UNIX MINI PROJECT REPORT

DEPARTMENT OF INFORMATION TECHNOLOGY

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

COURSE TITLE

Unix Programming And Practice (IT202)

SUBMITTED TO :

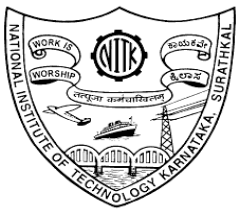
Ms Deepthi L
Assistant lecturer

SUBMITTED BY :

1.AKRITI KUMARI (15IT107)
2.ANKITA BHALAVI (15IT108)
3.RENU CHOUDHARY (15IT136)
4.SUPRIYA SAHOO (15IT145)

SUBMISSION DATE :

14 NOVEMBER 2016



NATIONAL INSTITUTE OF TECHNOLOGY,KARNATAKA

CERTIFICATE

THIS IS TO CERTIFY THAT

AKRITI KUMARI (15IT107)

ANKITA BHALAVI (15IT108)

RENU CHOUDHARY (15IT136)

SUPRIYA SAHOO (15IT145)

HAS SATISFACTORILY COMPLETED THE PROJECT WORK IN UNIX PROGRAMMING & PRACTICE ASSIGNED BY THE INSTITUTE FOR THIRD SEMESTER MINI PROJECT IN THE YEAR 2016-2017.

DATE: 14/11/2016

SIGNATURE OF THE FACULTY HEAD OF THE DEPARTMENT

ABSTRACT

The basic aim behind the project topic i.e."THE SNAKE GAME" is to create a visual environment in which snake and its food appears automatically on the command prompt. Using some basic unix commands we are trying to make a game area within which this game execution is taking place.

For the very first, upon changing the file permission the command prompt will show the users the basic control keys require for the movement of the snake which are-

W - UP

S - DOWN

A - LEFT

D - RIGHT

X - QUIT

These keys will undertake the task of making the snake move in desired direction in such a way so that the snake succeeds in eating the food without hitting the walls.

Once the user is introduced to these keys , message appears along with it to press return so upon pressing ENTER the user is landed into the game area where the food for the snake is already present inside the game area and then the snake appears in its smallest possible size.

Every time the snake succeds in eating the food , its size/length grows .

However there are some conditions for the game termination viz.

- The snake's body should not hit the wall.
- The snake should not eat itself ,in the sense that its mouth should not collide with its body in any manner.

TABLE OF CONTENTS

INTRODUCTION

This project focuses on the application part of shell script and its implementation to create a **Snake game** which is compatible with bash shell.It can run on Linux based systems and makes use of various unix shell commands for its implementation.

The project documentation is concerned with describing the delivered software product in this case the single player Snake game project. Project documentation includes user documentation which tells users how to use the software product and system documentation which is principally intended for further development and understanding.

Snake is a casual video game that originated during the late 1970s in arcades and has maintained popularity since then, becoming something of a classic. After it became the standard pre-loaded game on Nokia mobile phones in 1998, Snake found a massive audiencech is principally intended for further develoment and understanding.

