**Serpent Rush Game**

**A PROJECT REPORT**

***Submitted by***

**Rohit Kumar (23BCS13034)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

COMPUTER SCIENCE ENGINEERING



**Chandigarh University**

June 2025



**BONAFIDE CERTIFICATE**

Certified that this project report **“Serpent Rush Game”** is the bonafide work of “**Rohit Kumar (23BCS13034)**”who carried out the project work under my/our supervision.

**SIGNATURE**

**HEAD OF THE DEPARTMENT**

CSE

**SIGNATURE**

**SUPERVISOR**

CSE

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER EXTERNAL EXAMINER

**TABLE OF CONTENTS**

| Contents | Page |
| --- | --- |
| List of Figures | i |
| Abstract | ii |
| Graphical Abstract | iii |
| Abbreviations | iv |
| Chapter 1. Introduction | 1 |
| 1.1 Client Identification | 1 |
| 1.2 Identification of Problem | 2 |
| 1.3 Identification of Tasks | 3 |
| 1.4 Timeline | 4 |
| 1.5 Organization of Report | 5 |
| Chapter 2. Literature Review/Background Study | 6 |
| 2.1 Timeline of Reported Problem | 6 |
| 2.2 Proposed Solutions | 6 |
| 2.3 Bibliometric Analysis | 7 |
| 2.4 Review Summary | 7 |
| 2.5 Problem Definition | 7 |
| 2.6 Goals/Objectives | 8 |
| Chapter 3. Design Flow/Process | 9 |
| 3.1 Evaluation of Features | 9 |
| 3.2 Design Constraints | 9 |
| 3.3 Feature Finalization | 10 |
| 3.4 Design Flow | 10 |
| 3.5 Design Selection | 10 |
| 3.6 Implementation Plan | 11-12 |
| Chapter 4. Results Analysis and Validation | 13 |
| 4.1 Implementation of Solution | 13 |
| Chapter 5. Conclusion and Future Work | 16 |
| 5.1 Conclusion | 16 |
| 5.2 Future Work | 16-17 |
| References | 18 |
| User Manual | 19-25 |

List of Figures

|  |  |  |
| --- | --- | --- |
| Figure | Title | Page No |
| Figure 1 | **Flow Diagram** | **iii** |
| Figure 2 | **Project Timeline (Gantt chart)** | **4** |
| Figure 3 | **Visual Flow Guide** | **12** |
| Figure 4 | **Project Folder Structure** | **19** |
| Figure 5 | **Game Main Menu** | **23** |
| Figure 6 | **Gameplay Window** | **23** |
| Figure 7 | **High Score Screen** | **24** |
| Figure 8 | **Game Over Screen with Scores** | **24** |
| Figure 9 | **Exit Window** | **25** |

**ABSTRACT**

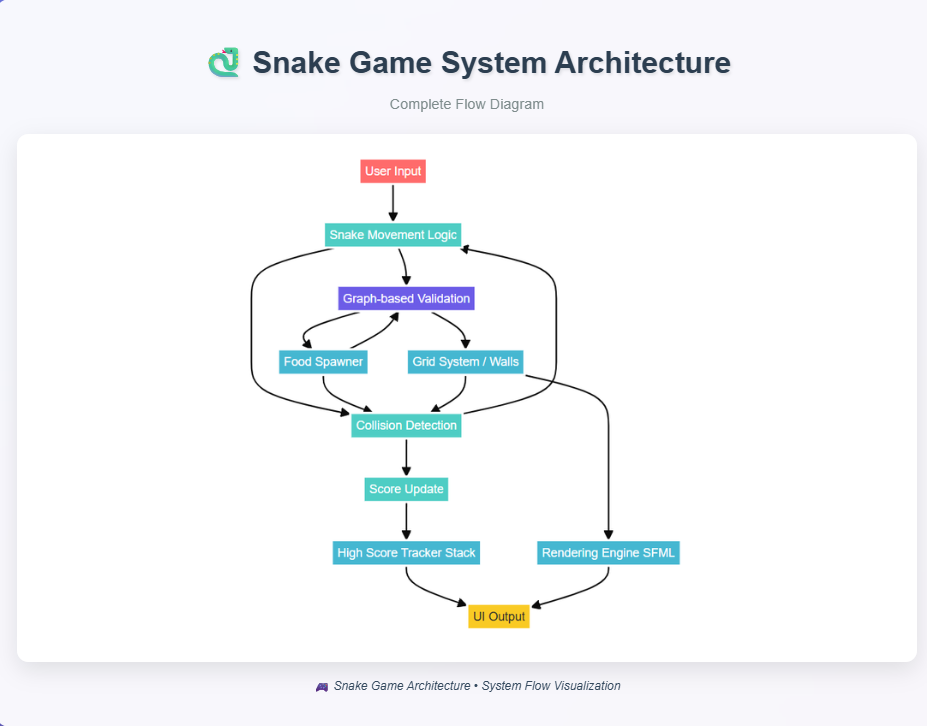
This project presents the development of an interactive **Snake Game** built using **C++** with the **SFML (Simple and Fast Multimedia Library)**. The objective is to demonstrate the practical application of **data structures and algorithms (DSA)** through an engaging, real-time game. Rather than focusing solely on gameplay, the project emphasizes applying **linked lists**, **arrays**, **stacks**, and **graphs** to address real-world software problems.

The snake’s growth and movement are dynamically managed using **linked lists and arrays**, ensuring efficient and smooth gameplay. To maintain fair mechanics, **graph-based validation** prevents food from spawning on obstacles or the snake’s body. A **stack** is used to store and display a **score history**, encouraging players to track progress and compete with themselves.

The project enhances engagement with **background music**, **sound effects**, and **custom graphics**. Its intuitive **menu interface** allows players to easily navigate between **Start**, **Pause**, **Continue**, and **High Scores** options.

