# **R.V. COLLEGE OF ENGINEERING**

# **BENGALURU – 560059**

(Autonomous Institution Affiliated to VTU, Belagavi)

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



“ **OpenGL Snake Game”**

**COMPUTER GRAPHICS (12CS72)**

**ASSIGNMENT**

**VII SEMESTER**

**2018-2019**

**Submitted by**

**Chethan K P** **1RV15CS048**

**Layeeq Ahmed** **1RV15CS075**

**Under the Guidance of**

**Dr. Krishnappa H K**,

**Associate Professor, Department of CSE,**

**R.V.C.E, Bengaluru - 59**.

**R.V. COLLEGE OF ENGINEERING, BENGALURU - 560059**

**(Autonomous Institution Affiliated to VTU, Belagavi)**

# **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

Certified that the project titled “**OpenGL Snake Game**” has been carried out by **Chethan K P (1RV15CS048)** and **Layeeq Ahmed (1RV15CS075),** bonafide students of R.V. College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS (12CS72)** during the year 2018-2019. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

**Dr. Krishnappa H K**

Assistant Professor,

Department of CSE,

R.V.C.E, Bengaluru - 59.

**List of Contents**

1. Source Code………………………………………………………………………….

2. Implementation Results and Snapshots………………………………………………

**SOURCE CODE**

This project has been developed with object oriented programming framework. Object oriented programming has been effectively used to isolate and modularize the graphics part and the logic parts of the app. The source code is defined in 3 files: **snake.h, snake.cpp** and **run.cpp.**

* Snake.h contains the declarations for the Snake class which contains the logic to handle movements, fruit generation, length of snake, score, death etc. It also contains the header declarations for required packages.
* Snake.cpp contains the definitions for the snake class for all the functions that have been declared in the snake.h file.
* Run.cpp contains the main UI loop for the application. It is responsible for handling windows and rendering frames periodically, and communication with the snake object that gets created here.

**Snake.h:**

#include <bits/stdc++.h>

#include <GL/glut.h>

using namespace std;

#define GRID\_SIZE 40

#define CELL\_SIZE 20

#define INITIAL\_SNAKE\_LENGTH 4

#define TIMEOUT 100

class Snake

{

int velX, velY, fruitX, fruitY, score, lives;

bool eaten, dead;

vector<pair <int, int>> snake;

public:

Snake();

int getLives();

int getScore();

bool getDead();

void drawMaze();

void drawRect(int x, int y, int r, int g, int b);

void drawFruit();

void drawSnake();

void death();

void updateSnake();

void updateVelocities(int x, int y);

void reset();

};

**Snake.cpp**

#include "snake.h"

using namespace std;

Snake::Snake()

{

// Initial snake creation

for(int i = 0; i < INITIAL\_SNAKE\_LENGTH; i++)

snake.push\_back(make\_pair(GRID\_SIZE / 2 - i, GRID\_SIZE / 2));

// Initial snake velocities

velX = 1;

velY = 0;

score = 0;

lives = 3;

eaten = true;

dead = false;

}

int Snake::getLives()

{

return lives;

}

int Snake::getScore()

{

return score;

}

bool Snake::getDead()

{

return dead;

}

void Snake::drawMaze()

{

glColor3f(1, 1, 1);

glRecti(5, 15, GRID\_SIZE \* CELL\_SIZE - 5, 5);

glRecti(5, GRID\_SIZE \* CELL\_SIZE - 5, GRID\_SIZE \* CELL\_SIZE - 5, GRID\_SIZE \* CELL\_SIZE - 15);

glRecti(5, GRID\_SIZE \* CELL\_SIZE - 5, 15, 5);

glRecti(GRID\_SIZE \* CELL\_SIZE - 15, GRID\_SIZE \* CELL\_SIZE - 5, GRID\_SIZE \* CELL\_SIZE - 5, 5);

}

void Snake::drawRect(int x, int y, int r, int g, int b)

{

glColor3f(r, g, b);

glRecti(x \* CELL\_SIZE + 1, y \* CELL\_SIZE + 1, (x + 1) \* CELL\_SIZE - 2, (y + 1) \* CELL\_SIZE - 2);

}

void Snake::drawFruit()

{

if(eaten)

