Snake Game – Design and Implementation Documentation

This document explains the design decisions, architecture, and implementation details of the classic Snake Game developed using C# Windows Forms. The project demonstrates event handling, timers, custom drawing, and basic game logic.

1. Project Overview

The Snake Game is a grid-based 2D game where the player controls a snake that moves continuously in one direction. The objective is to eat food, grow in length, and achieve the highest possible score without colliding with walls or the snake's own body.

2. Game Environment & Grid Design

The game area is implemented using a Panel control that represents a 30×30 grid, which satisfies the minimum 20×20 requirement. Each grid cell has a fixed size of 16 pixels. The grid is visually rendered using the Paint event, ensuring smooth graphics through double buffering.

3. Snake Representation & Movement

The snake is represented using a List<Point>, where each Point corresponds to a cell in the grid. The first element of the list is treated as the snake's head. Movement is handled by a Timer-based game loop, which updates the snake's position at fixed intervals. Direction control is achieved using arrow keys, with logic preventing instant reversal to maintain valid movement.

4. Keyboard Input Handling

To ensure reliable keyboard input regardless of which control has focus, the ProcessCmdKey method is overridden. This guarantees that arrow key presses are always captured by the game logic, preventing buttons from intercepting input.

5. Food System & Scoring

Food is randomly generated in empty grid cells using a Random object. When the snake's head reaches the food position, the snake grows by one segment, the score increases by 10 points, and new food is spawned. The score is displayed in real time, and a session-based high score is maintained.

6. Collision Detection & Game Over

Collision detection checks for wall boundaries and self-intersection of the snake. If a collision occurs, the game transitions to the Game Over state, stops the timer, updates the high score if applicable, and displays the final score to the user.

7. Game States & User Interface

The game supports multiple states: Start, Active, Paused, and Game Over. The user interface includes Start, Pause/Resume, and Restart buttons, along with labels displaying the current score, high score, and game status. Overlay messages are drawn on the game panel for paused and game-over states.

8. Conclusion

This Snake Game project demonstrates a clear separation of game logic, rendering, and user input handling. The use of timers, event-driven programming, and structured data representations results in a clean, maintainable, and extensible implementation. The design allows for future enhancements such as levels, sound effects, and settings.