**ACKNOWLEDGEMENTS**

Although this project was an effort of three individuals, it would have been very difficult for us to complete this efficiently and timely without the support, guidance and suggestion of our seniors and advisors.

We have no words that can adequately carry our sincere gratitude to our lecture, Mr. Ajay Kumar Shah, for providing us the opportunity and valid platform to conduct the project according to our competence and proficiency. His way of making us do all the rough work and combine them as the fundamentals of the project was a very good concept indeed. Since he has provided us with all the basic concepts on the programming languages like C and C++, as well as the techniques essential for carrying out this project so, we personally are now aware of how benefitted we are with this incredible method of learning. We are very grateful to him for his valuable suggestions and supervision in this third semester project. We will always be indebted to him for inspiring us to develop programs in the very first place.

We would also like to acknowledge and thank all the faculty members for their kind assistance and regulation on the way to build this project.

Finally, we would like to provide our sincere appreciation to everyone who has helped us in completing and developing this project in highly efficient manner.

**ABSTRACT**

This project titled, “FOOD HUNT” is a mini concept to the snake game. We have used “ C programming language” to code this game. Without graphics although this game may not look cozy but this is an interesting game to play and we have focused on the logic rather than the outlook. This game has features like every other game nearby us. Since we have also used the concept of file handling, this game has the feature of data file where anyone can save their name after they have played the game and eventually the file will have the player’s name, score and played date. The user can also view the past record of the players who played the game. Thus, all in all we have tried to make this project like a real one and user friendly too.

**TABLE OF CONTENTS**

Background 4

Program Output 5

Methodology 8

Source Code: 19

Conclusion 31

Future Plans 32

References 33

**BACKGROUND**

This game is primarily designed for the third semester project of BE COMPUTER for the partial fulfillment of grades. This program is written in C programming language using Code Blocks. This project, as its title “FOOD HUNT” suggests, is an attempt to develop a mini snake game that is not only a game but is an approach to create much more complicated software like these.

We don’t boast that the project is complete by itself and bug-free at its present version. However, we believe that it can serve as a starting point for future for whoever wants to develop more interactive and user friendly software.

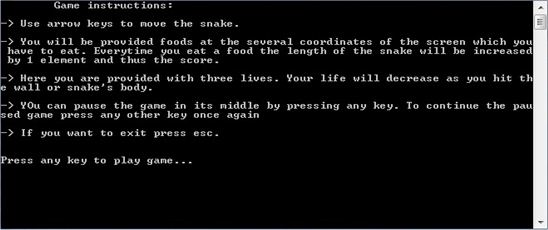
We sincerely hope that the program justifies its purpose and proves helpful to anyone who might use it as the base of logic along with the source of entertainment.

Anyone wishing to improve and upgrade the program is heartily welcome to do so.

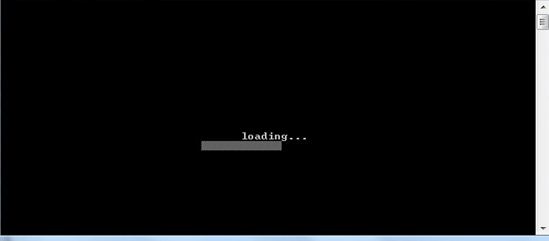
**PROGRAM OUTPUT**

One can say that this program is a very fragile program in a sense. The biggest problem during the execution of following was that since we had to use clear screen command large no of times so the screen would flicker making the user a bit difficult to play the game but nevertheless amidst the shortage of time and capability to use a large no of commands we had tried very hard to make the game according to as it is being played or to its international standard .Similarly ,lack of simulation of colors and shapes the game may see a bit dull as we are in running the game in a Windowed Console environment .

