POLITÉCNICO DE LEIRIA

Advanced Game Programming Topics

2025/2026

2st Examination Period

Exam

By:

2231864 | Afonso Alexandre

February 08th, 2026

Abstract

This report presents the design and implementation of a custom 2D game engine written in C++, developed to support a Unity-like workflow based on Scenes, GameObjects, and component composition. The engine provides a deterministic lifecycle system (Awake/ Start /Update/ FixedUpdate/ LateUpdate), safe object creation and destruction through deferred adoption and queued teardown, and a practical runtime framework using per-scene GameModes and an application-lifetime GameInstance. Rendering is implemented using SDL3's 2D rendering capabilities, organized as a centralized pipeline (RenderSystem → RenderQueue → Renderer) with stable sorting and virtual-resolution viewport handling (letterbox/stretch/crop). Assets such as textures, sprite sheets, bitmap fonts, and audio clips are loaded and cached through an AssetManager to prevent redundant I/O. Physics is integrated via a Box2D wrapper that synchronizes rigid bodies with transforms and dispatches collision/trigger events to scripts. The engine also includes a navigable UI system with focus management for keyboard/gamepad, bitmap-text rendering, and basic widgets, as well as a sprite animation system driven by clips and controllers. Overall, the project prioritizes usability, modularity, and runtime efficiency for small-to-medium 2D games while remaining general-purpose and reusable across projects.

To Table of Contents

Objectives

Accomplished:

- SDL3-based engine (windowing, 2D rendering, input, audio)
- Box2D integrated for 2D physics (rigid bodies, colliders, triggers, collision/trigger callbacks)
- Engine follows an object-oriented and generic design (reusable outside this specific game)
- Engine and game exist as separate projects, with the game using the engine as an external library
- Game loop separates logic from rendering
- Xenon 2000 clone features implemented:
    a. Spaceship movement via keyboard/gamepad and turning animations
    b. Missiles fired via keyboard/gamepad buttons
    c. Three missile types (light/medium/heavy) with 1x/2x/4x power scaling through weapon upgrades
    d. Missile collision with enemies destroys missile and triggers an explosion
    e. Loner enemy: horizontal movement + fires toward the spaceship every 2 seconds
    f. Enemy projectiles explode when colliding with the spaceship
    g. Rusher enemy: vertical movement
    h. Drone enemy: sinusoidal horizontal movement and pack spawning behavior
    i. Stone asteroids: 3 sizes; big/medium split into 3 smaller; small explodes when hit
    j. Metal asteroids: 3 sizes; indestructible
    k. Companion system: up to two companions; mirror ship behavior; fire when ship fires; die when ship dies; can pick up power ups
    l. Shield power up restores life/energy to ship or companion depending on who collects it
    m. Weapon power up upgrades missile type for ship or companion depending on who collects it
    n. Horizontal scrolling gameplay with at least two background layers for parallax
- Extra credits: audio implemented using SDL3's newer audio system (audio device + streams) (throws exception if shutdown mid soundclip)

Not accomplished

- GLSL vertex and fragment shaders (rendering uses SDL 2D renderer instead of an OpenGL/GLSL pipeline)

To Table of Contents

Table of Contents

To Table of Contents

To Table of Contents

To Table of Contents

To Table of Contents

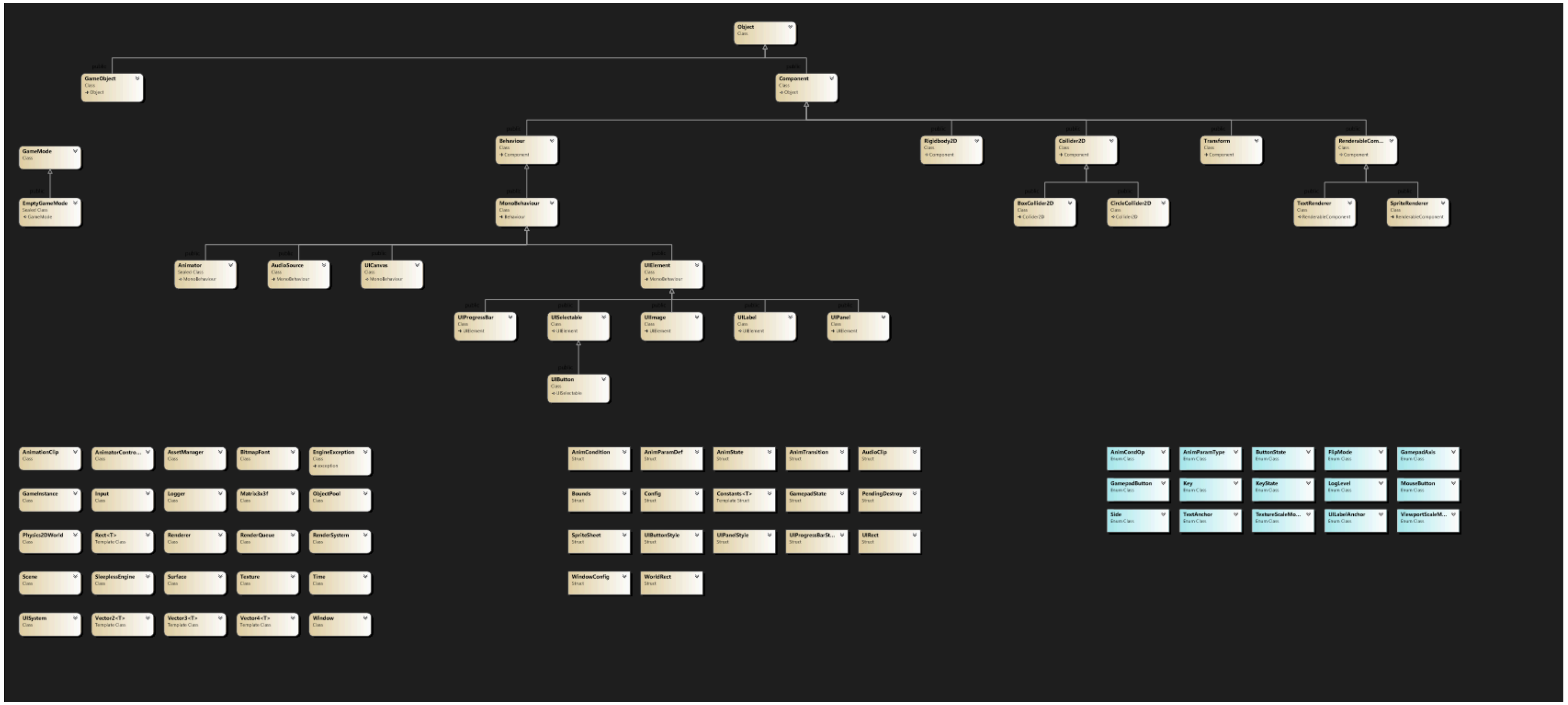
Sleepless Game Engine

Architecture

Overview

The engine is a modular C++ 2D runtime built around a Unity-like composition model. Scenes contain GameObjects, and GameObjects are extended through Components. Gameplay logic is implemented as MonoBehaviours (script-like components) driven by a deterministic lifecycle. Rendering, UI, physics, audio, animation, and assets are separate subsystems that integrate through shared services and lightweight registries. Rendering uses SDL’s 2D renderer (no OpenGL/GLSL pipeline).

To Table of Contents



To Table of Contents

Core object model

The engine follows the chain Scene → GameObject → Component.

A Scene owns and updates all GameObjects in that level and coordinates lifecycle processing.

A GameObject is an entity container that always has a Transform, stores a list of components, and controls activation through two states: activeSelf (its own flag) and activeInHierarchy (its own flag combined with its parents).

A Component is the base unit for all features (render, physics, audio, UI, scripts). Components can be queried by type on the same object, on parents, or on children via the Transform hierarchy.

A Behaviour (Component) adds enable/disable semantics and an “active and enabled” check that combines Behaviour enabled state with the GameObject active state.

A MonoBehaviour (Component) provides the gameplay callback interface: Awake, Start, Update, FixedUpdate, LateUpdate, OnEnable, OnDisable, and OnDestroy. It also supports collision and trigger callbacks, plus a timer/invoke system for delayed and repeating actions.

Transform hierarchy

Transform stores local position, rotation, and scale, and also stores parent/child links.

World transforms are derived from the hierarchy and cached using dirty flags, so world-space values are only recomputed when something changes.

Ownership and lifetime

There are two main lifetimes: engine lifetime and scene lifetime.

Engine lifetime (application-wide): SleeplessEngine owns the window, renderer, time and input services, asset manager, and the physics world. A single GameInstance is created at engine startup and persists until shutdown to store global/persistent data.

To Table of Contents

Scene lifetime (level-wide): The active Scene owns its GameObjects and their components. Each Scene owns exactly one GameMode, which represents the rules and state for that level.

Safe mutation of the object list strategy

The engine avoids mutating collections while iterating them by separating “requesting a change” from “applying a change”.

New GameObjects are created freely but are adopted into the Scene at safe and scheduled points (deferred adoption). Destruction is also deferred: objects are queued for destruction and processed after update phases. MonoBehaviour initialization is coordinated through a lifecycle queue so that Awake, OnEnable, and Start are run in a controlled and consistent order. This keeps iteration stable, reduces mid-frame pointer issues, and keeps lifecycle transitions consistent.

Why does lifecycle use deferred adoption instead of snapshotting the object list?
Lifecycle (Awake/OnEnable/Start) does not rely on rebuilding a per-frame snapshot of all objects/components. Snapshotting typically means keeping a second full list of pointers for iteration (or repeatedly building/resizing one), which increases memory usage and creates allocation churn proportional to the scene size. With deferred adoption, the Scene keeps one authoritative object list plus a much smaller “to be adopted” list. In other words, memory stays closer to “object list + pending adds” rather than “object list + a full duplicate list”. This reduces peak memory, reduces temporary allocations, and makes the cost proportional to what changed (usually rather diminished in comparison to the alternative) instead of proportional to the entire scene every time lifecycle work needs to be processed.

Frame pipeline (single-threaded)

The runtime is single-threaded and deterministic. Each frame follows this structure:

- First, input is polled (keyboard, mouse, gamepad).
- Then a fixed-step loop runs zero or more times depending on the accumulator (Scene.FixedUpdate, physics step, behaviour fixed callbacks for Monobehaviours).
- After that, the variable update runs (Scene.Update, behaviour update callbacks). In this step deferred adoption is processed both at start and at end)
- Then late update runs (Scene.LateUpdate, final per-frame adjustments).
- Next, pending destroys are processed.
- Finally, rendering happens (build render queue, draw world, draw UI, present).

To Table of Contents

Rendering architecture

Rendering is centralized and registry-based. The flow is RenderableComponents > RenderSystem > RenderQueue > Renderer.

SpriteRenderer and TextRenderer are RenderableComponents. RenderSystem maintains a registry of active renderables so the engine does not need to scan all objects each frame. Each frame, RenderQueue is built and sorted deterministically using stable keys (layer, per-component order, optional axis sorting, deterministic tie-breakers +Y and +X). Renderer wraps SDL_Renderer and handles virtual resolution (letterbox/stretch/crop), viewport caching, and world-to-screen mapping. This keeps rendering work proportional to the number of active renderables and produces consistent draw order. Its layering system could be improved to be customizable by the user. And possibly, by adding a camera system to the current viewport system.

Why does rendering use a registry instead of per-frame scanning/snapshotting?
A registry makes the render pipeline scale with the number of renderable components rather than the total number of objects and components. It also reduces per-frame allocations, keeps ordering deterministic through an explicit sorting stage, and avoids performance spikes in spawn-heavy gameplay because only the registry changes when renderables are created or destroyed. This way we don't have to iterate through a list of the objects, even the ones that don't override a "draw" function, causing unnecessary overhead.

Asset architecture

AssetManager is the central cache for textures, sprite sheets, bitmap fonts, and audio clips (And possibly more if this project is ever extended). Assets are loaded once, stored by key/path, and reused to avoid duplicate disk reads and duplicate GPU uploads. There could be some improvements in Quality of Life on the use of the Manager still.

Physics architecture (Box2D wrapper)

Physics is integrated through a Physics2DWorld service. Rigidbody2D owns a Box2D body and synchronizes it with Transform. Collider2D owns a Box2D shape/fixture and can behave as a trigger (sensor). The world steps at a fixed timestep and dispatches collision/trigger enter, stay, and exit events into MonoBehaviours, using cached contact pairs so "stay" can be generated reliably and without duplicated detection.

To Table of Contents

UI architecture

UI is a separate overlay system with focus navigation for keyboard/gamepad. It is structured as a UISystem plus a UICanvas and UIElement hierarchy. UICanvas groups UI elements and defines sorting order and visibility. UIElement provides rect layout, visibility/interactability, hit testing, and a render hook. UISelectable adds focus and navigation links. Widgets include UIButton, UILabel, UIImage, UIPanel, and UIProgressBar. UI runs after world rendering and is drawn in UI space derived from the renderer's viewport. I could be improved by adding its own render pass.

Animation architecture

Animation is currently sprite-driven. AnimationClip defines frames, FPS, looping, and optional events. AnimatorController defines states and parameter-driven transitions. Animator evaluates the controller each frame and updates the SpriteRenderer's frame index. This system was made with great inspiration from Unity's state machines.

Reflection

Currently the system does not support true reflection, thus being not as flexible in some parts as in others. For example, the animation state machine only is able to check conditions of certain native types. It's complicated to force flexible assertion, like if a user makes a component or game object that requires another, resulting in taxing writing of rules manually.

Time and timestep management

The engine uses a Unity-like time model. Each frame it measures real (machine) time to produce an unscaled delta time, then applies a timeScale to produce the gameplay delta time used by most Update() logic. Large frame deltas are clamped to avoid destabilizing gameplay and physics after stalls. For physics and other simulation work, the engine uses a fixed timestep with an accumulator, running FixedUpdate() (and stepping physics) zero or more times per frame depending on how much time has accumulated.

The engine does not smooth/interpolate the timestep for rendering, which could be a point to improve as it can lead to some hiccups if there's a single, heavy work frame. It renders using the current state after stepping.

To Table of Contents

Systems

Engine Core Loop

Implemented in files:

- SleeplessEngine.h
- SleeplessEngine.cpp

Config (startup configuration)

Purpose: A single struct that defines how the engine boots (timing, window settings, renderer scaling, asset root, and debug toggles). The idea is that the engine can be configured once, then every subsystem reads consistent settings.

Fields:

- Timing:
  - fixedDeltaTime — fixed-step duration used by physics + FixedUpdate() (default 1/60).
  - maximumDeltaTime — clamp for big frame stalls (prevents a single hitch from producing an absurd delta).
  - targetFPS — frame cap target used by the engine's pacing.

- Assets:
  - assetBasePath — base folder used by AssetManager when loading files.

- Window:
  - windowConfig — title/size/fullscreen/borderless/resizable (defaults are defined in WindowConfig).
- Renderer / viewport scaling:
  - virtualResolution — "virtual game size" used for consistent pixel coordinates; (0,0) disables it.
  - viewportScaleMode — how virtual resolution maps to the real window: Letterbox, Stretch, Crop.
  - integerScale — if true, scaling snaps to whole-number multiples.

To Table of Contents

    - fitWindowToScale — windowed-only option: when letterboxing, it resizes the window to match the virtual aspect ratio (reduces black bars by changing the window size instead).

    - clearColor — clear color inside the game viewport.
    - letterboxColor — color used for the bars outside the game viewport.

    - textureScaleMode — default texture filtering when scaled (Nearest/Linear).
    - spriteSortOptions — default sprite sort axes and direction (commonly Y primary, X secondary, ascending).

- Debug:
    - debugDrawColliders — draws collider outlines during render.

