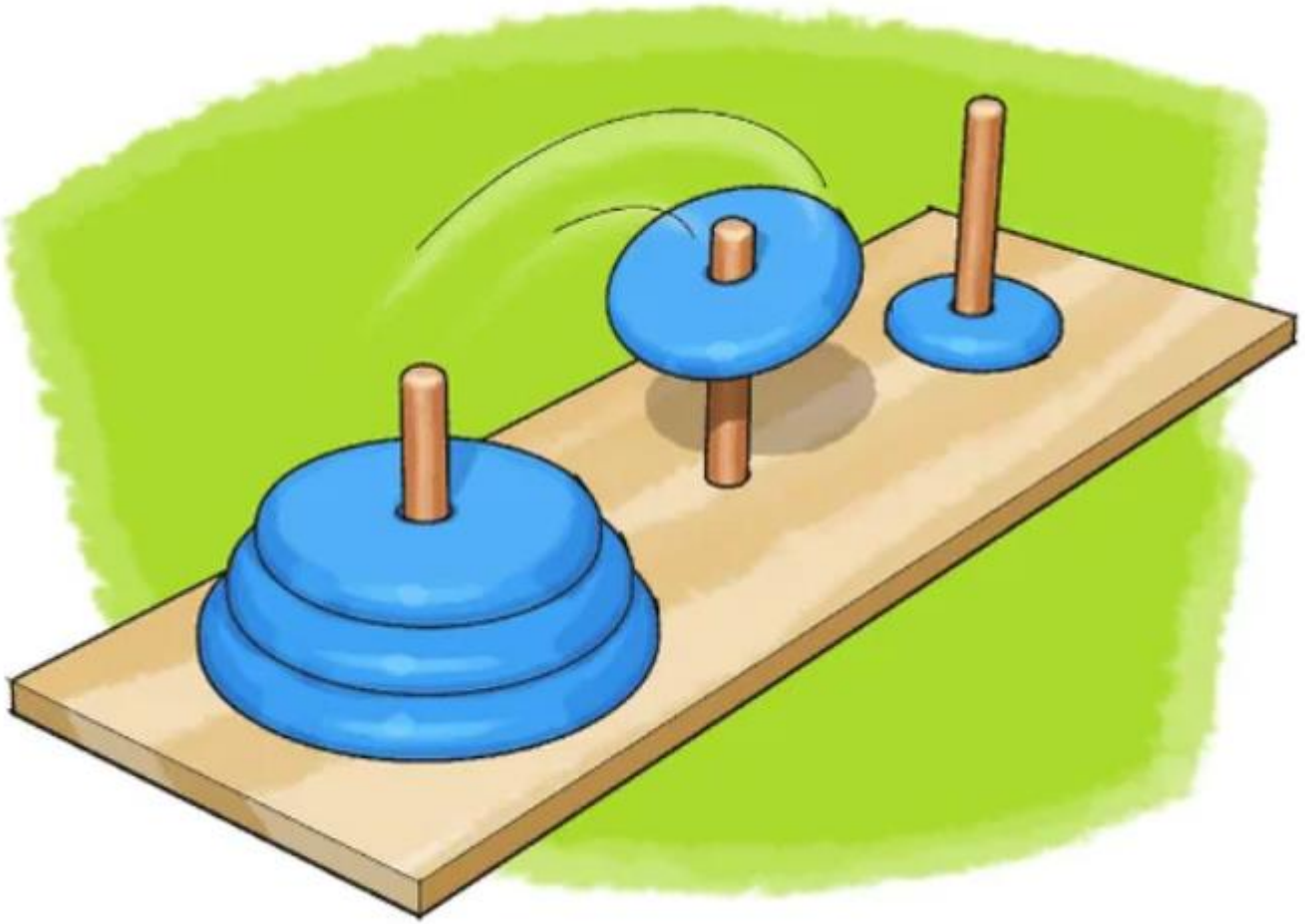
Interactive Tower of Hanoi
v 1.4



History

- There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it, surrounded by 64 golden disks. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks in accordance with the immutable rules of Brahma since that time. The puzzle is therefore also known as the Tower of Brahma puzzle. According to the legend, when the last move of the puzzle is completed, the world will end.