{

fruitX = rand() % 38 + 1;

fruitY = rand() % 38 + 1;

}

drawRect(fruitX, fruitY, 1, 0, 0);

eaten = false;

}

void Snake::drawSnake()

{

for(int i = 0; i < snake.size(); i++)

drawRect(snake[i].first, snake[i].second, 0, 1, 0);

}

void Snake::death()

{

velX = velY = 0;

}

void Snake::updateSnake()

{

if(snake[0].first == GRID\_SIZE - 1 || snake[0].first == 0 || snake[0].second == GRID\_SIZE - 1 || snake[0].second == 0)

{

if(lives--)

{

snake.clear();

for(int i = 0; i < INITIAL\_SNAKE\_LENGTH; i++)

snake.push\_back(make\_pair(GRID\_SIZE / 2 - i, GRID\_SIZE / 2));

velX = 1;

velY = 0;

return;

}

else

dead = true;

}

int x = snake[0].first;

int y = snake[0].second;

int last\_x = snake[snake.size() - 1].first;

int last\_y = snake[snake.size() - 1].second;

drawRect(last\_x, last\_y, 0, 0, 0);

for(int i = snake.size() - 1; i >= 1; i--)

snake[i] = snake[i - 1];

x += velX;

y += velY;

snake[0] = make\_pair(x, y);

if(fruitX == x && fruitY == y)

{

snake.push\_back(make\_pair(0, 0));

eaten = true;

for(int i = snake.size() - 1; i >= 1; i--)

snake[i] = snake[i - 1];

snake[0] = make\_pair(x, y);

score++;

}

for(int i = 2; i < snake.size(); i++)

if(snake[0].first == snake[i].first && snake[0].second == snake[i].second)

{

if(lives--)

{

snake.clear();

for(int i = 0; i < INITIAL\_SNAKE\_LENGTH; i++)

snake.push\_back(make\_pair(GRID\_SIZE / 2 - i, GRID\_SIZE / 2));

velX = 1;

velY = 0;

return;

}

else

dead = true;

}

}

void Snake::updateVelocities(int x, int y)

{

if(velX == 1 && x == -1 || velX == -1 && x == 1 || velY == 1 && y == -1 || velY == -1 && y == 1)

return;

velX = x;

velY = y;

}

void Snake::reset()

{

snake.clear();

snake.push\_back(make\_pair(GRID\_SIZE / 2, GRID\_SIZE / 2));

snake.push\_back(make\_pair(GRID\_SIZE / 2 - 1, GRID\_SIZE / 2));

snake.push\_back(make\_pair(GRID\_SIZE / 2 - 2, GRID\_SIZE / 2));

snake.push\_back(make\_pair(GRID\_SIZE / 2 - 3, GRID\_SIZE / 2));

velX = 1;

velY = 0;

dead = false;

score = 0;

lives = 3;

}

**run.cpp**

#include "snake.h"

using namespace std;

int window;

list<pair<int, int>> velocityBuffer;

int delay = TIMEOUT;

void \* font = GLUT\_BITMAP\_TIMES\_ROMAN\_24;

Snake snake\_obj;

void myInit()

{

glClearColor(0.0, 0.0, 0.0, 1.0);

gluOrtho2D(0, 800, 0, 850);

}

void menu(int value)

{

switch (value) {

case 0:

{

glutDestroyWindow(window);

exit(0);

}

case 1:

{

snake\_obj.reset();

break;

}

}

}

void createMenu()

{

int menu\_id = glutCreateMenu(menu);

glutAddMenuEntry("Restart", 1);

glutAddMenuEntry("Quit", 0);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void renderString(float x, float y, float r, float g, float b, string s)

{

glColor3f(r, g, b);

glRasterPos2f(x, y);

for(int i = 0; i < s.length(); i++)

glutBitmapCharacter(font, s[i]);

}

void updateVelocities(int key, int x, int y)

{

if(snake\_obj.getDead())

return;

int velX, velY;

switch (key) {

case GLUT\_KEY\_UP:

velX = 0;

velY = 1;

break;

case GLUT\_KEY\_DOWN:

velX = 0;

velY = -1;

break;

case GLUT\_KEY\_RIGHT:

velX = 1;

velY = 0;

break;

case GLUT\_KEY\_LEFT:

velX = -1;

velY = 0;

break;