Sleepless Engine

Purpose: The top-level runtime object. It creates and owns the core subsystems, runs the game loop, owns the active Scene, and guarantees a consistent update + render order.

Main responsibilities:

- Initialize SDL + engine subsystems (Time, Window, Renderer, Assets, Input, Audio, Physics). Could probably remove SDL from here, but didn't have time to test if every SDL depended system was properly initialising SDL themselves.
- Runs the loop: time > input > fixed updates > update > late update > destroy > render > frame cap wait.
- Owns the application-lifetime GameInstance.
- Switches scenes safely (Unload() old, Start() new).

Variables:

- m_isInitialized, m_isRunning — state flags for boot + loop.
- m_config — the active Config used by subsystems.
- m_window — SDL window wrapper.
- m_renderer — SDL renderer wrapper + virtual resolution handling.

To Table of Contents

- m_assetManager — texture/font/audio caches using the configured base path.
- m_physicsWorld — Box2D wrapper and collision dispatch.
- m_currentScene — active scene (raw pointer; the game code owns the scene object lifetime).
- m_gameInstance + m_gameInstanceFactory — one persistent instance per engine run (created once, destroyed on shutdown).
- m_lastFitWindowSize — avoids repeatedly setting the same window size when fitWindowToScale is enabled.

Methods:

- Initialize(config)
    - Goal: create every system in a known order, apply config once, and leave the engine ready to run.

  High-level steps:
    1. Store m_config.
    2. Initialize Time and apply fixedDeltaTime, maximumDeltaTime, targetFPS.
    3. Initialize SDL (video + events).
    4. Initialize Audio (opens the default audio device / subsystem).
    5. Create the Window, then create the Renderer from that window.
    6. Apply renderer-related config: virtual resolution, scale mode, integer scaling, clear/letterbox colors.
    7. Apply sprite sorting defaults (used by both SpriteRenderer and RenderQueue sorting).
    8. Create AssetManager, apply assetBasePath and default texture scale mode.
    9. Initialize Input.
    10. Create and initialize the Physics world (initial gravity set here, later adjustable via reset).
    11. Create the GameInstance once (factory if provided; otherwise default), then call GameInstance::OnInit().
    12. Mark initialized.
- Run() — main loop
    - Goal: deterministic, single-threaded loop with fixed-step simulation and variable-step gameplay/render.

  High Level Steps - Per-frame order:
    1. tick time, updating delta time and others.
    2. Poll inputs, set input state to later query.
    3. Apply fit-to-scale window adjustment.
    4. If a quit is requested then shutdown.
    5. If debug hotkey: F9 toggles FPS display.

To Table of Contents

6. Fixed-step simulation:
    a. Calculates the amount of steps to catch up
    b. Repeat steps times:
        i. Scene fixed update, scene tells behaviours to fixed update
        ii. Physics step
        iii. Consumes step
7. Variable update:
    a. Scene update
        i. Scene processes pending lifecycles from fixed update
        ii. Scene tells behaviours to update
        iii. Scene processes pending lifecycles from update
        iv. UI update runs after gameplay update (so UI consumes the final input state consistently)
8. Late update:
    a. Scene late update, scene tells behaviours to late update
9. Garbage collection, objects queued for destruction this frame are truly destroyed.
10. Render:
    a. Rendering passes
        i. World
        ii. Debug
        iii. UI
11. Wait for frame time to reach target if target fps is set

- Render()
    - Goal: produce a stable draw order and always draw UI last.

  Steps:
    1. Clear (handles both letterbox area + game viewport clear).
    2. If scene active:
        a. Build a RenderQueue from RenderSystem (registry of renderables).
        b. Execute the queue (sorted draw calls).
        c. Scene tells behaviours to use OnRender
        d. Optional debug pass: collider outlines if enabled.
        e. UI pass last (overlay).
        f. Present frame.
- Shutdown()
    - Goal: tear down cleanly, even if shutdown is requested from gameplay.

To Table of Contents

Steps:

- If shutdown is called while the loop is still running, it does not tear everything down immediately; it just requests quit, and teardown happens after the loop exits.
- Actual teardown order (once not running):
    1. Unload current scene.
    2. GameInstance OnShutdown() then delete it.
    3. Shutdown UI first (UI may rely on input state).
    4. Shutdown Input
    5. Audio (currently gives exception if a audio is playing at the moment of shutdown)
    6. Shutdown Physics world.
    7. Destroy AssetManager
    8. Renderer
    9. Window.
    10. SDL_Quit()
    11. Mark not initialized.

- ApplyFitWindowToScale() (optional windowed QoL feature)
    - Goal: when using Letterbox + virtual resolution, “snap” the window size to the correct aspect ratio to reduce bars.

Steps:

- Only runs if: fitWindowToScale == true, windowed mode, virtual resolution enabled, and scale mode is Letterbox.

1. Compare current window aspect vs target virtual aspect.
2. If the window is too wide: reduce width to match aspect; if too tall: reduce height.
3. If integerScale enabled: snap to the largest integer multiple of the virtual resolution that fits.
4. Avoid calling SetSize() every frame by caching the last applied size and ignoring tiny 1px differences.

Time and timestep management

Implemented in files:

- GameEngine/Time.hpp

Purpose: Provides the engine’s timing state for every frame. It measures real time, produces per-frame delta values, supports time scaling (pause/slow/fast/reverse), schedules fixed-step simulation through an accumulator, tracks FPS, and can optionally enforce a target FPS through frame pacing.

To Table of Contents

Type aliases:

- using Clock = std::chrono::steady_clock - monotonic clock (not affected by OS clock changes)
- using TimePoint = Clock::time_point - timestamp type produced by the steady clock

Variables:

- Timestamps
    - TimePoint m_startTime - timestamp when timing was initialized
    - TimePoint m_lastTime - timestamp from the previous frame (used to compute real delta)
    - TimePoint m_now - current frame timestamp snapshot
    - TimePoint m_frameStartTime - start-of-frame timestamp used for frame pacing

- Per-frame deltas
    - float m_unscaledDeltaTime = 0.0f - real seconds between frames (clamped)
    - float m_deltaTime = 0.0f - scaled seconds between frames (unscaledDeltaTime timeScale)

- Fixed step configuration and limits
    - float m_fixedDeltaTime = 1.0f / 60.0f - size of one fixed step in seconds
    - float m_maxDeltaTime = 0.25f - clamp for unscaled delta to avoid huge jumps after stalls

- Accumulated and elapsed time counters
    - float m_unscaledElapsedTime = 0.0f - total real time since start (always increases)
    - float m_elapsedTime = 0.0f - total scaled time since start (can go backwards if timeScale < 0)
    - float m_elapsedFixedTime = 0.0f - total time advanced through fixed steps
    - float m_accumulator = 0.0f - scaled time waiting to be converted into fixed steps

- Time control
    - float m_timeScale = 1.0f - effective time scale used to compute scaled delta time
    - bool m_paused = false - pause state tracked separately from timeScale
    - float m_savedTimeScale = 1.0f - stored timeScale restored after

To Table of Contents

unpausing

- Frame pacing
    - float m_targetFPS = 60.0f - desired FPS cap (<= 0 disables pacing)
    - float m_targetFrameTime = 1.0f / 60.0f - target seconds per frame derived from targetFPS

- FPS tracking
    - float m_fps = 0.0f - computed FPS value (updated roughly once per second)
    - float m_fpsTimer = 0.0f - accumulated unscaled time for FPS reporting window
    - uint32_t m_frameCount = 0 - number of frames counted during the FPS window
    - bool m_showFPS = false - when true, logs FPS information each report interval

Methods:

- Singleton access
    - static Time& Instance()
        - Return
            - Time& - reference to the global Time singleton that stores all timing state

- Initialization and per-frame tick
    - static void Initialize()
        - Return
            - void - resets timestamps, counters, scaling and FPS tracking to defaults
        - Steps
            - Captures a starting timestamp and initializes last/now/frameStart to the same value
            - Clears deltas, elapsed counters, fixed accumulator, and FPS tracking state
            - Sets timeScale to 1 and sets paused state to false

    - static void Tick()
        - Return
            - void - updates delta values, elapsed timers, fixed-step accumulator, and FPS tracking for the current frame
        - Steps
            - Captures the frame start timestamp and computes

To Table of Contents

            unscaledDeltaTime from the previous frame
            - Clamps unscaledDeltaTime to maxDeltaTime to avoid large time jumps after stalls
            - Computes scaled deltaTime using timeScale
            - Advances unscaledElapsedTime by unscaledDeltaTime
            - Advances elapsedTime by deltaTime (can decrease when timeScale < 0)
            - Only advances the fixed-step accumulator when deltaTime is positive
            - Updates FPS counters using unscaled time and refreshes the FPS value about once per second

- Fixed timestep helpers
    - static int CalculateFixedSteps()
        - Return
            - int - number of fixed steps currently available (accumulator / fixedDeltaTime)
        - Steps
            - Computes how many full fixedDeltaTime slices fit in the accumulator

    - static void ConsumeFixedStep()
        - Return
            - void - consumes exactly one fixed step from the accumulator and advances elapsedFixedTime
        - Steps
            - Subtracts fixedDeltaTime from the accumulator
            - Adds fixedDeltaTime to elapsedFixedTime

- Time getters
    - static float Now()
        - Return
            - float - current scaled elapsed time (affected by timeScale; can go backwards)
    - static float UnscaledNow()
        - Return
            - float - current unscaled elapsed time (real time; always increases)
    - static float DeltaTime()
        - Return
            - float - scaled delta time for this frame
    - static float UnscaledDeltaTime()
        - Return
            - float - unscaled (real) delta time for this frame (clamped)

[To Table of Contents]

- static float FixedDeltaTime()
  - Return
    - float - fixed step size in seconds
- static float ElapsedTime()
  - Return
    - float - total scaled elapsed time (same value as Now)
- static float UnscaledElapsedTime()
  - Return
    - float - total unscaled elapsed time (same value as UnscaledNow)
- static float ElapsedFixedTime()
  - Return
    - float - total time advanced via fixed steps
- static float Accumulator()
  - Return
    - float - current accumulated scaled time waiting to be simulated with fixed steps
- static float FPS()
  - Return
    - float - last computed frames-per-second value
- static float TargetFPS()
  - Return
    - float - configured FPS cap value
- static float TargetFrameTime()
  - Return
    - float - target seconds per frame derived from TargetFPS

- Time scale and pause control
  - static void SetTimeScale(float scale)
    - Parameters
      - float scale - new time scale to apply when not paused, or to store for later when paused
    - Return
      - void - updates time scale behavior without forcing pause state changes
    - Steps
      - If paused, stores the value into savedTimeScale while keeping effective timeScale at 0
      - If not paused, applies scale immediately to timeScale

  - static float GetTimeScale()
    - Return
      - float - current effective timeScale (0 while paused)

To Table of Contents

        - static float GetUnpausedTimeScale()
            - Return
                - float - the timeScale that will be restored after unpausing (or the current timeScale if not paused)

        - static bool IsPaused()
            - Return
                - bool - true if pause mode is active

        - static void SetPaused(bool paused)
            - Parameters
                - bool paused - whether pause mode should be enabled
            - Return
                - void - toggles pause state while preserving the previous non-paused timeScale
            - Steps
                - When pausing, saves the current timeScale and forces the effective timeScale to 0
                - When unpausing, restores the saved timeScale and clears the paused flag

        - static void Pause()
            - Return
                - void - convenience wrapper that pauses the time system

        - static void Resume()
            - Return
                - void - convenience wrapper that resumes the time system to the previously saved timeScale

- Configuration setters
    - static void SetFixedDeltaTime(float dt)
        - Parameters
            - float dt - new fixed step size in seconds
        - Return
            - void - updates the fixed timestep size used by the accumulator

    - static void SetMaxDeltaTime(float dt)
        - Parameters
            - float dt - new clamp value for unscaledDeltaTime
        - Return
            - void - limits the largest real delta that can be applied in a single Tick

To Table of Contents

        - static void SetTargetFPS(float fps)
            - Parameters
                - float fps - desired FPS cap value (<= 0 disables frame pacing)
            - Return
                - void - updates pacing configuration and recomputes targetFrameTime

- Frame pacing and debug
    - static void WaitForTargetFPS()
        - Return
            - void - delays the thread so the frame duration approaches the configured target frame time
        - Steps
            - Computes when the current frame should end based on frameStartTime + targetFrameTime
            - Sleeps most of the remaining time (with a small guard)
            - Yields in a short loop until the target end time is reached

    - static void ToggleShowFPS()
        - Return
            - void - toggles periodic FPS logging on or off

Notes:
- Fixed-step accumulation only advances when scaled deltaTime is positive. This means fixed-step simulation (and physics stepping) does not run while timeScale is negative.
- A potential improvement is allowing physics to run under negative timeScale, which would require a dedicated reverse-time strategy rather than simply stepping the physics engine backwards.

Window and windowing

Implemented in files:
- GameEngine/Window.h
- GameEngine/Window.cpp

Purpose: Provides a small engine-facing wrapper around the SDL window. It centralizes window creation options (title, size, fullscreen, borderless, resizable), exposes runtime operations (resize, show/hide, focus), and supports mouse capture modes (grab and relative). SDL types are hidden behind a private implementation struct.

Variables:

- WindowConfig (startup configuration)
  - std::string title = "Game Window" - operating system window title
  - Vector2i windowSize = Vector2i(800, 600) - initial window size in pixels
  - bool fullscreen = false - starts in fullscreen mode when true
  - bool borderless = false - uses a borderless window style when true
  - bool resizable = true - allows user resizing when true

- Window (public wrapper state)
  - std::unique_ptr<Impl> impl - pointer to internal implementation that hides SDL headers/types

- Window::Impl (hidden internal state)
  - SDL_Window window = nullptr - native SDL window handle owned by this wrapper
  - WindowConfig config - cached configuration values used by getters

Methods:

- Construction and lifetime
  - Window(const WindowConfig& config = {})
    - Parameters
      - const WindowConfig& config - startup configuration (title, size, flags)
    - Return
      - none - constructs the wrapper and creates the underlying SDL window
    - Steps
      - Initializes SDL video and event systems
      - Stores the provided configuration into the internal cache
      - Builds SDL window flags from fullscreen, borderless, and resizable
      - Creates the SDL_Window using the configured title and size
      - Throws an engine exception if SDL initialization or window creation fails

  - ~Window()
    - Return

To Table of Contents

            - none - destroys the wrapper and releases the SDL window handle via RAII

- Move semantics (copy disabled)
    - Window(Window&& other) noexcept
        - Parameters
            - Window&& other - source window wrapper whose internal handle is moved
        - Return
            - none - transfers ownership of the internal implementation to the new object
        - Steps
            - Moves the internal implementation pointer from other into this object

    - Window& operator=(Window&& other) noexcept
        - Parameters
            - Window&& other - source window wrapper whose internal handle is moved
        - Return
            - Window& - reference to this window after taking ownership
        - Steps
            - Replaces this object's implementation with the other implementation (move assignment)

- Getters
    - Vector2i GetSize() const
        - Return
            - Vector2i - current window size in pixels
        - Steps
            - Queries SDL for the current window size and returns it

    - std::string GetTitle() const
        - Return
            - std::string - cached window title string

    - bool IsFullscreen() const
        - Return
            - bool - cached fullscreen flag

    - bool IsBorderless() const
        - Return
            - bool - cached borderless flag

To Table of Contents

        - bool IsResizable() const
            - Return
                - bool - cached resizable flag

        - bool IsVisible() const
            - Return
                - bool - true if the window is not hidden
            - Steps
                - Reads SDL window flags and checks whether the hidden flag is set

- Setters
    - void SetTitle(const std::string& title)
        - Parameters
            - const std::string& title - new title to apply
        - Return
            - void - updates the cached title and applies it to the SDL window
        - Steps
            - Stores title in the cache
            - Sends the title to SDL

    - void SetSize(const Vector2i& windowSize)
        - Parameters
            - const Vector2i& windowSize - new window size in pixels
        - Return
            - void - updates the cached size and resizes the SDL window
        - Steps
            - Stores windowSize in the cache
            - Requests SDL to resize the window

    - void SetFullscreen(bool fullscreen)
        - Parameters
            - bool fullscreen - new fullscreen state
        - Return
            - void - updates cached state and toggles fullscreen in SDL
        - Steps
            - Stores fullscreen in the cache
            - Requests SDL to change fullscreen mode

    - void SetVisible(bool visible)
        - Parameters

To Table of Contents

            - bool visible - whether the window should be shown
        - Return
            - void - shows or hides the window
        - Steps
            - Shows the window when visible is true, otherwise hides it

- Operations
    - void Minimize()
        - Return
            - void - requests OS minimize through SDL
    - void Maximize()
        - Return
            - void - requests OS maximize through SDL
    - void Restore()
        - Return
            - void - restores from minimized/maximized state through SDL
    - void Show()
        - Return
            - void - convenience wrapper that shows the window
    - void Hide()
        - Return
            - void - convenience wrapper that hides the window
    - void Focus()
        - Return
            - void - requests focus/raise for the window through SDL

- Mouse input modes
    - void SetMouseGrab(bool grab)
        - Parameters
            - bool grab - whether the mouse cursor should be confined to the window
        - Return
            - void - enables or disables window mouse grab

    - void SetMouseRelativeMode(bool relative)
        - Parameters
            - bool relative - whether relative mouse mode should be enabled
        - Return
            - void - enables or disables relative mouse mode for capturing deltas

- Native handle access

To Table of Contents

- void GetNative() const
  - Return
    - void - SDL_Window pointer returned as void to avoid exposing SDL headers

Input and event processing

Implemented in files:

- GameEngine/Input.h
- GameEngine/Input.cpp

Purpose: Collects keyboard, mouse, and gamepad input once per frame from SDL's event queue, stores "previous" and "current" device states, and exposes higher-level queries like Down, Pressed, and Released. It also tracks mouse position, per-frame mouse delta, scroll deltas, and a quit-request flag.

