Skills and Topics for KidCoder: Game Programming

Our Self-Study Approach

Our courses are **self-study** and can be completed on the student's own computer, at their own pace. You can steer your student in the right direction with no prior programming knowledge. Students only need typical computer usage skills to start; we will teach them programming from the ground up!

Each course comes with student activity starters, supplementary instructional documents, a **Solution Guide**, fully coded solutions for all activities, tests and answer keys, and guidance on evaluating projects.

Most questions about how to code individual activities are easily answered by referring to the Solution Guide (with or without parental involvement). We also provide **free technical support** to assist with any aspect of the courses!

Teachers who wish to closely monitor and grade student progress for credit purposes can administer chapter tests which are provided (with answer keys). We also provide advice and guidelines for evaluating student activities.

What Skills do Students Need to Begin?

All of our courses assume the student is already familiar with using a keyboard and mouse to select and run software, navigate the menus in a typical software program, and generally interact with their computer.

Students should understand how to use the built-in operating system software (Windows Explorer or Mac Finder) to find, save and retrieve files on their computer. It may also be helpful to have some familiarity with text editors (like Notepad or TextEdit) and some experience using web browsers to find information on the Internet.

This course requires a Windows computer with CD-ROM or DVD-ROM.

KidCoder: Game Programming is a second-semester course. **Students must have successfully completed the first-semester KidCoder: Windows Programming course prior to starting KidCoder: Game Programming!**

Topics Covered In This Course

The following are some of the computer programming topics that are covered in this course. For a full list of topics and sections, please see the Table of Contents for this course.

- Game design concepts
- Drawing shapes on the screen
- Responding to keyboard clicks and mouse movements
- Displaying and animating images
- Object position, movement and acceleration
- Collision detection
- Playing music and creating sound effects
- Artificial "game" intelligence
- Saving and loading games
- Game physics
- Printing screens to the printer

KidCoder™ Series

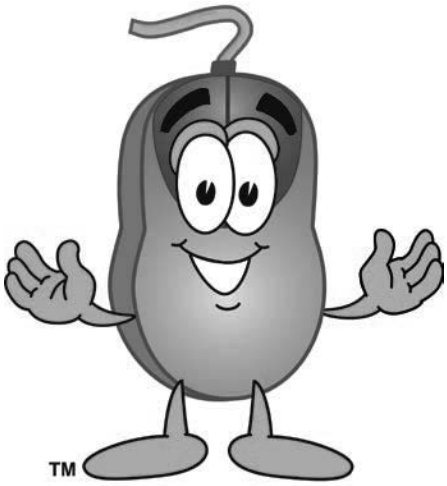
Game Programming

A hands-on introduction to the field of Game programming for elementary or middle-school students.

Student Textbook

Third Edition

KidCoder™ Series



KidCoder™: Game Programming

Student Textbook

Third Edition

Homeschool Programming, Inc.

KidCoder™: Game Programming

Third Edition

980 Birmingham Rd, Suite 501-128, Alpharetta, GA 30004

ISBN: **978-0-9887033-0-8**

Terms of Use

Disclaimer

Homeschool Programming, Inc., and their officers and shareholders, assume no liability for damage to personal computers or loss of data residing on personal computers arising due to the use or misuse of this course material. Always follow instructions provided by the manufacturer of 3rd party programs that may be included or referenced by this course.

Contact Us

You may contact Homeschool Programming, Inc. through the information and links provided on our website: http://www.HomeschoolProgramming.com. We welcome your comments and questions regarding this course or other related programming courses you would like to study!

Other Courses

Homeschool Programming, Inc. currently has two product lines for students: the *KidCoder™* series and the *TeenCoder™* series. Our *KidCoder™* series provides easy, step-by-step programming curriculum for 4th through 12th graders. These courses use readily available software products that come shipped with the operating system or are free to install in order to teach introductory programming concepts in a fun, graphical manner. Our *TeenCoder™* series provides introductory programming curriculum for high-school students. These courses are college-preparatory material designed for the student who may wish to pursue a career in Computer Science or enhance their transcript with a technical elective.

3rd Party Copyrights

This course uses Microsoft's Visual Basic 2010 Express as the programming platform. Visual Studio, Visual Studio Express, Windows, and all related products are copyright of Microsoft Corporation. Please see http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products for more details.

Instructional Videos

This course may be accompanied by optional Instructional Videos. These Flash-based videos will play directly from a DVD drive on the student's computer. Instructional Videos are supplements to the Student Textbook, covering every chapter and lesson with fun, animated re-enforcement of the main topics.

