Contents

List of Figures

List of Tables

1. Introduction

A header file ‘graphics.h’ which is a C programming language library is commonly used for creating simple computer graphics applications. It provides a set of functions for drawing basic shapes, colors, and images on the screen. The library also allows to set up a graphics environment, to capture mouse and keyboard events for user to interact with the graphics window and simple animations.

1.1 Objectives

The main objectives of the project are:

- To learn and implement different functions of graphics.
- To interface the application of graphics to the real world.
- To familiarize with graphics and its logical coding.

2. Literature Review

2.1 Related Theory

- **`GRAPHICS.H`** is a header file in C that is used to include graphics functions in a C program. It is used for drawing various shapes and also to animate objects. It is also used to color the objects drawn.

  Syntax:

  ```
  #include <graphics.h>
  ```

- **`TIME.H`** is a C standard library header that provides functions and types for working with time and date-related operations. It allows C programs to access and manipulate time values, handle date and time information, and measure time intervals.

  Syntax:

  ```
  #include <time.h>
  ```

- **`WINDOWS.H`** is a header file in the Microsoft Windows environment primarily used in C and C++ programming languages to access various Windows-specific functionalities and create Windows applications.

  Syntax:

  ```
  #include <windows.h>
  ```

- **`initgraph()`**: The `initgraph()` function initializes the graphics system by loading a graphics driver from disk and puts the system into graphics mode. It also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults, and resets the graph result to 0.

  Syntax:

```
void initgraph(
    int *graphdriver, int *graphmode, char *pathtodriver
);
```

- **`initwindow()`**: The `initwindow()` function initializes the graphics system by opening a graphics window of the specified size. It has three parameters: width, height, and title. The title parameter will be printed at the top of the window.

  Syntax:

  ```
  initwindow(width, height, title);
  ```

- **`srand()`**: The `srand()` function is used to initialize random number generators. The `srand()` function sets the starting point for producing a series of pseudo-random integers.

  Syntax:

  ```
  srand(time(NULL));
  ```

- **`setfillstyle()`**: The `setfillstyle()` function sets the current fill pattern and fill color. It consists of two parameters: *pattern* and *color*.

  Syntax:

  ```
  void setfillstyle(int pattern, int color);
  ```

- **`bar()`**: The `bar()` function is used to draw a 2-dimensional, rectangular filled-in bar.

  Syntax:

  ```
  void bar(int left, int top, int right, int bottom);
  ```

Color	**INT VALUES**
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

Table 2.1: Color Values

Pattern Fill	**INT VALUES**
EMPTY_FILL	0
SOLID_FILL	1
LINE_FILL	2
LTSTLASH_FILL	3
SLASH_FILL	4
BKSLASH_FILL	5
LTBKSLASH_FILL	6
HATCH_FILL	7
XHATCH_FILL	8
INTERLEAVE_FILL	9
WIDE_DOT_FILL	10
CLOSE_DOT_FILL	11
USER_FILL	12

Table 2.2: Pattern Fill Values

Table 2.3: Color and Pattern Fill Values

- **`rand()`**: The `rand()` function is used to generate pseudo-random numbers. Pseudo-random numbers are numbers that appear to be random but are generated by a deterministic algorithm.

  Syntax:

  ```
  int rand(void);
  ```

- **`getpixel()`**: The `getpixel()` function returns the color of a pixel present at location *(x, y)*.

  Syntax:

  ```
  int getpixel(int x, int y);
  ```

- **`GetAsyncKeyState()`**: The `GetAsyncKeyState()` function provides information about a key. It checks whether a key is pressed or not.

  Syntax:

  ```
  short GetAsyncKeyState(int key);
  ```

CODE	**MEANING**
VK_LSHIFT	LEFT-SHIFT KEY
VK_RSHIFT	RIGHT-SHIFT KEY
VK_LCONTROL	LEFT-CONTROL KEY
VK_RCONTROL	RIGHT-CONTROL KEY
VK_LMENU	LEFT-MENU KEY
VK_RMENU	RIGHT-MENU KEY

Table 2.4: Key Code Meanings

- **`delay()`**: The `delay()` function is used to stop the execution of the program for a specified period. It accepts a time in milliseconds to pause the execution.

