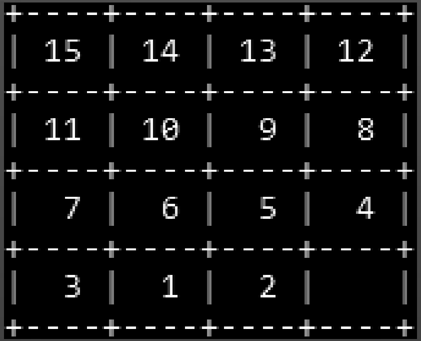
MBA Tech.
Computer Engineering Presentation

Game of Tiles

Presented by -
Anmaya Agarwal (N002)
Yash Katariya (N024)
Shreyash Palodkar (N036)

Problem Statement

- The goal of this puzzle is to arrange the board's tiles from smallest to largest, left to right, top to bottom, with an empty space in board's bottom-right corner. Here we show an example of 4x4 that has numbers from 1 to 15 and the bottom-right corner is empty.



```
+----+----+----+----+
| 15 | 14 | 13 | 12 |
+----+----+----+----+
| 11 | 10 |  9 |  8 |
+----+----+----+----+
|  7 |  6 |  5 |  4 |
+----+----+----+----+
|  3 |  1 |  2 |    |
+----+----+----+----+
```

Introduction

- A sliding puzzle, sliding block puzzle, sliding tile puzzle or The Game of Tiles is a combination puzzle that challenges a player to slide pieces along certain directions (vertically and horizontally) to establish a certain end-configuration (from smallest to largest, left to right, top to bottom) in the least possible number of moves.

- It is a puzzle played on a square, two-dimensional board which can be of dimensions from 3x3 to 9x9 with numbered tiles that slide.

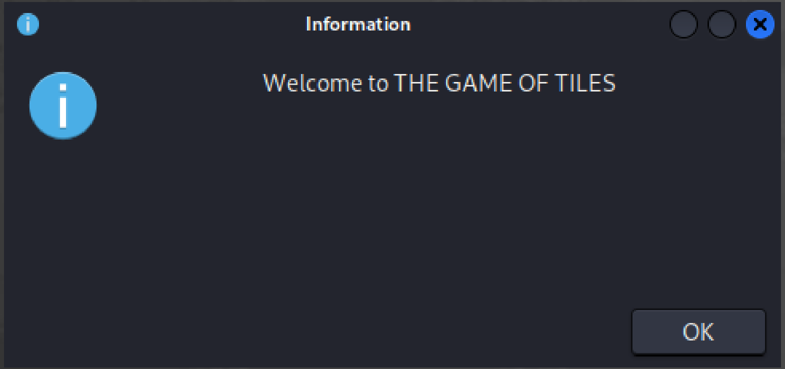
- Sliding any tile that borders the board’s empty space in that space constitutes a "move."

Objective and Scope of Project

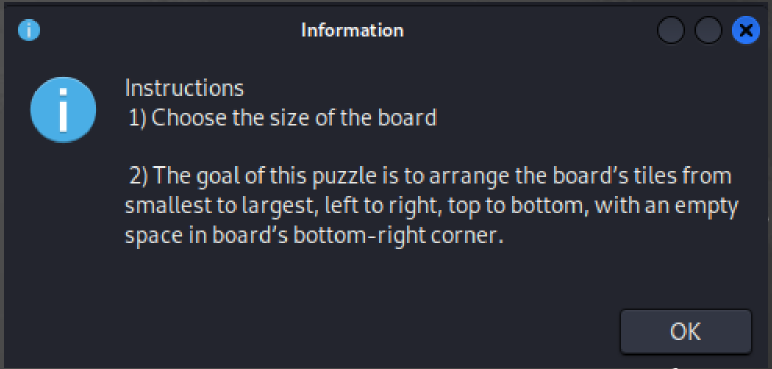
- The Tiles Swap Game is a sliding-numbered tile puzzle played on a square, two-dimensional board. The objective of this puzzle is to place the tiles on the board in the following order: smallest to largest, left to right, top to bottom, with an empty space in the bottom-right corner.

