BSc. Artificial Intelligence & Data Science
Level 04

CM 1601
PROGRAMMING FUNDAMENTALS

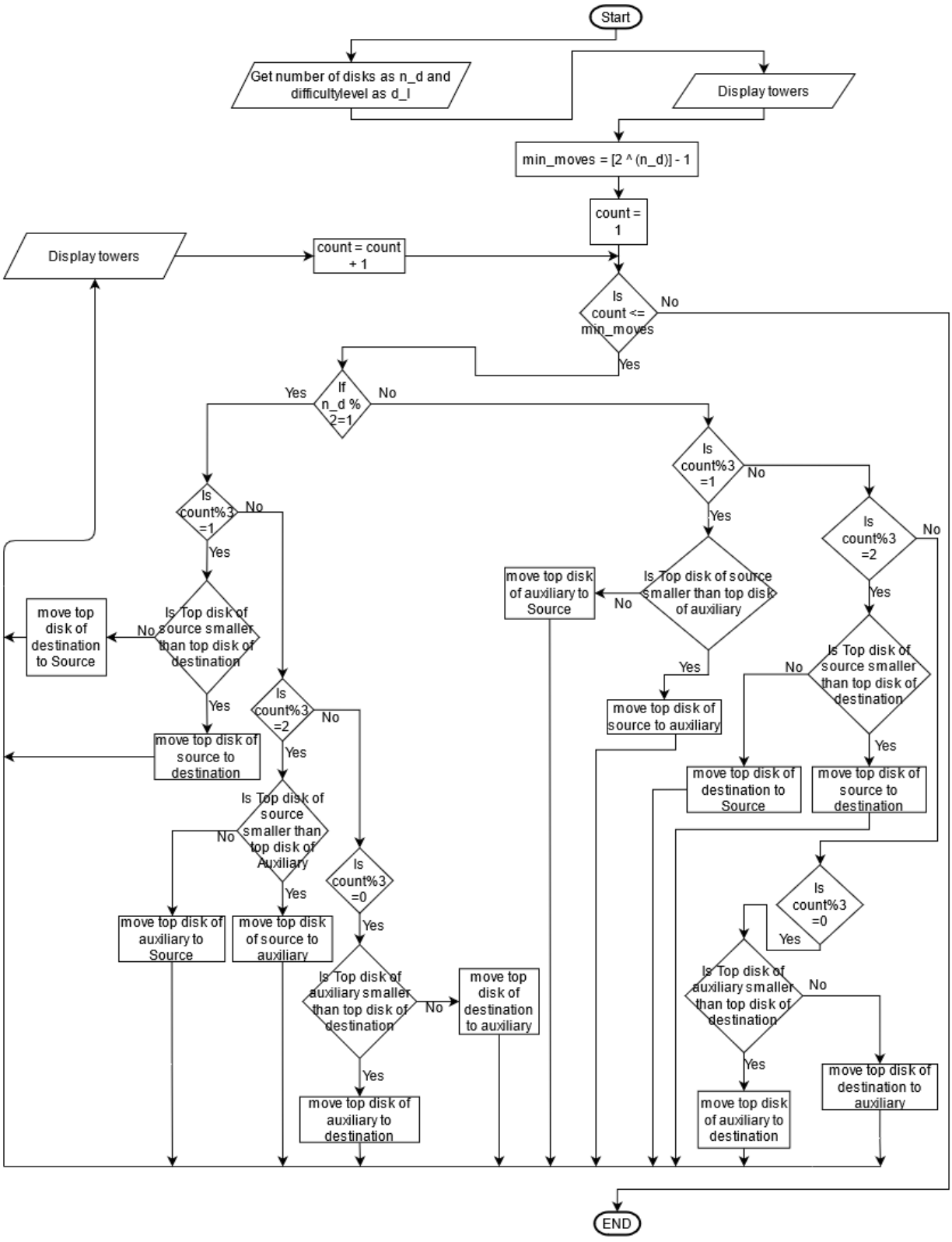
TOWER OF HANOI

COURSEWORK-I REPORT

Thilina Dilanka Jaysuriya
IIT ID: 20200994
RGU ID: 2017922

Table of Contents

Flow Chart


Start
Get number of disks as n_d and difficultylevel as d_l
Display towers
min_moves = [2 ^ (n_d)] - 1
count = 1
Display towers
count = count + 1
Is count <= min_moves
No
Yes
If n_d % 2=1
Yes
No
Is count%3 =1
No
Yes
move top disk of destination to Source
Is Top disk of source smaller than top disk of destination
No
Yes
move top disk of source to destination
Is count%3 =2
No
Yes
Is Top disk of source smaller than top disk of Auxiliary
No
Yes
move top disk of auxiliary to Source
move top disk of source to auxiliary
Is count%3 =0
Yes
Is Top disk of auxiliary smaller than top disk of destination
No
Yes
move top disk of destination to auxiliary
move top disk of auxiliary to destination
Is count%3 =1
No
Yes
move top disk of auxiliary to Source
Is Top disk of source smaller than top disk of auxiliary
No
Yes
move top disk of source to auxiliary
Is count%3 =2
No
Yes
Is Top disk of source smaller than top disk of destination
No
Yes
move top disk of destination to Source
move top disk of source to destination
Is count%3 =0
Yes
Is Top disk of auxiliary smaller than top disk of destination
No
Yes
move top disk of auxiliary to destination
move top disk of destination to auxiliary
END

