Computer Graphics (UCS505)

Project on

HUNGRY SNAKE

Submitted By

Mohit	102197021
Jitesh Garg	102017180
Ananya Singh	102016083

Group No. 6

B.E. Third Year – CSE

Submitted To:

Ms. Diksha Arora

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

Introduction to Project

This project uses C++ and OpenGL to create an interactive snake game which is typically played on a rectangular grid. The snake is initially made up of a fixed length, but as it eats food, it grows longer.

The objective of the game is to guide the snake to eat as much worms as possible and gaining points for the same without running into any walls or its own body. If the snake collides with a wall the game ends at the same moment while if it collides by itself some fixed number of times (having some lives in this case) and the player loses.

To control the snake, the player typically uses the arrow keys on the keyboard to move it up, down, left, or right. The snake moves continuously, and the player must avoid walls and its own body which gradually changes size and plan ahead to guide it in the right direction for the same.
There also exists a "special food" which provides more score the player. These types of foods namely normal and special are distinguishable by color.

As the game progresses, the snake becomes longer, making it more challenging to avoid obstacles and navigate through the board. The game has multiple levels of difficulty, with faster movement as the player may navigate through the levels. Additionally it has a luck factor which can also be modified which decides which type of food appears on screen.

. Overall, the Snake Game is a simple game that requires quick reflexes, strategicthinking, and good hand-eye coordination.

Features of the project:
- Keys for movement
- Pause/resume the game
- Restart the game
- Change the difficulty level
- Change map size
- Help key
- Exit game

The keys and guide for these features will also be displayed in terminal for reference and help to player.
The final score of the game is displayed in the terminal at the end of the game.

Parameters that determine the various features of the game:

- Luck: determines the level of luck the user would have for the special food to appear on the screen.
- Board size: this determines the size of the screen as bigger screen would allow more space and thus easing the game.
- Self bite lives: these would include the no of lives the user gets if snake comes incontact with its own body.
- Difficulty: This determines how fast the snake moves

Computer Graphics concepts used

Libraries used:

- iostream
- glut.h
- deque
- assert.h

Some other concepts/functions used are:

- **Animation:** glutPostRedisplay() signals the event loop that the window needs to be redrawn. This function is typically called from the display callback function, which is registered with the GLUT windowing system. The display callback function is responsible for rendering the graphics for the current window.
- **Coordinate systems:** A coordinate system is used to specify the position and orientation of objects in a scene. Snake, food, walls everything is made by coordinate systems and geometry only.
- **Rendering:** Rendering refers to the process of creating an image from a set of geometric primitives and other information. In this snake game, rendering is used to display the snake and food on the screen using basic shapes like rectangles or squares.
- **User input:** User input allows the player to interact with the game by controlling the movement of the snake. In this case, not only movement but there are whole set of options available which are mentioned before and will be displayed on terminal too which can be used to enhance the experience.

User Defined Functions

1. **void guide():** displays the basic instructions of the game on the terminal. Helps user identify the use of the various keys on the keyboard so that operations like movement (uptown, left, right), pause/play, quit or help can be performed.

2. **void showFinalScore():** Displays the final score on the terminal.

3. **void draw_body_part(int x, int y)** : Draws each of the square in the snake's body length that increases once it gets a worm. Uses simple glBegin(GL_POLYGON) followed by the vertices of the next box of the snakes body which is a square of 1x1 size translates from the coordinates of the last box of snake's body (x,y).

4. **void food_texture(int x, int y):** Function to draw a worm, that is a rhombus.

5. **void drawBrick(int x, int y):** Draw a brick of size using GL_PLOYGON.

6. **void drawWalls():** Function that draws the walls of the game by iterating over the given size of board(map_size) and drawing a brick for each of the walls(top, bottom, left, right) simultaneously for every iteration.

7. **bool overlap(int x, int y):** Checks for the given coordinates if they lie on the body of the snake.

