HUNGRY SNAKE

PROJECT REPORT

Programming concept
Group number -08
D.M.T.Sandunika
L.P.T.Mithara
N.H.Muthukumarana
W.G.H.P.Adithya

Content **page**

1) Introduction

- Name of the game – **HUNGRY SNAKE**
- 'HUNGRY SNAKE' is a simple text-based and single player game developed using C++, also with the use of ASCII characters.
- This is a classic game where the player controls a snake to eat fruits while avoiding collisions
- This game is built using features such as real time input handling, coordinate-based movements.

2) Game description

2.1) game objectives

- Player have to control snake to eat food , increase score and avoid collisions with borders, walls and snake's own body.
- Player have to play with limited number of lives of snake

2.2) game rules

- Snake should avoid hitting walls and itself.
- There are two types of fruits (normal fruit and big fruit).
- eating normal fruit increases score by 10 and eating big fruit increases score by 20.
- Big fruit appears as a bonus fruit after every 5 successful eats of normal fruit.
- Game layout change when score hits 200, 400 and 600 (game has 3 layouts) and each layout gives harder gameplay.
- Game has 3 lives and lives decrease when snake hits walls or its own body and game ends when lost all 3 lives of snake.

2.3) game features

- Game play area, snake, big fruit, walls are printed using ASCII characters for better view and improve user experience.

- 3 default layout defined in the game and change them according to score and each layout increases hardness of game play.
- Bonus fruit appeared more than 4 seconds on the screen.
- Fruit is flickering for increase it's visibility.
- High score system for every gameplay.
- When decrease a life, game start again.(when all 3 lives are lost game over).
- An instruction panel appear on right side of the game play area for player help and display high score, current score, life count and controls.
- Restart and exit options appear when game is over.
- Game speed increase by 2 every time after snake eat fruit.

3) Game play

3.1) system requirements

- Windows operating system.
- C++ compiler.
- Make sure that the compiler supports C++ 11(or GNU C++ 11) standards or later for smooth gameplay.

3.2) starting game and controls

- After loading the start menu, please choose 1 to start the game and 2 to read the instructions. (It is better to read the instructions well before start to play).
- W - move up
- A - move left
- S - move down
- D - move right
- R - restart game
- X – exit from game

3.3) gameplay flow

- Launch the game

- Load intro screen as displaying “ BY GROUP 08”
- Appear the start menu and select 1 for start, 2 for instructions and 3 for quit game.
- After starting game control snake and trying to eat fruit.
- Survive all 3 layout levels without losing 3 lives.
- After game over, press R to restart and X to exit.

4) Code design and structure

4.1) code structure

- Source code of the game is built up with the main part of the program and different functions.
- These functions are specified to do a necessary part of the game play and game logic

4.2) code design

- Designed with 13 different functions and the main part of the program that integrates these functions.
- 13 functions are specified to do the following:
    1. Set the cursor position according to the given coordinates.
    2. Make cursor invisible to better view and experience.
    3. Set the game console size to given parameters.
    4. Draw the outline of the game area with given parameters.
    5. Appear the fruit on game screen.
    6. Generate the walls in layouts.
    7. Draw elements in game (walls, fruits, snake).
    8. Handle inputs of the game.
    9. Setup of the game ready to play.
    10. Entire game logic (snake moves, random positions, life count etc.).
    11. Play the game according to game logic with game setup.
    12. Show the intro screen.
    13. Load the instructions.

5) Challenges and solutions

5.1) challenges and current solutions

5.1.1) As this was a group project, one of the main challenges we faced was how to develop the program collaboratively as a team of four members, especially since we were not working on the same computer.

We had to figure out:

- How to divide program into parts.
- How to integrate those parts successfully at the end.

Solution:
We discussed as a team and decided to divide the game program to 13 different functions that handle the entire game and that 13 functions made by 4 members.
At the end we integrated all these functions into a main program and made our final program.

5.1.2) As this is a snake game, the snake size should be increased while program is running, so we got a challenge that how to make snake that can change its size while running the program.

Solution:
Array is the necessary solution for making snake body, but normal array can't change its size while running the program. So we used **dynamic array** specifically the **vector** data structure from C++'s Standard Template Library. This allows us to add segments or remove segments from snake body while running the program.

5.1.3) We had to measure the bonus food appeared time duration while running the program.

Solution:

We solved this challenge by using time-based logic in C++. Specifically, we used the functions available in the <ctime> header, such as: GetTickCount()

5.1.4) The blinking console cursor can distract the player during the gameplay. So we needed to hide the console cursor during the gameplay

Solution:
We used Windows-specific API functions to hide the blinking console cursor during gameplay. This is done by using the SetConsoleCursorInfo() function provided in <windows.h>, by setting the cursor visibility flag to false.

5.2) Improvements and What We Would Do Differently

5.2.1) Early planning of code structure

At the beginning, we did not clearly define how each function would interact with others. This caused confusion during integration and delayed progress.

If we could do it again,
we would start by designing a clear architecture with detailed function prototypes and how those functions will be integrated at the end before any coding begins. This would make collaboration smoother.

5.2.2) Use modern IDE and GitHub for better collaboration

We developed the project using **Dev C++**, which is a simple and Lack of modern features.

If we could do it again,

We would use modern IDE like visual studio code and use git hub to manage Our code collaboratively. This would allow each member to work on separate branches, track changes, and merge updates smoothly, improving teamwork and reducing errors.

5.2.3) Use advanced graphics

Our game is text-based, limited to console output using ASCII characters. While functional, it restricts visual quality and animation smoothness.

If we could do it again,

We would use C++ graphic library such as raylib to develop graphical version.

