2D SNAKE GAME

KORE SHIRISHA (2451-16-733-052)

SAYANNAGARI SAISHASHANK (2451-16-733-301)

BE ¾ SEM 1, SEC-1



SUBJECT: COMPUTER GRAPHICS

FACULTY INCHARGE: T. SUJANAVAN

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

MVSR Engineering College

(Sponsored by Matrusri Education Society, Estd. 1980)

Affiliated to Osmania University & Recognised by AICTE

Nadergul (P.O), Hyderabad - 501 510.

**2018-2019**

**ABSTRACT**

The project’s aim was to create a 2d snake game. This snake keeps on growing with the consumption of feed. The game ends with the snake either colliding with itself or with the window. This game is implemented in C language including standard glut library.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
|  | Introduction | 4 |
|  | System Design | 5 |
|  | Code | 6 |
|  | Screenshots | 17 |
|  | References | 20 |

1. **INTRODUCTION**

The following is an example game written in C using glut based on the game called ‘snake’ which has been re-emerged in recent years on mobile phones. It isn’t the greatest game, but it does give you an idea of what you can achieve with OpenGL, and perhaps the basis by which to extend the principles and create more interesting games of your own.

Let’s talk about game and its controls to move the snake use

* ‘UP ARROW’ for UP,
* ‘DOWN ARROW’ for DOWN,
* ‘LEFT ARROW’ for LEFT,
* ‘RIGHT ARROW’ for RIGHT.

The aim of the game is to collect the dots (food) and avoid obstacles (window and the snake itself). As you collect food, the snake gets longer, so increasing your chance to crashing yourself. If the snake crashes with obstacles you get score according to the length of the snake. There is no concept of lives. Once you hit an obstacle, that’s it, game over.

1. **SYSTEM DESIGN**

Alive

Snake

GAME IS GOING TO START

Void moveSnakeAuto()

Void Keyboard()

Void moveSnake()

Void display()

Food

Food

Void spawnFood()

Game over

1. **CODE**

// It is used to generate random values in the code

#include <ctime>

// It means Double ended queue

#include <deque>

#include <GL/glut.h>

// It is your course entry point to various constant, type and function declaration

#include <unistd.h>

#include <stdlib.h>

// A macro for unused variables (to bypass those pesky G++ warnings)

#define UNUSED

// Snake direction macros

#define UP 1

#define DOWN 2

#define LEFT 3

#define RIGHT 4

char title[] = "Snake Game";

bool gameOver=false;

int score=0;

int i =1;

float map\_half\_length = 30.0f;

int direction = DOWN;

int move\_speed = 100;

bool moved = false;

std::deque< std::deque<float> > part\_coords;

bool food\_available = false;

int food\_coords[2];

int growth\_stage = 0;

int growth = 2;

void spawnFood(){

if(!food\_available){

while(true){

bool collides = false;

// Produce a temporary random coordinate

int temp\_food\_coords[2] = { food\_coords[0] = 2 \* (rand() % ((int) map\_half\_length + 1)) - (int) map\_half\_length,

food\_coords[1] = 2 \* (rand() % ((int) map\_half\_length + 1)) - (int) map\_half\_length };

// Does it collide with the snake?

for(unsigned int a = 0; a < part\_coords.size(); a++){

if(temp\_food\_coords[0] == part\_coords[a][0] &&

temp\_food\_coords[1] == part\_coords[a][1]){

collides = true;

}

}

// If it doesn't collide with the snake, then make it the real food coordinates

if(collides == false){

food\_coords[0] = temp\_food\_coords[0];

food\_coords[1] = temp\_food\_coords[1];

food\_available = true;

break;

}

}

}

glLoadIdentity();

glTranslatef(food\_coords[0], food\_coords[1], -40.0f);

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_POLYGON);

glVertex2d( 1.0f, 1.0f);

glVertex2d( 1.0f, -1.0f);

glVertex2d(-1.0f, -1.0f);

glVertex2d(-1.0f, 1.0f);

glEnd();

