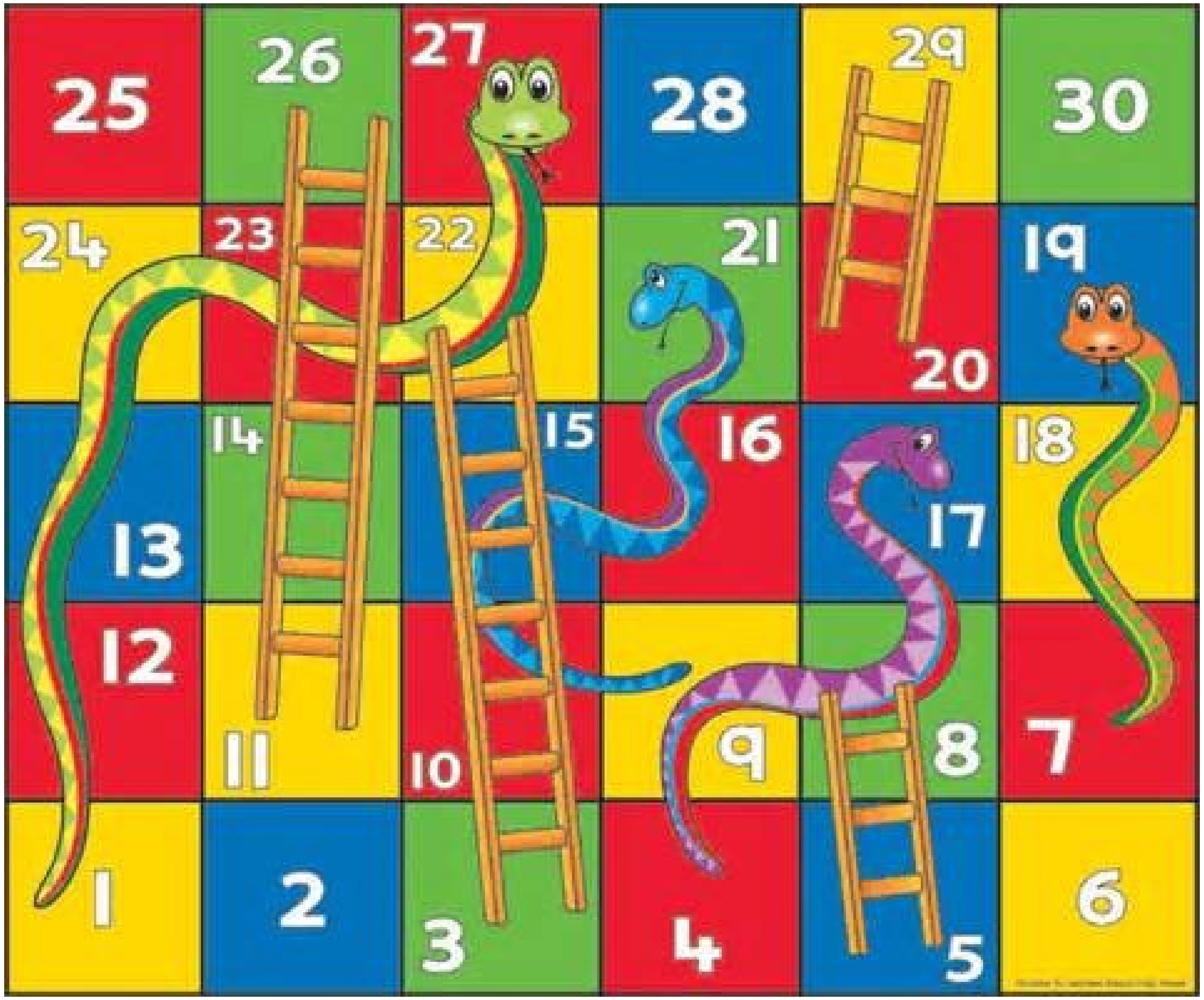
Snakes and Ladders Game.



Rules!

```
tput clear

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

read choice

if ((choice == 1))
then
    #print the rules of the game
fi
```

Menu!

```
if ((choice == 1 || choice == 2))
then
     #Print the board and how the positions ladders and snakes>
     printf "Welcome to Snakes and Ladders.\n"


     printf " 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"
```

Validity of position?

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

if((key==0))
then
    echo " "
    echo "PLAYER 1"
```

Rolling the dice.

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

Checking the position!

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

Checking the position!

