# Tutorial #6 – If/Else

## Overview

This tutorial will show a few examples of how to create achievements using the logical constructs \*\*If\*\* and \*\*Else\*\*. [Tetris DX](<https://retroachievements.org/game/4939>) was chosen since it has a couple of achievements in the set that test for a certain puzzle pattern. The examples show a method of creating these test patterns using a two-dimensional array and logical constructs. They also show how you can make functions more versatile with logical constructs. If you are unfamiliar with the concept of \*\*If/Else\*\* then please review the Khan Academy videos [If/Else - Part 1](https://www.khanacademy.org/computing/computer-programming/programming/logic-if-statements/pt/ifelse-part-1) and [If/Else - Part 2](<https://www.khanacademy.org/computing/computer-programming/programming/logic-if-statements/pt/ifelse-part-2>).<\br>

![Tetris DX Title Screen](Tetris\_DX\_Title.png)<\br>



## Logical Constructs

Logical constructs allow you to control which blocks of code are executed. Which block of code that gets executed depend on value of the test condition. The test condition must be a \*\*Boolean\*\* value meaning the result is either \*\*true\*\* or \*\*false\*\*. For the following example assume a number that is randomly selected between 0 and 10. Depending on the value of the number different blocks of the code will be executed if number is between less than 5 or greater than or equal to 5.

```

If (number < 5)

{

// Execute this block of code if the number is less than 5

}

else

{

// Execute this block of code if the number is greater than or equal to

}

```

Note that if the block of code is only one line the curly brackets can be omitted.

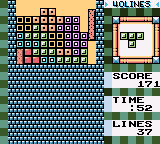
## Compile Time vs. Runtime

A very important thing to note about logical constructs in RAScripts is that they are \*\*Compile Time\*\* operations! Being \*\*Compile Time\*\* means that the RAScript code is ran once in RATools where it is translated into the achievement code used by RALibretro and other emulators. As a result, you \*\*cannot\*\* used emulation memory values for logical constructs since that memory does not exist when the RAScript is compiled. Conversely, the achievement code is ran at \*\*Runtime\*\* meaning that it is ran with the emulator and has access to the emulator’s memory values. This might be a confusing point if you are new to the world of programming, just know that if you are having an issue with a logical construct it may be that you are trying to access emulator memory with it.

# Example 6A: 40 Lines Mode

Tetris is a puzzle game where the main goal is to rotate and align the falling Tetriminos to create a solid horizontal line. When a solid horizontal line is made the game will remove the line from the board and award points depending on how many lines were cleared at once. The 40 Lines Mode in [Tetris DX](https://retroachievements.org/game/4939) is a special mode were you must complete 40 lines at various starting heights and speeds. The higher the starting height and/or speed the more difficult the challenge is.<\br>

![Tetris DX screen of 40 lines mode at 5 starting height](Tetris\_DX\_40Lines.png)<\br>



```

// Tetris DX

// #ID = 4939

// $AC0A: 0=marathon 1=ultra

// 2=40lines 3=vs.com

function GameType() => byte(0x00AC0A)

// $AC24: Number of lines completed

function Lines() => byte(0x00AC24)

// $DB10: speed level in 40 lines

function Speed() => byte(0x00DB10)

// $DB11: high-level

function HighLevel() => byte(0x00DB11)

// Create a forty lines mode achievements with starting high-level, speed conditions, or both

function FortyLines(title, description, points, highLevel, speed)

{

// Always check for game mode 40 lines and

// if previous lines was between 36 to 39 and the current lines is 40 or more

trigger = GameType() == 2 &&

prev(Lines()) >= 36 &&

prev(Lines()) <= 39 &&

Lines() >= 40

// If the high level setting is greater than zero then add the high level condition

if (highLevel > 0)

trigger = trigger && HighLevel() == highLevel

// If the speed setting is greater than zero then add the speed condition

if (speed > 0)

trigger = trigger && Speed() == speed

achievement(title, description, points, trigger)

}

// Create seven forty lines mode achievements using variations of the same function

FortyLines("Example 6A: One-High", "Beat [40 Lines] mode on 1-High.", 2, 1, 0)

FortyLines("Example 6A: Two-High", "Beat [40 Lines] mode on 2-High.", 3, 2, 0)

FortyLines("Example 6A: Three-High", "Beat [40 Lines] mode on 3-High.", 5, 3, 0)

FortyLines("Example 6A: Four-High", "Beat [40 Lines] mode on 4-High.", 5, 4, 0)

FortyLines("Example 6A: Five-High", "Beat [40 Lines] mode on 5-High.", 10, 5, 0)

FortyLines("Example 6A: Mach 40", "Beat [40 Lines] mode on Speed Level 9.", 5, 0, 9)

FortyLines("Example 6A: Light Speed", "Beat [40 Lines] mode on 5-High and Speed Level 9.", 10, 5, 9)