- If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves it would take them $2^{64} - 1$ seconds or roughly 585 billion years to finish, which is about 42 times the current age of the Universe !

What is Tower of Hanoi ?

- Is a mathematical game or puzzle It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

- To win the Game you should move the entire stack to another rod,

  obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

code development phases

Our code has four phases of development :

- Version 1.0: procedural programming, and implement the code using recursion
- Version 1.1: procedural programming, and implement the code using iterative algorithms
- Version 1.2: we add OOP paradigm by making class Solution which contains all the methods
- Version 1.3: we add drawing towers at run time
- Version 1.4: we add protecting features to our code to prevent users from typo

Code description

Our code is following object-oriented paradigm and its divided into two classes :

- The first one is stack that implemented in linked nodes
- The second one is the solution class which divided into 5 sections:

Section (1)

welcome message and let the user choose way of solution :

a) User Solution b) Machine Solution

Section(2):

Solution algorithms : this section include

- checking if movement is legal or not to protect rules of the game.
- protecting user from illegal movement , that leads the user to make fewer movements to help him win the game as optimal movement as he can.

Section(3):

prepare to show the user what happened in every step :

by converting the (3) Stacks into (3) Arrays and passing them to the drawing function.

Section(4):

Drawing algorithm to draw the towers in command line at the run time.

Code description cont.

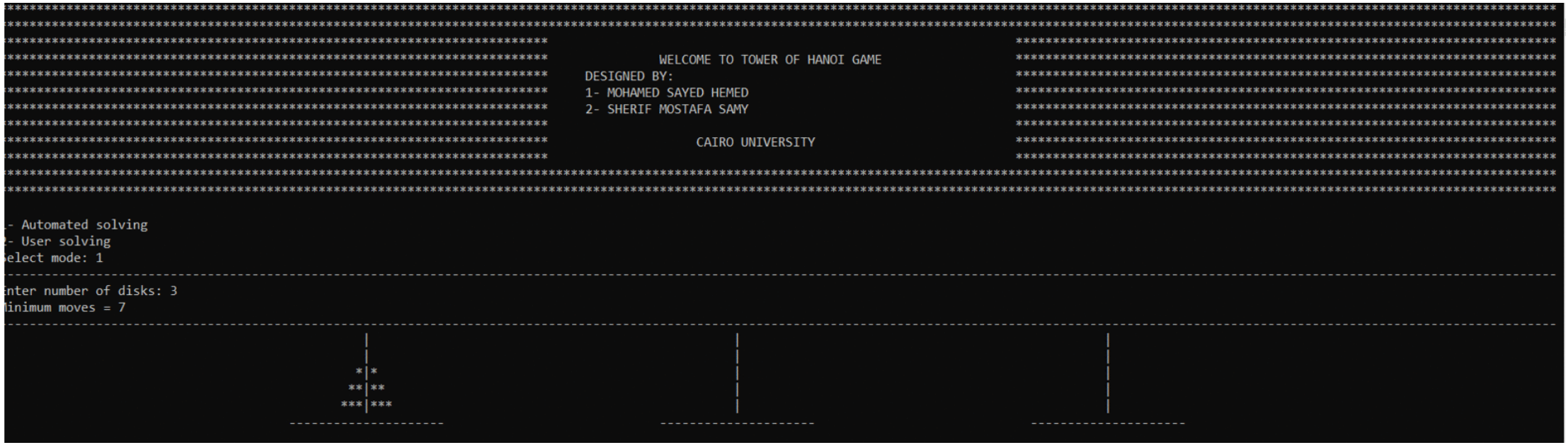
- To solve tower of Hanoi problem each tower is represented by a stack in linked implementation to make number of nodes according to user desire.

- Every movement during solution journey must be displayed at the screen in one of two ways :

1) if number of disks less than or equal eight disks then the towers will be drawn at runtime in command line , we choose the max number = (8) to be friendly and comfortable to human eyes.

