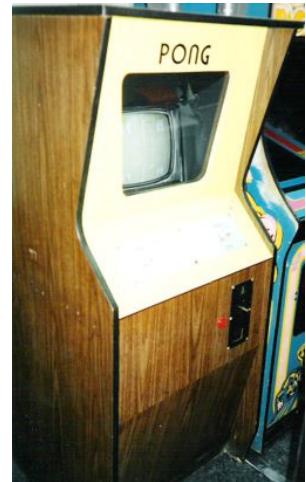


SwinGame: Computer Game Programming

Ping Pong

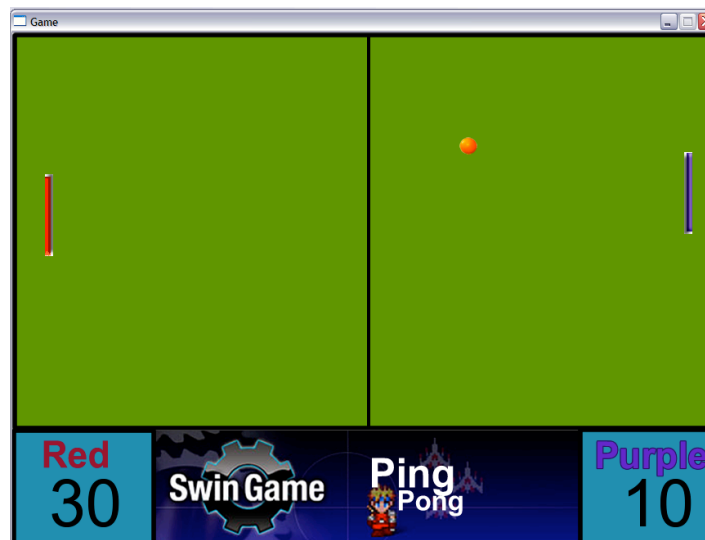
In this session you will build, from scratch, a small computer game called **Pong**. Did you know that *Pong* was the first video game to achieve widespread popularity in both arcade and home console versions? Pong launched the initial boom in the video game industry, so we think it's a great place to start writing your own games.

To build Pong, we will use Visual Basic .NET and the SwinGame SDK. Visual Basic is a computer programming language that can be used to create a variety of programs, including games. The SwinGame SDK is a games development kit that was made by the Professional Software Development group at Swinburne University, you can find out more about it at www.swingame.com.



Finished Game

Below is a picture of how the game will look after we have finished. It's a two player game where each player controls a paddle. The ball is served and each player moves their paddle to return the ball. If a player misses their opponent scores a point.



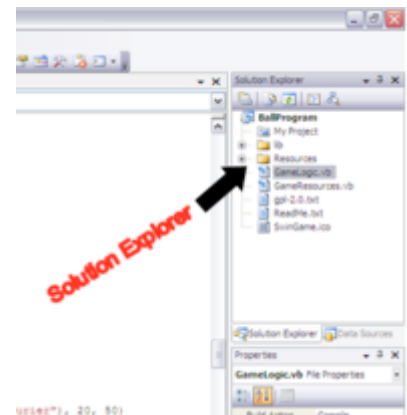
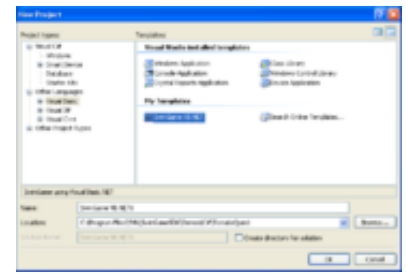
Getting Started


Before we can start creating the game we need to make sure that we have got the SwinGame project setup correctly. Follow these steps to get a "Hello World" program working with SwinGame.

1. Open Visual Studio, and click File > New > Project.
2. Select the Visual Basic/Basic Language and Click the SwinGame VB.NET Project Template. Click OK.
3. Select File - Save. Enter "**PingPong**" as the file name, and choose the location it is to be saved and click OK.
4. In the Solution Explorer on the right hand side of the screen double click on "GameLogic". You should be presented with the following code:

Module GameLogic

```
Public Sub Main()  
    'Opens a new Graphics Window  
    Core.OpenGraphicsWindow("Game", 800, 600)  
  
    'Open Audio Device  
    Audio.OpenAudio()  
  
    'Load Resources  
    LoadResources()  
  
    'Game Loop  
    Do  
        'Clears the Screen to Black  
        SwinGame.Graphics.ClearScreen()  
  
        Graphics.FillRectangle(Color.Red, 20, 200, 200, 100)  
        Graphics.FillRectangle(Color.Green, 220, 200, 200, 100)  
        Graphics.FillRectangle(Color.Blue, 420, 200, 200, 100)  
  
        Text.DrawText("Hello World", Color.Red, GameFont("Courier"), 20, 310)  
        Text.DrawText("Hello World", Color.Green, GameFont("Courier"), 220, 310)  
        Text.DrawText("Hello World", Color.Blue, GameFont("Courier"), 420, 310)  
  
        Text.DrawFramerate(0, 0, GameFont("Courier"))  
        Text.DrawText("Hello World", Color.White, GameFont("ArialLarge"), 50, 50)  
  
        'Refreshes the Screen and Processes Input Events  
        Core.RefreshScreen()  
        Core.ProcessEvents()  
  
    Loop Until SwinGame.Core.WindowCloseRequested() = True  
  
    'Free Resources and Close Audio, to end the program.  
    FreeResources()  
    Audio.CloseAudio()  
End Sub  
  
End Module
```



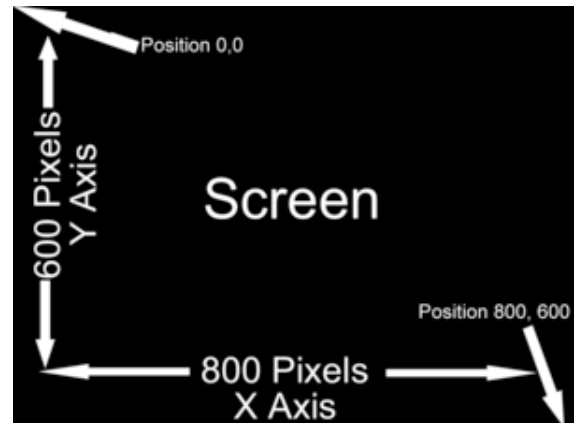
The purpose of this code is to create a starting point for your project. If you press the "Start Debugging" button at the top of the screen (looks like a green arrow , F5 works too) you will see what it does. Have a look, then close the window.

You can basically read what it is doing

'comments are in green with a single parenthesis at the start like this sentence

See if you can find the parts of the code that are doing the following things:

1. Loading all the resources you need for the game
2. Opening up a blank screen 800 [pixels](#) wide (x axis) by 600 tall (y axis).
3. Starting a [Loop](#), inside which it does:
 - a. Draws 3 filled rectangles setting their sizes and different colours width height and position
 - b. Putting the game's "[Frame rate](#)" up on the screen in the top right hand corner
 - c. Writing the words "Hello World" 3 times in different colours below the rectangles
 - d. Writing the words "Hello World" once at the top in large letters
4. Telling the loop to stop when someone closes the window



Exercise 1

To help your understanding of what is going on in this code do the following:

1. Change the big "Hello World" to your name
2. Move the Red Rectangle so it is in the bottom right hand corner
3. Make the Red Rectangle twice as big

You have completed your "Hello World" next we look at adding an image.

Adding Images

We will add the images that we need for our game, the images were made in Macromedia Fireworks and saved as PNG (Portable Networks Graphics), for the purpose of this activity all your resources have been provided for you but you could of course go back later and make your own.

Removing Hello World

So now we want to get rid of all that “Hello World” stuff so we can put in our own stuff so **delete** the following code from the GameLogic.vb file we were playing with before:

```
SwinGame.Graphics.ClearScreen()

Graphics.FillRectangle(Color.Red, 20, 200, 200, 100)
Graphics.FillRectangle(Color.Green, 220, 200, 200, 100)
Graphics.FillRectangle(Color.Blue, 420, 200, 200, 100)

Text.DrawText("Hello World", Color.Red, GameFont("Courier"), 20, 310)
Text.DrawText("Hello World", Color.Green, GameFont("Courier"), 220, 310)
Text.DrawText("Hello World", Color.Blue, GameFont("Courier"), 420, 310)

Text.DrawFramerate(0, 0, GameFont("Courier"))
Text.DrawText("Hello World", Color.White, GameFont("ArialLarge"), 50, 50)
```

Now if you run the code by hitting the “Start Debugging” button (or F5) you should see a blank screen 800x600 in size.

Copy the Images

From your resource folder **copy** the following images:

ball.png



redpaddle.png



purplepaddle.png



table.png



This picture is
800 x 600

And **paste** them in your image resources folder for the PingPong game which is in “My Documents” for example on my computer it is in this folder:

***C:\Documents and Settings\rmercercer\My Documents\Visual Studio
2005\Projects\PingPong\PingPong\Resources\images***

Note: If you create these images yourself make sure they are stored as 32bit png files.