```

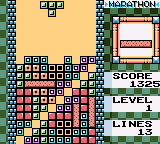
## Dealing with Variations

The above example takes advantage of the fact that only one or two conditions change for each of the 40 Line Mode achievements. Using \*\*logical constructs\*\*, the resulting achievement is built differently depending on the passed parameters. If the \*highLevel\* parameter is greater than zero then an additional condition is added to the achievement to check if the current high level value is the passed high level value. Similarly, if the \*speed\* parameter is greater than zero then an additional condition is added to the achievement to check if the current speed value is the passed speed value. By adding a few \*\*logical constructs\*\* to the function \*FortyLines()\* we expanded its capabilities to handle multiple starting heights and/or speeds.

# Example 6B: Not Quite a Grandmaster

The Tetris board is limited to 10 blocks across and 20 blocks high (10x19 on the game boy due to the reduced screen size). Once blocks stack over the height limit the game is over. To become a Tetris Grandmaster you must be able to control where the gaps in a line are no matter what pieces you are given. One way to practice controlling the board is to perform the Grandmaster build which starts with a gap on the far bottom left of the board and then has a gap one cell to the right and up until the gap is on the other side of the board. Master this build and you are one step closer to becoming a Tetris Grandmaster!<\br>

![Tetris DX screenshot of a Grandmaster build](Tetris\_DX\_Grandmaster.png)<\br>



```

// Tetris DX

// #ID = 4939

// $AD02: Tetris Board 10x19

// Cell(0,0) = $AD02, Cell(9,0) = $AD0B,

// Cell(0,1) = $AD12, Cell(9,1) = $AD1B,

// ...

// Cell(0,18) = $AE22, Cell(9,18) = $AE2B

function Cell(x,y) => byte(0xAD02 + x + (y \* 16))

// Shortcut cell definitions

MT = 0xfb // Empty square

HL = 0xb0 // I horizontal left piece (orange)

HC = 0xb1 // I horizontal center piece

HR = 0xb2 // I horizontal right piece

VT = 0xb3 // I vertical top piece

VM = 0xb4 // I vertical center piece

VB = 0xb5 // I vertical bottom piece

LP = 0x88 // L piece (red)

JP = 0x90 // J piece (light blue)

TP = 0x98 // T piece (light green)

ZP = 0xa0 // Z piece (yellow)

SP = 0xa8 // S piece (purple)

OP = 0x80 // O piece (black)

// Special cells

NE = 0xff // Any square (not empty)

NO = 0x0 // Nothing (ignore)

// Build a set of conditions to test for the passed map

function TetrisMap(map)

{

mapArray = []

for y in range(0,18)

{

for x in range(0,9)

{

// Get the value of current cell

cell = map[y][x]

if cell == NE

// If any cell then test not empty

array\_push(mapArray, Cell(x,y) != MT)

else if cell != NO

// If not ignoring cell then test for current cell value

array\_push(mapArray, Cell(x,y) == cell)

}

}

return all\_of(mapArray, a => a)

}

grandmaster= [

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NE,NE,NE,NE,NE,NE,NE,NE,NE,MT],

[NE,NE,NE,NE,NE,NE,NE,NE,MT,NE],

[NE,NE,NE,NE,NE,NE,NE,MT,NE,NE],

[NE,NE,NE,NE,NE,NE,MT,NE,NE,NE],

[NE,NE,NE,NE,NE,MT,NE,NE,NE,NE],

[NE,NE,NE,NE,MT,NE,NE,NE,NE,NE],

[NE,NE,NE,MT,NE,NE,NE,NE,NE,NE],

[NE,NE,MT,NE,NE,NE,NE,NE,NE,NE],

[NE,MT,NE,NE,NE,NE,NE,NE,NE,NE],

[MT,NE,NE,NE,NE,NE,NE,NE,NE,NE]

]

achievement(

title = "Example 6B: Not Quite a Grandmaster",

description = "Perform the secret grade grandmaster build up to the opposite wall",

points = 25,

trigger = TetrisMap(grandmaster)

)