Source Code

```
import pandas as pnds
objects = ["|       ", "=       ", "===      ", "=====     ", "=======    ",
"=========   ", "==========  ", "============ ", "==============="]
ps = list() #To store players every steps
i = 1
total_moves = 0
store = []  #To store players current step.Used to store in ps list
nff, ffd, err, emc = 0, 0, 0, 0  #nff for a lock to fast forward feature, ffd
to inform the program that the user has
# fast-forwarded err to determine when to lock fast forward


def towers():    #for print towers. aa,ab,...ah represents places of
1st(source) rod.
    # ba,bb,..bh and ca,cb,...ch are for other rods.  aa is the top place,
and ah is the bottom place
    data1 = [[objects[aa], objects[ba], objects[ca]]]
    data2 = [[objects[ab], objects[bb], objects[cb]]]
    data3 = [[objects[ac], objects[bc], objects[cc]]]
    data4 = [[objects[ad], objects[bd], objects[cd]]]
    data5 = [[objects[ae], objects[be], objects[ce]]]
    data6 = [[objects[af], objects[bf], objects[cf]]]
    data7 = [[objects[ag], objects[bg], objects[cg]]]
    data8 = [[objects[ah], objects[bh], objects[ch]]]
    df1 = pnds.DataFrame(data1, columns=["                ", "
", "                "])
    df2 = pnds.DataFrame(data2, columns=["                ", "
", "                "])
    df3 = pnds.DataFrame(data3, columns=["                ", "
", "                "])
    df4 = pnds.DataFrame(data4, columns=["                ", "
", "                "])
    df5 = pnds.DataFrame(data5, columns=["                ", "
", "                "])
    df6 = pnds.DataFrame(data6, columns=["                ", "
", "                "])
    df7 = pnds.DataFrame(data7, columns=["                ", "
", "                "])
    df8 = pnds.DataFrame(data8, columns=["                ", "
", "                "])
    #Below part of this function is to show only relevant places of rods.
    # otherwise, there are more non-accessible spaces for disks when using
the game for lower disk amounts.
    if disk_level == 3:
        print(df6)
        print(df7)
        print(df8)
    if disk_level == 4:
        print(df5)
        print(df6)
```

```
        print(df7)
        print(df8)
    if disk_level == 5:
        print(df4)
        print(df5)
        print(df6)
        print(df7)
        print(df8)
    if disk_level == 6:
        print(df3)
        print(df4)
        print(df5)
        print(df6)
        print(df7)
        print(df8)
    if disk_level == 7:
        print(df2)
        print(df3)
        print(df4)
        print(df5)
        print(df6)
        print(df7)
        print(df8)
    if disk_level == 8:
        print(df1)
        print(df2)
        print(df3)
        print(df4)
        print(df5)
        print(df6)
        print(df7)
        print(df8)
    print("total moves: ", total_moves)


def algo():     #tower of hanoi algorithm without move validation
    all_guesses = list()
    each_guess = []
    if disk_level % 2 == 0:
        for each_move in range(min_moves):
            if (each_move + 1) % 3 == 1:
                each_guess = [1,2]
            elif (each_move + 1) % 3 == 2:
                each_guess = [1,3]
            elif (each_move + 1) % 3 == 0:
                each_guess = [2,3]
            all_guesses.append(each_guess)
    elif disk_level % 2 == 1:
        for each_move in range(min_moves):
            if (each_move + 1) % 3 == 1:
                each_guess = [1,3]
            elif (each_move + 1) % 3 == 2:
                each_guess = [1,2]
            elif (each_move + 1) % 3 == 0:
                each_guess = [2,3]
            all_guesses.append(each_guess)
    return all_guesses
```

```
def change1(rr):  #used to make a free space after a disk is moved from 1st
rod
    global aa, ab, ac, ad, ae, af, ag, ah
    if rr == aa:
        aa = 0
    elif rr == ab:
        ab = 0
    elif rr == ac:
        ac = 0
    elif rr == ad:
        ad = 0
    elif rr == ae:
        ae = 0
    elif rr == af:
        af = 0
    elif rr == ag:
        ag = 0
    elif rr == ah:
        ah = 0


def change2(rr):   #used to make a free space after a disk is moved from 2nd
rod
    global ba, bb, bc, bd, be, bf, bg, bh
    if rr == ba:
        ba = 0
    elif rr == bb:
        bb = 0
    elif rr == bc:
        bc = 0
    elif rr == bd:
        bd = 0
    elif rr == be:
        be = 0
    elif rr == bf:
        bf = 0
    elif rr == bg:
        bg = 0
    elif rr == bh:
        bh = 0


def change3(rr):   #used to make a free space after a disk is moved from 3rd
rod
    global ca, cb, cc, cd, ce, cf, cg, ch
    if rr == ca:
        ca = 0
    elif rr == cb:
        cb = 0
    elif rr == cc:
        cc = 0
    elif rr == cd:
        cd = 0
    elif rr == ce:
        ce = 0
```

```
    elif rr == cf:
        cf = 0
    elif rr == cg:
        cg = 0
    elif rr == ch:
        ch = 0


def check_places(mm, nn):  #take the top disk from the selected rod and move
it to the selected rod
    #check for free spaces in the selected rod and validate the move.
    # error messages are included for invalid moves
    global aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf, bg, bh,
ca, cb, cc, cd, ce, cf, cg, ch, error, emc
    if mm == 1:
        lia = [aa, ab, ac, ad, ae, af, ag, ah]
        for ai in lia:
            if ai != 0:
                if nn == 2:
                    if bh == 0:
                        bh = ai
                        change1(ai)
                        break
                    elif bg == 0 and bh > ai:
                        bg = ai
                        change1(ai)
                        break
                    elif bf == 0 and bg > ai:
                        bf = ai
                        change1(ai)
                        break
                    elif be == 0 and bf > ai:
                        be = ai
                        change1(ai)
                        break
                    elif bd == 0 and be > ai:
                        bd = ai
                        change1(ai)
                        break
                    elif bc == 0 and bd > ai:
                        bc = ai
                        change1(ai)
                        break
                    elif bb == 0 and bc > ai:
                        bb = ai
                        change1(ai)
                        break
                    elif ba == 0 and bb > ai:
                        ba = ai
                        change1(ai)
                        break
                    else:
                        emc = 1
                        error = 1
                        break
                elif nn == 3:
                    if ch == 0:
```

```
                        ch = ai
                        change1(ai)
                        break
                    elif cg == 0 and ch > ai:
                        cg = ai
                        change1(ai)
                        break
                    elif cf == 0 and cg > ai:
                        cf = ai
                        change1(ai)
                        break
                    elif ce == 0 and cf > ai:
                        ce = ai
                        change1(ai)
                        break
                    elif cd == 0 and ce > ai:
                        cd = ai
                        change1(ai)
                        break
                    elif cc == 0 and cd > ai:
                        cc = ai
                        change1(ai)
                        break
                    elif cb == 0 and cc > ai:
                        cb = ai
                        change1(ai)
                        break
                    elif ca == 0 and cb > ai:
                        ca = ai
                        change1(ai)
                        break
                    else:
                        emc =1
                        error = 1
                        break
                else:
                    emc =2
                    error = 1
                    break
            elif aa == 0 and ab == 0 and ac == 0 and ad == 0 and ae == 0 and
af == 0 and ag == 0 and ah == 0:
                emc =3
                error = 1
                break

    elif mm == 2:
        lib = [ba, bb, bc, bd, be, bf, bg, bh]
        for bi in lib:
            if bi != 0:
                if nn == 1:
                    if ah == 0:
                        ah = bi
                        change2(bi)
                        break
                    elif ag == 0 and ah > bi:
                        ag = bi
                        change2(bi)
```

```
                    break
                elif af == 0 and ag > bi:
                    af = bi
                    change2(bi)
                    break
                elif ae == 0 and af > bi:
                    ae = bi
                    change2(bi)
                    break
                elif ad == 0 and ae > bi:
                    ad = bi
                    change2(bi)
                    break
                elif ac == 0 and ad > bi:
                    ac = bi
                    change2(bi)
                    break
                elif ab == 0 and ac > bi:
                    ab = bi
                    change2(bi)
                    break
                elif aa == 0 and ab > bi:
                    aa = bi
                    change2(bi)
                    break

                else:
                    emc = 1
                    error = 1
                    break
            elif nn == 3:
                if ch == 0:
                    ch = bi
                    change2(bi)
                    break
                elif cg == 0 and ch > bi:
                    cg = bi
                    change2(bi)
                    break
                elif cf == 0 and cg > bi:
                    cf = bi
                    change2(bi)
                    break
                elif ce == 0 and cf > bi:
                    ce = bi
                    change2(bi)
                    break
                elif cd == 0 and ce > bi:
                    cd = bi
                    change2(bi)
                    break
                elif cc == 0 and cd > bi:
                    cc = bi
                    change2(bi)
                    break
                elif cb == 0 and cc > bi:
                    cb = bi
```

```
                        change2(bi)
                        break
                    elif ca == 0 and cb > bi:
                        ca = bi
                        change2(bi)
                        break
                    else:
                        emc = 1
                        error = 1
                        break
                else:
                    emc = 2
                    error = 1
                    break
            elif ba == 0 and bb == 0 and bc == 0 and bd == 0 and be == 0 and
bf == 0 and bg == 0 and bh == 0:
                emc = 3
                error = 1
                break

    elif mm == 3:
        lic = [ca, cb, cc, cd, ce, cf, cg, ch]
        for ci in lic:
            if ci != 0:
                if nn == 1:
                    if ah == 0:
                        ah = ci
                        change3(ci)
                        break
                    elif ag == 0 and ah > ci:
                        ag = ci
                        change3(ci)
                        break
                    elif af == 0 and ag > ci:
                        af = ci
                        change3(ci)
                        break
                    elif ae == 0 and af > ci:
                        ae = ci
                        change3(ci)
