## VBugs

## Chapter 8

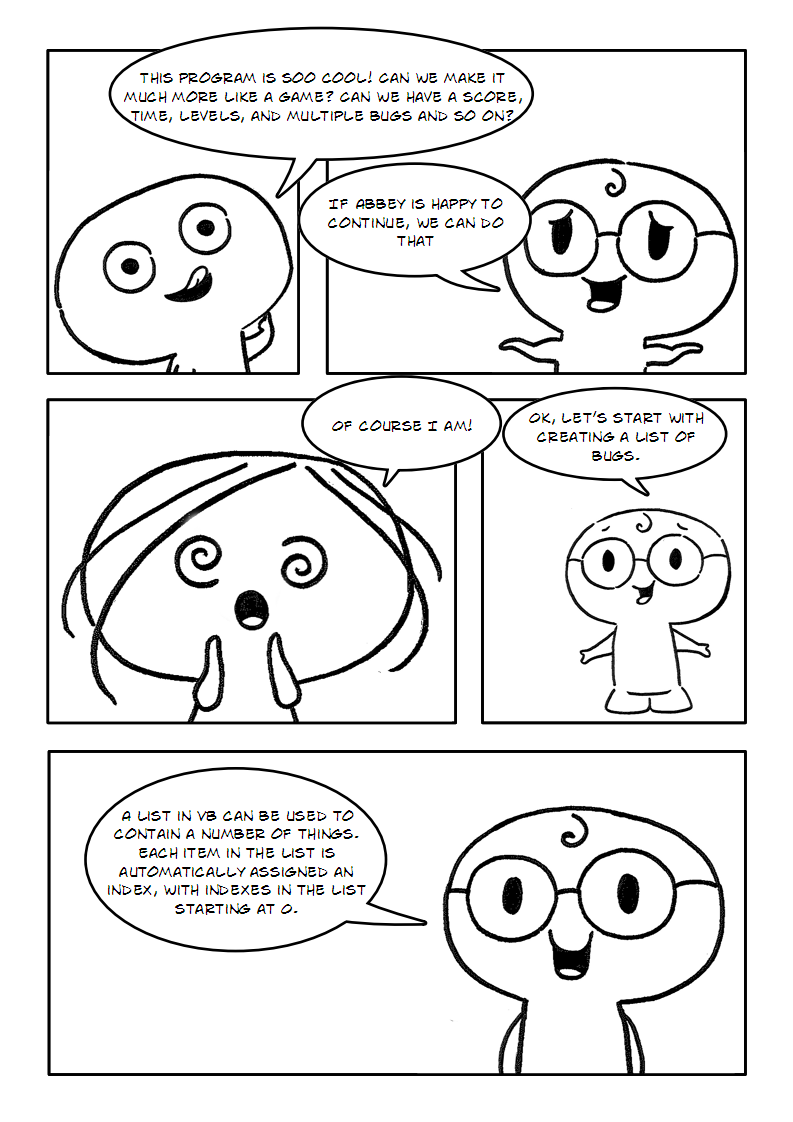
###### Level and Score

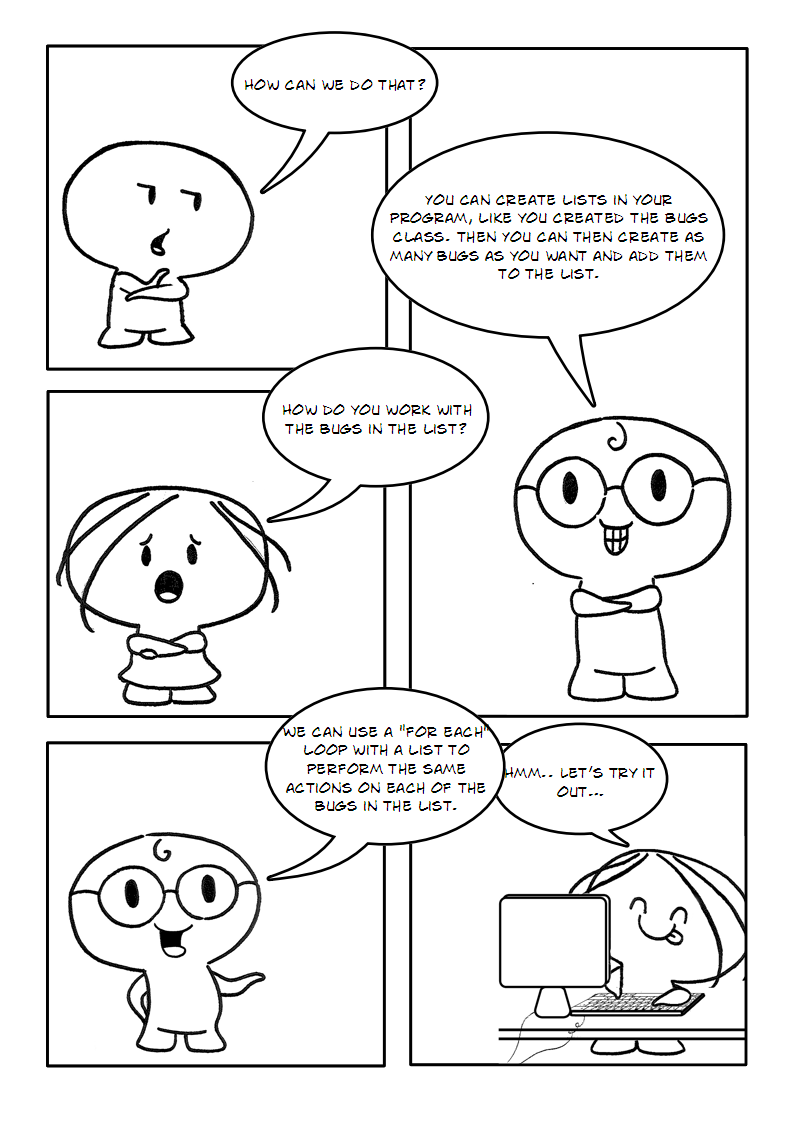


# Summary:

This chapter will help you create different levels for your project and show you how to make your program look like a game. To complete this chapter you are required to use the solution from the previous one. You will also need new fonts, which have been provided to you.

This chapter is the last one in a relation to “Bug Squash” game. Next series of chapters will be based on knowledge collected from Chapter 1 to 8 and will develop a new game – “Buggy Loves Food”.

****

****

## Part 1

### Our Game

The way our game will work is that there will be timed levels. Each bug the player kills they get a score for. The quicker they kill the bug the higher the score they get. If they kill a bug and the timer read 27 then 27 gets added to their score. The player has to kill all the bugs before the time runs out, and if they don’t they get a score penalty, the higher the level the higher the score penalty (see Figure 1). Level 1 you start with 2 bugs, and every level after that it doubles. The game never ends it just keeps getting harder. In between levels and when you lose splash screens will come up telling you what is going on.



Figure 1

### Creating Lists

In order to create our range of bugs on the screen we will use lists. In programming a **list** is a collection of similar things. You can perform a number of specific actions with things in a list. In our case we want to create a list of Bugs. To do this we need to declare a field which will create our list of bug objects.

Following Module GameLogic (in GameLogic.vb) we will declare our list of bugs the like this:

|  |
| --- |
| Private listBugs As List(Of Bug) |

Then we will assign a value to the list. A List is a class that has features that you can use to manage a number of objects. You will remember that for our previous class (Bugs) we had to use “New” to **construct** the object based on the class. Here we have to do the same for the list. To create a new list object we use the following after the Randomize() sub call inside the Main() method:

|  |
| --- |
| listBugs = New List(Of Bug ) |

To perform any action to our list such as drawing and updating each bug in the list, we can use a “For Each” loop. The logic of the “For Each” loop is shown in Figure 2 below:

