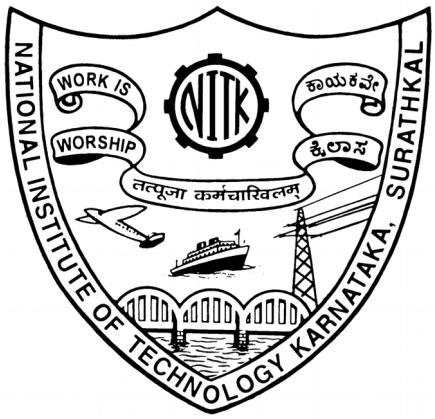
UNIX MINI PROJECT

Course Code: IT202



Submitted to:

Ms.Deepthi
Assistant Lecturer
Department of Information Technology
NITK Surathkal

Ms.Sangeeta
Assistant Lecturer
Department of Information Technology
NITK Surathkal

GVS Tushaar – 15IT117
Akshay U Prabhu – 15IT203
Aniket Kumar – 15IT205
A Vivek Reddy – 15IT109

11th November 2016

Contents:

*(This document consists of **29** pages in total, including both the cover pages.)*

Certificate:

This is to certify that the Btech Mini project entitled "**UNIX MINI PROJECT"** submitted by :

GVS Tushaar (Roll Number: 15IT117)
Akshay U Prabhu (Roll Number: 15IT203)
Aniket Kumar (Roll Number: 15IT205)
A Vivek Reddy (Roll Number: 15IT209)

as the record of the work carried out by then is accepted as the Btech Mini project submission in partial fulfillment of the requirements for mandatory learning course of IT202, Unix Programming and Practice in the Department of Information Techonlogy.

Date: 11th November 2016.
Place: Surathkal.

Signature and Seal

Abstract:

There are two levels in the Game made, Level 1: Classic **Snakes and Ladders Game** using BASH Shell Scripting.

Here the code is divided into three basic parts.

- **Rolling the die** : *$RANDOM* is used to generate the number between 1-6 which functions like rolling.

- **Upgrading Position** : Function *CheckPosition()* is defined for each player which check for the snake and ladder condition on the new space and then upgrade it accordingly.

- **Check winning** : After upgrading we check whether the respective player satisfies the winning condition or not.

The law of the game includes the player to roll the die, then the position of respective player is incremented. The new position then told which space the current player has to be landed on. If landed on a snake the space will move down the board and if landed on ladder, will move up. In order to win, the player must land on any position greater than 64. An alternate implementation is also commented out in the Code part, i.e, if space go over 64 then the imaginary piece will be moved backwards for the remainder of the dice roll. For ease of construction and time, we have commented out that particular part of the code.

Now, the second level is for the player who wins it all in the first level and goes on to the Classic **Snake Game** using BASH Shell Scripting.

Here code is implemented using Six different functions, namely:

- **Build Wall** : Concerns with wall building.
- **Draw Apple** : Function to draw an Apple/ Score point, here.
- **Position Apple** : $RANDOM is used here to randomly decide the Apple's position.
- **Move Snake** : Movements of the Snake are tracked by this.
- **Update Score** : Function to keep updating the score.
- **Game Over** : Function to end the game if the Snake bites itself/ hits the wall.

The law of the game includes the snake to move around and collect apples (or) more precisely, increase the game score while the size of the snake keeps increasing with increasing score. With randomly generated apples in the stipulated amount of space, the size of the snake keeps increasing as and when it eats an apple until the snake bites itself or collides with the wall, then the game terminates, and the final updated score is returned. The code related to this classic snake game will be categorised into functions (as mentioned above) to perform the very same.

Introduction:

The Game aims at implementing ***two*** levels of which the first part is the Classic Snakes and Ladders game i.e, a Multi-Player game while the next level is the Snakes game where in the implementation is divided into functions.

The first game uses a ***newPosition*** function that implements the final position as to where the snake lands.

