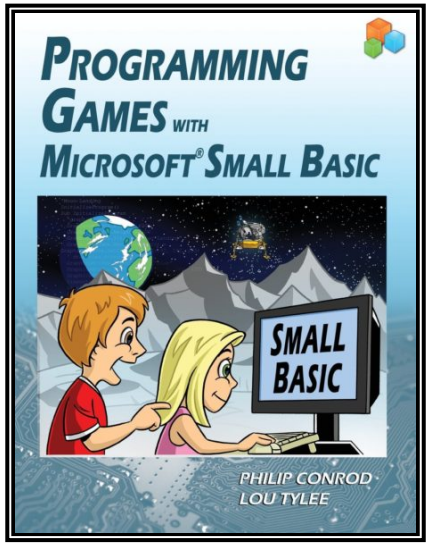
Programmeerspellen met Small Basic: Hoofdstuk 6: Tic Tac Toe-programma

Dit hoofdstuk is een bewerking van het boek Programming Games with Microsoft Small Basic van Philip Conrod en Lou Tylee.

Om dit boek in zijn geheel te kopen, zie de Computer Science For Kids website.



6. Tic Tac Toe Programma

Bekijken en bekijken

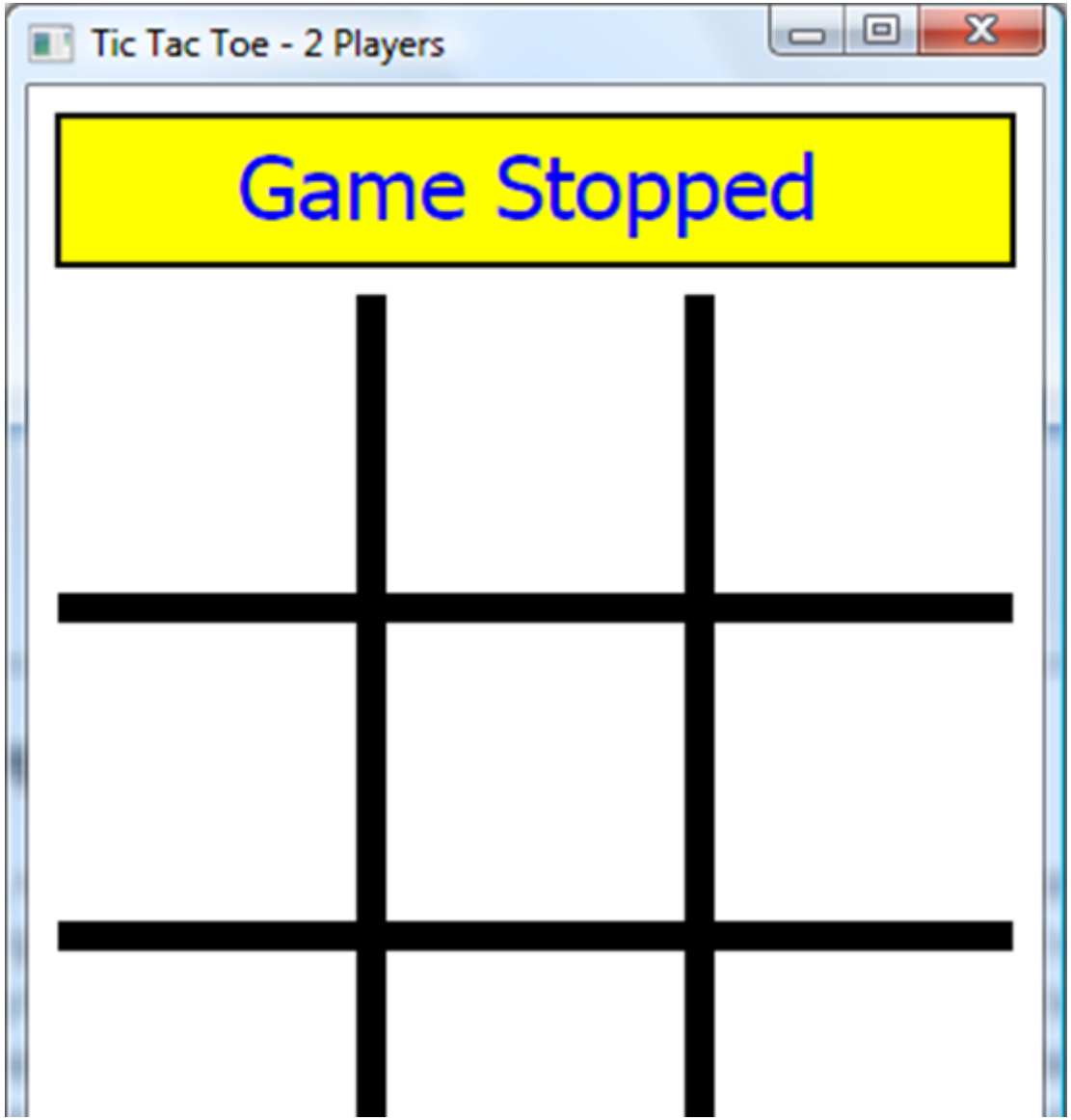
Het volgende programma dat we bouwen is het klassieke **Tic Tac Toe** spel, waarbij je 3 X's of 3 O's probeert op een rij te zetten in een 3 bij 3 raster. We ontwikkelen een versie voor twee spelers en een versie waarin je tegen de computer kunt spelen.

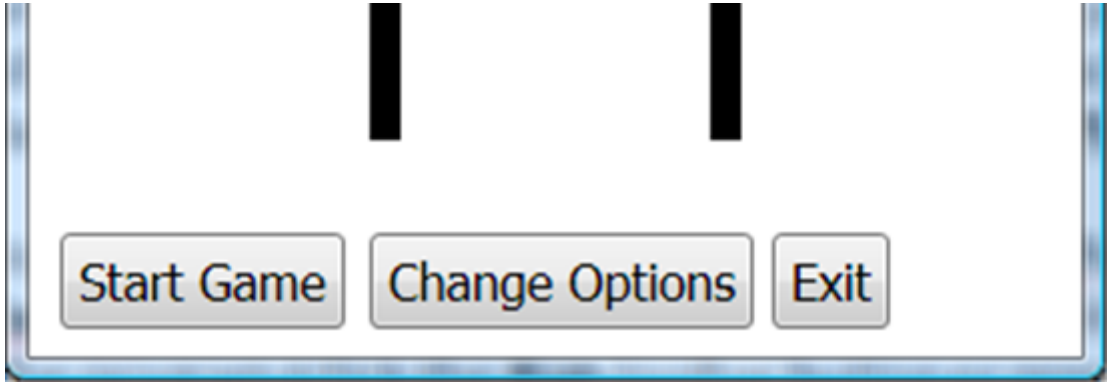
Tic Tac Toe Programma Preview

In dit hoofdstuk bouwen we een **Tic Tac Toe-spel**. Dit is naar verluidt het eerste spel ooit geprogrammeerd op een computer en een van de eerste ooit geprogrammeerd door Bill Gates toen hij een tiener was. Het doel van het spel is het opstellen van 3 X markers of 3 O markers in een 3 bij 3 raster. De markeringen kunnen horizontaal, verticaal of diagonaal lopen. Beurten wisselen af tussen spelers. De versie die we hier

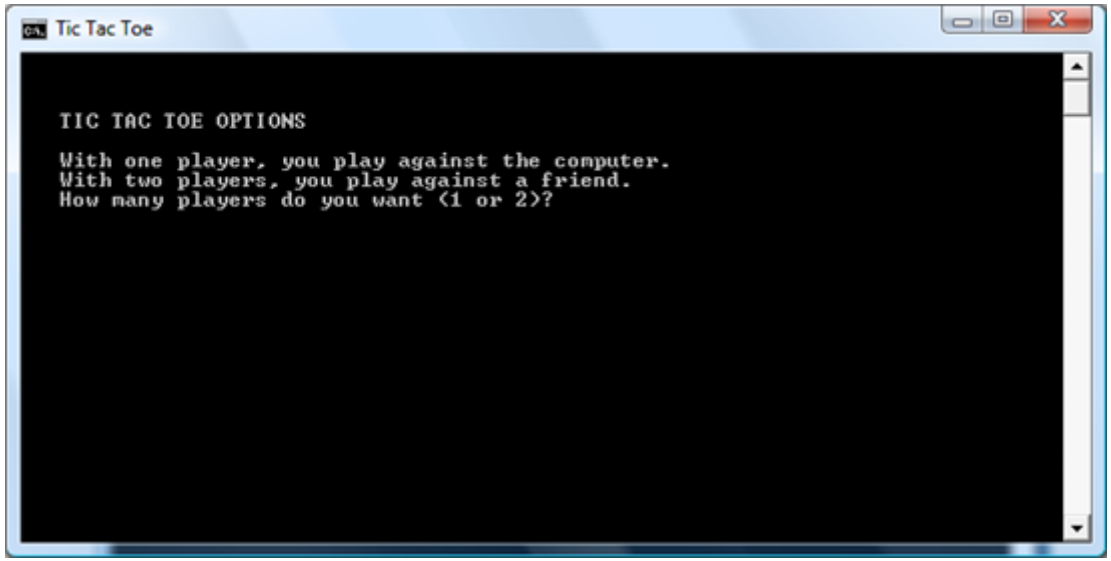
bouwen, stelt twee spelers in staat om tegen elkaar te strijden of om een enkele speler te laten concurreren tegen een behoorlijk slimme computer.

Het voltooide programma wordt opgeslagen als **TicTacToe** in de map **KidGamesSB\KidGamesSB Programs\TicTacToe**. Start Small Basic en open het voltooide programma. Voer het programma uit (klik op de werkbalkknop **Uitvoeren** of druk op <**F5**>). Het spel verschijnt in de status 'gestopt'. Op dit punt kunt u op de knop **Game starten** klikken om het spel te starten, op **Opties wijzigen** klikken om programma-opties te wijzigen of op **Afsluiten** klikken om het programma af te sluiten.

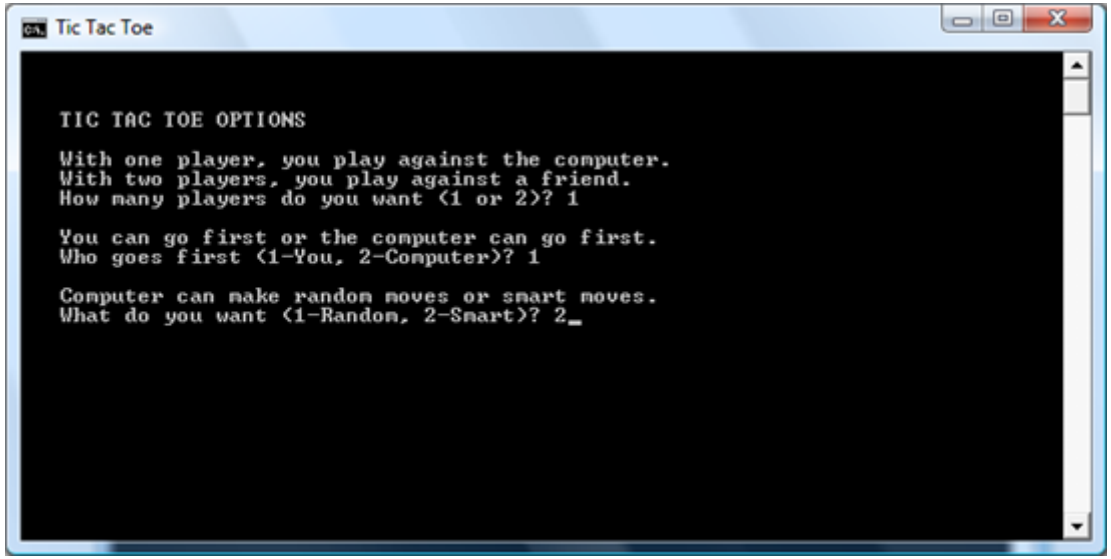




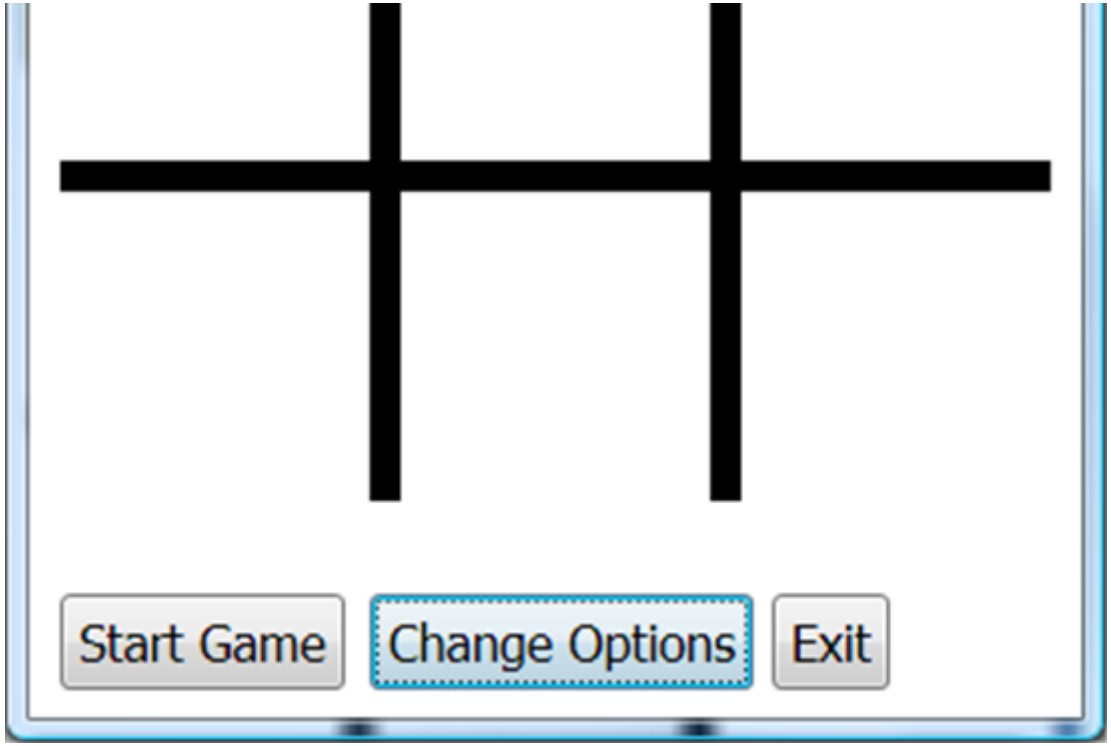
Klik op **Opties wijzigen** om opties te wijzigen. Er verschijnt een tekstvenster:



Je krijgt een paar vragen om spelopties te kiezen (een of twee spelers en, als één speler, wie als eerste gaat en hoe slim je wilt dat de computer is). Maak uw keuzes (druk na elke keer op **Enter**). Voor dit voorbeeld heb ik een spel voor één speler geselecteerd, waarbij ik eerst ga (waardoor ik X's krijg) en een slimme computer:

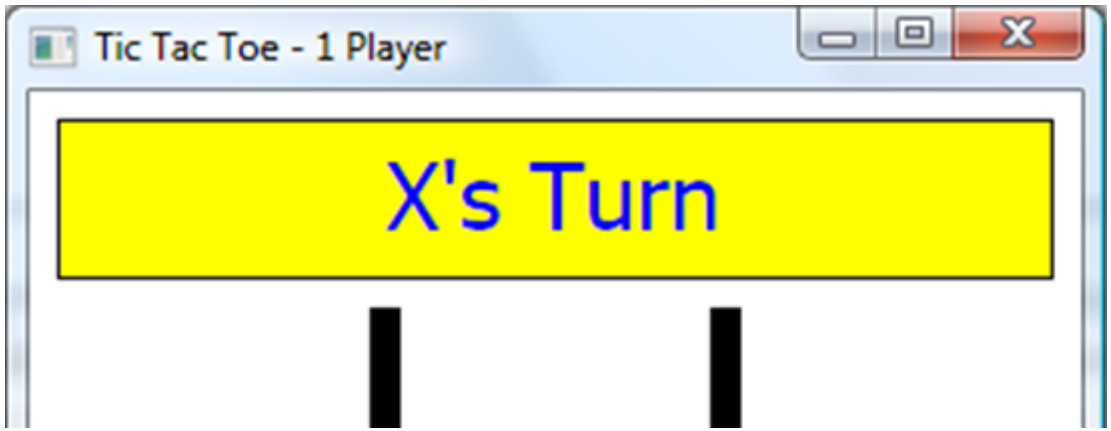


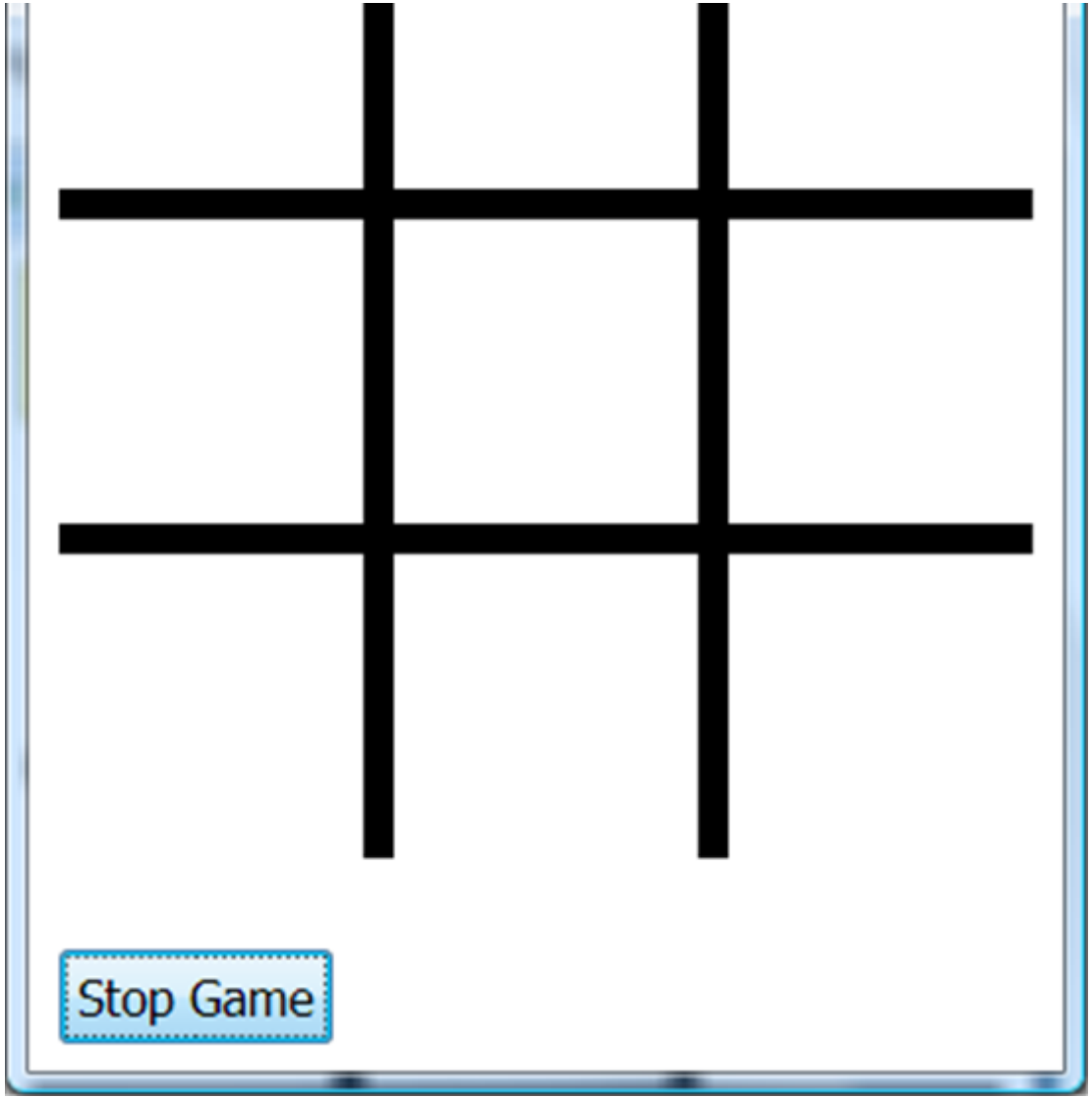
The graphics window reappears with the game still in the ‘stopped’ state.