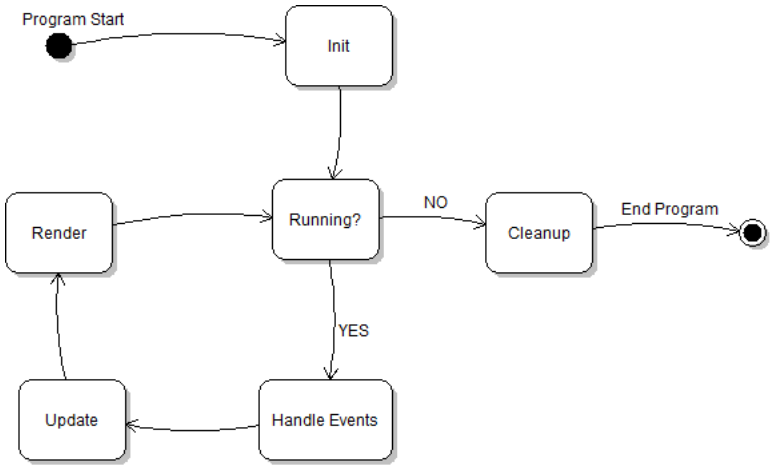


Figure-1.1

This is called the game-loop. Every single game has something like this inside of it. From Pong to Halo to Farmville to World of Warcraft. Every Single Game Has A Game Loop of some kind. This fun little loop is what keeps the game going. If nothing ever looped, the game would start, and then they would just about immediately end. That would be really boring.We used a very simple design for our little Snake game. For brevity and to help keep the focus on the core subject, we neither implemented sound nor used any external bitmaps, and we did not implement any sort of game state management or anything that would otherwise further complicate the program.

For our game, we need two objects.The Snake, and a "Collectable Food" which causes the Snake to grow in length.The rules are simple as well. If the Snake collides with the wall of the screen or itself, then the Snake will die. The player uses the four keys(i.e. W, A, S, D) to control the Snake in any of the four directions; Up, Down, Left, and Right. The Snake cannot be made to move in the opposite direction of it's movement, as that would cause the snake to bite itself instantly causing death. We will use simple '@' to represent the various visuals of the game.

The **scope** of this project is to inspire programmers to learn and make use of unix shell environment and develop application based products for the audience.It also encourages developers to reasearch various aspects and possibilities of shell programming in the field of game development.

