***PF***

***PROJECT PROPOSAL : SNAKE GAME***

***COD : COBRA ON DUTY***

***GAME NAME:-***

COD stands for Cobra On Duty.

***GROUP NAME:-***

CYBER CRACK

***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 DISCRITION:-***

* Programed in C language,
* Simple snake game with more difficulties.
* It works on Dev C++ .
* It shows your record.
* You got 3 lives in each game.
* Rules are mentioned.

***OVERVIEW:-***The snake game is a simple game where players control a snake that grows in length as it "eats" items that appear on the screen. The objective is to grow the snake as long as possible without colliding with the walls or itself.

1\_  *GAMEPLAY:-*

* + The player uses arrow keys to control the snake's direction: up, down, left, or right.
  + As the snake moves, it leaves a trail, representing its body, which grows longer each time it eats an item.
  + If the snake collides with itself or the game boundaries, the game ends.
  + You will only have 3 lives.
  + It shows your score in last

2\_ ***SCORING:-***

* + The player earns points each time the snake consumes its food.
  + Score can increase based on the number of items eaten.

3\_ ***FEATURES:-***

* + **Speed Increase**: The snake’s speed increases as the game progresses, making it more challenging.
  + **Obstacles**: Walls or other barriers can appear on the screen to make navigation harder.
  + **LIVES**: The snake has fewer lives to increase.

**4\_ *WIN/LOSE CONDITIONS****:-*

* + The player wins by achieving a maximum score or reaching a set length.
  + The game is over if the snake collides with itself or the boundaries.
  + The game will also end if the snakes loses all his three lives.

***Abstract:-***

This report provides a comprehensive analysis of a \*\*Snake Game\*\* developed in the C programming language. The Snake Game is a popular arcade game where the player controls a snake that must eat food and avoid collisions with the wall or its own body. As the snake eats the food, it grows longer, and the objective is to score as much as possible without dying. This report includes an overview of the game's structure, key functions, data structures used, and highlights the various game mechanics implemented in the code.

***1. Introduction***

The Snake Game is a classic game that was first developed in the 1970s, with various versions implemented over the years. In this implementation, the game is coded using C, where the user controls the snake using the arrow keys and must eat the randomly appearing food on the screen to score points. Each time the snake eats food, it grows longer, and the difficulty increases. The player must avoid running into the walls or the snake's body. The game is also designed with a \*\*score tracking\*\* system and \*\*multiple lives\*\* for the player.

***Objectives:-***

- Develop a simple snake game that operates in a text-based console environment.

- Allow players to control the snake using the arrow keys.

- Implement mechanics for snake movement, collision detection, food spawning, and scoring.

- Provide a system to track high scores and record the player’s name after the game ends.

***2. Game Design:-***

***2.1 Basic Game Flow:-***

***1. \*\*Game Start:\*\****

- The game initializes by displaying a welcome message and the rules.

- The player presses any key to start the game.

- A loading animation is shown before the game begins.

***2. \*\*Game Mechanics:\*\****

- The snake moves in the direction determined by the arrow key pressed by the player.

- The snake grows longer as it eats food, and each food item generates at a random position on the screen.

- The game ends when the snake collides with the wall, itself, or runs out of lives.

***3. \*\*Game Over:\*\****

- The player is provided with 3 lives, which decrement each time the snake collides with the wall or itself.

- Once all lives are lost, the game ends, and the player can record their score.

***2.2 Game Board***

- The game board is displayed in a 70x30 grid, with a border around the edges to simulate the walls.

- The snake is represented by `o` characters, and the head is represented by a special character (`>`, `<`, `^`, `v` depending on direction).

- The food is represented by `0` and is randomly placed within the grid.

- The game features a border with vertical (`|`) and horizontal (`-`) lines.

***3. Data Structures:-***

***3.1 Snake Representation***

The snake is represented using a `coordinate` structure which stores the `x` and `y` positions as well as the direction the snake is moving in. The snake's body is stored as an array of these coordinates.

```c

struct coordinate {

int x; // X-coordinate

int y; // Y-coordinate

int direction; // Direction of movement (UP, DOWN, LEFT, RIGHT)

};

```

***3.2 Key Variables:-***

1. \*\*`head`:\*\* Represents the snake’s head.
2. \*\*`bend[]`:\*\* Array of the snake’s body parts, storing the positions and directions.
3. \*\*`food`:\*\* Coordinates of the food to be consumed by the snake.
4. \*\*`length`:\*\* Current length of the snake, which increases as the snake eats food.
5. \*\*`life`:\*\* The number of lives the player has remaining.
6. \*\*`key`:\*\* The current key pressed by the player.