                        break
                    elif ad == 0 and ae > ci:
                        ad = ci
                        change3(ci)
                        break
                    elif ac == 0 and ad > ci:
                        ac = ci
                        change3(ci)
                        break
                    elif ab == 0 and ac > ci:
                        ab = ci
                        change3(ci)
                        break
                    elif aa == 0 and ab > ci:
                        aa = ci
                        change3(ci)
                        break
```

```
                    else:
                        emc = 1
                        error = 1
                        break
                elif nn == 2:
                    if bh == 0:
                        bh = ci
                        change3(ci)
                        break
                    elif bg == 0 and bh > ci:
                        bg = ci
                        change3(ci)
                        break
                    elif bf == 0 and bg > ci:
                        bf = ci
                        change3(ci)
                        break
                    elif be == 0 and bf > ci:
                        be = ci
                        change3(ci)
                        break
                    elif bd == 0 and be > ci:
                        bd = ci
                        change3(ci)
                        break
                    elif bc == 0 and bd > ci:
                        bc = ci
                        change3(ci)
                        break
                    elif bb == 0 and bc > ci:
                        bb = ci
                        change3(ci)
                        break
                    elif ba == 0 and bb > ci:
                        ba = ci
                        change3(ci)
                        break
                    else:
                        emc = 1
                        error = 1
                        break
                else:
                    emc = 2
                    error = 1
                    break
            elif ca == 0 and cb == 0 and cc == 0 and cd == 0 and ce == 0 and
cf == 0 and cg == 0 and ch == 0:
                emc = 3
                error = 1
                break

    else:
        emc = 4
        error = 1


def game():  #Take player's inputs, store those, and send error messages
```

```
properly,
    global aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf, bg, bh,
ca, cb, cc, cd, ce, cf, cg, ch, i, \
        total_moves, store, error, dif_level, disk_level, ffd, err, nff,
loop, emc
    while i == 1:
        if error == 0:  #To stop store the same data when a wrong input has
entered
            store = [aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf,
bg, bh, ca, cb, cc, cd, ce, cf, cg, ch]
            ps.append(store)
        elif error == 1:
            error = 0
        ukey= 0
        k = input("To move a disk, Input the rod number [1,2,3] to select
from which rod or\n"
                  "input f to fast forward or input u to undo: ")
        try:         #to prevent traceback if the player input anything other
than integer
            m = int(k)
            n = int(input("To Which rod? [1,2,3]: "))
        except:
            if (k =="f" or k == "F"):
                if nff == 0:      #nff to lock and unlock fast forward option
                    fast_forward(total_moves)
                    if loop == 1:
                        loop = 0
                        continue
                else:
                    print("you have made",err,"mistake\mistakes
previously.There for you can\'t use the fast "
                                                "forward feature until game
ends")
            elif (k == "u" or k == "U") and total_moves > 0:
                ukey = 1
            else:
                print("Wrong Input!")
                towers()
                error = 1
                continue
        if ukey == 0:
            if ffd == 0:
                check_places(m, n)
                if emc == 1:
                    print("Can\'t put a bigger disk on a smaller disk")
                    towers()
                elif emc == 2:
                    print("Nothing moved")
                    towers()
                elif emc == 3:
                    print("Nothing to move!")
                    towers()
                elif emc == 4:
                    print("Wrong Input!")
                    towers()

                if error == 1:
```

```
                    emc = 0
                    continue

                total_moves += 1
        #check whether the player's move correct or not in the expert level
                checker = [aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd,
be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch]
                if (checker == a_m[total_moves]):
                    if dif_level == "e":
                        print("Excellent Move!")
                else:
                    if dif_level == "e":
                        print("Oops! Wrong move!")
                    err += 1
                    if err == 1:
                        nff = 1

            towers()

            replay = 0
    #check whether the player has won or lost the game
            if ((disk_level == 3 and cf == 1) or (disk_level == 4 and ce ==
1) or (disk_level == 5 and cd == 1) or
                    (disk_level == 6 and cc == 1) or (disk_level == 7 and cb
== 1) or (disk_level == 8 and ca == 1)):
                print("You won!")
                replay = 1
            elif dif_level == "e":
                if total_moves == min_moves:
                    print("Oops! You could not make it in minimum moves!")
                    undo()