The various concepts utilized in the making of the game include, ***Arrays***, ***Functions*** ***$RANDOM***, ***Process ID's*** and related shell variables to implement the same in Bash shell scripting.

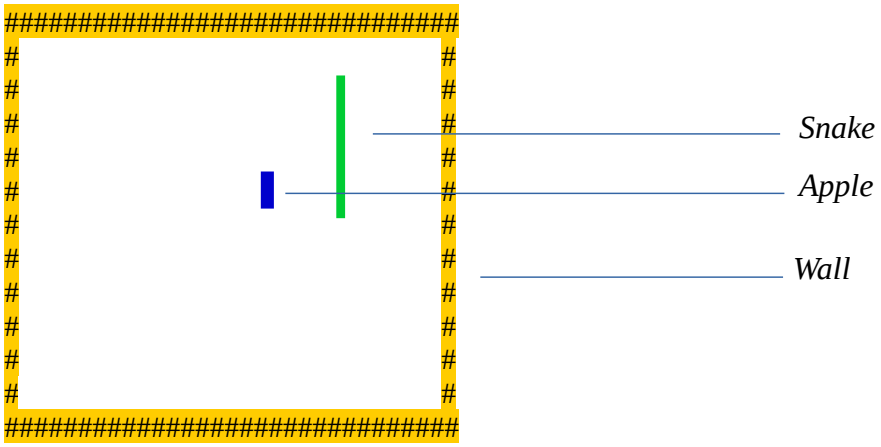
The expected results from the above concepts would include:

$RANDOM is used to generate the number between 1-6 which functions like rolling. Function *CheckPosition()* is defined for each player which check for the snake and ladder condition on the new space and then upgrade it accordingly. After upgrading we check whether the respective player satisfies the winning condition or not.

In the Level-1 of our code, we have implemented colors using tput commands, i.e, tput setab #value and tput set sgr0 to clear it off.

Now, for the implementation of the next part of the game i.e, level-2 of the game, we have mainly focussed on the escape-sequences for the color, design i.e, the *UI* part of the program code while the logical implementation included much dealing of the *Process ID's* and the foreground process `getchar()` ignores `SIGINT` and `SIGQUIT`, and replies to the signal of death `SIG_HEAD` by returning from the function `getchar()`. The background process `game_loop()` traps direction control signals from the keyboard, and self-defined signal `SIG_QUIT` which indicates the press of Q button.

The complete code and the outline of the experiment is given below:



Code (Level 1)

```
#!/bin/bash

declare i=0

function checkPosition()
{
    case $Position in
     4)
         newPosition=35
         ;;

     7)
         newPosition=23
         ;;

     9)
         newPosition=5
         ;;

     14)
         newPosition=43
         ;;

     17)
         newPosition=13
         ;;

     21)
         newPosition=3
         ;;

     24)
         newPosition=58
         ;;

     27)
         newPosition=37
         ;;

     34)
         newPosition=20
         ;;
```

```
    42)
        newPosition=11
        ;;

    46)
        newPosition=53
        ;;

    49)
        newPosition=32
        ;;

    63)
        newPosition=2
        ;;

    *)
        newPosition=$Position
        ;;
    esac
}

function checkPosition1()
{
    case $Position1 in
    4)
        newPosition1=35
        ;;

    7)
        newPosition1=23
        ;;

    9)
        newPosition1=5
        ;;

    14)
        newPosition1=43
        ;;

    17)
        newPosition1=13
        ;;

    21)
```

```
        newPosition1=3
        ;;

    24)
        newPosition1=58
        ;;

    27)
        newPosition1=37
        ;;

    34)
        newPosition1=20
        ;;

    42)
        newPosition1=11
        ;;

    46)
        newPosition1=53
        ;;

    49)
        newPosition1=32
        ;;

    63)
        newPosition1=2
        ;;

    *)
        newPosition1=$Position1
        ;;
   esac
}


tput clear

printf "Would you like to:\n    1)Read the rules\n    2)Play
the game\n\n"
read choice

if ((choice == 1))
then
```

```
    printf "\nThe rules are simple: \n   You press enter to
roll the die\n   You are then told which space you have
landed on\n   If you land on a snake you will move down the
board\n   If you land on a ladder then you will move up it\n
In order to win you must land on 64 exactly\n   If you go
over 64 then your imaginary piece will be moved backwards for
the remainder of your dice roll\n\n Good Luck\n\n"
fi


if ((choice == 1 || choice == 2))
then
    printf "\nWelcome to Snakes and Ladders.\n"
    printf "\n  64  63  62  61  60  59  58  57       1=Start
27=Ladder to 37\n"
    printf "  49  50  51  52  53  54  55  56       4=Ladder
to 35    34=Snake  to 20\n"
    printf "  48  47  46  45  44  43  42  41       7=Ladder
to 23    42=Snake  to 11\n"
    printf "  33  34  35  36  37  38  39  40       9=Snake
to 5     46=Ladder to 53\n"
    printf "  32  31  30  29  28  27  26  25      14=Ladder
to 43    49=Snake  to 32\n"
    printf "  17  18  19  20  21  22  23  24      17=Snake
to 13    63=Snake  to 2\n"
    printf "  16  15  14  13  12  11  10   9      21=Snake
to 3     64=End\n"
    printf "   1   2   3   4   5   6   7   8      24=Ladder
to 58\n"


    while ((Position < 64 && Position1 < 64))
    do
     key=$((i%2))

     if((key==0))
     then
         echo " "
         echo "PLAYER 1"
         echo -e "Please press 'enter' to roll!"
         read ch

         dice=$(echo "$RANDOM%6+1" | bc)

         echo -e "\nYou have rolled a $dice."
```

```
        Position=$((Position+dice))

        #if (($Position > 64))
        #then
        #   above=$((Position-64))
        #   Position=$((64-above))
        #fi

        echo -e "You have landed on space $Position.\n"

        checkPosition

        if ((Position < newPosition))
        then
         echo -e "\nWell done, you have landed on a ladder.
You are now on space $newPosition."
        fi

        if ((Position > newPosition))
        then
         echo -e "\nUnlucky, you have landed on a snake. You
are now on space $newPosition."
        fi

        echo -e "\t\t\t\t\t\t\t\t\t\t\tPlayer 1:
$newPosition."
        Position=$newPosition


    else
        echo " "
        echo "PLAYER 2"
        echo -e "Please press 'enter' to roll!"
        read ch

        dice=$(echo "$RANDOM%6+1" | bc)

        echo -e "\nYou have rolled a $dice."

        Position1=$((Position1+dice))

        #if (($Position1 > 64))
        #then
        #    above=$((Position1-64))
        #    Position1=$((64-above))
        #fi
```

```
        echo -e "You have landed on space $Position1.\n"

        checkPosition1

        if ((Position1 < newPosition1))
        then
         echo -e "Well done, you have landed on a ladder. You
are now on space $newPosition1."
        fi

        if ((Position1 > newPosition1))
        then
         echo -e "Unlucky, you have landed on a snake. You
are now on space $newPosition1."
        fi
        echo -e "\t\t\t\t\t\t\t\t\t\t\tPlayer 2:
$newPosition."
        Position1=$newPosition1

     fi

     i=$((i+1))

    done
    if((Position1 >= 64))
    then
     tput setab 4  #Sets a background.
     echo -e "PLAYER 2 WINS!\c"
     tput sgr0     #Resets the terminal settings.
     echo " "
    elif((Position >= 64))
    then
     tput setab 4  #Sets a background.
     echo -e "PLAYER 1 WINS!\c"
     tput sgr0     #Resets the terminal settings.
     echo ""
    fi

    echo -e "\n\n\nYour Game starts in 5 Seconds!\n\n\n"
    printf "\n\n CONTROLS: 'h': Left.\t'l': Right.\t'k':
Up.\t'j': Down.\n\n"
    for((i = 5; i > 0 ; i--))
    do
     sleep 2
     echo "Starts in $i Seconds!"
```

```
    done

      chmod 755 snake.sh
      ./snake.sh
fi
```

