Programming Guidelines

Owner	Shlok Prabhu
Tags	

1. Build environment setup

Toolchain + OS

OS: Windows

Compiler: LLVM

Build System: CMake (make generator)

IDE

Primary: VS Code

Extensions:

- C++ Language Support (clangd recommended)
- TODO / FIXME
- Git integration
- Git Blame
- CMake Tools
- Doxygen preview
- Google Test ← VS Code does not come with unit test handler

2. Repo file structure

Canonical Layout

`CMakeLists.txt` (root)

`src/` (implementation)

`include/` (public headers)

`tests/` (unit/integration tests)

`docs/` (design notes, diagrams, Doxygen config)

`scripts/` (build helpers)

`.github/` (CI workflows, templates)

Rules

1. Avoid committing large binaries.
2. Headers should not depend on private headers from `src/` .

3. Git Workflow

.gitignore

1. Ignore build outputs: `build/`, `out/`, `cmake-build-*`
2. Ignore all IDE files: `.vscode/` , `.idea/`
3. Ignore local env files: `.env`, caches, logs

Branching

1. Main is a protected branch. All edits will require a PR with one approval to ensure merge.
2. Work will happen on branches:

   `feat/<short-desc>` when introducing a new feature or functionality

   `bug/<short-desc>` when resolving an issue or bug in code

   `chore/<short-desc>` when not updating code logic

Pull Requests

1. No direct pushes to main.
2. See PR template.

Commit Messages

Commits will follow the conventional commits specification.

> `<type>: <subject>`

Types

- `feat`: A new feature.
- `fix`: A bug fix.
- `docs`: Documentation only changes.
- `style`: Code formatting, white-space, etc..
- `refactor`: A change that neither fixes a bug nor adds a feature.
- `test`: Adding missing tests or correcting existing tests.
- `chore`: Routine tasks or changes to build process/tooling.

4. Styling & Standards

C++ Standard

Standard: C++20

Format: Use clang-format with committed `.clang-format`; run with PR.

Style

Headers: `#pragma once`

Includes: minimal; use forward declarations

Namespaces: avoid `using namespace` in headers

Const-correctness: default to `const` where applicable

Ownership: RAII policy. Use `std:unique_ptr` or `std:shared_ptr`. Raw pointers are okay when they do not own memory.

Errors: Error codes and exceptions

Warnings: ?

Conventions

Types/Classes: `PascalCase`

Functions/variables: `snake_case`

Constants: `UPPER_SNAKE_CASE`

Files: `snake_case.cpp/.hpp`

5. Documentation

What must be documented

- Public classes/functions in `include/`
- Non-trivial algorithms, invariants, tricky edge cases
- Ownership/lifetime rules for pointers/references
- Any API that other modules call

How to write docs (recommended)

- Use Doxygen blocks on public interfaces:
    - `@brief` one-liner
    - `@param` for each parameter (include units/ranges if relevant)
    - `@return` description
    - `@throws` if exceptions are used
    - `@note` for relevant notes
    - `@warning` for relevant warnings
- Keep "why" in docs when the "how" is obvious from code
- Put examples in `docs/` or as `@code ... @endcode`

- Use per-line triple `///` , rather than `/** */`

```
/// Represents the snake game.
class SnakeGame {
public:
    /// Starts the main game loop.
    /// @return Snake object
    /// @warning Requires valid object
    /// @throws Error
    Snake get();

    /// Moves the snake in the given direction.
    /// @note requires System Vector2 object Direction, not Godot Vector2 object Direction
    /// @param direction Direction to move the snake
    void move(Direction direction);

private:
    /// Current score.
    int score_;
};
```

6. Templates

Pull Request

```
## What
Briefly describe what this PR does.

## Why
Why is this change needed?

## Testing
How did you verify it works?
```

```
## Checklist
- [ ] Builds successfully
- [ ] Follows style guidelines
- [ ] No leftover debug code
- [ ] Ready for review
```

Bug Report

```
---
name: Bug
title: "[BUG] "
labels: bug
---

## Problem
What is wrong?

## Steps
1.
2.
3.

## Expected
What should happen?

## Actual
What happened instead?
```

Feat Request

```
---
name: Feature
title: "[FEAT] "
```

```
labels: feature
---

## Goal
What needs to be added?

## Notes
Optional details.
```