At the first of the execution of the of program we see the following instruction screen telling the user about how to play the game

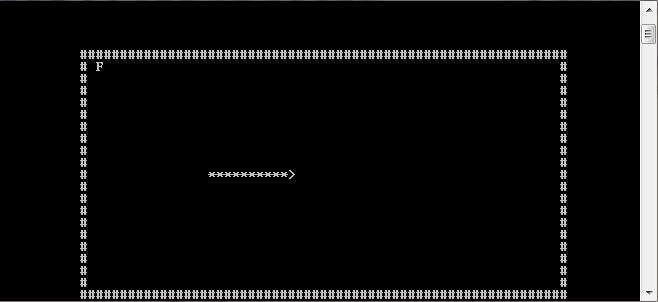


After the user gets the necessary instructions then the loading screen is displayed which is a trial simulation of advanced software that we see around us which looks as follows.



Then we come to another and the real phase of the programmers where we actually play the game .During the program execution we get to see foods represented by ‘F’ displayed randomly in the screen as well as we also see

that the length of snake varies of as the snake gets to eat the food then we also have displayed the moving of snake in very original and actual way as we see in real high graphics mode games.



Then after the user sets to play the game the snake eats the food and goes on increasing the length until a fixed number. Accordingly ,the score increases .But if the snake touches its own body part or the boundary the lives decreases by 1 until 0 after which another screen appears telling the user that the game is completed.



Then the user is asked to give his/her name then user is asked whether he wants to see the past records played by other players or not. After he presses ‘y’ another screen appears showing the log of all players who played the game with their name, played date and their score.



**METHODOLGY**

**ALGORITHM**

1. **Main()**

As the program run, it starts executing from main function. Since the global variable could not be initialized in header file required global variables were initialized here. Many functions like Print(), Border() and Move() functions are called here for different functions. As it calls Move() function the movement of snake takes place. And the main function is never called again. So the program never terminates from the main function in this project.

Algorithm for function main()

Step 1: Start

Step 2: Initialize globally declared variables as:

Length=5

Head.x=25

Head.y=20

Head.direction=RIGHT

Life=3

Step 3: Call the function Border() to print walls of the game.

Step 4: Call the function Food() to specify the x and y coordinates of food.

Step 5: Call the function Move() for snake’s movement.

Step 6: End

1. **Print():**

This function just displays different information about the game e.g. the rules of the games, instructions and the control keys. Neither any value is passed to this function nor it returns any. The header file “header,h” is included in this function as well to include all the header files required for different functions used in this function.

Algorithm for function Print():

Step 1: Start

Step 2: Display:

“Welcome to the mini Snake game.(press any key to continue)”

Step 3: Make the screen still until any character is entered by the user using getch() function.

Step 4: Display:

“Game instructions:

->Use arrow keys to move the snake.

-> You will be provided foods at the several coordinates of the screen which you have to eat. Everytime you eat a food the length of the snake will be increased by 1 element and thus the score.

-> Here you are provided with three lives. Your life will decrease as you hit the wall or snake's body.

-> You can pause the game in its middle by pressing any key. To continue the paused game, press any other key once again.

-> If you want to exit press esc.

Press any key to play game...”

Step 5: Once again make the screen still until the user enters any character

Step 6: Check if(getch()==27)

Step 7: If yes exit the game, else continue the game.

Step 8: End

1. **Food()**

This function too is called every time before displaying snake’s length, to check the coordinates of the head and the food and to change the food coordinate when its equal.The length is increased by 1 element when the coordinates are equal. This function also uses time function. A function ‘rand()’ is used to ramdomize the x and y coordinates of the food. As it returns same values every time the program is run, a function srand() is used before ‘rand()’ using a time variable. Thus it ramdomizes different values every time the program is run, taking respect to the system time.

Algorithm for function Food():

Step 1: Start

Step 2: Declare a variable of type time\_t

Step 3: Assign a=time(NULL)

Step 4: Call ‘srand(a)’ function

Step 5: Check if food.x==0

Step 6: If true, perform:

6.1 food.x=rand()%70

6.2 if food.x<=10, perform food.x+=11

6.3 food.y= rand()%30

6.4 if food.y<=30, perform food.y+=11

Step 7: Else if check head.x=food.x and head.y=food.y

Step 8 If yes, perform:

8.1 increase the length by 1.