8. **void drawFood():**it generates a random value of x and y that lies within the board size and. The overlap function is then called to ensure that the generated coordinates of worm food are not overlapping with the snake Body, else generate again.A random value within the range of 1-100 is also generated and if the value is less than the luck set of the game, then the special worm with greater points is displayed instead (worm with different colour and more points value). Then food_texture is called to display the given point on the screen.

9. **void drawSnake():** This function draws the head of the snake and they of the snake first. Then it iterates over the drawbodypart() function over the length of the snake to draw the snake upto the length it has attained at the given instance by using the coordinates of the snake's body stored in the deque.

10. **void moveSnake(int newDirection):** Function that moves the snake in the given direction. This is done by first identifying the direction given and hence updating variables delx, dely depending on the direction so that the new coordinates created can be used to verify one of the following conditions:
    - If the snake hit its body, which means that the newly generated coordinates already exist in the deque that contains the snake's body coordinates.
    - If the snake hit the walls, that is the coordinates of head of the snake overlap with the map borders.

Otherwise the length of the snake is increased and points are given (1 for normal worm, 100 for special worm) and the updation of the deque occurs.

11. **void TimerFunc(int val):** checks if game is paused and if it is not paused is calls movesnake function.

12. **TimerFunc(gameSpeed, TimerFunc, 0):** this function calls itself drawing new snake after some time and difficulty varies as if snake is drawn faster it creates a feel as if snake is moving fast its arguments are time after which second argument which function is to be called is mentioned.

13. **void display():** used to display the game screen and initialising the snake, walls and worm.

14. **void reshape(GLsizei, GLsizei):** used to inscrease/decrease board size/screen size.

15. void keyboard(unsigned char key, int, int) : includes the working of the directions and movement of the snake according to the keyboard pressed and also defines how the game should change if any of the other defines keyboard presses occur while the game runs.

16. void initializeGame(): this intiliazes the games and the variables and displays the snake on the screen. Also ensures that the set parameters by the user lie within the range specified in the start.