CODE :

```
#!/bin/bash
drawborder() {
  tput setf 6
  tput cup $FIRSTROW $FIRSTCOL
  x=$FIRSTCOL
  while [ "$x" -le "$LASTCOL" ];
  do
    printf %b "$WALLCHAR"
    x=$(( $x + 1 ));
  done
  x=$FIRSTROW
  while [ "$x" -le "$LASTROW" ];
  do
    tput cup $x $FIRSTCOL; printf %b "$WALLCHAR"
    tput cup $x $LASTCOL; printf %b "$WALLCHAR"
    x=$(( $x + 1 ));
  done
 tput cup $LASTROW $FIRSTCOL
  x=$FIRSTCOL
  while [ "$x" -le "$LASTCOL" ];
  do
    printf %b "$WALLCHAR"
    x=$(( $x + 1 ));
  done
  tput setf 9
}
apple() {
 APPLEX=$[( $RANDOM % ( $[ $AREAMAXX - $AREAMINX ] + 1 ) ) + $AREAMINX ]
  APPLEY=$[( $RANDOM % ( $[ $AREAMAXY - $AREAMINY ] + 1 ) ) + $AREAMINY ]
}
drawapple() {
  LASTEL=$(( ${#LASTPOSX[@]} - 1))
  x=0
  apple
  while [ "$x" -le "$LASTEL" ];
  do
    if [ "$APPLEX" = "${LASTPOSX[$x]}" ] && [ "$APPLEY" = "${LASTPOSY[$x]}" ];
```

```
      then
     x=0
        apple
      else
        x=$(( $x + 1 ))
      fi
    done
    tput setf 4
    tput cup $APPLEY $APPLEX
    printf %b "$APPLECHAR"
    tput setf 9
}

growsnake() {

    LASTPOSX=( ${LASTPOSX[0]} ${LASTPOSX[0]} ${LASTPOSX[0]} ${LASTPOSX[@]} )
    LASTPOSY=( ${LASTPOSY[0]} ${LASTPOSY[0]} ${LASTPOSY[0]} ${LASTPOSY[@]} )
    RET=1
    while [ "$RET" -eq "1" ];
    do
      apple
      RET=$?
    done
    drawapple
}
move() {
    case "$DIRECTION" in
      u) POSY=$(( $POSY - 1 ));;
      d) POSY=$(( $POSY + 1 ));;
      l) POSX=$(( $POSX - 1 ));;
      r) POSX=$(( $POSX + 1 ));;
    esac
     ( sleep $DELAY && kill -ALRM $$ ) &
    if [ "$POSX" -le "$FIRSTCOL" ] || [ "$POSX" -ge "$LASTCOL" ] ; then
      tput cup $(( $LASTROW + 1 )) 0
      stty echo
      echo " GAME OVER! You hit a wall!"
      gameover
    elif [ "$POSY" -le "$FIRSTROW" ] || [ "$POSY" -ge "$LASTROW" ] ; then
```

```
    tput cup $(( $LASTROW + 1 )) 0
    stty echo
    echo " GAME OVER! You hit a wall!"
    gameover
  fi
  LASTEL=$(( ${#LASTPOSX[@]} - 1 ))
  tput cup $ROWS 0
  printf "LASTEL: $LASTEL"


  x=1
  while [ "$x" -le "$LASTEL" ];
  do
    if [ "$POSX" = "${LASTPOSX[$x]}" ] && [ "$POSY" = "${LASTPOSY[$x]}" ];
    then
      tput cup $(( $LASTROW + 1 )) 0
      echo " GAME OVER! YOU ATE YOURSELF!"
      gameover
    fi
    x=$(( $x + 1 ))
  done
  tput cup ${LASTPOSY[0]} ${LASTPOSX[0]}
  printf " "
LASTPOSX=( `echo "${LASTPOSX[@]}" | cut -d " " -f 2-` $POSX )
  LASTPOSY=( `echo "${LASTPOSY[@]}" | cut -d " " -f 2-` $POSY )
  tput cup 1 10
  #echo "LASTPOSX array ${LASTPOSX[@]} LASTPOSY array ${LASTPOSY[@]}"
  tput cup 2 10
  echo "SIZE=${#LASTPOSX[@]}"



  LASTPOSX[$LASTEL]=$POSX
  LASTPOSY[$LASTEL]=$POSY



  tput setf 2
  tput cup $POSY $POSX
  printf %b "$SNAKECHAR"
  tput setf 9
```

```
  if [ "$POSX" -eq "$APPLEX" ] && [ "$POSY" -eq "$APPLEY" ]; then
    growsnake
    updatescore 10
  fi
}

updatescore() {
  SCORE=$(( $SCORE + $1 ))
  tput cup 2 30
  printf "SCORE: $SCORE"
}
randomchar() {
   [ $# -eq 0 ] && return 1
   n=$(( ($RANDOM % $#) + 1 ))
   eval DIRECTION=\${$n}
}

gameover() {
  tput cvvis
  stty echo
  sleep $DELAY
  trap exit ALRM
  tput cup $ROWS 0
  exit
}
SNAKECHAR="@"
WALLCHAR="Y"
APPLECHAR="o"
SNAKESIZE=3
DELAY=0.2
FIRSTROW=3
FIRSTCOL=1
LASTCOL=40
LASTROW=20
AREAMAXX=$(( $LASTCOL - 1 ))
AREAMINX=$(( $FIRSTCOL + 1 ))
```