Instructional Videos are intended for students who enjoy a more audio-visual style of learning. They are not replacements for the Student Textbook which is still required to complete this course. However by watching the Instructional Videos first, students may begin each textbook chapter and lesson already having some grasp of the material to be read. Where applicable, the videos will also show "screencasts" of a real programmer demonstrating some concept or activity within the software development environment.

This Student Textbook and accompanying material are entirely sufficient to complete the course successfully. Instructional Videos are optional for students who would benefit from the alternate presentation of the material. For more information or to purchase the videos separately, please refer to the product descriptions on our website: http://www.HomeschoolProgramming.com.

Table of Contents

Before You Begin

Please read the following topics before you begin the course.

Minimum Hardware and Software Requirements

This is a hands-on programming course. You will be installing Microsoft's Visual Basic 2010 Express software on your computer. Your computer must meet the following minimum requirements in order to run Visual Basic 2010 Express:

Computer Hardware

Your computer must meet the following minimum specifications:

	Minimum
CPU	1.6GHz or faster processor
RAM	1024 MB
Display	1024 x 768 compatible video card
Hard Disk Size	3GB available space
DVD Drive	DVD-ROM drive

Operating Systems

In order to install the course software, your computer operating system must match one of the following:

Windows XP (x86) with Service Pack 3 or above
Windows Vista (x86 and x64) with Service Pack 2 or above
Windows 7 (x86 and x64)
Windows 8 (all versions except RT)

Conventions Used in This Text

This course will use certain styles (fonts, borders, etc.) to highlight text of special interest.

```
Source code will be in 11-point Consolas font, in a single box like this.
```

Variable names will be in **`12-point Consolas bold`** text. For example: **`myVariable`**.

Function and subroutine names, properties, and keywords will be in **bold face** type.

This picture highlights important concepts within a lesson.

Sidebars may contain additional information, tips, or background material.

A chapter review section is included at the end of each chapter.

Every chapter includes a "Your Turn" activity that allows you to practice the ideas you have learned in a real program.

What You Will Learn and Do In This Course

KidCoder™: Game Programming will teach you how to write your own computer games! This course is designed for students who have already completed the *KidCoder™: Windows Programming* introductory course. Computer game programming, as you might imagine, is a very large subject. There are many different games you can play, and many different programming skills are needed to make those games. Some game programs are simple, while others may take many years and many programmers to make.

This course will build on your Visual Basic programming skills learned in the first *KidCoder™: Windows Programming* course. You will learn some simple game programming techniques and apply these to several games. At the end of the course you will know enough to create your own games! Of course your games will not be as complex as the fancy games you can buy in the store today. But once you understand the basics, you can use your imagination to create many exciting things.

What You Need to Know Before Starting

You are expected to already know the basics of computer use before beginning this course. You need to know how to use the keyboard and mouse to select and run programs, use application menu systems, and work with the Windows operating system. You should understand how to store and load files on your computer and how to use Windows Explorer to walk through your file system and directory structures. You should also have some experience with using text editors and using web browsers to find helpful information on the Internet.

You should have completed and understand all *KidCoder™: Windows Programming* topics covered in that course. You should have installed the Microsoft Visual Basic 2010 Express development environment on your computer. If you are using a new computer the student files include detailed installation instructions.

Software Versions

You will be using the *Microsoft Visual Basic 2010 Express* software to complete this course. This program can be freely downloaded from Microsoft's website. Our website contains download and install instructions for this software, and your course Student Menu contains a direct link to this page. All supplemental documents are in Adobe Acrobat (PDF) format. You must have the Adobe Acrobat Reader installed to view these documents.

Course Errata

We welcome your feedback regarding any course details that are unclear or that may need correction. You can find a list of course errata for this edition on our website.

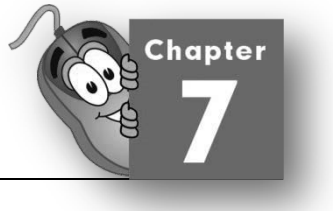
Getting Help

All courses come with a Solution Guide and fully coded solutions for all activities. Simply install the "Solution Files" from your course setup program and you will be able to refer to the solutions as needed from the "Solution Menu". If you are confused about any activity, this will allow you to see how we solved the problem!

We also offer free technical support for students and teachers. Simply fill out the help request form in the "Support" area of our website with a detailed question and we will assist you.

