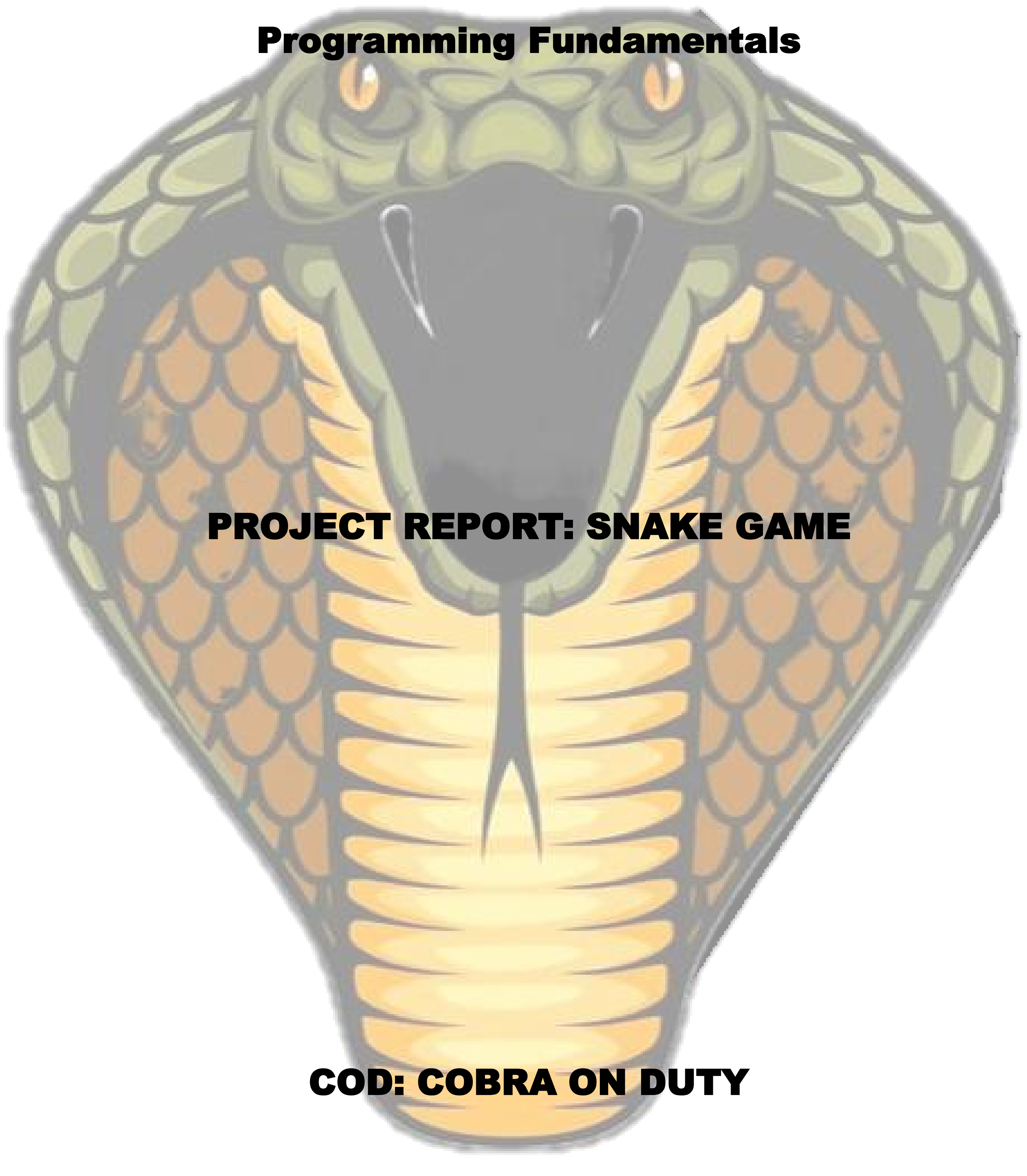


Programming Fundamentals

PROJECT REPORT: SNAKE GAME

COD: COBRA ON DUTY

GAME NAME:-

COD stands for Cobra On Duty.

GROUP NAME:-

CYBER PUNK

GROUP MEMBERS:-

1_ Muhammad Saad CT-278

2_ Muhammad Ayan Anwer CT-279

3_ Muhammad Huzaifa CT-280

4_ Ghazi Hasher Ali CT-297

GAME DESCRIPTION:-

- Programmed in **C** language,
- Simple **snake game** with more difficulties.
- It works on **Visual Studio.**
- Graphics has been used through **SDL**.
- Rules are mentioned.

TABLE OF CONTENTS:-

1.INTRODUCTION:-

The **Snake Game** is a classic arcade-style game where players control a snake that grows longer as it consumes food, navigating a grid-based environment. The objective is to achieve the highest score possible by eating food items while avoiding collisions with the game borders and the snake's own body.

This implementation of the Snake Game is developed using **SDL2** (Simple DirectMedia Layer) for graphics rendering and user interaction, along with **SDL_image** and **SDL_ttf** for handling images and fonts. The game combines engaging gameplay mechanics with visual appeal, leveraging textures and animations for the snake, food, and background.

In the Snake Game, you control a snake navigating a grid-based environment. The goal is to eat as much food as possible, causing the snake to grow longer with each item consumed. As the snake grows, maneuvering becomes increasingly challenging, requiring precise control and quick thinking. The game ends when the snake collides with the game's borders or its own body.

