KHULNA UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Department of Electronics and Communication Engineering

COURSE TITLE: **DATA STRUCTURE & ALGORYTHM**

COURSE NO: **CSE- 2210**

DATE OF SUBMISSION: **14.11.23**

PROJECT NAME

SNAKE GAME

PROJECT SUPERVISOR

Md Nazirul Hasan Shawon

Lecturer

Dept. of CSE ,KUET

Nawaz Talukdar Sanglap

Lecturer

Dept. of ECE ,KUET

PRESENTED BY

Md. Shafe-Ul-Alam

ROLL: 2009035

YEAR: 2nd

SEMESTER: 2nd

Objectives:

1. This Project in C++ language of Snake Game is a simple console application with very simple graphics. In this project, we can play the popular "Snake Game" just like we played it elsewhere. We have to use the up, down, right or left arrows to move the snake.

2. Foods are provided at the several co-ordinates of the screen for the snake to eat. Every time the snake eats the food, its length will be increased by one element along with the score.

3. It isn't the world's greatest game, but it does give us an idea of what we can achieve with a relatively simple C++ program, and perhaps the basis by which to extend the principles and create more interesting games of our own

Introduction:

The game called "Snake" or "Snake Game" typically involve the player controlling a line or snake, there is no official version of the game, so gameplay varies. The most common version of the game involves the snake or line eating items which make it longer, with the objective being to avoid running into a border or the snake itself for as long as possible.

The player loses when the snake either runs into border or into its own body. Because of this, the game becomes moredifficult as it goes on,due to the growth of the snake. Nokia has installed the "Snake Game" on many of its phones. The game is also available on several websites, including YouTube, which allows viewers to play the game while a video loads.

Theory:

The Snake Game involves several key concepts and principles that contribute to its gameplay and mechanics. Here's a theoretical overview:

1. Movement and Controls: The snake moves through a grid-based playing area, and the player provides input, typically through arrow keys or other directional controls, to change the snake's direction. The snake's continuous movement adds a dynamic element to the game.

2. Grid System: The playing area is often structured as a grid, where both the snake and food items are positioned. The grid facilitates movement and helps define boundaries, contributing to the game's structure and rules.

3. Snake Growth: When the snake consumes a food item, its length increases. This growth poses a challenge for players, requiring them to navigate a longer snake through the playing area without collisions.

4. Collision Detection: The game incorporates collision rules to determine when the snake collides with the game borders or its own body. Collisions result in the end of the game, creating a risk-reward dynamic as players aim to grow their snake without hitting obstacles.

5. Scoring System: Points are typically awarded for each food item consumed. The scoring system provides a quantitative measure of the player's performance and progress in the game.

6. Difficulty Progression: As the snake grows longer, the game becomes more challenging. This progression is essential for maintaining player engagement and skill development, encouraging strategic thinking and quick reflexes.

7. Randomized Food Placement: Food items appear at random locations on the grid, requiring players to adapt to changing circumstances and preventing them from following a fixed pattern.

8. Game Loop: The game operates in a continuous loop, where the player's actions lead to consequences, such as the snake's movement and growth, until a game-ending event occurs. The loop encourages players to continuously strive for improvement and higher scores.

9. Game Over State: When the snake collides, the game enters a "Game Over" state. At this point, the player receives their final score, and they may have the option to restart the game.

10. Variations and Enhancements: Different versions of the Snake Game may introduce variations and enhancements, such as additional obstacles, power-ups, or special features, to add complexity and diversity to the gameplay.

Implementation of Data Structure :

1) Linked list: To make the snake's body and dynamiccaly allocated the transition of snake's body.

2) Linear search: to find the high scorer and score

3) Array: To store the player's name.

Code algorithm:

The provided code is a C++ implementation of a Snake Game with additional features such as saving and viewing scores, a welcome screen, and a high score search. Below is a simplified algorithmic overview of the main functionalities:

1. Class Definition (`class snake`):

- Contains private variables for game parameters, player information, and linked list nodes.

- Defines private helper functions for console manipulation (`gotoxy`, `textcolour`), score tracking (`result`), and linked list operations (`drawlist`, `destroylist`, `dolist`).

2. Game Initialization (`void setup()`):

- Sets up the initial game state, including the game area, snake head, and initial food position.

- Initializes flags, counters, and player information.

3. Game Loop (`void game1()`):

- Executes the main game loop that continues until the game ends.