                    if error == 1:
                        continue
                    else:
                        replay = 1
            elif dif_level == "i":
                if total_moves == min_moves + 3:
                    print("Oops! You could not make it in minimum moves + 3
moves!")
                    undo()
                    if error == 1:
                        continue
                    else:
                        replay = 1
            if replay == 1:
                while True:
                    print("Select an option below and input the
number\n\t\t1.play the same level again\n\t\t2.Exit to Main Menu\n\t\t3.Exit
the game")
                    try:
                        i = int(input("Select an option and enter the number:
"))
                    except:
                        continue
                    if i == 3:
                        exit()
                    break
```

```
                while i != 1 and i != 2 and i != 3:
                    print("Oops! wrong input")
                    print(
                        "Select an option below and input the
number\n\t\t1.play the same level again\n\t\t2.Exit to Main Menu\n\t\t3.Exit
the game")
                    i = input("Select an option and enter the number: ")
                #resetting values to prepare the program for another game if
the user wants to play again
                aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
                be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = ps[0][0],
ps[0][1], ps[0][2], ps[0][3], ps[0][4], \
                                                                 ps[0][5],
ps[0][6], ps[0][7], ps[0][8], ps[0][9], \
                                                                 ps[0][10],
ps[0][11], ps[0][12], ps[0][13], ps[0][14], \
                                                                 ps[0][15],
ps[0][16], ps[0][17], ps[0][18], ps[0][19], \
                                                                 ps[0][20],
ps[0][21], ps[0][22], ps[0][23]
                total_moves = 0
                err, nff, ffd = 0, 0, 0
                ps.clear()
                if i == 1:
                    towers()
                    continue
                elif i == 2:
                    i = 1
                    break
        if ffd != 1 and ukey == 1:
            undo()
        ffd = 0


def fast_forward(t):   #fast forward if player is stuck
    global aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf, bg, bh,
ca, cb, cc, cd, ce, cf, cg, ch, \
        total_moves, error, ffd, ps, min_moves,loop
    ffd = 0
    ff = input("Do you want to fast forward the next move? [y/n]: ")
    qq = 0
    for q in ps:   #check whether the player has made a wrong move previously
        if ps[qq] != a_m[qq]:
            error = 1
            break
        qq += 1
    if err == 1 and ff == "y":
        print("You have made a mistake in previous move. There for you can't
fast forward unless you undo now.")
        loop = 1
    elif error == 0 and ff == "y":
        #assigning values to go to the next step
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = a_m[t+1][0],
a_m[t+1][1], a_m[t+1][2], a_m[t+1][3], \
                                                        a_m[t+1][4],
a_m[t+1][5], a_m[t+1][6], a_m[t+1][7], \
```

```
                                                                a_m[t+1][8],
a_m[t+1][9], a_m[t+1][10], a_m[t+1][11], \
                                                                a_m[t+1][12],
a_m[t+1][13], a_m[t+1][14], a_m[t+1][15], \
                                                                a_m[t+1][16],
a_m[t+1][17], a_m[t+1][18], a_m[t+1][19], \
                                                                a_m[t+1][20],
a_m[t+1][21], a_m[t+1][22], a_m[t+1][23]
        total_moves += 1
        ffd = 1
    elif error == 1:
        error = 0
        loop = 1
    else:
        loop = 1


def undo():   #undo a move or some moves if the player needs to. used the ps
list to get back users previous moves
    global aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf, bg, bh,
ca, cb, cc, cd, ce, cf, cg, ch, total_moves,\
        error, err, nff
    undo = input("Do you want to undo the last move? [y/n]: ")
    while True:
        if total_moves > 0 and undo == "y":
            tm = total_moves - 1
            #assigning values to undo moves
            aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd,\
            be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = ps[tm][0],
ps[tm][1], ps[tm][2], ps[tm][3], ps[tm][4],\
                                                                ps[tm][5],
ps[tm][6], ps[tm][7], ps[tm][8], ps[tm][9],\
                                                                ps[tm][10],
ps[tm][11], ps[tm][12], ps[tm][13],\
                                                                ps[tm][14],
ps[tm][15], ps[tm][16], ps[tm][17],\
                                                                ps[tm][18],
ps[tm][19], ps[tm][20], ps[tm][21],\
                                                                ps[tm][22],
ps[tm][23]
            total_moves -= 1
            towers()
            del ps[total_moves]
            if tm != 0:
                del ps[tm]
            if err == 1:
                err = 0
                nff = 0
            break
        elif total_moves == 0 and undo == "y":
            print("Nothing to undo. You have already come to the Beginning.")
            towers()
            break
        elif undo == "n":
            store = [aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf,
bg, bh, ca, cb, cc, cd, ce, cf, cg, ch]
            ps.append(store)
```

```
            error = 1
            if err == 1:
                nff = 1
            break