  Syntax:

  ```
  void delay(unsigned int);
  ```

- **`cleardevice()`**: The `cleardevice()` function clears the screen in graphics mode and sets the current position to *(0, 0)*. It fills the screen with the current background color.

  Syntax:

  ```
  void cleardevice();
  ```

- **`settextstyle()`**: The `settextstyle()` function is used to change the way text appears. It can modify the size of the text, change the direction, and change the font.

  Syntax:

  ```
  void settextstyle(int font, int direction, int charsize);
  ```

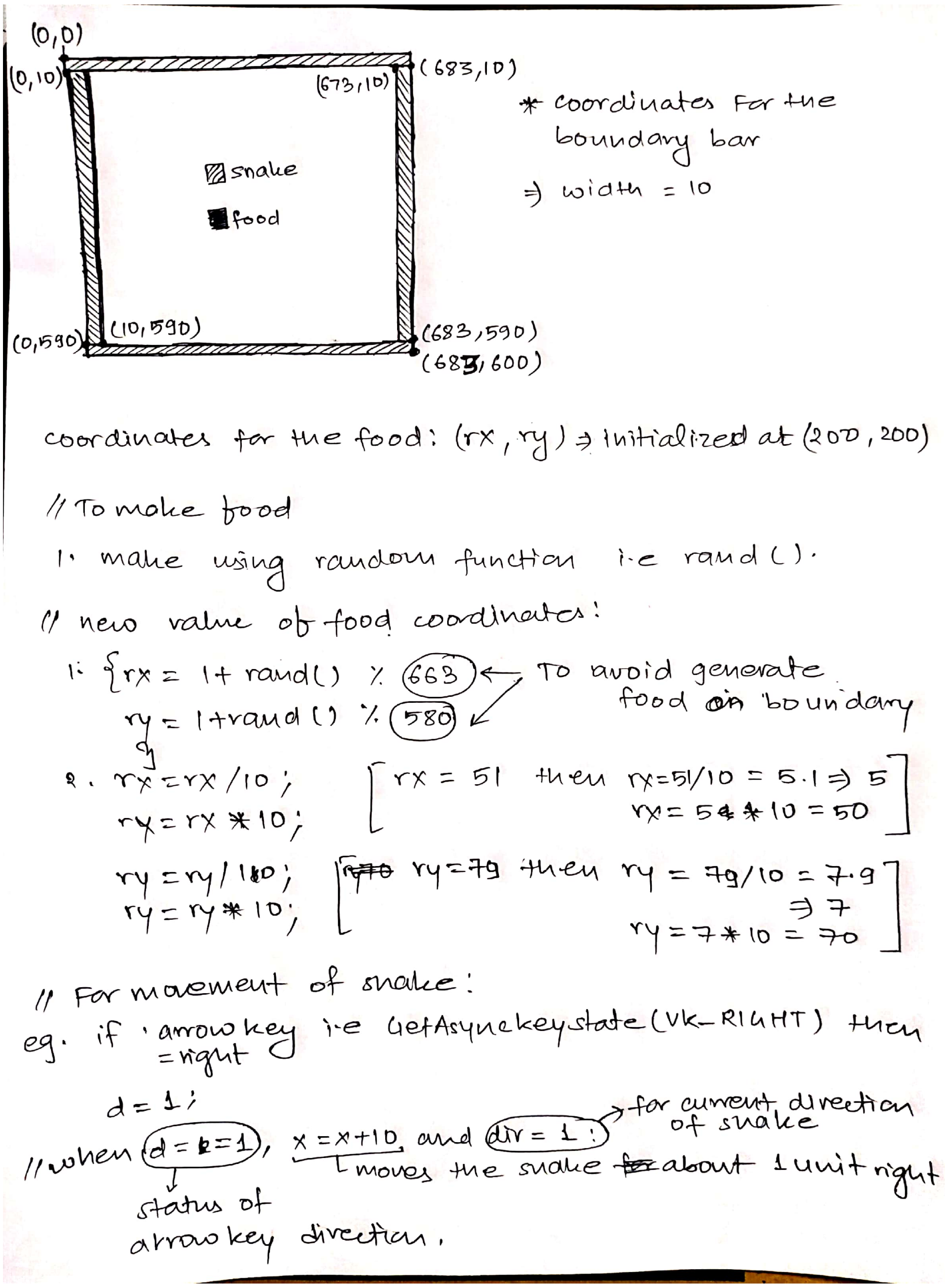
- **`outtextxy()`**: The `outtextxy()` function is used to display text at a specified position *(x, y)* on the screen.