- Calls `food()` to display food, `play()` to handle player input, `run()` to update snake position, and `check()` to determine collisions and score.

4. Player Input Handling (`void play()`):

- Uses `kbhit()` and `getch()` to detect and process keyboard input during the game.

- Handles arrow key inputs to change the snake's direction, 'p' for pausing, and ignores other keys.

5. Snake Movement (`void run()`):

- Updates the snake's position based on its current direction.

6. Collision Detection and Score Tracking (`void check()`):

- Checks for collisions with the game boundaries, the snake's own body, and food items.

- Updates the score, adjusts the game speed, and generates new food accordingly.

7. Saving and Viewing Scores (`void save()`, `void viewscore()`, `void highscoresearch()`):

- Utilizes file I/O operations to save player scores and display saved scores.

- `highscoresearch()` identifies the player with the highest score.

8. Main Function (`int main()`):

- Displays a main menu with options for starting a new game, viewing scores, viewing the highest score, and exiting.

- Utilizes a switch statement to execute the corresponding functionality based on the user's choice.

9. Instructions Screen (`void instructions()`):

- Displays instructions for playing the game and waits for user acknowledgment before clearing the screen.

10. Welcome Screen (`int welcome()`):

- Displays a welcome message, prompts the user to enter their name, and clears the screen.

11. End-of-Game Screen (`char end()`):

- Displays the game over message, player's score, saves the score, and prompts the user to play again or exit.

12. Console Manipulation Functions (`gotoxy`, `textcolour`):

- Used for setting the cursor position and changing text color in the console.

This algorithmic overview provides a high-level understanding of the code's structure and functionality. The actual code implementation contains more details and specific logic.

Game Features:

Certainly! The provided C++ code implements a simple console-based Snake game. Here are the key features of the game:

1. Game Initialization:

- The program welcomes the player to the Snake game.
- The player is prompted to enter their name.

2. Main Menu:

- The main menu presents the following options:
  - New Game: Starts a new game.
  - View Scores: Displays the scores from previous games.
  - View Highscore: Displays the highest score achieved.
  - Exit: Exits the game.

3. New Game (Snake Gameplay):

- Upon selecting "New Game," the player receives game instructions.
- The player's name is displayed, and the actual gameplay begins.
- The snake starts in the middle of the game area.
- The goal is to control the snake to eat the food (denoted by '*').

4. Controls:

- The player controls the snake using the arrow keys (UP, DOWN, LEFT, RIGHT).
- Pressing 'P' pauses the game, and any other key resumes the game.
- Pressing 'ESC' quits the game without saving the score.

5. Game Over:

- The game continues until the snake collides with the walls or itself.

- Upon game over, the player sees their final score based on the number of times the snake has eaten food.

- The score is displayed along with the player's name.

6. Score Tracking:

- The game keeps track of the player's score in real-time.

- The score is calculated based on the number of times the snake eats the food.

7. File Operations:

- Scores are saved to a file ("record.txt") after each game.

- The "View Scores" option allows the player to see their previous scores.

8. Highscore Feature:

- The "View Highscore" option displays the highest score achieved and the corresponding player's name.

- It searches through the scores recorded in "record.txt" to find and display the highest score.

9. Loop Structure:

- The game follows a loop structure, where the player can choose to play again after a game over.

- The player can return to the main menu or exit the program after viewing scores or the highscore.

10. Console Graphics:

- The game is presented in a console window.

- The snake, food, and game environment are displayed using basic ASCII characters.

- The console window is updated in real-time to create the illusion of movement.

Overall, the Snake game provides a classic gaming experience with score tracking, highscore comparison, and basic console-based graphics.

Limitations:

The provided Snake Game code exhibits several limitations:

1. Code Organization and Readability:

- The code lacks modularization, making it challenging to understand and maintain.

- Limited use of comments makes it difficult to comprehend the purpose and functionality of certain code segments.

2. Global Variables:

- The use of global variables within the class can lead to potential issues, such as naming conflicts and reduced code encapsulation.

3. Hard-Coded Values:

- The code contains hard-coded values for screen dimensions, speed, and color codes, making it less adaptable for customization.

4. Limited Error Handling:

- The code lacks comprehensive error-handling mechanisms, potentially leading to unexpected behavior in certain scenarios.