Code

```
#include <iostream>
#include <glut.h>
#include <deque>
#include <assert.h>
using namespace std;


#define mapBgColor  0.48,0.98,0.00
#define snakeColor  0.00,0.00,1.00
#define foodColor    1.00,0.00,0.00
#define wallColor    0.36,0.25,0.20
#define splFoodColor 0.50,0.00,0.00

int map_size = 30;     // keep value bw 10 and 50
int luck = 50;          // keep between 1 and 100 to get the worm.
int initialLives = 2; // saves when you bit yourself
int maxDifficulty = 6; // keep atleast 5.


// assigning codes to directions
#define UP     1
#define DOWN  2
#define LEFT  3
#define RIGHT  4

#define delay(n) for(int i = 0; i < 1e5;i++)for(int j = 0; j < n*1e3;j++);

deque<pair<int, int>> snake_body; // dsa for storing snake cordinates
int food_pos[2];                  // x,y position of food.


int foodAvailable = 0;
int score = 0;
int  special  =  0;
int  paused  =  1;
int finished = 0;
int direction = RIGHT;
int Difficulty = maxDifficulty; //higher is easy
int currentLives = initialLives;
int gameSpeed = Difficulty * 25; //also timer val to be passed in glutTimerFunc
```

```
void showFinalScore()
{
   cout << "****\n";
   cout << "YOUR FINAL SCORE IS: " << score;
}


void guide()
{
   printf("\n**********");
   printf("\nUse WASD as movement keys.");
   printf("\nPress W for Up, A for left, S for Down and D for Right");
   printf("\nM to change difficuly.");
   printf("\nR to Restart Game.");
   printf("\n+ to Increase map size.The game will restart though.");
   printf("\n- to Decrease map size.The game will restart though.");
   printf("\nP to pause/resume the game.");
   printf("\nPress Esc or Q to Quit.");
   printf("\nPress H for help.");
   printf("\n\nPress P to Start.");
   printf("\n**********\n");
}



void draw_body_part(int x, int y)
{
   glColor3f(snakeColor);
   glBegin(GL_POLYGON);
   glVertex2i(x + 1, y + 1);
   glVertex2i(x + 1, y + 0);
   glVertex2i(x + 0, y + 0);
   glVertex2i(x + 0, y + 1);
   glEnd();

   glLineWidth(2);
   glColor3f(mapBgColor);
   glBegin(GL_LINES); // vertical line through middle of square
   glVertex2f(x + 0.5, y + 1);
   glVertex2f(x + 0.5, y + 0);
   glEnd();
   glBegin(GL_LINES); // horizontal line through middle of square
   glVertex2f(x + 0, y + 0.5);
   glVertex2f(x + 1, y + 0.5);
   glEnd();


}
```

```
void food_texture(int x, int y)
{
   glBegin(GL_POLYGON);
   glVertex2f(x + 0.5, y + 1.0);
   glVertex2f(x + 0.0, y + 0.5);
   glVertex2f(x + 0.5, y + 0.0);
   glVertex2f(x + 1, y + 0.5);
   glEnd();

}

void drawBrick(int x, int y)
{
   glColor3f(wallColor);
   glBegin(GL_POLYGON);
   glVertex2f(x + 0.95, y + 0.95);
   glVertex2f(x + 0.05, y + 0.95);
   glVertex2f(x + 0.05, y + 0.05);
   glVertex2f(x + 0.95, y + 0.05);
   glEnd();

}


void drawWalls()
{


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



      glColor3f(wallColor);
      drawBrick(i, 0);        // bottom wall
      drawBrick(i, map_size - 1);// top wall
      drawBrick(0, i);        // left wall
      drawBrick(map_size - 1, i);// right wall

   }
}



bool overlap(int fx, int fy)
{
   for (auto part : snake_body)
   {
```

```
        if (part.first == fx && part.second == fy)
        {
            return false;
        }
    }
    return true;
}

void drawFood()
{
    if (!foodAvailable)
    {
        // create food
        int fx = rand() % (map_size - 2) + 1, fy = rand() % (map_size - 2) + 1;
        int lottery = 1 + rand() % 100;
        if (lottery <= luck && score != 0)
        {
            special = 1;
        }
        while (!overlap(fx, fy))// make sure no overlap of food with snake body
        {
            fx = rand() % (map_size - 2) + 1, fy = rand() % (map_size - 2) + 1;
        }
        food_pos[0] = fx;
        food_pos[1] = fy;
    }
    foodAvailable = 1;

    if (special == 1)
        glColor3f(splFoodColor);
    else
        glColor3f(foodColor);

    food_texture(food_pos[0], food_pos[1]); // using a brick for food