Loading the Images

Now we need to load these images into your game. You will remember we played with the Game Logic before but this time we are adding new resources so **open** the **GameResources.vb** file from the Solution Explorer and scroll down to the section titled: "Private Sub LoadImages()" and add the following code directly below it:

```
Sub LoadImages()  
    NewImage("ball", "ball.png")  
    NewImage("table", "table.png")  
    NewImage("redpaddle", "redpaddle.png")  
    NewImage("purplepaddle", "purplepaddle.png")  
End Sub
```

This will add the pictures to our Game Resources and when we can refer to the pictures from now on using the name "ball" or "redpaddle", etc.

Drawing the background

Now we need to add the background picture to our presently empty black frame. Open up the GameLogic.vb file again from the Solution Explorer; find the **Do loop** (this is where all the action in our game occurs). **Add** the code so that the loop appears as shown below. You will also need to edit the code Core.RefreshScreen(65) to include the value 65, this will ensure that the game runs at the right speed.

```
'Game Loop  
Do  
    DrawGame()  
  
    'Refreshes the Screen and Processes Input Events  
    Core.RefreshScreen(65)  
    Core.ProcessEvents()  
  
Loop Until SwinGame.Core.WindowCloseRequested() = True
```

Now we need to add the code to define what occurs when *DrawGame* is called. Go to the top of the file and **add** code just below *Module GameLogic* so that it appears as shown below.

```
Module GameLogic  
  
Sub DrawGame()  
    Graphics.DrawBitmap(GameImage("table"), 0, 0)  
End Sub  
  
Sub Main()...
```

This code basically says to draw the "Background" image at the top left corner (0,0). The image is the same size as the window (800,600) so it takes up the whole background.

Hit F5 and see the image appear drawn as the background.

Getting Started with Sprites

Now the Ball and Paddles should all move, and any thing that will move or perform some action will be a "Sprite" in the program. To start we need to declare variables to store the values for the ball, and the two paddles. At the top of the file just below *Module GameLogic* add code so that it appears as shown in the following code.

```
Module GameLogic

    Const FIELD_TOP = 7
    Const FIELD_BOTTOM = 452
    Const FIELD_LEFT = 7
    Const FIELD_RIGHT = 794
    Const PADDLE_OFFSET = 50

    Dim Ball As Sprite
    Dim PurplePaddle As Sprite
    Dim RedPaddle As Sprite

    Sub DrawGame() ...
```

Now we need to initialise these variables with some values. To do so open up the GameLogic file and underneath LoadResources() add the following code:

```
...
LoadResources()

ball = Graphics.CreateSprite(GameImage("ball"))
redPaddle = Graphics.CreateSprite(GameImage("redpaddle"))

'Game Loop
Do ...
```

Please Note: We are going to get this working with just the ball and red paddle for the moment, later we will come back and add the Purple paddle.

Initialising the Sprites

Before we can draw of more these sprites we need to set their location. For this purpose we can create a **StartUpGame** procedure that will be called at the start each game. The code for this is shown below. Add it to your program just before the code that declares the ball and paddles, and before the DrawGame procedure.

```
...
Dim RedPaddle As Sprite

Sub StartUpGame()
    'Position of ball
    ball.X = 400 - ball.Width / 2
    ball.Y = 240 - ball.Height / 2

    'Position of RedPaddle
    redPaddle.X = FIELD_LEFT + PADDLE_OFFSET
    redPaddle.Y = 240 - redPaddle.Height / 2

    'Play 3..2..1..
    Dim i As Integer

    For i = 3 to 1 step -1
        DrawGame()
        Text.DrawText(i.ToString(), Color.Black, GameFont("ArialLarge"), 370, 200)
        Core.RefreshScreen()
        Core.Sleep(300)
    Next
End Sub

Sub DrawGame() ...
```

Now that we have declared the **StartUpGame** procedure we can call it from the main part of the program. Locate the start of the *Game Loop* and **add** the call to StartUpGame there, the code is shown below.

```
...
redPaddle = Graphics.CreateSprite(GameImage("redpaddle"))

StartUpGame()

'Game Loop
Do ...
```

Drawing the Sprites

With the sprites created, and initialised it is now time to draw them within the DrawGame procedure. Add the two missing lines to the **DrawGame** procedure.

```
Sub DrawGame()  
    Graphics.DrawBitmap(GameImage("table"), 0, 0)  
    Graphics.DrawSprite(Ball)  
    Graphics.DrawSprite(RedPaddle)  
End Sub
```

If you hit "Start Debugging" now to view the program. You should see the following:

1. A countdown at the start of the game
2. The paddle and the ball are drawn in front of the background but will not be moving yet...