Upon launching, players are greeted with a welcoming introduction screen displaying the game title and instructions to start. Pressing the "Enter" key transitions the player to the main gameplay screen. The snake's speed and length dynamically increase as the player progresses, escalating the challenge. Food appears in random positions, ensuring a unique experience in every session. The game tracks and displays the player's score in real time, providing instant feedback on performance. The score reflects the total amount of food consumed, with higher scores representing greater skill and endurance. A clear "Game Over" message is displayed when the player loses, coupled with a visual cue to enhance the experience. The game includes precise collision detection to ensure realistic interactions, such as the snake eating food or colliding with walls. The game includes visible borders, providing a defined play area and a visual boundary to enhance gameplay clarity.

Control the direction of the snake:

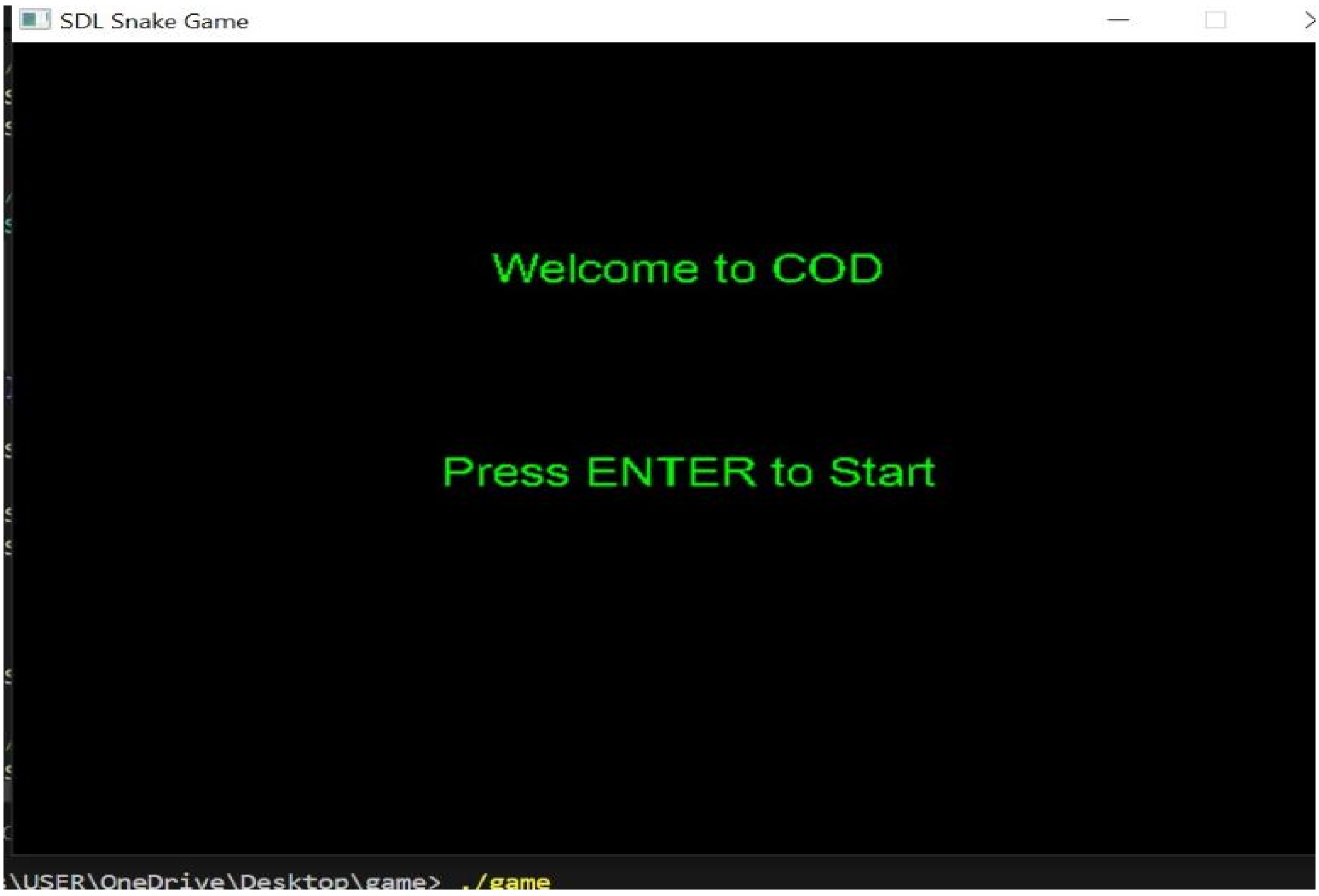
Up (↑): Move the snake upwards

Down (↓): Move the snake downwards.

Left (←): Move the snake left.

Right (→): Move the snake right.

Enter Key: Starts the game from the introduction screen.

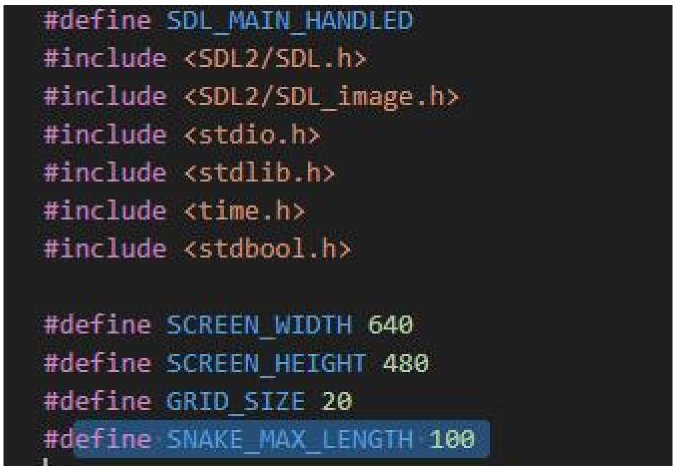


Welcome to COD

Press ENTER to Start

```
\USER\OneDrive\Desktop\game> ./game
```

2.Header Files and Macros :-

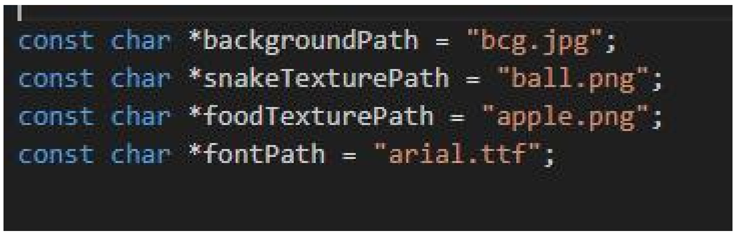


```c
#define SDL_MAIN_HANDLED
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>

#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480
#define GRID_SIZE 20
#define SNAKE_MAX_LENGTH 100
```