}

void display(){

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

// The vertex order is clockwise

// The side order is front, back, left, right, top, bottom (if applicable)

// Loop over snake size and draw each part at it's respective coordinates

for(unsigned int a = 0; a < part\_coords.size(); a++){

glLoadIdentity();

glTranslatef(part\_coords[a][0], part\_coords[a][1], -40.0f);

glColor3f(0.0f, 1.0f, 0.0f);

glBegin(GL\_POLYGON);

glVertex2d( 1.0f, 1.0f);

glVertex2d( 1.0f, -1.0f);

glVertex2d(-1.0f, -1.0f);

glVertex2d(-1.0f, 1.0f);

glEnd();

}

spawnFood();

glutSwapBuffers();

if(gameOver)

{

char \_score[10];

itoa(score,\_score,10);

char text[50]="your score :";

strcat(text,\_score);

MessageBox(NULL,text,"GAME OVER",0);

exit(0);

}

}

void moveSnake(int new\_direction){

direction = new\_direction;

int last\_part = part\_coords.size() - 1;

std::deque<float> new\_head = part\_coords[last\_part];

if(direction == UP){

// Did we slither into ourself?

for(unsigned int a = 0; a < part\_coords.size(); a++){

if(part\_coords[0][0] == part\_coords[a][0] &&

part\_coords[0][1] + 2.0f == part\_coords[a][1]){

gameOver=true;

}

}

// Did we slither into a wall?

if(part\_coords[0][1] == map\_half\_length){

gameOver=true;

}

// Did we get food?

if(part\_coords[0][0] == food\_coords[0] &&

part\_coords[0][1] + 2.0f == food\_coords[1]){

growth\_stage++;

food\_available = false;

}

new\_head[1] = part\_coords[0][1] + 2.0f;

} else if(direction == DOWN){

// Did we slither into ourself?

for(unsigned int a = 0; a < part\_coords.size(); a++){

if(part\_coords[0][0] == part\_coords[a][0] &&

part\_coords[0][1] - 2.0f == part\_coords[a][1]){

gameOver=true;

}

}

// Did we slither into a wall?

if(part\_coords[0][1] == -map\_half\_length){

gameOver=true;

}

// Did we get food?

if(part\_coords[0][0] == food\_coords[0] &&

part\_coords[0][1] - 2.0f == food\_coords[1]){

growth\_stage++;

food\_available = false;

}

new\_head[1] = part\_coords[0][1] - 2.0f;

} else {

new\_head[1] = part\_coords[0][1];

}

if(direction == LEFT){

// Did we slither into ourself?

for(unsigned int a = 0; a < part\_coords.size(); a++){

if(part\_coords[0][0] - 2.0f == part\_coords[a][0] &&

part\_coords[0][1] == part\_coords[a][1]){

gameOver=true;

}

}

// Did we slither into a wall?

if(part\_coords[0][0] == -map\_half\_length){

gameOver=true;

}

// Did we get food?

if(part\_coords[0][0] - 2.0f == food\_coords[0] &&

part\_coords[0][1] == food\_coords[1]){

growth\_stage++;

food\_available = false;

}

new\_head[0] = part\_coords[0][0] - 2.0f;

} else if(direction == RIGHT){

// Did we slither into ourself?

for(unsigned int a = 0; a < part\_coords.size(); a++){

if(part\_coords[0][0] + 2.0f == part\_coords[a][0] &&

part\_coords[0][1] == part\_coords[a][1]){

gameOver=true;

}

}

// Did we slither into a wall?

if(part\_coords[0][0] == map\_half\_length){

gameOver=true;

}

// Did we get food?

if(part\_coords[0][0] + 2.0f == food\_coords[0] &&

part\_coords[0][1] == food\_coords[1]){

growth\_stage++;

food\_available = false;

}

new\_head[0] = part\_coords[0][0] + 2.0f;

} else {

new\_head[0] = part\_coords[0][0];