- A sliding block puzzle prohibits lifting any piece off the board. This property separates sliding puzzles from rearrangement puzzles. Hence, finding moves and the paths opened up by each move within the two-dimensional confines of the board are important parts of solving sliding block puzzles.

Layout and GUI


Information

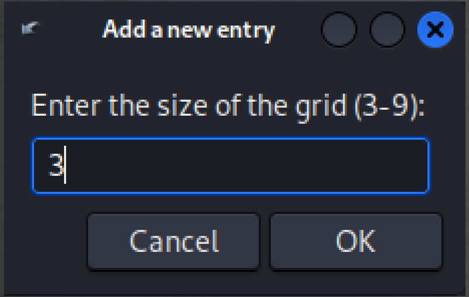
Welcome to THE GAME OF TILES


Information

Instructions
 1) Choose the size of the board

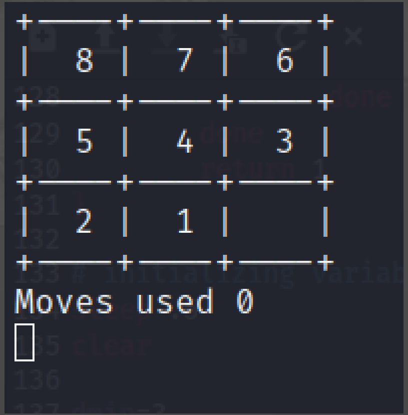
 2) The goal of this puzzle is to arrange the board's tiles from smallest to largest, left to right, top to bottom, with an empty space in board's bottom-right corner.

Layout and GUI

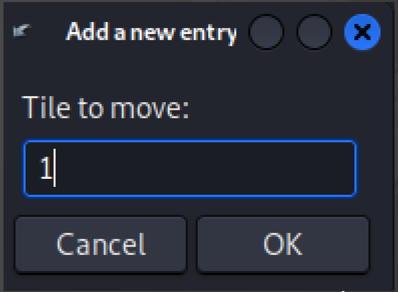


Enter the size of the grid (3-9):

3



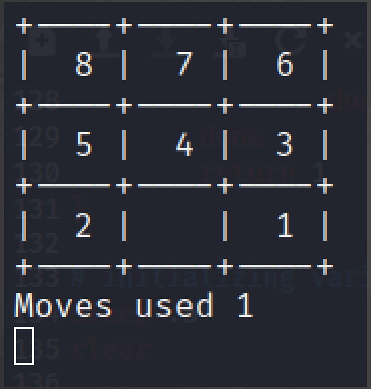
```
+----+----+----+
|  8 |  7 |  6 |
+----+----+----+
|  5 |  4 |  3 |
+----+----+----+
|  2 |  1 |    |
+----+----+----+
Moves used 0
```



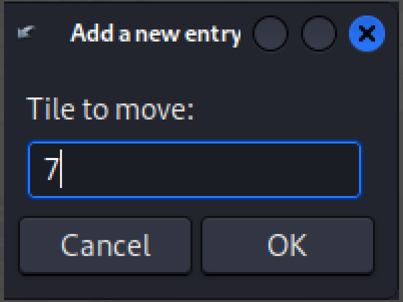
Tile to move:

1

Layout and GUI



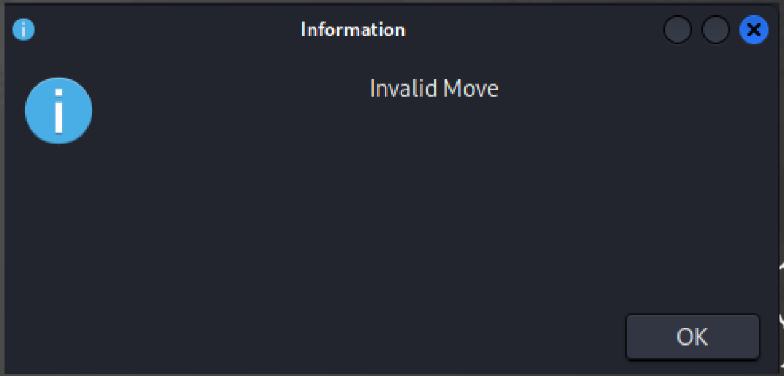
```
+-----+-----+-----+
|  8  |  7  |  6  |
+-----+-----+-----+
|  5  |  4  |  3  |
+-----+-----+-----+
|  2  |     |  1  |
+-----+-----+-----+
Moves used 1
```