Notice the title bar area reflects a single player game.

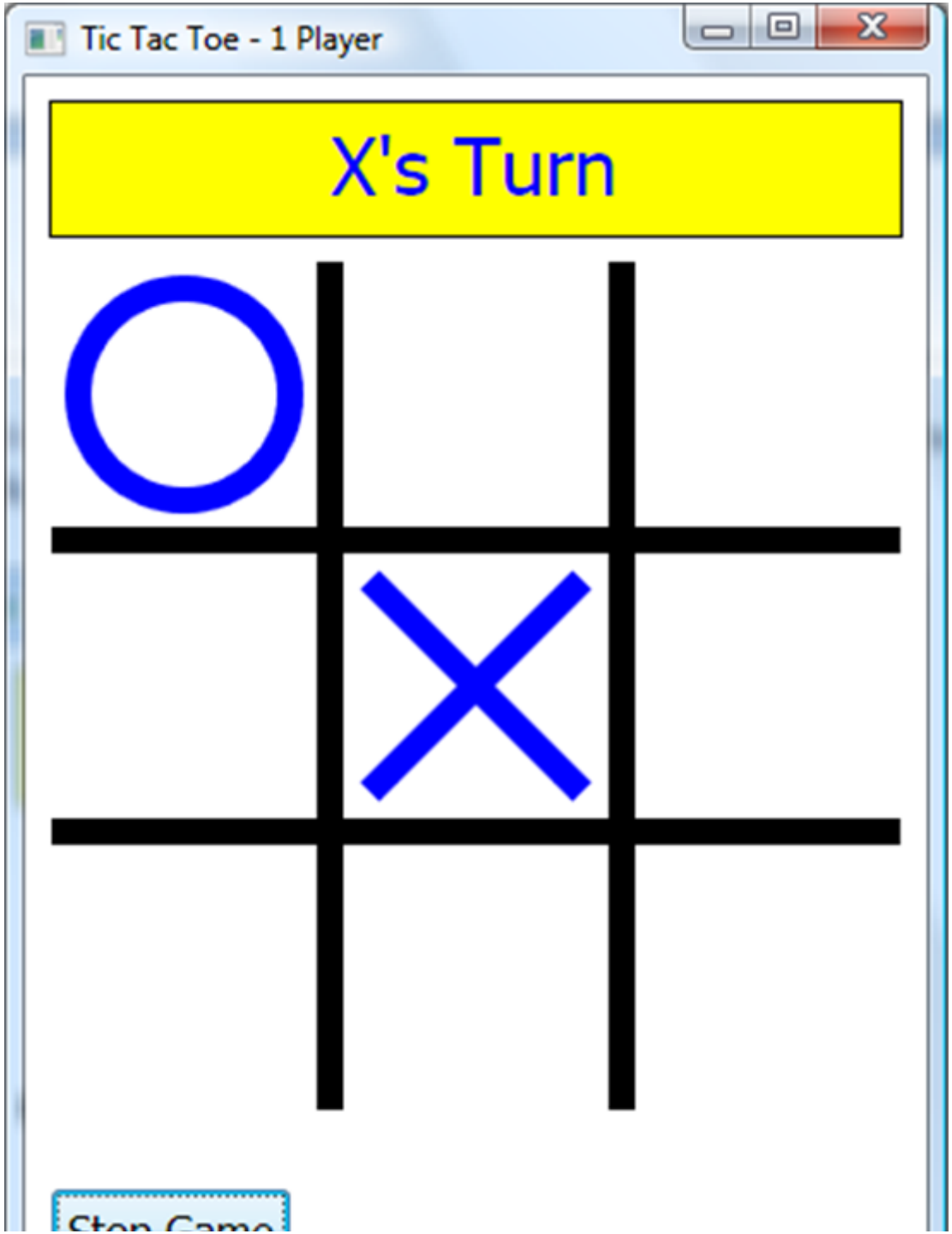
Click the **Start Game** button to start playing. The button choices change. The message at the top of the grid says **X's Turn**. X always goes first in this game (whether it's you, the human player, or the computer). This is the game's 'playing' state:



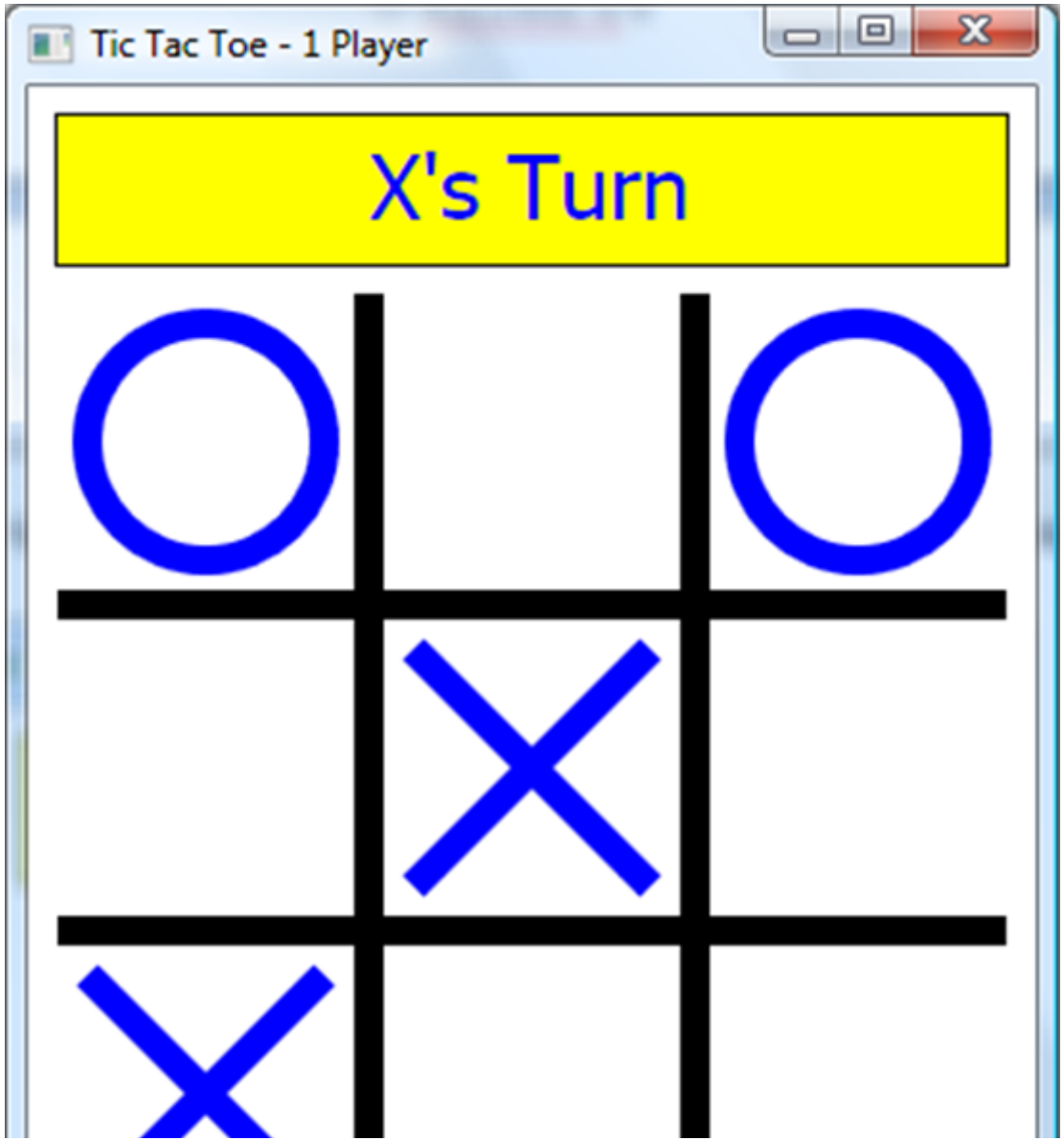


In this state, you make a mark in the grid by clicking on the desired square. The computer will then place its mark, making it your turn again. After each mark, the board is examined for a win or a draw. You keep alternating turns until there is a win, the grid is full or until you click **Stop Game**. The game works the same way for two players, with the two players alternating turns marking the grid.

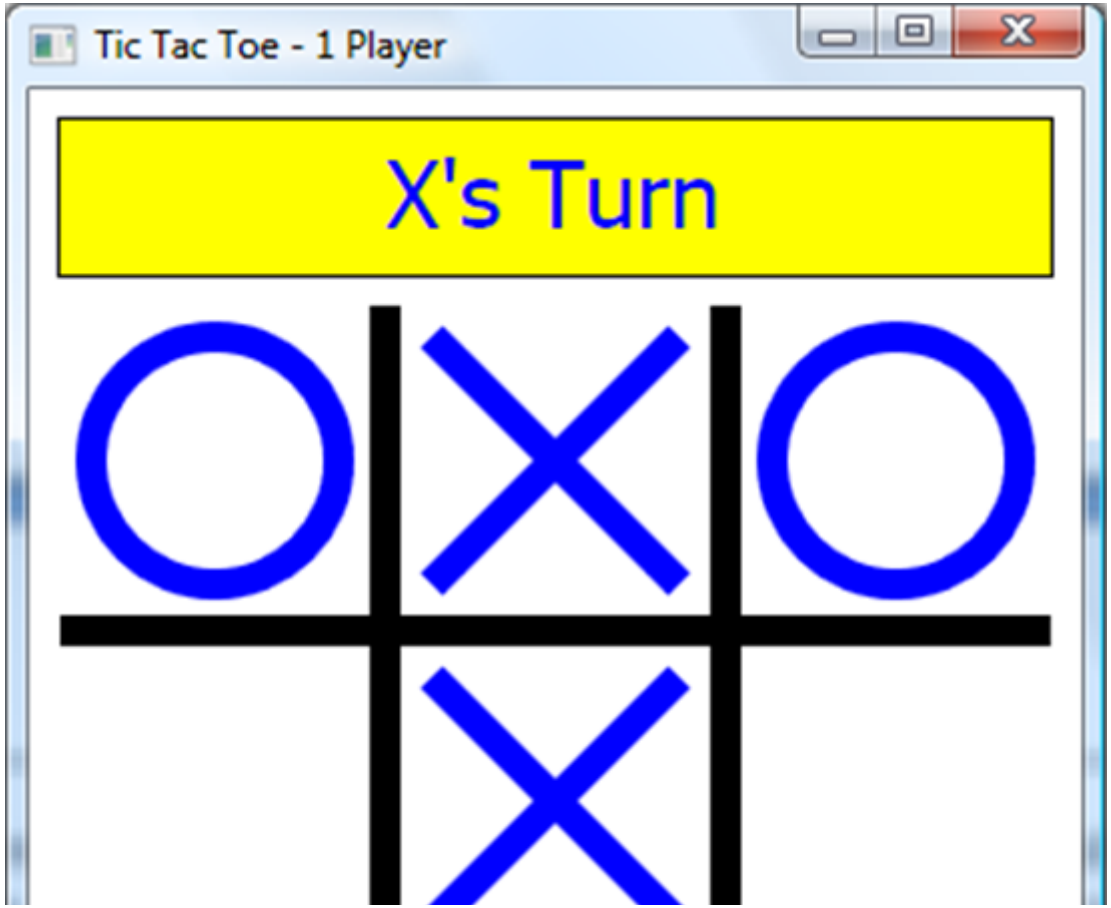
Enter a mark; I chose the center square and the computer immediately placed its mark (an O) in the upper left corner:

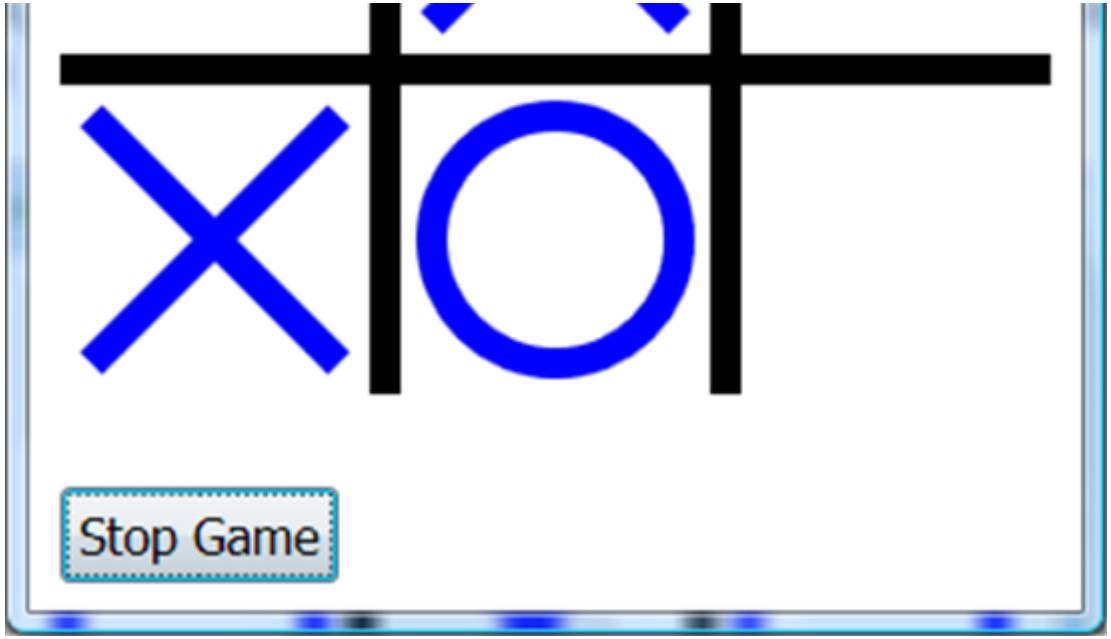

Tic Tac Toe - 1 Player
X's Turn

I try the lower left corner and the computer blocks me from winning:



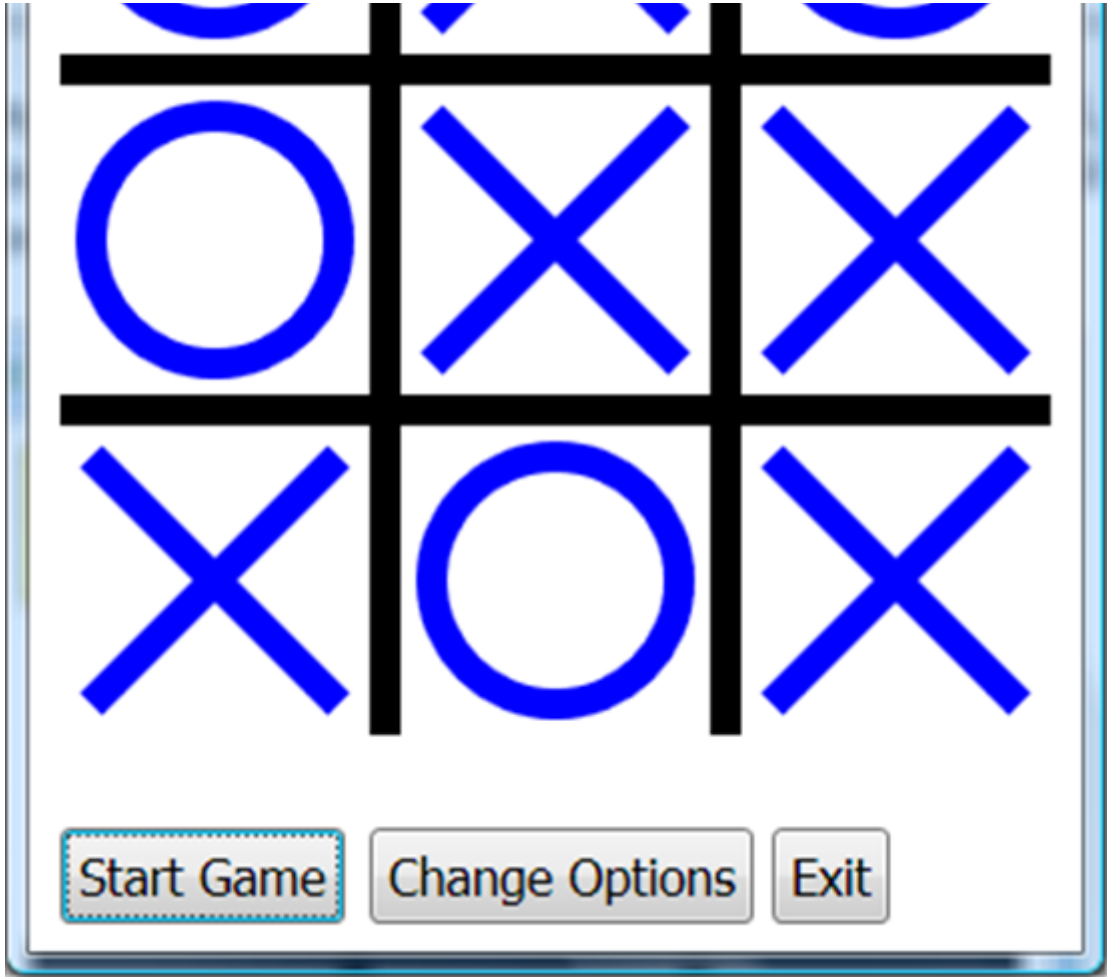
I block the computer's possible win by putting an X in the middle box of the top row. Then, the computer blocks me with a move to the middle of the bottom row:





Looks like no one is going to win here.