8.2 food.x=rand()%70

8.3 if food.x<=10, perform food.x+=11

8.4 food.y= rand()%30

8.5 if food.y<=30, perform food.y+=11

**(d) Border()**

This function contains the coordinates of the walls or boarder of the snake game. We have specified the game between 10 and 30 in Y-Coordinate and between 10 and 70 in X-Coordinate. The function is called each time before displaying the snake i.e. before displaying the snake in any other position. It also consists a function system(“cls”), under header file <stdlib.h> which clears the screen. Screen is to be cleared every time while calling function Boarder for, to erase the previous position of the snake.

Algorithm for function Boarder():

Step 1: Start

Step 2: Call the function ‘system(“cls”)’ to clear the screen.

Step 3: Initialize i=10

Step 4: Check if i<71

Step 5: If yes perform:

5.1: Call the function GotoXY(i,10)

5.2: display ‘#’

5.3: Call the function GotoXY(i,30)

5.4: Display ’#’

5.5: i++

5.6: Repeat step 4

Step 6: Again assign i=10

Step 7:check if i<30

Step 8: If yes perform:

8.1: Call the function GotoXY(10,i)

8.2: display ‘#’

8.3: Call the function GotoXY(70,i)

8.4: Display ’#’

8.5: i++

8.6: Repeat step 4

Step 9: GotoXy(food.x,food.y)

Step 10: Display ‘F’

Step 11: End

**(e) Move()**

This is the main function of the program. It moves the snake in the required direction until any character is pressed. To move the snake until any character is pressed, a function ‘kbhit()’ is used under header file ‘<conio.h>’, which returns a true value if there is any character in the buffer, else false value. Thus the snake is moved until it returns true value. It also calls different functions like Boarder(), Delay(), fflush()(to clear the buffer if any character is pressed while running the funcition for the first time). For movement of the snake, it calls different functions like Right(), Left() etc. according to the direction of the head. Whenever any charcter is pressed it checks the character and execute accordingly. Of the character is among the arrow keys, it saves the current position of the head in an element-bend \_no of an array bend[] after increasing its value by 1. And the key is assigned to the head.direction. If the entered key is ‘esc’, it exits the game, and if the key is other, then is just gives an alert sound. Here double getch() function is used to get the key because arrow keys are constitue of two ASCII values. But if characters other than arrow keys are entered, the game will be paused due to next getch() function until any other key is pressed. Finally it calls itself(recursive function).

Algorithm for the function Move():

Step 1: Start

Step 2: Call the function Food()

Step 3: Assign len=0

Step 4: i=0

Step 5: Check if i<30

Step 6: If yes perform:

6.1 body[i].x=0

6.2 body[i].y=0

6.3 if i=length, break

6.4 i++

Step 7: Repeat step 5

Step 8: Call the function Delay() passing the length

Step 9: Call the function Boarder() to print the boarder

Step 10: Check head.direction

10.1 If head direction=RIGHT, call Right()

10.2 If head.direction=LEFT, call Left()

10.3 If head.direction=UP, call Up()

1.4 If head.direction=DOWN, call Down()

Step 11: Call function ExitGame()

Step 13: Call function kbhit(), check the return value

Step 14: If returned 0, repeat step 2

Step 15: Else, perform:

15.1 a=getch()

15.2 Check if a=27, if yes exit the game

15.3 key=getch()

15.4 Check if ((key==RIGHT&&head.direction!=LEFT&&head.direction!=RIGHT)|| (key==LEFT&&head.direction!=RIGHT&&head.direction!=LEFT)||(key==UP&&head.direction!=DOWN&&head.direction!=UP)||(key==DOWN&&head.direction!=UP&&head.direction!=DOWN))

15.5 if true , assign key into the head.direction and change the coordinates of head accordingly

15.6 else call the function Move()

Step 16: End

**(f) Right()**

This function is called if the head.direction is RIGHT. Head.direction will not be assigned RIGHT if the previous head.direction is right or left. As soon as the function is called it displays the elements of the snake from head coordinate to the last bending coordinate taking reference to len and length. After printing each element it increases the len by 1 and checks if len<length. If its false then it breaks the loop. But if e difference is less than the length then it calls the function ‘Bend()’ for further displaying the snake’s elements.