Constants:

- constexpr static size_t MaxGamepads = 2 - maximum number of gamepads the engine tracks at once (each connected pad is assigned to the first free slot)
- constexpr static float GamepadDeadzone = 0.15f - stick deadzone threshold; stick axes inside this magnitude return 0 to avoid drift

Types:

- enum class Key - engine-level keyboard key identifiers (based on SDL scancodes, so it maps to physical keys rather than layout-dependent characters)
- enum class MouseButton - engine-level mouse buttons (Left, Right, Middle, X1, X2)
- enum class KeyState - keyboard state query result (Up, Down, Pressed, Released)
- enum class ButtonState - mouse button state query result (Up, Down, Pressed, Released)
- enum class GamepadAxis - normalized gamepad axes (LeftX, LeftY, RightX, RightY, LeftTrigger, RightTrigger)

To Table of Contents

- enum class GamepadButton - gamepad buttons (South/East/West/North, Start/Back, shoulders, sticks, d-pad)
- struct GamepadState - lightweight snapshot returned by GetGamepads()
    - int id - SDL device id
    - bool connected - connection flag
    - float axes[GamepadAxis::Max] - current normalized axes
    - bool buttons[GamepadButton::Max] - current button states

Variables:

- Implementation storage (internal, Input.cpp)
    - std::unique_ptr<Input::Impl> Input::impl - private state holder used so the header does not expose SDL types

- Keyboard state (internal, Input::Impl)
    - std::array<bool, size_t(Key::Max)> prevKeyState - key state from previous PollEvents call
    - std::array<bool, size_t(Key::Max)> currKeyState - key state updated by SDL key events in the current PollEvents call

- Mouse state (internal, Input::Impl)
    - std::array<bool, size_t(MouseButton::Max)> prevMouse - mouse button state from previous PollEvents call
    - std::array<bool, size_t(MouseButton::Max)> currMouse - mouse button state updated by SDL mouse button events
    - float mouseX, mouseY - last known cursor position in window pixel coordinates
    - float lastX, lastY - previous cursor position used to compute per-frame delta
    - float scrollX, scrollY - accumulated scroll wheel movement since the last GetScroll call

- Gamepad state (internal, Input::Impl::Pad)
    - int id = -1 - SDL id; -1 means the slot is unused
    - SDL_Gamepad *sdlPad - opened SDL gamepad handle (owned by this slot)*
    - bool connected - whether this slot currently represents an opened device
    - std::array<float, size_t(GamepadAxis::Max)> axes - current normalized axis values
    - std::array<float, size_t(GamepadAxis::Max)> prevAxes - previous axis values (for “pressed/released” style comparisons if needed later)
    - std::array<bool, size_t(GamepadButton::Max)> buttons - current button

To Table of Contents

state
    - std::array<bool, size_t(GamepadButton::Max)> prevButtons - previous button state (used for Pressed/Released)

- App quit flag (internal, Input::Impl)
    - bool quitRequested - set when SDL emits a quit event; read by ShouldQuit()

Methods:

- Initialization and shutdown
    - static void Initialize()
        - Return
            - void - allocates the internal implementation and initializes SDL's gamepad subsystem
        - Steps
            - Creates the internal Impl instance
            - Initializes the SDL gamepad subsystem (throws an engine exception if it fails)

    - static void Shutdown()
        - Return
            - void - closes opened gamepads, releases internal state, and shuts down the SDL gamepad subsystem
        - Steps
            - Closes any SDL_Gamepad handles stored in the tracked slots
            - Destroys the Impl instance
            - Quits the SDL gamepad subsystem

- Per-frame polling
    - static void PollEvents()
        - Return
            - void - consumes SDL events and updates current input state for this frame
        - Steps
            - Copies current keyboard/mouse state into the previous state arrays
            - For each connected gamepad slot, copies current axes/buttons into previous axes/buttons
            - Polls SDL events in a loop and updates only the affected pieces of state:

To Table of Contents

            - Quit events set quitRequested
            - Key down/up events update currKeyState for the translated key
            - Mouse button down/up events update currMouse for the translated mouse button
            - Mouse motion updates mouse position and stores the previous position (for delta)
            - Mouse wheel events accumulate scrollX/scrollY
            - Gamepad add/remove events open/close devices and claim/free slots
            - Gamepad button events update the button array for the matching device
            - Gamepad axis events normalize the raw value and store it into the axis array

- Keyboard queries
    - static KeyState GetKey(Key key)
        - Parameters
            - Key key - key to query
        - Return
            - KeyState - Up, Down, Pressed, or Released based on previous versus current state
        - Steps
            - Reads prev = prevKeyState[key] and cur = currKeyState[key]
            - Returns Pressed when cur is true and prev is false
            - Returns Released when cur is false and prev is true
            - Otherwise returns Down if cur is true, else Up

    - static bool IsKeyDown(Key key)
        - Parameters
            - Key key - key to query
        - Return
            - bool - true when GetKey(key) is Down

    - static bool IsKeyPressed(Key key)
        - Parameters
            - Key key - key to query
        - Return
            - bool - true when GetKey(key) is Pressed

    - static bool IsKeyReleased(Key key)
        - Parameters
            - Key key - key to query

To Table of Contents

        - Return
            - bool - true when GetKey(key) is Released

- Mouse button and pointer queries
    - static ButtonState GetMouseButton(MouseButton btn)
        - Parameters
            - MouseButton btn - mouse button to query
        - Return
            - ButtonState - Up, Down, Pressed, or Released based on previous versus current state
        - Steps
            - Reads prev = prevMouse[btn] and cur = currMouse[btn]
            - Returns Pressed when cur is true and prev is false
            - Returns Released when cur is false and prev is true
            - Otherwise returns Down if cur is true, else Up

    - static bool IsMouseButtonDown(MouseButton btn)
        - Parameters
            - MouseButton btn - mouse button to query
        - Return
            - bool - true when GetMouseButton(btn) is Down

    - static bool IsMouseButtonPressed(MouseButton btn)
        - Parameters
            - MouseButton btn - mouse button to query
        - Return
            - bool - true when GetMouseButton(btn) is Pressed

    - static bool IsMouseButtonReleased(MouseButton btn)
        - Parameters
            - MouseButton btn - mouse button to query
        - Return
            - bool - true when GetMouseButton(btn) is Released

    - static float GetMouseX()
        - Return
            - float - current cursor X position in window pixel coordinates

    - static float GetMouseY()
        - Return
            - float - current cursor Y position in window pixel coordinates

To Table of Contents

- static Vector2f GetMousePosition()
    - Return
        - Vector2f - convenience wrapper around GetMouseX and GetMouseY

- static float GetMouseDeltaX()
    - Return
        - float - per-frame cursor delta X (mouseX - lastX)

- static float GetMouseDeltaY()
    - Return
        - float - per-frame cursor delta Y (mouseY - lastY)

- static Vector2f GetMouseDelta()
    - Return
        - Vector2f - convenience wrapper around GetMouseDeltaX and GetMouseDeltaY

- static float GetScrollX()
    - Return
        - float - accumulated scroll X since the last call, then resets the stored value to 0

- static float GetScrollY()
    - Return
        - float - accumulated scroll Y since the last call, then resets the stored value to 0

- static Vector2f GetScroll()
    - Return
        - Vector2f - convenience wrapper around GetScrollX and GetScrollY

- Gamepad queries
    - static const std::vector<GamepadState>& GetGamepads()
        - Return
            - const std::vector<GamepadState>& - list of connected gamepads as lightweight snapshots (rebuilt each call)
        - Steps
            - Clears a static vector and reserves space for MaxGamepads
            - For each connected slot:
                a. copies id, axes, and buttons into a GamepadState
                b. pushes it into the list

To Table of Contents

            - Returns the list reference

- static int GetGamepadCount()
    - Return
        - int - number of connected gamepads in the tracked slots

- static bool IsGamepadConnected(int index = 0)
    - Parameters
        - int index - slot index in [0 .. MaxGamepads-1]
    - Return
        - bool - true if that slot currently holds a connected gamepad

- static bool IsGamepadButtonDown(GamepadButton btn, int index = 0)
    - Parameters
        - GamepadButton btn - button to query
        - int index - slot index
    - Return
        - bool - true while the button is held down on that gamepad slot

- static bool IsGamepadButtonPressed(GamepadButton btn, int index = 0)
    - Parameters
        - GamepadButton btn - button to query
        - int index - slot index
    - Return
        - bool - true only on the transition from up to down (this frame down, last frame up)

- static bool IsGamepadButtonReleased(GamepadButton btn, int index = 0)
    - Parameters
        - GamepadButton btn - button to query
        - int index - slot index
    - Return
        - bool - true only on the transition from down to up (this frame up, last frame down)

- static float GetGamepadAxis(GamepadAxis axis, int index = 0)
    - Parameters
        - GamepadAxis axis - axis to query
        - int index - slot index
    - Return

To Table of Contents

                - float - normalized axis value (sticks are -1..1, triggers are 0..1); stick axes return 0 inside the deadzone
            - Steps
                - Reads the stored axis value for the slot
                - For stick axes (LeftX, LeftY, RightX, RightY), returns 0 when abs(value) < GamepadDeadzone
                - Otherwise returns the value as stored

    - static Vector2f GetGamepadLeftStick(int index = 0)
        - Parameters
            - int index - slot index
        - Return
            - Vector2f - pair of (LeftX, LeftY) after deadzone filtering

    - static Vector2f GetGamepadRightStick(int index = 0)
        - Parameters
            - int index - slot index
        - Return
            - Vector2f - pair of (RightX, RightY) after deadzone filtering

- Quit request
    - static bool ShouldQuit()
        - Return
            - bool - true after SDL emits a quit event, allowing the main loop to exit cleanly

Notes:

- SDL scancodes are translated into the engine Key enum. This keeps the public API SDL-free and makes key queries consistent.
- Gamepad axis values are normalized from SDL's signed 16-bit range to floats:
    - Sticks remain in approximately -1..1
    - Triggers are remapped to 0..1 so they can be treated like "amount pressed"
- Currently doesn't properly separate input between multiple "players". Thus could be improved so that users can make player 1 and 2 and further extend their games.

To Table of Contents

World model and lifecycle

- Implemented in files:
  - GameEngine/Object.h
  - GameEngine/Object.cpp
  - GameEngine/Component.h
  - GameEngine/Component.cpp
  - GameEngine/Behaviour.h
  - GameEngine/Behaviour.cpp
  - GameEngine/MonoBehaviour.h
  - GameEngine/MonoBehaviour.cpp
  - GameEngine/Transform.h
  - GameEngine/Transform.cpp
  - GameEngine/GameObject.h
  - GameEngine/GameObject.cpp
  - GameEngine/GameObject.inl
  - GameEngine/Scene.h
  - GameEngine/Scene.cpp
  - GameEngine/ObjectPool.h
  - GameEngine/ObjectPool.cpp
  - GameEngine/GameMode.h
  - GameEngine/GameInstance.h

- Purpose:
  - Defines the runtime object graph (Scene → GameObject → Component) and the deterministic lifecycle used to safely create, enable/disable, update, and destroy gameplay objects.

Object:

- Files:
  - GameEngine/Object.h
  - GameEngine/Object.cpp

- Summary:
  - Root base type for engine objects. Assigns a unique instance ID, participates in a global registry (for lookups), and supports deferred destruction through a global destroy queue.

To Table of Contents

Variables:

- int m_instanceID - unique instance ID for this object (assigned at construction).
- std::string m_name - human-readable name for debugging and lookups.
- bool m_destroyed - set once the object has been destroyed and should no longer be used.
- bool m_markedForDestruction - set when destruction has been requested but not yet processed.

- static int s_nextID - global counter used to generate unique IDs.
- static std::vector<std::shared_ptr<Object>> s_objects - global owning registry of all live Objects.
- static std::vector<std::shared_ptr<Object>> s_destroyQueue - global owning queue of objects pending destruction.

Methods:

- Object(const std::string& name = "Object")
  - Parameters:
    - const std::string& name - initial object name (debug only).
  - Return:
    - void - constructs the object and assigns a unique ID.
  - Description:
    - Increments the global ID counter and stores name/state.

- virtual ~Object()
  - Parameters:
    - None
  - Return:
    - void - virtual destructor for safe polymorphic deletion.

- int GetInstanceID() const
  - Parameters:
    - None
  - Return:
    - int - unique instance ID of this object.

- virtual const std::string& GetName() const
  - Parameters:
    - None
  - Return:

To Table of Contents

        - const std::string& - current debug name for this object.

- virtual void SetName(const std::string& name)
    - Parameters:
        - const std::string& name - new debug name.
    - Return:
        - void - updates the stored name.

- bool IsDestroyed() const
    - Parameters:
        - None
    - Return:
        - bool - true if the object has been destroyed.

- bool IsMarkedForDestruction() const
    - Parameters:
        - None
    - Return:
        - bool - true if the object is queued for destruction.

- static void RegisterObject(const std::shared_ptr<Object>& obj)
    - Parameters:
        - const std::shared_ptr<Object>& obj - object to add to the global registry.
    - Return:
        - void - stores the object in the global owning list.

- static void UnregisterObject(const int instanceID)
    - Parameters:
        - const int instanceID - ID to remove from the registry.
    - Return:
        - void - removes registry entries that match the ID.

- static std::shared_ptr<Object> FindObjectByID(const int instanceID)
    - Parameters:
        - const int instanceID - ID to search for.
    - Return:
        - std::shared_ptr<Object> - owning reference if found, otherwise null.