Code (Level 2):

```
#The whole code draws attention towards the shell variables:
#$$, $!, & and others and the working is based on the pid of
#the processes.

#!/bin/bash
IFS=''

declare -i height=30 width=60

# row and column number of head
declare -i head_r head_c tail_r tail_c

declare -i alive
declare -i length
declare body

declare -i direction delta_dir
declare -i score=0

border_color="\E[30;43m"
snake_color="\E[32;42m"
food_color="\E[34;44m"
text_color="\E[31;43m"
no_color="\E[0m"

# signals
SIG_UP=35
SIG_RIGHT=36
SIG_DOWN=37
SIG_LEFT=38
SIG_QUIT=39
SIG_DEAD=40

# direction arrays: 0=up, 1=right, 2=down, 3=left
move_r=([0]=-1 [1]=0 [2]=1 [3]=0)
move_c=([0]=0 [1]=1 [2]=0 [3]=-1)

init_game() {
    clear
    echo -ne "\033[?25l"
    stty -echo
    for ((i=0; i<height; i++)); do
```

```
        for ((j=0; j<width; j++)); do
            eval "arr$i[$j]=' '"
        done
    done
}

move_and_draw() {
    echo -ne "\E[${1};${2}H$3"
}

# print everything in the buffer
draw_board() {
    move_and_draw 1 1 "$border_color+$no_color"
    for ((i=2; i<=width+1; i++)); do
        move_and_draw 1 $i "$border_color-$no_color"
    done
    move_and_draw 1 $((width + 2)) "$border_color+$no_color"
    echo

    for ((i=0; i<height; i++)); do
        move_and_draw $((i+2)) 1 "$border_color|$no_color"
        eval echo -en "\"\${arr$i[*]}\""
        echo -e "$border_color|$no_color"
    done

    move_and_draw $((height+2)) 1 "$border_color+$no_color"
    for ((i=2; i<=width+1; i++)); do
        move_and_draw $((height+2)) $i "$border_color-
$no_color"
    done
    move_and_draw $((height+2)) $((width + 2))
"$border_color+$no_color"
    echo
}

# set the snake's initial state
init_snake() {
    alive=0
    length=10
    direction=0
    delta_dir=-1

    head_r=$((height/2-2))
    head_c=$((width/2))

    body=''
```

```
    for ((i=0; i<length-1; i++)); do
        body="1$body"
    done

    local p=$((${move_r[1]} * (length-1)))
    local q=$((${move_c[1]} * (length-1)))

    tail_r=$((head_r+p))
    tail_c=$((head_c+q))

    eval "arr$head_r[$head_c]=\"${snake_color}o$no_color\""

    prev_r=$head_r
    prev_c=$head_c

    b=$body
    while [ -n "$b" ]; do
        # change in each direction
        local p=${move_r[$(echo $b | grep -o '^[0-3]')]}
        local q=${move_c[$(echo $b | grep -o '^[0-3]')]}

        new_r=$((prev_r+p))
        new_c=$((prev_c+q))

        eval "arr$new_r[$new_c]=\"${snake_color}o$no_color\""

        prev_r=$new_r
        prev_c=$new_c

        b=${b#[0-3]}
    done
}

is_dead() {
    if [ "$1" -lt 0 ] || [ "$1" -ge "$height" ] || \
        [ "$2" -lt 0 ] || [ "$2" -ge "$width" ]; then
        return 0
    fi

    eval "local pos=\${arr$1[$2]}"

    if [ "$pos" == "${snake_color}o$no_color" ]; then
        return 0
