**Making a Basic Platformer**

**Introduction**

This tutorial will go over some basic mechanics that you can use to make a platformer in Visual Basic. This is not the only, or necessarily the best way to implement mechanics for a 2D platformer. Making subtle changes to the code can have large effects on the mechanics for the game. While you go through this tutorial, I will make some suggestions at various points for mechanics that you can experiment with by modifying the code. Interesting or unique mechanics can make a game much more fun.

In order to successfully complete this tutorial, I would recommend that you are familiar with the following topics:

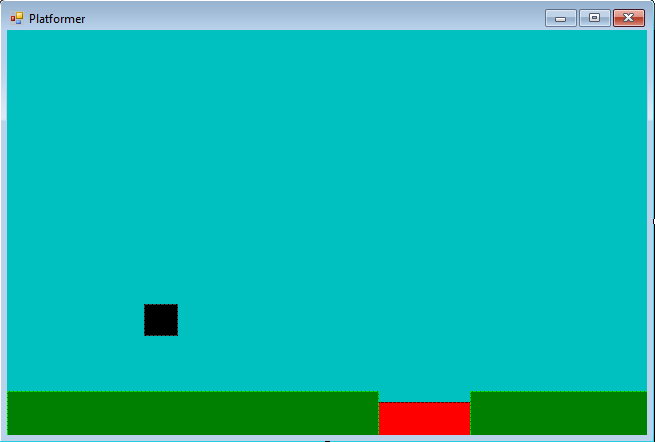
* Loops
* Timers
* Moving **PictureBoxes** with both the **Location** property (using **Points**), and the **Top** and **Left** properties (includes an understanding the coordinate system of a **Form**)
* Key Events: **KeyUp** and **KeyDown** specifically
* Lists

**Part 1 – Setup Our Basic Form**

Create a new project. Change the **Name** property of your **Form** to frmPlatformer and set its **BackColor** property to Blue. Add four **PictureBoxes** and give them the following names and properties:

|  |  |
| --- | --- |
| **Name**: imgPlayer  **BackColor**: Black | **Name**: imgLava1  **BackColor**: Red |
| **Name**: imgGround1  **BackColor**: Green | **Name**: imgGround2  **BackColor**: Green |

Once you have made them, lay them out on the **Form** so they look similar to this:



imgPlayer

imgGround1, imgGround2

imgLava1

**Part 2 - Gravity**

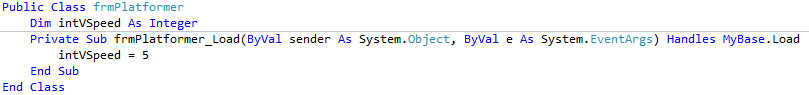
The first thing we are going to add to our program is gravity. We will use a **Timer** that will attempt to pull our player down at all times, except when we are jumping.

Add a **Timer** to your **Form** and change its **Name** property to tmrGravity. Change the following properties to the specified values:

|  |  |
| --- | --- |
| **Enabled:** True | **Interval**: 25 |

This **Timer** will act much like gravity here on earth does, and will attempt pull our player down when they are not jumping.

We will make an **Integer** variable called intVSpeed to store how fast we want our player to fall (and jump later on). Assign it a value of 5 in the **Form Load** event.



*You should experiment with values of the* ***Interval*** *properties of your gravity* ***Timer*** *(*tmrGravity*), and the size of your gravity variable (*intVSpeed*) until you are satisfied with how it works.*

We are going to add the following line of code to the **Tick** event of **tmrGravity** so that each time the **Tick** event code runs, our player moves downwards:

imgPlayer.Top += intVSpeed

*Run your program. Notice that the player falls off the bottom of the screen.*

**Part 3 – The Ground**

In order to keep our player from falling off the bottom of the screen, we will need to add some collision detection. We want to detect when imgPlayer collides with any of the **PictureBoxes** we are using for the ground (and any platforms that we add later on).

Each time tmrGravity ticks, we will need to see if imgPlayer collides with the ground, or any other obstacle as a result of falling. Eventually we will check to see if we fall into lava as well.

In order to save time repeating similar code over and over, we are going to store our **PictureBoxes** in Lists. This will save us coding by allowing us to use loops to do all of our collision checks.