I continued playing (moving to the right of the middle row and, following the computer's block, a move to the right of the bottom row) until finally we ended in a draw:



Looks like the computer is pretty smart! We'll see how this intelligence is programmed later. Notice the game returns to its 'stopped' state to allow another game, changing options or to stop the program.

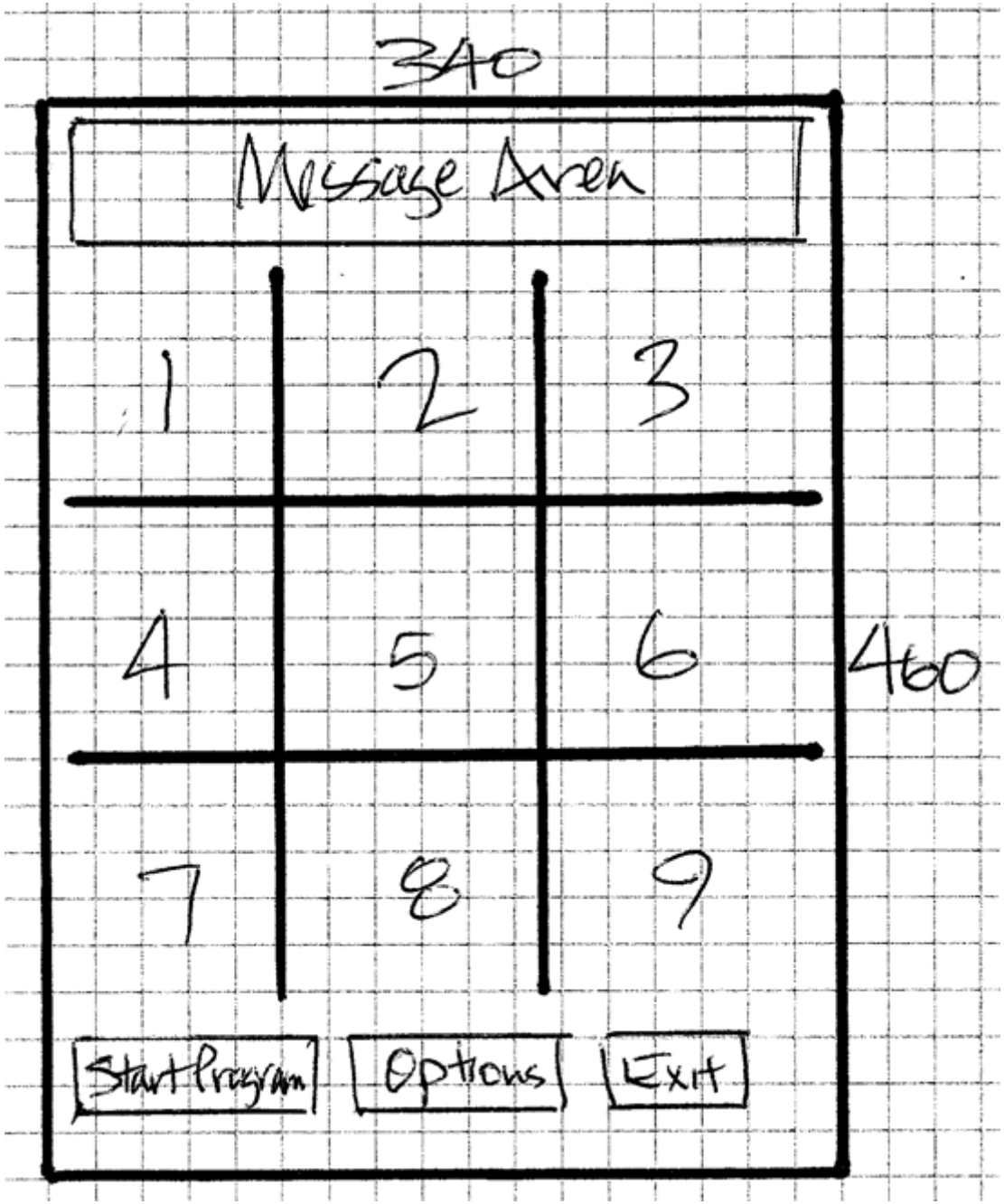
Continue playing the game to understand its operation. Try using the **Random Computer** option (the computer just makes random moves making it easier to beat). Click the **Exit** button when you're done to exit the game. Look over the code in the editor if you like.

You will now build this program in stages. As you build Small Basic programs, we always recommend taking a slow, step-by-step process. It minimizes programming errors and helps build your confidence as things come together in a complete program.

We address **window design**. And, we address **code design**. We discuss how to mark the grid, handle multiple players, check for a win and decide how to make computer-generated moves.

Tic Tac Toe Window Design

Here is a sketch for the window layout of the **Tic Tac Toe** game:



A message area is at the top. The game grid (drawn with skinny rectangles) is below. Identifying numbers are given to the nine boxes in the grid. Three button controls will be used: one to start and stop the game,

one to change options and one to exit the program.

We now begin writing code for the **Tic Tac Toe** game. We will write the code in several steps. As a first step, we will write the code that draws the window, then starts the game and establishes its 'stopped' state. We write code to set game options in a text window. Then, we look at how to go to 'playing' state following clicking of the **Start Game** button. During the code development process, recognize you may modify a particular procedure several times before arriving at the finished product.

Start a new program in Small Basic. Once started, we suggest you immediately save the program with a name you choose. This sets up the folder and file structure needed for your program.

Window Design – Message Area

The first element of the **Tic Tac Toe** window is the bordered message area at the top of the window. It is used to inform the user of whose turn it is and who wins (if there is a win).

Add this code to your blank editor:

```
'Tic Tac Toe

InitializeProgram()

Sub InitializeProgram

  'graphics window

  GraphicsWindow.Width = 340

  GraphicsWindow.Height = 460

  'Draw message area

  GraphicsWindow.BrushColor = "Yellow"

  GraphicsWindow.FillRectangle(10, 10, 320, 50)

  GraphicsWindow.PenColor = "Black"

  GraphicsWindow.PenWidth = 2

  GraphicsWindow.DrawRectangle(10, 10, 320, 50)

  GraphicsWindow.BrushColor = "Blue"

  GraphicsWindow.FontBold = "false"
```

```
  GraphicsWindow.FontSize = 30
  Message = "Game Stopped"
  MessageX = 70
  MessageArea = Shapes.AddText(Message)
  DisplayMessage()
EndSub
```

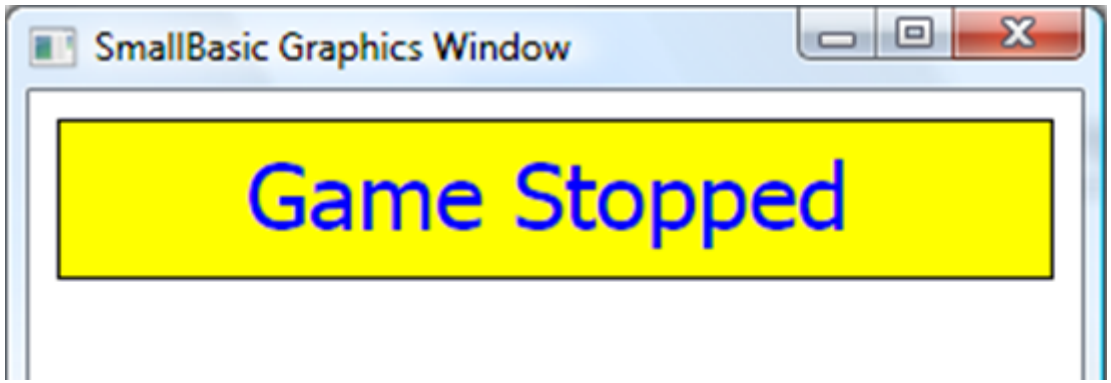
The first line of code calls a subroutine **InitializeProgram** where we will put all code needed to set up the program for use. All remaining code here goes in that subroutine. The code establishes the window size and draws a filled, bordered rectangle to surround the message area. The message to display is **Message** and its horizontal location is **MessageX**. A **Shapes** object (**MessageArea**) is created to display the message.

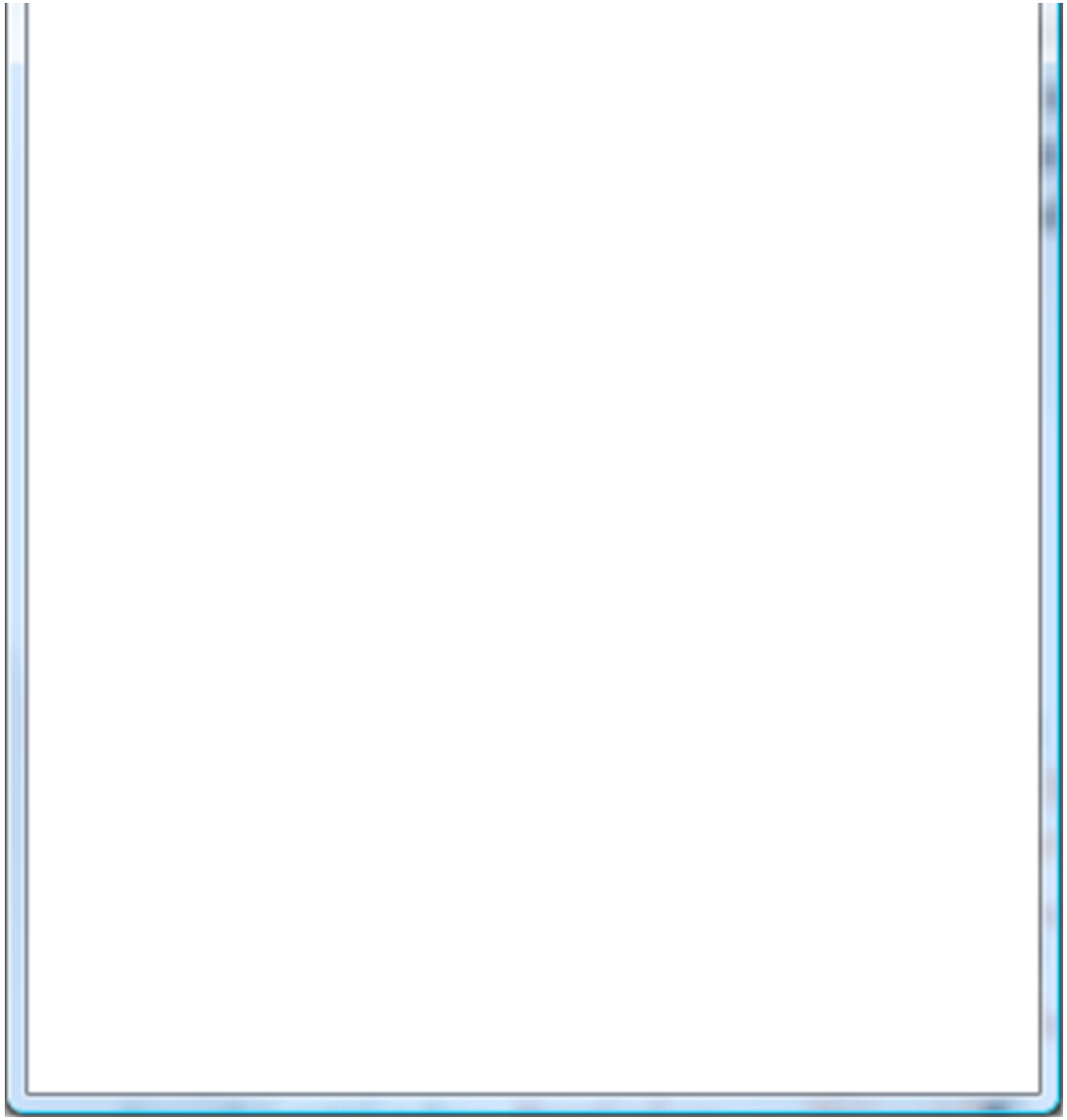
The subroutine **DisplayMessage** writes the message. Add this to your code:

```
Sub DisplayMessage
  Shapes.Move(MessageArea, MessageX, 15)
  Shapes.SetText(MessageArea, Message)
EndSub
```

This is similar to code we will use often in building game programs. To display information that changes often, the **Shapes** object that displays text is invaluable. To update the display, we simply change the text using **SetText** (and in this case **Move** the text to make it appear centered in **MessageArea**). An immediate, flicker-free update is seen.

Save and **Run** the program. You should see the initial '**Game Stopped**' message:



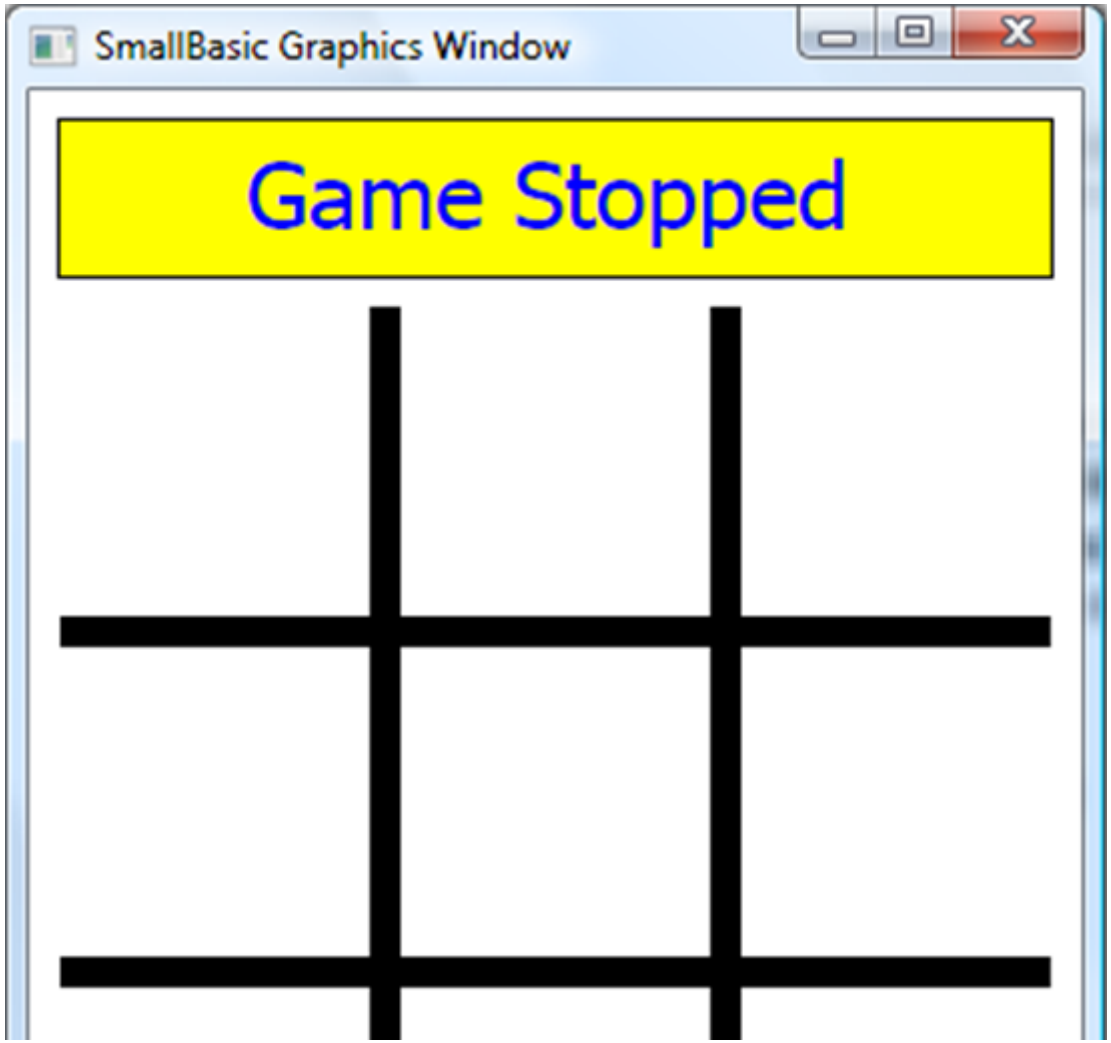


Window Design – Draw Grid

The next element of the game window is the **Tic Tac Toe** grid. It is drawn with ‘skinny’ rectangles. Add these lines of code at the end of the **InitializeProgram** subroutine:

```
'draw grid
GraphicsWindow.BrushColor = "Black"
GraphicsWindow.FillRectangle(10, 170, 320, 10)
GraphicsWindow.FillRectangle(10, 280, 320, 10)
GraphicsWindow.FillRectangle(110, 70, 10, 320)
GraphicsWindow.FillRectangle(220, 70, 10, 320)
```

Save and **Run** the program to see the grid:



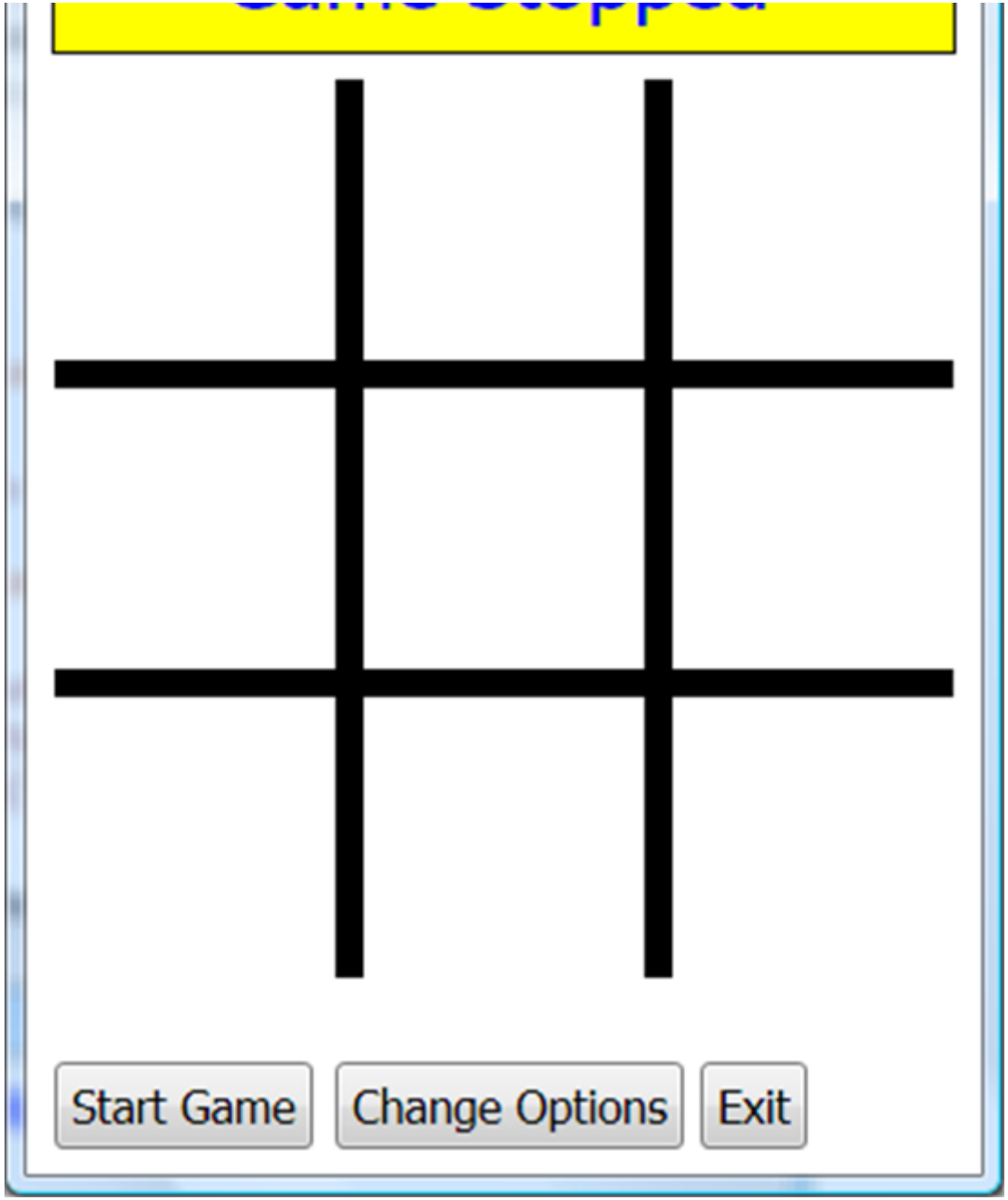
Window Design - Add Buttons

The final item we need are three button controls used to start/stop the game, change options and exit the program.

Add this code at the end of **InitializeProgram**:

```
'define buttons
GraphicsWindow.BrushColor = "Black"
GraphicsWindow.FontSize = 16
StartStopButton = Controls.AddButton("Start Game", 10, 420)
OptionsButton = Controls.AddButton("Change Options", 110, 420)
ExitButton = Controls.AddButton("Exit", 240, 420)
```

Save and **Run** the program. The finished window is displayed:



We establish three buttons (**StartStopButton**, **OptionsButton**, **ExitButton**).

Code Design – Initializing Stopped State

Any time we start a program, there are certain initializations that must take place. Let's look at the initializations needed in the **Tic Tac Toe** game. All initializations are done in the main program.

We want to initialize the number of players in the game (**NumberPlayers**), whether you go first (**YouGoFirst**) and whether the computer is smart (**SmartComputer**). We will use this information to update the graphics window title. We also want to know the location of the nine areas (boxes) within the **Tic Tac Toe** grid. This will help us place marks there later.

Add this code at the end of **InitializeProgram**:

```
'Default Options
NumberPlayers = 2
YouGoFirst = "true"
SmartComputer = "true"
SetTitle()
'initialize box locations and marks
x = 20
y = 80
For I = 1 to 9
  BoxX[I] = x
  BoxY[I] = y
  x = x + 110
  If (x > 240) Then
    x = 20
    y = y + 110
  EndIf
EndFor
```

```
    GameStatus = "Stopped"
```

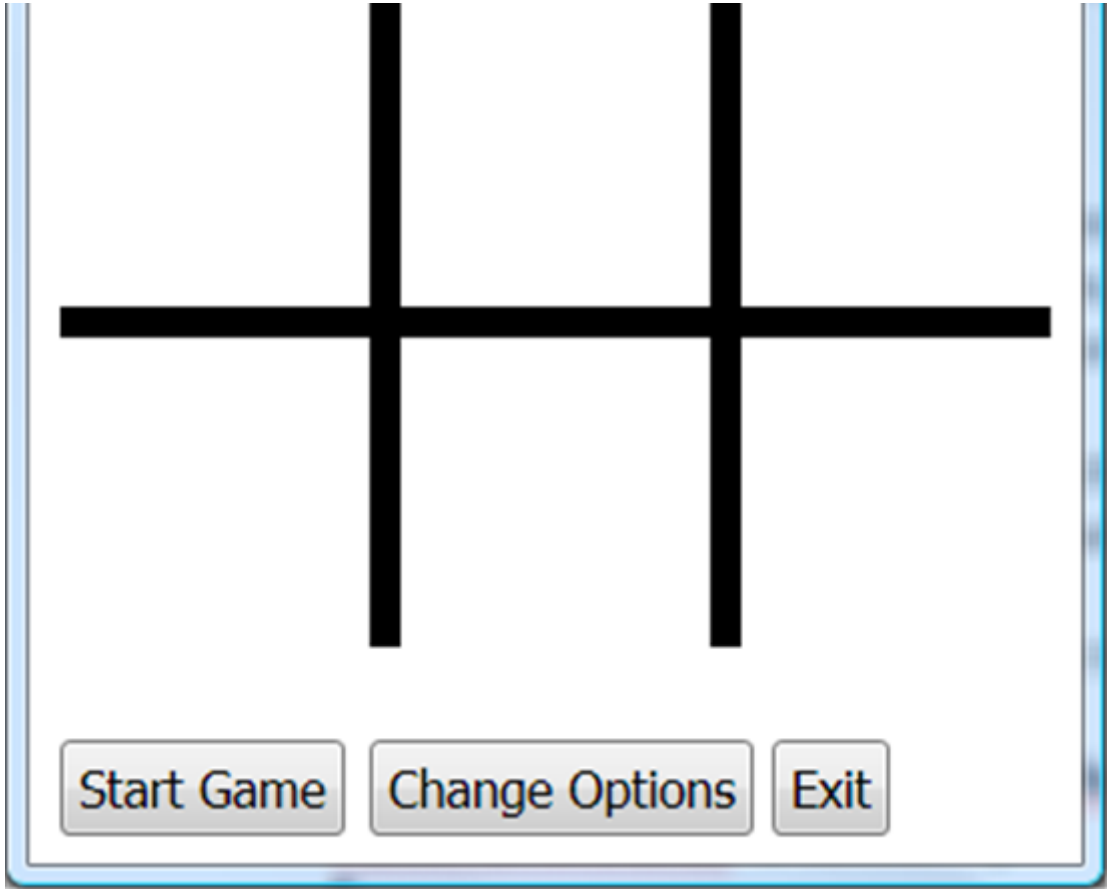
This code initializes the options variables. Next, the (x, y) location of the nine boxes in the grid is established. Lastly, the **GameStatus** is set to **"Stopped"**.

The subroutine **SetTitle** is also needed:

```
  Sub SetTitle
    If (NumberPlayers = 1) Then
      GraphicsWindow.Title = "Tic Tac Toe - 1 Player"
    Else
      GraphicsWindow.Title = "Tic Tac Toe - 2 Players"
    EndIf
  EndSub
```

This sets the window title based on selected options.

Once again, **Save** and **Run**. You'll see the game in the 'stopped' state (using default properties):



As desired, the game initializes in **Two Players** mode.

We have three choices at this point – either click **Start Game** to start, click **Change Options** to change options, or click **Exit** to exit the program. We will write code for each of these options in reverse order.

First, we need to be able to detect button clicks so add this line at the end of **InitializeProgram**:

```
Controls.ButtonClicked = ButtonClickedSub
```

With this, each time a **ButtonClicked** event occurs, the subroutine named **ButtonClickedSub** is called.

The code for exiting is simple. It is placed in the **ButtonClickedSub** subroutine:

```
Sub ButtonClickedSub
```

```
  B = Controls.LastClickedButton
  If (GameStatus = "Stopped") Then
    If (B = ExitButton) Then
      Program.End()
    EndIf
  EndIf
EndSub
```

This simply says whenever **Exit** is clicked, the program ends. **Run** the program. Click **Exit** to make sure the game stops.

If the user clicks **Change Options**, we want to provide the ability to change program options. We will use the Small Basic text window to establish game options. Add the shaded code to the **ButtonClickedSub** subroutine:

```
Sub ButtonClickedSub
  B = Controls.LastClickedButton
  If (GameStatus = "Stopped") Then
    If (B = ExitButton) Then
      Program.End()
    ElseIf (B = OptionsButton) Then
      SetOptions()
    EndIf
  EndIf
EndSub
```

The options are set in the **SetOptions** subroutine – add this routine:

```
Sub SetOptions
  GraphicsWindow.Hide()
```

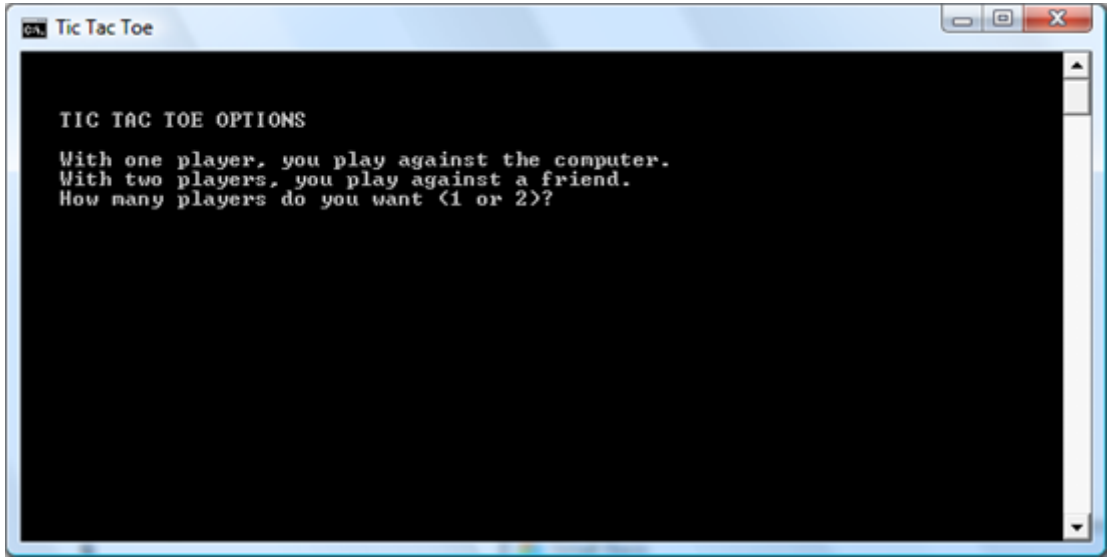
```
TextWindow.Show()
TextWindow.Title = "Tic Tac Toe"
TextWindow.CursorLeft = 3
TextWindow.CursorTop = 3
TextWindow.WriteLine("TIC TAC TOE OPTIONS")
TextWindow.WriteLine("")
GetPlayers:
TextWindow.CursorLeft = 3
TextWindow.WriteLine("With one player, you play against the computer.")
TextWindow.CursorLeft = 3
TextWindow.WriteLine("With two players, you play against a friend.")
TextWindow.CursorLeft = 3
TextWindow.Write("How many players do you want (1 or 2)? ")
NumberPlayers = TextWindow.ReadNumber()
If (NumberPlayers < 1 Or NumberPlayers > 2) Then
  Goto GetPlayers
EndIf
If (NumberPlayers = 1) Then
  GetWhoFirst:
  TextWindow.WriteLine("")
  TextWindow.CursorLeft = 3
  TextWindow.WriteLine("You can go first or the computer can go first.")
  TextWindow.CursorLeft = 3
  TextWindow.Write("Who goes first (1-You, 2-Computer)? ")
  T = TextWindow.ReadNumber()
```

```
  If (T < 1 Or T > 2) Then
    Goto GetWhoFirst
  EndIf
  If (T = 1) Then
    YouGoFirst = "true"
  Else
    YouGoFirst = "false"
  EndIf
  GetSmart:
  TextWindow.WriteLine("")
  TextWindow.CursorLeft = 3
  TextWindow.WriteLine("Computer can make random moves or smart moves.")
  TextWindow.CursorLeft = 3
  TextWindow.Write("What do you want (1-Random, 2-Smart)? ")
  T = TextWindow.ReadNumber()
  If (T < 1 Or T > 2) Then
    Goto GetSmart
  EndIf
  If (T = 2) Then
    SmartComputer = "true"
  Else
    SmartComputer = "false"
  EndIf
EndIf
SetTitle()
```

```
    TextWindow.Hide()
    GraphicsWindow.Show()
EndSub
```

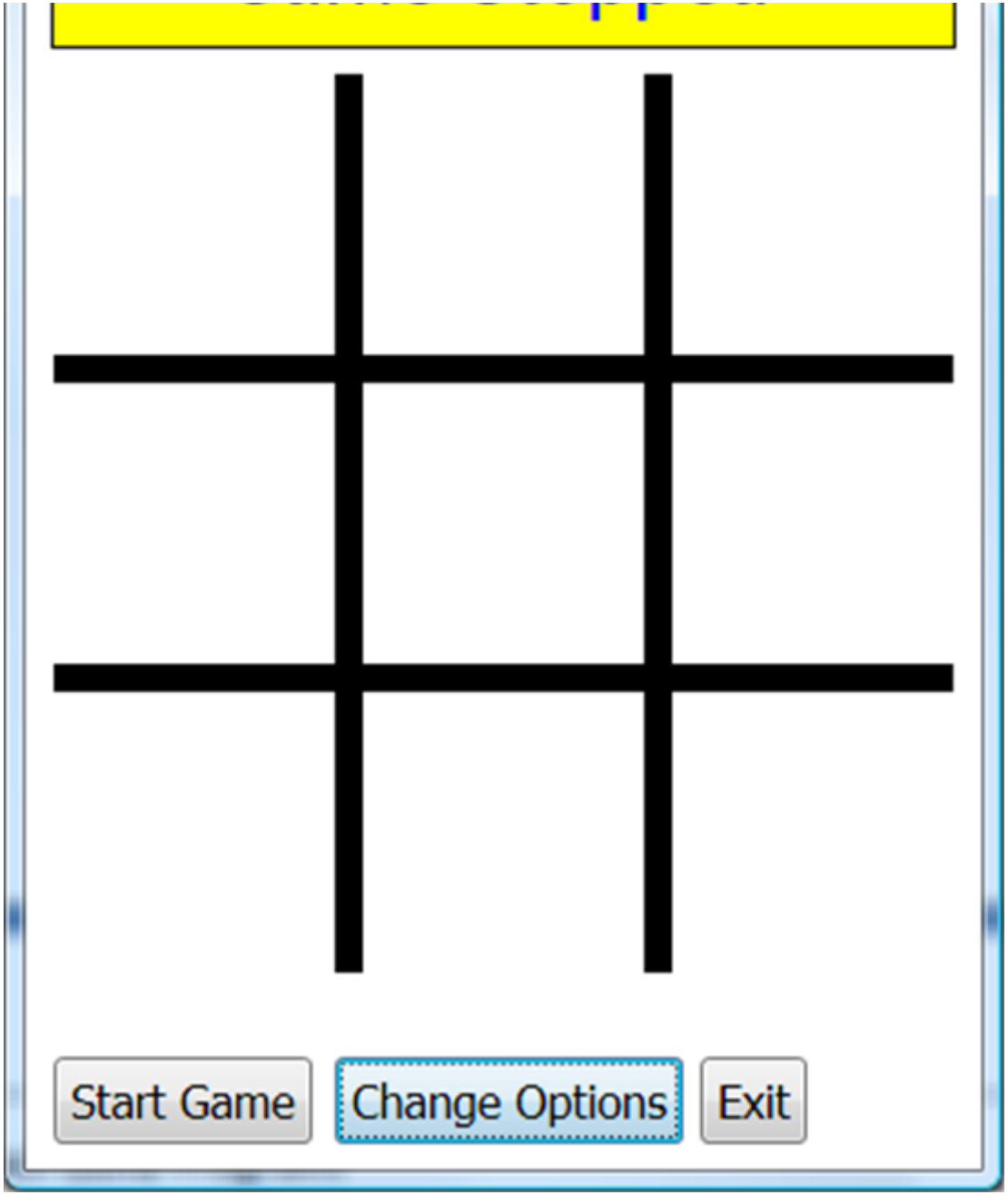
This routine hides the graphics window, displays a text window and asks the user questions necessary to establish the three options (**NumberPlayers**, **YouGoFirst**, **SmartComputer**). Note, if two players are selected, there is no need to ask who goes first or how smart the computer should be. Once the questions are answered, the program returns to the game in 'stopped' state with a new window title.