SAMPLE STUDENT LESSON

The following pages contain a sample student lesson from the KidCoder: Game Programming textbook.

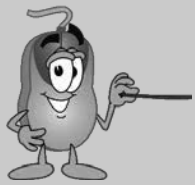


Chapter Seven: Sprites

In this chapter we will discuss one of the most important building blocks of any game: sprites. We will also begin building a new game to put our sprites to good use!

Lesson One: Sprite Concepts

What is a "sprite"? In game programming, a sprite is a graphical object on the screen that we can move and manage with our code. Sprites can be simple, fixed objects or complicated, animated, movable objects that can collide with one another. A single game can have any number of sprites. Sprites can represent your good guys, bad guys, ships, missiles, walls, and so on.



In game programming, a sprite is not a refreshing drink! A sprite can be any graphical object on the screen. A spaceship, ping-pong ball, and a rock are all possible examples of game Sprites.

Although your sprites may each look very different, you will find that writing the code behind them requires the same basic set of tasks each time. Any sprite may need to move, speed up, slow down, bounce off a wall, or display an animated image.

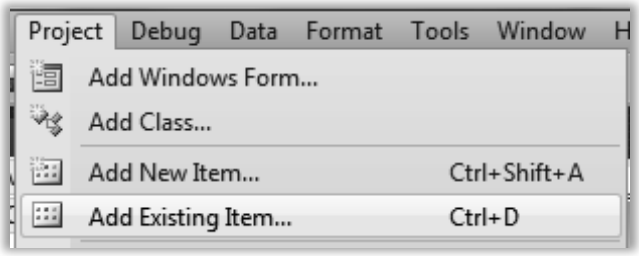
Instead of writing lots of the same code to handle each sprite individually, most game programs use a sprite library as part of their game logic. For this course we provide a sprite library for you. The sprite library defines a new **Sprite** data type you can declare as a variable. The **Sprite** contains properties to represent the object's size, position, speed, image, and other parameters. The **Sprite** also contains functions that will make the sprite move, collide, rotate, and other useful features. By using a sprite library you will be able to write programs more quickly and easily instead of re-inventing the wheel within each program!

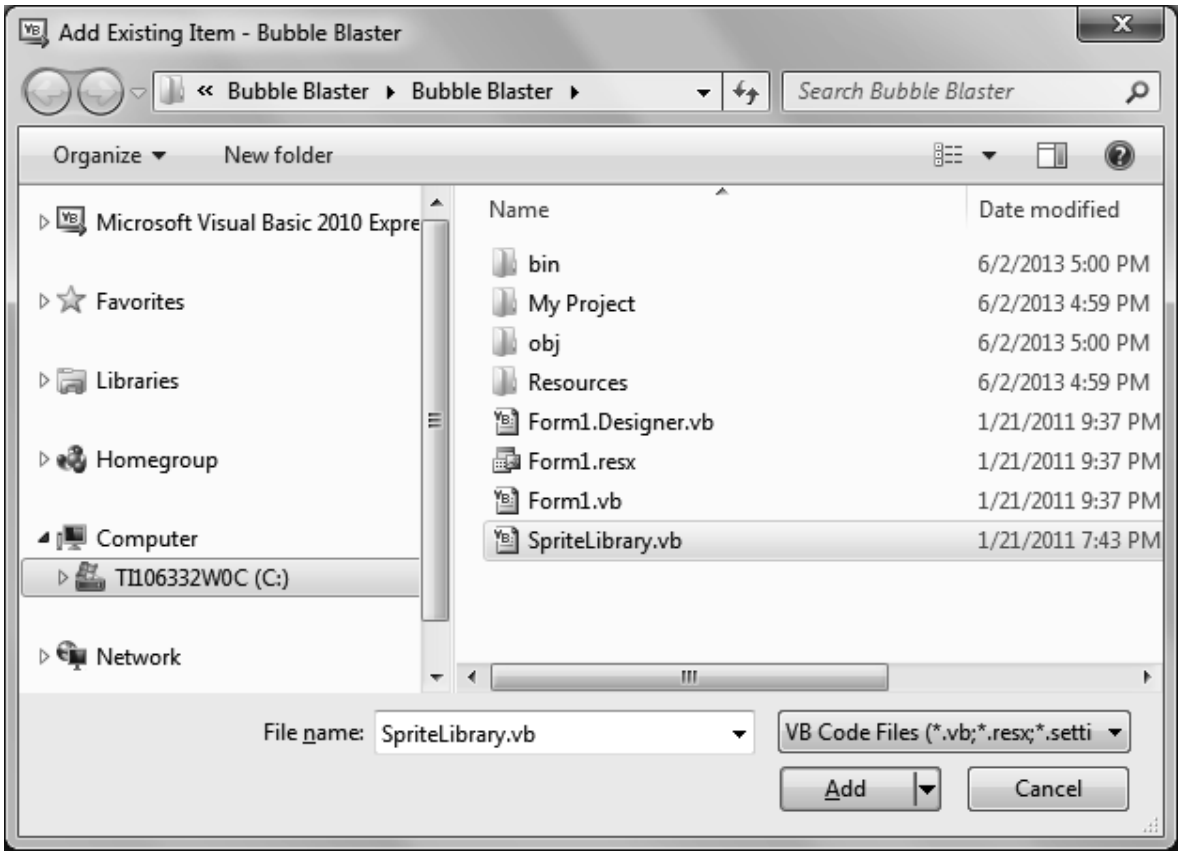
In the next two chapters we will describe several important sprite concepts and show how those concepts are supported by our **Sprite** library. Along the way you will be developing a fully functional "Bubble Blaster" game with the techniques you have learned.