}

part\_coords.push\_front(new\_head);

if(!growth\_stage){

part\_coords.pop\_back();

} else if(growth\_stage == growth){

growth\_stage = 0;

} else {

score++;

growth\_stage++;

}

glutPostRedisplay();

}

void keyboard(int key, int x, int y){

UNUSED(x);

UNUSED(y);

switch(key){

case GLUT\_KEY\_UP:{

if(direction == LEFT || direction == RIGHT){

moved = true;

moveSnake(UP);

}

break;

}

case GLUT\_KEY\_DOWN:{

if(direction == LEFT || direction == RIGHT){

moved = true;

moveSnake(DOWN);

}

break;

}

case GLUT\_KEY\_LEFT:{

if(direction == UP || direction == DOWN){

moved = true;

moveSnake(LEFT);

}

break;

}

case GLUT\_KEY\_RIGHT:{

if(direction == UP || direction == DOWN){

moved = true;

moveSnake(RIGHT);

}

break;

}

}

glutPostRedisplay();

}

void initGL(){

glMatrixMode(GL\_PROJECTION);

gluPerspective(75.0f, 1, 0.0f, 35.0f);

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

}

void moveSnakeAuto(int value){

if(!moved){

UNUSED(value);

if(direction == UP){

moveSnake(UP);

} else if(direction == DOWN){

moveSnake(DOWN);

} else if(direction == LEFT){

moveSnake(LEFT);

} else if(direction == RIGHT){

moveSnake(RIGHT);

}

} else {

moved = false;

}

glutTimerFunc(move\_speed, moveSnakeAuto, 0);

}

void reshape(GLsizei width, GLsizei height){

UNUSED(width);

UNUSED(height);

// Make the window non-resizable so we don't have to worry about size changes

glutReshapeWindow(600, 600);

}

int main(int argc, char\*\* argv){

glutInit(&argc, argv);

glutInitWindowSize(600, 600);

glutCreateWindow("snake game");

MessageBox(NULL,"GAME IS GOING TO START","SNAKE GAME",0);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutSpecialFunc(keyboard);

glutTimerFunc(move\_speed, moveSnakeAuto, 0);

int initSize = 3;

// Specify the coordinates to each part of the snake

for(int a = 1; a <= initSize; a++){

std::deque<float> row;

row.push\_back(0.0f);

row.push\_back((map\_half\_length + 2.0f + (initSize \* 2)) - (a \* 2));

part\_coords.push\_front(row);

}

srand(time(NULL));