Save and **Run** the program. Click the **Change Options** button the change options:



Make your choices and press **Enter** after each. The **Tic Tac Toe** game window will again appear.

Here's what I got for a single player (the only difference is in the title bar):



The code for clicking the **Start Game** button (to start the game) is much more complicated. We will build it in several steps. First, we look at switching the game from stopped to playing state.

Code Design -Stopped to Playing State

When the user clicks the **Start Game** button in the 'stopped' state, several things must happen to switch the **Tic Tac Toe** game to 'playing' state:

- Change **GameStatus** to "**Playing**"
- Change the caption of **StartStopButton** to **Stop Game**.
- Hide **OptionsButton** and **ExitButton**.
- Establish this as X's turn (since X always goes first).
- Blank out grid boxes displaying marks.
- Allow player to input a mark on the grid.

We use two variables to help keep track of where we are in the game. If **XTurn** is **"true"**, it is X's turn, otherwise it is O's turn. **NumberClicks** keeps track of how many of the grid boxes have been clicked on (9 maximum).

Add the shaded code to the **ButtonClickSub** subroutine:

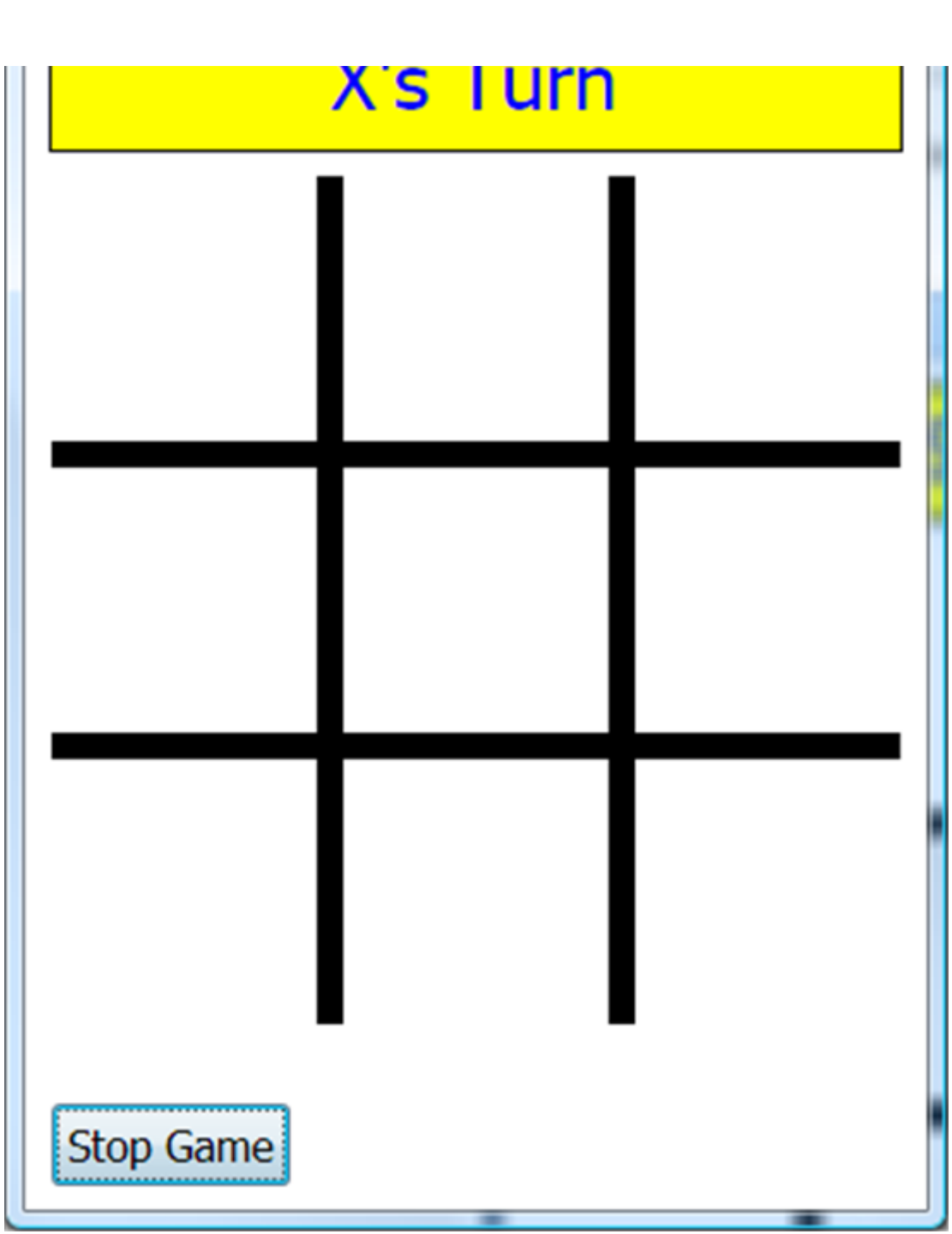
```
Sub ButtonClickedSub

  B = Controls.LastClickedButton

  If (GameStatus = "Stopped") Then

    If (B = ExitButton) Then

      Program.End()

    ElseIf (B = OptionsButton) Then

      SetOptions()

    ElseIf (B = StartStopButton) Then

      StartGame()

    EndIf

  EndIf

EndSub
```

Clicking **Start Game** calls the **StartGame** subroutine:

```
Sub StartGame
  GameStatus = "Playing"
  XTurn = "true"
  Message = "X's Turn"
  MessageX = 115
  DisplayMessage()
  'clear boxes
  GraphicsWindow.BrushColor = GraphicsWindow.BackgroundColor
  For I = 1 To 9
    BoxMark[I] = ""
    GraphicsWindow.FillRectangle(BoxX[I] - 5, BoxY[I] - 5, 90, 90)
  EndFor
  GraphicsWindow.BrushColor = "Black"
  GraphicsWindow.FontSize = 16
  Controls.SetButtonCaption(StartStopButton, "Stop Game")
  Controls.HideControl(OptionsButton)
  Controls.HideControl(ExitButton)
  NumberClicks = 0
EndSub
```

Can you see how the needed steps are implemented in code?

Save and **Run** the program. Click **Start Game** and the game should switch to two player 'playing' state:


X's Turn
Stop Game

The game is waiting for the first player to click one of the grid locations (we'll write code for that soon). Notice clicking **Stop Game** does nothing at the moment. Let's fix that. **Stop** the program.

Code Design –Playing to Stopped State

When the user clicks the **Stop Game** button in the two player 'playing' state, we want the **Tic Tac Toe** game to change to 'stopped' state. The steps for this are:

- Change **GameStatus** to "**Stopped**"
- Change the caption of **StartStopButton** to **Start Game**.
- Show **OptionsButton** and **ExitButton**.

Add the shaded code to the **ButtonClickedSub** subroutine:

```
Sub ButtonClickedSub
  B = Controls.LastClickedButton
  If (GameStatus = "Stopped") Then
    If (B = ExitButton) Then
      Program.End()
    ElseIf (B = OptionsButton) Then
      SetOptions()
    ElseIf (B = StartStopButton) Then
      StartGame()
    EndIf
  ElseIf (GameStatus = "Playing") Then
    If (B = StartStopButton) Then
      'stop program
      Message = "Game Stopped"
      MessageX = 70
      StopGame()
```

```
      EndIf
    EndIf
  EndSub
```

Clicking **Stop Game** establishes a message to display and calls the **StopGame** subroutine.

Add the **StopGame** subroutine:

```
  Sub StopGame
    'restore buttons
    GameStatus = "Stopped"
    GraphicsWindow.BrushColor = "Black"
    GraphicsWindow.FontSize = 16
    Controls.SetButtonCaption(StartStopButton, "Start Game")
    Controls.ShowControl(OptionsButton)
    Controls.ShowControl(ExitButton)
    DisplayMessage()
  EndSub
```

Save and **Run** the program. You should be able to now move from 'stopped' to 'playing' state and back. Try the **Exit** button. Let's write the code for the two player game (the default value for **NumberPlayers**), first looking at how to mark the grid.

Code Design –Marking Grid

In the **Tic Tac Toe** two player game, when a player clicks a box in the grid, we want the proper mark (X or O) to appear. After each mark, we then need to see if anyone won. If there is no win, and there are still empty locations, we switch to the next player. Recall, for reference purposes, the nine areas of the grid are numbered as follows:

1	**2**	**3**

4	5	6
7	8	9

So, when a box in the game grid is clicked, we follow these steps:

- Make sure there is not a mark there already.
- Increment **NumberClicks**.
- Place proper mark in corresponding location (**X** if **XTurn** is **"true"**, otherwise **O**).
- Switch to next player (not needed if there is a win or draw).
- Check for win. If there is a win, declare the winner and stop the game.
- Check if **NumberClicks = 9** (board is full with no win). If so, declare the game a draw and stop.

To detect mouse clicks, we need this line at the end of the **InitializeProgram** subroutine:

```
GraphicsWindow.KeyDown = KeyDownSub
```

With this, each time a **KeyDown** event occurs, the subroutine named **KeyDownSub** is called. The code to perform the steps above will be in this subroutine.

Add this subroutine (**MouseDownSub**) to your program. This implements the needed steps to accept the player's marking of the grid:

```
Sub MouseDownSub
  If (GameStatus = "Playing") Then
    'find which box was clicked
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    ClickedBox = 0
    For I = 1 To 9
      If (x > BoxX[I] And x < BoxX[I] + 80) Then
        If (y > BoxY[I] And y < BoxY[I] + 80) Then
          ClickedBox = I
          Goto GotIt
```

```
        EndIf
      EndIf
    EndFor
    GotIt:
    If (ClickedBox <> 0) Then
      'if already clicked then exit
      If (BoxMark[ClickedBox] <> "") Then
        Goto LeaveSub
      EndIf
      MarkAndCheck()
    EndIf
  EndIf
LeaveSub:
EndSub
```

Let's look at this code. Clicking the graphics window with the mouse will call this subroutine. The first part of the code determines which grid box area (if any) was clicked (the variable **ClickedBox**). If there is already a mark there (not blank), the subroutine is exited. If blank, another subroutine (**MarkAndCheck**) is called to do the actual marking and check for a win. We use a subroutine for this step because later we will want a way for the computer to mark the grid when it is the opponent.

Add the subroutine **MarkAndCheck** to your code (we are not checking for a win yet):

```
Sub MarkAndCheck
  NumberClicks = NumberClicks + 1
  If (XTurn) Then
    BoxMark[ClickedBox] = "X"
    DrawX()
    XTurn = "false"
```

```
      Message = "O's Turn"
      MessageX = 115
      DisplayMessage()
    Else
      BoxMark[ClickedBox] = "O"
      DrawO()
      XTurn = "true"
      Message = "X's Turn"
      MessageX = 115
      DisplayMessage()
    EndIf
    'check for win - will establish a value for WhoWon
    CheckForWin()
    If (WhoWon <> "") Then
      Message = WhoWon + " Wins!"
      MessageX = 115
      StopGame()
    ElseIf (NumberClicks = 9) Then
      'draw
      Message = "It's a Draw!"
      MessageX = 95
      StopGame()
    EndIf
  EndSub
```

Here, the proper mark is placed in **BoxMark[ClickedBox]**. After this, we check for a win in the subroutine **CheckForWin**. If the variable **WhoWon** is not blank (will be established by the check win logic), we declare

the winner. Otherwise, we keep accepting clicks until the grid is full, declaring a draw.

The above subroutine requires these subroutines to draw an X or O. Add them to your program:

```
Sub DrawX
  'draw blue X at ClickedBox
  GraphicsWindow.PenColor = "Blue"
  GraphicsWindow.PenWidth = 10
  GraphicsWindow.DrawLine(BoxX[ClickedBox], BoxY[ClickedBox], BoxX[ClickedBox] + 80,
BoxY[ClickedBox] + 80)
  GraphicsWindow.DrawLine(BoxX[ClickedBox], BoxY[ClickedBox] + 80, BoxX[ClickedBox] +
80, BoxY[ClickedBox])
EndSub
Sub DrawO
  'draw blue O at Clicked Box
  GraphicsWindow.PenColor = "Blue"
  GraphicsWindow.PenWidth = 10
  GraphicsWindow.DrawEllipse(BoxX[ClickedBox], BoxY[ClickedBox], 80, 80)
EndSub
```

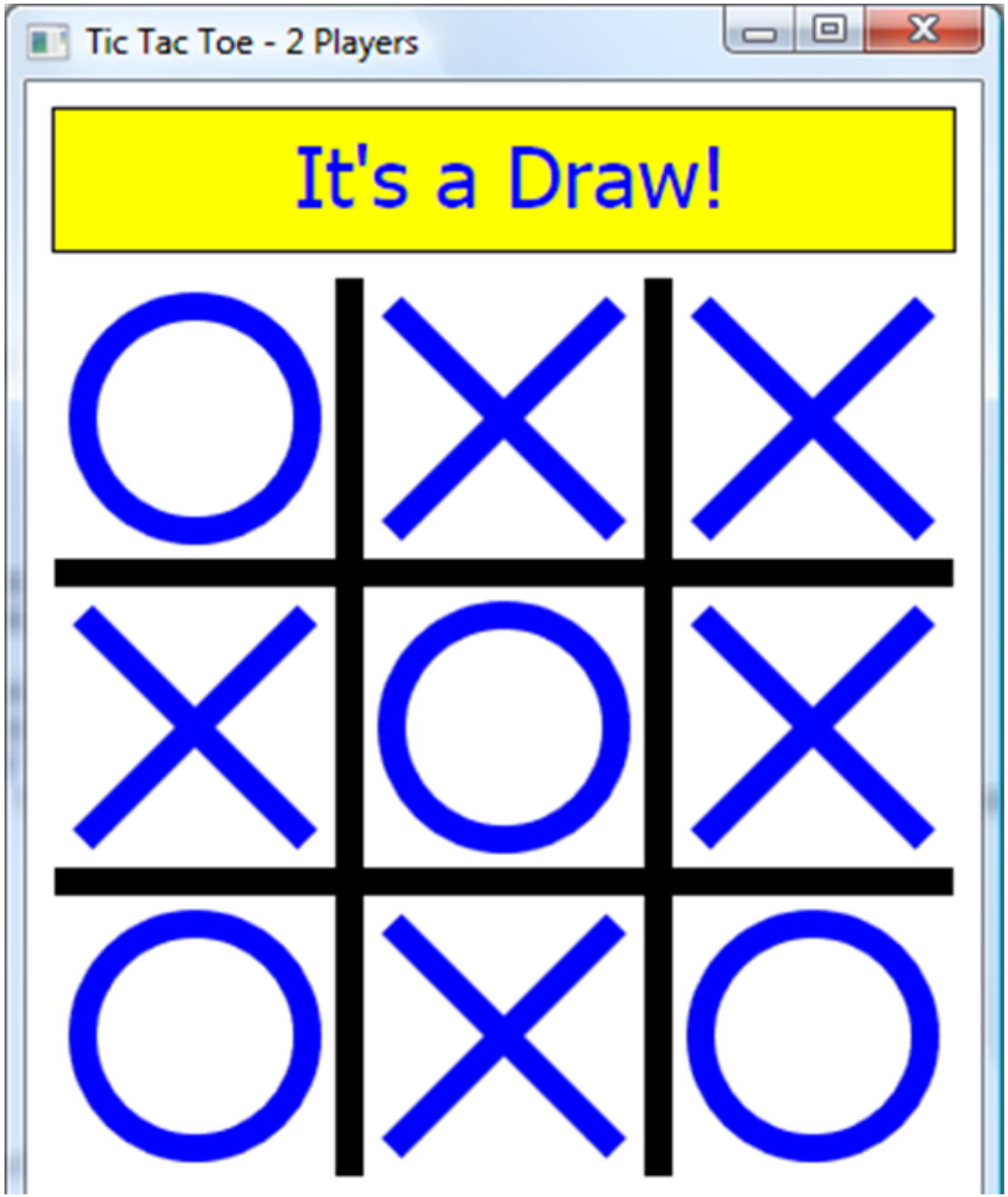
These routines use thick lines for the **X** and a thick ellipse for the **O**.

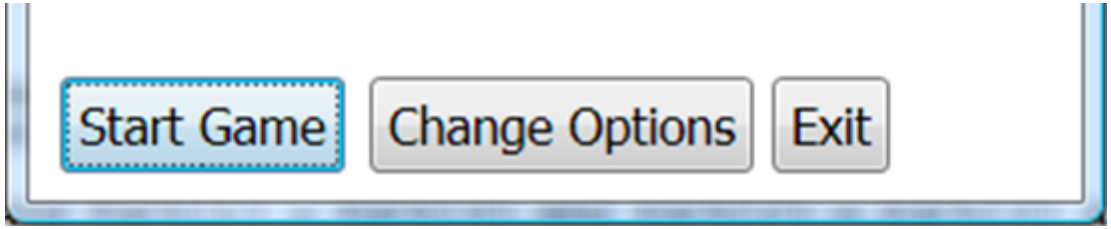
We also need the **CheckForWin** subroutine. For now, just use this which returns a blank for the **WhoWon** variable:

```
Sub CheckForWin
  WhoWon = ""
EndSub
```

We will write code for this subroutine next.

Save and **Run** the program. You should be able to click on each of the grid locations, placing X's and O's there in alternate turns. Once you have filled all the grid locations, the game will be returned to stopped state. Wins will not be recognized. Here's what I got when I tried it:





As expected, the computer completely missed the fact that O (Player 2) got a **Tic Tac Toe**!! Let's fix that. We will modify the **CheckForWin** subroutine to determine a value for the **WhoWon** variable used in the code.

Code Design – Checking For Win

The **CheckForWin** subroutine examines the playing grid and determines if there is a winner. If so, the win will be identified and the subroutine will establish a value for **WhoWon** - it will hold the marker (**X** or **O**) for the winner or a blank if there is no winner. Let's establish a strategy for doing this.

There are eight possible ways to win (3 horizontal, 3 vertical, 2 diagonal). Notice the indices for the **BoxMark** array used in the playing grid are laid out in this manner:

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

If we have a string array named **PossibleWins**, its 8 elements would be:

```
'possible wins
PossibleWins[1] = "123"
PossibleWins[2] = "456"
PossibleWins[3] = "789"
PossibleWins[4] = "147"
PossibleWins[5] = "258"
```

```
PossibleWins[6] = "369"

PossibleWins[7] = "159"

PossibleWins[8] = "357"
```

Add the above lines to the **InitializeProgram** subroutine to establish values for the **PossibleWins** array.

