Snake’s World — Technical Documentation

Contributions-

Piyush Khanna- Game Logic + Game Design
Tanubhav Juneja- Game Logic + Game Design
Rashmi Desai- Game Logic + Documentation
Gunjal Manocha- Documentation + QA

1. Project Overview

Snake’s World is a 2D grid-based C++ game built with the **Raylib** library.
Features include:

- Keyboard-controlled snake movement
- Multiple fruit textures
- Snake growth & increasing speed
- Eating & collision sound effects
- Game states: Menu, Running, Game Over
- UI buttons (start, exit, restart)
- Runtime high-score tracking

Main Classes:

1. **Snake** – movement, growth, rendering
2. **Food** – spawning, textures, collision logic
3. **Game** – gameplay loop, scoring, audio, states

---

2. System Architecture

2.1 Files

File	Purpose
`main.cpp`	Game loop, UI, state transitions
`button.hpp`	Custom clickable button widget
Raylib	Rendering, audio, timing, input

2.2 High-Level Structure

```
main.cpp → Game → { Snake, Food[3] }
```

---

3. Game States

`Main Menu → Running → Game Over → (Restart / Exit)`

---

4. Global Configuration

- **cellsize**: grid tile size (30 px)
- **cellcount**: grid dimension (25×25)
- **offset**: board margin
- **speed**: snake movement interval
- **score / high_score / temp_score**: tracking system
- **lastUpdateTime**: movement timing

---

5. Class Documentation

5.1 Snake

Handles movement, growth, and rendering.

Key Members:

- `body`: deque of grid positions
- `direction`: current movement vector
- `addSegment`: grow flag

Key Methods:

- **Draw()** – renders segments
- **Update()** – moves snake, grows if needed
- **Reset()** – restores starting state

5.2 Food

Represents fruits placed on the grid.

Static:

- `textures[4]` – shared textures
- `loaded` – prevents double-loading

Instance:

- `position` – grid location
- `textureIndex` – selected texture

Methods:

- **Constructor** – loads textures, randomizes texture & position
- **GenerateRandomCell()** – random tile
- **GenerateRandomPos()** – avoids snake body
- **Draw()** – renders fruit

5.3 Game

Central gameplay controller.

Members:

- `snake`
- `vector<Food> fruits`
- `score`, `speed`, `running`, `game_over`
- `eat`, `wall` sound assets

Methods:

- **Constructor/Destructor** – audio & asset setup
- **CheckCollisionWithFood()** – scoring, growth, respawn
- **CheckCollisionWithEdges()** – out-of-bounds death
- **CheckCollisionsWithTail()** – self-collision death
- **Update()** – main game logic
- **GameOver()** – reset & update high score

---

6. Main Loop

- Read keyboard & mouse input
- Update game state when movement timer triggers
- Draw buttons, texts, and game elements
- Switch between menus and gameplay

---

7. Data Flow

```
Input → Direction Logic → Update → {Food/Wall/Tail checks} → Render
```

---

8. Known Issues / Improvements

Issue	Fix
Passing deque by value	Use const reference
WaitTime freezes loop	Remove blocking delay
Fruit overlap	Add fruit-fruit collision checks
Float positions	Switch to int grid values
MP3 compatibility	Prefer WAV files
Duplicate speed reset	Clean GameOver() logic

---

9. Dependencies

- **Raylib** – rendering, sounds, input
- **Raymath** – vector utilities

Assets:
Textures: start, exit, restart, food1–food4
Sounds: eat, wall

---

10. Conclusion

The project is well-organized, functional, and uses Raylib effectively. With small optimizations and fixes, it can be further improved and extended.