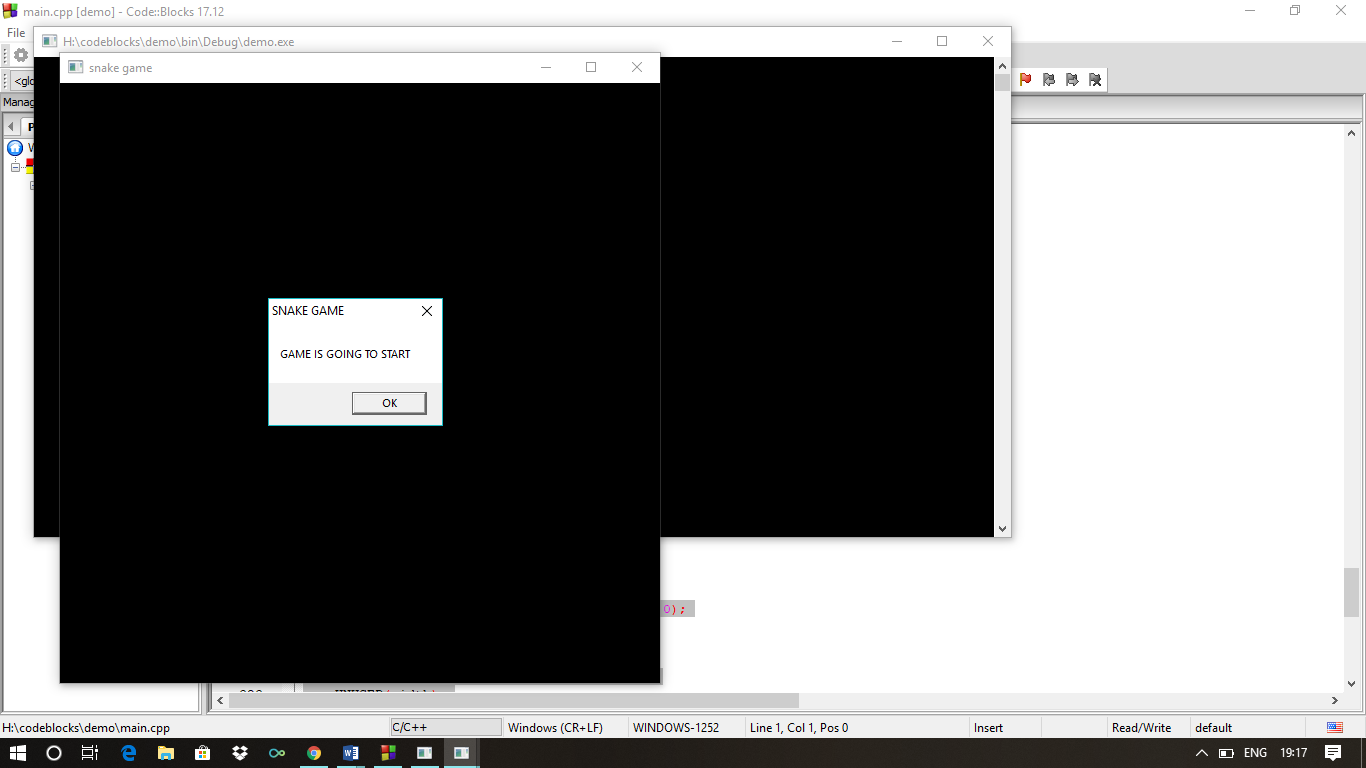
initGL();