Exercise – Add the Purple Paddle

Now you need to go back over this code and add the missing code for the *PurplePaddle*. This will include the following sections:

1. Add the code to create the sprite in the main part of the program
2. Add the code in **StartUpGame** to initialise the *PurplePaddle* :
 - a. X is set to `FIELD_RIGHT - PADDLE_OFFSET - purplePaddle.Width`
 - b. Y is set to `240 - purplePaddle.Height / 2`
3. Draw the *PurplePaddle* in **DrawGame**

Solution – Add the Purple Paddle

In the main part of the program you need:

```
...
ball = Graphics.CreateSprite(GameImage("ball"))
redPaddle = Graphics.CreateSprite(GameImage("redpaddle"))
purplePaddle = Graphics.CreateSprite(GameImage("purplepaddle"))
...
```

In StartUpGame you need to add:

```
'Position of paddle
purplePaddle.X = FIELD_RIGHT - PADDLE_OFFSET - purplePaddle.Width
purplePaddle.Y = 240 - purplePaddle.Height / 2
```

DrawGame is now:

```
Sub DrawGame()
    Graphics.DrawBitmap(GameImage("table"), 0, 0)
    Graphics.DrawSprite(Ball)
    Graphics.DrawSprite(RedPaddle)
    Graphics.DrawSprite(PurplePaddle)
End Sub
```

Moving Sprites: Getting the Ball Moving

In order to get our ball moving we first need to edit the code in **StartUpGame** to set the movement of the ball so that it appears as shown below:

```
...
ball.X = 400 - ball.Width / 2
ball.Y = 240 - ball.Height / 2

'Ball is moving down... need to change this later
ball.Movement.Y = 1

'Send ball... left or right
If Rnd() < 0.5 Then
    ball.Movement.X = -2.5
Else
    ball.Movement.X = 2.5
End If

'Position of RedPaddle
...
```

This code sets the ball moving either left or right (with an even chance of going either way).

Updating the Sprite

The sprite's movement only changes the sprite when it is updated. We need to add something to the game loop so every time the program goes through the loop the sprite is updated which will make the ball move a bit each loop.

Locate the Game Loop and **UpdateBall()** above the call to *DrawGame*, you can see the result below.

```
'Game Loop
Do
    UpdateBall()
    DrawGame()

    'Refreshes the Screen and Processes Input Events
    Core.RefreshScreen(65)
    Core.ProcessEvents()


Loop Until SwinGame.Core.WindowCloseRequested() = True
```

Next we need to create the *UpdateBall* procedure. Go to the top of the file and just below the variable declarations, but before the *StartUpGame* procedure, add the code shown below.

```
...
Dim RedPaddle As Sprite

Sub UpdateBall()
    Graphics.UpdateSprite(Ball);
End Sub

Sub StartUpGame() ...
```

Now run the program (hit "Start Debugging" ) you should see the ball move either left or right... and then off the screen. To fix this we need to add some logic to the program that tells it to change the direction of the ball when it goes off the screen.

Testing for the outside border

To get the ball bouncing off the outside of the screen we need to add some tests to our game loop. We will use four **If** statements to test for the four different walls. Here we can use the FIELD_TOP, FIELD_BOTTOM, FIELD_LEFT, and FIELD_RIGHT values that we added to the top of the program. These values were taken by examining the borders in the background image so that the ball will appear to bounce off these walls.

To do this we need to reverse the X or Y movement of the ball. The following code tests if the ball has hit the bottom or left of the field and then reverses the movement appropriately. You will need to add the missing code to the *UpdateBall* procedure.

```
Sub UpdateBall()  
    Graphics.UpdateSprite(Ball);  
  
    'Tests to see if ball has hit an outside border  
    'bounces the ball off the wall  
  
    'if the ball has hit the bottom border  
    If ball.Y + ball.Height > FIELD_BOTTOM Then  
        ball.Y = FIELD_BOTTOM - ball.Height  
        ball.Movement.Y = -ball.Movement.Y  
    End If  
  
    'if the ball has hit the lhs border  
    If ball.X < FIELD_LEFT Then  
        ball.X = FIELD_LEFT  
        'Reverse the x movement  
        ball.Movement.X = -ball.Movement.X  
    End If  
End Sub
```

Please Note: We have to add the Height of the ball when testing the bottom as we need to check the bottom of the ball, and the Y value indicates where the top of the Ball is.

Exercise – Bouncing off the walls

See if you can figure out what the other 2 if statements should be. The solution can be found on the next page.

1. When you check the right you need to add the Width of the Ball as you want to check the right side of the ball.
2. Checking the top will just use Y as it is the top of the ball.

We also need to make the paddle move but that will be dependent on keyboard input and that comes later on.