5.2.4) Earlier and Module-Wise Testing

Most of our testing happened after all members had submitted their parts, which made bug fixing difficult.

If we could do it again,
we would test each function/module separately as it's built and then test after every integration. This would reduce bugs, and improve team productivity.

6) Conclusion

This Snake Game project allowed us to practically apply our programming knowledge while learning valuable lessons in teamwork, planning, coding, bug fixing and problem-solving. Also we experienced facing challenges in coordination and implementation, finally we built a functional, interactive game using C++ programming. The project not only improved our technical skills but also enhanced our ability to collaborate effectively as a team.

7) References

https://www.geeksforgeeks.org/cpp/vector-in-cpp-stl/

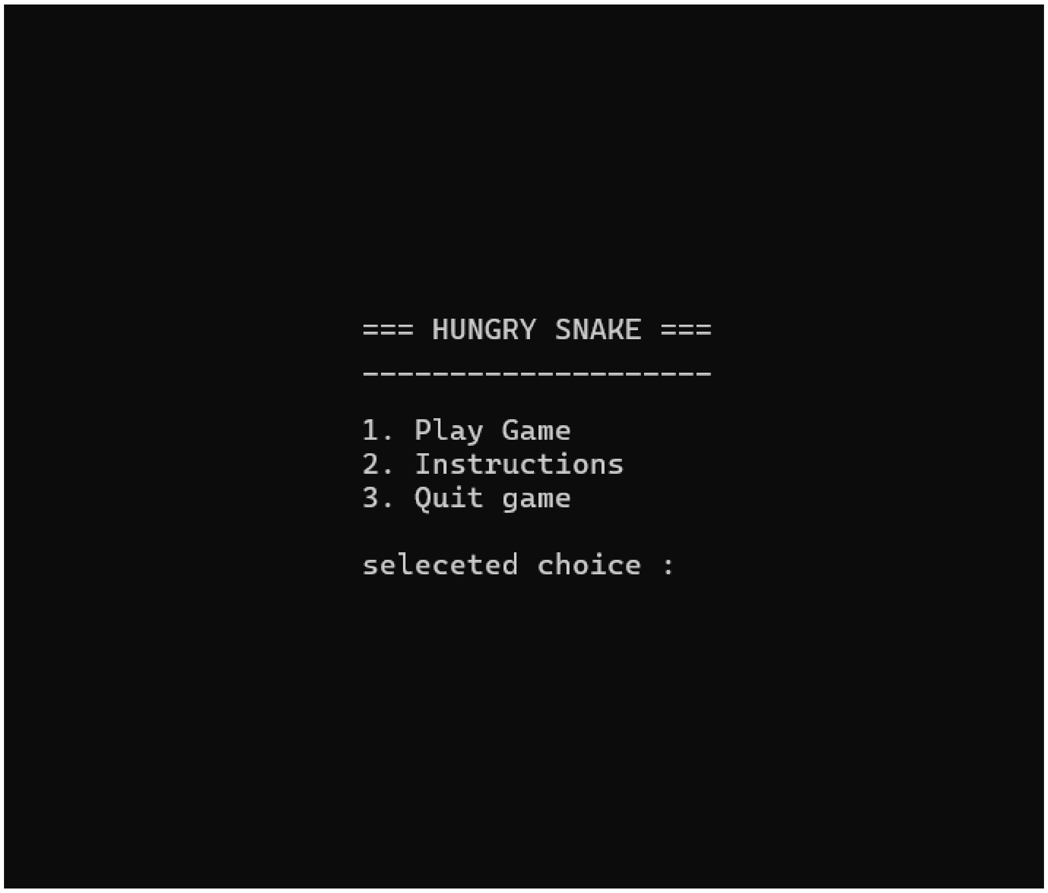
https://www.w3schools.com/cpp/default.asp

https://cplusplus.com/reference/

8) Appendix

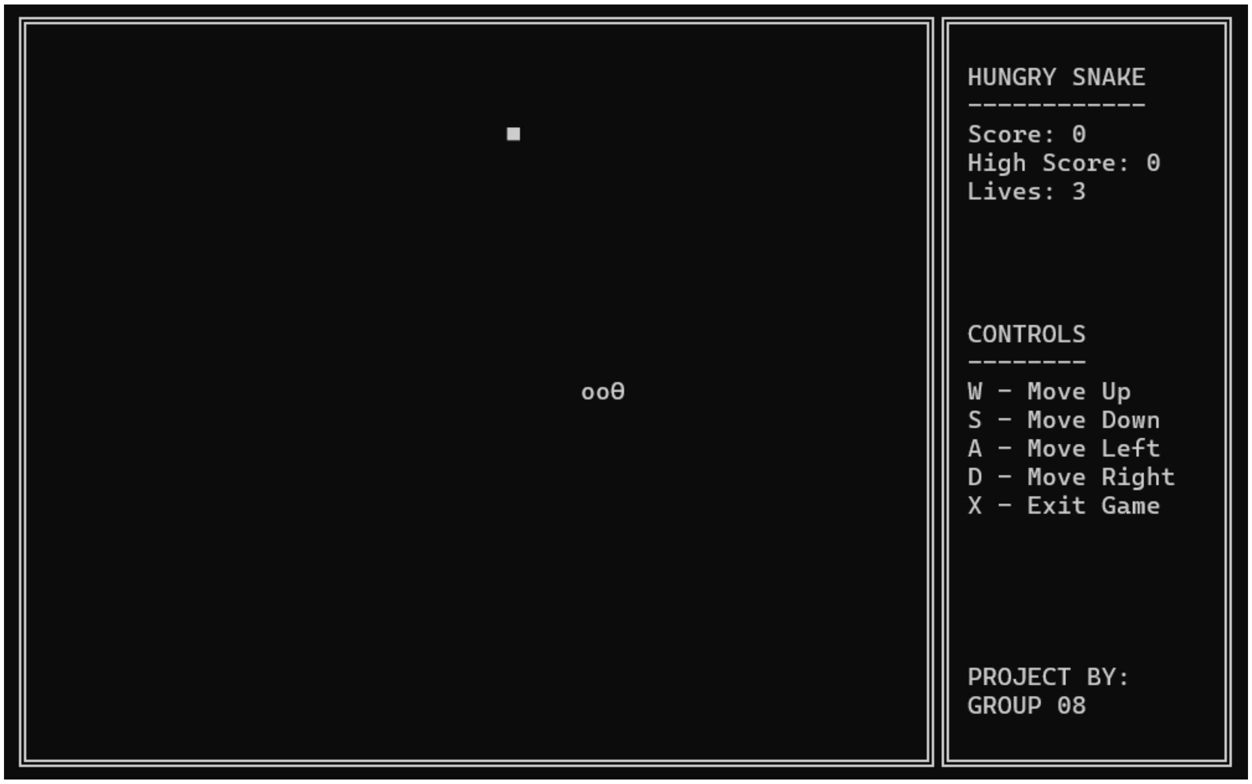
8.1) Appendix A – appearance of game screen