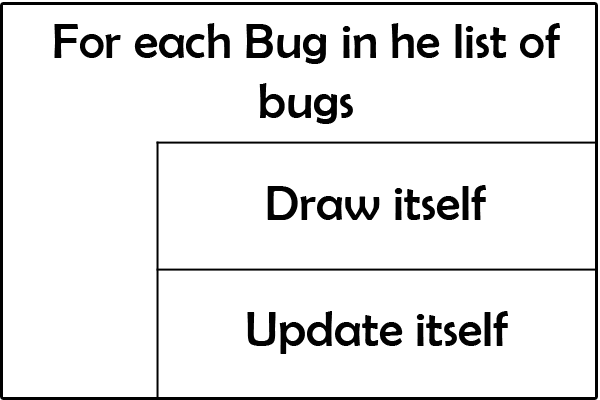


Figure 2

To implement this logic of telling each bug in the list to draw and update itself, we can use the following code inside the Game Loop after Graphics.ClearScreen(Color.White):

|  |
| --- |
| For Each Bug As Bug In listBugs  Bug.Draw()  Bug.Update()  Next |

Now that we are using a list of bugs we can **delete** the following code that was for our previous single bug:

|  |
| --- |
| Dim myBug As Bug  myBug = New Bug |

*And..*

|  |
| --- |
| myBug.Draw()  myBug.Update() |

*cha 2 - worksheet.pngQuestion 1: Define the term “list”.*

### Cleaning up

Remember each level we will double the number of bugs. And because we are using multiple sprites now (our bugs), we need to free each of them at the end of the levels and the end of the game. This frees up our memory so it can be used to store the bugs for the next. The following code when added to Bug class will get rid of all the bugs from the memory:

|  |
| --- |
| Public Sub CleanUp()  DeadSprite.Dispose()  AliveSprite.Dispose()  End Sub |

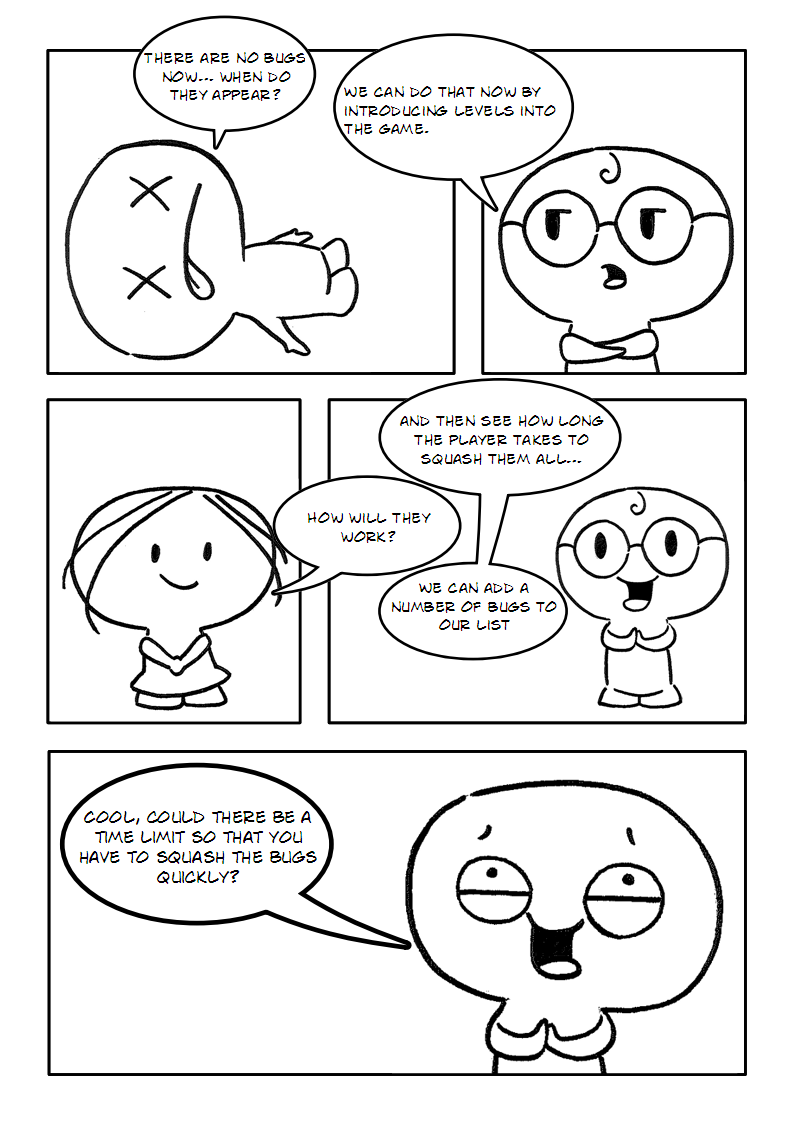
We can call this function from GameLogic.vb whenever we need to free our sprites.