Solution – Bouncing off the walls

Solution to testing for outside borders ():

```
'if the ball has hit the bottom border
If Ball.Y + Ball.Height > FIELD_BOTTOM Then
    Ball.Y = FIELD_BOTTOM - Ball.Height
    Ball.Movement.Y = -Ball.Movement.Y
End If

If Ball.Y < FIELD_TOP Then
    Ball.Y = FIELD_TOP
    Ball.Movement.Y = -Ball.Movement.Y
End If

'if the Ball has hit the lhs border
If Ball.X < FIELD_LEFT Then
    Ball.X = FIELD_LEFT
    'Reverse the x movement
    Ball.Movement.X = -Ball.Movement.X
End If

If Ball.X + Ball.Width > FIELD_RIGHT Then
    Ball.X = FIELD_RIGHT - Ball.Width
    'Reverse the x movement
    Ball.Movement.X = -Ball.Movement.X
End If
```

Playing Sounds

We want to add a sound for when the ball bounces off the outside walls and a different one when it hits the paddle. We are not up to putting in the code for the paddle yet so we will just focus on getting the sound for the outside walls working.

Adding the sounds

We will need to get the four sounds for these and other effects and put them in the project folder for audio. We will use .wav and .aif files. Once again you can find your own as with the images or use these ones supplied to save time.

- bounce.wav - the noise we will use when the ball bounces off the walls
- hit.wav - the noise we will use when the ball hits the paddle
- beep.wav – this will be used in the 1..2..3 at the start
- backwall.aif – this will be used when a player scores

Copy the sound files into the **Sounds** folder of your Resources folder of your BallProgram project for example on my computer the path for this folder is: C:\Documents and Settings\rmercet\My Documents\Visual Studio 2005\Projects\BallProgram\BallProgram\Resources.

Now we need to add them to the code. Open up the GameResource.vb file from your project and find *Private Sub LoadSounds()* underneath this is where we add any sounds we want for our project similar to when we added the images. The code we need to add for loading the sounds is:

```
Private Sub LoadSounds()  
    NewSound("bounce", "bounce.wav")  
    NewSound("backwall", "backwall.aif")  
    NewSound("hit", "hit.wav")  
    NewSound("beep", "beep.wav")  
End Sub
```

So from now on in the code "bounce.wav" will be referred to as "bounce", etc.

Now open up the GameLogic source as we need to tell the program when to play the sounds. The *bounce* sound should play when the ball hits one of the walls. You will remember that we previously added some "If" statement to test when the ball hit the wall in the *UpdateBall* procedure so it could bounce the ball. All we need to do now is find them and add a line of code to tell it to play the sound as well as changing the direction of the ball. Add the following line of code to the body (between the "if" and "End If") of the four If statements:

```
Audio.PlaySoundEffect(GameSound("bounce"))
```

so for example one of them will look like this now:

```
If Ball.Y < FIELD_TOP Then  
    Ball.Y = FIELD_TOP  
    Ball.Movement.Y = -ball.Movement.Y  
    Audio.PlaySoundEffect(GameSound("bounce"))  
End If
```

Once that is added to all four If statements then hit "Start Debugging" and test it out. It should play the "bounce" sound every time the ball hits an outside border.

Note: At this point we have not used the paddle sound because we have not added the code to get the paddle moving, we will do so next however

Exercise – The crowd goes wild

Play the *backwall* sound if the ball hits the left or right hand wall.

Note: If you find your own make sure they are in either .wav or .ogg format. You could even record your own in a program like "GarageBand", "Sound Recorder" or Audacity. Or search for *.wav files on your computer. If you do source your own sounds make sure you call them. A good site I found for sounds is www.soundsnap.com

If you want some background music in your game playing in loop it is almost the same process as we used when we added the other sounds.

1. First copy the sound to the sound resource folder of your project (I have included the file "amb.mp3" in the resource folder for you to test this)
2. Add this line to the Load Music section of Gameraresources.vb

```
NewMusic ("amb", "amb.mp3")
```

3. Then add the following line in the Gamelogic.vb after the bit that says "Load Resources () "

```
Audio.PlayMusic (GameMusic ("amb"))
```

Then you will have some nice ping pong noise looping in the background. You can play any MP3 file so you could use a song or any other MP3 files for your music.

Exercise – 1.. 2.. 3.. Beep it Out

Play the *beep* sound with the 1, 2, 3 being displayed in the *StartUpGame* procedure. This way it will beep 1, beep 2, and beep 3... then the game will start.

Keyboard Input

This is an easy step as we have already added the paddles and all we need to do is tell it to move up and down. For this we are going to need to add a **HandleInput** procedure to the game loop.

```
'Game Loop
Do
    UpdateBall()
    HandleInput()

    DrawGame()

    'Refreshes the Screen and Processes Input Events
    Core.RefreshScreen(65)
    Core.ProcessEvents()

Loop Until SwinGame.Core.WindowCloseRequested() = True
```