Overall, this project demonstrates how **core computer science concepts** can be practically applied to build **interactive, user-friendly software**, serving as both a learning tool and a demonstration of real-world **DSA implementation** in **game development**.

**GRAPHICAL ABSTRACT**



**Figure 1:** **Flow Diagram**

The above diagram illustrates the architectural flow of the *Serpent Rush* Snake Game developed using C++ and SFML. It visualizes the interaction between user input, the snake movement logic, graph-based validation for movement safety, dynamic food spawning, collision detection, and score management through a stack data structure. Rendering and UI output are handled by the SFML graphics library. This flowchart concisely demonstrates the complete operation and modular design of the project, emphasizing both functional and interactive components.

**ABBREVIATIONS**

|  |  |
| --- | --- |
| **Abbreviation** | **Full Form** |
| **SFML** | Simple and Fast Multimedia Library |
| **GUI** | Graphical User Interface |
| **FPS** | Frames Per Second |
| **OOP** | Object-Oriented Programming |
| **CPU** | Central Processing Unit |

# CHAPTER 1

# INTRODUCTION

# Client Identification / Need Identification / Identification of Relevant Contemporary Issue

As part of the in-house training program on **Data Structures and Algorithms (DSA)**, students were required to undertake a project that effectively demonstrates the application of DSA concepts in solving real-world problems or implementing functional systems. The primary objective of this exercise was to bridge the gap between theoretical knowledge and practical implementation.

Among the suggested projects, the development of a **Snake Game using DSA** was selected due to its relevance in applying multiple core data structures within an interactive environment. While the traditional Snake Game is a well-known arcade game, this project was proposed with significant modifications to emphasize data structures such as **arrays, linked lists, stacks, and graphs** in its logic and implementation.

**The need for such DSA-oriented projects has been widely emphasized by both academic bodies and industry reports**, which highlight that knowledge of DSA is not only essential for algorithmic problem-solving but also critical in designing efficient, optimized software systems. The increasing demand for competent developers in the job market further supports the significance of practical exposure to algorithmic thinking and DSA application.

By integrating **dynamic food spawning, boundary-based wall creation, and score tracking with data structures like stacks and arrays**, this project addresses both educational and practical requirements. It serves as a **consultancy-type solution** to the common issue of theoretical concepts not being adequately internalized without their real-world implementation.

This **relevant contemporary issue**—the disconnect between theoretical DSA and practical project-based learning—is regularly documented by organizations such as **NASSCOM** and **IEEE Education Society**, underscoring the necessity of such projects in academic settings.

Therefore, this project contributes directly to enhancing practical understanding and building proficiency in data structures through the design, development, and testing of an interactive, structured system.

**1.2 Identification of Problem**

While traditional Snake games serve as learning models for fundamental programming, they often lack advanced implementations that demonstrate the integration of **data structures**, **graphics libraries**, and **game logic optimization**. Furthermore, existing implementations rarely showcase:

* **Dynamic map generation or obstacle validation**
* **Graph-based movement validation**
* **Persistent or tracked score histories**

Thus, there exists a clear gap in educational projects that incorporate advanced data handling with real-time graphics and sound systems, bridging the theoretical concepts taught in academia with practical execution.