- template<typename T> static std::vector<std::shared_ptr<T>> FindObjectsByType(bool includeInactive = false)
    - Parameters:
        - bool includeInactive - if true, includes objects that are inactive in

To Table of Contents

        hierarchy; if false, filters out inactive GameObjects.
    - Return:
        - std::vector<std::shared_ptr<T>> - all matching objects in the registry.
    - Description:
        - Iterates the global registry, filters by type using dynamic casting, and (optionally) filters out inactive GameObjects.

- static void Destroy(const std::shared_ptr<Object>& obj)
    - Parameters:
        - const std::shared_ptr<Object>& obj - object to destroy later.
    - Return:
        - void - marks and queues the object for destruction.
    - Description:
        - Sets the marked flag once and pushes into the destroy queue to avoid mid-frame invalid pointers.

- static void ProcessDestroyQueue()
    - Parameters:
        - None
    - Return:
        - void - finalizes destruction for queued objects.
    - Description:
        - Calls each object's internal destruction hook, marks it destroyed, unregisters it, and clears the queue. This is called at safe points after update phases.

- template<typename T> static std::shared_ptr<T> Instantiate(const std::shared_ptr<T>& original)
    - Parameters:
        - const std::shared_ptr<T>& original - source object to clone.
    - Return:
        - std::shared_ptr<T> - cloned object registered in the global registry.
    - Description:
        - Calls original->Clone(), registers the returned clone, and returns it.

- template<typename T> static std::shared_ptr<T> GetShared(T *raw)*
    - Parameters:
        - T *raw - non-owning pointer to search for in the registry.*
    - Return:
        - std::shared_ptr<T> - owning shared pointer if present in the registry, otherwise null.

To Table of Contents

    - Description:
        - Used to recover ownership when only a raw pointer is available.

- protected virtual void DestroyImmediateInternal()
    - Parameters:
        - None
    - Return:
        - void - override hook for derived classes to release resources immediately.
    - Description:
        - Called by ProcessDestroyQueue() during teardown.

GameObject:

- Files:
    - GameEngine/GameObject.h
    - GameEngine/GameObject.cpp
    - GameEngine/GameObject.inl

- Summary:
    - Entity container that always owns a Transform and a list of Components. Tracks activation (activeSelf and activeInHierarchy), exposes component queries, and integrates with Scene lifecycle and subsystems (rendering, physics, scripting).

Variables:

- bool m_activeSelf - local activation flag set directly by SetActive().
- bool m_activeInHierarchy - computed activation flag that also accounts for parent activation.
- int m_layer - layer value used by systems like rendering (and any user-side filtering).
- Scene *m_scene - non-owning pointer to the owning Scene (null if not adopted yet).*

- std::shared_ptr<Transform> m_transform - always-present Transform component.
- std::vector<std::shared_ptr<Component>> m_components - owned components (Transform is stored separately but also behaves like a component).

To Table of Contents

Methods:

- explicit GameObject(const std::string& name = "GameObject")
    - Parameters:
        - const std::string& name - object debug name.
    - Return:
        - void - constructs a GameObject and creates its Transform.

- ~GameObject() override
    - Parameters:
        - None
    - Return:
        - void - destroys owned components through Object destruction flow.

- bool IsActiveSelf() const
    - Parameters:
        - None
    - Return:
        - bool - true if the object’s local active flag is set.

- bool IsActiveInHierarchy() const
    - Parameters:
        - None
    - Return:
        - bool - true if the object is active and all parents are active.

- int GetLayer() const
    - Parameters:
        - None
    - Return:
        - int - current layer value.

- void SetLayer(int layer)
    - Parameters:
        - int layer - new layer value.
    - Return:
        - void - updates the stored layer.

- Scene *GetScene() const*
    - Parameters:

To Table of Contents

    - None
  - Return:
    - Scene - *owning Scene pointer (may be null before adoption).*

- Transform *GetTransform() const*
  - Parameters:
    - None
  - Return:
    - Transform - *non-owning pointer to the Transform component.*

- void SetActive(bool active)
  - Parameters:
    - bool active - new local activation state.
  - Return:
    - void - updates activation and propagates changes through the hierarchy.
  - Description:
    - Updates activeSelf, recomputes activeInHierarchy, then triggers behaviour enable/disable transitions as needed.

- template<typename T, typename... Args> std::shared_ptr<T> AddComponent(Args&&... args)
  - Parameters:
    - Args&&... args - forwarded constructor arguments for component T.
  - Return:
    - std::shared_ptr<T> - owning reference to the newly created component.
  - Description:
    - Constructs the component, assigns ownership to this GameObject, and registers it so subsystem hooks can run (e.g., renderable registration, script lifecycle queueing, physics setup).

- template<typename T> std::shared_ptr<T> GetComponent()
  - Parameters:
    - None
  - Return:
    - std::shared_ptr<T> - first matching component on this object, or null.

- template<typename T> std::vector<std::shared_ptr<T>> GetComponents()
  - Parameters:
    - None
  - Return:

To Table of Contents

        - std::vector<std::shared_ptr<T>> - all matching components on this object.

- template<typename T> std::shared_ptr<T> GetComponentInChildren()
    - Parameters:
        - None
    - Return:
        - std::shared_ptr<T> - first matching component in children (DFS), or null.

- template<typename T> std::vector<std::shared_ptr<T>> GetComponentsInChildren()
    - Parameters:
        - None
    - Return:
        - std::vector<std::shared_ptr<T>> - all matching components in children.

- template<typename T> std::shared_ptr<T> GetComponentInParent()
    - Parameters:
        - None
    - Return:
        - std::shared_ptr<T> - first matching component in parents, or null.

- template<typename T> std::vector<std::shared_ptr<T>> GetComponentsInParent()
    - Parameters:
        - None
    - Return:
        - std::vector<std::shared_ptr<T>> - all matching components in parents.

- std::shared_ptr<Component> GetComponentByName(const std::string& componentName)
    - Parameters:
        - const std::string& componentName - component name to search for (component’s own name, not GameObject name).
    - Return:
        - std::shared_ptr<Component> - first match, or null.

- size_t GetComponentIndex(const Component *component) const*
    - Parameters:
        - const Component *component - component pointer to locate.*

To Table of Contents

    - Return:
        - size_t - index of the component in the owned component list (stable ordering for deterministic tie-breaks).

- std::shared_ptr<Object> Clone() const override
    - Parameters:
        - None
    - Return:
        - std::shared_ptr<Object> - cloned object (returned as Object base pointer).
    - Description:
        - Clones the GameObject and its components. When cloning, the Transform and component ordering is preserved so later systems (render ordering, physics constraints) behave consistently.

- static std::shared_ptr<GameObject> Find(const std::string& nameOrPath)
    - Parameters:
        - const std::string& nameOrPath - object name or path like Root/Child/SubChild.
    - Return:
        - std::shared_ptr<GameObject> - match if found in any active scene, otherwise null.

- static Scene *GetScene(int instanceID)*
    - Parameters:
        - int instanceID - GameObject instance ID.
    - Return:
        - Scene - *owning Scene if found, otherwise null.*

- static void SetGameObjectsActive(const std::vector<int>& ids, bool active)
    - Parameters:
        - const std::vector<int>& ids - list of instance IDs.
        - bool active - activation state to apply.
    - Return:
        - void - batch toggles active state.

- private void UpdateActiveInHierarchy()
    - Parameters:
        - None
    - Return:
        - void - recomputes activeInHierarchy and propagates to children.
    - Description:
        - Combines this object's activeSelf with its parent's

To Table of Contents

activeInHierarchy, then recursively updates children.

- private void RegisterComponent(const std::shared_ptr<Component>& component)
    - Parameters:
        - const std::shared_ptr<Component>& component - component to attach/register.
    - Return:
        - void - stores the component and triggers any system registration logic.

- private void UnregisterComponent(const std::shared_ptr<Component>& component)
    - Parameters:
        - const std::shared_ptr<Component>& component - component to detach/unregister.
    - Return:
        - void - removes the component and triggers system unregistration logic.

- private void RemoveComponent(const Component *component)*
    - Parameters:
        - const Component *component - raw pointer used to find and remove the matching component.*
    - Return:
        - void - removes the component from the list and clears its owner pointer.

- private void QueueLifecycle(MonoBehaviour *behaviour)*
    - Parameters:
        - MonoBehaviour *behaviour - script to enqueue for lifecycle processing.*
    - Return:
        - void - forwards lifecycle work to the owning Scene.

- private void HandleActivationChange(bool wasActive)
    - Parameters:
        - bool wasActive - previous activeInHierarchy value.
    - Return:
        - void - reacts to activation changes by queueing enable/disable work for behaviours.

- protected void DestroyImmediateInternal() override
    - Parameters:

To Table of Contents

        - None
    - Return:
        - void - immediate destruction hook called from Object::ProcessDestroyQueue.
    - Description:
        - Detaches from Scene, destroys components (and any hierarchy children via Transform links), and unregisters subsystem hooks through component destruction.

Component:

- Files:
    - GameEngine/Component.h
    - GameEngine/Component.cpp

- Summary:
    - Base class for attachable features on a GameObject. Holds a non-owning pointer to the owner and provides the common component query helpers (self, children, parents).

Variables:

- GameObject *m_gameObject - non-owning pointer to the owning GameObject (assigned when the component is added).*

Methods:

- explicit Component(const std::string& name = "Component")
    - Parameters:
        - const std::string& name - component debug name.
    - Return:
        - void - constructs the component.

- GameObject *GetGameObject() const*
    - Parameters:
        - None
    - Return:
        - GameObject *- owning GameObject pointer.*

To Table of Contents

- Transform *GetTransform() const*
    - Parameters:
        - None
    - Return:
        - Transform - *owner's Transform pointer.*

- const std::string& GetName() const override
    - Parameters:
        - None
    - Return:
        - const std::string& - returns the owning GameObject name (Unity-like behaviour).

- void SetName(const std::string& name) override
    - Parameters:
        - const std::string& name - new name.
    - Return:
        - void - sets the owning GameObject name (Unity-like behaviour).

- const std::string& GetComponentName() const
    - Parameters:
        - None
    - Return:
        - const std::string& - returns the component's own name (independent from GameObject name).

- void SetComponentName(const std::string& name)
    - Parameters:
        - const std::string& name - new component name.
    - Return:
        - void - sets the component's own name.

- template<typename T> std::shared_ptr<T> GetComponent()
    - Parameters:
        - None
    - Return:
        - std::shared_ptr<T> - first matching component on the same GameObject, or null.

- template<typename T> std::vector<std::shared_ptr<T>> GetComponents()
    - Parameters:
        - None
    - Return:
        - std::vector<std::shared_ptr<T>> - all matching components on

To Table of Contents

the same GameObject.

- template<typename T> std::shared_ptr<T> GetComponentInChildren()
  - Parameters:
    - None
  - Return:
    - std::shared_ptr<T> - first matching component found in children, or null.

- template<typename T> std::vector<std::shared_ptr<T>> GetComponentsInChildren()
  - Parameters:
    - None
  - Return:
    - std::vector<std::shared_ptr<T>> - all matching components found in children.

- template<typename T> std::shared_ptr<T> GetComponentInParent()
  - Parameters:
    - None
  - Return:
    - std::shared_ptr<T> - first matching component found in parents, or null.

- template<typename T> std::vector<std::shared_ptr<T>> GetComponentsInParent()
  - Parameters:
    - None
  - Return:
    - std::vector<std::shared_ptr<T>> - all matching components found in parents.

- size_t GetComponentIndex() const
  - Parameters:
    - None
  - Return:
    - size_t - index of this component in the owning GameObject's component list.

- virtual std::shared_ptr<Component> Clone() const
  - Parameters:
    - None
  - Return:
    - std::shared_ptr<Component> - clone of the component (default

To Table of Contents

returns null unless overridden).

- protected void DestroyImmediateInternal() override
    - Parameters:
        - None
    - Return:
        - void - clears ownership pointers and performs component-level immediate teardown.

Behaviour:

- Files:
    - GameEngine/Behaviour.h
    - GameEngine/Behaviour.cpp

- Summary:
    - Adds enable/disable state to a Component and provides the ActiveAndEnabled check used to gate updates and callbacks.

Variables:

- bool m_enabled = true - whether the behaviour is enabled.

Methods:

- explicit Behaviour(const std::string& name = "Behaviour")
    - Parameters:
        - const std::string& name - behaviour debug name.
    - Return:
        - void - constructs the behaviour.

- bool IsEnabled() const
    - Parameters:
        - None
    - Return:
        - bool - current enabled state.

- void SetEnabled(bool enabled)
    - Parameters:

To Table of Contents

        - bool enabled - new enabled state.
    - Return:
        - void - updates enabled state and notifies derived classes.
    - Description:
        - If the enabled state changes, calls OnEnabledStateChanged(enabled) so MonoBehaviour can dispatch OnEnable/OnDisable.

- bool IsActiveAndEnabled() const
    - Parameters:
        - None
    - Return:
        - bool - true only if the owner is active in hierarchy and this behaviour is enabled.

- protected virtual bool HasOnEnableBeenCalled() const
    - Parameters:
        - None
    - Return:
        - bool - whether OnEnable has been dispatched already (MonoBehaviour overrides to report real state).

- protected virtual void OnEnabledStateChanged(bool enabled)
    - Parameters:
        - bool enabled - new enabled state.
    - Return:
        - void - hook for derived behaviours (MonoBehaviour uses it to manage enable/disable callbacks and invoke policies).

MonoBehaviour:

- Files:
    - GameEngine/MonoBehaviour.h
    - GameEngine/MonoBehaviour.cpp

- Summary:
    - Script-style Behaviour with Unity-like lifecycle callbacks and an invoke/timer system. The engine drives its lifecycle deterministically through Scene queues instead of calling callbacks immediately at creation time.

To Table of Contents

Variables:

- bool m_didAwake = false - true after Awake has been executed once.
- bool m_didStart = false - true after Start has been executed once.
- bool m_onEnableCalled = false - tracks whether OnEnable has been dispatched for the current active/enabled state.
- bool m_hasEverBeenActive = false - tracks whether the behaviour has ever been active in hierarchy.
- bool m_destroyCallbacksSent = false - prevents duplicate OnDestroy dispatch.

- std::vector<InvokeRequest> m_invokes - scheduled invoke entries.
- std::unordered_map<std::string, std::function<void()>> m_invokeHandlers - methodName to handler mapping for name-based invokes.
- InvokeHandle m_nextInvokeId = 1 - monotonically increasing handle generator.

Methods:

- MonoBehaviour()
  - Parameters:
    - None
  - Return:
    - void - constructs the behaviour and initializes internal state.

- bool IsMarkedForDestruction() const
  - Parameters:
    - None
  - Return:
    - bool - true if destruction has been requested for this behaviour.

- bool DidAwake() const
  - Parameters:
    - None
  - Return:
    - bool - true if Awake has already run.

- bool DidStart() const
  - Parameters:
    - None
  - Return:
    - bool - true if Start has already run.

To Table of Contents

- Engine dispatch helpers (called by Scene, not user code):
  - void InternalUpdate()
    - Parameters:
      - None
    - Return:
      - void - calls Update().
  - void InternalTickInvokes(float now)
    - Parameters:
      - float now - current time in seconds used for invoke scheduling.
    - Return:
      - void - advances the invoke timers and executes any due invokes.
  - void InternalFixedUpdate()
    - Parameters:
      - None
    - Return:
      - void - calls FixedUpdate().
  - void InternalLateUpdate()
    - Parameters:
      - None
    - Return:
      - void - calls LateUpdate().
  - void InternalOnCollisionEnter(Collider2D *other)*
    - Parameters:
      - Collider2D *other - collider involved in the collision.*
    - Return:
      - void - calls OnCollisionEnter(other).
  - void InternalOnCollisionStay(Collider2D *other)*
    - Parameters:
      - Collider2D *other - collider involved in the collision.*
    - Return:
      - void - calls OnCollisionStay(other).
  - void InternalOnCollisionExit(Collider2D *other)*
    - Parameters:
      - Collider2D *other - collider involved in the collision.*
    - Return:
      - void - calls OnCollisionExit(other).
  - void InternalOnTriggerEnter(Collider2D *other)*
    - Parameters:
      - Collider2D *other - trigger collider involved.*
    - Return:
      - void - calls OnTriggerEnter(other).