7.1.1) start menu



```
=== HUNGRY SNAKE ===
--------------------

1. Play Game
2. Instructions
3. Quit game

seleceted choice :
```

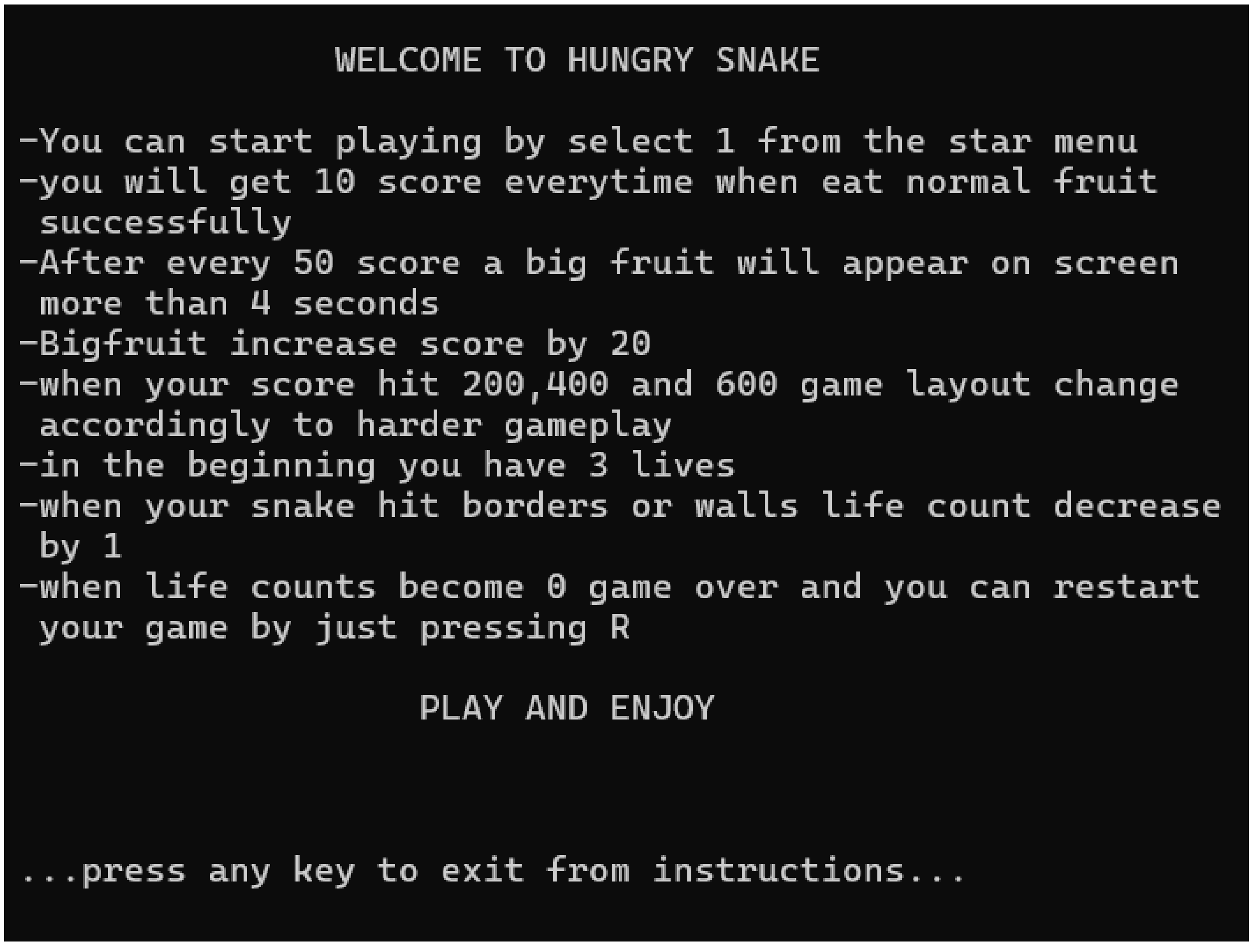
8.1.2) game play area and info-panel



```
■

ooθ
```