glutMainLoop();

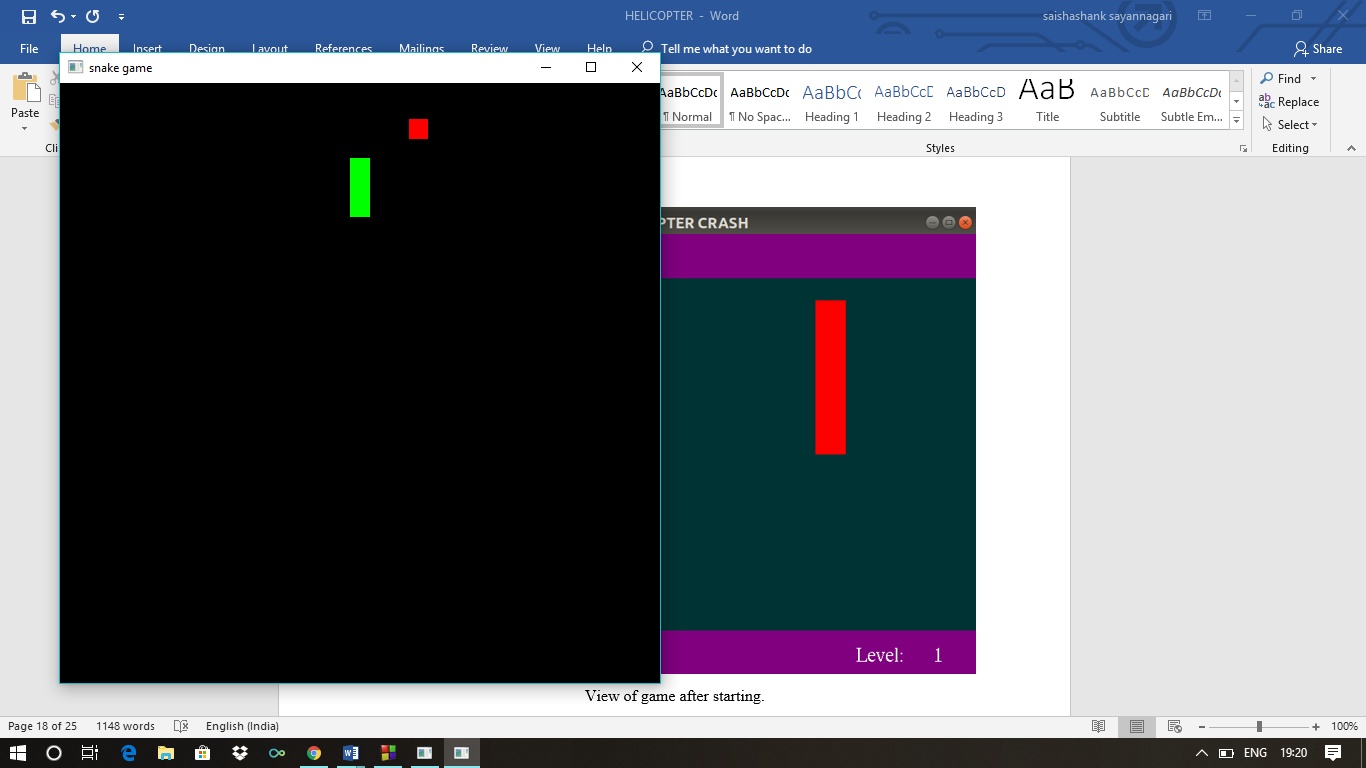
return 0;