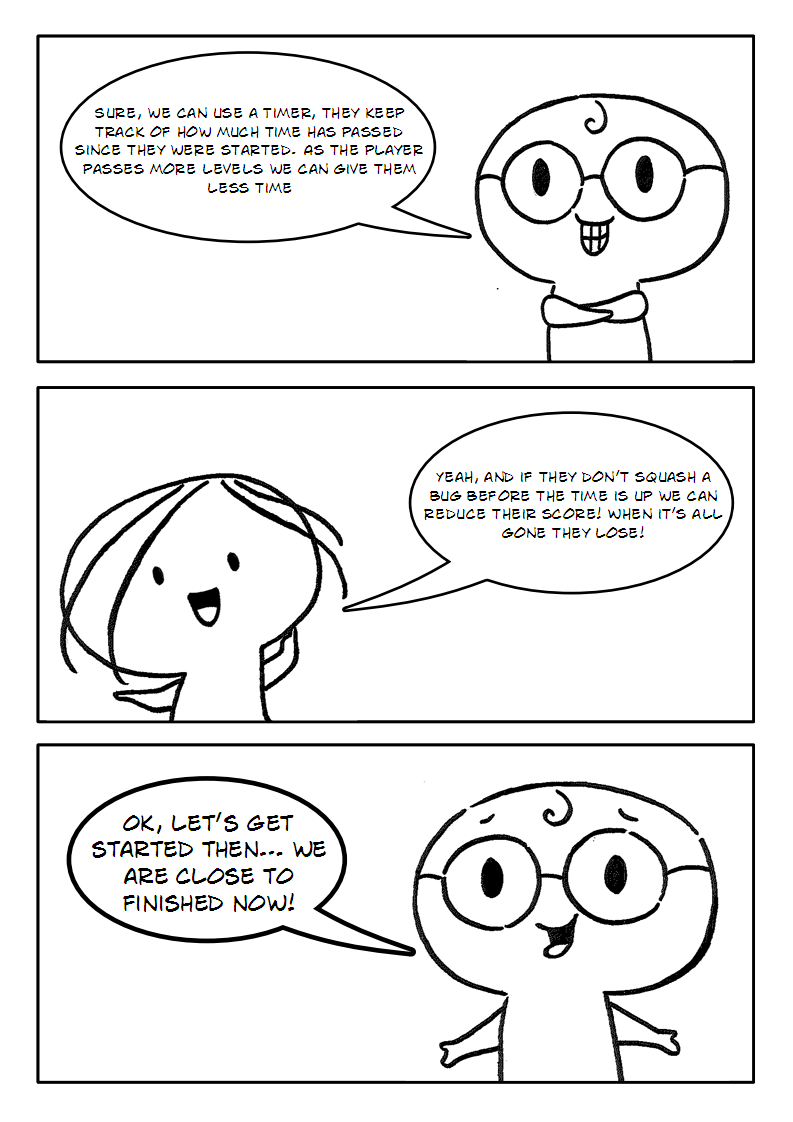
Exercise 1: *Creating a list*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Create a list of bugs in your program. (Use the information described previously in order to achieve this).
2. Write the code that will allow each bug in your program to draw and update itself.
3. Create the function that will free the sprites. (Delete the lines of code we don’t need any more)

cha 2 - worksheet.png*Question 2: Why do we have to free our sprites after we have finished with them?*