        else:
            print("Oops! wrong input")
            towers()
            undo = input("Do you want to undo the last move? [y/n]: ")


def main_menu():
    while True:
        print("Main Menu \n\t\t1. Play the game \n\t\t2. Automatic play
\n\t\t3. Exit the game\n\n")
        try:
            choice = int(input("Select an option and enter the number: "))
        except:
            print("Oops! Wrong Input.")
            continue
        if choice != 1 and choice != 2 and choice != 3:
            print("Oops! Wrong Input.")
            continue
        else:
            break
    return choice

print("TOWER OF HANOI \n  \n  \nRules and Instructions: \n \n  \t\t 1. Only
one disk can be moved at a time \n  \t\t "
      "2. You cannot Place a bigger disk on a smaller disk \n  \t\t 3. Your
objective is to take the pile from "
      "the source rod(the left rod) to destination rod(the right rod) \n
\t\t 4. input the number of the desired "
      "option in main menu \n  \t\t 5.when you are selecting a rod, input the
number of the rod as mentioned below."
      "\n  \t\t\t 1 for source rod(left rod) & 2 for auxiliary rod(middle
rod) & 3 for destination rod(right rod)."
      "\n\n\n")
choice = main_menu()
if choice == 3:
    exit()
while True:
    try:
        disk_level = int(input("Enter number of disks to play with [3 to 8]:
"))
        if disk_level < 3 or disk_level > 8:
            print("please select 3 to 8 disks(including 3 and 8)")
            continue
    except:
        continue
    if choice == 1:
        dif_level = input("Enter the difficulty level [Novice:n,
Intermediate:i, Expert:e]: ")
        if dif_level != "e" and dif_level != "i" and dif_level != "n":
            continue
    min_moves = (2**disk_level) - 1
    error, loop = 0, 0
    if disk_level == 3:
```

```
        #assigning initial values for three disks
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd,\
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = 0, 0, 0, 0, 0, 1, 2,
3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\
                                                        0, 0, 0, 0
    elif disk_level == 4:
        # assigning initial values for four disks
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = 0, 0, 0, 0, 1, 2, 3,
4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
                                                        0, 0, 0, 0
    elif disk_level == 5:
        # assigning initial values for five disks
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = 0, 0, 0, 1, 2, 3, 4,
5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
                                                        0, 0, 0, 0
    elif disk_level == 6:
        # assigning initial values for six disks
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = 0, 0, 1, 2, 3, 4, 5,
6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
                                                        0, 0, 0, 0
    elif disk_level == 7:
        # assigning initial values for seven disks
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = 0, 1, 2, 3, 4, 5, 6,
7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
                                                        0, 0, 0, 0
    elif disk_level == 8:
        # assigning initial values for eight disks
        aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, \
        be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch = 1, 2, 3, 4, 5, 6, 7,
8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
                                                        0, 0, 0, 0
    reset = [aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf, bg, bh,
ca, cb, cc, cd, ce, cf, cg, ch]  #get the initial values
    all_guesses = algo()
    count = 0
    a_m = list()
    a_m.append(reset)  #store initial values to the list, which made for
store auto generated steps.
    for each_guess in all_guesses:   #checking and validating auto generated
steps
        check_places(all_guesses[count][0], all_guesses[count][1])
        if error == 1:
            check_places(all_guesses[count][1], all_guesses[count][0])
            error = 0
        emc =0
        auto_e_m = [aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf,
bg, bh, ca, cb, cc, cd, ce, cf, cg, ch]
        a_m.append(auto_e_m)  #store auto generated steps
        count += 1
    aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd, be, bf, bg, bh, ca, cb,
cc, cd, ce, cf, cg, ch = reset[0], reset[1], reset[2], reset[3], reset[4],
reset[5], reset[6], reset[7], reset[8], reset[9], reset[10], reset[11],
reset[12], reset[13], reset[14], reset[15], reset[16], reset[17], reset[18],
```

```
reset[19], reset[20], reset[21], reset[22], reset[23]
    decide = "loop it"
    if choice == 1:  #manual play
        towers()
        game()
        choice = main_menu()
        if choice == 3:
            exit()
    if choice == 2:  #auto play
        counting = 0
        for n in range(min_moves + 1):
            total_moves = counting
            aa, ab, ac, ad, ae, af, ag, ah, ba, bb, bc, bd,\
            be, bf, bg, bh, ca, cb, cc, cd, ce, cf, cg, ch =
a_m[counting][0], a_m[counting][1], a_m[counting][2], \

a_m[counting][3], a_m[counting][4], a_m[counting][5], \

a_m[counting][6], a_m[counting][7], a_m[counting][8], \

a_m[counting][9], a_m[counting][10], a_m[counting][11], \

a_m[counting][12], a_m[counting][13], a_m[counting][14], \

a_m[counting][15], a_m[counting][16], a_m[counting][17], \

a_m[counting][18], a_m[counting][19], a_m[counting][20], \

a_m[counting][21], a_m[counting][22], a_m[counting][23]
            towers()
            counting += 1
            next = input("Press Enter key to continue")
        total_moves = 0
        choice = main_menu()
        if choice == 3:
            exit()
```