So our win logic would be to go through each possible win and see if the corresponding elements of **BoxMark** all contain the same mark (**X** or **O**, but not blank). If so, a winner is declared.

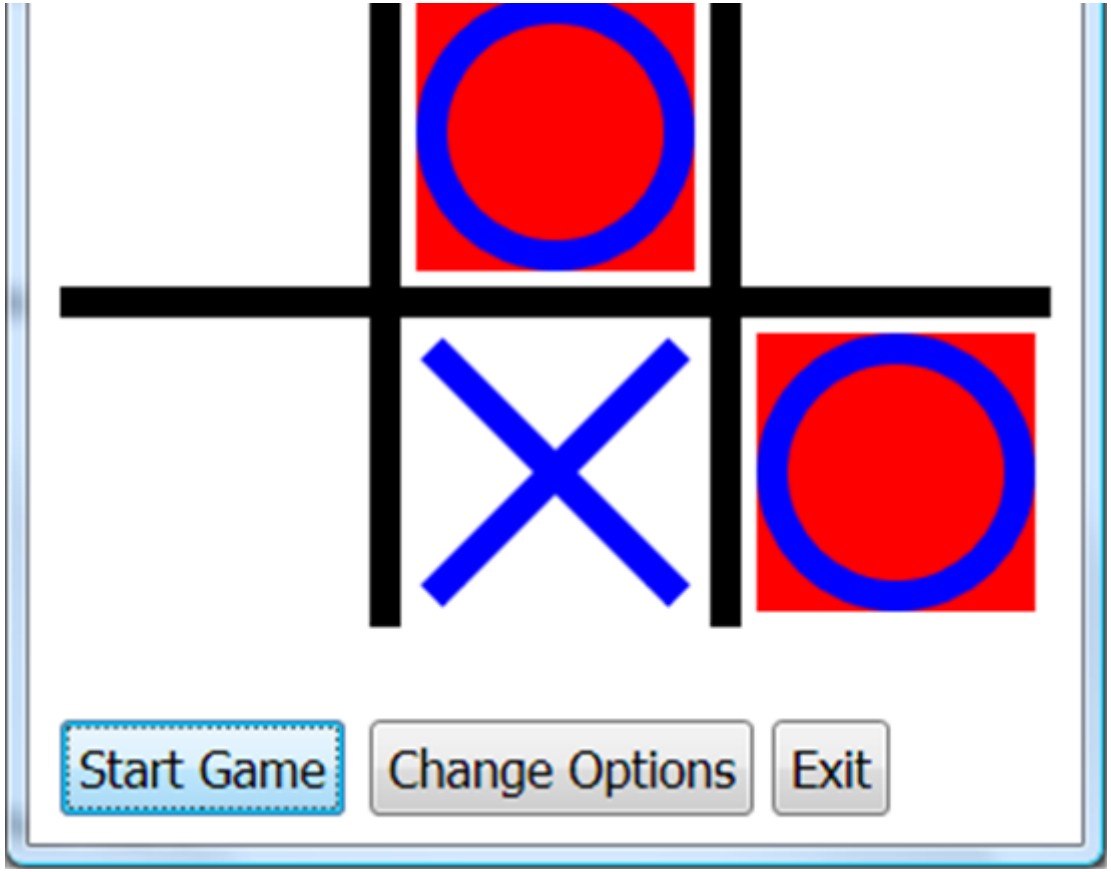
Put the modified **CheckForWin** subroutine in your program (new lines are shaded):

```
Sub CheckForWin

  WhoWon = ""

  'check all possible for win

  For I = 1 To 8

    For J = 1 To 3

      BoxNumber[J] = Text.GetSubText(PossibleWins[I], J, 1)

      Mark[J] = BoxMark[BoxNumber[J]]

    EndFor

    If (Mark[1] = Mark[2] And Mark[1] = Mark[3] And Mark[2] = Mark[3] And Mark[1] <>
"") Then

      'we have a winner

      WhoWon = Mark[1]

      For J = 1 To 3

        GraphicsWindow.BrushColor = "Red"

        GraphicsWindow.FillRectangle(BoxX[BoxNumber[J]] - 5, BoxY[BoxNumber[J]] - 5,
90, 90)

        ClickedBox = BoxNumber[J]

        If (WhoWon = "X") Then
```

```
            DrawX()
          Else
            DrawO()
          EndIf
        EndFor
      EndIf
    EndFor
  EndSub
```

This code goes through all the possible wins. If all the marks in a particular horizontal, vertical or diagonal line match, the marks in that line are redrawn in a red rectangle and the corresponding winner returned. Study the code to see how it works. The **BoxNumber** array holds the indices of the **BoxMark** array for each possible win.

Save and **Run** the program. You and a friend should be able to compete with wins and draws properly determined. Here is a replay of the game I tried before (notice the win by O is now declared):



The two player game is now complete. Let's start looking at how to implement a one player game versus a computer opponent. We'll start easy, just having the computer make random moves (no brains!). This will help us establish the logic of switching players when the computer is one of them.

Code Design – Random Computer Moves

A big part of allowing the computer to play against a human (you, the player) is to decide what "thought processes" to give the computer. We're allowing two choices: a **random computer** and a **smart computer**. We'll start by coding up the random computer. This will get all the logic of implementing computer moves along with human moves working properly. Then, we'll move on to a smart computer, developing a formidable opponent.

All of the logic behind a computer move will be implemented in a subroutine named **ComputerTurn**. For random moves, it will simply choose (at random) one of the empty boxes in the grid and place a marker in that box (**X** if computer goes first, **O** if human goes first.

Place this subroutine **ComputerTurn** in your program:

```
Sub ComputerTurn
  If (SmartComputer <> "true") Then
    'random logic
    'put mark in Nth available square
    N = Math.GetRandomNumber(9 - NumberClicks)
    I = 0
    For ClickedBox = 1 To 9
      If (BoxMark[ClickedBox]  = "") Then
        I = I + 1
        If I = N Then
          Goto GotMark
        EndIf
      EndIf
    EndFor
    GotMark:
    'put mark in ClickedBox
    MarkAndCheck()
  Else
    'smart computer
  EndIf
EndSub
```

The logic behind the code in **ComputerTurn** is straightforward. At any time, we have **9 - NumberClicks** empty boxes in the grid. The code selects a random number from **1** to **9 - NumberClicks** and counts ahead that number of empty boxes to identify the box to mark. To mark the identified box, **BoxMark[ClickedBox]**,

we call the subroutine **MarkAndCheck**. The program will know whether to place an X or O in the box based on previously implemented logic.

Now, let's implement the **ComputerTurn** subroutine to allow the computer to play. We need to modify two subroutines. First, when we start a game, if the computer moves first, we need to invoke **ComputerTurn**. Make the shaded changes to the **StartGame** subroutine:

```
Sub StartGame
  GameStatus = "Playing"
  XTurn = "true"
  Message = "X's Turn"
  MessageX = 115
  DisplayMessage()
  'clear boxes
  GraphicsWindow.BrushColor = GraphicsWindow.BackgroundColor
  For I = 1 To 9
    BoxMark[I] = ""
    GraphicsWindow.FillRectangle(BoxX[I] - 5, BoxY[I] - 5, 90, 90)
  EndFor
  GraphicsWindow.BrushColor = "Black"
  GraphicsWindow.FontSize = 16
  Controls.SetButtonCaption(StartStopButton, "Stop Game")
  Controls.HideControl(OptionsButton)
  Controls.HideControl(ExitButton)
  NumberClicks = 0
  If (NumberPlayers = 1 And YouGoFirst = "false") Then
    ComputerTurn()
  EndIf
```

```
EndSub
```

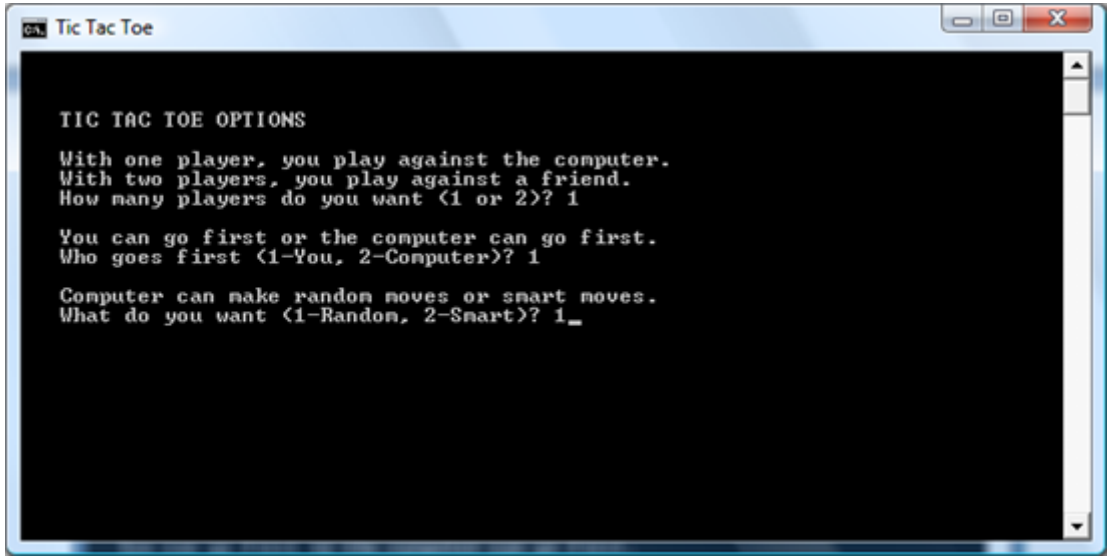
And, after a mark is placed by the human player, the computer needs to take a turn. This logic is in the **MarkAndCheck** subroutine. The needed changes are shaded:

```
Sub MarkAndCheck
  NumberClicks = NumberClicks + 1
  If (XTurn) Then
    BoxMark[ClickedBox] = "X"
    DrawX()
    XTurn = "false"
    Message = "O's Turn"
    MessageX = 115
    DisplayMessage()
  Else
    BoxMark[ClickedBox] = "O"
    DrawO()
    XTurn = "true"
    Message = "X's Turn"
    MessageX = 115
    DisplayMessage()
  EndIf
  'check for win - will establish a value for WhoWon
  CheckForWin()
  If (WhoWon <> "") Then
    Message = WhoWon + " Wins!"
    MessageX = 115
```

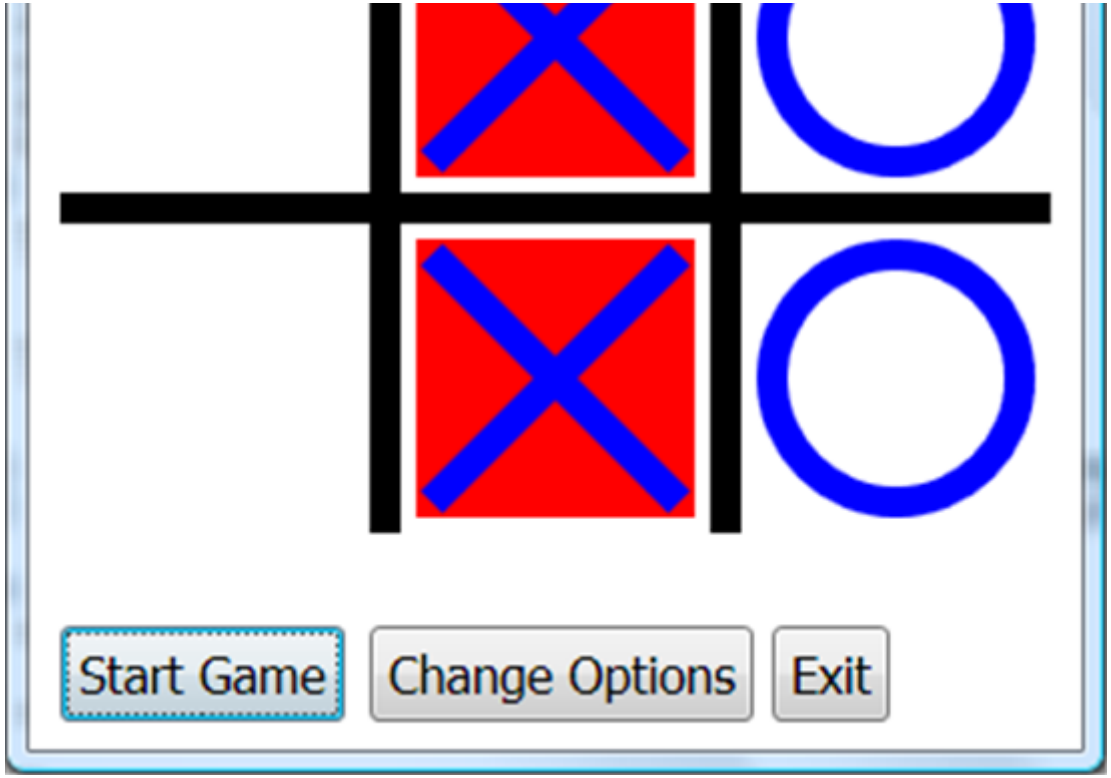
```
      StopGame()
    ElseIf (NumberClicks = 9) Then
      'draw
      Message = "It's a Draw!"
      MessageX = 95
      StopGame()
    EndIf
    If (NumberPlayers = 1 And WhoWon = "") Then
      If (XTurn = "true" And YouGoFirst = "false") Or (XTurn = "false" And YouGoFirst =
"true") Then
        ComputerTurn()
      EndIf
    EndIf
  EndSub
```

With the added code, if there is no win, the computer takes a turn when it goes first and it's X's turn or takes a turn when the human goes first and it's O's turn.

Save and **Run** the game. Click **Change Options** to change options. Choose **1 Player**, **You Go First** and **Random Computer**. Each option requires entering a 1:



Make sure things work properly, you should see the computer is pretty easy to beat! Here's a game I won when I went first:



Try playing a few games (you go first, computer going first).

Code Design – Smart Computer Moves

We've come to one of the more fun and more challenging parts of this game program. How can we make our computer a smarter opponent? In any game where a computer is playing against a human, we must be able to write down some rules that give the computer the appearance of intelligence. Many computer opponents are unbeatable. For example, it is very hard for a human to beat a computer at the game of chess.

So how do we make our computer a better **Tic Tac Toe** player? We try to imbed choices we would make if we were playing the game. So for the computer to be a smart player, the programmer needs to be a smart player. This usually takes practice and study. For the game of **Tic Tac Toe**, we can develop a fairly simple, yet very intelligent strategy. Let's do it.

When it is the computer's turn, what should its move be? The rules we will use are (in order of choice):

1. If the computer can win with a move, make that move and the game is over. So, if there's a line with two of the computer's markers and an empty space, the empty space is the place to mark!
2. If the computer can block with a move, make that move and the opponent can't win on the next move. So, if there's a line with two of the human player's markers and an empty space, the empty space is the place to mark!
3. If there is no possible win or possible block, make a move in this order: center square, one of the four corner squares, or one of the four side squares.

I think you see this logic makes sense. You may wonder about Step 3 - why we choose that particular order. Recall there are 8 possible ways to win in **Tic Tac Toe** (3 horizontal, 3 vertical, 2 diagonal). The center square is needed in 4 of these, any one corner is involved in 3 possibilities, while any side is involved in just 2 wins. Hence, the order of choices is made on basic probabilities. Let's implement this logic in code.

Here is the modified **ComputerTurn** subroutine that implements this 'smart' logic. The changes are shaded:

```
Sub ComputerTurn

  If (SmartComputer <> "true") Then

    'random logic

    'put mark in Nth available square

    N = Math.GetRandomNumber(9 - NumberClicks)

    I = 0

    For ClickedBox = 1 To 9

      If (BoxMark[ClickedBox]  = "") Then

        I = I + 1

        If I = N Then

          Goto GotMark

        EndIf

      EndIf

    EndFor

    GotMark:

    'put mark in ClickedBox
```

```
    MarkAndCheck()
  Else
    'smart computer
    BestMoves[1] = 5
    BestMoves[2] = 1
    BestMoves[3] = 3
    BestMoves[4] = 7
    BestMoves[5] = 9
    BestMoves[6] = 2
    BestMoves[7] = 4
    BestMoves[8] = 6
    BestMoves[9] = 8
    'determine who has what mark
    If (YouGoFirst) Then
      ComputerMark = "O"
      PlayerMark = "X"
    Else
      ComputerMark = "X"
      PlayerMark = "O"
    EndIf
    'Step 1 (K = 1) - check for win - see if two boxes hold computer mark and one is
empty
    'Step 2 (K = 2) - check for block - see if two boxes hold player mark and one is
empty
    For K = 1 To 2
      If K = 1 Then
```

```
      MarkToFind = ComputerMark
    Else
      MarkToFind = PlayerMark
    EndIf
    For I = 1 To 8
      N = 0
      EmptyBox = 0
      For J = 1 To 3
        BoxNumber[J] = Text.GetSubText(PossibleWins[I], J, 1)
        Mark[J] = BoxMark[BoxNumber[J]]
        If (Mark[J] = MarkToFind) Then
          N = N + 1
        ElseIf (Mark[J] = "") Then
          EmptyBox = BoxNumber[J]
        EndIf
      EndFor
      If (N = 2 And EmptyBox <> 0) Then
        'mark empty box to win (K = 1) or block (K = 2)
        ClickedBox = EmptyBox
        MarkAndCheck()
        Goto LeaveComputerMove
      EndIf
    EndFor
  EndFor
  'Step 3 - find next best move
```

```
      For I = 1 To 9
        If (BoxMark[BestMoves[I]] = "") Then
          ClickedBox = BestMoves[I]
          MarkAndCheck()
          Goto LeaveComputerMove
        EndIf
      EndFor
    EndIf
  LeaveComputerMove:
  EndSub
```

In the 'smart' logic, we first find out whether the computer has X or O. Steps 1 and 2 of the computer logic are done in a For loop with K as index. In that loop, we go through all the possible wins looking for a line with 2 identical marks and an empty box. For K=1, we look for the computer's mark and an empty box – giving the computer a win on the next move. For K=2, we look for the human's mark and an empty box – giving the computer a block on the next move. If neither Step 1 or Step 2 is successful, we move to Step 3. The next best moves are listed in desired order in the array **BestMoves**. In Step 3, we go through this array, finding the first empty box available and move there.

