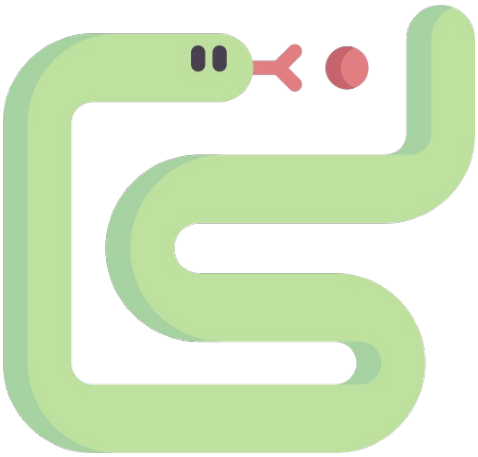
SNAKE

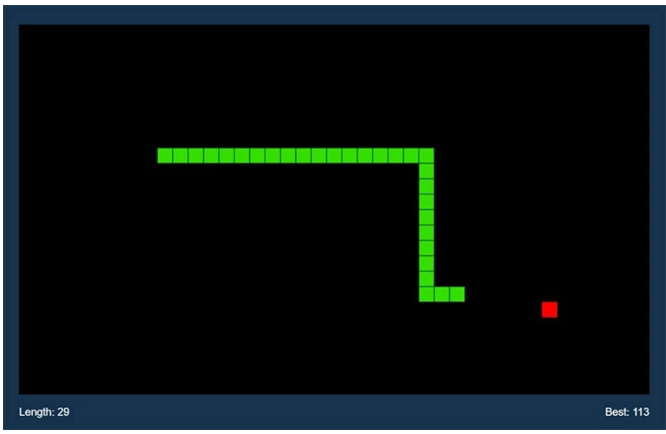


Final Project

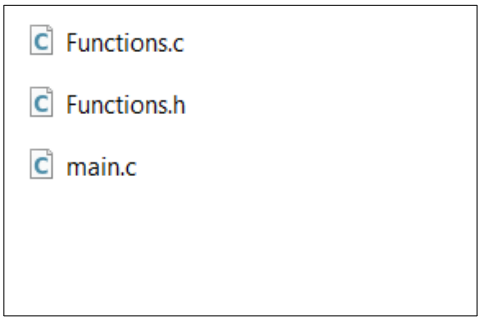
Gasham Huseynli
Tofig Alizade
Farhad Budagov
Javid Bayramli

Game Rules Overview:

1.The Snake is capable of moving in x,y directions
2.crossing boarders result to snake appearing from the opposite side
3.eating an apple grows snake's tail
4.hitting the tail = loose



Files Overview:



1.The game consists of 3 files
Functions.c – overall function used in game
Functions.h – the header for previous file to import inside main
main.c – the file manipulating functions to play the game

1.Functions.h

by itself this file consists of only 16 lines(12 functioning)
it includes 2 global variables responsible to snake's movements and 7 functions in Functions.c file

```c
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

extern int dy;
extern int dx;

void direction(int direction);
void greetings();
void print_hashes(int y_height, int x_length, int pos_y, int pos_x);
int print_snake(int coords_y[], int coords_x[], int *length, int dir);
int random_number(int min, int max);
void spawn_apple();
int is_apple_eaten(int coords_head_y, int coords_head_x, int dir);

#endif

```

2.Functions.c

the complitation of all functions and variables used in game
just to note, the library ncurses.h was also used

```c
extern int dy;
extern int dx;

// FUNCTIONS BLOCK
void print_hashes(int, int, int, int);
void greetings();
int print_snake(int[], int[], int*, int);
int random_number(int, int);
void spawn_apple();
```

```c
void greetings() {
    move(0, 25); // move the curso
    print_hashes(3, 25, 0, 25);

    // if hashbox starts in positi
    move(2, 36);
    addstr("SNAKE");
```

VOID Greetings

prints a border with title "Snake"

uses move() to move cursor and print_hashes function in order to print the box (explained below)

```c
void direction(int direction) {
    switch (direction) {
        case 1:
            dx = 0; dy = -1;
            break;
        case 2:
            dx = 0; dy = 1;
            break;
        case 3:
            dx = -1; dy = 0;
            break;
        case 4:
            dx = 1; dy = 0;
            break;
        default:
            dx = 0; dy = 0;
            break;
    }
}
```

Void Direction:

Based on the value given to function decides where to move snake (dy – vertical, dx – horizontal)
dy = -1 (down)
dy = 1(up)
dx = 1(right)
dx = -1(left)

void print_hashes

this function prints the borders of game “#”
variables follow next rules
y_heigh = number of rows in a box
x_length = number of columns in a box
pos_y = starting Y coordinate on screen, pos_x = starting X coordinate on screen

```c
void print_hashes(int y_height, int x_length, int pos_y, int pos_x) {
    move(pos_y, pos_x);
    for (int i = 0; i < x_length + 2; i++) {
        addch('#');

    }


    pos_y++; // switch to next line
    move(pos_y, pos_x);

    for (int i = 0; i < y_height; i++) {
        addch('#');

        for (int j = 0; j < x_length; j++) {
            addch(' ');

        }
        addch('#');
        pos_y++;
        move(pos_y, pos_x);
    }

    for (int i = 0; i < x_length + 2; i++) {
        addch('#');
    }
}
```

for example typing print_hashes(10,10,0,0) you ask to build a box 10 to 10 beginning from (0;0) coordinates

int print(snake)

```
    // print the snake
    for (int i = 0; i < *length; i++) {
        move(coords_y[i], coords_x[i]);
        addch('o');
    }

    // clear the path
    move(coords_y[*length], coords_x[*length]);
    addch(' ');

    return 1;
}
```