}

velocityBuffer.push\_back(make\_pair(velX, velY));

}

void handleMovement(int value)

{

glutTimerFunc(delay, handleMovement, 0);

snake\_obj.updateSnake();

if(!velocityBuffer.empty())

{

snake\_obj.updateVelocities(velocityBuffer.front().first, velocityBuffer.front().second);

velocityBuffer.pop\_front();

}

glutPostRedisplay();

}

void end()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

renderString((GRID\_SIZE / 2 - 4) \* CELL\_SIZE, (GRID\_SIZE + 2) / 2 \* CELL\_SIZE, 1, 1, 1, "GAME OVER!");

renderString((GRID\_SIZE / 2 - 3) \* CELL\_SIZE, (GRID\_SIZE - 2) / 2 \* CELL\_SIZE, 1, 1, 1, "SCORE: " + to\_string(snake\_obj.getScore()));

}

void myDisplay()

{

string scores\_string = "SCORE: ", lives\_string = "LIVES: ";

delay = TIMEOUT - snake\_obj.getScore() / 5;

glClear(GL\_COLOR\_BUFFER\_BIT);

snake\_obj.drawMaze();

snake\_obj.drawSnake();

snake\_obj.drawFruit();

scores\_string += to\_string(snake\_obj.getScore());

renderString(15, GRID\_SIZE \* CELL\_SIZE + 10, 1, 1, 1, scores\_string);

lives\_string += to\_string(snake\_obj.getLives());

renderString((GRID\_SIZE - 5) \* CELL\_SIZE, GRID\_SIZE \* CELL\_SIZE + 10, 1, 1, 1, lives\_string);

if(snake\_obj.getDead())

{

snake\_obj.death();

end();

}

glFlush();

}

int main(int argc, char \*argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100, 100);

glutInitWindowSize(800, 850);

window = glutCreateWindow("Snake");

myInit();

createMenu();

glutDisplayFunc(myDisplay);

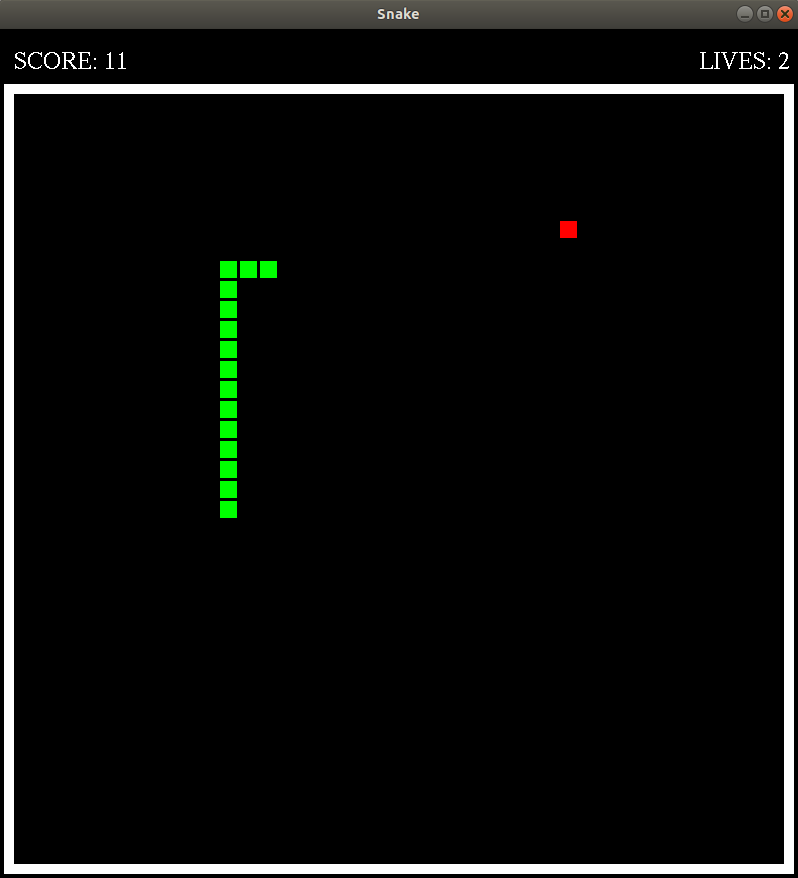
glutTimerFunc(delay, handleMovement, 0);

glutSpecialFunc(updateVelocities);

glutMainLoop();

return 0;