- **SDL_MAIN_HANDLED**: This macro prevents SDL from handling the main() function. It's required when using SDL2 with some IDEs (like Visual Studio) to avoid conflicts with their main entry point handling.
- **SDL2 and SDL_image**: The code includes SDL2 for graphics rendering and SDL_image for handling image files (PNG in this case).
- **Constants**: SCREEN_WIDTH and SCREEN_HEIGHT define the game window's dimensions. GRID_SIZE controls the grid size (each cell in the game grid is 20x20 pixels), and SNAKE_MAX_LENGTH limits the snake's maximum length to 100 segments.

3. Paths to Asset:-


```
const char *backgroundPath = "bcg.jpg";
const char *snakeTexturePath = "ball.png";
const char *foodTexturePath = "apple.png";
const char *fontPath = "arial.ttf";
```

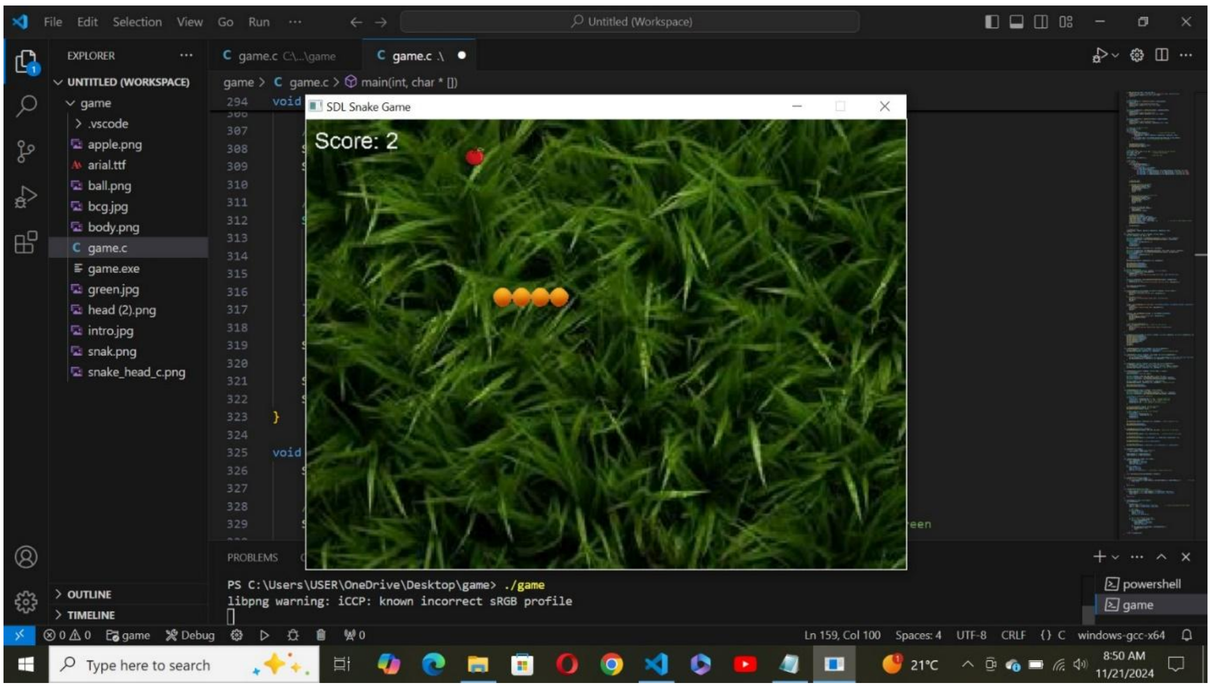
These variables hold the file paths to the images used in the game: a background image, a snake body texture, the font, and a food texture.

4. Structures for Snake and Food:-

```
typedef struct {
    int x, y;
} Point;


typedef struct {
    Point body[SNAKE_MAX_LENGTH];
    int length;
    Point direction;
} Snake;
```