Algorithm for function Right():

Step 1: Start

Step 2: Initialize i=0

Step 3: Check if (i<=(head.x-bend[bend\_no])&&len<length)

Step 4: if true, perform:

4.1 body[len].x=head.x-I and body[len].y=head.y

4.2 GotoXY(body[len].x,body[len].y)

4.2 Check if len=0

4.3 If yes, print ‘>’

4.4 Else print ’\*’

4.5 len++

4.6 Repeat step 3

Step 5: Call function Bend()

Step 6:Check if there is any character in the buffer, If no, perform:

6.1 head.x++

Step 5: End

**(g) Left():**

Similar to the function Right(), its too called when the head.direction is LEFT. Head.direction can’t be assigned the value LEFT when its RIGHT. As Right() function did, as soon as this function is called, it also prints the elements of snake’s body from the position of head to the position of last bend. It prints ‘<’ if the value of len is 0, to differentiate body from head. If the value of len is less than length after printing all the elements in the gap between head and latest bend position, it calls the function Bend() for further elements.

Algorithm for function Left():

Step 1: Start

Step 2: Initialize i=0

Step 3: Check if (i<=(bend[bend\_no]-head.x)&&len<length)

Step 4: if true, perform:

4.1 GotoXY((head.x+i),head.y)

4.2 Check if len=0

4.3 If yes, print ‘<’

4.4 Else print ’\*’

4.5 len++

4.6 body[len].x=head.x+I and body[len].y=head.y;

4.7 Repeat step 3

Step 5: Call function Bend()

Step 6: Check if there is any character in buffer, if no perform head.x--

Step 7: End

**(h) Up() and Down()**

Similar to Right () and Left () functions, these functions are also called when the head. Direction is respectively UP and DOWN. Initially it too prints the element from the head position to the last bend position and then call the function Bend() for printing remaining element.Up() function prints ‘^’ and down function prints ‘v’ when the value of len is

0.

**(I) Bend()**

This is too one of the important function of our project. It is called by various functions like Right(), Left(), Up() and Down() when the length is greater than the difference between the head position and the last bend position. It displays ‘\*’ between different positions of different bend numbers starting from bend\_no. As len=length, it breaks the loop i.e all the elements of the snake have been printed.

Algorithm for Bend():

Step 1: Start

Step 2: Initialize i=bend\_no

Step 3: Check if i>0 &&len<length)

Step 4: If true, perform:

4.1 Check if (bend[i].x==bend[i-1].x , if yes perform:

4.1.1 Print ‘\*’ from bend bend[i].y to bend[i-1].y and assign every coordinates to body[len] increasing the len till either len<lenth or loop continues

4.2 Check if(bend[i].y==bend[i-1].y&&len<length), if yes perform:

4.2.1 Print ‘\*’ from bend bend[i].x to bend[i-1].x and assign every coordinates to body[len] till either len<lenth or loop continues

4.3 i—

Step 5: Repeat step 3

Step 6: End

**(j) ExitGame():**

This function too is called every time after displaying all the length of the snake. The game exits when the head of the snake touches the wall or its own body. To check whether it has touched its body or not a loop is generated which checks the coordinates of the head and that of the other body parts. If they are found same the life is decreased by 1. To check whether it touched the wall or not, its x or y coordinates are compared with that of the boundary. If both of them are found same then the life is decreased by 1 again. As this way the game will come to an end when the user loses its all four lives, including extra three lives. After the completion of the life, the user is asked his name and is stored with the score in a file.