}



void drawSnake()
{


    glColor3f(snakeColor);

    int hx = snake_body[0].first;  // head x value
    int hy = snake_body[0].second; // head y value
```

```
    glBegin(GL_POLYGON); // 1x1 square
    glVertex2i(hx + 1, hy + 1);
    glVertex2i(hx + 1, hy + 0);
    glVertex2i(hx + 0, hy + 0);
    glVertex2i(hx + 0, hy + 1);
    glEnd();

    glColor3f(mapBgColor);
    glBegin(GL_POLYGON); // making a small square for eye of snake
    glVertex2f(hx + 0.85, hy + 0.9);
    glVertex2f(hx + 0.85, hy + 0.65);
    glVertex2f(hx + 0.65, hy + 0.65);
    glVertex2f(hx + 0.65, hy + 0.9);
    glEnd();

    // draw body other than head
    glColor3f(snakeColor);

    int len = snake_body.size();
    for (int i = 1; i < len; i++)
    {
        draw_body_part(snake_body[i].first, snake_body[i].second);
    }
}

void moveSnake(int newDirection)
{
    direction = newDirection;
    int delX = 0;
    int delY = 0;
    int mapEdge = 0;
    int snake_part_axis = 0;

    switch (direction)
    {
    case UP:
        delY = 1;
        break;
    case DOWN:
        delY  =  -1;
        break;
    case LEFT:
        delX = -1;
        break;
    case RIGHT:
```

```
        delX = 1;
        break;
    }

    for (auto part : snake_body)
    {
        if ((part.first == (snake_body[0].first + delX)) && (part.second ==
(snake_body[0].second + delY)))
        {
            currentLives--;
            cout << "Be Careful! You got bit.\n";
            if (currentLives <= 0)
            {
                cout << "Game Over.";
                showFinalScore();
                delay(2);
                exit(0);
            }
        }
    }


    if (snake_body[0].first <= 0 || snake_body[0].first >= map_size - 1 ||
snake_body[0].second <= 0 || snake_body[0].second >= map_size - 1)
    {
        cout << "Oh NO! You ran into wall. Game Over.\n";
        showFinalScore();
        delay(2);
        exit(0);
    }
    int grow = 0;
    if (snake_body[0].first + delX == food_pos[0] && snake_body[0].second + delY
== food_pos[1])
    {
        grow = 1;
        if (special)
        {
            grow = 100;
            special = 0;
            cout << "Legendary Food!!! +100 score.\n";
        }
        score += grow;
        foodAvailable = 0;
    }


    snake_body.push_front({ snake_body[0].first + delX,snake_body[0].second +
delY });
```

```
    if (!grow)
    {
        snake_body.pop_back();
    }

    glutPostRedisplay();
}



void TimerFunc(int val)
{
    if (!paused)
        moveSnake(direction);
    glutTimerFunc(gameSpeed, TimerFunc, 0);
}



void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, map_size, 0, map_size);
    glMatrixMode(GL_MODELVIEW);
    drawSnake();
    drawFood();
    drawWalls();

    glutSwapBuffers();
}

void reshape(GLsizei, GLsizei)
{
    glutReshapeWindow(map_size * 20, map_size * 20);
}

void initializeGame()
{


    // some assertions
    assert(map_size >= 15);
    assert(map_size <= 50);
    assert(luck <= 100);
    assert(luck >= 1);
```

```
    assert(maxDifficulty >= 5);
    assert(initialLives > 0);
    srand(time(0));
    // glutOrtho2d()

    glClearColor(mapBgColor, 0);
    score = 0;
    currentLives = initialLives;
    // create size 3 snake
    snake_body.clear();
    snake_body.push_back({ 5,map_size / 2 });
    snake_body.push_back({ 4,map_size / 2 });
    snake_body.push_back({ 3,map_size / 2 });

    moveSnake(RIGHT);

    paused = 1;
    foodAvailable = 0;
    guide(); // show game guide

}


void keyboard(unsigned char key, int, int) {

    switch (key)
    {
    case 'W':
    case 'w':
    {
        if (!paused)
            if (direction == LEFT || direction == RIGHT) {
                moveSnake(UP);
            }

        break;
    }
    case 'S':
    case 's':
    {
        if (!paused)
            if (direction == LEFT || direction == RIGHT) {
                moveSnake(DOWN);
            }

        break;
```