5. Memory Management:

- The linked list structure used for the snake may pose memory management challenges. The code does not include proper memory deallocation (e.g., using `delete` for dynamically allocated nodes), which could lead to memory leaks.

6. Platform Dependency:

- The code uses Windows-specific functions (`conio.h`, `windows.h`), making it platform-dependent and unsuitable for cross-platform compatibility.

7. Pause Mechanism:

- The pause mechanism using `system("pause")` can be problematic, as it depends on system-specific behavior and may not work consistently on all platforms.

8. Limited Object-Oriented Design:

- The code lacks a well-defined object-oriented structure. It could benefit from better encapsulation, separation of concerns, and adherence to object-oriented principles.

9. Limited User Interaction:

- The code does not handle various user inputs comprehensively. For example, it assumes correct inputs during player name entry and lacks a robust input validation mechanism.

10. Limited Scalability:

- The code may become complex and difficult to extend as new features or modifications are introduced, limiting its scalability.

-To improve the code, consider addressing these limitations through refactoring, adopting better coding practices, and enhancing the overall design and functionality of the Snake Game.

Discussion:

The Snake Game implementation provides a nostalgic and engaging gaming experience, offering a classic arcade-style challenge. The code lacks a clear separation of concerns and modularization. Consideration should be given to organizing the code into functions and classes to enhance readability and maintainability. The use of global variables within the class may lead to naming conflicts and hinder code encapsulation. Encouraging encapsulation and minimizing global variables can improve code maintainability. The use of console manipulation functions (`gotoxy` and `textcolour`) aids in creating a visually appealing interface. However, consideration should be given to making the code platform-independent for broader accessibility. The core gameplay mechanics, including snake movement, collision detection, and food generation, function effectively. The incorporation of a linked list structure to represent the snake demonstrates a suitable approach for tracking its segments. The welcome screen and instructions enhance the user experience by providing context and guidance. However, the pause mechanism utilizing `system("pause")` may not be consistent across all platforms.The scoring system, displayed in real-time, effectively tracks the player's performance. However, the code does not implement a mechanism for persistently storing and retrieving high scores across different game sessions.The implementation of file I/O for saving and viewing scores adds a competitive element to the game. However, additional error checking and validation for file operations would enhance the robustness of this feature.The ability to pause the game provides convenience for players. However, relying on the system-specific `system("pause")` function may not be universally supported, and alternative methods should be considered for cross-platform compatibility. The code exhibits certain limitations, such as hard-coded values, limited error handling, and platform dependency. Refactoring the code to address these issues would enhance its flexibility, robustness, and maintainability.Consider introducing additional gameplay features, levels, or power-ups to increase the game's depth and replay value. Exploring object-oriented design principles could lead to a more extensible and scalable codebase.

Conclusion:

The Snake Game implementation presents a functional and nostalgic rendition of the classic arcade game. Despite its engaging gameplay and features, the code exhibits certain strengths and areas for improvement. The fundamental aspects of the Snake Game, such as snake movement, collision detection, and real-time scoring, function effectively, providing an engaging gaming experience. The welcome screen, instructions, and pause feature contribute to a user-friendly interface, enhancing the overall gaming experience. The implementation of file I/O for saving and viewing scores adds a competitive element and persistence to player achievements. Code Structure and Readability:* The code lacks a clear structure and modularization, making it challenging to comprehend and maintain. Refactoring the code into functions and classes could improve

readability and maintainability.Minimizing the use of global variables and promoting encapsulation would enhance code organizati/on and reduce the risk of naming conflicts.The code relies on Windows-specific functions, limiting its portability. Adopting platform-independent alternatives would improve cross-platform compatibility. Implementing comprehensive error handling, especially for file I/O operations, would enhance the code's robustness and reliability. The Snake Game implementation successfully captures the essence of the classic arcade game, providing entertainment and nostalgia for players. While the code demonstrates functional gameplay and scoring mechanisms, addressing its limitations through thoughtful refactoring and enhancements presents an opportunity to create a more polished and adaptable version of this timeless game. The Snake Game code serves as a foundation for an enjoyable gaming experience, and with strategic improvements, it has the potential to become a more versatile and maintainable implementation of this iconic arcade classic.

Reference:

1. https://www.geeksforgeeks.org/snake-game-in-c/amp/
2. https://www.scribd.com/document/541104132/snake-game-Doc
3. https://robertheaton.com/2018/12/02/programming-project-5-snake/