

Towers of Hanoi

**Project Game Technologies**



**Name:** Jenny Sun

**Student Number:** 500746763

**Subject:** Project Game Technologies

**Teachers:**

D. G. A van der Meulen

J. van Oerle

E. M. J. Hofstede

B. W. Pijls-van Kooten

**Mobile Number:** 06-43958927

**Date:** 19-03-2019

# Introduction

The Towers of Hanoi problem is a mathematical puzzle (or game) invented by the French mathematician Édouard Lucas in 1883. It exists of three rods and a number of disks of different dizes which can slide onto any rod. The goal is to move the tower of disks from the left rod to the right-most rod. Thus, our assignment is to create a program in C++ which allows us to solve this puzzle. In this report, I will describe the steps which were required to find the solution to the puzzle. This includes an in-depth explanation with code snippets and several reference images.

Contents

[Introduction 1](#_Toc3935228)

[The Towers of Hanoi Puzzle 3](#_Toc3935229)

[Visual Representation 4](#_Toc3935230)

[Exemplar: Disk Number 5](#_Toc3935231)

[Recursive Method 6](#_Toc3935232)

[Pointers and References 7](#_Toc3935233)

[Self-assessed points 8](#_Toc3935234)

[Conclusion 9](#_Toc3935235)

# The Towers of Hanoi Puzzle

The amount of disks is dependent on the user input and in our case varying between 3 to 7 disks to keep a neat overview. According to this input, the program will take an x amount of steps to reach the end goal. The goal is reached when the stack of disks from the source pillar is moved to the destination pillar according to the three simple rules:

1. One disk can move at a time.
2. Only the uppermost disk on a stack can be moved.
3. A larger disk may not be placed on top of a smaller disk.

And to complete this puzzle according to the assignment, we will also have these requirements:

* The programming language has to be C++.
* The program makes use of pointers and references.
* There is a visual representation of both execution and the result of the algorithm (ASCII).
* The puzzle should be able to solve itself.

The **exemplar** that I’ve chosen is as stated in the intro:

* Disk number can be set by the user (4, 5, 6, 7 *or more disks*).

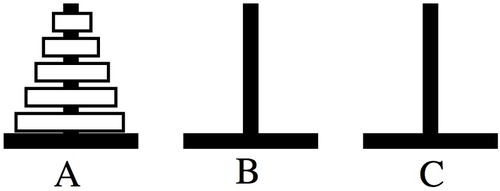
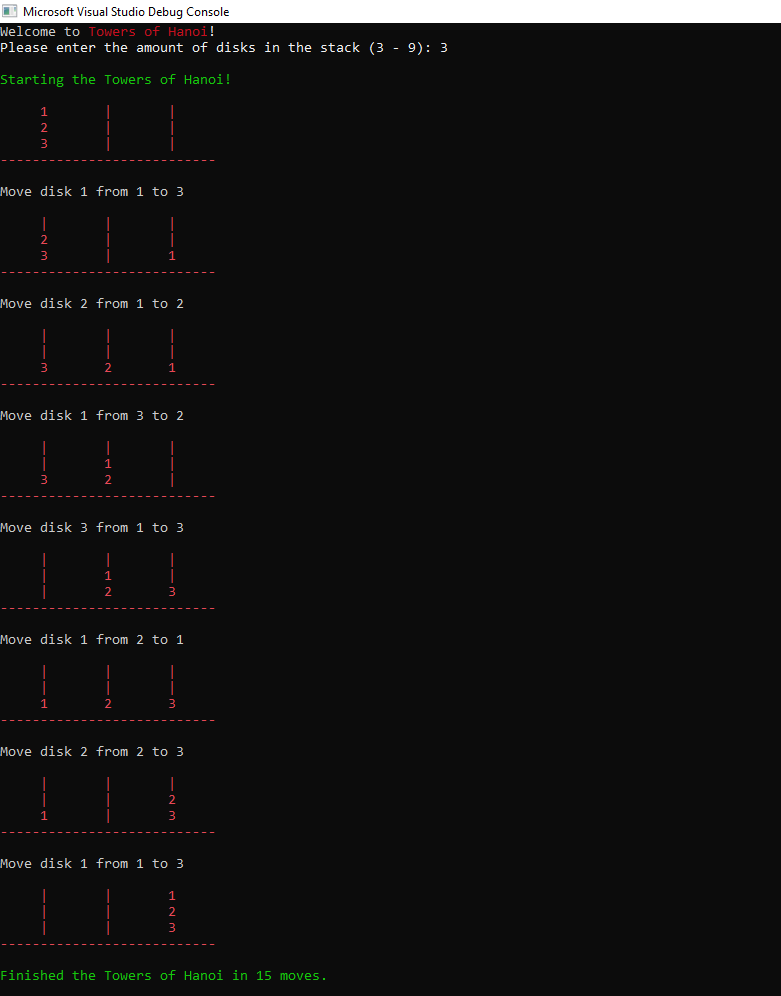


Image: Visual Representation of the Towers of Hanoi.