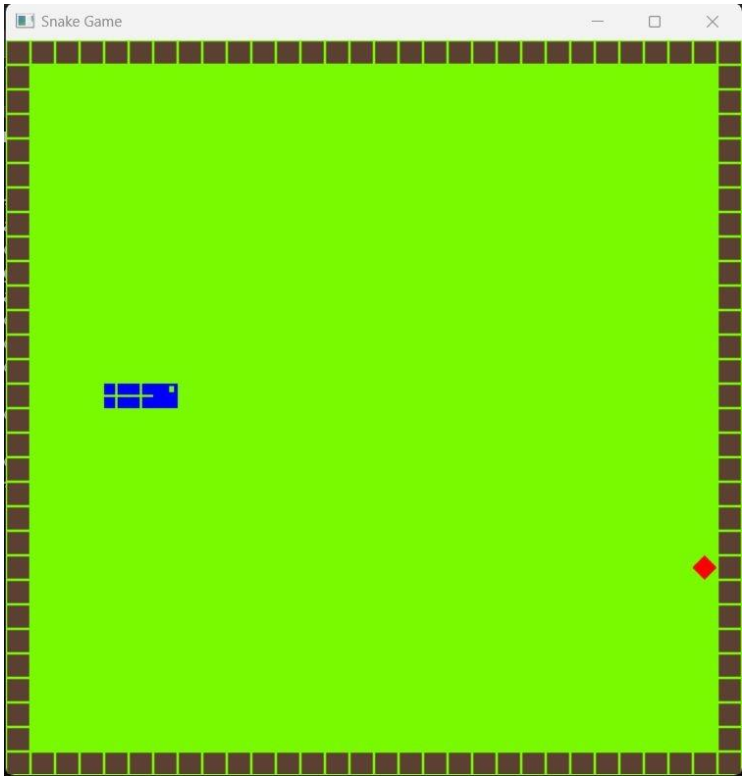
```
}

case 'A':
case 'a':
{
   if (!paused)
      if (direction == UP || direction == DOWN) {
         moveSnake(LEFT);
      }

   break;
}
case 'D':
case 'd':
{
   if (!paused)
      if (direction == UP || direction == DOWN) {

            moveSnake(RIGHT);
      }

   break;
}


case 'M':
case 'm':
{
   Difficulty = (Difficulty - 1) % (maxDifficulty + 1);
   if (Difficulty <= 0)
      Difficulty = maxDifficulty;
   gameSpeed = Difficulty * 20;
   cout << "New Difficulty is " << (maxDifficulty + 1) - Difficulty << endl;
   break;
}


case 'P':
case 'p':
{  if (paused == 0)
   cout << "You Stopped\nPaused the game\n";
else
   cout << "You Resumed/Started Game\n";
paused = !paused;
break;
}
```