 Algorithm for function ExitGame():

Step 1: Start

Step 2: Initialize i=4, check=0

Step 3: Check if body[0].x=body[i].x&&body[0].y==body[i].y

Step 4: If true, perform

4.1 check++

Step 5: If false, perform

5.1 Check if i==length||check!=0

                        5.1.1 if true, break

            5.2 i++

            5.3 Repeat step 3

Step 6: Check if head.x<=10||head.x>=70||head.y<=10||head.y>=30||check>0

            6.1 If true, perform

                        6.1.1 life—

                        6.1.2 check if life>=0

                                    6.1.2.1 If true, perform:

                                                6.1.2.1.1 head.x=25

                                                6.1.2.1.2head.y=20

                                                6.1.2.1.3 bend\_no=0

                                                6.1.2.1.4 head.direction=RIGHT

                                                6.1.2.1.5 Call Move()

                                    6.1.2.2 Else perform:

                                                6.1.2.2.1 Clear the screen

                                                6.1.2.2.2 Display:

                                                            “All lives completed

Better Luck Next Time!!!

Press any key…”

6.1.2.2.3 Call function Record()

Step 7: End

**APPENDIX**

**Our game has included the following source code .**

**FOOD HUNT CODES**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <conio.h>

#include <ctype.h>

#include <windows.h>

#include <process.h>

#define UP 72

#define DOWN 80

#define LEFT 75

#define RIGHT 77

int length;

int bend\_no;

int len;

char key;

void record();

void load();

int life;

void Delay(long double);

void Move();

void Food();

int Score();

void Print();

void gotoxy(int x, int y);

void GotoXY(int x,int y);

void Bend();

void Border();

void Down();

void Left();

void Up();

void Right();

void ExitGame();

int Scoreonly();

struct coordinate{

int x;

int y;

int direction;

};

typedef struct coordinate coordinate;

coordinate head, bend[500],food,body[50];

int main()

{

char key;

Print();

system("cls");

load();

length=5;

head.x=25;

head.y=20;

head.direction=RIGHT;

Border();

Food(); //to generate food coordinates initially

life=3; //number of extra lives

bend[0]=head;

Move(); //initializing initial bend coordinate

return 0;

}

void Print()

{

printf("\tWelcome to the mini Snake game- FOOD HUNT .(Press any key to continue..)\n");

getch();

system("cls");

printf("\tGame Instructions:\n");

printf("\n-> Use arrow keys to move the snake.\n\n-> You will be provided foods at the several coordinates of the screen which you have to eat. Every time you eat a food the length of the snake will be increased by 1 element and so the score.\n\n-> Here you are provided with three lives. Your life will decrease as you hit the wall or snake's body.\n\n-> You can pause the game in its middle by pressing any key. To continue the paused game press any other key once again.\n\n-> If you want to exit press 'Esc'. \n");

printf("\n\nPress any key to Hunt...");

if(getch()==27)

exit(0);

}

void load()

{

int row,col,r,c,q;

gotoxy(36,14);

printf("loading...");

gotoxy(30,15);

for(r=1;r<=20;r++){

for(q=0;q<=100000000;q++);//to display the character slowly

printf("%c",177);}

getch();

}

void Move()

{

int a,i;

do{

Food();

fflush(stdin);

len=0;

for(i=0;i<30;i++)

{

body[i].x=0;

body[i].y=0;

if(i==length)

break;

}

Delay(length);

Border();

if(head.direction==RIGHT)

Right();

else if(head.direction==LEFT)

Left();

else if(head.direction==DOWN)

Down();

else if(head.direction==UP)

Up();

ExitGame();

}while(!kbhit());

a=getch();

if(a==27)

{

system("cls");

exit(0);

}

key=getch();

if((key==RIGHT&&head.direction!=LEFT&&head.direction!=RIGHT)||(key==LEFT&&head.direction!=RIGHT&&head.direction!=LEFT)||(key==UP&&head.direction!=DOWN&&head.direction!=UP)||(key==DOWN&&head.direction!=UP&&head.direction!=DOWN))