The Sprite Library

Our **Sprite** library is contained in a code file called "SpriteLibrary.vb". This file will be present in each of your Activity Starter projects. During the next few chapters we will show you how this library works. Once we are done you will have a fully-featured **Sprite** library that can be used in any game program.

To use the **Sprite** library, first you need to add the "SpriteLibrary.vb" file to your project. This is done by clicking on the Project menu and then choosing "Add Existing Item". Then you just need to find the directory with the "SpriteLibrary.vb" file and add that file to the project.





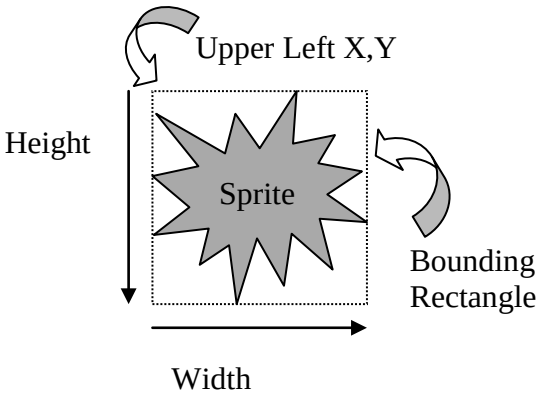
This step is already done for you in all of the Activity Starter projects, but when you write your own games you will have to do it yourself. Once you have the **Sprite** library file added to the project, you can then create as many variables as you like where the data type is **Sprite**:

```
Dim mySprite As Sprite = New Sprite()
```

Notice that you have to use the **New** keyword to create a new copy of the **Sprite** to store in the variable.

Sprite Positioning and Size

Sprites are visual objects you see on the screen. Sprites therefore must have some properties describing their position and size. The position is represented by the same X and Y coordinates we have worked with in previous chapters. Since we are representing the position of a sprite with a single pair of coordinates, we need to know which part of the sprite those coordinates represent. Sprites are usually positioned using the coordinates of the upper-left corner of the sprite. If the sprite is not a rectangle, you can mentally draw a rectangle around the outer edges of the sprite shape. This rectangle is called a "bounding rectangle". The sprite is then located using the upper-left corner of the bounding rectangle.



The **Sprite** library contains a function called **GetBoundingRectangle**(), which will return a **Rectangle** with the **Sprite's** location coordinates, width, and height.

```
Public Function GetBoundingRectangle() As Rectangle
```

Using the **GetBoundingRectangle**() function is pretty simple. For example:

```
Dim boundingRectangle As Rectangle = mySprite.GetBoundingRectangle()
```

This will provide you with a rectangle that represents the sprite's boundaries on the screen. Since this information is often useful in your game logic, the **GetBoundingRectangle**() function comes in handy!

So far you have learned about two properties of a sprite: position and size. These properties are part of the **Sprite** data type as follows:

```
Public UpperLeft As Point
Public Size As Point
```

Both properties are **Points**, which means they have an X and Y component. The **UpperLeft** Point represents the screen coordinates of the upper left corner. The **Size** Point represents the width (X) and height (Y) of the sprite.

