**Feeding Frenzy‑ OOP2 – README**

**1  Authors**

| **Name** | **Student ID** |
| --- | --- |
| **Abed Alrhman Khalil** | 315270728 |
| **Rayan Abdalhalim** | 325229102 |

**2  Project Overview**

This repository contains a C++23 implementation of a small 2D fish-eat-fish game inspired by the classic **\*\*Feeding Frenzy\*\***. The project was developed as the final assignment for an Object-Oriented Programming course.

- Player controlled fish that grows by eating smaller fish

- Predators and hazards that must be avoided

- Power‑ups such as speed boost and frenzy mode

- Multiple stages with increasing difficulty

- Score and high‑score system saved to disk

- Structured using a simple state machine and light ECS architecture

**3  Controls**

| **Action** | **Keyboard** |
| --- | --- |
| Move | **W A S D** / Arrows |
| Pause | **P** |
| Confirm | **Enter** |

**4  Gameplay Mechanics**

* **Player growth & scoring:** The player grows by eating smaller fish and can tail‑bite bigger ones. Growth/score handling appears in Player::grow and related methods.
* **Bonus items:** Items spawn as Starfish, PearlOyster or PowerUp. Oysters can contain pearls and may harm the player when closing.
* **Power‑ups:** The PowerUpType enum lists Score Doubler, Frenzy Starter, Speed Boost, Freeze, Extra Life and Add Time. Concrete classes such as FreezePowerUp, ExtraLifePowerUp, SpeedBoostPowerUp and AddTimePowerUp provide the effects. Durations are defined in constants (e.g., Freeze lasts 5 s).
* **Hazards:** The game uses two hazard types—Bomb and Jellyfish. Bombs explode, and jellyfish stun and push the player on contact.
* **Special fish:** Spawn rates for Barracuda, Pufferfish, Angelfish and PoisonFish are configurable via SpecialFishConfig. Each fish has unique behavior, e.g. pufferfish inflate to push players and poisonfish invert controls with applyPoisonEffect on collision.
* **Frenzy system:** Rapid eating builds up a Frenzy or Super Frenzy multiplier.
* Environment effects: The environment cycles through CoralReef, OpenOcean and KelpForest biomes, with day/night settings and ocean currents influencing movement.
* Overall gameplay centers on eating to grow, avoiding hazards, using power‑ups, exploiting special fish, and managing the environment to reach stage score targets.

**5  Codebase Layout**

├── CMakeLists.txt

├── src/

│ ├── core/ # Game loop, State machine, Resource holders

│ ├── states/ # MenuState, GameplayState, GameOverState, …

│ ├── entities/ # Player, GenericFish, Barracuda, PowerUp, …

│ ├── systems/ # CollisionSystem, ScoreSystem, ParticleSystem

│ ├── managers/ # FishSpawner, BonusItemManager, HUDController

│ └── utils/ # AnimatedSprite, Timers, Math helpers

**6  Detailed Architecture**

**6.1 Main Loop**

Initialize → Load Resources → Push IntroState → while (window.open) {

handleInput();

currentState.update(dt);

currentState.render();

}

* Fixed‑time physics tick (1/120 s) decoupled from variable render FPS.
* Subsystem order: **Physics → AI → Collision → Gameplay Logic → UI → Render**.

**6.2 State Machine**

StateManager maintains a stack so transitions (e.g., pause overlay) are trivial.

**6.3 Entity Component Slice (Light ECS)**

Instead of a full ECS library we keep Entity as a thin polymorphic base with virtual update/draw. Shared functionality (position, velocity, collider) is mixed‑in via CRTP traits to avoid RTTI overhead.

**6.4 Resource Management**

A templated ResourceHolder<ID, sf::Texture> loads assets on demand and guarantees unique non‑copyable storage (move‑only).

**6.5 Collision Detection**

* **Broad phase:** Uniform grid hashing (cell = 64 px).
* **Narrow phase:** Continuous Swept‑AABB; fish assumed axis‑aligned.

**6.6 AI & Pathfinding**

Predators run a 2‑tier behavior:

1. Heuristic *seek* if distance < vision radius (600 px).
2. A\* path over 60×34 virtual grid when obstacles present (rocks, coral).

**7  Design Patterns Used**

| **Pattern** | **Location** | **Purpose** |
| --- | --- | --- |
| **Singleton** | ResourceHolder, MusicPlayer | Global access to heavy shared resources |
| **State** | StateManager & subclasses | Switch game modes cleanly |
| **Factory Method** | FishSpawner::spawn() | Create fish types based on stage config |
| **Strategy** | BarracudaAI vs GenericFishAI | Swap behaviors at runtime |

**8  Core Algorithms**

1. **Barracuda Predictive Chase**

Grids are cached per frame; open list is a binary heap.