****

****

## Part 2

### New Variables

Our list of bugs is empty for now. We need to fill it up; we also need to introduce a level variable, a timer, a score and tell the program at which point the level will end. To do so, the first thing we will need is six variables. We will store all these values in variables GameLogic.vb:

|  |
| --- |
| Private level As Integer  Private gameTimer As Timer  Private endLevelAt As Integer  Public bugsKilled As Integer  Public score As Integer  Public time As Integer |

* The variable “level” will be used to store the current game level number.
* gameTimer is a clock that will count time as a game goes.
* endLevelAt is a number indicating the length of the level in time. Each level will get consequently shorter.
* bugsKilled is a number which will represents a how many bugs the user has killed.
* score is a number which represents the player’s current score in your program. Each bug you kill, you get a score for, the quicker you kill the bug the higher the score.
* time is a variable we use to store the display time as opposed to the actual time, display time is divided by 100 so it does not look too confusing.

Notice that the last three variables (bugsKilled score and time) are “Public” variables. This is because both of these variable need to be accessed from within the “bugs” class in the CheckIfClicked() method.

### Level Set Up

Now we need to have a procedure that will set up each level for us in GameLogic.vb. This procedure basically:

* Stops then restarts the timer for each level.
* Then it will frees each sprite created in previous levels.
* It then fills up the list with a number of bugs (the number is doubled each level starting with 2 bugs in Level 1).
* Then it tells each level at which point the current level stops and when the next one starts using that logic that 1000 is 1 second (time is measured in milliseconds). Each level will end after 10,000 (10 seconds) minus 500 (half a second) times the level. Therefore each level is a bit shorter that the last (Level 1 - 9500; Level 2 - 9000 etc.).
* It also stops a level getting any shorter than 500 milliseconds.

The code for this procedure is shown below:

|  |
| --- |
| Public Sub LevelSetUp()  Core.StopTimer(gameTimer)  For Each Bug As Bug In listBugs  Bug.CleanUp()  Next  listBugs.Clear()  For i As Integer = 1 To level \* 2  listBugs.Add(New Bug)  Next  endLevelAt = 10000 - 500 \* (level - 1)  If endLevelAt < 500 Then  endLevelAt = 500  End If  Core.StartTimer(gameTimer)  End Sub |

*Exercise 1: Variables and Level Set Up*

*cha 2 - worksheet.png*Make the following changes in your program and write your solutions to the worksheet:

1. *Declare the five new variables we need for our game.*
2. *Create the* LevelSetUp() *sub in your code then label it on your worksheet.*

*cha 2 - worksheet.pngQuestion 1: Using the logic above how long (in seconds) will level 6 be?*

*cha 2 - worksheet.pngQuestion 2: Using the logic above how may bugs will be on the screen in level 6?*

*cha 2 - worksheet.pngQuestion 3: Using the logic above how long (in seconds) will level 20 be? Show your working.*

### Initializing the Game Values

In the main method we need to set up the starting point for the game. To do so, we need to first create the game timer. We will put the timer in the Main() method right after listBugs = New. List(Of Bug). We then need to tell our program to start from level 1 with score = 0, then call the LevelSetUp() sub to set up the first level. The code for this is shown below:

|  |
| --- |
| gameTimer = Core.CreateTimer()  level = 1  score = 0  LevelSetUp() |

### End of Level

A level is over if either the time has run out or all the bugs are dead. So we need to create a check for both of these things. Let’s start with the check to see if all the bugs are dead. To do this we will create a function called EndOfLevel() in GameLogcic.vb which will check whether all of bugs in the listBugs are dead. If they are dead it will return “True” and if any of them are alive it will return “False”. The code for this function is shown below:

|  |
| --- |
| Public Function EndOfLevel() As Boolean  For Each myBug As Bug In listBugs  If myBug.IsAlive Then  Return False  End If  Next  Return True  End Function |

Now in the Game Loop we need to tell our game what to do if the the end of the level has been reached (when all the bugs have been killed). We need to tell it to increase the level by one and call the LevelSetUp() sub to set up the next level as below:

|  |
| --- |
| If EndOfLevel() = True Then  level = level + 1  LevelSetUp()  End If |