To Table of Contents

    - void InternalOnTriggerStay(Collider2D *other)*
        - Parameters:
            - Collider2D *other - trigger collider involved.*
        - Return:
            - void - calls OnTriggerStay(other).
    - void InternalOnTriggerExit(Collider2D *other)*
        - Parameters:
            - Collider2D *other - trigger collider involved.*
        - Return:
            - void - calls OnTriggerExit(other).
    - void InternalReset()
        - Parameters:
            - None
        - Return:
            - void - calls Reset().

- Invoke API (function-based):
    - InvokeHandle Invoke(std::function<void()> fn, float delaySeconds, InvokeTickPolicy policy = InvokeTickPolicy::WhileGameObjectActive)
        - Parameters:
            - std::function<void()> fn - callback to run.
            - float delaySeconds - delay before the first call.
            - InvokeTickPolicy policy - controls whether time advances while active or while enabled.
        - Return:
            - InvokeHandle - handle used to cancel/pause/resume the invoke.
    - InvokeHandle InvokeRepeating(std::function<void()> fn, float delaySeconds, float rateSeconds, InvokeTickPolicy policy = InvokeTickPolicy::WhileGameObjectActive)
        - Parameters:
            - std::function<void()> fn - callback to run.
            - float delaySeconds - delay before the first call.
            - float rateSeconds - repeat interval (seconds).
            - InvokeTickPolicy policy - controls whether time advances while active or while enabled.
        - Return:
            - InvokeHandle - handle used to cancel/pause/resume the invoke.
    - void CancelInvoke(InvokeHandle handle)
        - Parameters:
            - InvokeHandle handle - handle to cancel.
        - Return:
            - void - cancels the scheduled invoke if it exists.

To Table of Contents

    - bool IsInvoking(InvokeHandle handle) const
        - Parameters:
            - InvokeHandle handle - handle to check.
        - Return:
            - bool - true if a scheduled invoke exists and is not cancelled.
    - bool PauseInvoke(InvokeHandle handle)
        - Parameters:
            - InvokeHandle handle - handle to pause.
        - Return:
            - bool - true if a matching invoke was found and paused.
    - bool ResumeInvoke(InvokeHandle handle)
        - Parameters:
            - InvokeHandle handle - handle to resume.
        - Return:
            - bool - true if a matching paused invoke was found and resumed.
    - bool RestartInvoke(InvokeHandle handle)
        - Parameters:
            - InvokeHandle handle - handle to restart.
        - Return:
            - bool - true if a matching invoke was found and restarted.
    - void PauseAllInvokes()
        - Parameters:
            - None
        - Return:
            - void - pauses all scheduled invokes on this behaviour.
    - void ResumeAllInvokes()
        - Parameters:
            - None
        - Return:
            - void - resumes all paused invokes on this behaviour.

- Invoke API (name-based):
    - void Invoke(const std::string& methodName, float time)
        - Parameters:
            - const std::string& methodName - message name to execute.
            - float time - delay before execution.
        - Return:
            - void - schedules a named message to be dispatched later.
    - void InvokeRepeating(const std::string& methodName, float delay, float rate)

        - Parameters:
            - const std::string& methodName - message name to execute.
            - float delay - initial delay.
            - float rate - repeat interval.
        - Return:
            - void - schedules a named message repeatedly.
    - void CancelInvoke(const std::string& methodName = "")
        - Parameters:
            - const std::string& methodName - name to cancel; empty cancels all.
        - Return:
            - void - cancels named invokes or all invokes.
    - bool IsInvoking(const std::string& methodName) const
        - Parameters:
            - const std::string& methodName - name to test.
        - Return:
            - bool - true if an invoke for this name is scheduled.
    - void RegisterInvokeHandler(const std::string& methodName, std::function<void()> handler)
        - Parameters:
            - const std::string& methodName - name to bind.
            - std::function<void()> handler - callback to call when that name is invoked.
        - Return:
            - void - registers the handler.
    - void UnregisterInvokeHandler(const std::string& methodName)
        - Parameters:
            - const std::string& methodName - name to unbind.
        - Return:
            - void - unregisters the handler.

- Lifecycle hooks (override in user scripts):
    - virtual void Reset()
        - Parameters:
            - None
        - Return:
            - void - restores default values (editor-like pattern).
    - virtual void Awake()
        - Parameters:
            - None
        - Return:
            - void - called once when the behaviour is first initialized.
    - virtual void Start()

To Table of Contents

        - Parameters:
            - None
        - Return:
            - void - called once before the first Update when enabled.
    - virtual void Update()
        - Parameters:
            - None
        - Return:
            - void - called every frame while active and enabled.
    - virtual void FixedUpdate()
        - Parameters:
            - None
        - Return:
            - void - called every fixed step while active and enabled.
    - virtual void LateUpdate()
        - Parameters:
            - None
        - Return:
            - void - called after Update while active and enabled.
    - virtual void OnEnable()
        - Parameters:
            - None
        - Return:
            - void - called when the behaviour becomes enabled while active.
    - virtual void OnDisable()
        - Parameters:
            - None
        - Return:
            - void - called when the behaviour becomes disabled or inactive.
    - virtual void OnDestroy()
        - Parameters:
            - None
        - Return:
            - void - called once when the object is being destroyed.
    - virtual void OnCollisionEnter(Collider2D *other)*
        - Parameters:
            - Collider2D *other - collider involved in the collision.*
        - Return:
            - void - called on collision start.
    - virtual void OnCollisionStay(Collider2D *other)*
        - Parameters:
            - Collider2D *other - collider involved in the collision.*

To Table of Contents

        - Return:
            - void - called while collision continues.
    - virtual void OnCollisionExit(Collider2D *other)*
        - Parameters:
            - Collider2D *other - collider involved in the collision.*
        - Return:
            - void - called on collision end.
    - virtual void OnTriggerEnter(Collider2D *other)*
        - Parameters:
            - Collider2D *other - trigger collider involved.*
        - Return:
            - void - called on trigger start.
    - virtual void OnTriggerStay(Collider2D *other)*
        - Parameters:
            - Collider2D *other - trigger collider involved.*
        - Return:
            - void - called while trigger continues.
    - virtual void OnTriggerExit(Collider2D *other)*
        - Parameters:
            - Collider2D *other - trigger collider involved.*
        - Return:
            - void - called on trigger end.

- Internal lifecycle and invoke helpers (managed by Scene/GameObject):
    - void ReceiveMessage(const std::string& methodName)
        - Parameters:
            - const std::string& methodName - invoked name.
        - Return:
            - void - default handler for name-based invokes (used when a matching handler is registered).
    - bool HasOnEnableBeenCalled() const override
        - Parameters:
            - None
        - Return:
            - bool - reports whether OnEnable has already been dispatched.
    - void OnEnabledStateChanged(bool enabled) override
        - Parameters:
            - bool enabled - new enabled state.
        - Return:
            - void - manages OnEnable/OnDisable dispatch and invoke policies when enabled toggles.

To Table of Contents

Transform

- Files: Transform.h, Transform.cpp
- Summary: Stores local position, rotation, and scale, plus parent/child links. Provides cached world-space values and matrices. When a Rigidbody2D exists on the same GameObject, Transform also pushes position/rotation changes into the physics body, and can be updated from physics without causing feedback loops.

Variables:

- Vector2f m_localPosition - Local-space position relative to parent. World position is derived from this and the parent transform.
- float m_localRotation - Local-space rotation in degrees (counter-clockwise).
- Vector2f m_localScale - Local-space scale relative to parent. Negative values are allowed and are used for mirroring.

- Transform *m_parent - Parent transform in the hierarchy. Null means this transform is a root.*
- std::vector<Transform> *m_children - Direct children transforms.*

- mutable bool m_hasChanged - Dirty flag used to signal that local values (or hierarchy) changed since last cache update.
- mutable bool m_worldMatrixDirty - Dirty flag for the cached world matrix and derived world values.

- mutable Vector2f m_worldPosition - Cached world-space position (center-based, +Y up).
- mutable float m_worldRotation - Cached world-space rotation in degrees.
- mutable Vector2f m_worldScale - Cached world-space scale.
- mutable Matrix3x3f m_worldMatrix - Cached world transform matrix.

Methods:

- Transform()
  - Return:
    - void - Initializes local values to defaults and marks caches as dirty.

- Vector2f GetPosition() const
    - Return:
        - Vector2f - The current local-space position.

- float GetRotation() const
    - Return:
        - float - The current local-space rotation in degrees.

- Vector2f GetScale() const
    - Return:
        - Vector2f - The current local-space scale.

- Vector2f GetWorldPosition() const
    - Return:
        - Vector2f - Cached world-space position. Updates caches first if dirty.

- float GetWorldRotation() const
    - Return:
        - float - Cached world-space rotation in degrees. Updates caches first if dirty.

- Vector2f GetWorldScale() const
    - Return:
        - Vector2f - Cached world-space scale. Updates caches first if dirty.

- Transform *GetParent() const*
    - Return:
        - Transform - *Pointer to the parent transform (or null if root).*

- const std::vector<Transform>& *GetChildren() const*
    - Return:
        - const std::vector<Transform>& - *Reference to the child list.*

- void SetParent(Transform *parent)*
    - Return:
        - void - Reparents this transform under a new parent while keeping hierarchy lists consistent.
    - Parameters:
        - Transform *parent - New parent transform, or null to become a root.*
    - Steps:
        - If the parent is unchanged, do nothing.

To Table of Contents

        - Remove this transform from the old parent's child list.
        - Assign the new parent and add this transform to the new parent's child list.
        - Mark this transform (and hierarchy) dirty so world caches recompute.

- void AddChild(Transform *child)*
    - Return:
        - void - Adds a child to this transform.
    - Parameters:
        - Transform *child - The transform to add as a child.*
    - Steps:
        - If child is null, do nothing.
        - If child is already present, do nothing.
        - Push child into the children list and mark dirty.

- void RemoveChild(Transform *child)*
    - Return:
        - void - Removes a child from this transform.
    - Parameters:
        - Transform *child - The transform to remove.*
    - Steps:
        - If the child is null, do nothing.
        - Find and erase the child pointer from the vector.
        - Mark is dirty.

- void SetPosition(const Vector2f& position)
    - Return:
        - void - Sets local position and updates derived state.
    - Parameters:
        - const Vector2f& position - New local-space position.
    - Steps:
        - Assign m_localPosition.
        - Mark dirty so caches recompute.
        - If a Rigidbody2D exists, push the new world position into the physics body.

- void SetPosition(float x, float y)
    - Return:
        - void - Convenience overload to set local position by scalars.
    - Parameters:
        - float x - Local x position.
        - float y - Local y position.

To Table of Contents

- void SetRotation(float rotation)
    - Return:
        - void - Sets local rotation (degrees) and updates derived state.
    - Parameters:
        - float rotation - New local rotation in degrees.
    - Steps:
        - Assign m_localRotation.
        - Mark dirty so caches recompute.
        - If a Rigidbody2D exists, push the new world rotation into the physics body.

- void SetRotationRadians(float radians)
    - Return:
        - void - Sets local rotation using radians input.
    - Parameters:
        - float radians - New local rotation in radians.
    - Steps:
        - Convert radians to degrees.
        - Call SetRotation(degrees).

- void SetScale(const Vector2f& scale)
    - Return:
        - void - Sets local scale and marks caches dirty.
    - Parameters:
        - const Vector2f& scale - New local scale (negative allowed).

- void SetScale(float x, float y)
    - Return:
        - void - Convenience overload to set local scale by scalars.
    - Parameters:
        - float x - Local x scale.
        - float y - Local y scale.

- void SetScale(float uniformScale)
    - Return:
        - void - Convenience overload to set uniform local scale.
    - Parameters:
        - float uniformScale - Uniform scale applied to both axes.

- void Translate(const Vector2f& offset)
    - Return:
        - void - Adds an offset to the local position.
    - Parameters:
        - const Vector2f& offset - Local offset to add.

To Table of Contents

  - Steps:
    - Call SetPosition(GetPosition() + offset).

- void Translate(float x, float y)
  - Return:
    - void - Convenience overload for Translate.
  - Parameters:
    - float x - Local x offset.
    - float y - Local y offset.

- void Rotate(float rotation)
  - Return:
    - void - Adds an offset to local rotation.
  - Parameters:
    - float rotation - Delta degrees to add.
  - Steps:
    - Call SetRotation(GetRotation() + rotation).

- Vector2f GetRight() const
  - Return:
    - Vector2f - The transform's right direction in world space.
  - Steps:
    - Convert world rotation to radians.
    - Return (cos, sin).

- Vector2f GetUp() const
  - Return:
    - Vector2f - The transform's up direction in world space.
  - Steps:
    - Convert world rotation to radians.
    - Return (-sin, cos).

- Vector2f TransformDirection(const Vector2f& localDir) const
  - Return:
    - Vector2f - The local direction rotated into world space (ignores position).
  - Parameters:
    - const Vector2f& localDir - Direction in local space.
  - Steps:
    - Rotate localDir by the world rotation.

- Matrix3x3f GetWorldMatrix() const
  - Return:
    - Matrix3x3f - Cached world matrix. Updates caches first if dirty.

To Table of Contents

- Matrix3x3f GetLocalMatrix() const
    - Return:
        - Matrix3x3f - Matrix built from local position/rotation/scale (no parent).

- bool HasChanged() const
    - Return:
        - bool - True if local/hierarchy values changed since the last cache update.

- void SetDirty() const
    - Return:
        - void - Marks this transform and its descendants as dirty (forces cache recompute next read).
    - Steps:
        - Set local dirty flags.
        - Propagate dirty to all children.

- std::shared_ptr<Component> Clone() const
    - Return:
        - std::shared_ptr<Component> - A new Transform component with copied local values and dirty flags.
    - Steps:
        - Copy local position/rotation/scale.
        - Copy dirty flags so the clone will recompute world caches when needed.

- void SetWorldPositionFromPhysics(const Vector2f& worldPosition)
    - Return:
        - void - Updates Transform values using a world-space position coming from physics, converting it back into local space.
    - Parameters:
        - const Vector2f& worldPosition - New world position from the physics body.
    - Steps:
        - If there is no parent, set m_localPosition = worldPosition.
        - If there is a parent:
            1. Compute parent world rotation (degrees) and undo it (rotate by -rotation).
            2. Divide by parent world scale to undo scaling.
            3. Set local position to the resulting value.
        - Mark dirty so cached world values match the new local values.
    - Notes:

To Table of Contents

        - This method updates the Transform without writing back into physics, preventing feedback loops during physics syncing.

- void SetWorldRotationFromPhysics(float worldRotationDegrees)
    - Return:
        - void - Updates Transform rotation from physics, converting world rotation into local rotation.
    - Parameters:
        - float worldRotationDegrees - New world rotation in degrees from the physics body.
    - Steps:
        - If there is no parent, set m_localRotation = worldRotationDegrees.
        - If there is a parent, set local rotation to (worldRotation - parentWorldRotation).
        - Mark dirty so cached world values match the new local values.
    - Notes:
        - This method is intended to be used by Rigidbody2D during SyncTransformFromBody() to avoid transform->body->transform loops.