1. **Dynamic Spawn‑Rate Balancer**

Logistic growth curve:

1. **Continuous Swept‑AABB**

Detects first time of overlap t\_entry, resolves at t\_entry - ε to avoid tunneling.

**9  Data Structures**

struct EntityHandle { uint32\_t id; uint8\_t generation; };

std::vector<std::unique\_ptr<Entity>> entities;

std::unordered\_map<TextureID, sf::Texture> textures;

std::priority\_queue<TimedEvent, std::vector<TimedEvent>, std::greater<>> events;

**10  Known bugs**

1. When the level completed the medium fish doesn't swim away.
2. After bonus level its not showing what we have in the next level.

**File list – each file name with a 1-line description**

include/Core/Game.h – declares the Game class and main loop.

include/Core/GameConstants.h – holds constants for gameplay and UI.

include/Core/GameExceptions.h – basic exception classes for resources.

include/Core/MusicPlayer.h – wrapper for background music playback.

include/Core/ResourceHolder.h – generic template for loading assets.

include/Core/SoundPlayer.h – manages sound effect playback.

include/Core/State.h – base class for all game states.

include/Core/StateManager.h – stack-based manager for states.

include/Core/StateUtils.h – helpers and type traits for states.

include/Entities/AdvancedFish.h – fish with special movement patterns.

include/Entities/Angelfish.h – friendly angelfish implementation.

include/Entities/Barracuda.h – aggressive large fish enemy.

include/Entities/BonusItem.h – base class for collectible bonuses.

include/Entities/Entity.h – base drawable object with position.

include/Entities/ExtendedPowerUps.h – power-up subclasses.

include/Entities/Fish.h – base fish entity with AI and states.

include/Entities/GenericFish.h – simple fish used for schools.

include/Entities/Hazard.h – abstract hazard like bombs or jellyfish.

include/Entities/ICollidable.h – interface for collision handling.

include/Entities/IPowerUpManager.h – interface for power-up systems.

include/Entities/Player.h – player controlled fish with growth.

include/Entities/PlayerGrowth.h – tracks size progression.

include/Entities/PlayerInput.h – reads keyboard and mouse input.

include/Entities/PlayerStatus.h – handles damage and invulnerability.

include/Entities/PlayerVisual.h – manages player animations.

include/Entities/PoisonFish.h – enemy that poisons the player.

include/Entities/PowerUp.h – base class for power-ups.

include/Entities/Pufferfish.h – enemy fish that inflates defensively.

include/Entities/SchoolMember.h – small fish that form schools.

include/Managers/BonusItemManager.h – spawns starfish and power-ups.

include/Managers/EnhancedFishSpawner.h – spawns special enemy fish.

include/Managers/FishSpawner.h – generic fish spawning logic.

include/Managers/GenericSpawner.h – template for timed spawners.

include/Managers/OysterManager.h – manages pearl oysters.

include/Managers/PowerUpFactory.h – creates power-up instances.

include/Managers/SpriteManager.h – loads textures and provides sprites.

include/States/BonusStageState.h – time-limited bonus round.

include/States/EnvironmentController.h – controls currents and effects.

include/States/GameOptionsState.h – options menu and audio sliders.

include/States/GameOverState.h – final screen after losing all lives.

include/States/GameSystems.h – aggregates HUD and power-up systems.

include/States/HUDController.h – updates in-game HUD elements.

include/States/HighScoresState.h – shows saved high score table.

include/States/IntroState.h – brief logo splash screens.

include/States/MenuState.h – main menu with animated options.

include/States/PlayLogic.h – utility class for game logic steps.

include/States/PlayState.h – primary gameplay state.

include/States/PlayerNameState.h – collects the player's name.

include/States/SpawnController.h – configures spawn rates per level.

include/States/StageIntroState.h – display summary of upcoming stage.

include/States/StageSummaryState.h – show collected stats between stages.

include/Systems/CameraController.h – manages view scrolling and zoom.

include/Systems/CollisionDetector.h – helper for intersection tests.

include/Systems/CollisionSystem.h – orchestrates entity collisions.

include/Systems/EnvironmentSystem.h – simulates water currents.

include/Systems/FishCollisionHandler.h – double-dispatch visitor.

include/Systems/FishFactory.h – creates fish instances.

include/Systems/FrenzySystem.h – tracks score multiplier chain.

include/Systems/HUDSystem.h – draws score and growth meter.

include/Systems/IScoreSystem.h – score system interface.

include/Systems/InputHandler.h – translates input events for the player.

include/Systems/InputStrategy.h – normal or reversed controls.