When you want to get or set the **Sprite** size or position, just read or set the **UpperLeft** and **Size** properties with new values. In this example we set the position to (50, 100), the width to 20, and the height to 30.

```
' Set the sprite position to (50,100)
mySprite.UpperLeft.X = 50
mySprite.UpperLeft.Y = 100

' Set the sprite size to width = 20, height = 30
mySprite.Size.X = 20
mySprite.Size.Y = 30
```

Sometimes you may find it more convenient to get or set the sprite position in terms of the sprite's center instead of the upper-left corner. Our **Sprite** library has functions that will do that too!

```
Public Sub SetCenter(ByVal center As Point)
Public Function GetCenter() As Point
```

You can call those methods on your sprite objects as follows:

```
' Get the sprite's current center location
Dim myCenter = mySprite.GetCenter()

' Set the sprite's new location using the center point at (200,150)
myCenter.X = 200
myCenter.Y = 150
mySprite.SetCenter(myCenter)
```

Setting Sprite Images

Sprites are almost always represented as an image on the screen. This means that we must be able to set and display the Sprite's image in a game program. The Sprite Library contains two different methods for setting the Sprite's image: **SetImage**() and **SetImageResource**(). The method that you choose to use depends on how you will be adding the images to your game program.

If you simply want to load the image from a file on your hard disk, you can use the **SetImage**() method.

```
SetImage(filename As String)
```

To use this method, you can simply pass in the name of an image on your computer, like this:

```
mySprite.SetImage("c:\myimage.jpg")
```

The example above assumes that the "myimage.jpg" file is in the root (or main) directory of the C drive. You would have to type the correct location of any image file that you want to load. This method is similar to the **Image.FromFile**() method mentioned in the previous chapter.

If you have chosen to add your images to your project as resources, you can use the **SetImageResource**() method to set your Sprite's image.

```
SetImageResource(resourceName As System.Drawing.Image)
```

To use this method, you will pass in the name of your image resource:

```
mySprite.SetImageResource(My.Resources.myImage)
```

This will load a resource called "myImage" into your Sprite. In general, using resources is better than using filenames for your game images. Resource files will actually become part of the game program when it is compiled and built in the IDE. This means that you do not have to worry about copying images in order to play the game on different computers.

Painting Sprite Images

In order to paint a sprite's image on the screen, you can use the **Sprite**.**PaintImage**() function. This function takes two parameters: a **Graphics** object, which represents the paintable area of the game screen and a **Boolean** value which controls the image transparency. If the value is **True**, then transparent pixels will be drawn to show the underlying image. If the value is **False**, the image is not drawn with any transparency.

```
PaintImage(myGraphics As System.Drawing.Graphics, transparent As Boolean)
```