2) if disks larger than 8 disks then solution , we write every step by step for example : move from tower one to tower three

Solution Algorithm



Solution Algorithm

- We try to solve this problem ourselves until we reach a pattern :

  if we consider that three towers Are :

A B C

Then if number of disks is even :

Move from A ---> B in the direction which determined by the lesser top of (A,B)

Move from A ---> C in the direction which determined by the lesser top of (A,C)

Move from B ---> C in the direction which determined by the lesser top of (B,C)

Until the A,B become empty , and C have all disks stacked on each other.

But if number of disks is odd:

Move from A ---> C in the direction which determined by the lesser top of (A,B)

Move from A ---> B in the direction which determined by the lesser top of (A,C)

Move from B ---> C in the direction which determined by the lesser top of (B,C)

Code wiki

Method prototype	Return type	Arguments	description
initialization			
void intro();	void	No arguments	- Welcome message - initialization screen
void printLine();	void		
void startingScreen()	void		
Solving algorithm			
void machineSolve(int&,int&);	Boolean	Alias int	This method take alias int which equal number of disks and raise true flag whenever the solution done. Other int is for detect delay time that user desired
void userSolve(int&);	void	Alias int	This method take take alias int which equal number of disks , to enable the user to put disks into towers manually.

void fromAToB(StackLinked<int>&,StackLinked<int>&);
void fromAToC(StackLinked<int>&,StackLinked<int>&);
void fromBToC(StackLinked<int>&,StackLinked<int>&);
	void	Two Alias objects of type Stack	These three methods take two stacks each of them represent one tower and decide the correct direction depends on the lesser top of them.

bool checkEmpty(StackLinked<int>&,StackLinked<int>&);	Boolean		This method take two stacks and check if all of them are empty ?
bool checkMove(StackLinked<int>&,StackLinked<int>&);	Boolean		This method take two stacks (A,B) Respectively and return true if the top of A is lesser than top of B.
Drawing algorithm			
void drawingTowers(StackLinked<int>&,StackLinked<int>&,StackLinked<int>&,int);	void	Three Alias objects of type Stack	The main algorithm method that take the the three stacks to draw them , to do that we make a method in the stack implementation class convertToArr To convert each stack into array.
void ascendingOrder(int*&,int);	void	Alias pointer , int	This method take alias pointer and int , to order the array ascending. The incoming array is equivalent to each tower construction which indeed in descending order if we let it as it comes the towers base and peak will be take the place of each other.
void draw(int[],int[],int[],int);	void	Three Arrays of type int	This method take three arrays , and number of disks and draw them at the run time on the screen.
void drawPeaks(int&);	void	Alias int	Draw constant two peaks in each tower to make them user friendly.
void drawBottoms();	void	No arguments	Draw constant base for each tower even if the tower is empty.
void displayStep(int,int);	void	Two int	Write every step in cmd

Abstraction and clean code

- In this code we try to write a code to write a clean code as much as we can :

- Readable : we make the code like a story

anyone can read whatever his mentality.

- Easy to maintain : that lead us to improve

our code not once or twice , but we improve our code

four times and make four versions

of the game the last one is v1.4

- Prevent users from typo: to protect the user from himself ,

to make him enjoy the game ,

and prevent any unexpected crashing of the game even

if the user IQ = 85 or less !

- The beauty of abstraction appeared when

we pass (a pointer to function) as argument to

a method called traverse in the stack implementation to

do any process the user want and traverse

the stack and do this process (read , ...) .

Drawing Algorithm

- This algorithm to draw the towers at the same time in the run time :

we start from top to down and from left to right.

First we have three arrays : towerOne, towerTwo , towerThree each of them represent one tower

So we divided the screen into three towers and 4 spaces :

bigSpace= (screenWidth - (3 * 8)) / 4;

diffSpaceOne = bigSpace - towerOne[index];

diffSpaceTwo = bigSpace - (towerOne[index] + towerTwo[index]);

diffSpaceThree = bigSpace - (towerTwo[index] + towerThree[index]);

diffSpaceFour = bigSpace - towerThree[index];

So to make the view user friendly and the spaces are symmetric between

each tower we imagine that each

line is an equation so

Let bigSpace = B , diffSpaceOne = s1, diffSpaceTwo = s2, diffSpaceThree = s3, diffSpaceFour=s4

towerOne = t1 , towerTwo = t2 , towerThree = t3

So line Equation = B s1 t1[i] "|" t1[i] B s2 t2[i] "|" t2[i] B s3 t3[i] "|" t3[i] B s4

We do for loops goes from zero till the value of each variable.