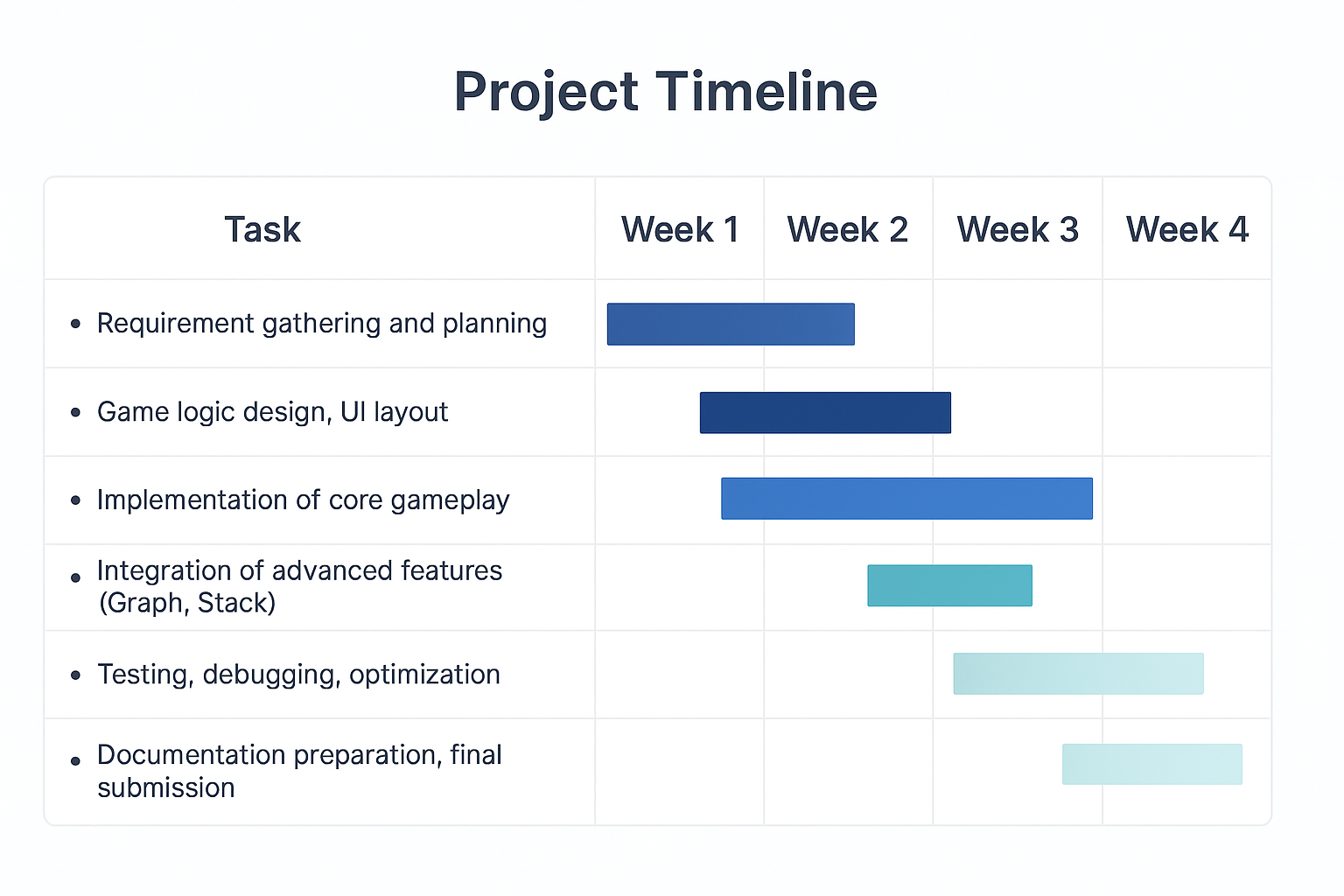
**1.3 Identification of Tasks**

To address the identified problem, the project is divided into **structured tasks**, each focusing on specific outcomes to ensure clarity and systematic execution.

**Task Breakdown:**

|  |  |
| --- | --- |
| **Task** | **Description** |
| **Requirement Analysis** | Define game features, gameplay mechanics, required data structures |
| **System Design** | Design system flow diagrams, class structures, and graphical architecture |
| **Implementation** | Develop game logic in C++, implement SFML rendering, handle input |
| **Data Structure Integration** | Use stacks for score history, arrays/vectors for snake body, graphs for wall collision validation |
| **Testing and Debugging** | Verify correct gameplay behavior, score tracking, collision validation |
| **Documentation and Final Report** | Prepare detailed project report following standard academic format |

**1.4 Timeline**



**Figure 2: Project Timeline (Gantt chart)**

**1.5 Organization of the Report**

This report is organized into structured chapters for clarity and systematic understanding:

* **Chapter 1: Introduction**  
  Identifies the problem, justifies the need for the project, outlines objectives, tasks, and timeline.
* **Chapter 2: Literature Survey**  
  Overview of previous research, related work, and proposed solutions relevant to the problem.
* **Chapter 3: Design Flow/Process**  
  Includes system design flow, evaluation of features, design constraints, and finalized architecture.
* **Chapter 4: Results Analysis And Validation**  
  Covers technologies used, key features, implementation methodology, and system development details. Presents testing results, analysis of project outcomes, and validation of features.
* **Chapter 5: Conclusion and Future Scope**  
  Summarizes project achievements and suggests potential improvements or future work.

**CHAPTER 2**

**LITERATURE REVIEW / BACKGROUND STUDY**

**2.1 Timeline of the Reported Problem**

Games have always played a vital role in the development of logical and problem-solving abilities in learners. Snake games, specifically, gained popularity during the late 1990s with their inclusion on early mobile phones like the Nokia 3310. Over the years, versions of snake games have appeared across devices, from feature phones to desktops and smartphones.  
With the evolution of programming, educators began to use games like Snake as projects to introduce **data structures and algorithms (DSA)** concepts practically. Research studies emphasize that **game-based learning improves engagement and enhances algorithmic understanding.** Various academic courses have integrated game development as project-based learning.

**2.2 Proposed Solutions**

Several versions of Snake games have been developed in different programming languages like Java, Python, and C++. Early implementations were mostly linear and used simple arrays for storing positions. Recent projects and open-source contributions have introduced **Linked Lists, Queues, Stacks, and Graphs** to model the snake’s body, store score history, and validate path movements in advanced snake games. Integration of **graphical libraries** like SFML (Simple and Fast Multimedia Library) has enabled rich visual experiences.

**2.3 Bibliometric Analysis**

| **Source** | **Approach** | **Strength** | **Drawback** |
| --- | --- | --- | --- |
| Academic Projects | Basic snake with arrays | Easy to implement | No advanced DSA usage |
| Open-source (GitHub) | Linked List for snake movement | Dynamic growth, better structure | Less emphasis on UI |
| Educational Papers | Stack for Score History | Highlights DSA applications | Mostly theoretical |
| Game Engines | Complete engines with graphics | Excellent UI and effects | Overkill for academic purposes |

**2.4 Review Summary**

The literature indicates that **Snake games are highly effective for demonstrating data structures practically**. However, while many focus on gameplay or visuals, fewer projects integrate **multiple DSA concepts cohesively** (Linked Lists, Stacks, Graphs).  
This project bridges that gap by developing an interactive snake game **with advanced data structures and enhanced UI using SFML.**

**2.5 Problem Definition**

There is a need for a **practical, interactive, and educational Snake game** that goes beyond basic implementations to provide a deeper understanding of how various **data structures** can work together. This project focuses on building a Snake game using **Linked Lists** (for snake body), **Stacks/Queues** (for score history), and **Graphs** (for wall and path validation), implemented in **C++ with SFML**, to offer both entertainment and educational value.

**2.6 Goals/Objectives**

* Develop a Snake Game using advanced data structures.
* Integrate Linked Lists for real-time snake body management.
* Implement Stack/Queue to maintain score history.
* Utilize Graphs for movement validation and obstacle generation.
* Provide interactive GUI using SFML.
* Demonstrate how DSA concepts apply in real-world applications.
* Ensure smooth gameplay with increasing difficulty and multiple features.