Now go back to the top of the file and add the *HandleInput* below the variable declarations.

```
...
Dim RedPaddle As Sprite

Sub HandleInput()
    'Clear Movement
    redPaddle.Movement.X = 0
    redPaddle.Movement.Y = 0
    purplePaddle.Movement.X = 0
    purplePaddle.Movement.Y = 0

    'moves the paddle when user presses left and right keys
    If Input.IsKeyPressed(SwinGame.Keys.VK_A) Then
        redPaddle.Movement.Y = -2
    End If

    If Input.IsKeyPressed(SwinGame.Keys.VK_Z) Then
        redPaddle.Movement.Y = 2
    End If

    'Move Paddles
    Graphics.MoveSprite(purplePaddle)
    Graphics.MoveSprite(redPaddle)
End Sub

Sub UpdateBall() ...
```

So basically when the user hits the A key it will move the paddle sprite up at a rate of 2 pixels per loop. If the user holds down the Z key it will move the paddle sprite down at a rate of 2 pixels per loop. You can change these numbers if you want it to move faster or slower.

Exercise – Getting Purple Moving

- Add the code to make the PurplePaddle go up and down (using “Up” and “Down” keys)
- Add the code to stop the paddles from disappearing off the screen to the top and bottom. (Think about the Y position of the paddle) - Solution on next page

"But the ball just goes through the paddle" I hear you say well that comes next in "Sprite Collisions".

Solution – Getting Purple Moving

```
Sub HandleInput()  
    'Clear Movement  
    redPaddle.Movement.X = 0  
    redPaddle.Movement.Y = 0  
    purplePaddle.Movement.X = 0  
    purplePaddle.Movement.Y = 0  
  
    'moves the paddle when user presses left and right key  
    If Input.IsKeyPressed(SwinGame.Keys.VK_A) Then  
        redPaddle.Movement.Y = -2  
    End If  
  
    If Input.IsKeyPressed(SwinGame.Keys.VK_Z) Then  
        redPaddle.Movement.Y = 2  
    End If  
  
    If Input.IsKeyPressed(SwinGame.Keys.VK_UP) Then  
        purplePaddle.Movement.Y = -2  
    End If  
  
    If Input.IsKeyPressed(SwinGame.Keys.VK_DOWN) Then  
        purplePaddle.Movement.Y = 2  
    End If  
  
    'Move Paddles  
    Graphics.MoveSprite(purplePaddle)  
    Graphics.MoveSprite(redPaddle)  
  
    'stops the paddles moving off the screen  
    If purplepaddle.Y < FIELD_TOP Then  
        purplepaddle.Y = FIELD_TOP  
    End If  
  
    If purplepaddle.Y + purplepaddle.Height > FIELD_BOTTOM Then  
        purplepaddle.Y = FIELD_BOTTOM - purplepaddle.Height  
    End If  
  
    If redpaddle.Y < FIELD_TOP Then  
        redpaddle.Y = FIELD_TOP  
    End If  
  
    If redpaddle.Y + redPaddle.Height > FIELD_BOTTOM Then  
        redpaddle.Y = FIELD_BOTTOM - redPaddle.Height  
    End If  
End Sub
```


Sprite Collisions

Although it is fun watching a ball bounce aimlessly around the screen it would be nice if we could interact with it. To do this we need to tell our program to look out for a collision, namely the collision between the ball and the paddles. If this collision occurs then obviously we need to tell the program to bounce the ball off in the other direction the same as if we had hit the bottom or top by reversing the Y axis of the path of the ball.

We can now create a new **CheckCollision** procedure that checks if the ball has collided with either of the paddles and acts accordingly. The new game loop will look like this:

```
'Game Loop
Do
    UpdateBall()
    HandleInput()
    CheckCollisions()

    DrawGame()

    'Refreshes the Screen and Processes Input Events
    Core.RefreshScreen(65)
    Core.ProcessEvents()

Loop Until SwinGame.Core.WindowCloseRequested() = True
```

The code to test if there is a collision between the two sprites needs to be added into the new *CheckCollision* code:

```
...
Dim RedPaddle As Sprite

Sub CheckCollisions()
    'if the ball and paddle have collided
    If Physics.HaveSpritesCollided(ball, redPaddle) Then
        'reverse the x axis of the path
        ball.Movement.Y = -ball.Movement.Y
    End If
End Sub

Sub HandleInput() ...
```

Exercise – Purple is Hitting Back

1. Play the "hit" sound play when there is a collision between the ball and the paddle (easy)
2. Add the code for collisions with the PurplePaddle to the game so you can play against a friend

Tip: You can of course copy, paste and edit the from the code for the redpaddle for the purplepaddle.