}

**IMPLEMENTATION RESULTS AND SNAPSHOTS**

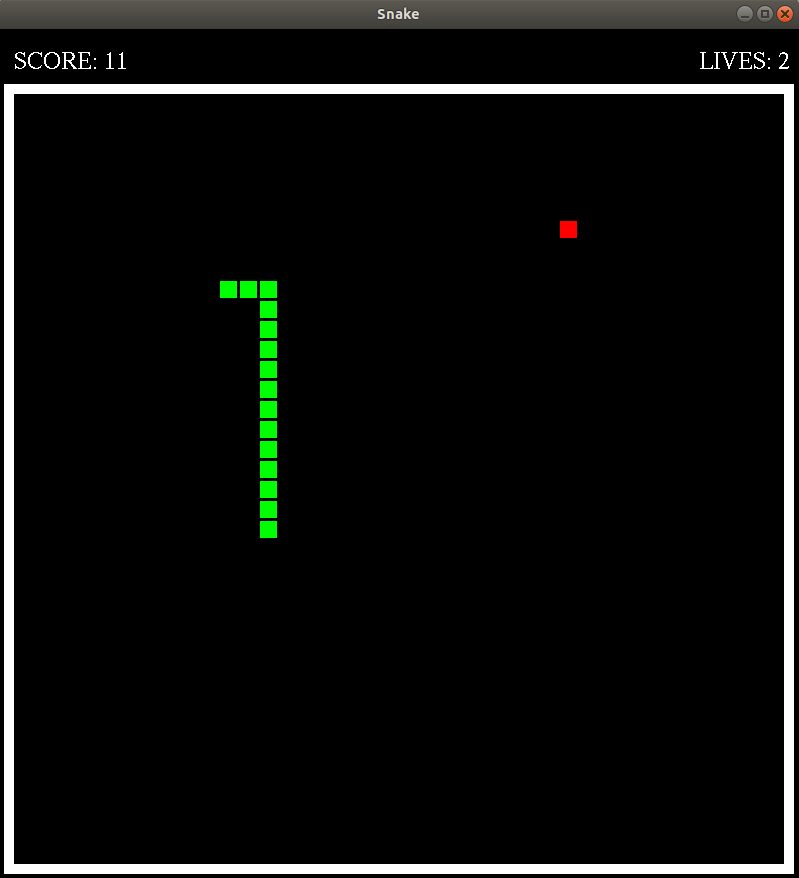
****

Fig1. Left movement using left arrow