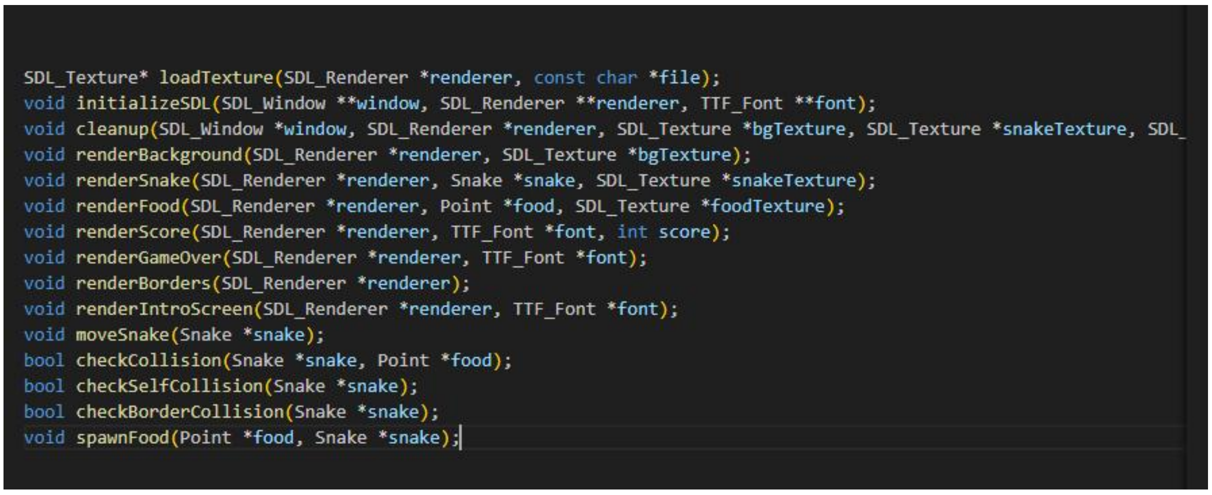
- **Point Struct**: Represents a 2D coordinate (x, y).
- **Snake Struct**: Represents the snake. It holds an array of Point structs for the body of the snake, an integer length to track the number of segments, and a direction (another Point) that dictates the movement direction (x and y components).



```
PS C:\Users\USER\OneDrive\Desktop\game> ./game
libpng warning: iCCP: known incorrect sRGB profile
```