Use the following code to declare a **List** called platforms, another **List** called lava and a **Boolean** variable called bolOnGround in the same place you declared intVSpeed (at the top of your program):

*As we add more platforms or lava, we will add them to our lists in the same way.*

Dim platforms As New List(Of PictureBox)

Dim lavas As New List(Of PictureBox)

Dim bolOnGround As Boolean

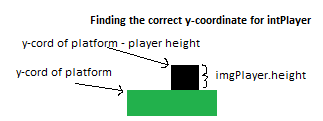
We now need to add the **PictureBoxes** we created

to these lists. Add this code to your **FormLoad** event:

platforms.Add(imgGround1)

platforms.Add(imgGround2)

lavas.Add(imgLava1)

Now that we have our platforms stored in **Lists**, we can use a **For…Each** loop to see if imgPlayer will land on any of them in our gravity **Timer**. Add the following code to the tmrGravity tick event:

If imgPlayer is intersecting/overlapping with the platform, we want to change the *y-coordinate* of its **Location** property to be just above it, **not** overlapping it.  *Recall that y-coordinates INCREASE as you go down!*

This will loop through our **List** of platforms. We can use the variable platform to reference each one.

Moves imgPlayer down

Determines if imgPlayer is intersecting (overlapping) with a platform.

imgPlayer.Top += intVSpeed

For Each platform In platforms

If platform.Bounds.IntersectsWith(imgPlayer.Bounds) Then

imgPlayer.Location = New Point(imgPlayer.Location.X, platform.Location.Y - imgPlayer.Height)

End If

Next

**Important Information:** The **Bounds** property of a **PictureBox** is stored as type **Rectangle**. This is like a variable (that is called a C**lass**) that stores 4 pieces of information. The *(x, y)* coordinate of the upper left corner of a rectangle, its *width* and *height.* This **Rectangle** class has some useful built-in functions for us to use.

The function IntersectsWith will allow us to determine whether two **Rectangles** are intersecting/overlapping. It takes the format: rectangle1.IntersectsWith(rectangle2) and gives us a **True** if they intersect or a **False** otherwise.

*Run your program.*

You may notice an occasional flicker of your imgPlayer. This will get worse as we add more **Timers** to our program. The reason for this flicker is that we are first moving imgPlayer, then looking for an overlap, and then moving imgPlayer again right above our platform if there is an overlap detected. This causes us to draw our player in places we don’t want it to be and then re-drawing it again in the correct place. We can fix unnecessary drawing by changing our gravity **Timer** to the code below.

bolOnGround = False

For Each platform In platforms

If New Rectangle(imgPlayer.Location.X, imgPlayer.Location.Y + intVSpeed, imgPlayer.Width, imgPlayer.Height).IntersectsWith(platform.Bounds) Then

imgPlayer.Location = New Point(imgPlayer.Location.X, platform.Location.Y - imgPlayer.Height)

bolOnGround = True

End If

Next

If Not bolOnGround Then

imgPlayer.Top += intVSpeed

End If

The big change here is we are going to create a *virtual* **Rectangle** where we plan on moving imgPlayer to. This allows us to check for a collision with this virtual **Rectangle** BEFORE we actually move imgPlayer so we can save our program from drawing it overlapping with our platform for an instant before re-drawing it just above the platform.

We are on the ground.

We will only move **imgPlayer** down if we know it will not collide with any platforms.

**Part 4 – Jumping**

We are now ready allow our player to jump. Add a **Timer** to your **Form** and set the following properties:

|  |  |  |
| --- | --- | --- |
| **Name:** tmrJump | **Enabled:** False | **Interval**: 20 |

This **Timer** will subtract our vertical speed (intVSpeed) from the y coordinate of imgPlayer for a set certain number of ticks.

Create an **Integer** variable called intJumpCounter at the top of your program. We will use this variable to keep track (count) of our how many times tmrJump has ticked, which will allow us to control how high imgPlayer can jump.

Dim intJumpCounter As Integer

When tmrJump ticks we will increase intJumpCounter so that we know how many times we have moved imgPlayer up. The larger this number, the higher our jump will be.