    fi

    return 1
```

```
}

give_food() {
    local food_r=$((RANDOM % height))
    local food_c=$((RANDOM % width))
    eval "local pos=\${arr$food_r[$food_c]}"

    while [ "$pos" != ' ' ]; do
        food_r=$((RANDOM % height))
        food_c=$((RANDOM % width))
        eval "pos=\${arr$food_r[$food_c]}"
    done

    eval "arr$food_r[$food_c]=\"$food_color@$no_color\""
}

move_snake() {
    local newhead_r=$((head_r + move_r[direction]))
    local newhead_c=$((head_c + move_c[direction]))

    eval "local pos=\${arr$newhead_r[$newhead_c]}"

    if $(is_dead $newhead_r $newhead_c); then
        alive=1
        return
    fi

    if [ "$pos" == "$food_color@$no_color" ]; then
        length+=1
        eval "arr$newhead_r[$newhead_c]=\"$
{snake_color}o$no_color\""
        body="$(((direction+2)%4))$body"
        head_r=$newhead_r
        head_c=$newhead_c

        score+=1
        give_food;
        return
    fi

    head_r=$newhead_r
    head_c=$newhead_c

    local d=$(echo $body | grep -o '[0-3]$')

    body="$(((direction+2)%4))${body%[0-3]}"
```

```
    eval "arr$tail_r[$tail_c]=' '"
    eval "arr$head_r[$head_c]=\"${snake_color}o$no_color\""

    # new tail
    local p=${move_r[(d+2)%4]}
    local q=${move_c[(d+2)%4]}
    tail_r=$((tail_r+p))
    tail_c=$((tail_c+q))
}

change_dir() {
    if [ $(((direction+2)%4)) -ne $1 ]; then
        direction=$1
    fi
    delta_dir=-1
}

getchar() {
    trap "" SIGINT SIGQUIT
    trap "return;" $SIG_DEAD

    while true; do
        read -s -n 1 key
        case "$key" in
            [qQ]) kill -$SIG_QUIT $game_pid
                  return
                  ;;
            [kK]) kill -$SIG_UP $game_pid
                  ;;
            [lL]) kill -$SIG_RIGHT $game_pid
                  ;;
            [jJ]) kill -$SIG_DOWN $game_pid
                  ;;
            [hH]) kill -$SIG_LEFT $game_pid
                  ;;
       esac
    done
}

game_loop() {
    trap "delta_dir=0;" $SIG_UP
    trap "delta_dir=1;" $SIG_RIGHT
    trap "delta_dir=2;" $SIG_DOWN
    trap "delta_dir=3;" $SIG_LEFT
    trap "exit 1;" $SIG_QUIT
```

```
    while [ "$alive" -eq 0 ]; do
        echo -e "\n\t\t\t\t\t    ${text_color} Your score:
$score $no_color"

        if [ "$delta_dir" -ne -1 ]; then
            change_dir $delta_dir
        fi
        move_snake
        draw_board
        sleep 0.1
    done

    echo -e "\n  ${text_color}Oh, No! You are dead!$no_color"

    # signals the input loop that the snake is dead
    kill -$SIG_DEAD $$  #$$ is the pid of the current process
}

clear_game() {
    stty echo
    echo -e "\033[?25h"
}

init_game
init_snake
give_food
draw_board

game_loop &   #Opens up to the game loop, providing the pid
game_pid=$!   #The pid of the previous process
getchar

clear_game
exit 0
```