  Syntax:

  ```
  void outtextxy(int x, int y, char *string);
  ```

3. Methodology

3.1 Design


(0,0)
(0,10)
(673,10)
(683,10)
▨ snake
■ food
(0,590)
(10,590)
(683,590)
(68~~3~~, 600)

* coordinates for the boundary bar

⇒ width = 10

coordinates for the food: (rx, ry) ⇒ initialized at (200, 200)

// To make food

1. make using random function i.e rand().

// new value of food coordinates:

```
1: {rx = 1+ rand() % (663) ← To avoid generate
    ry = 1+rand() % (580) ←       food ~~on~~ boundary

2. rx=rx/10;          [rx = 51  then rx=51/10 = 5.1 ⇒ 5
   rx=rx*10;                         rx= 5~~4~~*10 = 50 ]

   ry=ry/1~~8~~0;        [~~ry70~~ ry=79 then ry = 79/10 = 7.9
   ry=ry*10;                                    ⇒ 7
                                     ry=7*10 = 70 ]
```

// For movement of snake:

eg. if 'arrow key = right' i.e GetAsynckeystate(VK_RIGHT) then

d = 1;

//when (d = ~~k~~=1), x = x+10 and (dir = 1:) → for current direction of snake

↳ status of arrow key direction.

↳ moves the snake ~~for~~ about 1 unit right

Figure 3.1: Design of game

3.2 Source Code

```
// Include necessary libraries
```

```
#include<stdio.h>
#include<time.h>
#include<windows.h>
#include<graphics.h>
#include<stdlib.h>

void gameover();
int  endfunk(int e);

int main()
{
    int gd=DETECT, gm, x=200, y=200, d=1, dir=1, rx=200, ry=200, c=0 , fx, fy;
    initgraph(&gd,&gm,"");
    initwindow(683,600,"SNAKEMAN"); //window resolution and name
    delay(1000);
    srand(time(NULL)); //for starting point of food to be random every time
    setfillstyle(1,2);
    for(;;)
    {
        setfillstyle(1,0);//screen clear to black
        bar(0,0,683,600);
        setfillstyle(1,2);//boundary color for snake

        //boundary bars
        bar(0,0,683,10);
        bar(0,590,683,600);
        bar(0,10,10,590);
        bar(673,10,683,590);

        //to make food
        if(x == rx && y == ry)
        {
            c = c + 1; //food  counter for  score

            setfillstyle(1,0); //color to erase the previous food
            bar(rx,ry,rx+10,ry+10); //previous food
            do
            {
                rx = (1 + rand() % 663);
                ry = (1 + rand() % 580);
            }while(getpixel(rx,ry) != 0 && rx > 10 && ry > 10);
            rx = rx / 10;
            rx = rx * 10;
            ry = ry / 10;
            ry = ry * 10;
            setfillstyle(1,14); // color for when snake reach the food
        }
        setfillstyle(1,14); // color for when new food is displayed
        bar(rx,ry,rx+10,ry+10); //new food
        setfillstyle(1,2);

        //arrow keys
        if(GetAsyncKeyState(VK_RIGHT))
        {
            d = 1;
        }
        else if(GetAsyncKeyState(VK_LEFT))
        {
            d = 2;
        }
        else if(GetAsyncKeyState(VK_UP))
        {
            d = 3;
        }
        else if(GetAsyncKeyState(VK_DOWN))
        {
            d = 4;
        }
        else
        {
            d = 0;
        }

        switch(d)
        {
        case 0: //when no arrow key is pressed
            if(dir == 1)
            {
                x = x+10;
            }
            else if(dir == 2)
            {
                x = x-10;
            }
            else if(dir == 3)
            {
                y = y - 10;
            }
            else if(dir == 4)
            {
                y = y + 10;
            }

            else
            {
                d = 0;
            }
            break;

        case 1: //right key
            x = x + 10;
            dir = 1;
            break;

        case 2: //left key
            x = x - 10;
            dir = 2;
            break;

        case 3: //up key
            y = y - 10;
            dir = 3;
            break;

        case 4: //down key
            y = y + 10;
            dir = 4;
            break;

        }
        bar(x,y,x+10,y+10); //next move of snake
        delay(100);
        if(x >= 683 || x <= 0 || y <= 0 || y >= 600)// when snake cross the boundary
        {
            cleardevice();
            gameover();
            delay(2000);
            endfunk(c);
            break;
        }
    }
}

void gameover()
{
    setfillstyle(1,WHITE);
    settextstyle(3,0,5);
    outtextxy((getmaxx()/2)-130,(getmaxy()/2)-50,"Game Over");
}

int endfunk(int e)
{
    e=e-1;
    system("cls");  //to clear the console window
    printf("You died outside the boundary!!!\n");
    printf("Your score is : %d\n", e);
}
```