Add a new entry

Tile to move:

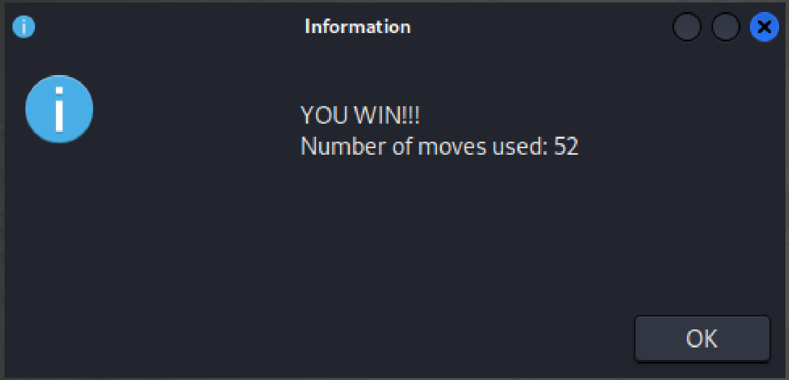
7



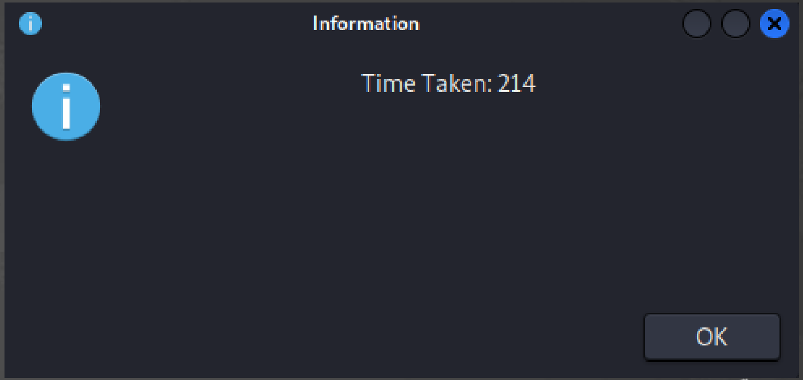
Information

Invalid Move

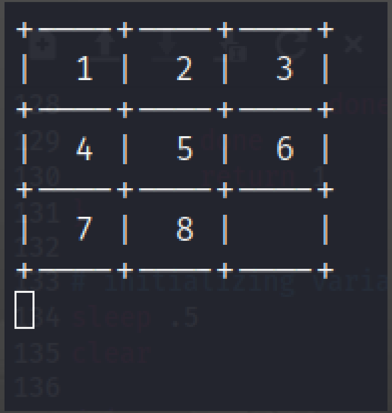
Layout and GUI



YOU WIN!!!
Number of moves used: 52



Time Taken: 214



```
+-----+-----+-----+
|  1  |  2  |  3  |
+-----+-----+-----+
|  4  |  5  |  6  |
+-----+-----+-----+
|  7  |  8  |     |
+-----+-----+-----+
```

Unix commands used

- Sleep: sleeps current thread
- Date: gives the date
- Cat: for file operations
- Head: gives the first n lines of file
- Sort: sorts the data
- Exit, break, clear
- Local, if, while, array declare, functions
- Expr: for calculations
- Printf, echo, read, clear
- Grep, sed
- Command line arguments

Modules

- **Init:** initialize the board according to user input
- **Draw:** draws current status of the board. Here we use loops and global variable to print it.
- **Move:** swaps the values according to user value. The user enters the tile number so we have to find the index of the number and check whether it is a valid move. If the move is valid then we swap the indices.
- **Won:** checks whether game is completed or not. This is done by iterating through each element and comparing with the true positional value
- **High score:** gives the high score and updates it. We have create a file that stores high score and we retrieve the data and compare with current high score.