```
    if ((Position > newPosition))
    then
      echo -e "\nUnlucky, you have landed on a snake.”
      echo -e “You are now on space $newPosition."
    fi
      Position=$newPosition
    done
printf "Congratulations, you have won!"
fi
```

Function (Check Position):

```
case $Position in
4)  newPosition=35
     ;;
7)  newPosition=23
    ;;

9)  newPosition=5
    ;;
    ...
*)  newPosition=$Position
   ;;
esac
```

Part-2.

Snake’s Game

Declaring variables:

- **`#!/bin/bash`**

- **`IFS=' '`**

- **`declare height=30 width=60`**

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

- **`declare alive`**
- **`declare length`**
- **`declare body`**
- **`declare direction delta_dir`**
- **`declare score=0`**

Setting colors.

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

Declaring signals & arrays:

```
# 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)`**

Initializing the game:

```
 init_game() {
•     clear
•     #switches off the cursor
•     setterm -cursor off
•     #echo -ne "\033[?25l"
•     #pressing the keys will not echo on the screen
•     stty -echo
•
•     for ((i=0; i<height; i++)); do
•         for ((j=0; j<width; j++)); do
•             #setting all the values of the array to ' '
•              eval "arr$i[$j]=' '"
•         done
•     done
• }
```

Drawing the table.

```
move_and_draw(){
    # this takes x and y coordinate
and the colour
    echo -ne "\E[${1};${2}H$3"
}
```

Drawing the board partwise.

```
draw_board() {
    move_and_draw 1 1 "$border_color+$no_color"
    #this prints the top part of the border
    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
```

Continued!

- #this prints the left part of the border
- `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"`

Continued!

```
    #this prints the bottom part of the frame
    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
}
```

Initializing snake:

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

  #heads column number and row number
•     head_r=$((height/2-2))
•     head_c=$((width/2))
```

The direction variables:

```
•   body=' '
•     for ((i=0; i<length-1; i++)); do
•         body="1$body"
•     done
•
  #declaring local variables p and q
•     local p=$((${move_r[1]} * (length-1)))
•     local q=$((${move_c[1]} * (length-1)))
```

Keeping track.

```
   #tails row number and column number
```

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

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

```
  #to keep track of the last moved position
```

- `prev_r=$head_r`
- `prev_c=$head_c`

- `b=$body`

Moving the Snake:

```
• 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 the snake dead?

- 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
- }

Food (Score)!

```
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\""
}
```

Moving the Snake.

```
• 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
```

Moving the Snake.

```
• 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
```

Continued.

```
•     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 snake’s direction:

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

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

Getting input from keyboard:

```
•     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
• }
```

Main game loop:

```
• 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    ${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
```

Continued.

```
•     echo -e "\n  ${text_color}Oh, No! You are dead!$no_color"
•
•     # signals the input loop that the snake is dead
•     kill -$SIG_DEAD $$
• }

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

Calling functions.

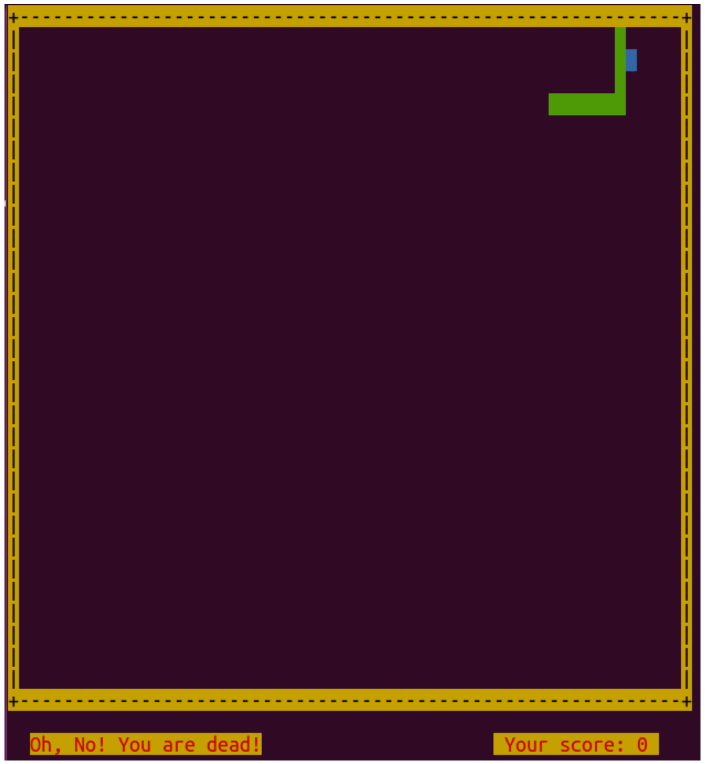
- `init_game`
- `init_snake`
- `give_food`
- `draw_board`

- `game_loop &`
- `game_pid=$!`
- `getchar`

- `clear_game`
- `exit 0`

The Game

The game looks something like this..



```
Oh, No! You are dead!                    Your score: 0
```

Thank you