**GAMEDEVREJECTS**

**C# Programming**

**DECISION STATEMENTS IF ELSE, ELSE IF & LOGICAL OPERATIONS**

**Date: 22-Apr-22**

**All Copyright GAMEDEVREJECTS©2022**

**Contents**

[**7.0** **Why Do We Need IF, ELSE IF & ELSE Decision (or Conditional) Statements** 3](#_Toc101548005)

[**7.1** **Create A New Project & Script** 3](#_Toc101548006)

[**Glossary** 9](#_Toc101548007)

[**References** 10](#_Toc101548008)

# **7.0 Why Do We Need IF, ELSE IF & ELSE Decision (or Conditional) Statements**

**IF, ELSE IF and ELSE** Statements aka Conditional (or Decision) statement are used for making basic decisions in the game program. So why do we need decision statements in games. Well for example, suppose in a game a player has a health bar. Which starts at 100%, but every time that player gets hit by the enemy. The health decreases by 10%, until it gets to zero; then at that point we want the “Game Over” function to be happening. On the other hand, ...every time the players health is > zero. We just want the player to continue playing in the game or in the level. As one can see “a-value-based decision” needs to be made to determine the life of the player. **IF, ELSE IF and ELSE** STATEMENTS provide a framework to check the values and decide every frame in a game. In other words. For each frame, every time the value is less-than, equal to or greater than something we want to do something.

# **7.1 Create A New Project & Script**

Create a new Project, Scene then attach a Script. See Chapter 00.0 Create A Scene.

Or create an empty object, then attach a new script, since we are just using the Console Output anyway as follows

(NavigateTo).SCENE -- (IN).HIERARCHY -- / (Right Click).**Create Empty** /

(IN).INSPECTOR – Transform – (3 dots) => / **(Select).Reset** /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / **NewBehaviourScript** /

(IN).\_SCRIPTS -- (Select).**NewBehaviourScript**.(Drag and Drop) – HIERARCHY => / (Onto).**GameObject** /

(IN).\_SCRIPTS => / (Select).**NewBehaviourScript**.(Double Left Click) – VSCODE /

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class NewBehaviourScript : MonoBehaviour

{

    private int health=100;

    // Start is called before the first frame update

    void Start()

    {

    }

    // Update is called once per frame

    void Update()

    {

        if(health < 1)

        {

            Debug.Log("Game Over !!!");

        }

        else

        {

            Debug.Log("Keep Playing !");

        }

    }

}

Graphical user interface, application

Description automatically generated

Ok, so, Let’s review what we've entered in so far. We've declared and defined a variable. Health has a data type of int and gets a value of 100. We can ignore the void start method. It does not do anything yet. We could probably delete it. But for the sake of completeness we’ll keep it in.

So, inside the update method we've created an if block, and an else block. So, first of all we write an 'if' statement, then in a pair parenthesis. We write the condition or test. Then in curly braces we write what we want to do if the condition is true. So here we are checking if health is “less than 1 then only” print to the console **"Game Over"**

Otherwise if health is not less than 1 i.e health is greater than 1 then execute the code in the “else block”.

So, from the start of the program we can see that 100 is not less than 1 therefore the condition is false. We execute the statements inside of the “else block”.

So, let’s save it, then run it ! > Check the main public class "name of script" matches the filename, otherwise the script will not be found. Drag and drop on the script on the object (cube) in the screen. OR add the script as a component if we prefer then now **“Save it & Run it”**. We should see output from the dialog box ! => Debug.Log ("Keep Playing") -> which means the 1st condition is not satisfied i.e. false therefore we print "Keep Playing" in the console.

Ok, so now inside the Start method. We are going re-define health a value to zero. The first IF condition should be satisfied, and we should see GAME OVER echo outputted to the console.

So, go back to the script > double click on the script icon. We don’t need to redeclare the data type. We only do that once. Then type health =0; (CTRL + SHIFT to recompile and hit SAVE). Then F5 > Hit Play. Now we should see “GAME OVER” output to the console. Because the 1st condition is satisfied. So, we've just gone over the basics of IF-ELSE statements.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class NewBehaviourScript : MonoBehaviour

{

    private int health=100;

    // Start is called before the first frame update

    void Start()

    {

        health=0;

    }

    // Update is called once per frame

    void Update()

    {

        if(health < 1)

        {

            Debug.Log("Game Over !!!");

        }

        else

        {

            Debug.Log("Keep Playing !");

        }

    }

}

Graphical user interface, application

Description automatically generated**7.2 ELSE IF Statements AND Logical operators**

To check more than one condition at the same time…what we do is use the ELSE IF statement after the IF BLOCK, but before the ELSE block. So, for example if we wanted to check IF health > 10…we could echo output => "In good health".

Also, if health > 50 we could echo output => "Ultra-Power"

So, let’s go back to our script and insert an ELSE IF block

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class NewBehaviourScript : MonoBehaviour

{

    private int health=100;

    // Start is called before the first frame update

    void Start()

    {

        health=60;

    }

    // Update is called once per frame

    void Update()

    {

        if(health < 1)

        {

            Debug.Log("Game Over");

        }

        else if(health > 10)

        {

            Debug.Log("In good Health");

        }

        else if(health > 50)

        {

            Debug.Log("Ultra Health");

        }

        else

        {

            Debug.Log("Playing");

        }

    }

}

Graphical user interface

Description automatically generated

ok to recap our 1st condition is to check if health is < 1 we debug.log Game Over. Our 2nd condition is ELSE IF health > 10 => then print "In Good Health". Our 3rd condition is ELSE IF health > 50 => then print "Ultra Health". Else if none of the conditions are true then it will skip all the IF, ELSE IF blocks and use the default ELSE, and print out "keep playing".

So, if we go back to our unity script. And change the value of health to 11. Then when we execute the script it should output "In Good Health".

Last run if we change the value of health to 60. Then when we re-run the script we should output "Ultra-Health"

OK, as we can see if I change the health to 60. It still prints in good health. 60 although is > than 50. It is also greater than 10. And because the greater than 10 condition gets satisfied first. The program exits the script before reaching the 3-rd. condition.

So, to correct this. To make all the check conditions distinct from each other. We need to introduce a concept called "LOGICAL OPERATORS & OPERANDS" to our repertoire of IF ELSE IF ELSE blocks. So, for the 2nd condition we need to change the health is less than or equal 10 and the 3rd condition is greater than 10 but also less than or equal to 80.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class NewBehaviourScript : MonoBehaviour

{

    private int health=100;

    // Start is called before the first frame update

    void Start()

    {

        health=60;

    }

    // Update is called once per frame

    void Update()

    {

        if( health < 1)

        {

            Debug.Log("Game Over");

        }

        else if(health <= 10)

        {

            Debug.Log("In good Health");

        }

        else if(health > 11 && health <=80)

        {

            Debug.Log("Ultra Health");

        }

        else

        {

            Debug.Log("Playing");

        }

    }

}

Graphical user interface

Description automatically generated

# **Glossary**

# **References**

/End