key Fig2. Right movement using right arrow

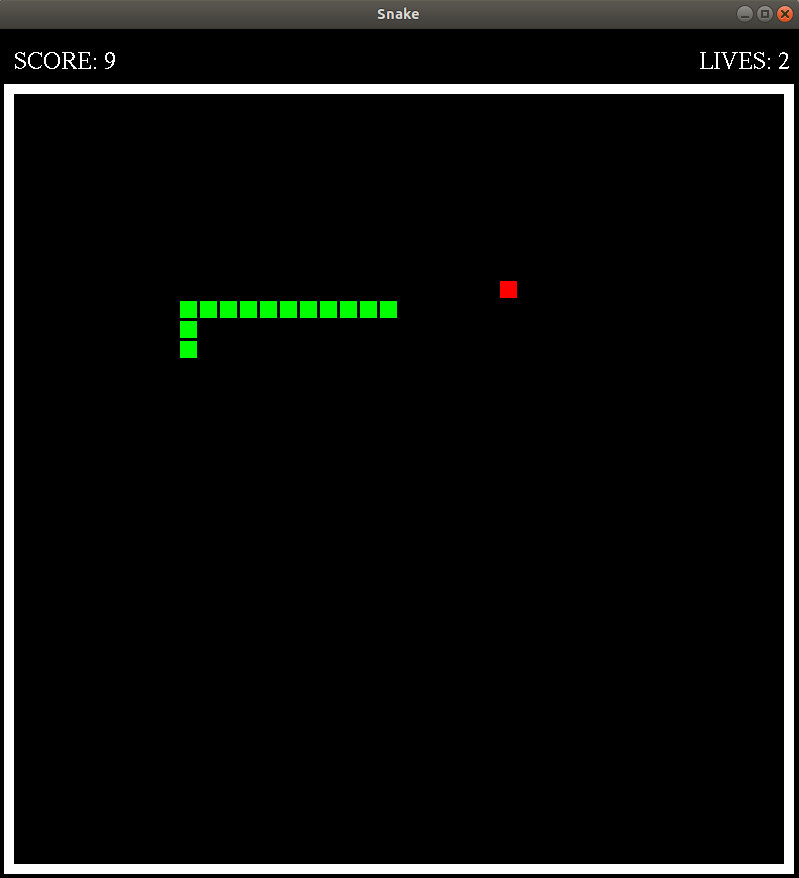
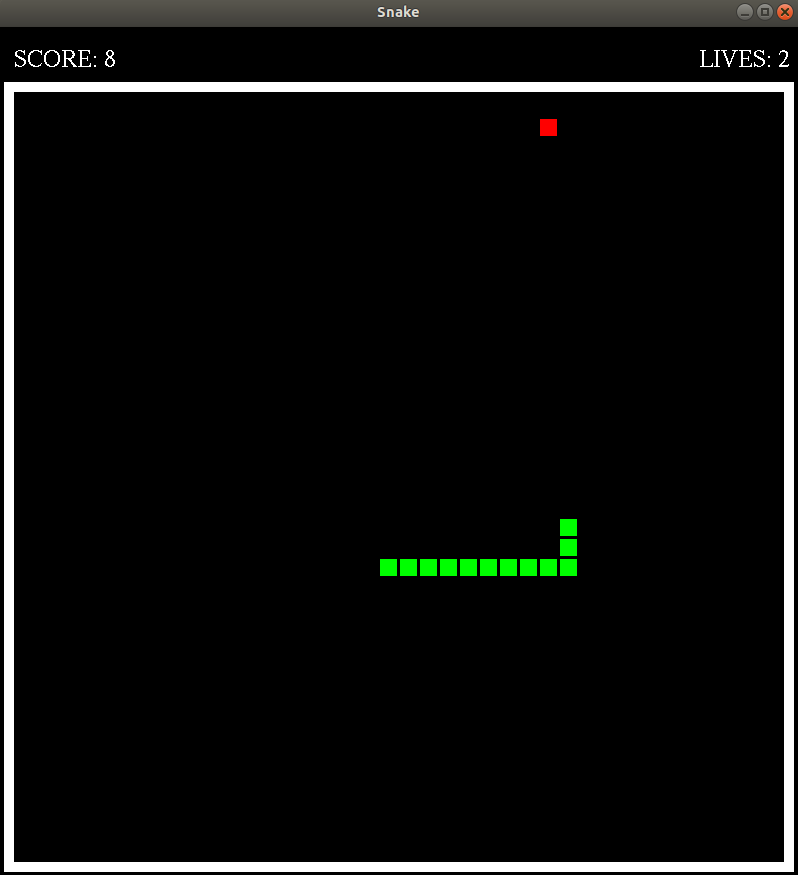