**CHAPTER 3**

**DESIGN FLOW / PROCESS**

**3.1 Evaluation & Selection of Specifications/Features**

After analyzing previous works and the potential of using Snake games for learning DSA, the following **key features** were finalized for this project:

|  |  |
| --- | --- |
| **Feature** | **Justification** |
| **Snake movement using Linked List** | To demonstrate dynamic data management |
| **Dynamic Food Generation** | Random spawning for enhanced challenge |
| **Obstacle Generation (Walls)** | Using Graph-based validation for advanced levels |
| **Score Tracking (Stack/Queue)** | To maintain player history and past performance |
| **Pause, Resume, and Restart** | For better user control over gameplay |
| **Graphical Interface with SFML** | To provide professional and engaging visuals |
| **Sound Effects** | Enhance player engagement and feedback |
| **Score History Visualization** | Educational: Showing how stacks/queues store data |

**3.2 Design Constraints**

|  |  |
| --- | --- |
| **Constraint Type** | **Details** |
| **Regulations** | None required (academic project) |
| **Economic** | Must use free/open-source tools only (SFML) |
| **Health & Safety** | Screen time awareness for prolonged usage |
| **Professional/Ethical** | Code clarity, modularity, and attribution |
| **Environmental** | Digital-only; no environmental impact |
| **Social & Political** | Neutral – purely educational and recreational |
| **Cost** | Zero development cost using existing resources |

**3.3 Analysis and Feature Finalization subject to Constraints**

|  |  |  |
| --- | --- | --- |
| **Proposed Feature** | **Action** | **Reason** |
| Advanced AI Snake | **Removed for scope** | Beyond current time/resource constraints |
| Multiplayer functionality | **Postponed (Future)** | Adds complexity; left for future expansion |
| Visual Themes | **Kept** | Adds to usability and engagement |
| Score History (Stack) | **Finalized** | Key for educational DSA demonstration |

Finalized features prioritize **DSA learning integration**, simplicity, and engagement for the user.

**3.4 Design Flow (Alternative Designs)**

|  |  |
| --- | --- |
| **Approach** | **Description** |
| **Approach 1 (Basic Snake)** | Single Linked List for snake body only; simple wall collisions; basic score display. |
| **Approach 2 (Advanced with DSA Integration)** | Linked List + Graphs (for obstacles) + Stack (for score history); graphical UI and interactive buttons. |

**3.5 Design Selection**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Approach 1 (Basic)** | **Approach 2 (Advanced with DSA)** |
| **Educational Value** | ✗ | ✅ |
| **Complexity** | ✅ (Simple) | Moderate |
| **Demonstrates DSA** | ✗ | ✅ |
| **Visual Appeal** | Basic | ✅ |

**3.6 Implementation Plan / Methodology**

**Tools & Technologies Used:**

* **Programming Language:** C++
* **Graphics Library:** SFML 2.6.1
* **IDE/Environment:** Visual Studio Code with MinGW-w64
* **Other Tools:** HTML for Diagram Generation, Online free audio assets
* **Version Control:** Git + GitHub for version control

****

**Figure 3: Visual Flow Guide**

**CHAPTER 4**

**RESULTS ANALYSIS AND VALIDATION**

**4.1 Implementation of Solution**

The Snake Game was successfully developed using **modern programming practices** and **software tools** to ensure a professional and educational outcome. The following outlines the specific stages of implementation and the corresponding tools:

**Tools Used for Implementation:**

|  |  |
| --- | --- |
| **Activity** | **Tools Used** |
| **Programming** | C++ with SFML 2.6.1 |
| **Graphics Rendering** | SFML Library (Graphics, Window, Audio) |
| **Design Drawings (Flowcharts/Diagrams)** | Use HTML and CSS |
| **Sound Effects** | Audio assets |
| **Report Preparation** | MS Word |
| **Project Management** | Timeline via Gantt chart |
| **Version Control** | Git |

**Validation and Testing:**

|  |  |
| --- | --- |
| **Component** | **Validation Method** |
| **Snake Movement** | Verified dynamically during gameplay; ensured correct linked list updates per movement |
| **Wall Collision** | Checked via **Graph-based validation** to confirm obstacle placement and detection |
| **Food Generation** | Tested across multiple runs to confirm randomness and no overlap with walls/snake |
| **Score History** | Verified that previous scores were pushed to a **stack** and displayed on screen |
| **User Interface** | All buttons (Start, Continue, Music Toggle) tested for responsiveness and accuracy |
| **Sound** | Audio cues confirmed for food generation and consumption |

**Testing Scenarios Performed:**

1. **Normal Gameplay**: Tested complete game cycles to ensure snake movement, food consumption, and game over logic worked.
2. **Boundary Testing**: Snake directed toward edges/walls to validate collision detection.
3. **Stress Testing**: Intentional rapid key presses to check program stability.
4. **Feature Testing**: Music toggle, pause/resume flow, high score view, and graphical representation verified.

**Result Summary**

* **Expected Outcome Achieved**: All core features were successfully implemented.
* **User Engagement Tested**: Visuals, audio, and interactivity met educational and recreational expectations.
* **Data Structures Demonstrated**: Linked Lists (Snake body), Graph (Wall validation), and Stack (Score history) **successfully integrated.**

**Data Validation Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Status** |
| Snake collides with wall | Game Over | Game Over triggered | ✅ |
| Snake consumes food | Grows & Score increases | Works as expected | ✅ |
| Random Food Generation | Appears away from obstacles | Works as expected | ✅ |
| High Score Stack | Shows last 5 scores in order | Works as expected | ✅ |

**CHAPTER 5**

**CONCLUSION AND FUTURE WORK**

**5.1 Conclusion**

The developed project, *Serpent Rush - A Snake Game with Advanced Features*, successfully fulfills its primary goal of providing an engaging and interactive gaming experience while demonstrating the application of fundamental Data Structures and Algorithms (DSA). The game incorporates essential DSA concepts such as arrays (for the snake body), stacks (for score history), and graphs (for food spawning logic and wall collision validation).