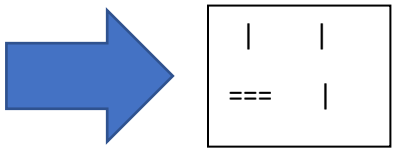
The Logic of Visualizing and Moving Disks

Visualizing

I used data framing to create towers as a table that has rows and columns but no borders. Then insert the list with variable indexes to each place. So each place has a variable value that will determine the disk size.

Eg:

```
Import pandas as pd

objects =["|",”= ”,]

aa, ab = 0, 1

d1 = [[objects[aa]], [objects[aa]]]

d2 = [[objects[ab]], [objects[aa]]]

df1 = pd.DataFrame(d1, columns=["   ", "   "])

df2 = pd.DataFrame(d2, columns=["   ", "   "])

print(df1)

print(df1)
```



By that method, changing variables (aa, ab) will change the output, resulting in changing the disks.

Moving Disks

At first, the program will get two inputs from the player.1st input will determine “from which rod” and 2nd one will determine “to which rod”. Then the program will get variables from the rod relevant to the 1st input and check each variable from top to bottom. If a variable has been found assigned by any integer other than 0, the program will memorize its value.

Then the program will get variables from the rod relevant to the 2nd input, and each variable will be checked from bottom to top. If a variable has been found assigned by 0, then the program will check the place below it.

And if the below place has a lesser value than memorized value, then the program will exchange selected places' variables.

If the below place of the 2nd selected place has a higher value than memorized value, then the program will deliver a proper error message to the player.

Sample Outputs

With Simple Inputs:

```
0          =                ===              =====
total moves:  5
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 2
To Which rod? [1,2,3]: 3
Excellent Move!

0          |                |                |

0          |                |                ===

0          =                |              =====
total moves:  6
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 1
To Which rod? [1,2,3]: 3
Excellent Move!

0          |                |                =

0          |                |                ===

0          |                |              =====
total moves:  7
You won!
Select an option below and input the number
        1.play the same level again
        2.Exit to Main Menu
        3.Exit the game
Select an option and enter the number:
```

Undo Function:

```
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 2
To Which rod? [1,2,3]: 3
Excellent Move!

0          |                |                |

0          |                |               ===

0          =                |              =====
total moves:  6
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: u
Do you want to undo the last move? [y/n]: y

0          |                |                |

0          |                |                |

0          =               ===              =====
total moves:  5
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: u
Do you want to undo the last move? [y/n]: y

0          |                |                |

0          |                =                |

0          |               ===              =====
total moves:  4
To move a disk, Input the rod number [1,2,3] to select from which rod or
```

Fast Forward Function:

```
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: f
Do you want to fast forward the next move? [y/n]: y

0          |                |                |

0          |                |                |

0          =               ===              =====
total moves:  5
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 2
To Which rod? [1,2,3]: 3
Excellent Move!

0          |                |                |

0          |                |                ===

0          =                |              =====
total moves:  6
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: f
Do you want to fast forward the next move? [y/n]: y

0          |                |                =

0          |                |                ===

0          |                |              =====
total moves:  7
You won!
```

The player can choose fast forward at any given timestamp if he/she intend to. Each time the player runs the game, the program generates minimum steps, and as long as the player is going along the correct path, the program allows the player to fast forward steps one by one.

Comparison:

```
Enter number of disks to play with [3 to 8]: 3
Enter the difficulty level [Novice:n, Intermediate:i, Expert:e]: e

0          =                |                |

0         ===               |                |

0        =====              |                |
total moves:  0
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 1
To Which rod? [1,2,3]: 3
Excellent Move!

0          |                |                |

0         ===               |                |

0        =====              |                =
total moves:  1
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 3
To Which rod? [1,2,3]: 2
Oops! Wrong move!

0          |                |                |

0         ===               |                |

0        =====              =                |
total moves:  2
```

As the player inputs, each data processed by those inputs are compared with the data processed by auto-generated minimum steps and gives the relevant feedback.

Automation:

```
Main Menu
        1. Play the game
        2. Automatic play
        3. Exit the game


Select an option and enter the number: 2
Enter number of disks to play with [3 to 8]: 3

0          =                |                |

0         ===               |                |

0        =====              |                |
total moves:  0
Press Enter key to continue

0          |                |                |

0         ===               |                |

0        =====              |                =
total moves:  1
Press Enter key to continue

0          |                |                |

0          |                |                |

0        =====             ===               =
total moves:  2
Press Enter key to continue
```

The player can choose automatic play if wanted. That way, the player can learn about the inputs which cause the minimum steps. This automatic play option is based on the auto-generated steps too. The program reads the data of the minimum moves which have been stored and displays the steps.

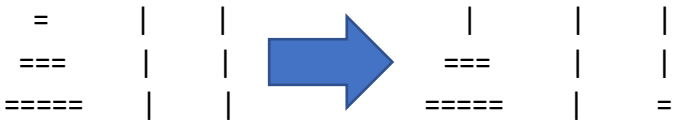
Test Plan

Only one disk can be moved at a time.