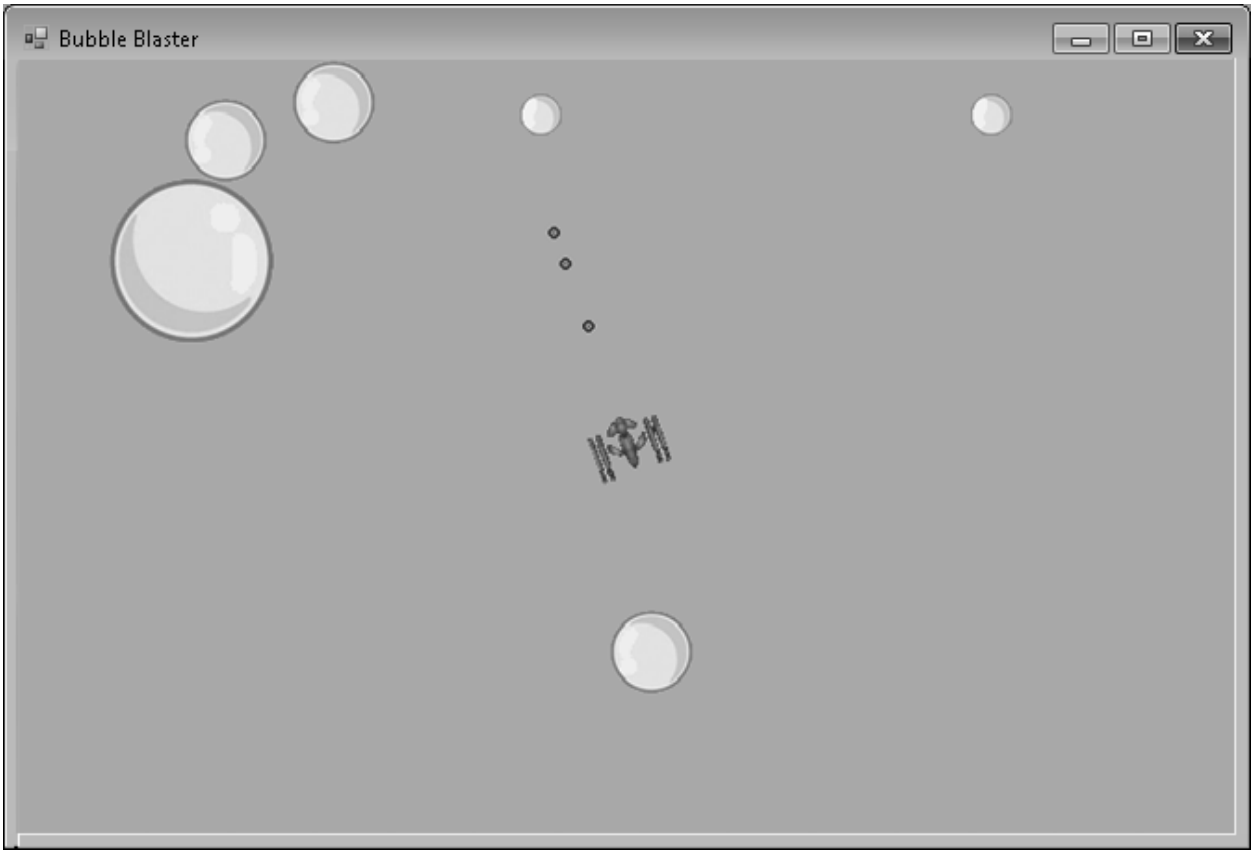
To use this method, you could use a line of code like the following one:

```
mySprite.PaintImage(myGraphics, True)
```

This would paint the Sprite's image using the current location and size properties, with transparency, to a **Graphics** object called **myGraphics**. Remember that only GIF or PNG images support transparency!

Lesson Two: Introducing Bubble Blaster

Over the next few chapters you will be writing a complete game -- Bubble Blaster! This game will show off many important sprite concepts and is fun to play! Here is an example screenshot of the finished game:



Game Play

Let's first take a look at what the game of Bubble Blaster is all about.

The Bubble Blaster game is similar to a popular arcade game called "Asteroids". "Asteroids" was originally released by Atari in 1979. The game was both simple and addictive. When the game begins, you see a spaceship in the middle of the screen. Floating around this spaceship is a series of large asteroids. These asteroids will float and spin in random directions around the screen. The spaceship shoots asteroids to break them down into smaller pieces or remove them from the game, while trying to avoid being hit.

In our Bubble Blaster game the ship is attempting to pop bubbles which are floating around the ship. The objective of Bubble Blaster is simple: you need to shoot the bubbles around you without letting any of them hit your ship. If any bubble hits your ship, your game is over. To shoot at the bubbles, you can hit the fire button, which will send out a single shot from the front of the spaceship each time the button is pressed. You can also spin your ship around in a full circle and move forward in the direction you are facing.

When you hit the large bubbles with your shots, the bubble will break into two smaller bubbles. When any of these medium-sized bubbles are hit, they will break into two even smaller bubbles. When you shoot these smaller bubbles, they will just disappear. The game is over when all the bubbles are cleared from the screen or until your ship is hit by a bubble.

Bubble Blaster Activity Starter

You will have to program a number of important features within the Bubble Blaster game as new skills are introduced in the following lessons and chapters. However, you don't have to start from scratch! We provide a Bubble Blaster Activity Starter project that takes care of some of the boring parts for you. As we complete lessons you will fill in specific functions in order to make things happen on the screen.

The starter project for this activity can be found in your "KidCoder\Game Programming\Activity Starters\Bubble Blaster" directory. Begin by copying this directory with Windows Explorer to your "My Projects" folder, then open the "My Projects\Bubble Blaster\Bubble Blaster.sln" project in your Visual Basic IDE. The activity starter project includes the following elements:

- The main screen Form named **`BlasterForm`**
- All of the images necessary for this game
- All of the variables in the game state are already created for you
- The **Sprite** library is included in the project and **Imported** at the top of the Form
- A Timer has been added to the Form
- Several functions are pre-defined for you – no need to change any of these methods!
- Some functions are empty -- these will be your responsibility!