At the top of each tower we draw a constant two peaks , and under every tower we draw a constant base.

Best lines of the code

```
void Solution::draw(int towerOne[],int towerTwo[],int towerThree[],int diskNumbers)
{
    base=diskNumbers*2,
    bigSpace=(screenWidth-(3*8))/4;

    ascendingOrder(towerOne,diskNumbers);
    ascendingOrder(towerTwo,diskNumbers);
    ascendingOrder(towerThree,diskNumbers);

    drawPeaks(bigSpace);
    drawPeaks(bigSpace);

    /*
    line equation
        bigSpace + diffSpaceOne + towerOne[index] + "|" + towerOne[index] +
        bigSpace + diffSpaceTwo + towerTwo[index] + "|" + towerTwo[index] +
        bigSpace + diffSpaceThree + towerThree[index] + "|" + towerThree[index] +
        bigSpace + diffSpaceFour + newLine
    */

    for(index=0;index<diskNumbers;index++)
    {
        diffSpaceOne=bigSpace-towerOne[index];
        diffSpaceTwo=bigSpace-(towerOne[index]+towerTwo[index]);
        diffSpaceThree=bigSpace-(towerTwo[index]+towerThree[index]);
        diffSpaceFour=bigSpace-towerThree[index];

        if(towerOne[index]==0&&towerTwo[index]==0&&towerThree[index]==0)
        {
            drawPeaks(bigSpace);
            continue;
        }

        for(counter=0;counter<diffSpaceOne;counter++)        cout<<" ";
        for(counter=0;counter<towerOne[index];counter++)     cout<<"*";      cout<<"|";
        for(counter=0;counter<towerOne[index];counter++)     cout<<"*";
        for(counter=0;counter<diffSpaceTwo;counter++)        cout<<" "
        for(counter=0;counter<towerTwo[index];counter++)     cout<<"*";      cout<<"|";
        for(counter=0;counter<towerTwo[index];counter++)     cout<<"*";
        for(counter=0;counter<diffSpaceThree;counter++)      cout<<" ";
        for(counter=0;counter<towerThree[index];counter++)   cout<<"*";      cout<<"|";
        for(counter=0;counter<towerThree[index];counter++)   cout<<"*";
        for(counter=0;counter<diffSpaceFour;counter++)       cout<<" ";
        cout<<endl;
    }
    drawBottoms();
}
```

```
//this section is responsible for algorithms of solving
/*********************************************************************************************/
void Solution::fromAToB(StackLinked<int> &t1,StackLinked<int> &t2)
{
    retOne=0, retTwo=0;
    t1.getTop(retOne);
    t2.getTop(retTwo);

    if(checkMove(t1,t2))
    {
        t1.pop(retOne);
        t2.push(retOne);
        fromTower=1;
        toTower=2;
    }

    else if(checkMove(t2,t1))
    {
        t2.pop(retTwo);
        t1.push(retTwo);
        fromTower=2;
        toTower=1;
    }
}
```

```
//this section is responsible for checking rules and end of game
/*********************************************************************************************/
bool Solution::checkMove(StackLinked<int> &t1,StackLinked<int> &t2) //return true if movement valid
{
    //topOfT1 represent pop element
    //topOfT2 represent push element
    topOfT1=0, topOfT2=0;
    t1.getTop(topOfT1);
    t2.getTop(topOfT2);

    //special cases
    if(topOfT1==0) return 0; //you can't push zero
    else if(topOfT1>=1&&topOfT2==0) return 1; //push large disk over small disk

    else if(topOfT1<topOfT2) return 1; //element (pop) less than element that will (push) over it
    else if(topOfT1>=topOfT2) return 0;
}
```