- void UpdateWorldMatrix() const
    - Return:
        - void - Rebuilds cached world position/rotation/scale/matrix when dirty.
    - Steps:
        - If parent exists, ensure the parent world matrix is up to date.
        - Build local matrix from local position/rotation/scale.
        - Multiply parentWorld *local to produce m_worldMatrix.*
        - Derive m_worldPosition, m_worldRotation, and m_worldScale from the matrix.
        - Clear dirty flags.

- void MarkDirty() const
    - Return:
        - void - Sets dirty flags on this transform and all descendants.

- void SyncBodyPositionFromTransform()
    - Return:
        - void - If a Rigidbody2D exists, writes the current world position into the Box2D body.
    - Steps:
        - Find Rigidbody2D on the same GameObject.
        - If the body id is valid, call body->SetPosition(worldPos).

To Table of Contents

- void SyncBodyRotationFromTransform()
    - Return:
        - void - If a Rigidbody2D exists, writes the current world rotation into the Box2D body.
    - Steps:
        - Find Rigidbody2D on the same GameObject.
        - If the body id is valid, call body->SetRotationRadians(worldRotationRadians).

Scene:

- Files:
    - GameEngine/Scene.h
    - GameEngine/Scene.cpp

- Summary:
    - Owns all GameObjects for a level, coordinates lifecycle processing, and applies safe mutation rules through deferred adoption and queued destruction. A Scene always has exactly one GameMode (defaulting to EmptyGameMode).

Variables:

- std::string m_name - scene name.
- bool m_isActive = false - whether the scene has started and is running.
- bool m_markedForUnload = false - whether the scene is scheduled to unload.

- std::vector<std::shared_ptr<GameObject>> m_rootGameObjects - root-level objects (no parent transform).
- std::vector<std::shared_ptr<GameObject>> m_allGameObjects - all objects owned by this scene.
- std::unordered_map<int, std::shared_ptr<GameObject>> m_gameObjectById - fast lookup by instance ID.

- std::vector<MonoBehaviour> *m_pendingLifecycle - queued lifecycle work for scripts/behaviours.*
- std::vector<std::shared_ptr<GameObject>> m_pendingAdopt - objects created but not yet adopted into the scene lists.

To Table of Contents

- ObjectPool m_objectPool - per-scene object pooling buckets.
- std::unique_ptr<GameMode> m_gameMode - the rules/state container for this scene.

- static std::vector<Scene> *s_scenes - global list of active scenes (for FindGameObject and cross-scene queries).*

Methods:

- explicit Scene(const std::string& name = "Scene")
    - Parameters:
        - const std::string& name - scene debug name.
    - Return:
        - void - constructs the scene.

- virtual ~Scene()
    - Parameters:
        - None
    - Return:
        - void - destroys scene resources on teardown.

- virtual void OnCreate()
    - Parameters:
        - None
    - Return:
        - void - hook for derived scenes before Start() is called.

- virtual void OnStart()
    - Parameters:
        - None
    - Return:
        - void - hook called when the scene becomes active.

- virtual void OnUpdate()
    - Parameters:
        - None
    - Return:
        - void - hook called each frame after core scene update.

- virtual void OnFixedUpdate()
    - Parameters:

To Table of Contents

        - None
    - Return:
        - void - hook called each fixed step after core fixed update.

- virtual void OnLateUpdate()
    - Parameters:
        - None
    - Return:
        - void - hook called each frame after core late update.

- virtual void OnRender()
    - Parameters:
        - None
    - Return:
        - void - hook called during render (after RenderSystem queue execution and before UI).

- virtual void OnDestroy()
    - Parameters:
        - None
    - Return:
        - void - hook called when the scene is unloading.

- template<typename T> std::shared_ptr<T> CreateGameObject(const std::string& name = "GameObject")
    - Parameters:
        - const std::string& name - debug name for the new object.
    - Return:
        - std::shared_ptr<T> - newly created GameObject (queued for adoption).
    - Description:
        - Creates the object, registers it globally, and queues adoption so the scene's lists are mutated only at safe points.

- void Start()
    - Parameters:
        - None
    - Return:
        - void - activates the scene and runs startup hooks.
    - Description:
        - Ensures a GameMode exists, calls OnCreate/OnStart hooks, then begins lifecycle processing for objects and behaviours.

- void Update()

To Table of Contents

- Parameters:
  - None
- Return:
  - void - runs variable-step update for the scene.
- Description:
  - Applies pending adopts, processes lifecycle queue, calls GameMode update hook, then updates eligible behaviours.

- void FixedUpdate()
  - Parameters:
    - None
  - Return:
    - void - runs fixed-step update for the scene.
  - Description:
    - Drives fixed callbacks for behaviours and is typically paired with physics stepping in the engine loop.

- void LateUpdate()
  - Parameters:
    - None
  - Return:
    - void - runs late update for the scene.

- void Render()
  - Parameters:
    - None
  - Return:
    - void - runs the scene render hook and GameMode render hook.

- void Unload()
  - Parameters:
    - None
  - Return:
    - void - unloads and tears down the scene.
  - Description:
    - Marks for unload, triggers destruction for owned objects, clears registries/pools, and calls OnDestroy hooks.

- GameMode& GetGameMode()
  - Parameters:
    - None
  - Return:
    - GameMode& - reference to the current mode (never null after EnsureGameMode).

To Table of Contents

- void SetGameMode(std::unique_ptr<GameMode> gameMode)
    - Parameters:
        - std::unique_ptr<GameMode> gameMode - new mode instance to own.
    - Return:
        - void - replaces the scene's mode and attaches it.

- template<typename T> T& SetGameMode()
    - Parameters:
        - None
    - Return:
        - T& - reference to the created mode.
    - Description:
        - Creates a new mode of type T, stores it, calls OnAttach, and returns it.

- const std::string& GetName() const
    - Parameters:
        - None
    - Return:
        - const std::string& - scene name.

- bool IsActive() const
    - Parameters:
        - None
    - Return:
        - bool - true if the scene is active/running.

- std::vector<std::shared_ptr<GameObject>>& GetRootGameObjects()
    - Parameters:
        - None
    - Return:
        - std::vector<std::shared_ptr<GameObject>>& - reference to root object list (used for iteration/editing).

- ObjectPool& GetObjectPool()
    - Parameters:
        - None
    - Return:
        - ObjectPool& - reference to the scene's object pool.

- template<typename T, typename Factory> std::shared_ptr<T> CreateGameObjectPooled(const std::string& poolKey, Factory&& factory)

To Table of Contents

  - Parameters:
    - const std::string& poolKey - bucket key.
    - Factory&& factory - function that creates a new object if the pool is empty.
  - Return:
    - std::shared_ptr<T> - object acquired from the pool or created by the factory.
  - Description:
    - Acquires from the pool, ensures scene adoption (if newly created), and returns the object for gameplay use.

- void ReleaseGameObjectToPool(const std::string& poolKey, const std::shared_ptr<GameObject>& obj)
  - Parameters:
    - const std::string& poolKey - bucket key.
    - const std::shared_ptr<GameObject>& obj - object to return to the pool.
  - Return:
    - void - deactivates the object and stores it in the pool.

- static std::shared_ptr<GameObject> FindGameObject(const std::string& nameOrPath)
  - Parameters:
    - const std::string& nameOrPath - object name or path like Root/Child/SubChild.
  - Return:
    - std::shared_ptr<GameObject> - match if found in any active scene, otherwise null.

- Internal helpers (used by GameObject and lifecycle processing):
  - void QueueLifecycle(MonoBehaviour *behaviour)*
    - Parameters:
      - MonoBehaviour *behaviour - behaviour to enqueue.*
    - Return:
      - void - adds to lifecycle queue.
  - void EnsureGameMode()
    - Parameters:
      - None
    - Return:
      - void - assigns an EmptyGameMode if no mode exists.
  - void ProcessLifecycleQueue()
    - Parameters:
      - None
    - Return:

To Table of Contents

            - void - runs Awake/OnEnable/Start work in a stable order.
    - void QueueAdoptGameObject(const std::shared_ptr<GameObject>& obj)
        - Parameters:
            - const std::shared_ptr<GameObject>& obj - object to adopt later.
        - Return:
            - void - adds to pending adopts.
    - bool ProcessPendingAdopts()
        - Parameters:
            - None
        - Return:
            - bool - true if any adopts were processed.
    - void AdoptGameObject(const std::shared_ptr<GameObject>& obj)
        - Parameters:
            - const std::shared_ptr<GameObject>& obj - object to adopt.
        - Return:
            - void - public adoption path (deferred).
    - void AdoptGameObjectImmediate(const std::shared_ptr<GameObject>& obj)
        - Parameters:
            - const std::shared_ptr<GameObject>& obj - object to adopt.
        - Return:
            - void - internal insertion into scene lists and ID lookup.
    - void RemoveGameObject(const GameObject *obj)*
        - Parameters:
            - const GameObject *obj - object to remove.*
        - Return:
            - void - removes object from tracking lists and lookup.
    - void UpdateRootGameObject(const std::shared_ptr<GameObject>& obj)
        - Parameters:
            - const std::shared_ptr<GameObject>& obj - object to re-evaluate as root vs child.
        - Return:
            - void - keeps root list in sync with transform parenting.

ObjectPool:

- Files:

To Table of Contents

    - GameEngine/ObjectPool.h
    - GameEngine/ObjectPool.cpp

- Summary:
    - Simple per-scene pooling map that reuses GameObject instances by key to reduce allocations during spawn-heavy gameplay (bullets, enemies, popups).

Variables:

- std::unordered_map<std::string, std::vector<std::shared_ptr<GameObject>>> m_pool - pool buckets, each a stack of inactive objects.

Methods:

- template<typename T, typename Factory> std::shared_ptr<T> Acquire(const std::string& poolKey, Factory&& factory)
    - Parameters:
        - const std::string& poolKey - bucket key to acquire from.
        - Factory&& factory - creation function used when the bucket is empty.
    - Return:
        - std::shared_ptr<T> - pooled object cast to T, or newly created object.
    - Description:
        - Pops from the back of the bucket (most recently released), skips null/destroyed entries, dynamic-casts to the requested type, reactivates it, and returns it. If nothing valid exists, calls factory().

- void Release(const std::string& poolKey, const std::shared_ptr<GameObject>& obj)
    - Parameters:
        - const std::string& poolKey - bucket key to release into.
        - const std::shared_ptr<GameObject>& obj - object to store.
    - Return:
        - void - deactivates the object and stores it for reuse.

- void Clear()
    - Parameters:
        - None

To Table of Contents

    - Return:
        - void - clears all buckets and releases pooled references.

Notes:

- Currently not working properly. Need redone.

GameMode:

- Files:
    - GameEngine/GameMode.h
- Summary:
    - Per-scene rules container (spawning, win/lose, score, wave logic). A Scene always owns exactly one GameMode instance.

Variables:

- None (interface class).

Methods:

- virtual void OnAttach(Scene& scene)
    - Parameters:
        - Scene& scene - scene that owns this mode.
    - Return:
        - void - called when the mode is assigned to a scene.
- virtual void OnStart()
    - Parameters:
        - None
    - Return:
        - void - called when the scene starts running.
- virtual void OnUpdate()
    - Parameters:
        - None
    - Return:
        - void - called once per frame before scene/behaviour Update.

To Table of Contents

- virtual void OnFixedUpdate()
    - Parameters:
        - None
    - Return:
        - void - called on each fixed step before scene/behaviour FixedUpdate.

- virtual void OnLateUpdate()
    - Parameters:
        - None
    - Return:
        - void - called after Update and before LateUpdate.

- virtual void OnRender()
    - Parameters:
        - None
    - Return:
        - void - called during render before the scene's render hook.

- virtual void OnDestroy()
    - Parameters:
        - None
    - Return:
        - void - called when the scene is unloading.

- virtual const char *GetDebugName() const*
    - Parameters:
        - None
    - Return:
        - const char - *debug label for logging.*

GameInstance:

- Files:
    - GameEngine/GameInstance.h

- Summary:
    - Global, long-lived object created once when the engine starts and destroyed on shutdown. Intended for persistent data (options, save data, global managers) that should survive scene transitions.

To Table of Contents

Methods:

- virtual void OnInit()
    - Parameters:
        - None
    - Return:
        - void - called once after the engine initializes core systems.

- virtual void OnShutdown()
    - Parameters:
        - None
    - Return:
        - void - called once right before the engine shuts down.

- virtual const char *GetDebugName() const*
    - Parameters:
        - None
    - Return:
        - const char - *debug label for logging.*

Physics subsystem (Box2D)

- Files:
    a. Physics2D.h/.cpp
    b. Rigidbody2D.h/.cpp
    c. Collider2D.h/.cpp
- Summary: Wraps Box2D for 2D simulation, keeps a registry of active Rigidbody2D and Collider2D components, steps the world on a fixed timestep, and dispatches collision and trigger callbacks to MonoBehaviours.

Physics2DWorld:

- Files: Physics2D.h/.cpp
- Summary: Stores the Box2D world handle, tracks registered bodies and colliders, steps the simulation, and dispatches collision/trigger events (Enter/Stay/Exit) by translating Box2D events into MonoBehaviour callbacks.

Variables:

- b2WorldId m_worldId - Stored Box2D world handle. b2_nullWorldId means

[To Table of Contents]

the world is not initialized.

- std::unordered_set<Rigidbody2D> *m_registeredBodies - Registry of all active Rigidbody2D components that currently belong to this world.*
- std::unordered_set<Collider2D> *m_registeredColliders - Registry of all active Collider2D components that currently belong to this world.*
- std::set<ColliderPair, ColliderPairLess> m_activeCollisions - Canonical collider pointer pairs used to synthesize OnCollisionStay.
- std::set<ColliderPair, ColliderPairLess> m_activeTriggers - Canonical collider pointer pairs used to synthesize OnTriggerStay.
- struct ColliderPair - Private canonical pair type where pointers are stored in a stable (order-independent) way so (A,B) equals (B,A).
- struct ColliderPairLess - Private comparator used by the sets to keep ColliderPair ordered and searchable.

Methods:

- ~Physics2DWorld()
  - Return:
    - void - Ensures Shutdown() is called so the Box2D world handle is destroyed and registries are cleared.

- void Initialize(const Vector2f& gravity)
  - Return:
    - void - Creates the Box2D world and sets its initial gravity.
  - Parameters:
    - const Vector2f& gravity - Gravity vector in world units per second squared.
  - Steps:
    - If a world already exists, early-out (prevents double initialization).
    - Build a Box2D world definition and copy gravity into it.
    - Create a new Box2D world handle and store it in m_worldId.
    - Clear active contact caches so there are no stale Stay pairs.

- void Reset(const Vector2f& gravity)
  - Return:
    - void - Recreates the Box2D world while keeping existing engine components registered, then rebuilds their Box2D bodies and shapes.
  - Parameters:
    - const Vector2f& gravity - Gravity vector for the new world instance.
  - Steps:

To Table of Contents

        - Copy the current registered body and collider pointers into local temporary sets.
        - Shutdown the existing world (destroy handle, clear caches).
        - Initialize a new world with the requested gravity.
        - For each saved Rigidbody2D pointer:
            - RegisterBody(body) so the registry is restored.
            - body->RecreateBody() to create a new Box2D body in the new world.
        - For each saved Collider2D pointer:
            - RegisterCollider(collider) so the registry is restored.
            - collider->RebuildShape() to create a new Box2D shape attached to the rebuilt body.

- void Shutdown()
    - Return:
        - void - Destroys the Box2D world and clears registries and contact caches.
    - Steps:
        - If the world handle is valid, destroy it via Box2D.
        - Set m_worldId back to b2_nullWorldId.
        - Clear registered bodies and colliders.
        - Clear active collision and trigger pair sets.