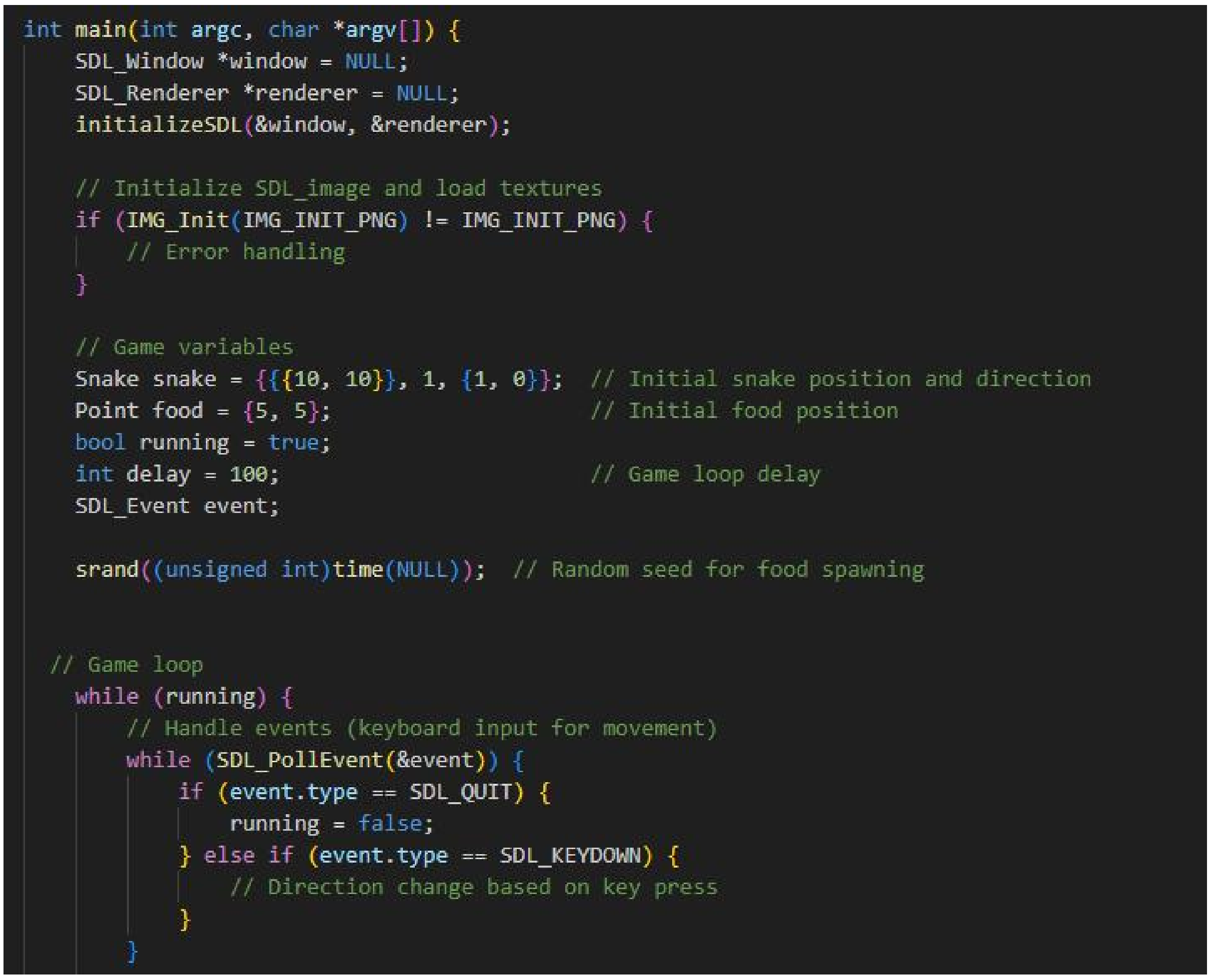
5. Function Prototypes:-

The functions in the program are declared, including utilities for loading textures, rendering game objects, and handling game logic like snake movement, collision detection, and food spawning.



```c
SDL_Texture* loadTexture(SDL_Renderer *renderer, const char *file);
void initializeSDL(SDL_Window **window, SDL_Renderer **renderer, TTF_Font **font);
void cleanup(SDL_Window *window, SDL_Renderer *renderer, SDL_Texture *bgTexture, SDL_Texture *snakeTexture, SDL_
void renderBackground(SDL_Renderer *renderer, SDL_Texture *bgTexture);
void renderSnake(SDL_Renderer *renderer, Snake *snake, SDL_Texture *snakeTexture);
void renderFood(SDL_Renderer *renderer, Point *food, SDL_Texture *foodTexture);
void renderScore(SDL_Renderer *renderer, TTF_Font *font, int score);
void renderGameOver(SDL_Renderer *renderer, TTF_Font *font);
void renderBorders(SDL_Renderer *renderer);
void renderIntroScreen(SDL_Renderer *renderer, TTF_Font *font);
void moveSnake(Snake *snake);
bool checkCollision(Snake *snake, Point *food);
bool checkSelfCollision(Snake *snake);
bool checkBorderCollision(Snake *snake);
void spawnFood(Point *food, Snake *snake);
```

6. Main Function:-



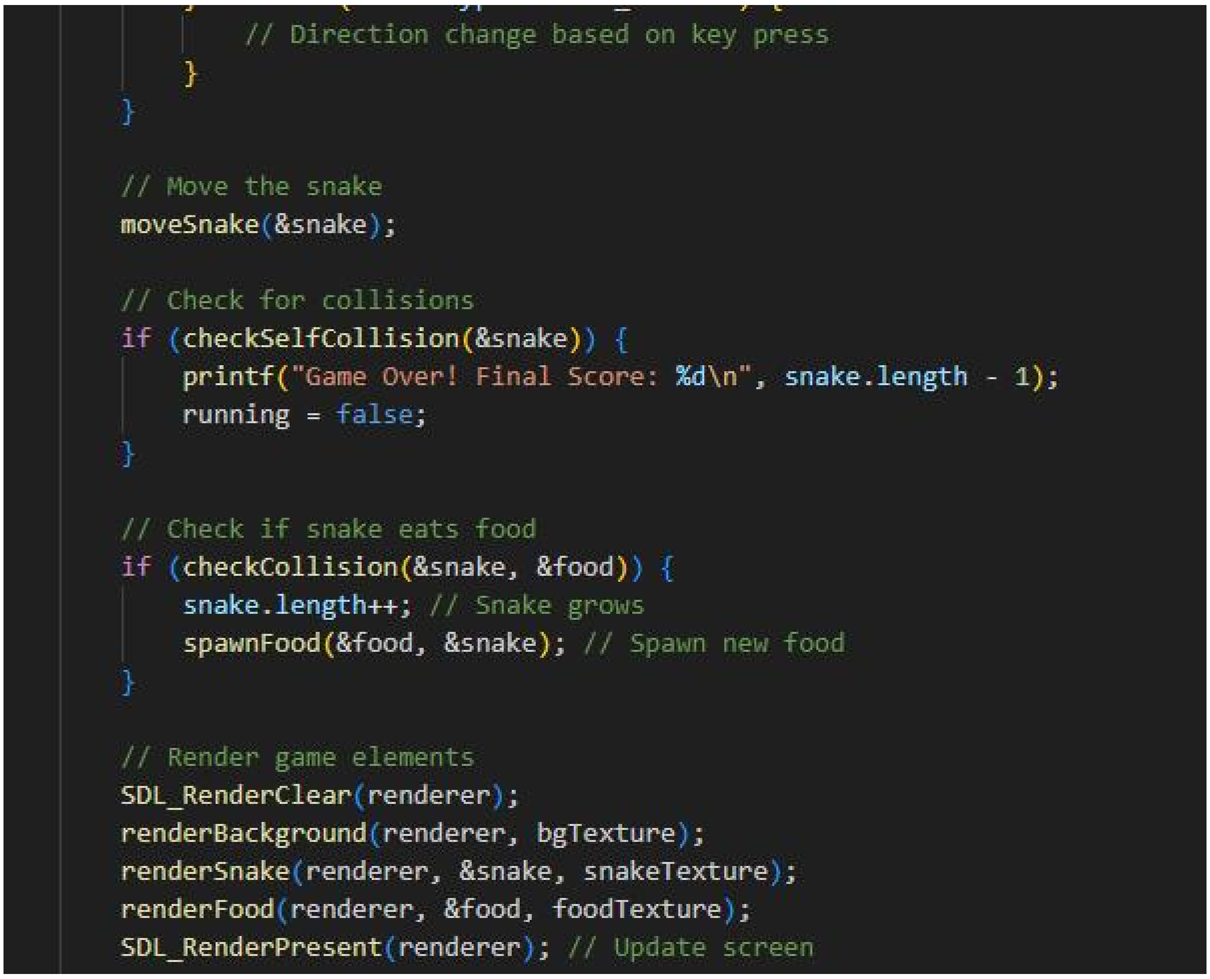
```c
int main(int argc, char *argv[]) {
    SDL_Window *window = NULL;
    SDL_Renderer *renderer = NULL;
    initializeSDL(&window, &renderer);

    // Initialize SDL_image and load textures
    if (IMG_Init(IMG_INIT_PNG) != IMG_INIT_PNG) {
        // Error handling
    }

    // Game variables
    Snake snake = {{{10, 10}}, 1, {1, 0}};  // Initial snake position and direction
    Point food = {5, 5};                     // Initial food position
    bool running = true;
    int delay = 100;                         // Game loop delay
    SDL_Event event;

    srand((unsigned int)time(NULL));  // Random seed for food spawning


  // Game loop
    while (running) {
        // Handle events (keyboard input for movement)
        while (SDL_PollEvent(&event)) {
            if (event.type == SDL_QUIT) {
                running = false;
            } else if (event.type == SDL_KEYDOWN) {
                // Direction change based on key press
            }
        }
```