Add the following code to tmrJump.

intJumpCounter += 1

If intJumpCounter >= 15 Then

tmrJump.Enabled = False

tmrGravity.Enabled = True

Else

imgPlayer.Top -= intVSpeed

End If

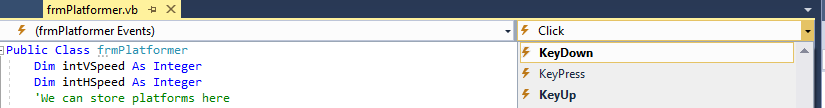
Once **intJumpCounter** reaches 15, we turn off this timer and re-enable gravity. Otherwise we move **intPlayer** up. You can experiment with different values to determine the maximum height of your jump.

You can make this a **variable** so that your maximum jump height can change while during your game.

*Run your program. Notice nothing happens! Why??*

**Part 4.1 – Keyboard Events**

In order to make our player jump, we need to detect a keyboard event so we can turn on our jump **Timer**. We are going to listen for the up arrow. You can make jump any key you wish.

In the **Code Editor** window, select **frmPlatformerEvents** and then select the **KeyDown** event.

Add the following code to the **KeyDown** event and *run your program*.

If e.KeyCode = Keys.Up Then

tmrGravity.Enabled = False

tmrJump.Enabled = True

intJumpCounter = 0

End If

We can hold down the jump key (up arrow) which keeps resetting

intJumpCounter causing us to keep jumping. We don’t want

to enable our jump and reset our counter unless we are on the ground.

We can fix this by using our onGround **Boolean**. We only start a jump when it is **True,** and then set it to **False** when we start jumping. Put the code in the **KeyDown** event that we used to stop the gravity **Timer**, start the jump **Timer** and reset our jump counter into an **If** statement as shown here.

If e.KeyCode = Keys.Up Then

If onGround Then

onGround = False

tmrGravity.Enabled = False

tmrJump.Enabled = True

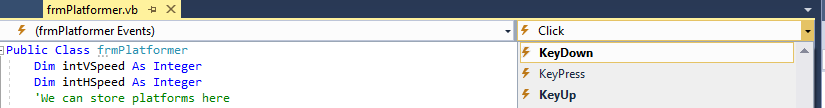
intJumpCounter = 0

End If

End If

You also may have noticed that the jump height is always the same. If we want to stop our jump early by releasing the jump button, we can do that with a **KeyUp** event.

In the Code Editor window select frmPlatformer **Events** and then select the **KeyUp** event.



In the **KeyUp** event, disable tmrJump, and enable tmrGravity as shown here.

If e.KeyCode = Keys.Up Then

tmrGravity.Enabled = True

tmrJump.Enabled = False

End If

*Run your program to test it.*

**Part 5 – Moving Left and Right**

Let’s make imgPlayer move left and right with the arrow keys. To make it smooth, we are going to put our movement code into timers, and turn them on and off using **KeyDown** and **KeyUp** events.

Add a two **Timer** controls to your **Form** and set their properties as follows:

|  |  |  |
| --- | --- | --- |
| **Name:** tmrLeft | **Enabled:** False | **Interval**: 25 |
| **Name:** tmrRight | **Enabled:** False | **Interval**: 25 |

Add an **Integer** variable to the top of your program called intHSpeed. We will use this variable to change the horizontal position of imgPlayer.

Dim intHSpeed As Integer

In the **Form\_Load** event, set the value of intHSpeed to 3. You can experiment with the value of intHSpeed and the **Interval** property of your **Timers** to change the speed and smoothness of your movement.

Add the following code to the specified **Timers** so that when activated, imgPlayer will move.

tmrRight: imgPlayer.Left += intHSpeed

This will **increase** the distance of imgPlayer from the left side of the game **Form**, thus moving it to the right.

*Note:* imgPlayer.Right *is read only, so we*

*must use* imgPlayer.Left *for movement.*

This will **decrease** the distance of imgPlayer from the left side of the game **Form**, thus moving it to the left.

tmrLeft: imgPlayer.Left -= intHSpeed

We now need to add to our **KeyUp** and **KeyDown** events so that we can enable tmrLeft and tmrRight when the appropriate arrow key is pressed, and disable them when that key is released.