{

bend\_no++;

bend[bend\_no]=head;

head.direction=key;

if(key==UP)

head.y--;

if(key==DOWN)

head.y++;

if(key==RIGHT)

head.x++;

if(key==LEFT)

head.x--;

Move();

}

else if(key==27)

{

system("cls");

exit(0);

}

else

{

printf("\a");

Move();

}

}

void gotoxy(int x, int y)

{

COORD coord;

coord.X = x;

coord.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

void GotoXY(int x, int y)

{

HANDLE a;

COORD b;

fflush(stdout);

b.X = x;

b.Y = y;

a = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleCursorPosition(a,b);

}

void Delay(long double k)

{

Score();

long double i;

for(i=0;i<=(10000000);i++);

}

void ExitGame()

{

int i,check=0;

for(i=4;i<length;i++) //starts with 4 because it needs minimum 4 element to touch its own body

{

if(body[0].x==body[i].x&&body[0].y==body[i].y)

{

check++; //check's value increases as the coordinates of head is equal to any other body coordinate

}

if(i==length||check!=0)

break;

}

if(head.x<=10||head.x>=70||head.y<=10||head.y>=30||check!=0)

{

life--;

if(life>=0)

{

head.x=25;

head.y=20;

bend\_no=0;

head.direction=RIGHT;

Move();

}

else

{

system("cls");

printf("All lives completed\nBetter Luck Next Time!!!\nPress any key to quit the game\n");

record();

exit(0);

}

}

}

void Food()

{

if(head.x==food.x&&head.y==food.y)

{

length++;

time\_t a;

a=time(0);

srand(a);

food.x=rand()%70;

if(food.x<=10)

food.x+=11;

food.y=rand()%30;

if(food.y<=10)

food.y+=11;

}

else if(food.x==0)/\*to create food for the first time coz global variable are initialized with 0\*/

{

food.x=rand()%70;

if(food.x<=10)

food.x+=11;

food.y=rand()%30;

if(food.y<=10)

food.y+=11;

}

}

void Up()

{

int i;

for(i=0;i<=(bend[bend\_no].y-head.y)&&len<length;i++)

{

GotoXY(head.x,head.y+i);

{

if(len==0)

printf("^");

else

printf("\*");

}

body[len].x=head.x;

body[len].y=head.y+i;

len++;

}

Bend();

if(!kbhit())

head.y--;

}

void Down()

{

int i;

for(i=0;i<=(head.y-bend[bend\_no].y)&&len<length;i++)

{

GotoXY(head.x,head.y-i);

{

if(len==0)

printf("v");

else

printf("\*");

}

body[len].x=head.x;

body[len].y=head.y-i;

len++;

}

Bend();

if(!kbhit())

head.y++;

}

void Left()

{

int i;

for(i=0;i<=(bend[bend\_no].x-head.x)&&len<length;i++)

{

GotoXY((head.x+i),head.y);

{

if(len==0)

printf("<");

else

printf("\*");

}

body[len].x=head.x+i;

body[len].y=head.y;

len++;

}

Bend();

if(!kbhit())

head.x--;

}

void Right()

{

int i;

for(i=0;i<=(head.x-bend[bend\_no].x)&&len<length;i++)

{

body[len].x=head.x-i;

body[len].y=head.y;

GotoXY(body[len].x,body[len].y);

if(len==0)

printf(">");

else

printf("\*");

len++;

}

Bend();

if(!kbhit())

head.x++;

}

void Bend()

{

int i,j,diff;

for(i=bend\_no;i>=0&&len<length;i--)

{

if(bend[i].x==bend[i-1].x)

{

diff=bend[i].y-bend[i-1].y;

if(diff<0)

for(j=1;j<=(-diff);j++)

{

body[len].x=bend[i].x;

body[len].y=bend[i].y+j;

GotoXY(body[len].x,body[len].y);

printf("\*");

len++;

if(len==length)

break;

}

else if(diff>0)

for(j=1;j<=diff;j++)

{

body[len].x=bend[i].x;

body[len].y=bend[i].y-j;