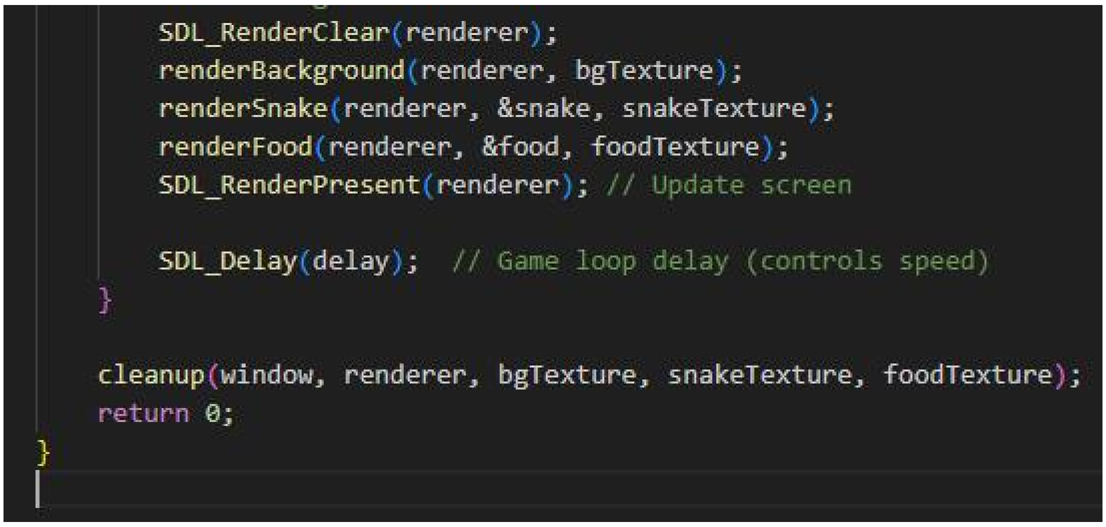
```c
                // Direction change based on key press
            }
        }

        // Move the snake
        moveSnake(&snake);

        // Check for collisions
        if (checkSelfCollision(&snake)) {
            printf("Game Over! Final Score: %d\n", snake.length - 1);
            running = false;
        }

        // Check if snake eats food
        if (checkCollision(&snake, &food)) {
            snake.length++; // Snake grows
            spawnFood(&food, &snake); // Spawn new food
        }

        // Render game elements
        SDL_RenderClear(renderer);
        renderBackground(renderer, bgTexture);
        renderSnake(renderer, &snake, snakeTexture);
        renderFood(renderer, &food, foodTexture);
        SDL_RenderPresent(renderer); // Update screen
```



```c
        SDL_RenderClear(renderer);
        renderBackground(renderer, bgTexture);
        renderSnake(renderer, &snake, snakeTexture);
        renderFood(renderer, &food, foodTexture);
        SDL_RenderPresent(renderer); // Update screen

        SDL_Delay(delay);  // Game loop delay (controls speed)
    }

    cleanup(window, renderer, bgTexture, snakeTexture, foodTexture);
    return 0;
}
```

SDL Initialization: The SDL window and renderer are created using initializeSDL(). If SDL fails to initialize, the program exits.

- **Event Handling**: The SDL_PollEvent loop listens for keyboard events (arrow keys) to change the snake's direction.
- **Game Logic**:
  - The snake moves each frame using the moveSnake() function.
  - If the snake collides with itself, the game ends.
  - If the snake eats food, it grows longer, and new food is spawned at a random location using spawnFood().
- **Rendering**: After handling logic, the screen is cleared and updated with the background, snake, and food textures.

7. Supporting Functions:-

- **Texture Loading** (loadTexture): This function loads a texture from a file and returns an SDL_Texture object for rendering.
- **SDL Initialization** (initializeSDL): Initializes SDL, creates the game window, and prepares the renderer.
- **Cleanup** (cleanup): Cleans up and frees resources before exiting.
- **Rendering Functions** (renderBackground, renderSnake, renderFood): These functions render the background, snake, and food to the screen, respectively.
- **Snake Movement** (moveSnake): Moves the snake one step forward in the direction indicated by direction. It also wraps the snake around the screen if it moves off the edge.
- **Collision Detection** (checkCollision, checkSelfCollision): Checks if the snake collides with food or with itself.
- **Food Spawning** (spawnFood): Randomly spawns new food at a location not occupied by the snake.

8.Key Functionality:-

- **Snake Movement**: The snake moves based on user input (arrow keys). The direction is updated only when there is no reverse movement (i.e., the snake cannot turn directly 180°).
- **Game Over Conditions**: The game ends if the snake collides with itself. The score is displayed as the snake's length minus one (since the initial length is 1).
- **Food and Growth**: Every time the snake eats food, it grows longer, and a new food item is spawned in a random location.
- **Screen Wrapping**: The snake can wrap around the screen edges, making the game more forgiving and keeping the action continuous.