***3.3 Key Constants:-***

The constants `UP`, `DOWN`, `LEFT`, `RIGHT` represent the directions in which the snake can move, corresponding to key codes for the arrow keys.

```c

#define UP 72

#define DOWN 80

#define LEFT 75

#define RIGHT 77

```

***4. Key Functions:-***

***4.1 Game Initialization and Setup:-***

1. \*\*`Print()`\*\*: Displays the introduction and instructions at the start of the game.
2. \*\*`Boarder()`\*\*: Draws the game board, including walls and the food item at its initial position.
3. \*\*`load()`\*\*: Simulates a loading sequence before starting the game.

***4.2 Snake Movement:-***

- \*\*`Move()`\*\*: Controls the main logic of snake movement and collision detection. It handles continuous movement and changes the snake’s direction when the user presses an arrow key.

- \*\*`Right()`, `Left()`, `Up()`, `Down()`\*\*: Functions for handling the movement in the respective directions. Each function moves the snake part by part, updating the body and drawing the new positions on the screen.

- \*\*`Bend()`\*\*: Ensures the snake's body follows the head as it moves.

***4.3 Food Generation***

- \*\*`Food()`\*\*: Generates food at random positions within the game grid. If the snake's head collides with the food, the snake's length increases.

***4.4 Collision Detection:-***

- \*\*`ExitGame()`\*\*: Checks if the snake collides with the walls or itself. If a collision occurs, the player loses a life. The game continues if there are remaining lives; otherwise, it ends.

***4.5 Scoring and Lives:-***

- \*\*`Score()`\*\*: Displays the current score based on the snake's length.

- \*\*`Scoreonly()`\*\*: Returns the score for recording.

- \*\*`record()`\*\*: Records the player's score and name to a text file, allowing the player to see past records.

***4.6 Console Output:-***

- \*\*`GotoXY()` and `gotoxy()`\*\*: Move the cursor to specific coordinates on the screen to display characters at precise locations.

- \*\*`Delay()`\*\*: Provides a delay to control the speed of the snake's movement, making the game playable.

***5. Game Logic:-***

***5.1 Snake Growth and Collision Detection:-***

As the snake moves, the positions of its body parts are updated in the `body[]` array. When the snake eats food, its length increases, and new segments are added to the body. The game checks for collisions with the snake’s body and the walls to determine if the game should end.

***5.2 Key Event Handling:-***

The game listens for key presses (`getch()`), which control the snake's direction. The snake cannot move in the opposite direction (i.e., if the snake is moving right, it cannot immediately move left).

***5.3 Pause and Resume:-***

While the game is running, the player can press any key to pause the game, and pressing any key again resumes the game.

***5.4 Game Over and Record Keeping:-***

When the game ends, the player is prompted to input their name. The score is saved to a text file, and the player can choose to view the records.

***6. Challenges and Improvements:-***

***6.1 Challenge: Screen Flickering:-***

Since the game operates in a console environment, the screen is cleared (`system("cls")`) frequently to update the game state, which can cause flickering. This could be improved by optimizing the drawing functions and reducing the number of screen clears.

***6.2 Improvement: Enhanced Graphics:-***

While the game is text-based, using more advanced graphics libraries such as \*\*ncurses\*\* (on Linux) or \*\*SDL\*\* could provide a more visually appealing version of the game with smoother animations.

***6.3 Improvement: Levels and Speed:-***

Adding a difficulty level with increasing speed as the snake grows could make the game more challenging and engaging.

***6.4 Improvement: High Score Tracking:-***

The game could include a high score table that persists even after the game is closed, allowing players to compete for the best scores.

***7. Conclusion:-***

The Snake Game implemented in C is a straightforward yet engaging project that demonstrates the power of basic game development concepts such as input handling, collision detection, and simple animation. While this version is text-based and runs in a console window, it can serve as the foundation for more complex game implementations with enhanced graphics, sound, and additional features.

Through this implementation, players can experience a fun and challenging game that harks back to the arcade classics, all while learning essential programming concepts. Further improvements could enhance the gaming experience and provide more depth and replayability.

**Future enhancement:**

More realistic 2d snake character will be added and improvements will be made