The following constant variables are declared at the top and will set things like the ship speed and number of bubbles that will appear. We will learn to use these over the next couple of chapters as we add more features to the game!

```
    ' maximum ship speed, acceleration rate
    Const MAX_SHIP_SPEED As Double = 15.0
    Const SHIP_ACCELERATION As Double = 1.5

    ' shot speed and lifespan
    Const SHOT_SPEED As Double = 20.0
    Const SHOT_TIME_TO_LIVE As Integer = 15
```

```
    ' speed of big, medium, and small bubbles
    Const BIG_BUBBLE_SPEED As Double = 2.0
    Const MED_BUBBLE_SPEED As Double = 3.0
    Const SML_BUBBLE_SPEED As Double = 4.0

    ' number of big bubbles to start
    Const NUM_BIG_BUBBLES As Integer = 3

    ' maximum number of ship shots that can be on the screen at once
    Const MAX_SHIP_SHOTS As Integer = 5
```

The following functions are completely finished already and you won't have to change them. We'll give a short description here and you can review the code in the activity starter if curious about those parts of the game. Don't worry if some features are strange right now; you will understand them all very soon!

Function Name	Description
BlasterForm_Load()	When the program is first run the Form **Load** method will call **StartGame**()
StartGame()	Performs the steps to initialize all of the game data and start the timer
StopGame()	Performs the steps to stop the game and display a message to the user
BlasterTimer_Tick()	Calls functions to move sprites, check for collisions, and process keys
BlasterForm_Paint()	Draws the bubbles, ship, and shots on the screen
PaintShip()	Draws the ship image, correctly rotated in the direction the ship is facing
ExplodeBubble()	Called when a ship's shot hits a bubble
CreateSmallerBubble()	Called from **ExplodeBubble**() when we want to create a new smaller bubble

The rest of the functions are empty and you will complete them over the next couple of chapters as we finish each lesson. All of the code you will write for Bubble Blaster will be in the "Form1.vb" source file. Here is a brief description of what each function will do:

InitializeBubbles()	Creates and initializes the **Sprites** in the bubbles array
PaintBubbles()	Draws the bubble images on the screen
MoveBubbles()	Moves bubbles according to their current direction and speed
InitializeShip()	Create the ship **Sprite** and initialize the ship's angle, speed, and position
BlasterForm_KeyDown()	Keeps track of which keys are pressed down by the user
BlasterForm_KeyUp()	Keeps track of which keys are released by the user
ProcessKeys()	Rotates the ship, accelerates, or shoots according to the keys held down
MoveShip()	Moves the ship according to its current direction and speed
InitializeShots()	Creates and initializes the sprites in the shots array

`Shoot()`	Find an available shot in the shots array and initialize it to the current position
`MoveShots()`	Moves shots according to their current speed and direction
`PaintShots()`	Draws active shots on the screen
`CheckShipCollisions()`	Determines if the ship has hit any bubble
`CheckShotCollisions()`	Determines if any active shots have hit any bubble
`CheckIfWinner()`	Determines if the player has destroyed all of the bubbles on the screen

If you build and run the activity starter as-is without making any code changes, you won't see anything on the screen and nothing will happen when you press any keys. Never fear, you will get things going quickly!

Your Turn: Starting Bubble Blaster

The first thing you will do to improve the Bubble Blaster game is create some bubbles to show on the screen. To do this you will complete the **InitializeBubbles**() and **PaintBubbles**() functions. Open the "Bubble Blaster.sln" in your Visual Basic IDE so we can get to work!

Note that this project uses a "random number generator" to choose a random location on the screen for our bubbles. Visual Basic contains a special object which easily creates random numbers called **System.Random**. We have already created a variable of this type (called `RandomNumGen`) at the top of the Bubble Blaster code. In order to get a new random number, we will call the **Next**() method on this variable and tell the method the highest possible number that it can choose. For example, this code will choose a random number between 0 and the width of the screen.

```
RandomNumGen.Next(Me.ClientSize.Width)
```

This enables us to choose a random X coordinate for our bubble. Since **System.Random** is a new object for you, we will give you the code line to use when necessary.