Output:

Would you like to:
1)Read the rules
2)Play the game

2

Welcome to Snakes and Ladders.

64 63 62 61 60 59 58 57	1=Start	27=Ladder to 37
49 50 51 52 53 54 55 56	4=Ladder to 35	34=Snake to 20
48 47 46 45 44 43 42 41	7=Ladder to 23	42=Snake to 11
33 34 35 36 37 38 39 40	9=Snake to 5	46=Ladder to 53
32 31 30 29 28 27 26 25	14=Ladder to 43	49=Snake to 32
17 18 19 20 21 22 23 24	17=Snake to 13	63=Snake to 2
16 15 14 13 12 11 10 9	21=Snake to 3	64=End
1 2 3 4 5 6 7 8	24=Ladder to 58	

PLAYER 1
Please press 'enter' to roll!

You have rolled a 1.
You have landed on space 1.

Player 1: 1.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 1.
You have landed on space 1.

Player 2: 1.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 5.
You have landed on space 6.

Player 1: 6.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 5.
You have landed on space 6.

Player 2: 6.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 2.
You have landed on space 8.

Player 1: 8.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 5.
You have landed on space 11.

Player 2: 8.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 6.
You have landed on space 14.

Well done, you have landed on a ladder. You are now on space 43.

Player 1: 43.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 1.
You have landed on space 12.

Player 2: 43.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 6.
You have landed on space 49.

Unlucky, you have landed on a snake. You are now on space 32.

Player 1: 32.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 3.
You have landed on space 15.

Player 2: 32.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 5.
You have landed on space 37.

Player 1: 37.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 6.
You have landed on space 21.

Unlucky, you have landed on a snake. You are now on space 3.

Player 2: 37.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 2.
You have landed on space 39.

Player 1: 39.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 4.
You have landed on space 7.

Well done, you have landed on a ladder. You are now on space 23.

Player 2: 39.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 2.
You have landed on space 41.

Player 1: 41.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 1.
You have landed on space 24.

Well done, you have landed on a ladder. You are now on space 58.

Player 2: 41.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 6.
You have landed on space 47.

Player 1: 47.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 1.
You have landed on space 59.

Player 2: 47.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 6.
You have landed on space 53.

Player 1: 53.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 1.
You have landed on space 60.

Player 2: 53.

PLAYER 1
Please press 'enter' to roll!

You have rolled a 6.
You have landed on space 59.

Player 1: 59.

PLAYER 2
Please press 'enter' to roll!

You have rolled a 5.
You have landed on space 65.

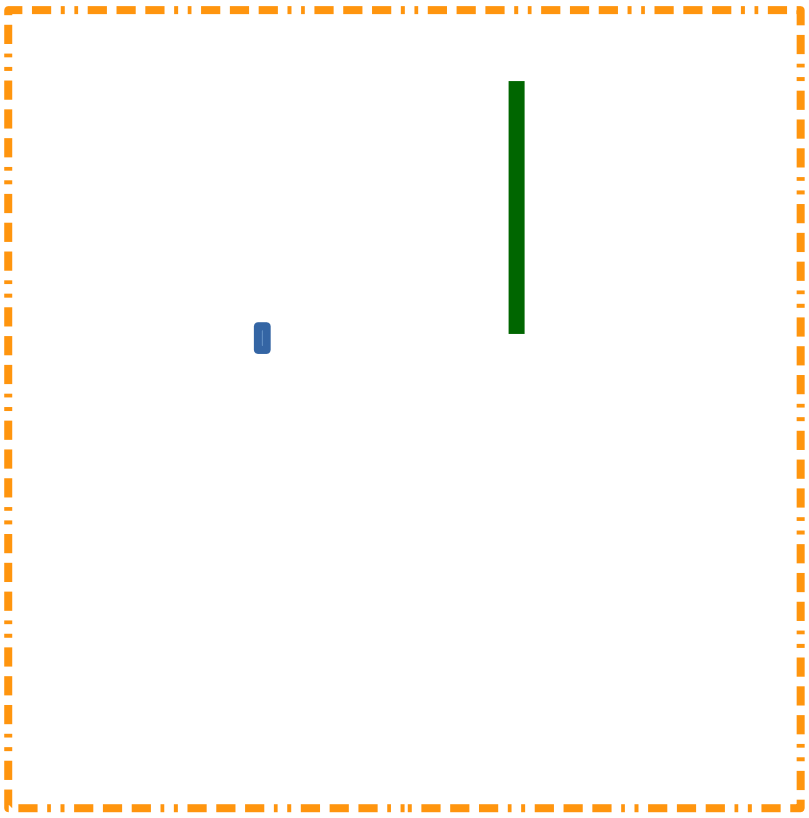
Player 2: 65.