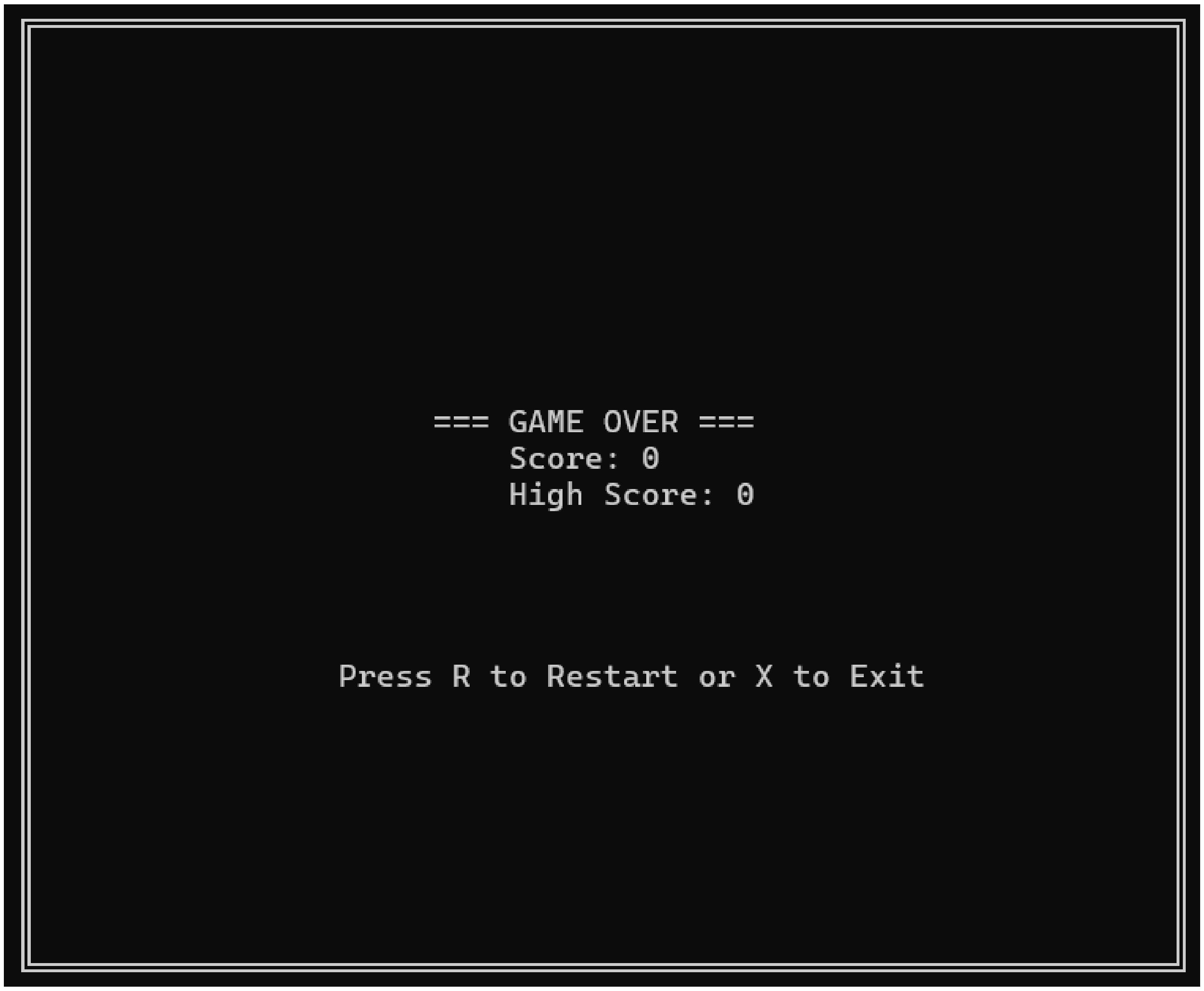
```
HUNGRY SNAKE
------------
Score: 0
High Score: 0
Lives: 3

CONTROLS
--------
W - Move Up
S - Move Down
A - Move Left
D - Move Right
X - Exit Game

PROJECT BY:
GROUP 08
```

8.1.3) instruction screen



```
               WELCOME TO HUNGRY SNAKE

-You can start playing by select 1 from the star menu
-you will get 10 score everytime when eat normal fruit
 successfully
-After every 50 score a big fruit will appear on screen
 more than 4 seconds
-Bigfruit increase score by 20
-when your score hit 200,400 and 600 game layout change
 accordingly to harder gameplay
-in the beginning you have 3 lives
-when your snake hit borders or walls life count decrease
 by 1
-when life counts become 0 game over and you can restart
 your game by just pressing R

                   PLAY AND ENJOY



...press any key to exit from instructions...
```

8.1.4) game over screen



```
=== GAME OVER ===
    Score: 0
    High Score: 0


Press R to Restart or X to Exit
```

8.2) Appendix B – full source code of the game

#include<iostream> //for handling input and output

#include<conio.h> //for handling console inputs and outputs of the game (use inbuilt functions like khbit(),getch())

#include<Windows.h> //for windows API functions (sleep(),setconsolesize(),setconsolewindow(),setconsolecursorinfo(),setconsolecursorposition())