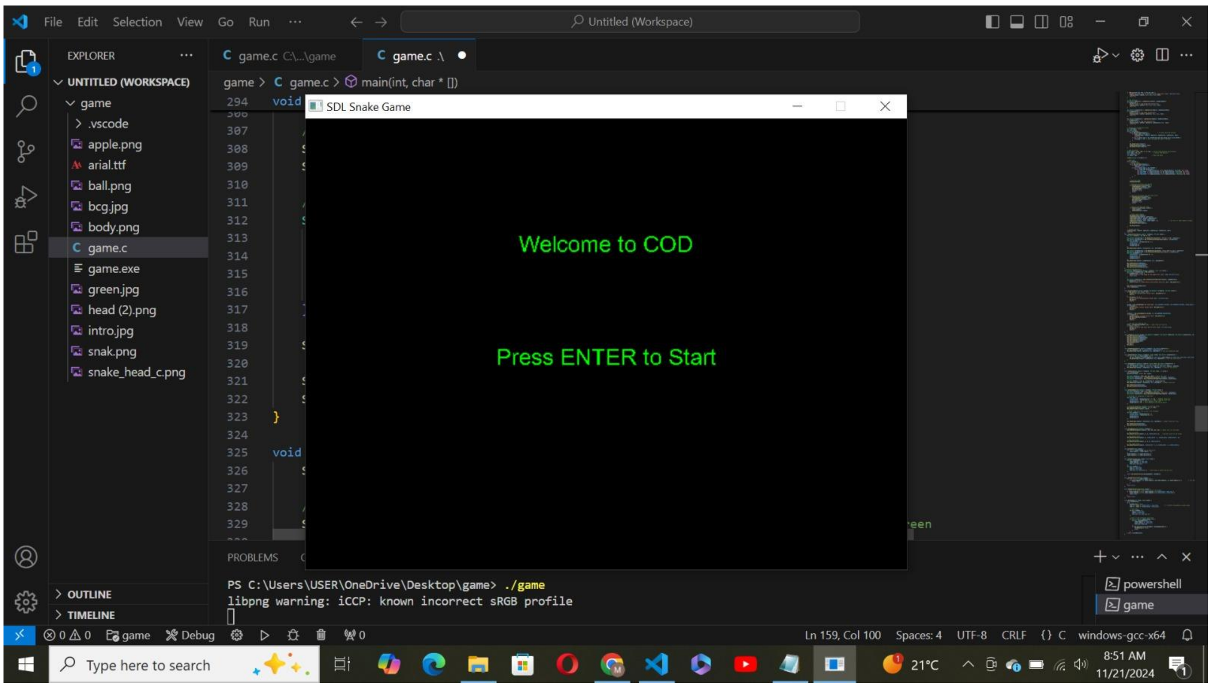
9.Improvements and Considerations:-

- **Difficulty**: The game speed (controlled by the delay variable) is constant. Adding a mechanism to increase the difficulty by reducing the delay as the game progresses would make the game more challenging.
- **Game Over Screen**: After the game ends, a proper game over screen (with an option to restart) could improve user experience.
- **Sound Effects**: Adding sound effects for eating food and colliding would enhance the gameplay.
- **Edge Case Handling**: The current collision detection with the snake's body works well, but additional checks could be added for edge cases like food spawning on the snake’s tail during rapid movement.

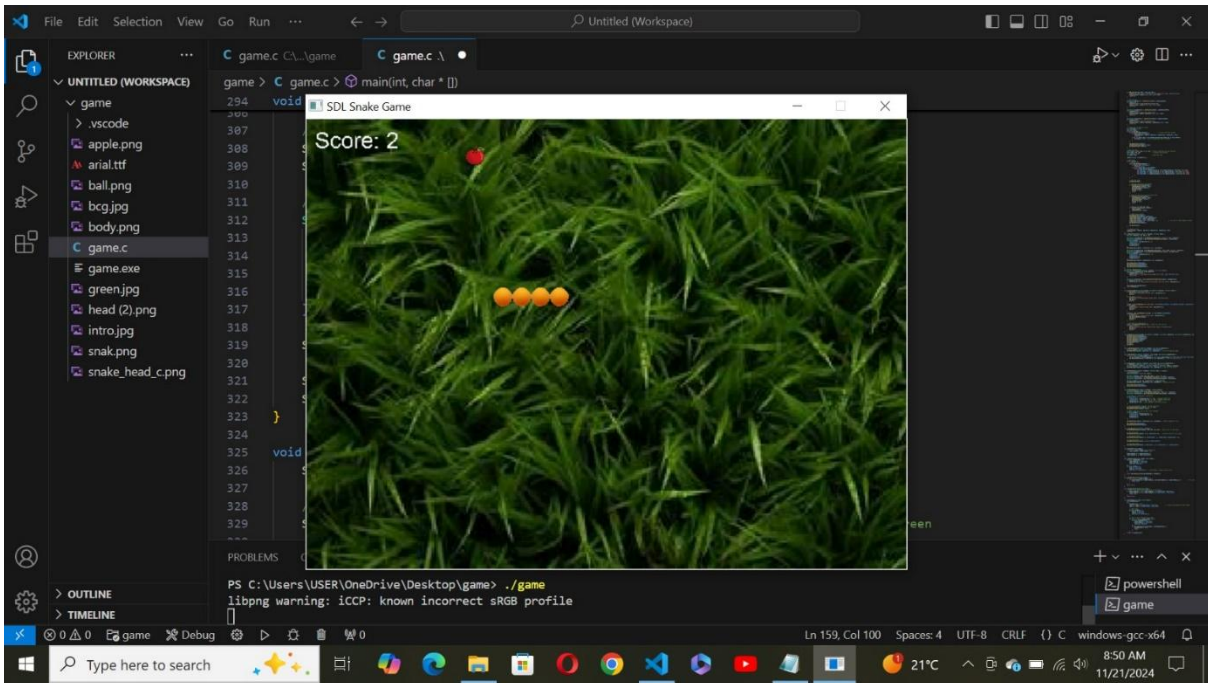
Conclusion:-

This SDL2-based Snake game is a functional and simple implementation that demonstrates basic game development concepts such as event handling, rendering, and collision detection. By managing the snake's movement, detecting collisions, and rendering textures, the code provides a solid foundation for further enhancement.