```

## Two-Dimensional Arrays

To test for the Grandmaster build a two-dimensional array (an array of arrays) is defined with the desired value for each cell. Puzzle and single screen games typically have a map of cells that represent the board game. In Tetris’s case the map of cells define several block types that could be in each cell. If the array is:

* ```NE``` then the cell must have a block in it but it can be any type of block
* ```MT``` then cell must be empty
* ```NO``` then cell is ignored so it can be any block or empty
* Otherwise the cell value must match what is in the array

To align the two-dimensional array with the in-game memory the function ```Cell(x,y)``` takes in a x,y coordinate and returns the byte of memory for that cell. When reading the desired value from the two-dimensional you can access the array by index ```map[y][x]```. Note that the x and y are reversed for the array access. This is because the array is define as an array of rows where ```row = map[y]``` gets you the array at row y. To get the cell value from the row you would need to index ```cell = row[x]``` which is the equivalent of ```map[y][x]```. This is just one method for building this achievement, as the designer you can define the array configuration differently or use an entirely different method.

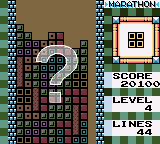
## all\_of

Because the ```TetrisMap()``` function is data driven we don’t know exactly how many conditions are needed until the function is called. The conditions are then built up for any cell that is not ignored (any cell not equal to ```NO```). An empty array ```mapArray[]``` is defined at the top of the function and all the not ignored cell conditions are pushed on to the array. At the end of the function the ```all\_of``` command takes the array of conditions and returns a single condition with everything &&’d together. Similarly, the ```any\_of``` command will take in an array and will ||’d together each condition. The conditions could have been built using a similar method as [Example 6A](Example\_6A.md) with ```trigger = trigger && condition``` however, the ```all\_of``` is more concise.

# Example 6C: Pattern Design

The ultimate goal of Tetris is survival since the blocks will fall faster as the game goes on giving you less time to react. Being able to quickly recognize patterns can improve your ability to play at the high levels. As such being able to create pre-defined patterns is the hallmark of a great Tetris player!<\br>

![Tetris DX screenshot of a question mark](Tetris\_DX\_Question\_Mark.png)<\br>



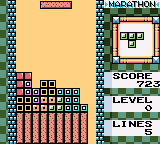
## Homework #6

A) Design a 10x19 Tetris pattern<\br>

B) Use the code from [Example 6B](Example\_6B.md) as a guide and replace the Tetris array map with the pattern defined in part A.

# Example 6C: Tetris Deluxe

1. For this example we chose the “Tetris Deluxe” pattern which requires you to fill the bottom 4 rows with vertically stacked ‘I’ beams. The ‘I’ beam is the most important Tetrimino type in the game since it is the only Tetrimino that can make a \*\*Tetris\*\* (clearing four lines at once). Note that the top, middle, and bottom cells of the ‘I’ beam are all different. All of the other Tetrimino types share the same cell for every part of the block.



![Tetris DX screenshot of a the Tetris Deluxe pattern](Tetris\_DX\_Deluxe.png)<\br>

1. Source code for “Tetris Deluxe” pattern.

```

// Tetris DX

// #ID = 4939

// $AD02: Tetris Board 10x19

// Cell(0,0) = $AD02, Cell(9,0) = $AD0B,

// Cell(0,1) = $AD12, Cell(9,1) = $AD1B,

// ...

// Cell(0,18) = $AE22, Cell(9,18) = $AE2B

function Cell(x,y) => byte(0xAD02 + x + (y \* 16))

// Shortcut cell definitions

MT = 0xfb // Empty square

HL = 0xb0 // I horizontal left piece (orange)

HC = 0xb1 // I horizontal center piece

HR = 0xb2 // I horizontal right piece

VT = 0xb3 // I vertical top piece

VM = 0xb4 // I vertical center piece

VB = 0xb5 // I vertical bottom piece

LP = 0x88 // L piece (red)

JP = 0x90 // J piece (light blue)

TP = 0x98 // T piece (light green)

ZP = 0xa0 // Z piece (yellow)

SP = 0xa8 // S piece (purple)

OP = 0x80 // O piece (black)

// Special cells

NE = 0xff // Any square (not empty)

NO = 0x0 // Nothing (ignore)

// Build a set of conditions to test for the passed map

function TetrisMap(map)

{

mapArray = []

for y in range(0,18)

{

for x in range(0,9)

{

// Get the value of current cell

cell = map[y][x]

if cell == NE

// If any cell then test not empty

array\_push(mapArray, Cell(x,y) != MT)

else if cell != NO

// If not ignoring cell then test for current cell value

array\_push(mapArray, Cell(x,y) == cell)

}

}

return all\_of(mapArray, a => a)

}

deluxe = [

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[NO,NO,NO,NO,NO,NO,NO,NO,NO,NO],

[VT,VT,VT,VT,VT,VT,VT,VT,VT,VT],

[VM,VM,VM,VM,VM,VM,VM,VM,VM,VM],

[VM,VM,VM,VM,VM,VM,VM,VM,VM,VM],

[VB,VB,VB,VB,VB,VB,VB,VB,VB,VB]

]

achievement(

title = "Example 6C: Tetris Deluxe",

description = "Get a Tetris by hoarding vertical I pieces in the bottom 4 blocks",

points = 10,

trigger = TetrisMap(deluxe)

)

```