used to print the snake coords_y – Y coordinates of snake's body coord_s – X coordinates *length – length of snake(it is a pointer since the size of snake is a modifiable value needed to be manipulated after eating an apple),dir – a variable send to direction() function determining where the snake should move, mvinch(coords_y[0] + dy, coords_x[0] + dx) detects what is in front of snake," " - means empty, "@" means an apple, so snake eats apple and grows it length, return 1 if a move is successful, and 0 if not (game over, ran into a wall)

```
int print_snake(int coords_y[], int coords_x[], int *length, int dir) {
    // direction: 1 - up, 2 - down, 3 - left, 4 - right
    // let dx and dy be the direction, as we estimated above
    direction(dir);

        // check if next is block
        if (mvinch(coords_y[0] + dy, coords_x[0] + dx) == ' '){}
        else if (mvinch(coords_y[0] + dy, coords_x[0] + dx) == '@') {
            // if ate an apple
            coords_y[*length + 1] = coords_y[*length] - dy;
            coords_x[*length + 1] = coords_x[*length] - dx;
            (*length)++;
        }
        else {
            return 0;
        }


        // the body of the snake
        for (int i = *length - 1; i > 0; i--) {
            coords_y[i] = coords_y[i - 1];
            coords_x[i] = coords_x[i - 1];
        }

        // the head
        coords_y[0] += dy;
        coords_x[0] += dx;
```

int random_number:

```c
int random_number(int min, int max) {
    return rand() % (max - min + 1) + min;
}
```

just returns a random number, used to put an apple in game

(max – min + 1) + min – writes a limit to keep apple in a box

void spawn_apple:

this function spawn an apple at coordinates from 1 – 10 x and y apple is “@”

```c
void spawn_apple() {
    int y = random_number(1, 10);
    int x = random_number(1, 10);
    mvaddch(y, x, '@');
}
```

int is_apple_eaten:

detects if the next move of snake consists of apple, if yes returns 1(there is an apple), if not 0, basically detecting if in next move of snake an apple will be eaten

```c
int is_apple_eaten(int coords_head_y, int coords_head_x, int dir) {
    direction(dir); // uploading dy and dx values for correct checking

    if (mvinch(coords_head_y + dy, coords_head_x + dx) == '@') {
        return 1;
    }
    return 0;
}
```

3.Main.c

```c
int dy = 0;
int dx = 0;
```

global variables tracking out snake

NCURSES

```
initscr();
noecho();
curs_set(0); // hide the cursor
cbreak();
keypad(stdscr, TRUE);
timeout(0);
```

initscr(); - starts ncurses mode before doing anything
noecho();prevents showing user inputing from keyboard on screen
curs_set(0) – disables the blinking cursor
cbreak(); - basically whatever is inputted does not wait to be confirmed, immediately inputs in programm
keypad(); enables arrows (arrow – up, arrow-down)
timeout(0);makes getch(); unblocking, basically code goes on even if user didn't press anything

```
greetings();
refresh();

getch();
clear();
```

greetings – print box with the title
snake; refresh(); - draws it to screen, getch(); waits for any key to be pressed by user
clear();wipes the screen to start the game

```
int x[10] = {1, 2, 3};
int y[10] = {1, 1, 1};
int len = 3;

int apple_x, apple_y; // coordinates of an
int is_apple_exist = 0; // if apple isnt ea
// an apple and hold them untill an apple w
```

int x[] = starting positions of snake in X
int y[] = starting positions of snake in Y

int len = default length of snake(3)

int apple_x,apple_y – coordinates of apple
int is_apple_exist = 0 – defaultly there is no apple in game, based on its value an apple will be created in random coordinates

```
int ch;
int dir = 4; // initially snake moves to the right

int status = 1;
srand(time(NULL)); // random here will create an apple in the same position evey time.
// Using this function to prevent it and add some randomness.
```

ch – stores user input
dir – movement of snakes (defaultly moves to right)
status = 1 if game in ongoing (0 – game over)
srand(time(NULL)) – to get random variables everytime

```
while (status) {
```

a loop with our game (stops only when status = 0 (game over))

```
if (!is_apple_exist) {
    apple_x = random_number(1, 10);
    apple_y = random_number(1, 10);

    is_apple_exist = 1;
} // if an apple is eaten or there is no ap

// print the map
print_hashes(10, 10, 0, 0);
mvaddch(apple_y, apple_x, '@');

if (is_apple_eaten(y[0], x[0], dir)) {
    is_apple_exist = 0;
}
```

checks if there is an apple in game, if not creates and sets is_apple_exist = 1(meaing yes it exists)

print_hashes (prints map)
mvaddch – adds an apple

then checks if apple has been eaten, if yes is_apple_exist goes again to 0

```
    status = print_snake(y, x, &len, dir);

    ch = getch();
    switch (ch) {
        case KEY_UP:
            dir = 1;
            break;
        case KEY_DOWN:
            dir = 2;
            break;
        case KEY_LEFT:
            dir = 3;
            break;
        case KEY_RIGHT:
            dir = 4;
            break;
        case 'q':
            endwin();
            return 0;
    }
    direction(dir);
    napms(300);
}

endwin();
return 0;
```

status – takes an integer (1 or 0) depending if snake hasnt got stuck anywhere (1 if alive, 0 – dead)
ch = getch(); gets a direction from player to move snake to it (q – exist from game)

then calls direction function to change direction of snake, namps(300) sets reset for 300milliseconds (time to wait for next frame),
endwin(); ends the game, called only if loop is broken, which is broken only when snake is dead or status = 0.