- void Step(float timeStep, int subStepCount)
    - Return:
        - void - Advances the Box2D simulation and dispatches physics events for this step.
    - Parameters:
        - float timeStep - The timestep used for this simulation step (typically Time::FixedDeltaTime()).
        - int subStepCount - Sub-iterations for the Box2D step (stability/quality control).
    - Steps:
        - Early-out if the world is not initialized.
        - Step Box2D using the provided timestep and substep count.
        - Read Box2D contact begin/end events and map shapes back to Collider2D pointers using the shape user data.
        - For each begin contact:
            - Build a canonical collider pair so ordering does not matter.
            - If either collider is configured as a trigger, store the pair in m_activeTriggers and call TriggerEnter on both sides.
            - Otherwise, store the pair in m_activeCollisions and call CollisionEnter on both sides.

To Table of Contents

        - For each end contact:
            - Remove the pair from the relevant active set and dispatch the matching Exit callback.
        - Read Box2D sensor begin/end events:
            - If neither collider requested sensor events, ignore the event.
            - Otherwise, insert/erase the pair in m_activeTriggers and dispatch TriggerEnter/TriggerExit.
        - Synthesize Stay callbacks:
            - For all pairs still in m_activeCollisions, dispatch CollisionStay to both sides.
            - For all pairs still in m_activeTriggers, dispatch TriggerStay to both sides.
        - Synchronize transforms:
            - For each registered Rigidbody2D, call SyncTransformFromBody() so Transform matches Box2D after simulation.

- void DebugDraw(Renderer& renderer) const
    - Return:
        - void - Draws collider outlines for debugging using the engine Renderer.
    - Parameters:
        - Renderer& renderer - Renderer used to draw debug primitives.
    - Steps:
        - Iterate all registered colliders.
        - Skip invalid colliders and inactive objects.
        - Compute world position from Transform world position plus collider offset rotated into world space.
        - If the collider is a box, draw a rotated rectangle outline.
        - If the collider is a circle, draw a circle outline.
        - Use a different color for triggers versus solid colliders so overlaps are easy to inspect.

- bool IsValid() const
    - Return:
        - bool - True if the world handle is initialized and not null.

- b2WorldId GetWorldId() const
    - Return:
        - b2WorldId - The underlying Box2D world handle.

- void SetGravity(const Vector2f& gravity)
    - Return:

        - void - Updates the world gravity in Box2D.
    - Parameters:
        - const Vector2f& gravity - New gravity vector.

- Vector2f GetGravity() const
    - Return:
        - Vector2f - Current gravity vector read from Box2D.

- void RegisterBody(Rigidbody2D *body)*
    - Return:
        - void - Adds a Rigidbody2D to the registry so it can be recreated and synchronized.
    - Parameters:
        - Rigidbody2D *body - Rigidbody component to register (non-owning).*

- void UnregisterBody(Rigidbody2D *body)*
    - Return:
        - void - Removes a Rigidbody2D from the registry.
    - Parameters:
        - Rigidbody2D *body - Rigidbody component to unregister (non-owning).*

- void RegisterCollider(Collider2D *collider)*
    - Return:
        - void - Adds a Collider2D to the registry so it can be debug-drawn and its contacts can be tracked.
    - Parameters:
        - Collider2D *collider - Collider component to register (non-owning).*

- void UnregisterCollider(Collider2D *collider)*
    - Return:
        - void - Removes a Collider2D from the registry and drops any cached contact pairs that involve it.
    - Parameters:
        - Collider2D *collider - Collider component to unregister (non-owning).*
    - Steps:
        - Erase collider from m_registeredColliders.
        - ClearContactCacheFor(collider) so OnCollisionStay/OnTriggerStay stops immediately.

- void ClearContactCacheFor(Collider2D *collider)*

To Table of Contents

- Return:
  - void - Removes any collision or trigger pairs involving this collider from the active caches.
- Parameters:
  - Collider2D *collider - Collider whose cached pairs must be removed.*
- Steps:
  - Remove any pair from m_activeCollisions where a or b matches the collider.
  - Remove any pair from m_activeTriggers where a or b matches the collider.

Rigidbody2D:

- Files: Rigidbody2D.h/.cpp
- Summary: Component that owns a Box2D body. It exposes velocity/force APIs and keeps the GameObject Transform synchronized with the simulated body.

Variables:

- b2BodyId m_bodyId - Box2D body handle owned by this component (b2_nullBodyId means no body exists).
- BodyType m_bodyType - Requested body type (Static, Kinematic, Dynamic).
- float m_gravityScale - Per-body gravity multiplier.
- float m_linearDamping - Linear damping applied by Box2D.
- float m_angularDamping - Angular damping applied by Box2D.
- bool m_fixedRotation - If true, prevents rotation in Box2D.
- bool m_allowSleep - If true, Box2D may allow this body to sleep when inactive.
- bool m_isBullet - If true, enables continuous collision detection for fast-moving bodies.

Methods:

- void Initialize()
  - Return:
    - void - Creates the Box2D body in the active Physics2DWorld and registers it.
  - Steps:
    - CreateBody() to build a Box2D body definition from current settings and Transform state.

To Table of Contents

        - AttachExistingColliders() so any Collider2D already present on the GameObject is rebuilt against the new body.

- void Shutdown()
    - Return:
        - void - Destroys the Box2D body and clears registration.
    - Steps:
        - DetachExistingColliders() to destroy shapes cleanly before the body disappears.
        - DestroyBody() to remove the Box2D body from the world and reset m_bodyId.

- void RecreateBody()
    - Return:
        - void - Rebuilds the Box2D body (destroy + create) while preserving component configuration.
    - Steps:
        - DetachExistingColliders() so shapes do not reference an old body id.
        - DestroyBody().
        - CreateBody().
        - AttachExistingColliders() to recreate shapes on the new body.

- void SetBodyType(BodyType type)
    - Return:
        - void - Sets the body type and, if a body exists, recreates it so Box2D reflects the new type.
    - Parameters:
        - BodyType type - New body type.

- BodyType GetBodyType() const
    - Return:
        - BodyType - Current requested body type.

- void SetGravityScale(float scale)
    - Return:
        - void - Sets gravity scale and updates the Box2D body if present.
    - Parameters:
        - float scale - New gravity scale.

- float GetGravityScale() const
    - Return:
        - float - Current gravity scale.

To Table of Contents

- void SetLinearVelocity(const Vector2f& velocity)
    - Return:
        - void - Directly sets the body linear velocity.
    - Parameters:
        - const Vector2f& velocity - New velocity in world units per second.

- Vector2f GetLinearVelocity() const
    - Return:
        - Vector2f - Current linear velocity.

- void SetAngularVelocity(float velocity)
    - Return:
        - void - Directly sets angular velocity.
    - Parameters:
        - float velocity - Angular velocity in radians per second.

- float GetAngularVelocity() const
    - Return:
        - float - Current angular velocity in radians per second.

- void ApplyForce(const Vector2f& force, const Vector2f& point, bool wake)
    - Return:
        - void - Applies a continuous force at a world-space point.
    - Parameters:
        - const Vector2f& force - Force vector.
        - const Vector2f& point - World-space point.
        - bool wake - If true, wakes the body.
    - Steps:
        - If there is no body, do nothing.
        - Forward force and point to Box2D.

- void ApplyForceToCenter(const Vector2f& force, bool wake)
    - Return:
        - void - Applies a continuous force at the center of mass.
    - Parameters:
        - const Vector2f& force - Force vector.
        - bool wake - If true, wakes the body.

- void ApplyLinearImpulse(const Vector2f& impulse, const Vector2f& point, bool wake)
    - Return:
        - void - Applies an instantaneous impulse at a world-space point.
    - Parameters:

To Table of Contents

        - const Vector2f& impulse - Impulse vector.
        - const Vector2f& point - World-space point.
        - bool wake - If true, wakes the body.

- void ApplyLinearImpulseToCenter(const Vector2f& impulse, bool wake)
    - Return:
        - void - Applies an instantaneous impulse at the center of mass.
    - Parameters:
        - const Vector2f& impulse - Impulse vector.
        - bool wake - If true, wakes the body.

- void SetPosition(const Vector2f& position)
    - Return:
        - void - Teleports the body to a new position.
    - Parameters:
        - const Vector2f& position - New world-space position.

- void SetRotation(float rotationDegrees)
    - Return:
        - void - Teleports the body rotation using degrees.
    - Parameters:
        - float rotationDegrees - New rotation in degrees.

- void SetRotationRadians(float rotationRadians)
    - Return:
        - void - Teleports the body rotation using radians.
    - Parameters:
        - float rotationRadians - New rotation in radians.

- void SetLinearDamping(float damping)
    - Return:
        - void - Updates linear damping and recreates the body so Box2D uses the new value.
    - Parameters:
        - float damping - New linear damping.

- float GetLinearDamping() const
    - Return:
        - float - Current linear damping.

- void SetAngularDamping(float damping)
    - Return:
        - void - Updates angular damping and recreates the body so Box2D uses the new value.

To Table of Contents

    - Parameters:
        - float damping - New angular damping.

- float GetAngularDamping() const
    - Return:
        - float - Current angular damping.

- void SetFixedRotation(bool fixedRotation)
    - Return:
        - void - Sets fixed-rotation and recreates the body so Box2D applies it.
    - Parameters:
        - bool fixedRotation - True to lock rotation.

- bool IsFixedRotation() const
    - Return:
        - bool - True if rotation is locked.

- void SetIsBullet(bool isBullet)
    - Return:
        - void - Sets bullet mode and recreates the body so Box2D applies it.
    - Parameters:
        - bool isBullet - True to enable continuous collision detection.

- bool IsBullet() const
    - Return:
        - bool - True if bullet mode is enabled.

- b2BodyId GetBodyId() const
    - Return:
        - b2BodyId - Underlying Box2D body handle.

- void SyncTransformFromBody()
    - Return:
        - void - Writes the simulated Box2D position and rotation back to the Transform.
    - Steps:
        - If there is no body, do nothing.
        - Read body position and angle (radians) from Box2D.
        - Convert radians to degrees.
        - Update Transform world position and rotation.

- std::shared_ptr<Component> Clone() const

To Table of Contents

    - Return:
        - std::shared_ptr<Component> - A new Rigidbody2D that copies configuration fields (but not the Box2D body instance).
    - Steps:
        - Copy body type, damping, gravity scale, fixedRotation, allowSleep, and bullet settings into the clone.
        - The clone creates its own body later, when initialized in a Scene.

Collider2D:

- Files: Collider2D.h/.cpp
- Summary: Base collider component that owns a Box2D shape. It can act as a solid collider or as a trigger (sensor) and rebuilds its shape whenever configuration changes.

Variables:

- b2ShapeId m_shapeId - Box2D shape handle created for this collider (b2_nullShapeId means no shape exists).
- Rigidbody2D *m_attachedBody - Rigidbody this collider is currently attached to (non-owning pointer).*
- Vector2f m_offset - Local-space offset applied to the collider geometry.
- float m_density - Shape density (used for mass on dynamic bodies).
- float m_friction - Shape friction used by contacts.
- float m_restitution - Shape bounciness used by contacts.
- bool m_isTrigger - If true, the shape is a sensor and does not create physical contact response.
- bool m_shouldSensorEvent - If true, sensor begin/end events are emitted and routed to MonoBehaviour trigger callbacks.

Methods:

- void Initialize()
    - Return:
        - void - Registers the collider with the Physics2DWorld and creates its Box2D shape on the owning rigidbody.
    - Steps:
        - Find the Rigidbody2D component on the same GameObject.
        - If no rigidbody exists, throw an engine error (this engine requires colliders to be attached to a body).
        - Store the rigidbody pointer in m_attachedBody.
        - Register this collider into the Physics2DWorld.

To Table of Contents

        - RecreateShape() to build the Box2D shape using current material, trigger flags, and geometry.

- void Shutdown()
    - Return:
        - void - Unregisters the collider and destroys its Box2D shape.
    - Steps:
        - Unregister this collider from the Physics2DWorld.
        - If the shape is valid, destroy it in Box2D and reset m_shapeId.

- void RebuildShape()
    - Return:
        - void - Recreates the Box2D shape while keeping all configuration fields.
    - Steps:
        - Call RecreateShape() (clear contact cache, destroy old shape, create new shape).

- bool IsTrigger() const
    - Return:
        - bool - True if the collider is configured as a trigger (sensor).

- void SetTrigger(bool isTrigger)
    - Return:
        - void - Changes trigger mode and recreates the shape so Box2D flags match the new behavior.
    - Parameters:
        - bool isTrigger - True to make this collider a sensor.
    - Steps:
        - If the value did not change, do nothing.
        - Update m_isTrigger.
        - RecreateShape() so the isSensor and event flags are applied.

- bool ShouldSensorEvent() const
    - Return:
        - bool - True if this collider requests trigger callbacks from sensor events.

- void SetShouldSensorEvent(bool shouldSensorEvent)
    - Return:
        - void - Enables or disables routing of sensor events and recreates the shape so Box2D emits the desired events.
    - Parameters:
        - bool shouldSensorEvent - True to allow sensor begin/end

To Table of Contents

        events.
  - Steps:
    - If the value did not change, do nothing.
    - Update m_shouldSensorEvent.
    - RecreateShape() so the sensor event flag is updated.

- void SetOffset(const Vector2f& offset)
  - Return:
    - void - Updates local offset and recreates the shape.
  - Parameters:
    - const Vector2f& offset - New local-space offset.

- Vector2f GetOffset() const
  - Return:
    - Vector2f - Current local-space offset.

- void SetDensity(float density)
  - Return:
    - void - Sets density and recreates the shape so the new value is used by Box2D.
  - Parameters:
    - float density - New density.

- void SetFriction(float friction)
  - Return:
    - void - Sets friction and recreates the shape.
  - Parameters:
    - float friction - New friction.

- void SetRestitution(float restitution)
  - Return:
    - void - Sets restitution and recreates the shape.
  - Parameters:
    - float restitution - New restitution.

- void AttachToRigidbody(Rigidbody2D *body)*
  - Return:
    - void - Attaches the collider to a rigidbody and recreates its shape on that body.
  - Parameters:
    - Rigidbody2D *body - Rigidbody to attach to (non-owning).*
  - Steps:
    - Store body in m_attachedBody.
    - RecreateShape() to attach the Box2D shape to the new body.

To Table of Contents

- void DetachFromRigidbody(Rigidbody2D *body)*
  - Return:
    - void - Detaches from the given rigidbody and destroys the shape.
  - Parameters:
    - Rigidbody2D *body - Rigidbody to detach from.*
  - Steps:
    - If body is not the currently attached body, do nothing.
    - Set m_attachedBody to null.
    - Ask the Physics2DWorld to drop cached contact pairs for this collider.
    - If a shape exists, destroy it in Box2D and reset m_shapeId.

- b2ShapeId GetShapeId() const
  - Return:
    - b2ShapeId - Underlying Box2D shape handle.

- void RecreateShape()
  - Return:
    - void - Internal helper that rebuilds the shape and ensures event routing and cached contact state remain consistent.
  - Steps:
    - Ask Physics2DWorld to drop cached contact pairs for this collider (prevents phantom Stay after rebuild).
    - ResolveBody() to obtain a valid b2BodyId.
    - If the body id is invalid:
      1. If a shape exists, destroy it.
      2. Leave m_shapeId as null and return.
    - Destroy the old shape if it exists.
    - Build a b2ShapeDef using BuildShapeDef().
    - Call the derived CreateShape(bodyId, shapeDef) to create geometry.
    - Store the returned shape id in m_shapeId.
    - Set the Box2D shape user data pointer to this Collider2D so event mapping works.

- b2ShapeDef BuildShapeDef() const
  - Return:
    - b2ShapeDef - A Box2D shape definition built from Collider2D material and event settings.
  - Steps:
    - Copy density, friction, and restitution.
    - Set isSensor based on m_isTrigger.