InitializeBubbles

InitializeBubbles() is called from the **StartGame**() method which is already completed for you. To initialize the bubbles, you will first need to know where they are stored! The activity starter declares this array of **Sprites** at the top of the Form to hold the bubbles:

```
    Dim bubbleArray(NUM_BIG_BUBBLES - 1) As Sprite
```

This **bubbleArray** will grow as you play the game because new bubbles are created. So, to start out, we want to **ReDim** the array back to the starting size. Find the **InitializeBubbles**() function in your Bubble Blaster project. Add this line at the top of **InitializeBubbles**() to reset the array to the starting size:

```
    Private Sub InitializeBubbles()
        ReDim bubbleArray(NUM_BIG_BUBBLES - 1)
```

Now, you need to add code to create and initialize the properties of a new **Sprite** for each of the elements in the **bubbleArray**. Create a **For** loop with an index that goes from 0 to NUM_BIG_BUBBLES – 1.

```
        For i = 0 To NUM_BIG_BUBBLES - 1
```

Within the loop, for each element in the **bubbleArray** you should complete these tasks:

- Create a new bubble **Sprite** and store it in the array

```
            bubbleArray(i) = New Sprite()
```

- Use the **Sprite.SetImageResource**() function to set the image to: "My.Resources.bubble_large"

```
            bubbleArray(i).SetImageResource(My.Resources.bubble_large)
```

- Set the bubble's **UpperLeft.X** position to a random value. You can pick a random value between 0 and the screen's width like this:

```
            bubbleArray(i).UpperLeft.X = _
                  RandomNumGen.Next(Me.ClientSize.Width)
```

- Set the bubble's **UpperLeft.Y** position to another random value between 0 and the screen's height:

```
                bubbleArray(i).UpperLeft.Y = _
                      RandomNumGen.Next(0, Me.ClientSize.Height)
        End For
    End Sub
```

At this point when you run your program you still won't see anything on the screen. That will change when you implement **PaintBubbles**().

PaintBubbles

The **PaintBubbles**() function is called from the main **Paint**() method that is already done for you in the activity starter. As you might guess, this function will need to draw all of the bubbles on the screen.

To paint the bubbles, you will simply need to call the **PaintImage**() function on each **Sprite** in the **bubbleArray**. Start by creating a **For** loop that will loop over each element of the **bubbleArray**. Note that the size of the array will change over time as bubbles are shot and explode into smaller bubbles. So you don't want to loop to a constant like NUM_BIG_BUBBLES as you did during **InitializeBubbles()**. Instead loop using the current size of the array like this:

```
    Private Sub PaintBubbles(ByRef myGraphics As Graphics)
        For i = 0 To bubbleArray.Length - 1

        Next
    End Sub
```

Within the loop for each bubble, add a line to call the bubble's **PaintImage**() function, passing in the **Graphics** object and **True** to have the image drawn with transparency.

```
            bubbleArray(i).PaintImage(myGraphics, True)
```

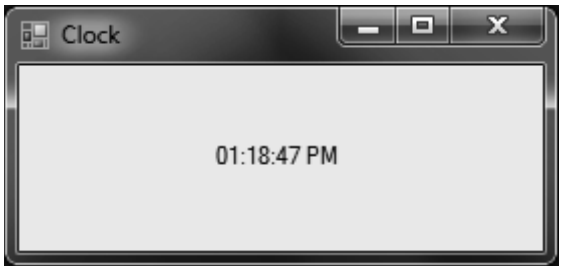
Now when you build and run your program you should see 3 big bubbles created on the screen in some random position! We'll learn how to make them move in the next lesson.

SAMPLE SOLUTION GUIDE

The following pages contain sample solution material for an activity in the KidCoder: Game Programming textbook.

Chapter Two Activity (Clock Application)

In this activity, the student is asked to use a Timer to create an automatically updated clock. The actual code for this program is only about three lines long, and the student is given most of these lines in the Activity assignment. The point of this activity is to make sure the student understands and can use Timers in a game. Timers will be used in most of the games developed in this course.



Here is the completed code for this program:

```
Public Class ClockForm

    Private Sub ClockForm_Load(ByVal sender As Object, ByVal e As System.EventArgs)
                               Handles Me.Load

        'Start the timer when the program starts
        ClockTimer.Start()
    End Sub

    Private Sub ClockTimer_Tick(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles ClockTimer.Tick
        'Create a variable that will hold the computer's current time
        Dim currentTime As DateTime = DateTime.Now

        'Format the time value so that it will show "hours: minutes: seconds"
        'and then either "AM" or "PM"
        TimeLabel.Text = currentTime.ToString("hh:mm:ss tt")
    End Sub
End Class
```

The completed project for this activity is located in the "Your Turn Solutions\Clock" folder underneath the Solution Files installation directory.