}

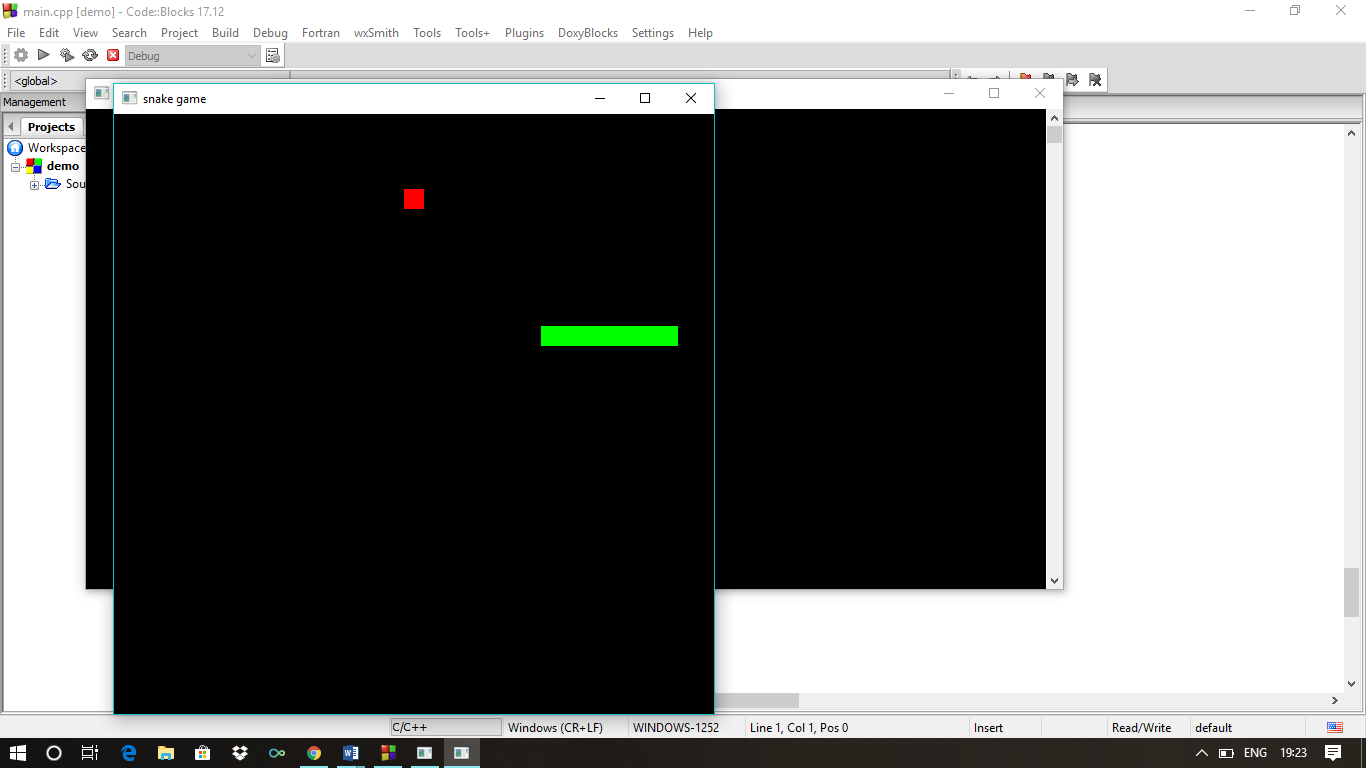
1. **SCREENSHOTS**

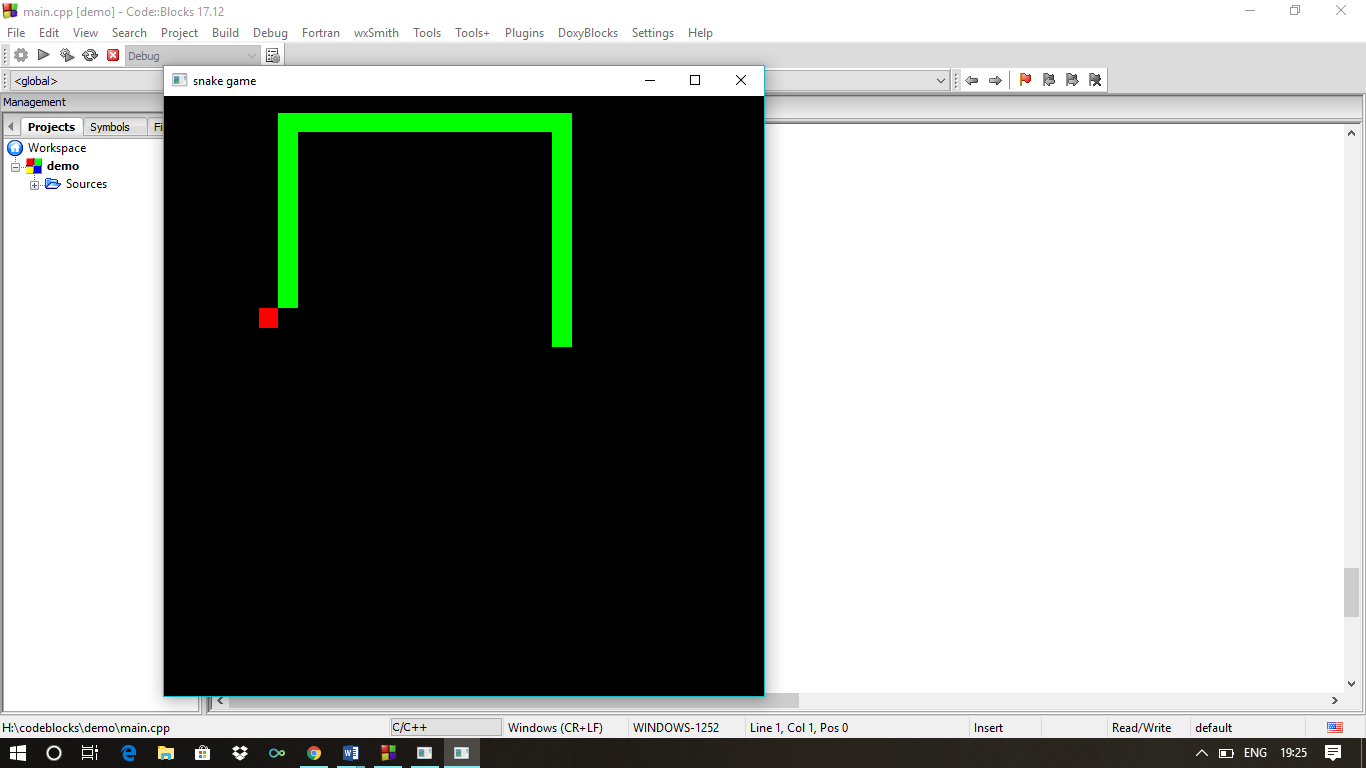