STARTING INTERFACE:

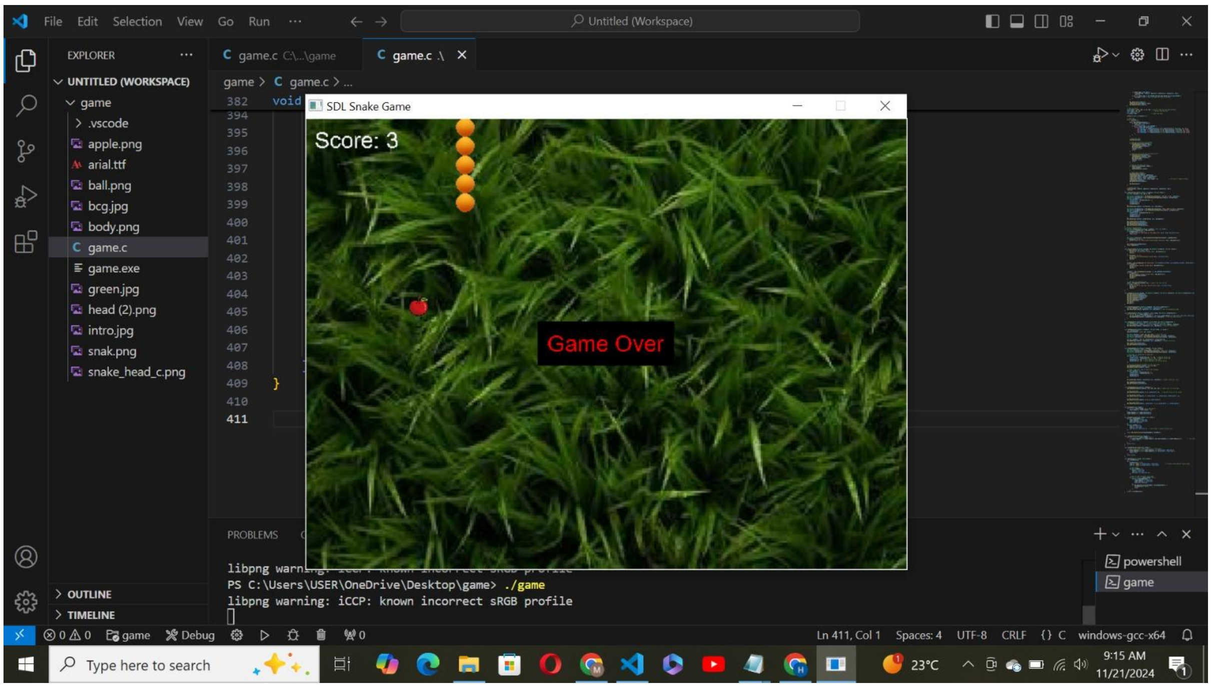


```
PS C:\Users\USER\OneDrive\Desktop\game> ./game
libpng warning: iCCP: known incorrect sRGB profile
```



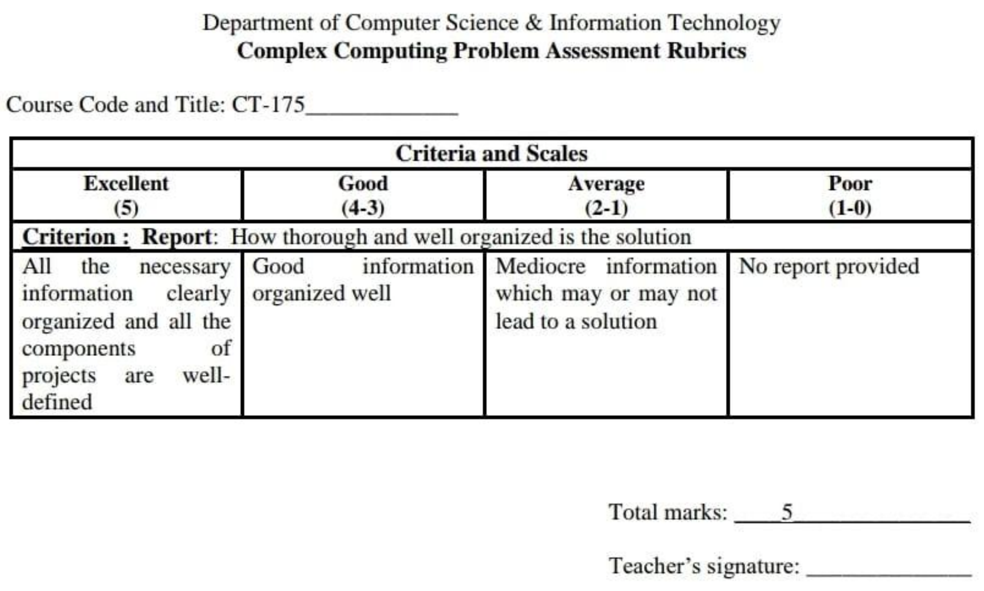
```
PS C:\Users\USER\OneDrive\Desktop\game> ./game
libpng warning: iCCP: known incorrect sRGB profile
```

GAME INTERFACE:



```
libpng warning: iCCP: known incorrect sRGB profile
PS C:\Users\USER\OneDrive\Desktop\game> ./game
libpng warning: iCCP: known incorrect sRGB profile
```

GAME OVER INTERAFACE:



Department of Computer Science & Information Technology

Complex Computing Problem Assessment Rubrics

Course Code and Title: CT-175_____________

Criteria and Scales			
Excellent (5)	**Good (4-3)**	**Average (2-1)**	**Poor (1-0)**
Criterion : Report: How thorough and well organized is the solution			
All the necessary information clearly organized and all the components of projects are well-defined	Good information organized well	Mediocre information which may or may not lead to a solution	No report provided

Total marks: ____5_______________

Teacher's signature: ______________