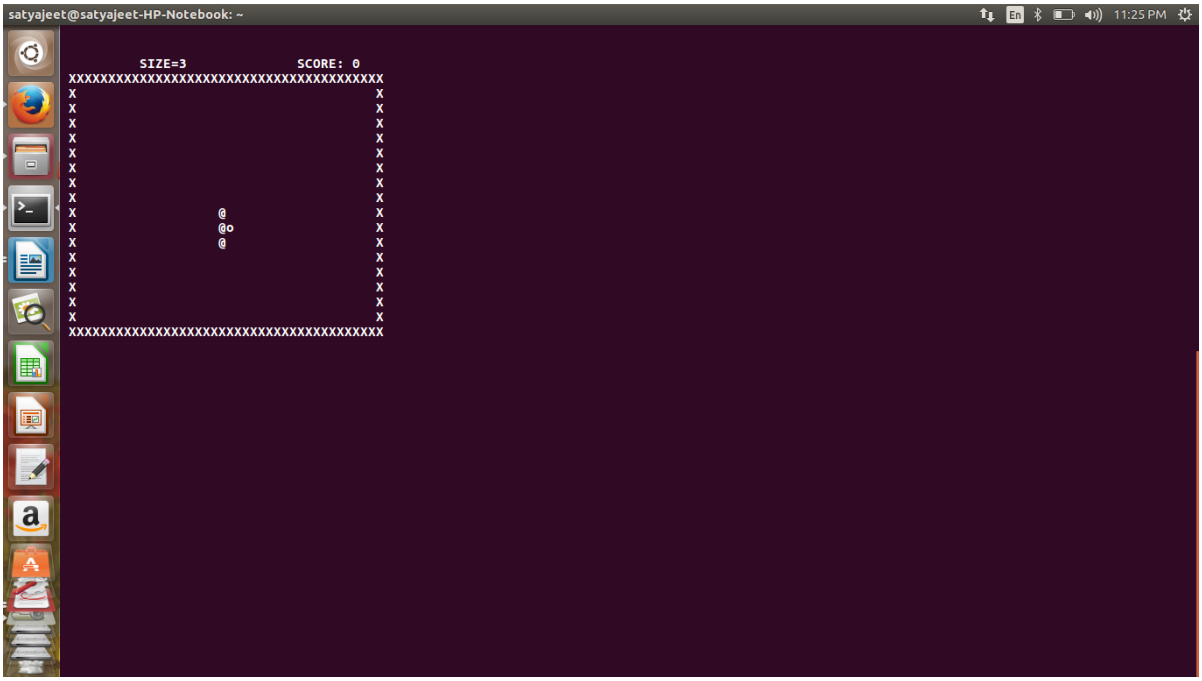
```
AREAMAXY=$(( $LASTROW - 1 )
AREAMINY=$(( $FIRSTROW + 1))
ROWS=`tput lines`
ORIGINX=$(( $LASTCOL / 2 ))
ORIGINY=$(( $LASTROW / 2 ))
POSX=$ORIGINX
POSY=$ORIGINY
ZEROES=`echo |awk '{printf("%0"$SNAKESIZE"d\n",$1)}'| sed 's/0/0 /g'`
LASTPOSX=( $ZEROES )
LASTPOSY=( $ZEROES )
SCORE=0
clear
echo "
Keys:
 W - UP
 S - DOWN
 A - LEFT
 D - RIGHT
 X - QUIT
Press Return to continue
"
stty -echo
tput civis
read RTN
tput setb 0
tput bold
clear
drawborder
updatescore 0
drawapple
sleep 1
trap move ALRM
DIRECTIONS=( u d l r )
randomchar "${DIRECTIONS[@]}"
sleep 1
move
while :
do
   read -s -n 1 key
```

```
   case "$key" in
  ` w)   DIRECTION="u";;`
  ` s)   DIRECTION="d";;`
  ` a)   DIRECTION="l";;`
  ` d)   DIRECTION="r";;`
  ` x)   tput cup $COLS 0
         echo "Quitting..."
         tput cvvis
         stty echo
         tput reset
         printf "Bye Bye!\n"
         trap exit ALRM
         sleep $DELAY
         exit 0`
         ;;
   esac
done
```