**KeyDown**: **KeyUp**:

If e.KeyCode = Keys.Up Then

If bolOnGround Then

bolOnGround = False

tmrGravity.Enabled = False

tmrJump.Enabled = True

intJumpCounter = 0

End If

ElseIf e.KeyCode = Keys.Right Then

tmrRight.Enabled = True

ElseIf e.KeyCode = Keys.Left Then

tmrLeft.Enabled = True

End If

If e.KeyCode = Keys.Up Then

tmrJump.Enabled = False

tmrGravity.Enabled = True

ElseIf e.KeyCode = Keys.Right Then

tmrRight.Enabled = False

ElseIf e.KeyCode = Keys.Left Then

tmrLeft.Enabled = False

End If

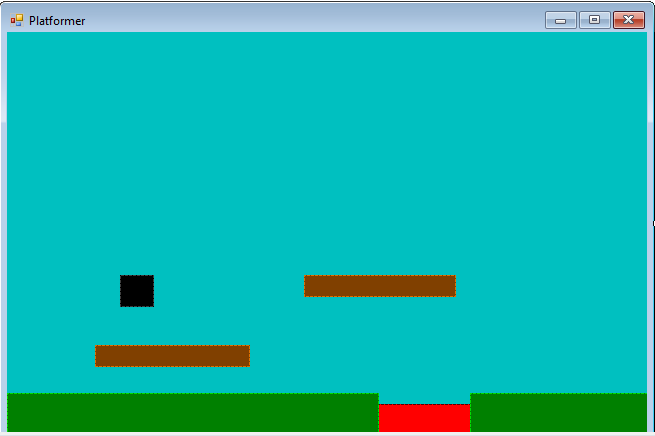
*Run your program.*

**Part 6 – Adding Platforms**

We are going to add two **PictureBoxes** to our **Form** and give them the following properties:

|  |  |
| --- | --- |
| **Name**: imgPlatform1  **BackColor**: Brown | **Name**: imgPlatform2  **BackColor**: Brown |

Lay them out so they look like this, and then run your program:



Notice that player falls right through them. All we have to do is add them to our **List** platforms and tmrGravity will automatically check for collisions with them as well. Ahh, the beauty of loops.

In our **Form\_Load** event, we will also need to add these two **PictureBoxes** to our **List** using the following lines of code:

platforms.Add(imgPlatform1)

platforms.Add(imgPlatform2)

*Run your program.*

Our player moves around pretty well at this point, and the jumping and gravity work. You may have noticed however that when we jump into the bottom of a platform or hit it from the sides, we are transported on top of it. This is because tmrGravity is running, and doesn’t distinguish whether we collide with a platform from the side, top or bottom. If we make sure that our player doesn’t collide with the sides or bottom of a platform, we won’t have this issue.

Try to solve at least one of these problems on your own. If you can’t, you may look at the solutions below.

The easiest way to solve this problem is to add checks to tmrJump, tmrLeft and tmrRight that check for collisions in the same way that we did in tmrGravity. We just need to adjust them for the direction each **Timer** is moving us, and what position we should place imgPlayer if that move causes a collision.

**Part 7 - Fixing Collisions**

Just like with tmrGravity, we are going to create a virtual **Rectangle** in the position where imgPlayer will be after we move it. If there will be a collision, we will place it right up against the **bottom** of the platform. This will also prevent tmrGravity from detecting a collision and putting imgPlayer on top of the platform.

**tmrJump**

We can use this **Boolean** to check whether we hit the bottom of a platform so we can turn off our tmrJump.

We end our jump if we reach the maximum height, **OR** we have hit the bottom of a platform.

Dim bolHitsPlatform As Boolean

For Each platform In platforms

If New Rectangle(imgPlayer.Location.X, imgPlayer.Location.Y - intVSpeed, imgPlayer.Width, imgPlayer.Height).IntersectsWith(platform.Bounds) Then

imgPlayer.Location = New Point(imgPlayer.Location.X, platform.Location.Y + platform.Height)

bolHitsPlatform = True

End If

Next

intJumpCounter += 1

If intJumpCounter >= 15 Or bolHitsPlatform Then

tmrJump.Enabled = False

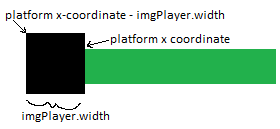
tmrGravity.Enabled = True

Else

imgPlayer.Top -= intVSpeed

End If

**tmrRight**

Try this on your own first before copying code. Use the same **For...Each** loop to iterate through all platforms. Create a virtual **Rectangle** in the location imgPlayer will be after moving to the right. If you detect a collision move imgPlayer so that its right side is against the left side of the platform. This will differ from our gravity and jump checks in that we will be changing the **x-coordinates** instead of the **y-coordinates**, and will need to use the **Width** properties instead of the **Height** properties.