GotoXY(body[len].x,body[len].y);

printf("\*");

len++;

if(len==length)

break;

}

}

else if(bend[i].y==bend[i-1].y)

{

diff=bend[i].x-bend[i-1].x;

if(diff<0)

for(j=1;j<=(-diff)&&len<length;j++)

{

body[len].x=bend[i].x+j;

body[len].y=bend[i].y;

GotoXY(body[len].x,body[len].y);

printf("\*");

len++;

if(len==length)

break;

}

else if(diff>0)

for(j=1;j<=diff&&len<length;j++)

{

body[len].x=bend[i].x-j;

body[len].y=bend[i].y;

GotoXY(body[len].x,body[len].y);

printf("\*");

len++;

if(len==length)

break;

}

}

}

}

void Border()

{

system("cls");

int i;

GotoXY(food.x,food.y); /\*displaying food\*/

printf("F");

for(i=10;i<71;i++)

{

GotoXY(i,10);

printf("#");

GotoXY(i,30);

printf("#");

}

for(i=10;i<31;i++)

{

GotoXY(10,i);

printf("#");

GotoXY(70,i);

printf("#");

}

}

void record(){

char plname[20],nplname[20],cha,c;

int i,j,px;

FILE \*info;

info=fopen("record.txt","a+");

getch();

system("cls");

printf("Enter your name\n");

scanf("%[^\n]",plname);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for(j=0;plname[j]!='\0';j++){ //to convert the first letter after space to capital

nplname[0]=toupper(plname[0]);

if(plname[j-1]==' '){

nplname[j]=toupper(plname[j]);

nplname[j-1]=plname[j-1];}

else nplname[j]=plname[j];

}

nplname[j]='\0';

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fprintf(info,"Player Name :%s\n",nplname);

time\_t mytime; //for date and time

mytime = time(NULL);

fprintf(info,"Played Date:%s",ctime(&mytime));

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fprintf(info,"Score:%d\n",px=Scoreonly()); //call score to display score

for(i=0;i<=50;i++)

fprintf(info,"%c",'\_');

fprintf(info,"\n");

fclose(info);

printf("Want to see past records ? press 'y'\n");

cha=getch();

system("cls");

if(cha=='y'){

info=fopen("record.txt","r");

do{

putchar(c=getc(info));

}while(c!=EOF);}

fclose(info);

}

int Score()

{

int score;

GotoXY(20,8);

printf("SCORE : %d",(length-5));

score=length-5;

GotoXY(50,8);

printf("Life : %d",life);

return score;

}

int Scoreonly()

{

int score=Score();

system("cls");

return score;

}

**CONCLUSION**

In conclusion, I still want to emphasize that the program is not complete by itself. There is still a lot of room for improvement. Graphics may be added to program to make it more attractive as well as interactive. The body of the snake may be made more beautiful by adding various color codes.

This is an open source program and therefore everybody is welcome to develop it. Future developers are very welcome to add their ideas to the program and improvise it.

Hopefully, all the users of this program will find it useful and entertaining.

**FUTURE PLANS**

This is our first version of game and needs improvement. We will be getting back to it as soon as we can. Anybody has permission to copy this game executable and source code and modify it, improve it, anything you want. For all those programmers excited to get their hands on we suggest you can start may be with the following things:

1. More interactive Introducing Menu

2. A two Player Game

3. Difficulty levels

4. Colors in even the snake, food and boundary

5. Special bonuses

6. Sound Effects

7. Background music and much more....

**REFERENCES**

**Books**

“Programming with C” by K.R. Venugopal and Sudeep R. Prasad

“A Book On C” by All Kelley Ira Pohl

“LET US C” by Yashwant Kantekar

**Internet**

[www.CProgramming.com](http://www.CProgramming.com)

[www.Bytes.com](http://www.Bytes.com)

[www.clickanerd.com](http://www.clickanerd.com)

[www.markhneedham.com](http://www.markhneedham.com)

[www.onlinecolleges.net](http://www.onlinecolleges.net)