# Visual Representation

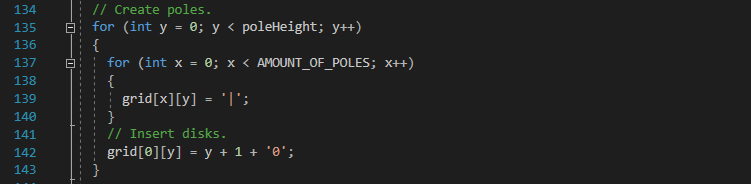
For this assignment I have used ASCII to visually show the work of the algorithm. Having the ‘|’ symbol representing the poles in which the disk will be placed and the numbers for the order of the disks from top-down low-high. The colours are just added for more pleasure to the eyes.



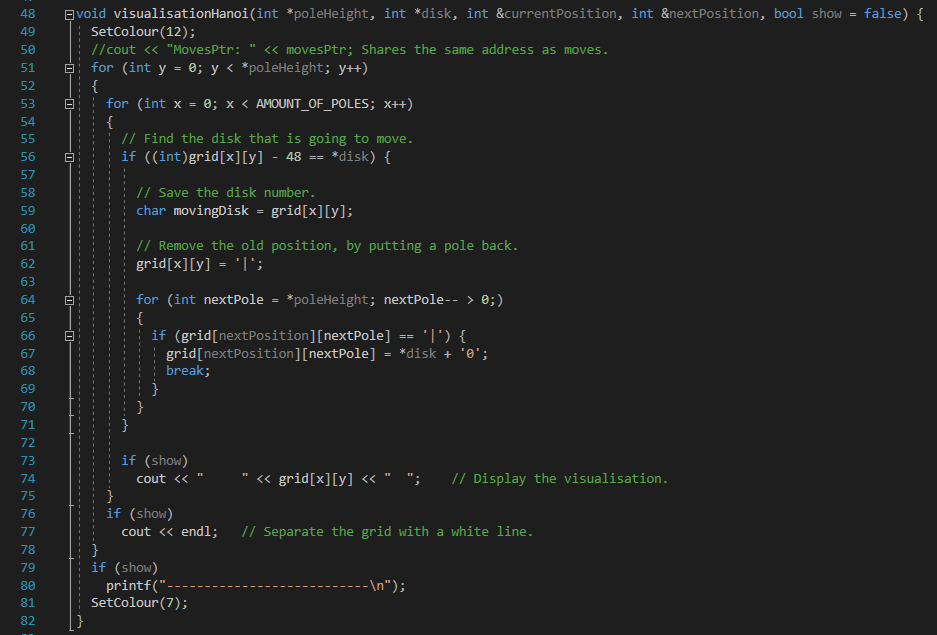
To create this ASCII representation, I have used a two-dimensional array, which provides as a grid. This array is then filled with pole symbols (‘|’). It could also be done by predefining the array and filling the array manually, like so:

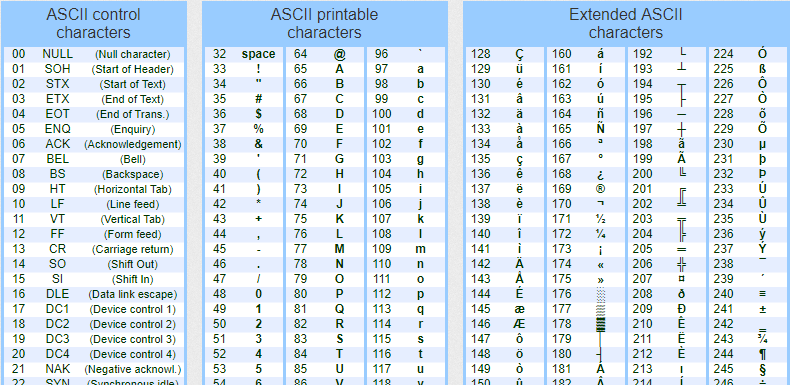
char[][] poles = { {'|', '|', '|',}, {'|', '|', '|'}, {'|', '|', '|'} }

but as the array needs to adapt its length to the user input, I had to find a dynamic way to do so. Thus, I created a for-loop and filled this with poles. I have written this in the main() function. Also for inserting disks, I have done this the same way I’ve done for the poles.