PLAYER 2 WINS!

Your Game starts in 5 Seconds!

CONTROLS: 'h': Left. 'l': Right. 'k': Up. 'j': Down.

Starts in 5 Seconds!
Starts in 4 Seconds!
Starts in 3 Seconds!
Starts in 2 Seconds!
Starts in 1 Seconds!



Oh, No! You are dead! Your score: 10

Algorithm:

Part 1:

Step-1:
Our project basically involves ***SNAKE and LADDER*** game,where players have to roll dice to get random numbers,whose value determine the number of spaces to move ahead. In our program we are generating random numbers using *$RANDOM* command .This command generates numbers more than 6,but in our case we need numbers only from 0-5 ,so we take the modulus of 6 and add 1.

Step-2:
Next we iterate over a loop,and have some number increamenting each time ,we give each players their turn on the basis of whether the number is odd or even.

Step-3:
Next,we add this number (we call it *dice number*) to the position of the respective players.

Step-4:
The tricky part of the program lies in the ladders and snakes of the game,whenever a player lands on a ladder or a snake,his position changes.To implement this we have used a function which uses a switch case tojump to their respective positions.

Step-5:
We have used a switch case to take the new position ,based on the original position.We have used Switch cases on the positions of ladders base and snake's head,and based on its final position we have changed the new position respectively.

Step-6:
If the new position is more than original position,then it means that the player has landed on a ladder and if the new position is less than original position,then it means that the player has landed on a snake .We print the message accordingly and then increment the counter to give other player his turn.

Step-7:
Whoever reaches to the Top *(>= 64)* first will win.

Step-8:
Whoever wins the game will go to the next level.

Step-9:
The next level consists of a snake which uses the Algorithm ***Part-2.***

Part 2:

Step-1:

As we are displaying it on the terminal(console) itself ,we have to hide the cursor on the screen and whatever we have typed ,we have done this by usinig the command:
` **setterm -cursor off** ` and ` **stty -echo** `.

Step-2:

We are initializing the board as a 2d array which contains ‘ ’ (space) everywhere,we are setting the board height and width to 30 and 60 respectively. Now to draw the board we have used a function, which takes x coordinate, y coordinate and the colour of the board and draws one block of the given colour. As we need the entire board we print these blocks of colour one at a time using a for loop for the entire board giving proper **x** and **y** co-ordinates.

Step-3:

Next we initialize the snake, we declare a variable `**alive**` which acts as a flag to see whether the snake is alive or not. And variable `**length**` for the length of the snake which is initialized to 10. As we want the snake to move,we take its head’s and tail’s row number and column number .To keep track of last moved position we have two other variables `**prev_r**` & `**prev_c**` which denote previous head row number and previous head column number respectively.

Step-4:

As the snake has to move to the next block, we have a local variable which keeps track of the next movement of the snake.If we dont press any key(direction) the local variables `**p**` and `**q**` will remain the same we determine the **new_head_row_number** and **new_head_column_number** using these local variables.We update the **previous_head_numbers** (row and column numbers) to the new_head_numbers (row and column numbers) .

Step-5:

Next we have initailized the food by taking a random spot inside the border,we have put a block of colour there ,and we have marked its position as `food_r` and `food_c` which denotes its row and column number respectively._Next we have a function which runs the main game loop.This function has two things to do ,one is to run the background process and the other background process and connect them. Basicaly we call several frames after every 0.1 seconds,all of which have a different process Id’s. Each of these frames will be showing the next state of the game,killing the previous frames using the `trap` command.