#include<vector> //for using dynamic arrays (dyabamic arrays used for snake body and walls which are printed during game layout)

```
#include<ctime>                    //for include current time(to measure time )(measure time between fruit appearing etc )
using namespace std;

//constants for game area and info panel dimensions
const int width = 60;             //game screen width set to 60
const int height = 25;            //game screen height set to 25
const int infoPanelWidth = 20;   //info panel width set to 20(info panel height same as game screen height)

//declaration of the variables using in the game
int x, y, fruitX, fruitY;        //snake head position and fruit position
int score, highScore =0;         //current score and highscore
int Lives;                       //number of lives
int direcX=0, direcY=0;          //direction of snake movement
int numFruits =0;                //number of fruits eat
int gameSpeed;                   //gap between frames
int currentLayout=0;             //current wall layout
DWORD bigFoodStartTime =0;       // big food appeared time
DWORD lastFlickerTime=0;         //last time fruit flickered
bool bigFood=false;              //is big food active
bool gameOver = false;           //is game is over
bool showFruit= true;            //is fruit visible

//structure of coordinates
struct Coord{ //coord is the name of this structure
   int X;  // x coordinate
   int Y;  // y coordinate
};

//declaration of dynamic arrays using vector
vector<Coord> snakeBody;        //dynamic array to store snake body coordinates
vector<Coord>walls;             //dynamic array to store wall coordinates

//function prototypes
```

```
void setCursorPosition(int x, int y);
void invisibleCursor();
void setConsoleSize();
void drawOutline(int x,int y,int w,int h);
void appearFruit();
void generateWalls(int layout);
void draw();
void input();
void setup();
void logic();
void playGame();
void introscreen();
void loadInstructions();



//main program
int main() {
   srand((unsigned)time(0)); // random number generator
   invisibleCursor();           // hide cursor
   setConsoleSize();         // set console window size
   introscreen();           // show intro screen

   while(1){
      system("cls");   //clear early screen before desplaying menu

      setCursorPosition(width / 2 - 10, height-16); //move cursor to the given position
      cout<<"=== HUNGRY SNAKE ===";              // display the name of the game (HUNGRY SNAKE)
      setCursorPosition(width / 2 - 10, height-15);
      cout<<"____________________";
      setCursorPosition(width / 2 - 10, height -13);
      cout<<"1. Play Game";   // display first option of menu (Play Game)
      setCursorPosition(width / 2 - 10, height -12);
      cout<<"2. Instructions"; // display second option of menu (Instructions)
```

```
        setCursorPosition(width / 2 - 10, height -11);
        cout<<"3. Quit game";   // display third option of menu (Quit Game)
        setCursorPosition(width / 2 - 10, height -9);
        cout<<"seleceted choice :";  // display the selected option by the user

        char choice = getchar(); // get user input

        if(choice == '1') {
           playGame(); //  if choice is 1 then start the game
        } else if(choice == '2') {
           loadInstructions(); // if choice is 2 thenload instructions
        } else if(choice == '3') {
           break; // if choice is 3 then exit the game
        } else {
           continue; // if input is invalid repeat the process untill a correct option is selected
        }

    }
return 0; // end of the main function
}



//function to show introscreen
void introscreen(){
    system("cls"); // clear screen
    setCursorPosition(width / 2 - 4, height -16);
    cout<<"B";   //display B
    Sleep(250);  //sleep screen for 250 miliseconds
    cout<<"Y ";  //display Y
    Sleep(250); //sleep screen for 250 miliseconds
    cout<<"G";  //display G
    Sleep(250);  //sleep screen for 250 miliseconds
    cout<<"R";  //display R
    Sleep(250); //sleep screen for 250 miliseconds
    cout<<"O";  //display O
    Sleep(250); //sleep screen for 250 miliseconds
    cout<<"U";  //display U
    Sleep(250);  //sleep screen for 250 miliseconds
    cout<<"P";  //display P
    Sleep(250);  //sleep screen for 250 miliseconds
    cout<<" 08"; //display 08
    Sleep(1000);   //sleep screen for 1000 mili seconds
    system("cls"); // clear console screen
}


//function to load instructions
void loadInstructions() {
    system("cls"); // clear screen
    FILE* f1; // f1 is a file pointer to read instructions
    char c;
    f1= fopen("instructions.txt", "r"); // open instructions file in read mode
    setCursorPosition(15,2);
    while((c=getc(f1))!=EOF){ // read file character by character
        cout<<c; // print character
    }
    fclose(f1); // close file
    getch(); // wait for user to press a key
}


//function to set the position of console cursor
void setCursorPosition(int x, int y) {
    COORD coordinate;
    coordinate.X = x;              // Set the X coordinate
    coordinate.Y = y;              //Set the Y coordinate
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinate);   // Move cursor
}


// function to cursor invisible
void invisibleCursor(){
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE); // Get the console handle
    CONSOLE_CURSOR_INFO cursorDetail={1,false};  // make cursor invisible  and size to 1 (minimum size)
    SetConsoleCursorInfo(console, &cursorDetail);      // Hide  console cursor
}


//function to set console size
void setConsoleSize(){
    HWND console = GetConsoleWindow();// console is a handle variable to the console window
    RECT rectangle; // RECT is a structure which holds the position and size of a rectangle
    GetWindowRect(console, &rectangle);                              // Get the current size of the console window
    MoveWindow(console, rectangle.left, rectangle.top, 900, 600, TRUE);           // Set the size of the console window
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);   // Get the console handle
    COORD bufferSize={(SHORT)(width+infoPanelWidth+2),(SHORT)(height+2)};       // Set the buffer size of the