As for the visualisation for the movement of the disks, I have written the following code:



What I have basically done is create two loop for my two-dimensional array. The two will loop through this array and the first thing they do is to find the disk that is going to move and as in this case we are comparing a char to an int, we need to convert this char to an int. As disk is always a number, we can substract 48 from the ASCII table, to give its original value, which is 0.

Then we remove the old position, by putting the pole back, using the ‘|’ symbol. The for-loop that comes after is used to move the disk to the next position. Since we are limited to the height of the pole, we use a reverse loop going from the bottom to the top to check if there is an available spot where we can put the disk. This prevents floating disks and allows the inserted disk to start at the bottom. As soon as we find an available spot (which is ‘|’ the pole symbol), we can proceed with inserting the disk in that position. Next is the:

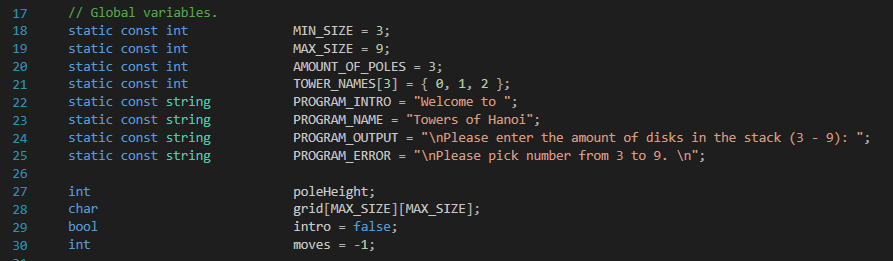
grid[nextPosition][nextPole] = \*disk + '0';

this code means that the disk will be inserted in this position on the grid. However, since the disk is an integer, we need to convert it to a char (the grid is made out of chars). This is possible by adding + ‘0’. At the end of this inverse loop, we will add a break to stop the loop, because then we are done searching for an available spot.

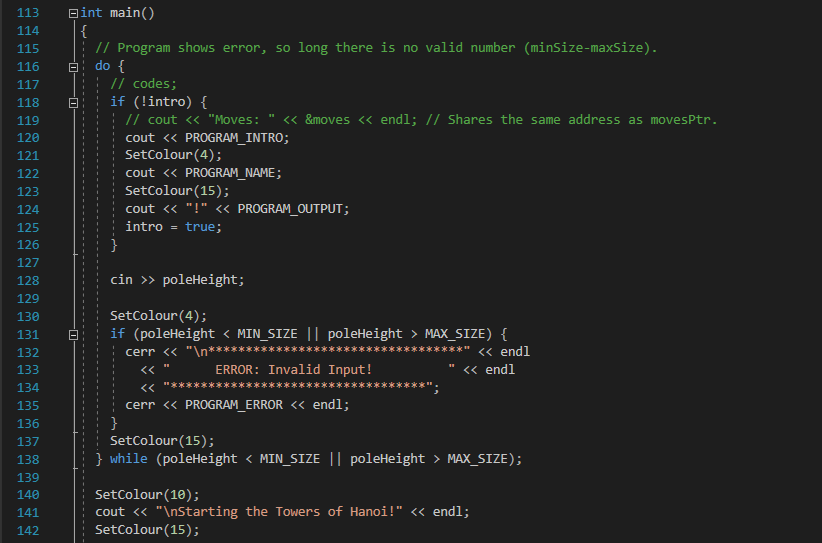
# }Exemplar: Disk Number

Just like the previous assignment, I have picked the user input: **Disk number can be set by the user. (4, 5, 6, 7 *or more disks*).** In the program I have created the user is able to have an input of 3 to 9. This is just because I prefer a limit to an input. Having bigger numbers means that the algorithm will take longer to find the solution. It also means that at some point the ASCII will not provide its visuals as effective anymore. The overview of its visuals will be lost.