Save and **Run** the program. The game is now fully functional. Try playing it against the computer. You should find it can't be beat. The best you can do against the computer is a draw. Before leaving, let's add a couple of sounds.

Code Design – Adding Sounds

We know sounds make games a bit more fun. Let's add a couple to the Tick Tac Toe game. In the **KidGamesSB\KidGamesSB Programs\TicTacToe** folder are two wav files that can be used for sound. The file **beep.wav** is a sound we'll use for games that end a draw. The **tada.wav** file is a celebratory sound we'll use for wins by either player. These files will be loaded when the program starts. Copy the two sound files to your program's folder.

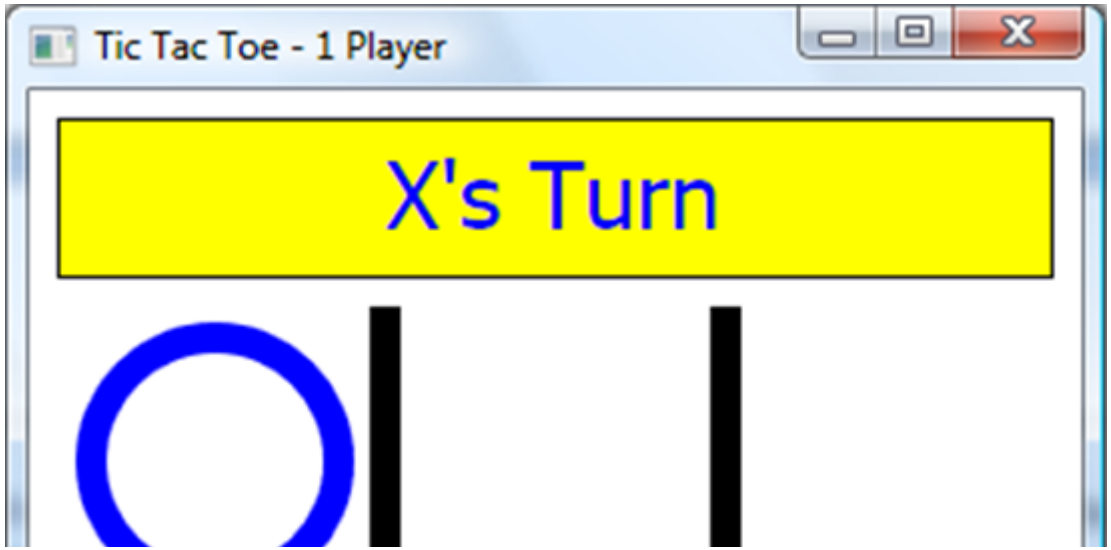
Play the two sounds at the appropriate (shaded) locations in the **MarkAndCheck** subroutine:

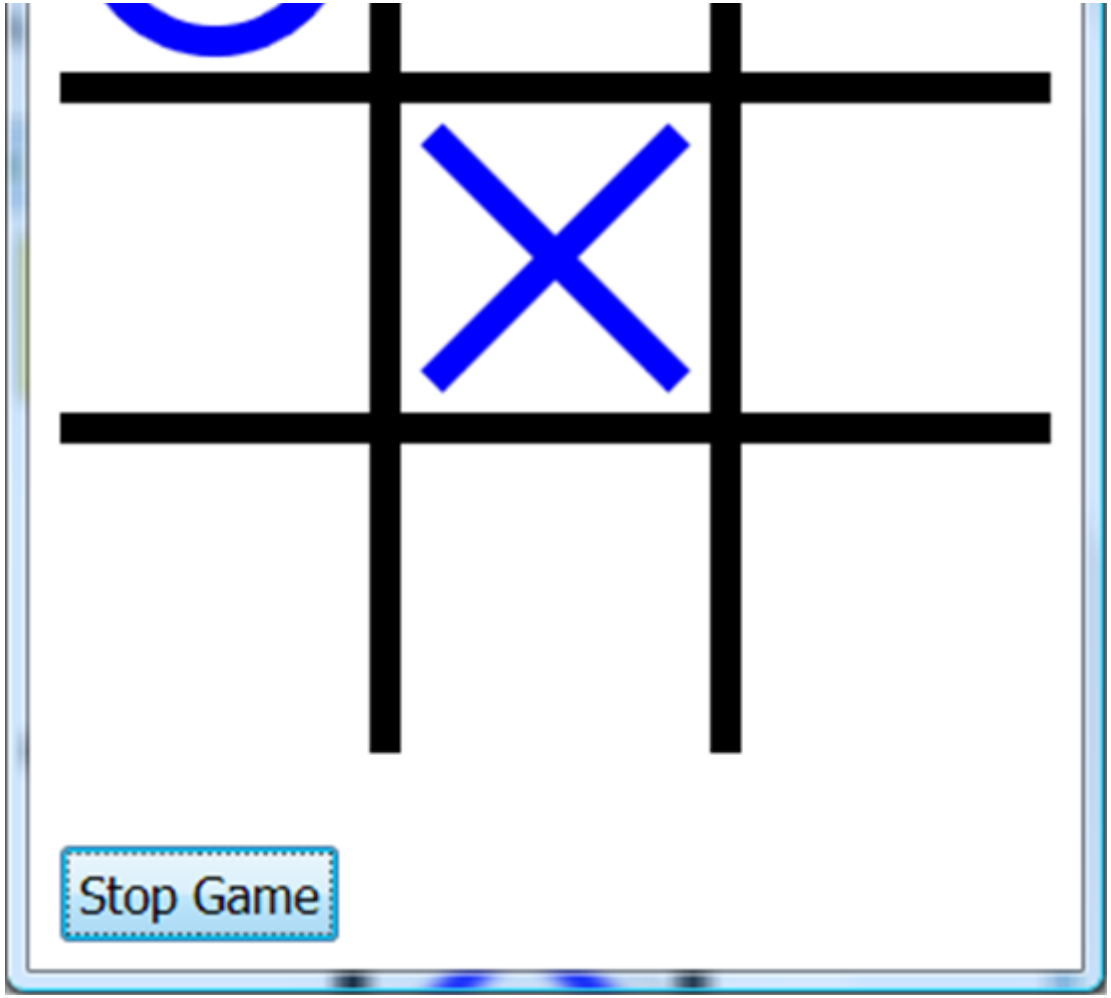
```
  Sub MarkAndCheck
```

```
NumberClicks = NumberClicks + 1
If (XTurn) Then
  BoxMark[ClickedBox] = "X"
  DrawX()
  XTurn = "false"
  Message = "O's Turn"
  MessageX = 115
  DisplayMessage()
Else
  BoxMark[ClickedBox] = "O"
  DrawO()
  XTurn = "true"
  Message = "X's Turn"
  MessageX = 115
  DisplayMessage()
EndIf
'check for win - will establish a value for WhoWon
CheckForWin()
If (WhoWon <> "") Then
  Sound.Stop(Program.Directory + "\tada.wav")
  Sound.Play(Program.Directory + "\tada.wav")
  Message = WhoWon + " Wins!"
  MessageX = 115
  StopGame()
ElseIf (NumberClicks = 9) Then
```

```
      'draw
      Sound.Stop(Program.Directory + "\beep.wav")
      Sound.Play(Program.Directory + "\beep.wav")
      Message = "It's a Draw!"
      MessageX = 95
      StopGame()
    EndIf
    If (NumberPlayers = 1 And WhoWon = "") Then
      If (XTurn = "true" And YouGoFirst = "false") Or (XTurn = "false" And YouGoFirst =
"true") Then
        ComputerTurn()
      EndIf
    EndIf
  EndSub
```

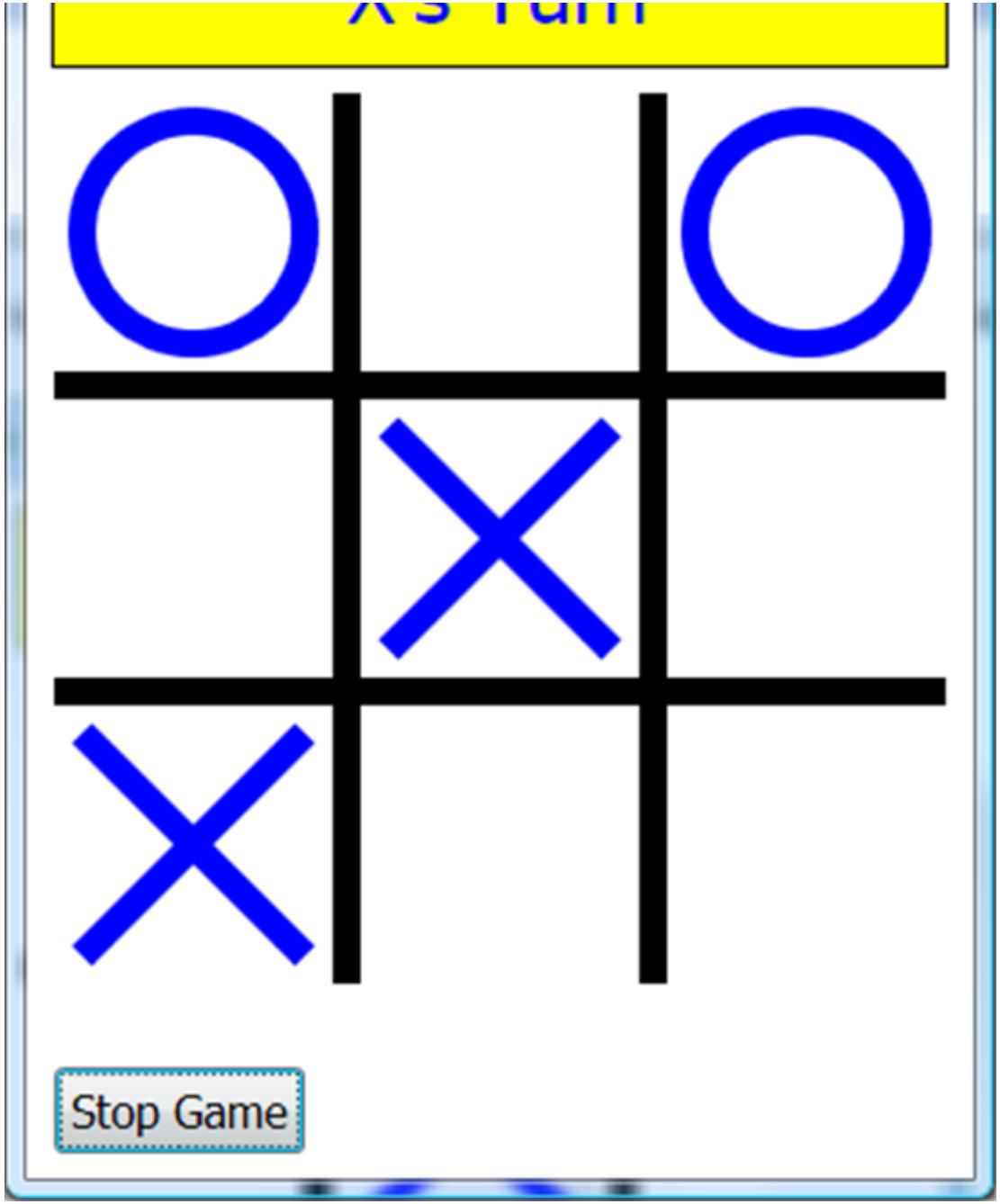
Save and **Run** the program. You should now have a complete, running version of the **Tic Tac Toe** game. Have fun playing it! Again, you will see the computer can't be beat. Here's a game I played against a 'smart' computer where I went first, taking the middle square:





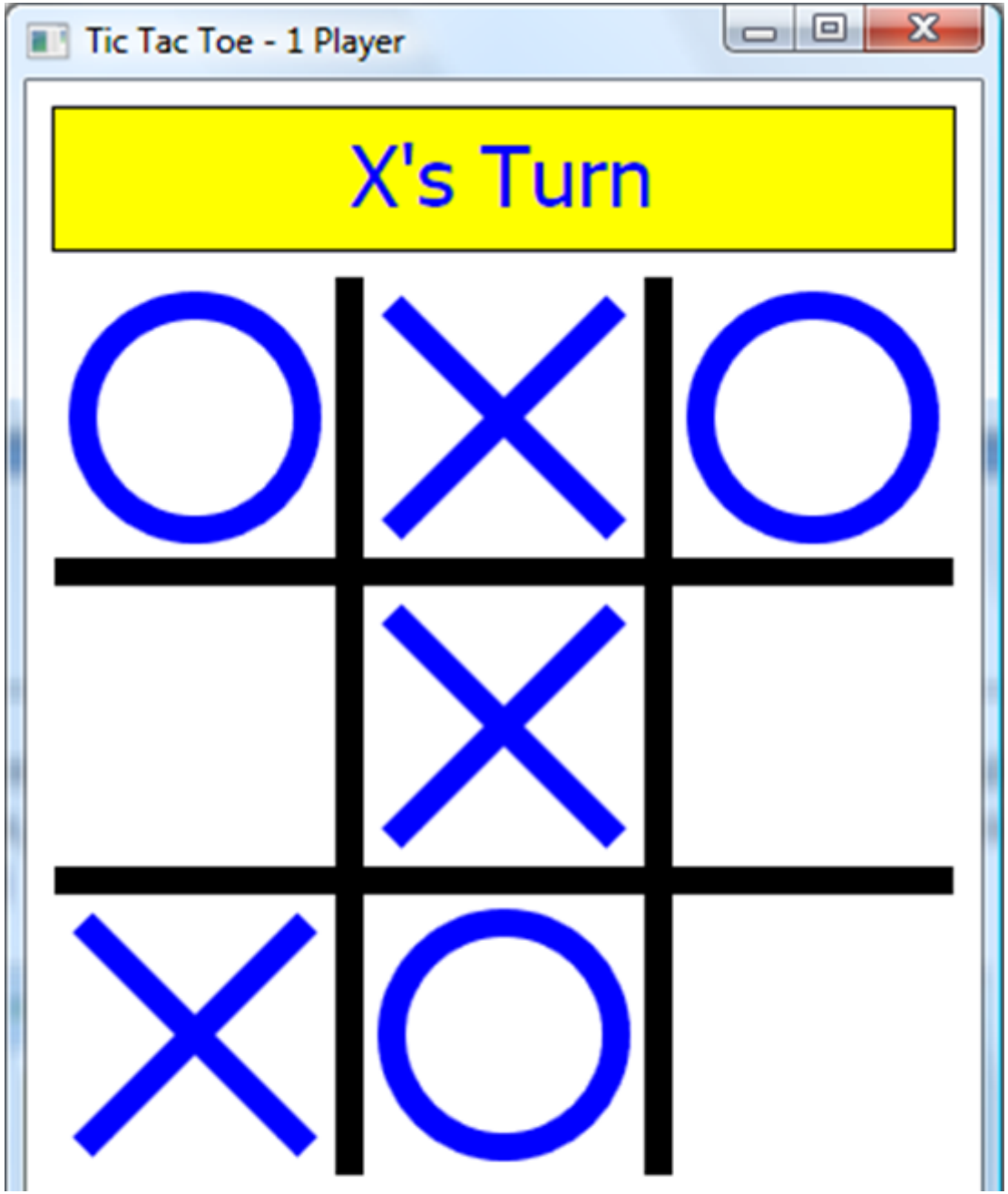
Notice, based on the logic we implemented, since the computer couldn't win or block, it took its next best move, the first available corner square.

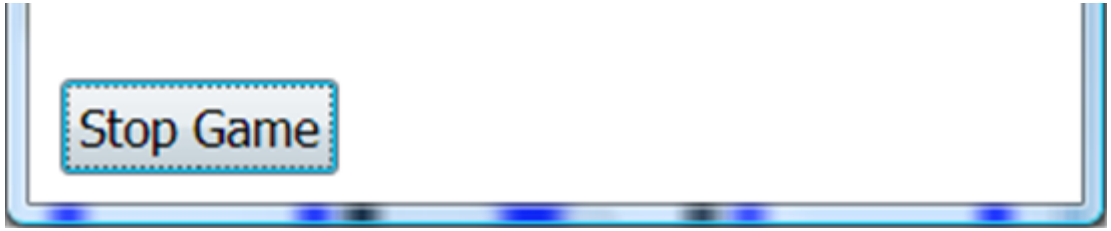
My next move was the lower left corner:



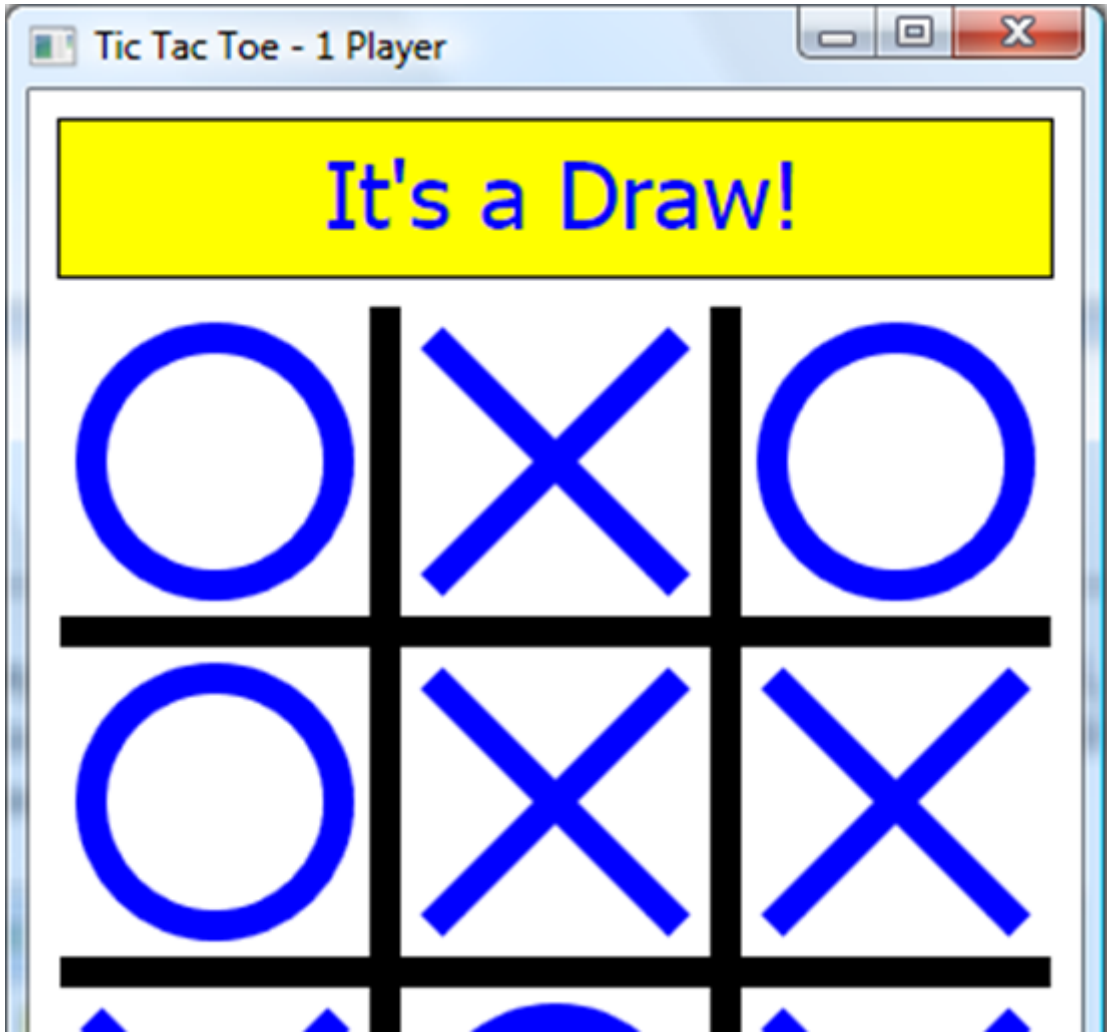
De geïmplementeerde computerlogica blokkeert mijn volgende zet.