*Exercise 2: Ending the Level and Initializing the Game Values*

*cha 2 - worksheet.png*Make the following changes in your program and write your solutions to the worksheet:

1. *Create the code for ending the level and initializing the game values.*

*cha 2 - worksheet.pngQuestion 4: What are the two conditions that will cause the level to end?*

### Increasing the Score

To increase their score a player must kill bugs, the quicker they kill bugs the higher the score so what we will do is every time a user kills a bug we will add the number on the timer to the current score. That way if kill a bug and 80 is on the timer 80 gets added to your score. Remember the test for checking if the bug has been clicked in the the Bugs class. You will need to add the following code to right place in that sub:

|  |
| --- |
| score = score + time |

Underneath this line you also need to add one to the total of the bugsKilled variable.

### Time’s Up

Now remember when time runs out the player gets a score penalty and the timer starts over. So lets say the score penalty is 100 times the currunt level that way the higher the level the bigger the penalty making the game progressively harder. We will need to declare a new variable “scorePenalty” at the top of GameLogic.vb with the other decalrations as type “Integer”. Then in the code below you will see we then figure out the score penalty, then minus that amount from the score. We then need to stop and start the timer over when the timer reaches zero. This code will of course have to go in the game loop (below the update routine):

|  |
| --- |
| If time < 0 Then  scorePenalty = 100 \* level  score = score - scorePenalty  Core.StopTimer(gameTimer)  Core.StartTimer(gameTimer)  End If |

Exercise 3: *Setting up the score and score penalty*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Implement scoring and score penalties in your game. Debug your game after you do.

*cha 2 - worksheet.pngQuestion 5: A player has a score of 1050, they are on level 4 of the game and the time runs out before they kill all the bugs. What will there score be after the score penalty? Use the logic above to answer this question.*

### New Fonts

Our game so far has no level separation (like a splash screen coming up and saying”Level 2”) and no graphic support for letting us know our score the time and how many bugs we have killed. So we can make these things lets first get some new fonts for them. You have been provided with three new fonts - cat\_scratch.ttf, bear.ttf, comic.ttf in your Resources folder. Copy them into the Resources->fonts folder in your solution as you did in chapter 2 and then add them to your program( GameResources.vb) using the following specifications:

|  |
| --- |
| NewFont("cat\_scratch", "cat\_scratch.ttf", 40)  NewFont("bear", "bear.ttf", 120)  NewFont("bear\_huge", "bear.ttf", 170)  NewFont("comic", "comic.ttf", 16) |

### Calculating Time Remaining for Display

So we can display the time left we need to first calculate it in the Game loop. We will do this subtracting game time from the endLevelAt time ((endLevelAt - Core.GetTimerTicks(gameTimer)). However you will remember that time is counted in milliseconds so 1000 is 1 second. We don’t really want to show that many digits on the screen it will be too hard to read so the display time we will how will be in tenths of a second. To do this we simply divide by 100. We need to decalre “time” as a Public variable at the top of the module with the other declarations. It is Public as it will be used in the Bug class when calculating the score. This code goes after SwinGame.Graphics.ClearScreen(Color.White) :

|  |
| --- |
| time = (endLevelAt - Core.GetTimerTicks(gameTimer)) / 100 |

### Displaying Player Status

Now after this using our new fonts and time variable we can draw a nice looking current level; current time; bugs killed and current score to the screen during the game, as follows:

|  |
| --- |
| Text.DrawText("Level: " & level, Color.Green, GameFont("cat\_scratch"), 320, 2)  Text.DrawText("Time: " & time, Color.Red, GameFont("comic"), 2, 2)  Text.DrawText("Bugs killed: " & bugsKilled, Color.Green, GameFont("comic"), 2, 60)  Text.DrawText("Score: " & score, Color.Green, GameFont("comic"), 2, 30) |

Exercise 4: *Printing current score and time on the screen*

cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:

1. Load new fonts into your program.
2. Draw the player status to the screen. (Test you game after you do)

*cha 2 - worksheet.pngQuestion 6: Why do you think one “Level” has quatations around it in the DrawText statement above and another one in the same statement does not (level). What do you think the quotations mean?*

*cha 2 - worksheet.pngQuestion 7: In the same statement what do you think the & symbol means?*

*cha 2 - worksheet.pngQuestion 8: If 4 seconds has passed and you are in level 3. What is being diplayed on the screen next to “Time”?*

### Level Introductions

To make a program look more like a proper game we can use level introductions and loser screens. For the level introduction screen, in GameLogic.vb we can create a new sub which will just print current level and score on the screen. To do so, in GameLogic.vb create a new procedure called DrawLevelIntro(). The way this procedure will work is it will loop 50 times to the screen the Level number and the Score. It will clear the screen to black first and each loop will redraw the screen 25 times per second, meaning it will take 2 seconds to draw 50 times, as below:

|  |
| --- |
| Public Sub DrawLevelIntro()  For i As Integer = 1 To 50  Graphics.ClearScreen(Color.Black)  Text.DrawText("Level " & level, Color.Green,  GameFont("bear"), 280, 200)  Text.DrawText("Score: " & score, Color.Green,  GameFont("cat\_scratch"), 320, 300)  Core.RefreshScreen(25)  Core.ProcessEvents()  Next  End Sub |

Now we need to call DrawLevelIntro() from the right place. We will call it from inside the LevelSetUp() procedure right after the Core.StopTimer(gameTimer) in LevelSetUp() procedure.

### Loser Screen

Now for the “You lose” screen. Remember that a player loses when their score goes below zero. If this happens then the game will stop and a message telling them they loose, what level they reached along with how many bugs they killed will be displayed. The screen will loop until the user hits the spacebar key or they close the screen. If they hit space then the score and level are reset and LevelSetUp() is called as below:

|  |
| --- |
| Public Sub DrawLoser()  Do  Graphics.ClearScreen(Color.White)  Text.DrawText("YOU ", Color.Green,  GameFont("bear"), 80, 150)  Text.DrawText("LOOOOSE!", Color.Red,  GameFont("bear\_huge"), 275, 125)  Text.DrawText("You Reached Level " & level,  Color.Green, GameFont("cat\_scratch"), 200, 340)  Text.DrawText("And Killed " & bugsKilled & " Bugs",  Color.Green, GameFont("cat\_scratch"), 240, 400)  Text.DrawText("Press SPACE to play again", Color.Green,  GameFont("cat\_scratch"), 120, 480)  Core.RefreshScreen(25)  Core.ProcessEvents()  Loop Until Input.WasKeyTyped(Keys.VK\_SPACE) Or  SwinGame.Core.WindowCloseRequested() = True  If Input.WasKeyTyped(Keys.VK\_SPACE) Then  score = 0  level = 1  LevelSetUp()  End If  End Sub |

This “loser” screen should be called from within the statement that checks if the time has gone below zero which is located in the Game Loop. Inside this statement you need to create another If statement that checks if the score is less than zero and if it is then to call DrawLoser() sub.

### Penalty Notice

Remember that when the timer runs out the player gets a score penalty. We want to make that more obvious to the user so they know how much score they have lost. We can use the same method we used for the level intro with the following modifications:

* Do not clear the screen instead create a red rectangle that fills up the screen with an alpha of 5 (look up chapter 1 and 2)
* Draw the text “Too Slow!” then “– “ then the scorePenalty underneath .
* The whole thing loops 25 times instead of 50.
* Change the name of the method to Penalty()

This method should be called after the score is penalized in the Game Loop.

Exercise 5: *Printing a start and end point of the game*

*cha 2 - worksheet.pngMake the following changes in your program and write your solutions to the worksheet:*

1. *Build into your program the level introductions.*
2. *Buld into your program the loser screen.*
3. *Build into your program the penalty notice*

*cha 2 - worksheet.pngQuestion 9: How long does the Penalty Notice stay on the screen for(in seconds)?*

Now test, play and enjoy your very own SwinGame……

saveicon.png Remember to save your project (File – Save All).