, set the boxes fill color to match the color of the disk to be drawn. Then, use a previously determined x and y coordinate to draw a circle of that color in the box. Set form level variables for the circles center and diameter in the form load routine. Determine the center of the box based on half its height and width. Height and width should be the same. The radius of the disk should be half the width of a box minus some additional amount to leave a border.

Clicking on a square starts an important and involved set of events. Create a form level variable to hold a flag indicating if a piece has been selected for movement. Determining if a valid square has been clicked on requires a number of checks.

1. If a piece has already been selected, clicking on a square with a piece on it is not valid.
2. If a piece has not been selected, clicking of a square that is empty or has a piece of the wrong color, that is not a valid click.
3. If a piece has been selected and the square clicked on is flagged as an invalid square, not on the proper set of diagonals, that is not a valid click.
4. If a piece has already been selected and an empty square on the proper set of diagonals has been clicked check the following cases.
 - Is the square adjacent and in the right direction?
 - If the square is not adjacent, is there a piece owned by the opponent between the square the piece is on and the square selected?

Once a valid move has been selected, move the piece appropriately. Remove the disk from its original location by clearing the box and re-setting the flag in the board array. Draw a new disk in the new square and set the board array to indicate what color disk occupies it. If an opponent's piece was jumped, clear the square the piece was on and re-set the board array to indicate that square is now empty.

Switch a current player flag after each valid move. Turn off the piece selected flag if a player clicks on the currently selected piece to allow a player to deselect a piece. One way to indicate the currently selected piece is to redraw the disk a different, but related, color. Deselecting a piece implies that it must be redrawn in its primary color after it has been deselected.

This program serves easily as a base for writing a more complete checkers program.

Checkers Game Program Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 6-8 hours

OBJECTIVES:

- Use complex and nested decision constructs

OVERVIEW OF PROJECT:

Create a program that allows two players to play a simple form of checkers. Draw a 64 square board with alternating color squares. Place different colored discs on the dark squares in the top and bottom three rows. Allow the user to select a disc and move it to diagonally adjacent squares or to “jump” opponent’s discs on diagonally adjacent squares. Discs may only move to empty squares. Jumps may only be made over opponent’s discs. A piece (disc) that is jumped must be removed from the board. The program should refresh its display if the board is resized or temporarily hidden and unhidden.

Allow the user to reset the board. Resetting the board puts all pieces back on their original squares.

Allow the user to quit the game and exit the program.

PROJECT INSTRUCTIONS:

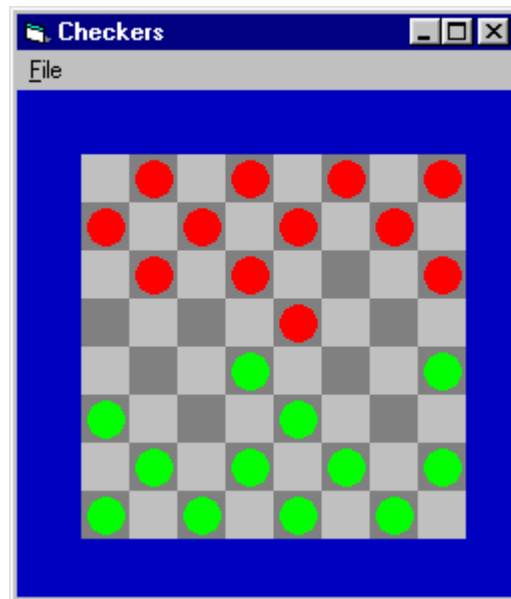
1. Write out a list of rules the program will enforce on piece movement.
2. Create an 8 by 8 board consisting of squares of different background colors.
3. Create a subroutine to populate the board with different color pieces as with a game of checkers.
4. Create code behind the object array representing the board squares to allow the user to select a playing piece.
5. Create code to allow the user to indicate a location to move a selected piece.
6. Create code to verify that a target square is a valid move according to the rules of the game.
7. If an opponent’s piece is jumped by a player’s move, remove that piece from the board and identify that pieces square as empty.
8. Create an option for the game to be reset to its initial set-up.
9. Write program code to redraw pieces on the board if a paint event occurs.
10. Allow the user to exit the program.

ADDITIONAL RESOURCES:

- Textbook
- Checker board and pieces to desk check algorithms

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

Allow the users to select the colors of:

- The discs used
- The squares on the board
- The form background

Determine the end of the game when there are no remaining moves.

Keep a running count of how many discs each player has on the board.

Make a more complete game of checkers by allowing pieces that reach the end of the board to become "kings" and move backwards. Provide an identification of kings on discs as appropriate.