The expected outcome was to create a fully functional, visually appealing Snake Game, complete with features like pause/resume, high score tracking, background music control, and interactive menus. These goals have been met. The scoring system, increasing difficulty, and user interface add to the gaming experience.

While most objectives were achieved, certain advanced features like AI-driven snake behavior or multiplayer modes were not included within the current scope due to time limitations.

**5.2 Future Work**

There is significant scope to enhance this project in the future:

* **Artificial Intelligence Integration:** Implementing an AI-controlled snake or enemy snake to increase competition and complexity.
* **Dynamic Wall and Level Generation:** Introducing levels with changing wall layouts to make gameplay more challenging.
* **Multiplayer Mode:** Adding support for multiplayer gameplay either locally or over a network.
* **Advanced Graphics and Animations:** Using improved graphic assets, smoother animations, and better sound design.
* **Cross-Platform Support:** Expanding the project for platforms like Android, Linux, or web-based versions.
* **Data Persistence:** Storing high scores or player progress in external files or databases for a persistent gaming experience.

These additions will not only improve the user experience but also serve as valuable learning opportunities to implement advanced programming concepts and further strengthen DSA understanding.

**REFERENCES**

1. **SFML Official Documentation**  
   <https://www.sfml-dev.org/documentation/2.6.1/>  
   *(For understanding and implementation of SFML graphics, window, audio, and event handling in C++)*
2. Bjarne Stroustrup, **"The C++ Programming Language"**, 4th Edition, Addison-Wesley, 2013.  
   *(For core C++ programming concepts relevant to the project’s structure and syntax)*
3. E. Horowitz, S. Sahni, and Dinesh Mehta, **"Fundamentals of Data Structures in C++"**, 2nd Edition, University Press, 2008.  
   *(For concepts related to Linked Lists, Stacks, and Queues used in the project logic)*
4. **GeeksforGeeks**, “Data Structures and Algorithms in C++”  
   <https://www.geeksforgeeks.org/data-structures/>  
   *(Used for algorithmic understanding, especially regarding stack implementation for score history and collision logic)*
5. Various **royalty-free images and music assets** obtained from publicly available online sources.  
   *(Used for graphical (apple image) and audio (background music and effects) assets in the project)*
6. **Stack Overflow**  
   <https://stackoverflow.com/>  
   *(Community platform referenced for resolving implementation-specific coding queries)*

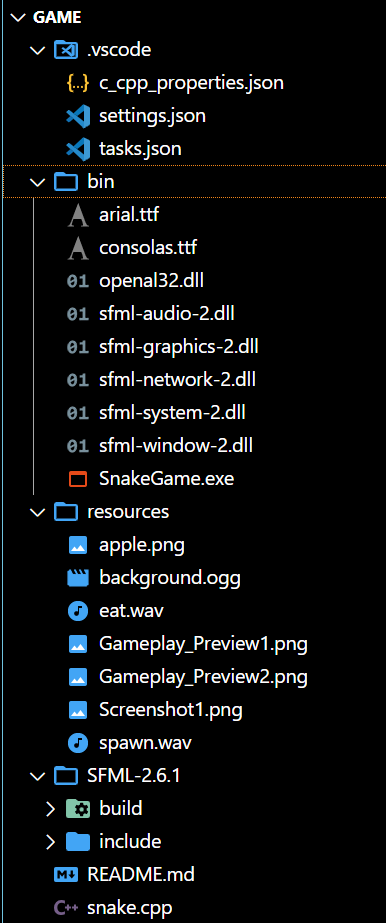
**USER MANUAL**

**Project Title: Serpent Rush – Advanced Snake Game**

**1. System Requirements**

|  |  |
| --- | --- |
| **Requirement** | **Details** |
| **Operating System** | Windows 10 or later |
| **Compiler** | MinGW-w64 (with g++ compiler) |
| **Graphics Library** | SFML 2.6.1 |
| **IDE (Recommended)** | Visual Studio Code (VS Code) |
| **RAM** | Minimum 2 GB |

**2. Project Folder Structure**



**Figure 4: Project Folder Structure**

**3. Installation Instructions**

**✅ Step 1: Install MinGW-w64**

* Download **MinGW-w64** and set the bin/ folder path to **Environment Variables > PATH**.
* Example: C:\mingw64\bin

**✅ Step 2: Install SFML 2.6.1**

* Place SFML-2.6.1 in your project directory.
* Configure **include** and **lib** paths in tasks.json or VS Code settings.

**✅ Step 3: Configure VS Code**

* Ensure your **tasks.json** file includes paths for SFML:
* {
* "version": "2.0.0",
* "tasks": [
* {
* "label": "Build SnakeGame",
* "type": "shell",
* "command": "g++",
* "args": [
* "-I",
* "D:\\Mastering\_in\_DSA\\Game\\SFML-2.6.1\\include",
* "-L",
* "D:\\Mastering\_in\_DSA\\Game\\SFML-2.6.1\\build\\lib",
* "snake.cpp",
* "-lsfml-graphics",
* "-lsfml-window",
* "-lsfml-system",
* "-lsfml-audio",
* "-o",
* "D:\\Mastering\_in\_DSA\\Game\\bin\\SnakeGame.exe"
* ],
* "group": {
* "kind": "build",
* "isDefault": true
* },
* "problemMatcher": [
* "$gcc"
* ]
* }
* ]
* }

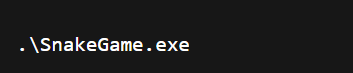
**4. How to Compile and Run**

**✅ Using Visual Studio Code (Recommended)**

1. **Open the Project Folder**  
   Open the folder containing the project (snake.cpp, .vscode/, bin/, etc.) in **Visual Studio Code**.
2. **Build the Project**
   * Press **Ctrl + Shift + B** to run the pre-configured build task.
   * This will create **SnakeGame.exe** in the bin/ directory.
3. **Run the Game**
   * **Option 1**: Navigate to the bin/ directory using **File Explorer** and **double-click** on SnakeGame.exe to start the game.
   * **Option 2 (Recommended)**:
     + Open the **integrated terminal** in VS Code.
     + Change directory to bin if not already there:



* + - Run the game using:



1. **Playing the Game**

|  |  |
| --- | --- |
| **Action** | **Control** |
| **Move Up** | ↑ Arrow Key |
| **Move Down** | ↓ Arrow Key |
| **Move Left** | ← Arrow Key |
| **Move Right** | → Arrow Key |
| **Pause/Resume** | Spacebar |
| **Start New Game** | Click Start Game (in menu) |
| **View High Scores** | Click High Scores |
| **Enable/Disable Music** | Click Music ON/OFF button |

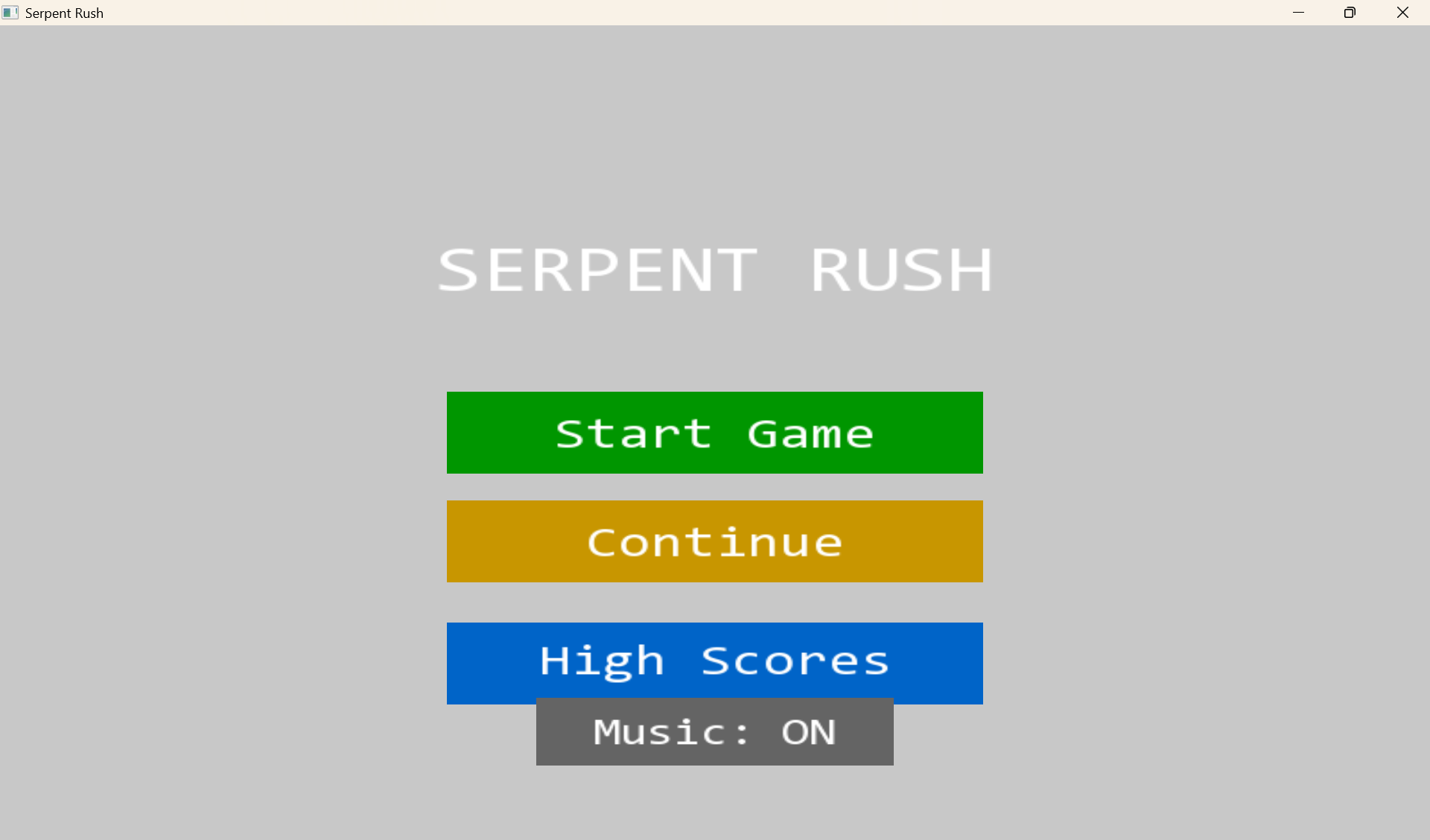
1. **Features**

* Dynamic Food Spawning
* Walls and Boundaries
* Menu Navigation (Start, Continue, High Scores)
* Score History (using Stack Data Structure)
* Sound Effects (Eating/Spawning)
* Background Music Toggle
* Graphical Display using SFML