Result and Conclusion

- For decades, games have interested and challenged mathematicians and computer scientists, from soccer to Sudoku, Go to Connect Four. Many times, a game may appear to have a simple premise or set of rules, but when researchers examine it more closely, a slew of intricate and intriguing issues arise. Scholars are drawn to games because of their deceptive and sometimes unexpected depth.

- In the Game of Tiles the user has to move the tiles keeping in mind that the final result should be in ascending order, and at the end when the ascending order pattern is achieved the game ends with the number of moves displayed.

Thank You!

Code for Game of Tiles –

```
#!/bin/bash

# initialise the board
init(){

        ((temp=$d*$d-1))
        for((i=0; i<d; i++))
        do
                for((j=0; j<d; j++))
                do
                        (($1[$i,$j]=$temp))
                        ((temp=$temp-1))
                done
        done
        ((temp3=$d-1))
        ((temp4=$d-2))
        ((temp5=$d-3))
        (($1[$temp3,$temp3]=0))
        ((temp2=$d%2))
        if [ $temp2 -eq 0 ]
        then
                (($1[$temp3,$temp4]=2))
                (($1[$temp3,$temp5]=1))
        fi
}

# draw the board
display(){
        board=$1
        println
        for((i=0; i<d; i++))
        do  echo -n "|"
                for((j=0; j<d; j++))
                do
                        if [ ${board[$i,$j]} -gt 0 ]
                        then
                                printf '%3d |' "${board[$i,$j]}"
                        elif [ ${board[$i,$j]} -eq 0 ]
                        then
                                echo -n "    |"
                        else
                                echo -n " _|"
                        fi
                done
                echo
                println
        done
}
```

```
println()
{
        local i
        echo -n "+"
        for((i=0;i<d;i++))
        do
                echo -n "----+"
        done
        echo
        return
}

# make a move
mov1()
{       board=$2
        echo "$1"
        for((i=0; i<d; i++))
        do
                for((j=0; j<d; j++))
                do
                        if [ ${board[$i,$j]} -eq $1 ]
                        then
                                ((temp1=$i+1))
                                ((temp2=$j+1))
                                ((temp3=$i-1))
                                ((temp4=$j-1))
                                ((temp5=$d-1))
                                if [[ ( ${board[$i,$temp2]} -eq 0 )  && ( $j -lt $temp5 ) ]]
                                then
                                   swap $i $j $i $temp2 board
                                   return 1
                                fi
                                if [[ (${board[$i,$temp4]} -eq 0) && ( $j -gt 0 ) ]]
                                then
                                   swap $i $j $i $temp4 board
                                   return 1
                                fi
                                if [[ (${board[$temp1,$j]} -eq 0) && ( $i -lt $temp5) ]]
                                then
                                   swap $i $j $temp1 $j board
                                   return 1
                                fi
                                if [[ (${board[$temp3,$j]} -eq 0) && ( $i -gt 0 ) ]]
                                then
                                   swap $i $j $temp3 $j board
                                   return 1
                                fi
                                return 0
                        fi
                done
        done
        return 0
}

swap(){
        board=$5
        ((tempor=${board[$1,$2]}))