console (buffer is the area where the output of the program is stored in memory)
    SetConsoleScreenBufferSize(hConsole,  bufferSize);        // Set the console buffer size
    SMALL_RECT window={0, 0, (SHORT)(width+infoPanelWidth+1), (SHORT)(height+1)}; // window size shoud
be like this
    SetConsoleWindowInfo(hConsole, TRUE, &window);                // Set the console window size according to the
given width and height
}


//function to draw game area outline
void drawOutline(int x,int y,int w,int h){
    setCursorPosition(x, y); // cursor position set according to x and y coordinates which are passed to the function as
parameters
```

```
    cout << (char)201; // print the top-left corner of the outline using ASCII charactor 201
    for (int i = 0; i < w - 2; i++)  //loop for printing the top horizontal outline
    {
        cout << (char)205; // printing the top horizontal outline using ASCII charactor 205
    }
    cout << (char)187; //  printing top-right corner of the outline using ASCII charactor 187
    for (int i = 1; i < h - 1; i++) { // loop for printing the vertical outline
        setCursorPosition(x, y + i); // x coordinate of cursor position is constant and y coordinate is incremented by i
        cout << (char)186; // printing left vertical outline using ASCII character 186
        setCursorPosition(x + w - 1, y + i); // x coordinate is incremented by w-1 and it is constant  and y coordinate is
incremented by i
        cout << (char)186; //printing right vertical outline using ASCII character 186
    }
    setCursorPosition(x, y + h - 1); // move cursor to the bottom left corner of the outline


    cout << (char)200; //printing the bottom-left corner of the outline using ASCII character 200
    for (int i = 0; i < w - 2; i++) // loop for printing the bottom horizontal outline
    {
        cout << (char)205; // printing the bottom horizontal outline using ASCII character 205
    }
    cout << (char)188; // printing bottom-right corner of the outline using ASCII character 188
}



//function to generate walls
void generateWalls(int layout) {
    walls.clear(); // clear current walls
    if (layout == 1) {          //check that layout is 1
        for (int i = 10; i < width - 10; i++) {  //loop for drawing horizontal walls from (x=10) to (x = width-10)
            walls.push_back({ i, height / 2 });   // horizontal wall in the middle(y coordinate is a constant and x coordinate
is incremented by i)
        }
    } else if (layout == 2) {     //check that layout is 2
        for (int i = 5; i < height - 5; i++) {  // loop for drawing vertical walls from (y=5) to (y = height-5)
```

```
            walls.push_back({ 15, i });  //draw left vertical wall (x coordinate is a constant and y coordinate is incremented
by i)

            walls.push_back({ width - 16, i });  // draw right vertical wall (x coordinate is a constant and y coordinate is
incremented by i)

        }

    } else if (layout == 3) {  //check that layout is 3


        int X = width / 4, Y = height / 4, w = width / 2, h = height / 2;  // (X,Y) is top left corner coordinates, w = width,
h = height of wall


        for (int i = X; i < X + w; i++) { //loop for drawing top and bottom walls of layout 3

            walls.push_back({ i, Y });       //draw top wall (y coordinate is constant and x coordinate is incremented by
i)(this draw a vertical line from X to X+w)

            walls.push_back({ i, Y + h - 1 }); // draw bottom wall (y coordinate is constant and x coordinate is incremented
by i)(this draw a vertical line from X to X+w)

        }

        for (int i = Y + 2; i < Y + h - 2; i++) {  // loop for drawing left and right walls of layout 3 (keep little space between
walls so snake can go in and come out )

            walls.push_back({ X, i });         // draw left wall (x coordinate is constant and y coordinate is incremented by
i)(this draw a horizontal line from Y to Y+h-2)

            walls.push_back({ X + w - 1, i }); // draw right wall (x coordinate is constant and y coordinate is incremented
by i)(this draw a horizontal line from Y to Y+h-2)

        }

    }

}



//function to appear fruit randomly

void appearFruit() {

  bool value = false; //check if fruit position is valid

  while(!value) // loop until fruit position is valid

{

    fruitX = rand()% (width - (bigFood ? 1:0)); //generate random x coordinate for fruit(if big food it is 2x2 square, so
the possible area for x coordinate is width-1)

    fruitY = rand()% (height - (bigFood ? 1:0)); //generate random y coordinate for fruit(if big food it is 2x2 square, so
the possible area for y coordinate is height-1)

    value = true;  // make fruit position valid
    // check if fruit position is not same as snake body
  for (int i =0; i< snakeBody.size(); i++){
    Coord body = snakeBody[i];  //check each body  part of snake one by one
    if ((!bigFood && body.X == fruitX && body.Y==fruitY)||       //if normal fruit and (x,y) coordinates of snake and
(x,y) coordinates of fruit are same
        (bigFood && body.X >= fruitX && body.X <= fruitX + 1 && body.Y >= fruitY && body.Y <= fruitY + 1)) //if
big food and (x,y) coordinates of snake and (x,y) coordinates of big food are same(big food is 2x2 square so have to
check all 4 coordinates)
        {
        value = false;  // if fruit position is same as snake body, make it invalid( to avoid bug that snake can eat itself)
        break; // break the loop if fruit position is invalid
        }
  }

 for (int i = 0; i < walls.size(); i++) {
    Coord wall = walls[i]; //check each wall one by one
    if ((!bigFood && wall.X == fruitX && wall.Y == fruitY) || //if normal fruit and (x,y) coordinates of wall and (x,y)
coordinates of fruit are same
        (bigFood && wall.X >= fruitX && wall.X <= fruitX + 1 && wall.Y >= fruitY && wall.Y <= fruitY + 1)) //if
big food and (x,y) coordinates of wall and (x,y) coordinates of big food are same(big food is 2x2 square so have to
check all 4 coordinates)
        {
         value = false; // if fruit position is same as wall, make it invalid( to avoid bug that snake can eat wall)
         break; // break the loop if fruit position is invalid
        }
  }