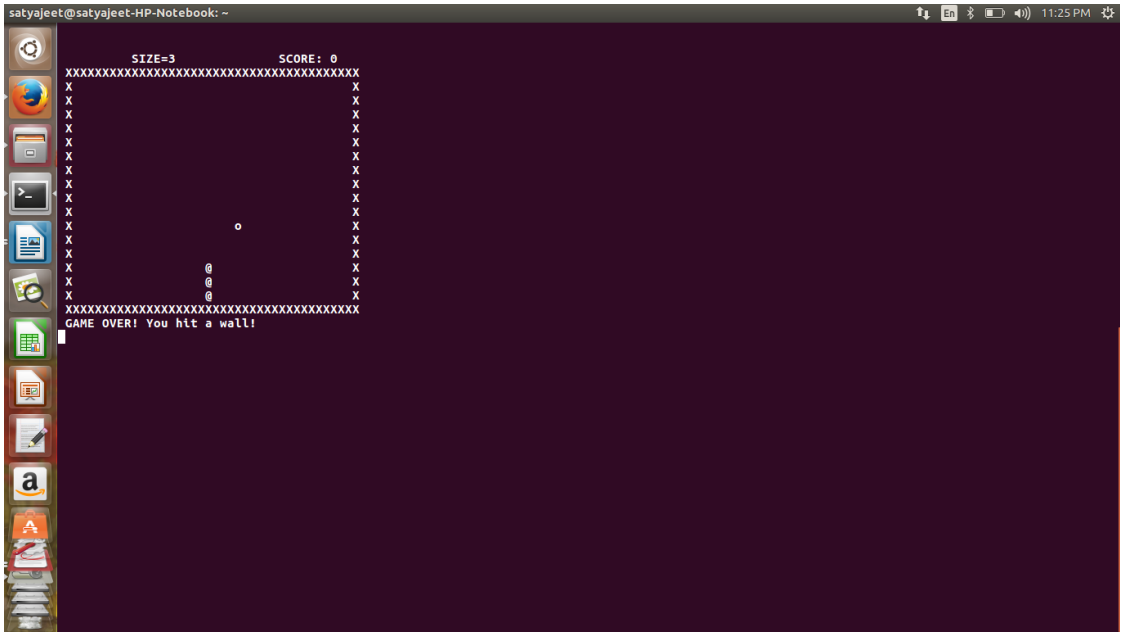
OUTPUT :

The basic control keys to facilitate the movement of snake in the desired direction is being generated here :

The snake is appearing on the screen:



The snake has hit the wall here thereby terminating the game :



FUNCTIONAL REQUIREMENTS

The main requirements of our project is specific unix commands .These commands when used in shell script creates a game environment .In our project these commands are used for proper movement of snake and other processes present in the game these commands are:

1.tput-

This command is used to change the environment of the graphical user interface by manipulating the terminal. In our project tput is used with other arguments which are:

1.1.tput cup : This command sends the sequence TO move the cursor to a particular Co-ordinate M & N by row and column number. For example: tput cup 0 0 moves the cursor to the upper left corner of the screen.In our project it is used to draw the borders of game area,to define the position of the snake and the apple.

1.2.tput setf: This command uses the terminal capabilities and sets the foreground colour to a vlaue given in the integer.

1.3.tput setb: This command uses the terminal capabilities and sets the background colour to a value given in the integer.

1.4.tput civis: This command is used to hide the terminal cursor during game execution.

1.5.tput cvvis: This command makes the cursor visible.

1.6.tput bold: It starts the bold text.

1.7.tput lines: It count the number of lines in terminal.

1.8.tput cols:It counts the number of columns in the treminal.

2.cut-

It will extract the selected portion of the text.

3.stty -echo:

stty command displays or changes the characteristics of TEH treminal.It is used to hide what is being typed in console by the user.

4.sleep:

Suspends program execution for a specified time.Here it does so for 0.2 seconds.

5.kill:

stops running process by sending signal to the process number of current shell.

6.awk :

It does the job of padding out the array with zeroes to start with.As per the code it does the job of appending of 3 more "@" to the body of the snake everytime it eats the food.

7.sed :

It does the job of substituting and replacing onr pattern with the other .

SYSTEM DESIGN

■ *DESIGN GOALS :*

1. To explore a new dimension in the traditional SNAKE GAME to make it more interesting and challenging .

2. To introduce the basic and important unix commands using shell script to make the game implementation more easy in a unix environment .