1. **Troubleshooting**

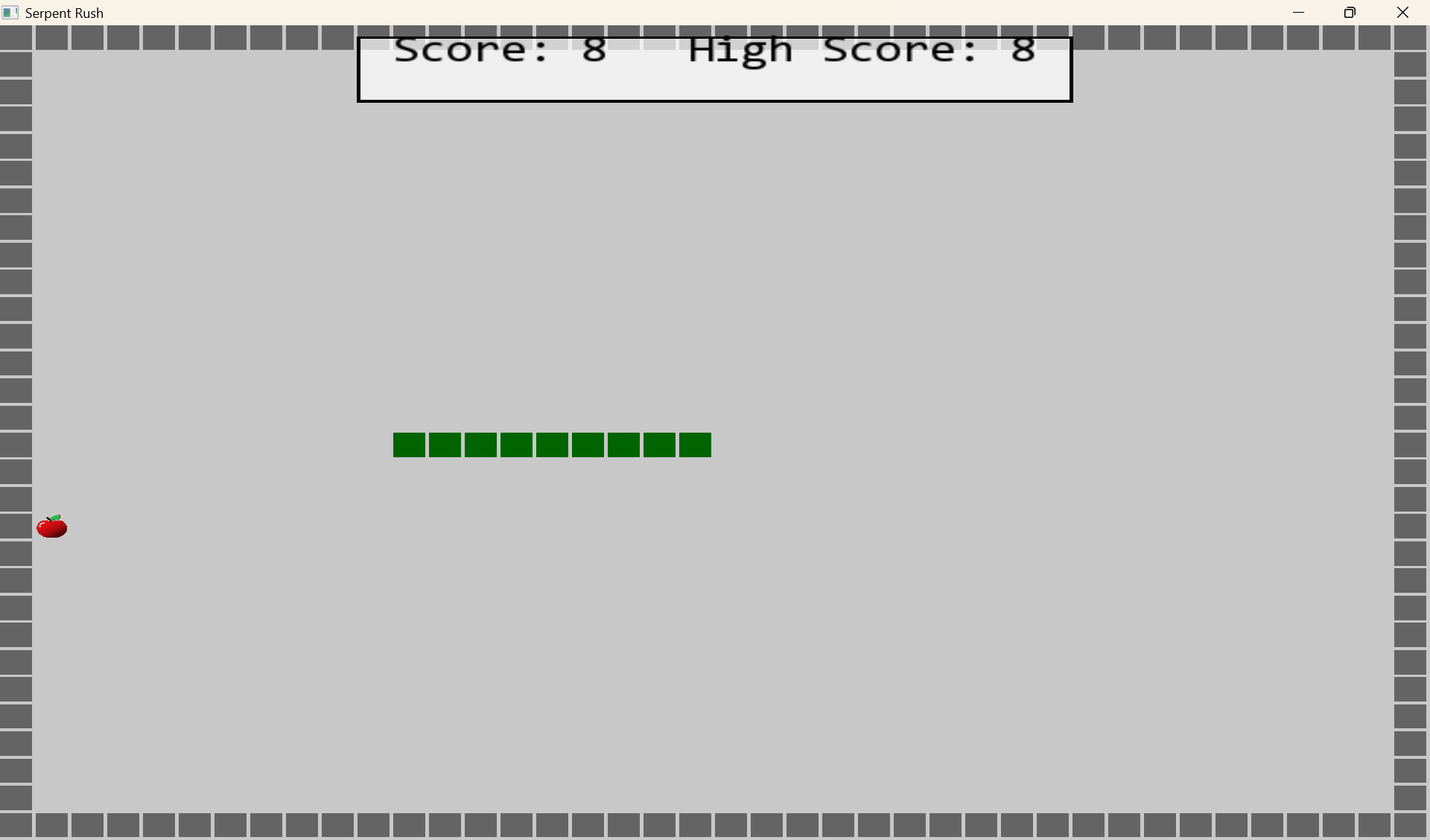
|  |  |
| --- | --- |
| **Problem** | **Solution** |
| SFML not found error | Check **SFML include/lib paths** in settings |
| Console window closes fast | Run from Terminal instead of clicking .exe |
| Sounds not playing | Ensure **resources** folder has all required files |
| Font not displaying properly | Make sure **consolas.ttf** is present in folder |