}
if (bigFood) bigFoodStartTime = GetTickCount();         //if big food appeared, count the time when it appeared
}


// function to get user inputs
void input() {
    if (_kbhit()) {  // check if a key is pressed
        switch (_getch()) {   // get the pressed key
```

```
            case 'w': case 'W': if (direcY != 1) { direcX = 0; direcY = -1; } break; //when press 'w' or 'W' key, change direction to up (if not already moving down)

            case 's': case 'S': if (direcY != -1) { direcX = 0; direcY = 1; } break;  //when press 's' or 'S' key, change direction to down (if not already moving up)

            case 'a': case 'A': if (direcX != 1) { direcX = -1; direcY = 0; } break;  //when press 'a' or 'A' key, change direction to left (if not already moving right)

            case 'd': case 'D': if (direcX != -1) { direcX = 1; direcY = 0; } break; //when press 'd' or 'D' key, change direction to right (if not already moving left)

            case 'x': case 'X': gameOver = true; break;      // when press 'x' or 'X' key, set gameOver to true to exit the game

        }

    }

}




//function to draw ( outline/ walls/ snake / fruit/ info panel)

void draw(){

    drawOutline(0,0, width+2, height+2); // draw the outline of the game area,outline should be out of gameplay area, so (width+2) and (height+2) passed to the function

    drawOutline(width+2,0, infoPanelWidth, height+2); // draw the outline of info panel, info panel height is same as gameplay area height

    for(int i =0; i< height ;i++){

        setCursorPosition(1, i+1);  // move cursor to the begining of each drawing lines(snake body, fruit, walls)

        for(int j =0; j< width; j++){  //loop for drawing each character

             bool printed = false;

            if(snakeBody.size()&& i== snakeBody[0].Y && j == snakeBody[0].X){ // check snake head position according to snake body coordinates

                cout << char(233); // print snake head using ASCII character 233

                printed = true;  // make printed true to avoid printing other characters in the same position

            }else if(showFruit){

                                            if(!bigFood && i == fruitY && j == fruitX){   //check if x coordinate and y coordinate of fruit match to i and j

                cout << char(254); // print fruit using ASCII character 254

                printed = true;

                }else if(bigFood && (i== fruitY || i == fruitY + 1) && (j == fruitX || j == fruitX + 1)){ // check all posible coordinates of big food( bigfood is 2x2 square )

                cout <<"*"; // print big food using character '*'

                printed = true;
                }
            }
            if(!printed){
                for(int k=0;k<snakeBody.size();k++){  // loop for printing snake body
                if(snakeBody[k].X==j&& snakeBody[k].Y==i){ // check snake body position
                        cout<< "o"; // print snake body using character 'o'
                        printed = true;// make printed true to avoid printing other characters in the same position
                        break; // break the loop if snake body part is printed
                     }
                 }
            }
            if(!printed){
                for (int w = 0; w < walls.size(); w++) { // loop for printing walls
                    if (walls[w].X == j && walls[w].Y == i) { // check wall position
                        cout << char(176); // print wall using ASCII character 176
                        printed = true; // make printed true to avoid printing other characters in the same position
                        break; // break the loop if wall part is printed
                    }
                }
            }
            if(!printed){
                cout << " "; // print empty space
            }
        }
    }

 //printing the info panel
 int panelX= width+4,panelY=2; // define panelX and panelY for print content of info panel position
 // X coordinate of cursor position is (width+4) and it is constant and info panel is drawn after game area outline, Y
coordinate is incremented by +1 using panelY variable

 setCursorPosition(panelX,panelY++);cout<<"HUNGRY SNAKE";  // print the name of the game as title of info panel
 setCursorPosition(panelX, panelY++); cout << " -----------"; // print the line under the title
 setCursorPosition(panelX,panelY++);cout<<"Score: "<<score; // print current score
 setCursorPosition(panelX,panelY++);cout<<"High Score: "<<highScore; // print high score
 setCursorPosition(panelX,panelY++);cout<<"Lives: "<<Lives; // print number of lives
 panelY+=4; // making 4 line gap between score and controls
 setCursorPosition(panelX, panelY++); cout << "CONTROLS"; // print control title
 setCursorPosition(panelX, panelY++); cout << " -------"; // print line under the control title
 setCursorPosition(panelX, panelY++); cout << "W - Move Up"; // print control for moving up
 setCursorPosition(panelX, panelY++); cout << "S - Move Down"; // print control for moving down
 setCursorPosition(panelX, panelY++); cout << "A - Move Left"; // print control for moving left
 setCursorPosition(panelX, panelY++); cout << "D - Move Right"; // print control for moving right
 setCursorPosition(panelX, panelY++); cout << "X - Exit Game"; // print control for exiting the game
 setCursorPosition(panelX, 23); cout << "PROJECT BY:"; // print the group number (we are group 08)
 setCursorPosition(panelX, 24); cout << "GROUP 08"; // print the group number (we are group 08)
}


//function of game logic
void logic(){
   for(int i = snakeBody.size()-1; i>0; i--) snakeBody[i]=snakeBody[i-1]; //moves each body part to the position of the
previous part (as a snake body moves)
   x+=direcX; // change snake head position by user input direction
   y+=direcY;
   if(x<0|| x>= width || y<0|| y>= height){ // check that snake head hit the outline or not
      Lives--; // if snake head hits the outline, life count decreases by 1
      if(Lives<=0){ // check that life count is less than or equal to 0
         gameOver=true; // when life count becomes 0, game is over(after all 3 lives lost)
      }else setup(); // if life count is greater than 0, setup the game again