3. To specify the importance of unix commands and to put it into practice for creation of visuality and movement of this traditional game .

4. To introduce some unknown features of the basic commands and utilities in unix which are used here in order to provide a visual look to this game .

5. To create a parallel contrast between the mobile environment and the unix environment .

6. To extend its viability from phone features to shell scripting features.

7.To offer the experience of commercial multilayer games to the player retaining the simplcity of traditional snake game .

■ *SYSTEM ARCHITECTURE :*

System architecture is the skeleton views behind the visual part of the game .For this particular game it can be divided into two parts :

1. game components

2. game architecture

GAME COMPONENT -:

Game component are behind the scene functions which make up the game and all the functionalities . The components are mesh together and there are a lot of interrelationships between them .

The main components are -

a) drawborder function

b) apple function

c) drawapple function

d) growsnake function

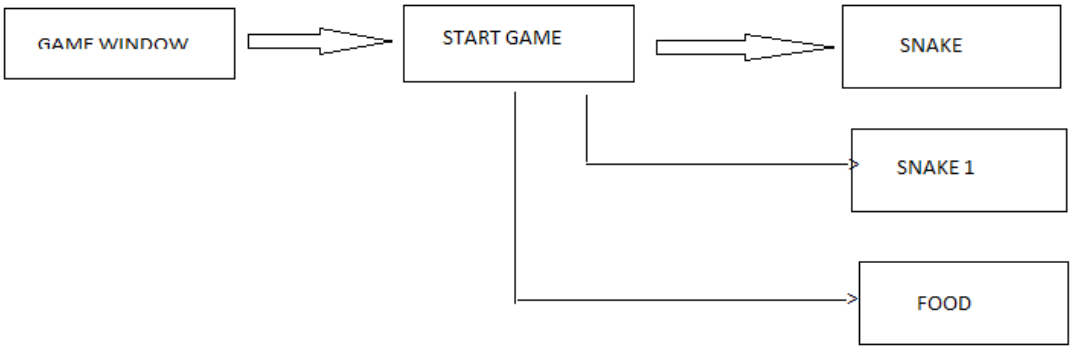
e) move function

f) updatescore function

g) gameover function

GAME ARCHITECTURE -:

The game architecture is the simplified garphical view of the game .It shows how thw component work and basic view of the game at action . The architectural view of the game is very important .simply it gives the overview of the game functionality and makes it easier to understand .