Advanced Sprite Collision

As you play the game for a while you will notice something is not right with what happens when the ball hits the side of the paddle. It seems to get stuck to the paddle, work its way across and then gets free. This is to do with the fact that the game does not understand side collisions. In order to make it understand this we need to add a more advanced collision procedure. Add the following code:

```
...
Dim RedPaddle As Sprite

Sub PerformCollision(ByVal ball As Sprite, ByVal paddle As Sprite)
    Dim hitLine, lines() As LineSegment
    Dim vectTemp As Vector
    Dim shortest As Double
    Dim temp, ballCenter As Point2D

    'Get rectangles that surround the paddle and ball
    Dim pdlRect As Rectangle = Shapes.CreateRectangle(paddle)
    Dim ballRect As Rectangle = Shapes.CreateRectangle(ball)

    'Get the 4 lines that make up the paddle - top, bottom, left and right
    lines = Shapes.LinesFromRect(pdlRect)

    'Get the vector to move the ball sprite out of the paddle sprite
    ' This vector indicates how to move the ball to back it out of the paddle
    Dim mv As Vector
    mv = Physics.VectorOutOfRectFromRect(ballRect, pdlRect,
ball.Movement.AsVector())

    'Now use the vector to move the ball out of the paddle
    Graphics.MoveSprite(ball, mv)

    'Get the center point of the ball
    ballCenter = Shapes.CenterPoint(ball)

    'Find the side of the paddle that is closest to the ball
    ' this is the side that it hit... so we will bounce off this side
    ' it must be < than width as we did collide... so start shortest at
    ' the width of the ball
    shortest = ball.Width

    'find the closest point
    For i As Integer = 0 To 3
        'Find the point on the line closest to the center of the ball
        temp = Shapes.ClosestPointOnLine(ballCenter, lines(i))
        vectTemp = Physics.VectorFromPoints(ballCenter, temp)

        'If this is shorter than the current shortest - change shortest
        If Physics.Magnitude(vectTemp) < shortest Then
            shortest = Physics.Magnitude(vectTemp)
            hitLine = lines(i)
        End If
    Next

    'Get the ball to bounce off the line that it hit...
    ' SwinGame will take care of calculating the angles etc.
    Physics.CircleCollisionWithLine(ball, hitLine)
End Sub

Sub CheckCollisions()...
```

Now change the collision code in *CheckCollisions* to the following (oh and don't forget to add the line for the "hit" sound to this as well):

```
'ball collision with paddles
If Physics.HaveSpritesCollided(Ball, PurplePaddle) Then
    PerformCollision(Ball, PurplePaddle)
End If

If Physics.HaveSpritesCollided(Ball, RedPaddle) Then
    PerformCollision(Ball, RedPaddle)
End If
```

Adding Spin and Acceleration

Now we are going to make it possible to “spin” the ball as it comes off the paddles. Firstly we need to fix the serve so that it goes straight left or right, and not up or down. Locate the code that sets the ball’s movement in *StartUpGame* and set Y so that it is always assigned the value 0, as shown below.

```
...
ball.Y = 240 - ball.Height / 2

ball.Movement.Y = 0

If Rnd() < 0.5 Then
```

Now locate *CheckCollisions* code and change it so that it appears as shown below. You will need to write the code for the RedPaddle, which is very similar to the code shown.

```
Sub CheckCollisions()
    'Ball collision with paddles
    If Physics.HaveSpritesCollided(ball, purplepaddle) Then
        Audio.PlaySoundEffect(GameSound("hit"))
        PerformCollision(ball, purplePaddle)

        'Add spin (30% of paddle movement)
        ball.Movement.Y = ball.Movement.Y + 0.3 * purplePaddle.Movement.Y

        'Accelerate by 5%
        ball.Movement.Y = ball.Movement.Y * 1.05
        ball.Movement.X = ball.Movement.X * 1.05
    End If
    ...
End Sub
```

Play the game, and you will notice if you play it for a while that the ball starts to move very, very, fast. In fact if you play it for long enough you can see it skip over the paddles totally! We need to limit the speed the ball can move. The code for this can be added to the end of the *CheckCollisions* code, and the changes are shown on the following page with the code for the RedPaddle.

Solution – Spin and Acceleration + Limits

```
Sub CheckCollisions()  
    'Ball collision with paddles  
    If Physics.HaveSpritesCollided(ball, purplepaddle) Then  
        Audio.PlaySoundEffect(GameSound("hit"))  
        PerformCollision(ball, purplePaddle)  
  
        'Add spin (30% of paddle movement)  
        ball.Movement.Y = ball.Movement.Y + 0.3 * purplePaddle.Movement.Y  
  
        'Accelerate by 5%  
        ball.Movement.Y = ball.Movement.Y * 1.05  
        ball.Movement.X = ball.Movement.X * 1.05  
    End If  
  
    If Physics.HaveSpritesCollided(ball, redpaddle) Then  
        PerformCollision(ball, redPaddle)  
        Audio.PlaySoundEffect(GameSound("hit"))  
  
        'Add spin (30% of paddle movement)  
        ball.Movement.Y = ball.Movement.Y + 0.3 * redPaddle.Movement.Y  
  
        'Accelerate by 5%  
        ball.Movement.Y = ball.Movement.Y * 1.05  
        ball.Movement.X = ball.Movement.X * 1.05  
    End If  
  
    If Physics.Magnitude(ball.Movement.AsVector) > Ball.Width Then  
        Dim v As Vector  
        v = Physics.LimitMagnitude(ball.Movement.AsVector, Ball.Width)  
        ball.Movement.SetTo(v)  
    End If  
End Sub
```

Next let's get a Scoring feature for our game so you can keep track with who is winning.