To store and receive the input to use them afterwards, I have created some variables outside the main, making them all global variables:



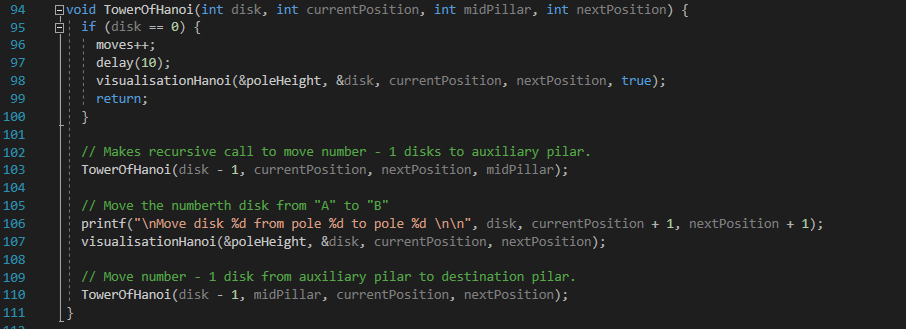
MIN\_SIZE here indicates the minimum amount of disks the user can put, which is three. Lower than three will make the algorithm quite pointless. The MAX\_SIZE is nine, because as explained above, we would like to keep a clean overview for the ASCII art. If the MAX\_SIZE is too huge, it will lose its functionality. As for the PROGRAM\_INTRO, PROGRAM\_NAME, PROGRAM\_OUTPUT and PROGRAM\_ERROR, these are displayed to the user to ask for certain actions or give some feedback in case the user puts an invalid number.



Then next in the main, the user will be able to pick a number and insert this in the program. This number will be received as poleHeight and as we have stored it as a global variable, we can use this as a parameter to call into our algorithm function.

# Recursive Method

To solve the Towers of Hanoi puzzle, I have used recursion. Recursion is calling the same action from that action. So there is one rule for doing any recursive work: there must be a condition to stop that action from executing, otherwise it will just loop forever and the program will crash. This condition is called a **terminal state.** It is a state where we are not going to call this function anymore.

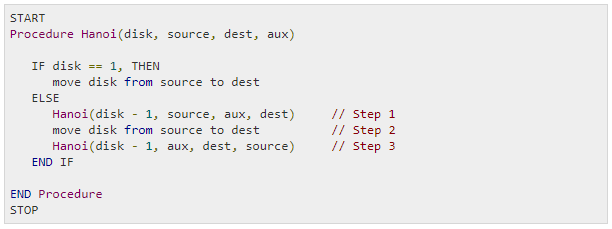


The steps which I followed were:

1. Move n-1 disks from **source** to **aux** (currentPosition to midPillar in the code).
2. Move nth disk from **source** to **dest** (currentPosition to nextPosition)/
3. Move n-1 disks from **aux** to **dest** (midPillar to nextPosition).

These steps are displayed in my code as comments.

This was the pseudo-code I have used:



Source: <https://www.tutorialspoint.com/data_structures_algorithms/tower_of_hanoi.html>

# Pointers and References

# Self-assessment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Bad | Moderate | Sufficient | Advanced |
|  | 0 point | 1 points | 2 points | 3 points |
| Development basics | Development environment not installed on student's ​laptop. | Game engine, IDE, and version control can be shown to be installed on student's laptop and C++ assignment meets ​all​ ​requirements | Game engine, IDE, and version control can be shown to be installed on student's laptop, C++ assignment meets all requirements and at least one exemplar (​see assignment). | Game engine, IDE, and version control can be shown to be installed on student's laptop, C++ assignment meets all requirements and​ is build and executed in the game engine​. |

The reason why I have given myself two points, is because:

* Microsoft Visual Studio is installed and works on my laptop.
* Unreal Engine is installed and works on my laptop.
* SourceTree keeps track of my version control of Towers of Hanoi and our team project “Zeeminmin”.
* All the requirements of the C++ assignment for Towers of Hanoi are met:
  + Programming language has to be C++.
  + The program makes use of pointers and references.
  + There is a visual representation (hint: console ASCII art is an easy solution).
  + There are at least three disks.
  + The disk movement follows the three rules.
  + The puzzle should be able to solve itself
* At least one exemplar is met:
  + Disk number can be set by the user (4, 5, 6, 7 *or more disks*).

# Conclusion

Overall this was a really interesting and fun assignment to do. I enjoyed this assignment more than Knight’s Tour and learnt even more about C++. In some aspects the C++ language was hard to implement, because it is quite a new language for me. Also C++ is a language that deals with things differently than languages as Java. For instance, C# has a garbage collector, which C++ does not have. That is why the use of pointers and references are important in C++. It prevents a memory leak and enables the possibility to manage memory allocation manually. Personally I have never made use of pointers nor references, so that was a challenge for me. Even so I would recommend anyone to try out C++ and do solve the Towers of Hanoi puzzle. It is really educational and a good base to start learning C++.