Fig3. Top movement using up arrow Fig4. Bottom movement using down arrow

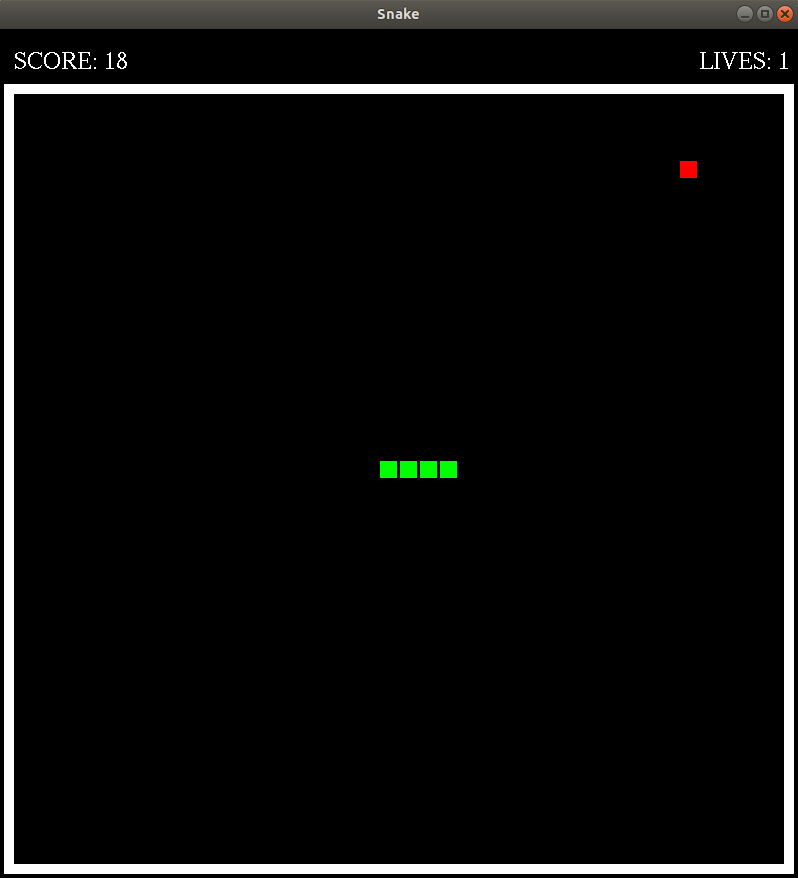
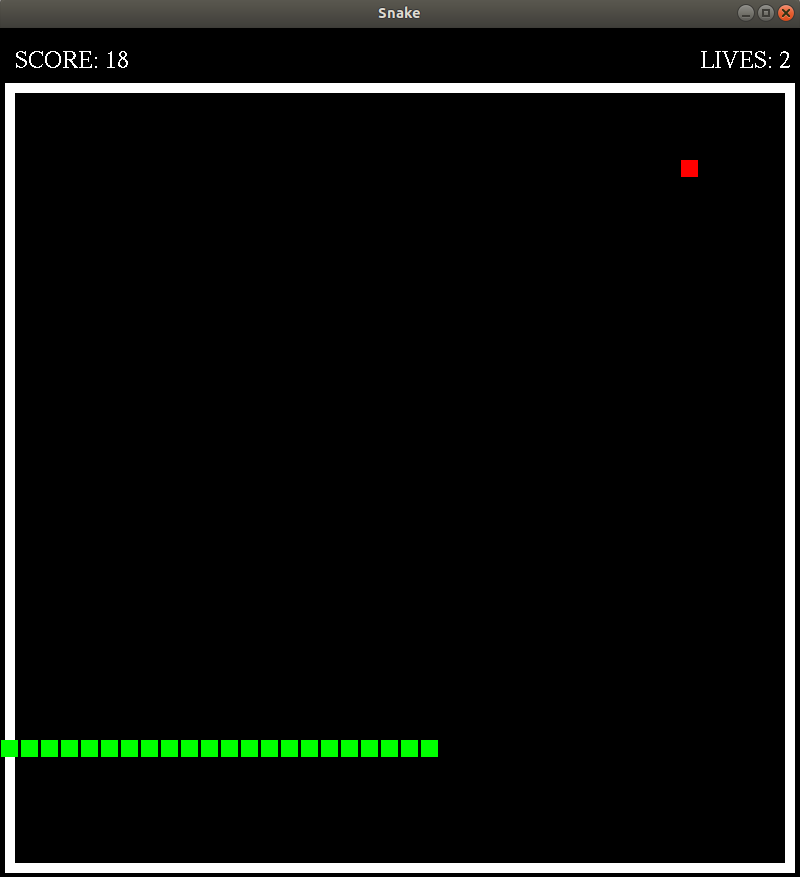
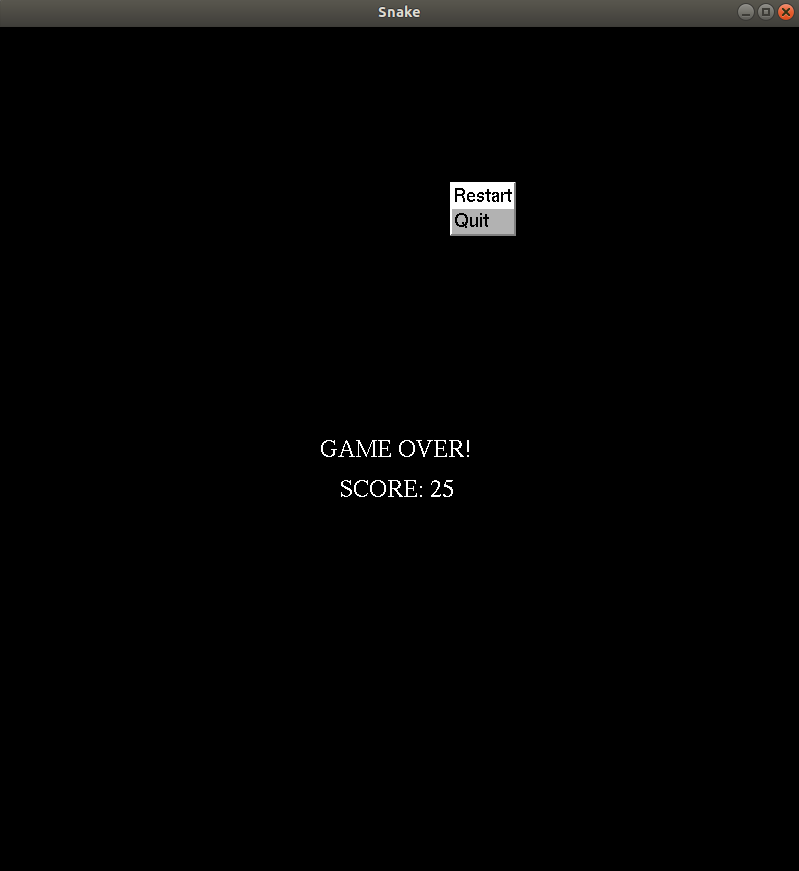
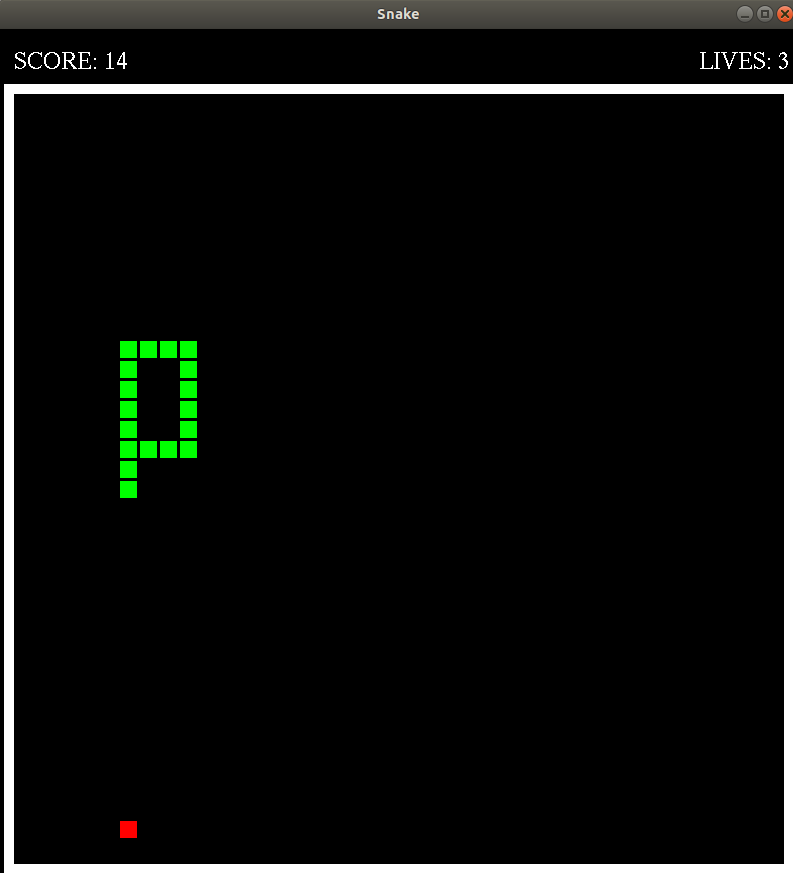


Fig5. Death with body contact Fig6. Death with maze contact

Fig7. Lives decreases Fig8. Game Over if all lives are used

A menu is provided upon mouse right click to either restart or quit the game