Scoring

So let's set it up so when the red player misses the ball the purple player gets 10 points and vice versa. The first thing we need to do is declare two variables (dim) as Integers (integers are just whole numbers like 1 and 50 but not 1.25). These need to be added to the top of the GameLogic file just below the other variables.

```
Dim RedPaddle As Sprite
Dim ScoreRed As Integer
Dim ScorePurple As Integer

Sub CheckCollisions() ...
```

Next we need to initialise these to 0; the score each player will start with. This needs to be done in the *Main* procedure, just below the *LoadResources* call.

```
LoadResources()

ScoreRed = 0
ScorePurple = 0

'Create balls and paddles
...
```

Next we just need to tell it to add to the opponents score when the ball is missed. You will remember our code already knows when the ball hits any of the walls. We just need to find out when it hits the left or right walls and when it does add 1 to ScoreRed or ScorePurple depending on if it is the left or the right border it has hit. The code for adding 1 to the score is simple, let's say the ball hits the right border then red would get 1 point. The code for this is:

```
ScoreRed = ScoreRed + 1
```

Find the correct location for this in the *UpdateBall* procedure. Remember to add it for both the Red and Purple players.

Now to display that score on the screen below the word "Red" you need to add the following code to the *DrawGame* procedure, add it near the end of that procedure (you will remember it from the "Hello World" tutorial). The new *DrawGame* is shown below.

```
Sub DrawGame()
    'Drawing the game elements
    Graphics.DrawBitmap(GameImage("table"), 0, 0)
    Graphics.DrawSprite(ball)
    Graphics.DrawSprite(PurplePaddle)
    Graphics.DrawSprite(RedPaddle)

    'Putting the score up
    Text.DrawText(ScoreRed.ToString(), Color.White, GameFont("ArialLarge"), 21,
511)
    Text.DrawText(ScorePurple.ToString(), Color.White, GameFont("ArialLarge"),
650, 511)
End Sub
```

Implementing Games

At the moment the game continues even after a player scores a goal. Lets change this so that there is a short break between each goal. For this we can use the original *StartUpGame* code and a new **ShowScoreScreen** procedure. The following is the new code for the *UpdateBall* procedure.

```
Sub UpdateBall()  
    'Tests to see if ball has hit an outside border  
    'bounces the ball off the wall  
    If ball.Y + ball.Height > FIELD_BOTTOM Then  
        ball.Y = FIELD_BOTTOM - ball.Height  
        ball.Movement.Y = -ball.Movement.Y  
        Audio.PlaySoundEffect(GameSound("bounce"))  
    End If  
  
    'if the ball has hit the rhs border  
    If ball.X + ball.Width > FIELD_RIGHT Then  
        ball.X = FIELD_RIGHT - ball.Width  
        ball.Movement.X = -ball.Movement.X 'reverse the x movement  
        Audio.PlaySoundEffect(GameSound("backwall"))  
        ScoreRed = ScoreRed + 1  
  
        ShowScoreScreen()  
        StartUpGame()  
    End If  
  
    If ball.X < FIELD_LEFT Then  
        ball.X = FIELD_LEFT  
        ball.Movement.X = -ball.Movement.X  
        Audio.PlaySoundEffect(GameSound("backwall"))  
        ScorePurple = ScorePurple + 1  
  
        ShowScoreScreen()  
        StartUpGame()  
    End If  
  
    If ball.Y < FIELD_TOP Then  
        ball.Y = FIELD_TOP  
        ball.Movement.Y = -ball.Movement.Y  
        Audio.PlaySoundEffect(GameSound("bounce"))  
    End If  
  
    Graphics.MoveSprite(ball)  
End Sub
```

The new ShowScoreScreen() is as follows. Add this to the top of the code file below the variable declarations.

```
Sub ShowScoreScreen()  
    DrawGame()  
    Text.DrawText("SCORE !!!", Color.Black, GameFont("ArialLarge"), 220, 200)  
    Core.RefreshScreen()  
    Core.Sleep(2000)  
End Sub
```

Now your game should be fully functional play against a friend. Enjoy your own SwinGame!

For more tutorials and details on game development and competitions go to www.swingame.com