Start of the game. Welcome screen

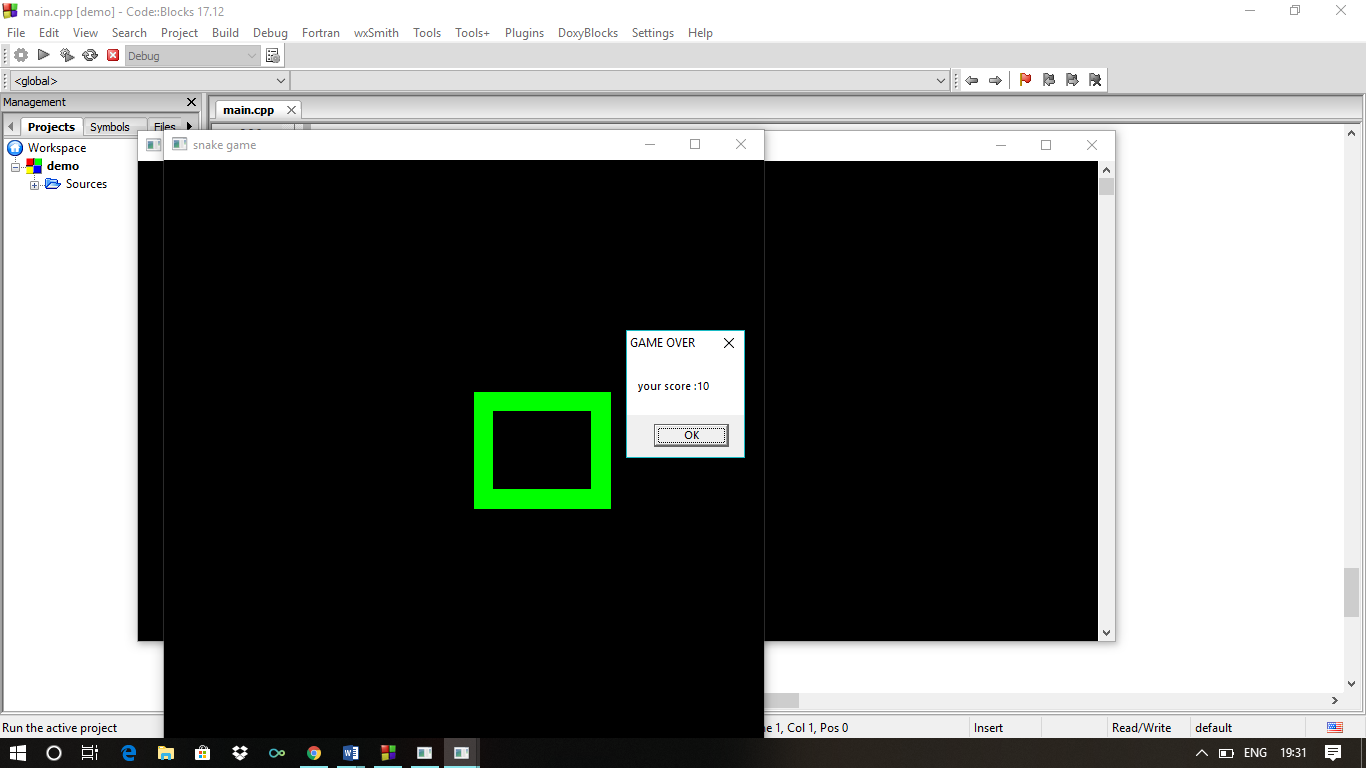


View of game after starting.