Step-6:

We are running this game loop till the snake is alive.We see if the snake is alive using is_dead() function which checks the new_head position of the snake whether it has gone out of the border or not. In the above game loop we implement the game play using the keys

h – to move left ***l – to move right***
j – to move down ***k – to move upwards***

These keys will update the local variables p and q which inturn determines the new_head position of the snake.

Step-7:

In this way the entire game-play runs in a loop with different frames.

Design Component:

- The *Snakes and Ladders* Game implements **functions** concept in bash shell scripting.

- The functions in bash are implemented using the function keyword and called using the function name, thus activating the newPosition function for both player-1 and Player-2.

- Bash-snake has two processes, the foreground one responds to user's control commands, the background one draws the board. `kill` and `trap` are used to enable communication between the two processes.

- The foreground process `getchar()` ignores `SIGINT` and `SIGQUIT`, and replies to the signal of death `SIG_HEAD` by returning from the function `getchar()`. The background process `game_loop()` traps direction control signals from the keyboard, and self-defined signal `SIG_QUIT` which indicates the press of Q button.

- Use $! to grasp the PID of the latest created background process.

- When setting up several traps, it's necessary to guarantee that the normal execution will not be interrupted by the signal handling if the handlers envolve modification of some variables it depends on.

- The snake is represented by the coordinate $head_r and $head_c indicating the row and column on which the snake head is, and a string $body which stores the directions from the head to tail.

- ***Ex***: a snake with this shape (@ is the head) :
  @
  oooo
  o
  This will have a $body with value '21112', meaning 'down,right,right,right,down'.
  With the above scheme and the coordinates of the snake head, we can figure out the position the the entire snake body, without storing every part of its body.

- The board is represented by a "2D array", which is actually implemented by a bunch of bash arrays and the `eval` command.
  For example, assigning 123 to the array entry `arr[5][6]` is done with `eval "arr$i[$j]=123"`.

- The board is re-drawn in each iteration, which happens every 0.1 seconds.
- Coloring is implemented with the escape sequences of the terminal and tput commands.

Conclusion

The main purpose of this project is to use BASH (shell scripting) instead of popular programming languages like C,Java etc .By using this we have created a snake and ladder along with snake's game.There are many commands which are very essential in linux like clear,setting variables,echo which we use in our day to day life.Along with these commands we have used other features of the language like arrays,functions,signals and commands like trap,eval,kill etc
In our project ,2 players compete with each other in snake and ladder ,the winner of the snake and ladder game will proceed to play snakes game.

New Concepts and commands used here:

1) RANDOM - generates a random number
2) tput setab 4 - Sets a background.
3) tput sgr0 - resets the terminal settings(alternative of stty sane)
4) setterm -cursor off - sets sursor off
5) stty -echo - switches off echoing on the screen
6) signals - these help us to do various things
   SIG_UP=35
   SIG_RIGHT=36
   SIG_DOWN=37
   SIG_LEFT=38
   SIG_QUIT=39
   SIG_DEAD=40
7) <process> & - creating a new background process
8) <process_pid>=$! - gives the background process pid
9) kill -$SIG_DEAD $$ - kills the current process
10) functions used are
    - init_game() - initializes the board by blank spaces.
    - move_and_draw() - the draws one character of the border.
    - draw_board() - draws the board using the move and draw function.
    - init_snake() - initializing the snake to its initial state(length=10).
    - is_dead() - to find out whether the snake is alive or not.
    - give_food() - place food at random position
    - move_snake() - moving the snake
    - getchar() - reads the character typed.
    - game_loop() - the main game loop.
    - clear_game() - clearing the game after the game gets over.

References:

https://lasr.cs.ucla.edu/vahab/resources/signals.html
http://man7.org/linux/man-pages/man7/signal.7.html
http://www.alexonlinux.com/signal-handling-in-linux
http://www.careerride.com/Linux-command.aspx

(This page has been left blank intentionally.)