Dim bolHitPlatform As Boolean

For Each platform In platforms

If New Rectangle(imgPlayer.Location.X + intHSpeed, imgPlayer.Location.Y, imgPlayer.Width, imgPlayer.Height).IntersectsWith(platform.Bounds) Then

imgPlayer.Location = New Point(platform.Location.X - imgPlayer.Width, imgPlayer.Location.Y)

bolHitPlatform = True

End If

Next

If Not bolHitPlatform Then

imgPlayer.Left += intHSpeed

End If

**tmrLeft**

Try this on your own, using the same idea as tmrRight. You will need to use a similar idea to place imgPlayer against the right side of the platform if you detect a collision with your virtual **Rectangle**. *Draw a sketch to help!*

Dim bolHitPlatform As Boolean

For Each platform In platforms

If New Rectangle(imgPlayer.Location.X - intHSpeed, imgPlayer.Location.Y, imgPlayer.Width, imgPlayer.Height).IntersectsWith(platform.Bounds) Then

imgPlayer.Location = New Point(platform.Location.X + platform.Width, imgPlayer.Location.Y)

bolHitPlatform = True

End If

Next

If Not bolHitPlatform Then

imgPlayer.Left -= intHSpeed

End If

**Part 8 – Colliding with Lava**

We are going to have our player ‘die’ when it touches any platform in our **List** of lava. Since our level design makes it only possible to hit this on the way down, we only need to check it when tmrGravity moves our player. To keep our code cleaner, we are going to create a **Sub Procedure** that scans through the lava obstacles for us. You can also use this idea to detect collisions with other types of items that you may wish to add to your program later on.

What kinds of things do we want to happen when we hit the lava? For our example, we will stop all of our **Timers**, pop up a **MessageBox**, and then close our program. You should customize this to suit the needs of your game.

Add the following code to your program:

Sub deathDetectFall()

For each lava In lavas

If imgPlayer.Bounds.IntersectsWith(lava.Bounds) Then

tmrGravity.Enabled = False

tmrLeft.Enabled = False

tmrRight.Enabled = False

MsgBox("You are dead")

Me.Close()

End If

Next

End Sub

We can call this **Sub Procedure** at the end of tmrGravity by adding the following line of code at the end of it:

deathDetectFall()

This is the end of the detailed tutorial. Below, I will suggest some mechanics that you may want to experiment with, and provide some basic ideas of how you may do some of them. At this point, you can either attempt them on your own, or do some research online to help. Good luck and have fun.

**Part 9 – Things to Try**

This is a very basic framework for a platformer. Here are some things you may want to try:

**Staying on the Form**

You already have experience keeping your player on the Form from previous tutorials. It is no different here. You could even add wrap around so your player leaves the left side of the form and returns on the right.

**Health and Lives**

Keep track of health or lives. Reset your player’s location when a life is lost.

**Add Enemies**

You may want to make a List of enemies so that you can use a loop within a single **Timer** to move them all and check for collisions.

**Add Obstacles and Items**

Use the same idea as above. If obstacles are similar, and will exhibit similar collision behaviour (such as disappearing, or removing health/lives), group them together in a List.

**Powerups and Other Effects**

You can change the speed and jump height easily by setting Timer Intervals, and speed values from a variety of events. For example, a collision with a certain type of item may allow a player to jump higher for a certain amount of time, or you could change the size of your player.

**Ducking**

Allow your player to duck while holding a certain key down by changing the Height property. A KeyUp event for that same key could return the player to its original size.

**Hanging**

Your player could hang on the bottom of a platform if you hit it. This could be done by turning off gravity when hitting the bottom of a platform. You must decide how and when you want to re-enable it.

**Falling Death from Heights**

When your gravity timer ticks, keep a count of how far you fall. Use this count to decide what happens when you hit the ground (death, loss of health etc.). Make sure you reset this falling counter each time you land.

**Visuals**

Put images in your PictureBoxes instead of just changing the BackColor. You can have pictures change when certain events occur, such as jumping, or landing on a platform if you wish. To help with performance, resize the image files to the exact size you need before adding to them to your project resources in order to save your game from having to scale the images over and over.

**Breaking Platforms**

Have your player smash a platform if they hit it from the bottom.

**Transporter**

Have some type of portal that transports your player to another spot on the map.