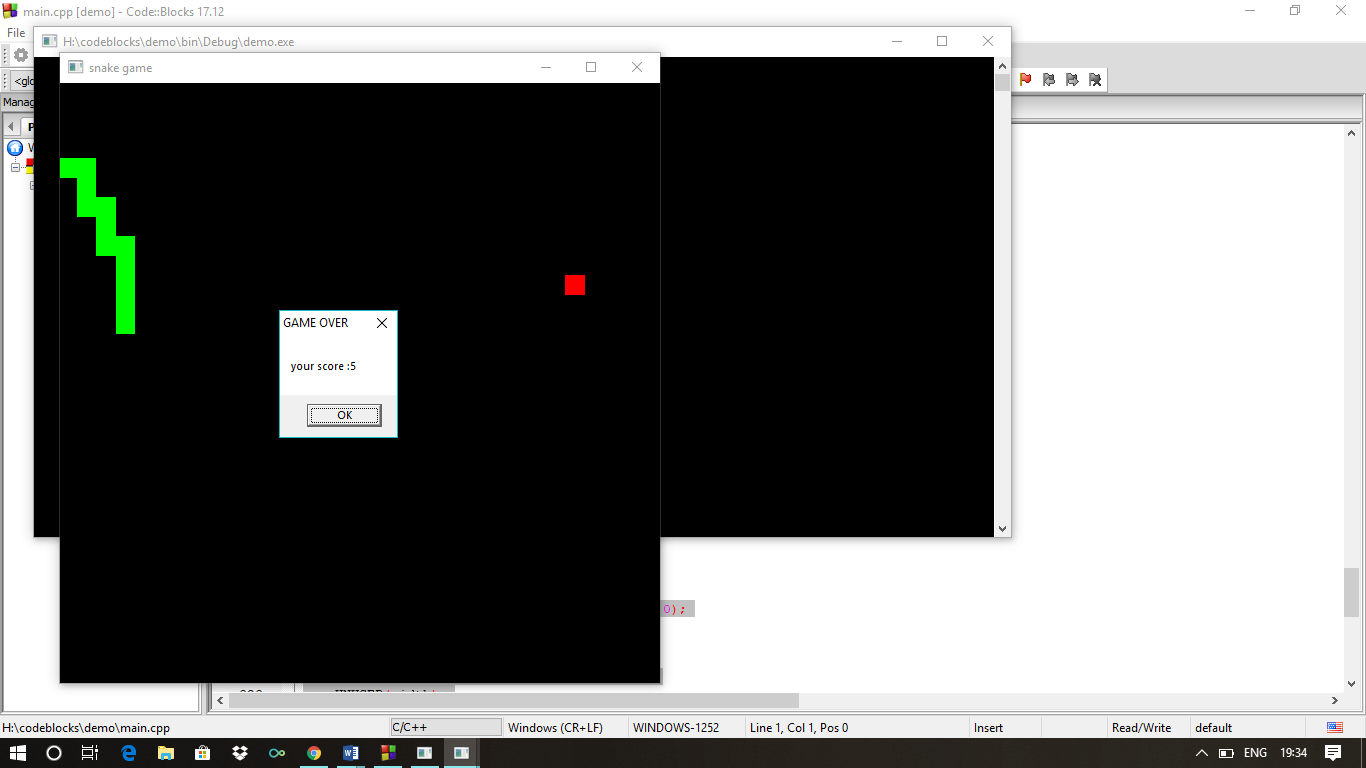




Size of the snake is increasing by consuming food



Snake colliding with itself. Game over



Snake collides with window. Game over.

**REFERENCES**

1. <https://www.opengl.org/resources/libraries/glut>
2. <https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glOrtho.xml>
3. <https://www.github.com/Uttamkr/Snake-game>
4. Interactive Computer Graphics Author: Edward Angel
5. Computer Graphics using OpenGL Author: Francis S Hill jr.