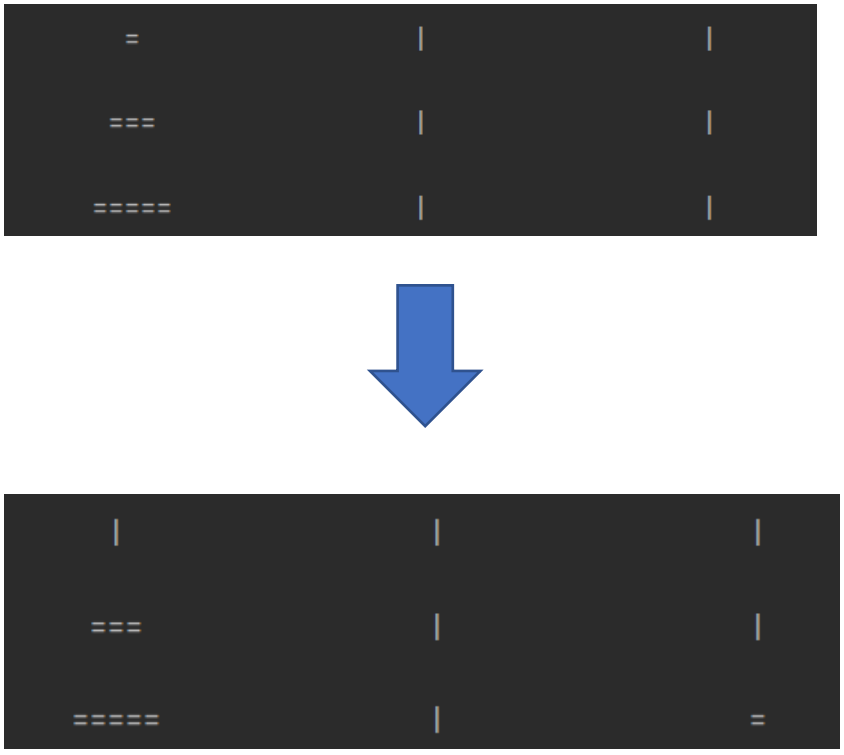
- Input:

```
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 1
To Which rod? [1,2,3]: 3
```

- Expected output:



- Actual output:



Taking a disk from the top of a stack and putting it on the top of another stack.

- Input:

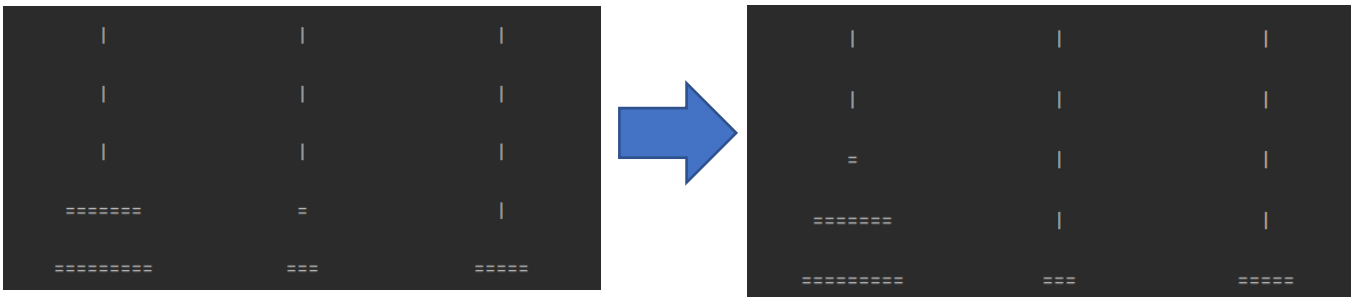
```
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 2
To Which rod? [1,2,3]: 1
```

- Expected output:

```
     |            |          |
     |            |          |
     |            |          |
   =======        =          |
  =========      ===        =====
```

```
     |            |          |
     |            |          |
     =            |          |
   =======        |          |
  =========      ===        =====
```

- Actual output:



Can’t place a disk on the top of a smaller disk

- Input:

```
To move a disk, Input the rod number [1,2,3] to select from which rod or
input f to fast forward or input u to undo: 2
To Which rod? [1,2,3]: 3
```

- Expected output: (a proper error message)
  Can’t put a bigger disk on a smaller disk

- Actual output:

```
      |                   |                   |

      |                   |                   |

     =====               ===                  =
```

```
From which rod? [1,2,3]2
To Which rod? [1,2,3]3
Can't put a bigger disk on a smaller disk

0          |                   |                   |

0          |                   |                   |

0        =====               ===                  =
```

Problems and Difficulties

Problem:- When converting the inputs into integers, The player might enter a string or float or some other data which are not integers by accident. Which brings a traceback.

Code:

```
while True:
    disk_level = int(input("Enter number of disks to play with [3 to 8]: "))
    if disk_level < 3 or disk_level > 8:
        print("please select 3 to 8 disks(including 3 and 8)")
        continue
```

Output:

```
Enter number of disks to play with [3 to 8]: e
Traceback (most recent call last):
  File "C:/Users/User/PycharmProjects/pythonProject/TOH.py", line 907, in <module>
    disk_level = int(input("Enter number of disks to play with [3 to 8]: "))
ValueError: invalid literal for int() with base 10: 'e'

Process finished with exit code 1
```

Solution:- Used try-except to prevent that. Conversion is inside the try block, and continue is in the except block. which will prevent the traceback and let the player to input again

Code:

```
while True:
    try:
        disk_level = int(input("Enter number of disks to play with [3 to 8]: "))
        if disk_level < 3 or disk_level > 8:
            print("please select 3 to 8 disks(including 3 and 8)")
            continue
    except:
        continue
```

Output:

```
Select an option and enter the number: 1
Enter number of disks to play with [3 to 8]: e
Enter number of disks to play with [3 to 8]:
```