Mijn volgende zet was om de computer te blokkeren, die op zijn beurt mij blokkeerde:





Ik bleef spelen (naar rechts van de middelste rij en, na het blok van de computer, een beweging naar rechts van de onderste rij) totdat we uiteindelijk eindigden in een gelijkspel:



I heard the little beep and the computer's ready to play again.

Tic Tac Toe Game Program Listing

Here is the complete listing of the **Tic Tac Toe** Small Basic program:

```
'Tic Tac Toe

InitializeProgram()

Sub InitializeProgram
  'graphics window
  GraphicsWindow.Width = 340
  GraphicsWindow.Height = 460
  'Draw message area
  GraphicsWindow.BrushColor = "Yellow"
  GraphicsWindow.FillRectangle(10, 10, 320, 50)
  GraphicsWindow.PenColor = "Black"
  GraphicsWindow.PenWidth = 2
```

```
GraphicsWindow.DrawRectangle(10, 10, 320, 50)
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.FontBold = "false"
GraphicsWindow.FontSize = 30
Message = "Game Stopped"
MessageX = 70
MessageArea = Shapes.AddText(Message)
DisplayMessage()
'draw grid
GraphicsWindow.BrushColor = "Black"
GraphicsWindow.FillRectangle(10, 170, 320, 10)
GraphicsWindow.FillRectangle(10, 280, 320, 10)
GraphicsWindow.FillRectangle(110, 70, 10, 320)
GraphicsWindow.FillRectangle(220, 70, 10, 320)
'define buttons
GraphicsWindow.BrushColor = "Black"
GraphicsWindow.FontSize = 16
StartStopButton = Controls.AddButton("Start Game", 10, 420)
OptionsButton = Controls.AddButton("Change Options", 110, 420)
ExitButton = Controls.AddButton("Exit", 240, 420)
'Default Options
NumberPlayers = 2
YouGoFirst = "true"
SmartComputer = "true"
SetTitle()
```

```
  'initialize box locations and marks
  x = 20
  y = 80
  For I = 1 to 9
    BoxX[I] = x
    BoxY[I] = y
    x = x + 110
    If (x > 240) Then
      x = 20
      y = y + 110
    EndIf
  EndFor
  GameStatus = "Stopped"
  Controls.ButtonClicked = ButtonClickedSub
  GraphicsWindow.MouseDown = MouseDownSub
  'possible wins
  PossibleWins[1] = "123"
  PossibleWins[2] = "456"
  PossibleWins[3] = "789"
  PossibleWins[4] = "147"
  PossibleWins[5] = "258"
  PossibleWins[6] = "369"
  PossibleWins[7] = "159"
  PossibleWins[8] = "357"
EndSub
```

```
Sub DisplayMessage
  Shapes.Move(MessageArea, MessageX, 15)
  Shapes.SetText(MessageArea, Message)
EndSub
Sub SetTitle
  If (NumberPlayers = 1) Then
    GraphicsWindow.Title = "Tic Tac Toe - 1 Player"
  Else
    GraphicsWindow.Title = "Tic Tac Toe - 2 Players"
  EndIf
EndSub
Sub ButtonClickedSub
  B = Controls.LastClickedButton
  If (GameStatus = "Stopped") Then
    If (B = ExitButton) Then
      Program.End()
    ElseIf (B = OptionsButton) Then
      SetOptions()
    ElseIf (B = StartStopButton) Then
      StartGame()
    EndIf
  ElseIf (GameStatus = "Playing") Then
    If (B = StartStopButton) Then
      'stop program
      Message = "Game Stopped"
```

```
      MessageX = 70
      StopGame()
    EndIf
  EndIf
EndSub
Sub SetOptions
  GraphicsWindow.Hide()
  TextWindow.Show()
  TextWindow.Title = "Tic Tac Toe"
  TextWindow.CursorLeft = 3
  TextWindow.CursorTop = 3
  TextWindow.WriteLine("TIC TAC TOE OPTIONS")
  TextWindow.WriteLine("")
  GetPlayers:
  TextWindow.CursorLeft = 3
  TextWindow.WriteLine("With one player, you play against the computer.")
  TextWindow.CursorLeft = 3
  TextWindow.WriteLine("With two players, you play against a friend.")
  TextWindow.CursorLeft = 3
  TextWindow.Write("How many players do you want (1 or 2)? ")
  NumberPlayers = TextWindow.ReadNumber()
  If (NumberPlayers < 1 Or NumberPlayers > 2) Then
    Goto GetPlayers
  EndIf
  If (NumberPlayers = 1) Then
```

```
GetWhoFirst:
TextWindow.WriteLine("")
TextWindow.CursorLeft = 3
TextWindow.WriteLine("You can go first or the computer can go first.")
TextWindow.CursorLeft = 3
TextWindow.Write("Who goes first (1-You, 2-Computer)? ")
T = TextWindow.ReadNumber()
If (T < 1 Or T > 2) Then
  Goto GetWhoFirst
EndIf
If (T = 1) Then
  YouGoFirst = "true"
Else
  YouGoFirst = "false"
EndIf
GetSmart:
TextWindow.WriteLine("")
TextWindow.CursorLeft = 3
TextWindow.WriteLine("Computer can make random moves or smart moves.")
TextWindow.CursorLeft = 3
TextWindow.Write("What do you want (1-Random, 2-Smart)? ")
T = TextWindow.ReadNumber()
If (T < 1 Or T > 2) Then
  Goto GetSmart
EndIf
```

```
    If (T = 2) Then
      SmartComputer = "true"
    Else
      SmartComputer = "false"
    EndIf
  EndIf
  SetTitle()
  TextWindow.Hide()
  GraphicsWindow.Show()
EndSub
Sub StartGame
  GameStatus = "Playing"
  XTurn = "true"
  Message = "X's Turn"
  MessageX = 115
  DisplayMessage()
  'clear boxes
  GraphicsWindow.BrushColor = GraphicsWindow.BackgroundColor
  For I = 1 To 9
    BoxMark[I] = ""
    GraphicsWindow.FillRectangle(BoxX[I] - 5, BoxY[I] - 5, 90, 90)
  EndFor
  GraphicsWindow.BrushColor = "Black"
  GraphicsWindow.FontSize = 16
  Controls.SetButtonCaption(StartStopButton, "Stop Game")
```

```
  Controls.HideControl(OptionsButton)
  Controls.HideControl(ExitButton)
  NumberClicks = 0
  If (NumberPlayers = 1 And YouGoFirst = "false") Then
    ComputerTurn()
  EndIf
EndSub
Sub StopGame
  'restore buttons
  GameStatus = "Stopped"
  GraphicsWindow.BrushColor = "Black"
  GraphicsWindow.FontSize = 16
  Controls.SetButtonCaption(StartStopButton, "Start Game")
  Controls.ShowControl(OptionsButton)
  Controls.ShowControl(ExitButton)
  DisplayMessage()
EndSub

Sub MouseDownSub
  If (GameStatus = "Playing") Then
    'find which box was clicked
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    ClickedBox = 0
    For I = 1 To 9
```

```
      If (x > BoxX[I] And x < BoxX[I] + 80) Then
        If (y > BoxY[I] And y < BoxY[I] + 80) Then
          ClickedBox = I
          Goto GotIt
        EndIf
      EndIf
    EndFor
    GotIt:
    If (ClickedBox <> 0) Then
      'if already clicked then exit
      If (BoxMark[ClickedBox] <> "") Then
        Goto LeaveSub
      EndIf
      MarkAndCheck()
    EndIf
  EndIf
LeaveSub:
EndSub

Sub MarkAndCheck
  NumberClicks = NumberClicks + 1
  If (XTurn) Then
    BoxMark[ClickedBox] = "X"
    DrawX()
    XTurn = "false"
```

```
    Message = "O's Turn"
    MessageX = 115
    DisplayMessage()
  Else
    BoxMark[ClickedBox] = "O"
    DrawO()
    XTurn = "true"
    Message = "X's Turn"
    MessageX = 115
    DisplayMessage()
  EndIf
  'check for win - will establish a value for WhoWon
  CheckForWin()
  If (WhoWon <> "") Then
    Sound.Stop(Program.Directory + "\tada.wav")
    Sound.Play(Program.Directory + "\tada.wav")
    Message = WhoWon + " Wins!"
    MessageX = 115
    StopGame()
  ElseIf (NumberClicks = 9) Then
    'draw
    Sound.Stop(Program.Directory + "\beep.wav")
    Sound.Play(Program.Directory + "\beep.wav")
    Message = "It's a Draw!"
    MessageX = 95
```

```
    StopGame()
  EndIf
  If (NumberPlayers = 1 And WhoWon = "") Then
    If (XTurn = "true" And YouGoFirst = "false") Or (XTurn = "false" And YouGoFirst =
"true") Then
      ComputerTurn()
    EndIf
  EndIf
EndSub
Sub DrawX
  'draw blue X at ClickedBox
  GraphicsWindow.PenColor = "Blue"
  GraphicsWindow.PenWidth = 10
  GraphicsWindow.DrawLine(BoxX[ClickedBox], BoxY[ClickedBox], BoxX[ClickedBox] + 80,
BoxY[ClickedBox] + 80)
  GraphicsWindow.DrawLine(BoxX[ClickedBox], BoxY[ClickedBox] + 80, BoxX[ClickedBox] +
80, BoxY[ClickedBox])
EndSub
Sub DrawO
  'draw blue O at Clicked Box
  GraphicsWindow.PenColor = "Blue"
  GraphicsWindow.PenWidth = 10
  GraphicsWindow.DrawEllipse(BoxX[ClickedBox], BoxY[ClickedBox], 80, 80)
EndSub
Sub CheckForWin
  WhoWon = ""
  'check all possible for wins
```

```
  For I = 1 To 8
    For J = 1 To 3
      BoxNumber[J] = Text.GetSubText(PossibleWins[I], J, 1)
      Mark[J] = BoxMark[BoxNumber[J]]
    EndFor
    If (Mark[1] = Mark[2] And Mark[1] = Mark[3] And Mark[2] = Mark[3] And Mark[1] <> "")
Then
      'we have a winner
      WhoWon = Mark[1]
      For J = 1 To 3
        GraphicsWindow.BrushColor = "Red"
        GraphicsWindow.FillRectangle(BoxX[BoxNumber[J]] - 5, BoxY[BoxNumber[J]] - 5, 90,
90)
        ClickedBox = BoxNumber[J]
        If (WhoWon = "X") Then
          DrawX()
        Else
          DrawO()
        EndIf
      EndFor
    EndIf
  EndFor
EndSub
Sub ComputerTurn
  If (SmartComputer <> "true") Then
    'random logic
```

```
  'put mark in Nth available square
  N = Math.GetRandomNumber(9 - NumberClicks)
  I = 0
  For ClickedBox = 1 To 9
    If (BoxMark[ClickedBox]  = "") Then
      I = I + 1
      If I = N Then
        Goto GotMark
      EndIf
    EndIf
  EndFor
  GotMark:
  'put mark in ClickedBox
  MarkAndCheck()
Else
  'smart computer
  BestMoves[1] = 5
  BestMoves[2] = 1
  BestMoves[3] = 3
  BestMoves[4] = 7
  BestMoves[5] = 9
  BestMoves[6] = 2
  BestMoves[7] = 4
  BestMoves[8] = 6
  BestMoves[9] = 8
```

```
    'determine who has what mark
    If (YouGoFirst) Then
      ComputerMark = "O"
      PlayerMark = "X"
    Else
      ComputerMark = "X"
      PlayerMark = "O"
    EndIf
    'Step 1 (K = 1) - check for win - see if two boxes hold computer mark and one is
empty
    'Step 2 (K = 2) - check for block - see if two boxes hold player mark and one is
empty
    For K = 1 To 2
      If K = 1 Then
        MarkToFind = ComputerMark
      Else
        MarkToFind = PlayerMark
      EndIf
      For I = 1 To 8
        N = 0
        EmptyBox = 0
        For J = 1 To 3
          BoxNumber[J] = Text.GetSubText(PossibleWins[I], J, 1)
          Mark[J] = BoxMark[BoxNumber[J]]
          If (Mark[J] = MarkToFind) Then
            N = N + 1
```

```
          ElseIf (Mark[J] = "") Then
            EmptyBox = BoxNumber[J]
          EndIf
        EndFor
        If (N = 2 And EmptyBox <> 0) Then
          'mark empty box to win (K = 1) or block (K = 2)
          ClickedBox = EmptyBox
          MarkAndCheck()
Ga naar LeaveComputerMove
        EindeAls
      Eindvoor
    Eindvoor
    'Stap 3 - vind de volgende beste zet
Voor I = 1 tot 9
Als (BoxMark[BestMoves[I]] = "") dan
        ClickedBox = BestMoves[I]
        MarkAndCheck()
Ga naar LeaveComputerMove
      EindeAls
    Eindvoor
  EindeAls
LeaveComputerMove:
Eindsub
```

Tic Tac Toe Game Programma Review

Het **Tic Tac Toe** spelprogramma is nu compleet. **Sla** het programma op en **voer** het uit en zorg ervoor dat het werkt zoals beloofd. Controleer of alle opties correct werken.

Als er fouten zijn in uw implementatie, gaat u terug naar de stappen van venster- en codeontwerp. Ga over de ontwikkelde code - zorg ervoor dat u begrijpt hoe verschillende delen van het programma zijn gecodeerd. Zoals vermeld in het begin van dit hoofdstuk, wordt het voltooide programma opgeslagen als **TicTacToe** in de map **KidGamesSB \KidGamesSB Programs \ TicTacToe.**

Tijdens het voltooien van dit programma zijn nieuwe concepten en vaardigheden die je zou moeten hebben opgedaan, onder meer:

- Goede stappen in programma-ontwerp.
- Meer oefening in het gebruik van subroutines.
- Hoe de computer intelligentie te geven bij het spelen van games.

Tic Tac Toe Game Programma Verbeteringen

Er zijn verschillende manieren om het **Tic Tac Toe** spel te verbeteren. Enkele mogelijkheden zijn:

- Gebruik afbeeldingen in het raster. Met zo'n verandering zou je iets anders dan X'en en O's kunnen gebruiken om de vierkanten te markeren. Probeer kleine digitale foto's (jpeg-bestanden) te gebruiken.
- Voeg betere berichten toe met een computertegenstander. Zoals geïmplementeerd, maakt de computer snel zijn beweging. Voeg misschien een vertraging toe (zie het object **Timer**) en misschien enkele kleurwijzigingen in het raster, zodat de computer de tijd heeft om u te vertellen waar de verplaatsing zal zijn.
- In de huidige computerlogica (stap 3) kiest de computer altijd hoeken of zijkanten in dezelfde volgorde. Om dingen een beetje onvoorspelbaarder te maken, voegt u logica toe zodat de computer willekeurig hoek- of zijwaartse bewegingen maakt op basis van het aantal hoek- of zijvakken dat leeg is.
- De computer in dit spel is behoorlijk slim, maar kan slimmer worden gemaakt. Tussen stap 2 en 3 in de huidige computerlogica zouden we nog een stap kunnen toevoegen waarbij de computer agressiever zou zijn. Laat de computer in deze nieuwe stap zoeken naar horizontale, verticale of diagonale lijnen met een van de markeringen en twee lege vierkanten. Door een van de lege vierkanten te kiezen, dwingt de computer je volgende zet af. Nog beter is om een plek te vinden waar met een extra markering de computer twee overwinningen instelt, waardoor het voor jou onmogelijk is om hem te blokkeren.
- Voor kleine kinderen is de computer misschien een beetje te slim. De willekeurige computer is een optie, maar gaat naar het andere uiterste van misschien niet slim genoeg zijn. Een verandering zou zijn om niveaus van intelligentie toe te voegen - waarbij de computer soms zijn 'slimme' logica gebruikt en soms zijn 'willekeurige' logica. Om dit te doen, kunt u een willekeurig getal van 1 tot 100 genereren en

op basis van een drempel bepalen of u een slimme of willekeurige zet wilt doen. Door de drempel te variëren, kun je de computer extreem slim of minder slim maken. We implementeren dergelijke logica in het volgende spelprogramma, **Match Game**.

Om dit boek in zijn geheel te kopen, zie de Computer Science For Kids website .