include/Systems/ParticleSystem.h – renders particle effects.

include/Systems/SchoolingSystem.h – keeps fish groups aligned.

include/Systems/ScoreSystem.h – calculates and displays points.

include/Systems/SpawnSystem.h – spawns hazards and power-ups.

include/Systems/SpriteComponent.h – drawable sprite component.

include/Systems/Strategy.h – movement strategy classes.

include/UI/GrowthMeter.h – UI bar showing growth progress.

include/Utils/AnimatedSprite.h – sprite animation helper.

include/Utils/Animator.h – manages frame sequences.

include/Utils/DrawHelpers.h – drawing utilities for debug.

include/Utils/HighScoreIO.h – file I/O for high scores.

include/Utils/SpawnTimer.h – simple timer for spawn logic.

src/Core/Game.cpp – implements the main loop and state transitions.

src/Core/Main.cpp – application entry point.

src/Core/MusicPlayer.cpp – manages background music playback.

src/Core/SoundPlayer.cpp – plays sound effects.

src/Core/State.cpp – base state implementation.

src/Core/StateManager.cpp – stack logic for states.

src/Entities/AdvancedFish.cpp – behavior for advanced fish types.

src/Entities/Angelfish.cpp – angelfish enemy implementation.

src/Entities/Barracuda.cpp – fast hunting predator AI.

src/Entities/BonusItem.cpp – base logic for bonus objects.

src/Entities/Entity.cpp – common entity functionality.

src/Entities/ExtendedPowerUps.cpp – specific power-up effects.

src/Entities/Fish.cpp – core fish behavior and AI.

src/Entities/Hazard.cpp – base class for hazards.

src/Entities/Player.cpp – player actions and growth handling.

src/Entities/PlayerGrowth.cpp – updates growth meter state.

src/Entities/PlayerInput.cpp – processes keyboard input.

src/Entities/PlayerStatus.cpp – life and invulnerability logic.

src/Entities/PlayerVisual.cpp – handles player animations.

src/Entities/PoisonFish.cpp – enemy causing control reversal.

src/Entities/PowerUp.cpp – base power-up code.

src/Entities/Pufferfish.cpp – pufferfish enemy behaviour.

src/Managers/BonusItemManager.cpp – spawns bonuses over time.

src/Managers/ConfiguredFishFactory.cpp – creates fish formations.

src/Managers/EnhancedFishSpawner.cpp – spawns special fish types.

src/Managers/FishSpawner.cpp – spawns generic fish enemies.

src/Managers/OysterManager.cpp – handles pearl oyster hazards.

src/Managers/SpriteManager.cpp – texture loading and sprite config.

src/States/BonusStageState.cpp – bonus level logic.

src/States/EnvironmentController.cpp – applies environmental forces.

src/States/GameOptionsState.cpp – handles options menu events.

src/States/GameOverState.cpp – displays game over screen.

src/States/HUDController.cpp – updates HUD info.

src/States/HighScoresState.cpp – reads and displays high scores.

src/States/IntroState.cpp – shows intro images.

src/States/MenuState.cpp – main menu interactions.

src/States/PlayLogic.cpp – helper methods for play state.

src/States/PlayState.cpp – core gameplay update loop.

src/States/PlayerNameState.cpp – input screen for player name.

src/States/SpawnController.cpp – adjusts spawn rates each level.

src/States/StageIntroState.cpp – displays stage objectives.

src/States/StageSummaryState.cpp – summary after each stage.

src/Systems/CameraController.cpp – controls camera movement.

src/Systems/CollisionSystem.cpp – collision handling logic.

src/Systems/EnvironmentSystem.cpp – current and bubble effects.

src/Systems/FrenzySystem.cpp – manages frenzy multiplier.

src/Systems/HUDSystem.cpp – renders HUD elements.

src/Systems/InputHandler.cpp – passes events to the player.

src/Systems/InputStrategy.cpp – implements reversed controls.

src/Systems/ParticleSystem.cpp – visual particle system.

src/Systems/SchoolingSystem.cpp – maintains fish schools.

src/Systems/ScoreSystem.cpp – score calculations and text.

src/Systems/SpawnSystem.cpp – creates hazards and power-ups.

src/Systems/SpriteComponent.cpp – sprite draw component.

src/Systems/Strategy.cpp – AI movement strategies.

src/UI/GrowthMeter.cpp – draws growth meter UI.

src/Utils/AnimatedSprite.cpp – handles animation frames.

src/Utils/Animator.cpp – updates sprite animations.

resources/Fonts/Regular.ttf – font used for all text.

resources/Textures/\* – images for fish, backgrounds and UI.

resources/Sound/\* – music tracks and sound effects.