      return;
   };
   snakeBody[0].X=x; snakeBody[0].Y=y; // update snake head position
   for(int i=1; i< snakeBody.size();i++){ // loop for checking snake body collision , check each body part one by one
      if(snakeBody[i].X == x && snakeBody[i].Y == y){ // check that snake head hit the body or not
         Lives--; // when snake head hits the snake body, life count decreases by 1
         if(Lives<=0){ // check that life count is less than or equal to 0
            gameOver=true; // when life count becomes 0, game is over(after all 3 lives lost)
            }else setup(); // if life count is greater than 0, setup the game again
            return;
        }
    };
    for(int i=0; i< walls.size(); i++){ // loop for checking snake head collision with walls, check each wall one by one
        if(walls[i].X == x && walls[i].Y == y){ // check that snake head hit the wall or not
            Lives--; // when snake head hits the wall, life count decreases by 1
            if(Lives<=0){ // check that life count is less than or equal to 0
                gameOver=true; // when life count becomes 0, game is over(after all 3 lives lost)
            }else setup(); // if life count is greater than 0, setup the game again
            return;
        }
    };
    bool ate = false;// boolean value to chech that snake ate food
    if(!bigFood && x == fruitX && y == fruitY){ate = true;} // check snake head and fruit position are same and
making food ate value true
    if(bigFood && x>=fruitX && x <= fruitX + 1 && y >= fruitY && y <= fruitY + 1){
        ate = true; // check snake head and big food position are same and making food ate value true
    }
    if(ate){
        numFruits++; // when snake ate a fruit, fruit count eaten  increases by 1
        score+=(bigFood ? 20 : 10); // score increase ( bigfood= 20 score , normal fruit = 10 score)
        if(score > highScore) highScore = score; // check  current score  greater than high score and update high score
        snakeBody.push_back({-1,-1}); // add new body part to snake when snake ate a fruit
        if(gameSpeed > 50){ gameSpeed-= 2; } //decrement game speed by 2 (maximum speed is 50)(game speed
increase when delay between frames decrease)
        bigFood = (numFruits % 5 == 0); // every 5 fruits eaten, big food appears
        appearFruit(); // appear fruit randomly(function that made above)
    }
    DWORD t = GetTickCount(); // get current time
    if(t-lastFlickerTime>400){ // if 400 milliseconds passed from last flicker
        showFruit = !showFruit; // make fruit invisible
        lastFlickerTime = t; // update last flicker time
    }
    if(bigFood && t - bigFoodStartTime>4000){ // if big food appeared more than 4 seconds(bigfood duration = t-
bigfoodstarttime)
        bigFood = false; // make big food invisible (after 4 seconds big food disappear)
        appearFruit();// normal fruit quickly appears after bigfood desappeared
    }
    int layoutNow= score/200; // get running layout by score
    if(layoutNow != currentLayout&& layoutNow<=3){ //change layout when score hits 200, 400 and 600
        currentLayout = layoutNow; // update current layout
        generateWalls(currentLayout); // generate walls by current layout
        appearFruit(); // make appear fruit after every layout changing, so fruit not appear on new walls
    }

}



//function of  game setup
void setup(){
    x= width /2; // set snake head x coordinate to middle of game area
    y= height /2; // set snake head y coordinate to middle of game area
    direcX = 1; direcY=0; // beginning of game snake moves to right
    snakeBody.clear(); // clear snake body
    for(int i=0; i<3; i++){ // add 3 body part to default snakebody
        snakeBody.push_back({x-i, y});
    };
    numFruits = 0; // set fruit eaten count to 0 when starting game
    bigFood = false; // set bigfood dont appear when starting game
    gameSpeed=150; // starting game speed set to 150( delay between frames is 150 milliseconds)(this is the lowest
speed of game)
    currentLayout=score/200; // change layout when score hits 200, 400 and 600
    if(currentLayout > 3) currentLayout = 3; // if score is greater than 600, set current layout to 3
    generateWalls(currentLayout); // generate walls according to current layout
    appearFruit(); // function of appearing fruit
    gameOver = false; // set game over to false when starting game
}
```

```
//function to play game
void playGame() {
   do {
      Lives = 3; // set life count to 3 when starting new game
      score = 0; // set score to 0 when starting new game
      setup(); // prepare game state




      while (!gameOver) { //whenever game is not over(in starting game is not over, so condition is true)
         draw();           // draw everything(outline/ walls/ snake / fruit/ info panel)
         input();          // take input from player
         logic();          // do game logic when play game
         Sleep(gameSpeed); // sleep the game loop for 'gameSpeed' milliseconds(controlling game speed)
      }


      system("cls"); // clear screen
      drawOutline(0, 0, width + 2, height + 2);  //draw outline when game is not running (when game over)
      setCursorPosition(width / 2 - 8, height / 2 - 1); cout << "=== GAME OVER ==="; //display game over message
around the middle of the screen
      setCursorPosition(width / 2 - 4, height / 2);     cout << "Score: " << score;  // display score
      setCursorPosition(width / 2 - 4, height / 2 + 1); cout << "High Score: " << highScore; // display high score (total
score of 3 lives)
      setCursorPosition(width / 2 - 13, height / 2 + 6); cout << "Press R to Restart or X to Exit"; //display message that
user can restart or exit from the game


      char ch; // character variable 'ch' for hold user input
      do {
         ch = _getch(); // get user input
      } while (ch != 'r' && ch != 'R' && ch != 'x' && ch != 'X'); // repeat do-while loop if user does not press x or X,
r or R
      if (ch == 'x' || ch == 'X') break; // if user presses x or X then break the loop for exit from game
   } while (true); // repeat loop for restart the game when user presses r or R}
```