4. Result

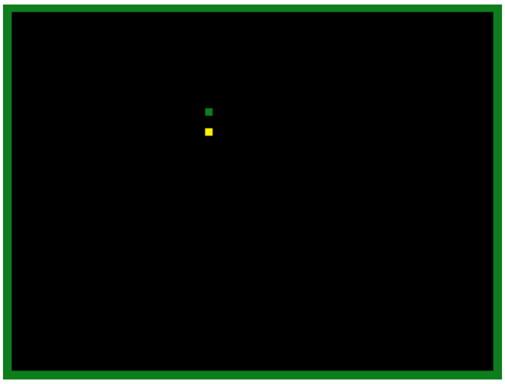


Figure 4.1: Start Screen 1

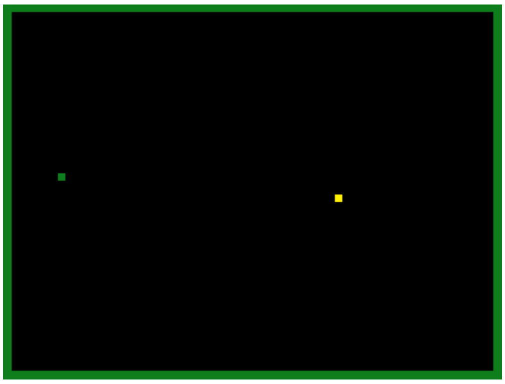
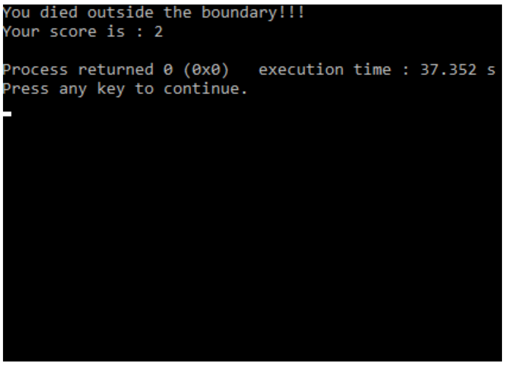


Figure 4.2: Start Screen 2

Game Over

Figure 4.3: Game Over Screen



```
You died outside the boundary!!!
Your score is : 2

Process returned 0 (0x0)   execution time : 37.352 s
Press any key to continue.
```

Figure 4.4: Result Screen