1. **Example Screenshots:**
   * + 1. **Main Menu**

****

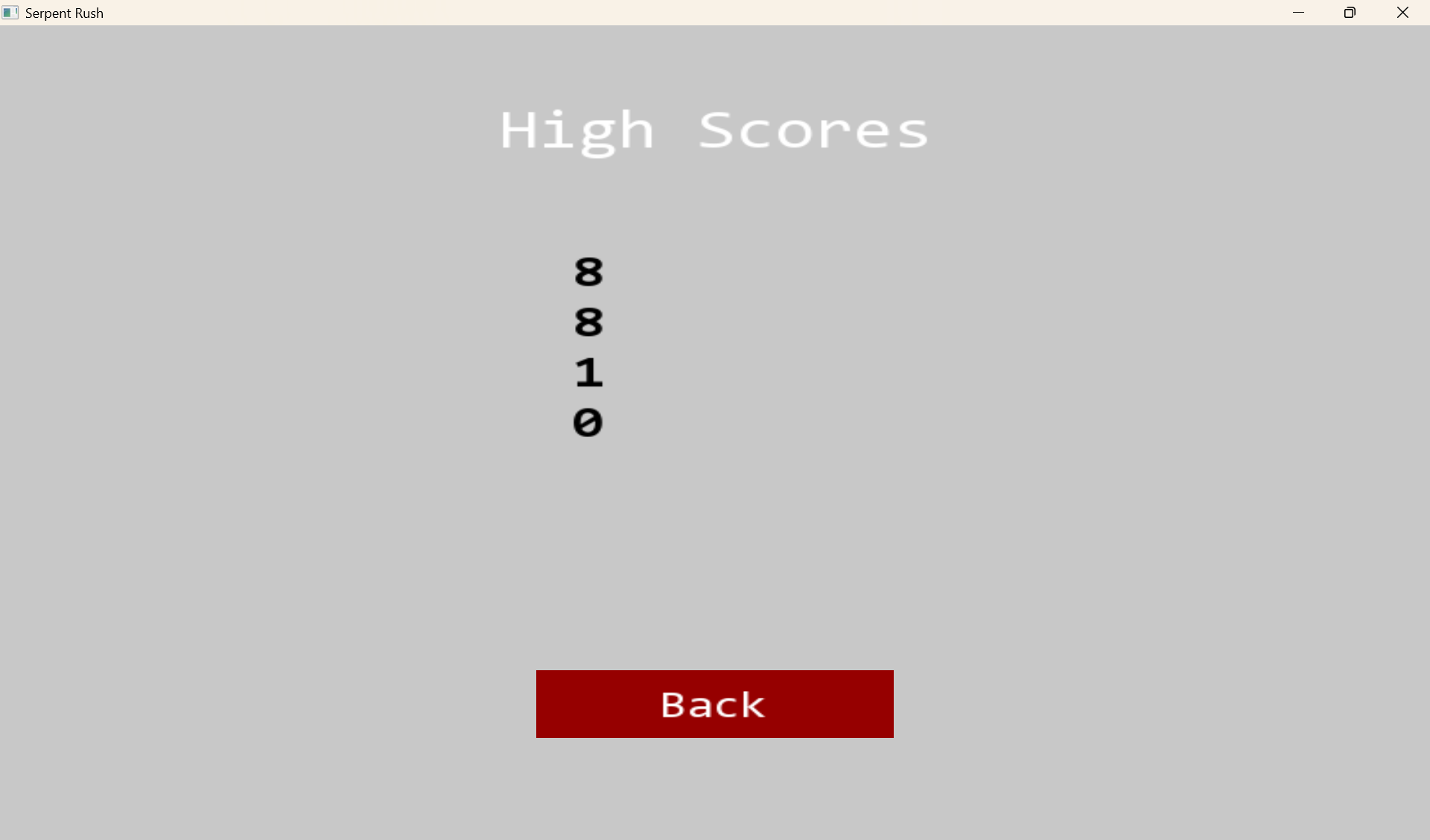
**Figure 5: Game Main Menu**

* + - 1. **Gameplay Window**

****

**Figure 6: Gameplay Window**

* + - 1. **High Scores Screen**

****

**Figure 7: High Score Screen**

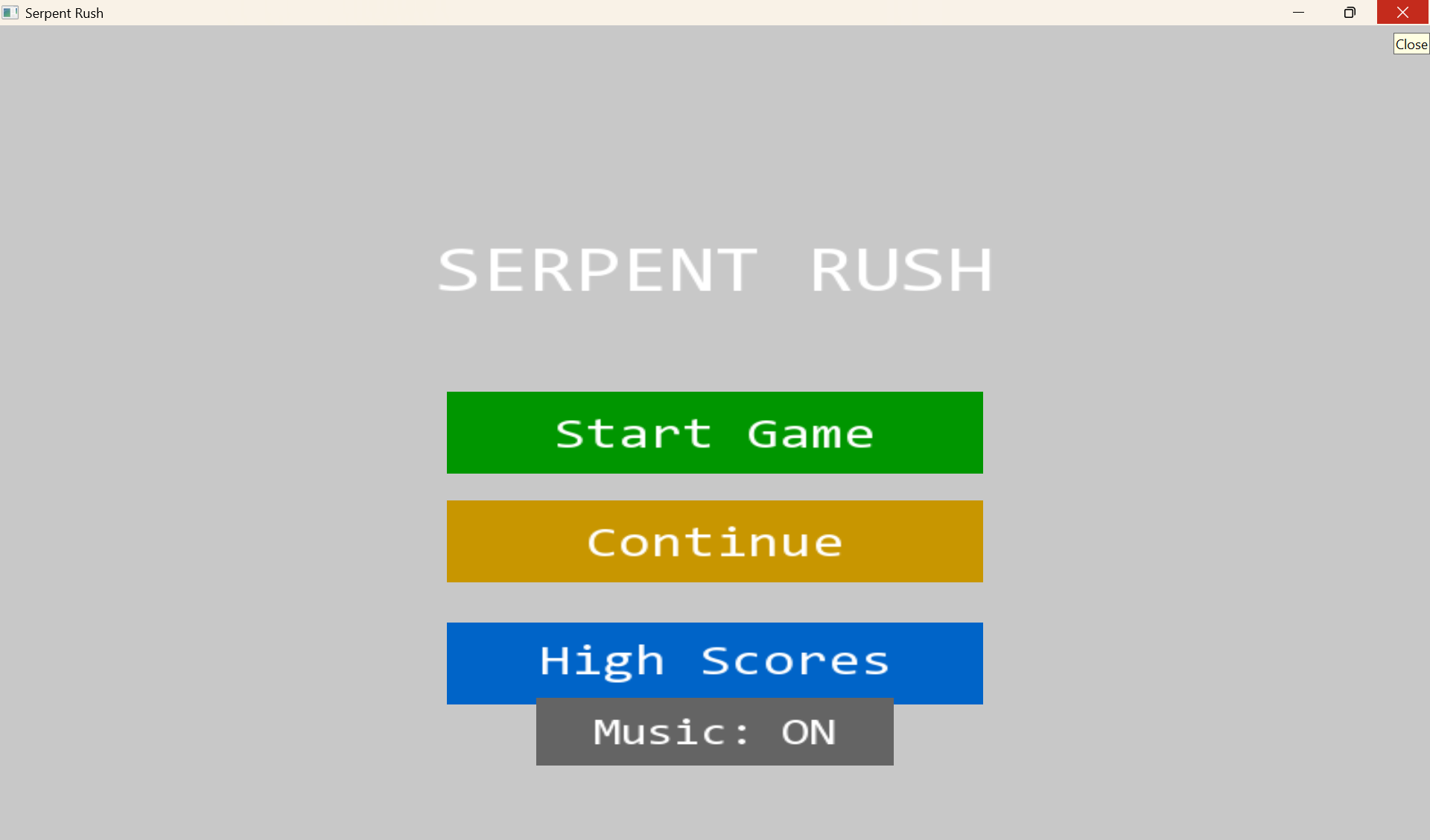
* + - 1. **Game Over Screen with Scores**

****

**Figure 8: Game Over Screen with Scores**

**9. Exit**

* **Close Game → Click on Window Close (X) button or press Alt + F4.**

****

**Figure 9: Exit Window**