        ((board[$1,$2]=${board[$3,$4]}))
        ((board[$3,$4]=$tempor))
}

# check whether the user won or not
won(){
        board=$1
        k=1
        ((tem=$d-1))
        for((i=0; i<d; i++))
        do
                for((j=0; j<d; j++))
                do
                        if [[ ( $i -ne $tem ) || ( $j -ne $tem ) ]]
                                then
                                if [[ ( ${board[$i,$j]} -ne k ) ]]
                                        then
                                        return 0
                                fi
                        fi
                        ((k=$k+1))
                done
        done
        return 1
}

# initializing variables
sleep .5
clear

dmin=3
dmax=9
move=0

# game starts
zenity --info --width=400 --height=200 --text "Welcome to THE GAME OF TILES"
zenity --info --width=400 --height=200 --text "Instructions \n 1) Choose the size of the
board\n\n 2) The goal of this puzzle is to arrange the board’s tiles from smallest to largest,
left to right, top to bottom, with an empty space in board’s bottom-right corner."
#echo -e "Welcome to THE GAME OF TILES\n"
if [ $# -eq 0 ]
then
```

```
	t=true
	j=0
	while $t
	do
		d=$(zenity --entry --text="Enter the size of the grid (3-9): ")
		#echo -n "Enter the size of the grid (3-9): "
		#read d
		if [[ ( d -lt dmin ) || ( d -gt dmax ) ]]
		then
			zenity --info --width=400 --height=200 --text "Invalid Input"
			#echo "Invalid Input"
		else
			t=false
		fi
	done
elif [ $# -eq 1 ]
then
	if [ "$1" == "highscore" ]
	then
		zenity --info --width=400 --height=200 --text "High Scores :-\n\nN TIME
MOVES\n`cat highscores`"
		#echo -e "High Scores :-\n\nN TIME MOVES"
		#cat highscores
		exit
	else
		zenity --info --width=400 --height=200 --text "Invalid Argument"
		#echo "Invalid Argument"
		sleep .5
		exit
	fi
else
	zenity --info --width=400 --height=200 --text "Invalid Argument"
	#echo "Invalid Argument"
	sleep .5
	exit
fi

clear
declare -A board
init board

startTime=`date +%s`
for((s=1; s>0; s++))
do
	printf "\033c"
	#zenity --info --width=400 --height=200 --text `display board`
	display board

	#check if won
	won board
	m=$?
	if [[ $m == 1 ]]
	then
		zenity --info --width=400 --height=200 --text "\nYOU WIN!!!\nNumber of
moves used: $move"
		#echo -e "\nYOU WIN!!!\n"
		#echo "Number of moves used: $move"
		endTime=`date +%s`
		time=`expr $endTime - $startTime`
		zenity --info --width=400 --height=200 --text "Time Taken: $time\n"
		echo -e "Time: $time\n"
		# printf "%d %3d %5d\n" $d $time $move | cat >> highscores
		o=`grep ^$d highscores | tr -s " " | cut -d " " -f 3`
		if [ "$o" == "--" ]
		then
			l=`expr $d - 2`
			s=`printf "%d %3d %5d\n" $d $time $move`
			sed "$l s/.*/$s/" highscores > temp
			cat temp | cat > highscores
		elif [ $move -lt $o ]
		then
			l=`expr $d - 2`
			s=`printf "%d %3d %5d\n" $d $time $move`
			sed "$l s/.*/$s/" highscores > temp
			cat temp | cat > highscores
		elif [ $move -eq $o ]
		then
			t=`grep ^$d highscores | tr -s " " | cut -d " " -f 2`
			if [ $time -lt $t ]
			then
				l=`expr $d - 2`
				s=`printf "%d %3d %5d\n" $d $time $move`
				sed "$l s/.*/$s/" highscores > temp
				cat temp | cat > highscores
			fi
		fi
		break
	else
		#zenity --info --width=400 --height=200 --text "Moves used $move"
		echo "Moves used $move"
	fi

	tile=$(zenity --entry --text="Tile to move: ")
	#echo -n "Tile to move: "
	#read tile
	sleep .2

	if [ $tile -eq 0 ]
		then
		zenity --info --width=400 --height=200 --text "Exiting"
		#echo "Exiting"
		break
	fi

	mov1 $tile board
	n=$?
	if [[ $n == 0 ]]
		then
		zenity --info --width=400 --height=200 --text "Invalid Move"
		#echo "Invalid Move"
		sleep 2
	else
		((move=$move+1))
	fi
done
```