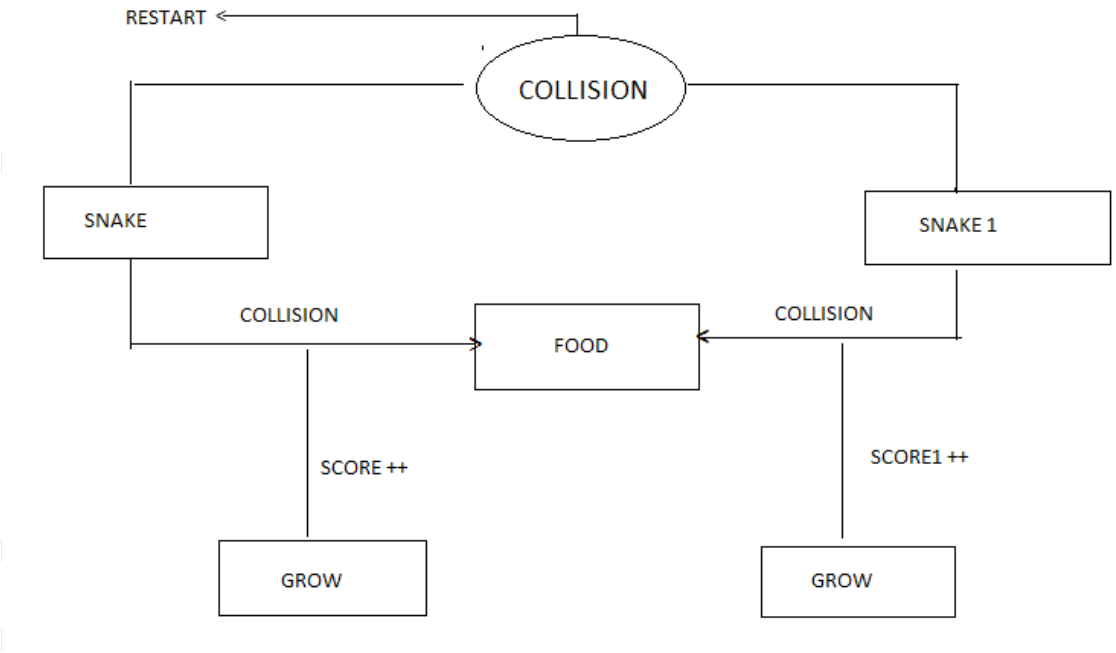


FIG :1.2 simple architecture of snake game

■ *DETAILED DESIGN METHODOLOGIES :*

The crucial methods used in this game implementation are as follows :

a) drawborder() - This function creates the game area within which the snake movement takes place .

b) apple() - This function assigns the coordinates within game area for the food of snake.

c) drawapple() - It checks for an unoccupied space and generates the food there .Food is randomly generated . When the snake eats the food its size grows .

d) growsnake() - It pads out the arrays with the oldest position 3 times to make the snake bigger each time it consumes the food .

e) move() - It facilitates the snake movement in desired direction by using proper direction keys .

f) updatescore() - It updates the player's score .

g) gameover() - It checks for the game termination condition by detecting the snake's collision with the wall or its body .

■ *DEVELOPMENT ENVIRONMENT :*

The basic design tools used in this project are -

- **TERMINAL** : It is an interface in which one can type and execute text based command. The alternative names for it are -

  CONSOLE

  SHELL

  COMMAND LINE

  COMMAND PROMPT

- **LIBRE OFFICE WRITER** : For doing the project documentation part we used the libre office writer software .

CONCLUSION

Proposed workplan of the project:-

1. How did we get the idea:-

Initially while searching for a problem statement we referred a lot of resources such as Steve Parker's shell scripting containing some interesting games using unix commands and shell scripting for example the SPACE INVADER game. We decided to do something simple and interesting and then the snake game came to our mind ,so we started working on it and collected some information through online resources.

2. Future scope of the project:-

As this game is implemented using only basics of shell scripting and some new commands,so we found it to be useful and good enough to choose it as a mini project. We got an idea about how to implement codes for games and while collecting informations about the project we got to know about some new and useful commands.

So in future we can try implementing some more complex and graphics based games and we can include animation as well to make the game interesting and realistic.

3. Improvisation:-

We can make it more intriguing by using new graphics.By using tput setaf command we can make it colorful. We can set different colours for the snake, it's food and boundary as well.Our game is terminating when it hits the wall but we can add a new function to generate a new snake when the snake gets hit by the wall and dies.

Contribution of group members:-

Everyone worked out on the new commands used in the project,and studied all of them. We took help from online resources to get the details about the newly used commands. We are avoiding the use of integrated graphics to keep the code simple and to reduce the complexity. So for the simplicity of the code we are using simple commands and loops to form the snake and it's food. For the documentation part we divided the required formats equally among the group members and so we completed this project .

REFERENCE

- http://www.academia.edu/12695666/snakegame
- https://github.com/pjhades/snakegame
- http://bruxy.regnet.cz/web/snakegame
- http://www.gamedev.net/page/
- Dave Taylor Copyright © 2004 Wicked Cool Shell Scripts—101 Scripts for Linux, Mac OS X and Unix Systems
- Steve Parker Copyright © 2011 Shell Scripting: Expert Recipes for Linux, Bash, and More Published byJohn Wiley & Sons, Inc.

27

36

37

38

39

40

41

42

43