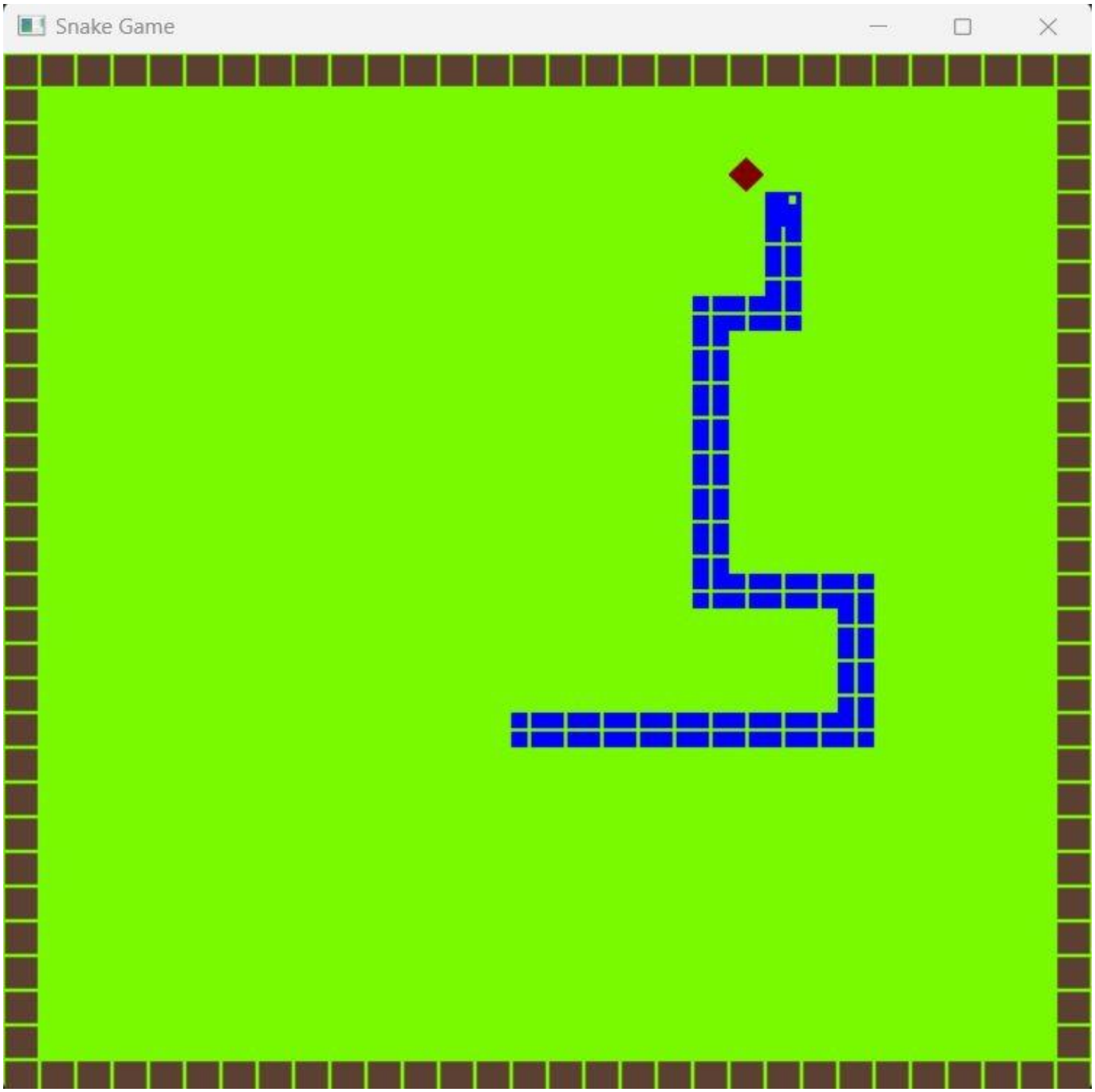
```
case 'H':
case 'h':
{
   guide();
   break;
}


case 'R':
case 'r':
{

   showFinalScore();
   cout << "You restarted the Game.\nRestarting...\n.All the best!!!.\n";
   initializeGame();
   break;
}

case '=':
case '+':
{
   if (map_size < 50)
      map_size = map_size + 1;
   glViewport(0, 0, 20 * map_size, 20 * map_size);
   showFinalScore();
   initializeGame();
   break;
}

case '_':
case '-':
{
   if (map_size > 15)
      map_size = map_size - 1;
   glViewport(0, 0, 20 * map_size, 20 * map_size);
   showFinalScore();
   initializeGame();
   break;
}
case 27:
case 'q':
{
   cout << "You pressed exit.\n";
   showFinalScore();
   exit(0);
}
```

```
    }
}

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

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowSize(map_size * 20, map_size * 20);
    glutInitWindowPosition(250, 100);
    glutCreateWindow("Snake Game");
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutReshapeFunc(reshape);
    glutTimerFunc(gameSpeed, TimerFunc, 0);
    initializeGame();
    glutMainLoop();
    //return 0;
}
```

Output/ Screen shots

```
**********************
Use WASD as movement keys.
Press W for Up, A for left, S for Down and D for Right
M to change difficuly.
R to Restart Game.
+ to Increase map size.The game will restart though.
- to Decrease map size.The game will restart though.
P to pause/resume the game.
Press Esc or Q to  Quit.
Press H for help.

Press P to Start.
**********************
```




Snake Game

```
Press P to Start.
**********
You Resumed/Started Game
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
You Stopped
Paused the game
You Resumed/Started Game
You Stopped
Paused the game
You Resumed/Started Game
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
Legendary Food!!! +100 score.
You Stopped
Paused the game
You Resumed/Started Game
You Stopped
Paused the game
You Resumed/Started Game
You Stopped
Paused the game
You Resumed/Started Game
You Stopped
Paused the game
You Resumed/Started Game
Legendary Food!!! +100 score.
Oh NO! You ran into wall. Game Over.
**********
YOUR FINAL SCORE IS: 1518
```