To Table of Contents

        - Enable contact events when not a trigger.
        - Enable sensor events when a trigger.
        - Copy m_shouldSensorEvent so sensor events are generated when requested.

- b2BodyId ResolveBody() const
    - Return:
        - b2BodyId - The attached rigidbody's Box2D body handle, or b2_nullBodyId if none exists.

BoxCollider2D:

- Files: Collider2D.h/.cpp
- Summary: Collider2D specialization that creates a rectangular box shape and exposes size configuration.

Variables:

- Vector2f m_size - Box size (width, height) in world units.

Methods:

- void SetSize(const Vector2f& size)
    - Return:
        - void - Sets box size and rebuilds the shape.
    - Parameters:
        - const Vector2f& size - New size.

- Vector2f GetSize() const
    - Return:
        - Vector2f - Current box size.

- b2ShapeId CreateShape(b2BodyId bodyId, const b2ShapeDef& shapeDef)
    - Return:
        - b2ShapeId - The created Box2D shape handle.
    - Parameters:
        - b2BodyId bodyId - Body to attach the shape to.
        - const b2ShapeDef& shapeDef - Box2D definition describing material and events.
    - Steps:
        - Compute half-extents from m_size.
        - Create an offset box geometry using the collider offset

To Table of Contents

        - (m_offset).
        - Create and return the Box2D shape using the provided shape definition.

- std::shared_ptr<Component> Clone() const
    - Return:
        - std::shared_ptr<Component> - New BoxCollider2D with copied base Collider2D fields and m_size.

CircleCollider2D:

- Files: Collider2D.h/.cpp
- Summary: Collider2D specialization that creates a circle shape and exposes radius configuration.

Variables:

- float m_radius - Circle radius in world units.

Methods:

- void SetRadius(float radius)
    - Return:
        - void - Sets radius and rebuilds the shape.
    - Parameters:
        - float radius - New radius.

- float GetRadius() const
    - Return:
        - float - Current radius.

- b2ShapeId CreateShape(b2BodyId bodyId, const b2ShapeDef& shapeDef)
    - Return:
        - b2ShapeId - The created Box2D shape handle.
    - Parameters:
        - b2BodyId bodyId - Body to attach the shape to.
        - const b2ShapeDef& shapeDef - Box2D definition describing material and events.
    - Steps:
        - Create a circle geometry with center at the collider offset (m_offset).
        - Set the circle radius to m_radius.

To Table of Contents

        - Create and return the Box2D shape using the provided shape definition.

- std::shared_ptr<Component> Clone() const
    - Return:
        - std::shared_ptr<Component> - New CircleCollider2D with copied base Collider2D fields and m_radius.

Rendering pipeline

- Files: Renderer.h/.cpp, RenderableComponent.h/.cpp, SpriteRenderer.h/.cpp, TextRenderer.h/.cpp, RenderSystem.h/.cpp, RenderQueue.h/.cpp
- Summary: Rendering is centralized around a registry and a per-frame render queue. Renderable components (sprites and text) register into RenderSystem. Each frame RenderSystem filters active + visible renderables into RenderQueue, which sorts deterministically and then issues draw calls through Renderer (an SDL3 wrapper with virtual-resolution viewport support).

Renderer:

Variables:

- void *m_renderer - SDL_Renderer pointer stored as void* (opaque in headers).
- void *m_window - SDL_Window pointer stored as void* so output pixel size can be queried reliably.

- int m_virtualW - Virtual width used for world-to-screen conversion and scaling.
- int m_virtualH - Virtual height used for world-to-screen conversion and scaling.
- ViewportScaleMode m_scaleMode - Virtual resolution mapping mode (Letterbox, Stretch, Crop).
- bool m_integerScale - If true, uniform scaling can be floored to an integer multiplier for pixel art.

- Vector4i m_clearColor - RGBA used to clear the game area inside the viewport.
- Vector4i m_letterboxColor - RGBA used to clear the full window (letterbox bars included).

- mutable bool m_cacheValid - Whether the cached viewport calculations are

To Table of Contents

valid for the current window size.
- mutable int m_cachedOutputW - Cached output pixel width.
- mutable int m_cachedOutputH - Cached output pixel height.
- mutable int m_cachedGameW - Cached virtual game width (typically m_virtualW).
- mutable int m_cachedGameH - Cached virtual game height (typically m_virtualH).
- mutable float m_cachedScaleX - Cached X scale (virtual pixels -> screen pixels).
- mutable float m_cachedScaleY - Cached Y scale (virtual pixels -> screen pixels).
- mutable Vector2f m_cachedOffset - Cached pixel offset (used for Crop and also as general mapping offset).
- mutable Rectf m_cachedViewport - Cached viewport rectangle in output pixels.

- mutable bool m_viewportAppliedThisFrame - Tracks whether SDL viewport/clip state was applied for the current frame, so draw calls can lazily ensure correct state.

Methods:

- Renderer(Window& window)
  - Return:
    - void - Creates the SDL_Renderer for the given window and stores the SDL_Window handle for pixel-size queries.
  - Parameters:
    - Window& window - The engine window wrapper providing the native SDL_Window pointer.
  - Steps:
    - Fetch native SDL_Window from Window.
    - Create SDL_Renderer via SDL_CreateRenderer.
    - Store both handles and log initialization.

- Renderer(Renderer&& other) noexcept
  - Return:
    - void - Move-constructs a renderer and invalidates the source handles.

- Renderer& operator=(Renderer&& other) noexcept
  - Return:
    - Renderer& - Move-assigns and destroys the existing SDL_Renderer if needed.

To Table of Contents

- ~Renderer()
    - Return:
        - void - Destroys the SDL_Renderer if it exists.

- void SetClearColor(const Vector4i& rgba)
    - Return:
        - void - Sets the game-area clear color.
    - Parameters:
        - const Vector4i& rgba - RGBA values (0-255).

- void SetLetterboxColor(const Vector4i& rgba)
    - Return:
        - void - Sets the letterbox clear color.
    - Parameters:
        - const Vector4i& rgba - RGBA values (0-255).

- Vector4i GetClearColor() const
    - Return:
        - Vector4i - Current clear color.

- void Clear()
    - Return:
        - void - Clears the full window to letterboxColor, applies the viewport, then clears the game area to clearColor.
    - Steps:
        - Reset SDL viewport/clip to full window.
        - Clear using letterbox color.
        - Apply viewport/clip (virtual-resolution mapping).
        - Clear the game region using clear color.

- void BeginFrame()
    - Return:
        - void - Alias for Clear().

- void Present()
    - Return:
        - void - Presents the SDL backbuffer and resets per-frame viewport tracking.

- void EndFrame()
    - Return:
        - void - Alias for Present().

To Table of Contents

- void *GetNative() const*
    - Return:
        - void - *Native SDL_Renderer pointer.*

- void SetVirtualResolution(int width, int height)
    - Return:
        - void - Sets the virtual resolution and invalidates cached viewport calculations.
    - Parameters:
        - int width - Virtual width in pixels.
        - int height - Virtual height in pixels.

- void SetVirtualResolution(const Vector2i& size)
    - Return:
        - void - Convenience overload for SetVirtualResolution.
    - Parameters:
        - const Vector2i& size - Virtual resolution.

- Vector2i GetVirtualResolution() const
    - Return:
        - Vector2i - Current virtual resolution.

- void SetLetterbox(bool enabled)
    - Return:
        - void - Convenience switch between Letterbox and Crop modes.
    - Parameters:
        - bool enabled - True uses Letterbox, false uses Crop.

- void SetViewportScaleMode(ViewportScaleMode mode)
    - Return:
        - void - Sets the viewport scale mode and invalidates cached viewport calculations.
    - Parameters:
        - ViewportScaleMode mode - Letterbox, Stretch, or Crop.

- ViewportScaleMode GetViewportScaleMode() const
    - Return:
        - ViewportScaleMode - Current viewport scaling mode.

- void SetIntegerScaling(bool enabled)
    - Return:
        - void - Toggles integer scaling and invalidates cached viewport calculations.
    - Parameters:

        - bool enabled - True enables integer scaling.

- bool IsIntegerScaling() const
    - Return:
        - bool - True if integer scaling is enabled.

- Rectf GetViewportRect() const
    - Return:
        - Rectf - Pixel rectangle of the active viewport in the output window.
    - Steps:
        - Ensure viewport cache is up to date and return the cached viewport.

- bool DrawTexture(const Texture& texture, const Vector2f& srcPos, const Vector2f& srcSize, const Vector2f& dstPos, const Vector2f& dstSize)
    - Return:
        - bool - True if the draw succeeds.
    - Parameters:
        - const Texture& texture - Source texture.
        - const Vector2f& srcPos - Source top-left in texture pixels.
        - const Vector2f& srcSize - Source size in texture pixels.
        - const Vector2f& dstPos - Destination top-left in world coordinates.
        - const Vector2f& dstSize - Destination size in world units (virtual pixels).
    - Steps:
        - Ensure viewport/clip is applied for this frame.
        - Convert world destination into screen pixels using the viewport mapping.
        - Call SDL_RenderTexture with src and dst rects.

- bool DrawTextureTinted(const Texture& texture, const Vector2f& srcPos, const Vector2f& srcSize, const Vector2f& dstPos, const Vector2f& dstSize, const Vector4i& tint)
    - Return:
        - bool - True if the draw succeeds.
    - Parameters:
        - const Vector4i& tint - RGBA tint to apply (0-255).
    - Steps:
        - Save old modulation on the SDL texture.
        - Apply color + alpha mods.
        - Render, then restore previous modulation.

To Table of Contents

- bool DrawTextureRotated(const Texture& texture, const Vector2f& srcPos, const Vector2f& srcSize, const Vector2f& dstPos, const Vector2f& dstSize, float angleDegrees, const Vector2f& pivot, FlipMode flip)
    - Return:
        - bool - True if the draw succeeds.
    - Parameters:
        - float angleDegrees - Rotation in degrees (world-space CCW).
        - const Vector2f& pivot - Pivot in destination local space; negative values mean "use center".
        - FlipMode flip - Optional horizontal/vertical flip.
    - Steps:
        - Convert world destination to screen pixels.
        - Convert pivot to screen-space pivot (scaled).
        - Negate angle for SDL so world CCW stays correct under Y-down screen coordinates.
        - Call SDL_RenderTextureRotated.

- bool DrawRectOutline(const Vector2f& worldTopLeft, const Vector2f& size, const Vector3i& color)
    - Return:
        - bool - True if the draw succeeds.

- bool DrawRectOutlineRotated(const Vector2f& worldCenter, const Vector2f& size, float angleDegrees, const Vector3i& color)
    - Return:
        - bool - True if the draw succeeds.

- bool DrawFilledRect(const Vector2f& worldTopLeft, const Vector2f& size, const Vector4i& color)
    - Return:
        - bool - True if the draw succeeds.

- bool DrawCircleOutline(const Vector2f& worldCenter, float radius, const Vector3i& color, int segments)
    - Return:
        - bool - True if the draw succeeds.

- bool DrawPoint(const Vector2f& worldPoint, const Vector3i& color)
    - Return:
        - bool - True if the draw succeeds.

- bool DrawLine(const Vector2f& start, const Vector2f& end, const Vector3i& color)
    - Return:

To Table of Contents

        - bool - True if the draw succeeds.

- bool DrawLineThickness(const Vector2f& start, const Vector2f& end, float thickness, const Vector3i& color)
    - Return:
        - bool - True if the draw succeeds.

RenderableComponent:

- Files: RenderableComponent.h/.cpp
- Summary: Base Component for anything that submits draw work. Adds a visible flag and ensures proper unregistration from RenderSystem when destroyed.

Variables:

- bool m_isVisible - If false, the component is ignored by RenderSystem::BuildQueue.

Methods:

- RenderableComponent(const std::string& componentName)
    - Return:
        - void - Constructs the component and sets a default visible state.

- void SetVisible(bool v)
    - Return:
        - void - Sets m_isVisible.
    - Parameters:
        - bool v - New visibility state.

- bool IsVisible() const
    - Return:
        - bool - Returns current visibility state.

- void DestroyImmediateInternal() override
    - Return:
        - void - Unregisters from RenderSystem before the GameObject releases the component.

To Table of Contents

SpriteRenderer:

- Files: SpriteRenderer.h/.cpp
- Summary: Draws a frame from a texture (single image or spritesheet). Uses Transform world position/rotation/scale; negative scale mirrors the sprite through FlipMode.

Variables:

- Texture *m_texture - Non-owning pointer to the texture to render.*
- Vector2i m_frameSize - Frame size for spritesheets. If invalid, falls back to full texture size.
- int m_frameIndex - Index of the frame to render (clamped to valid range).
- int m_layerOrder - Per-layer ordering inside a GameObject layer.

- static SortOptions g_sortOptions - Legacy sort options used by SpriteRenderer::RenderAll.

Methods:

- SpriteRenderer()
    - Return:
        - void - Creates an empty sprite renderer with no texture.

- SpriteRenderer(Texture *texture)*
    - Return:
        - void - Creates a sprite renderer and optionally initializes frameSize from the texture.
    - Parameters:
        - Texture *texture - Texture to assign (non-owning).*

- void SetTexture(Texture *texture)*
    - Return:
        - void - Sets the texture and resolves frame size if needed.
    - Parameters:
        - Texture *texture - New texture (non-owning).*

- void SetFrameSize(const Vector2i& size)
    - Return:
        - void - Sets the frame size used for spritesheet indexing.
    - Parameters:
        - const Vector2i& size - Frame size in pixels.

To Table of Contents

- void SetFrameIndex(int index)
    - Return:
        - void - Sets which frame index to display.
    - Parameters:
        - int index - Frame index (will be clamped at render time).

- void SetLayerOrder(int order)
    - Return:
        - void - Sets per-layer draw order.
    - Parameters:
        - int order - Sub-order within the GameObject layer.

- int GetLayerOrder() const
    - Return:
        - int - Current layer order.

- static void SetSortOptions(const SortOptions& options)
    - Return:
        - void - Sets the legacy global sort options for SpriteRenderer::RenderAll.
    - Parameters:
        - const SortOptions& options - Primary/secondary axis options and direction flags.

- static SortOptions GetSortOptions()
    - Return:
        - SortOptions - Returns the current legacy sort options.

- Vector2i GetResolvedFrameSize() const
    - Return:
        - Vector2i - Returns m_frameSize if valid, otherwise uses the texture size.

- void Render(Renderer& renderer) const
    - Return:
        - void - Renders the sprite using the current Transform and frame settings.
    - Parameters:
        - Renderer& renderer - The draw backend.
    - Steps:
        - Validate texture and transform.
        - Resolve frameSize and compute maxFrames from the texture dimensions.
        - Clamp frameIndex, compute (row, col) inside the spritesheet

To Table of Contents

grid.
        - Compute src rect in texture pixels.
        - Read world position (center), rotation (degrees), and scale.
        - Convert negative scale to FlipMode and use absolute scale for sizing.
        - Convert center-based world position into world top-left (x - w/2, y + h/2).
        - If rotation or flip is needed, call Renderer::DrawTextureRotated; otherwise call Renderer::DrawTexture.

- static void RenderAll(Renderer& renderer)
    - Return:
        - void - Legacy path: scans Object registry for SpriteRenderer, sorts them, and renders them.
    - Notes:
        - The preferred path in this engine is RenderSystem -> RenderQueue -> Renderer, which avoids scanning all objects every frame.

- std::shared_ptr<Component> Clone() const override
    - Return:
        - std::shared_ptr<Component> - A new SpriteRenderer with copied texture/frame/order settings.

TextRenderer:

- Files: TextRenderer.h/.cpp
- Summary: Draws bitmap text in world space using BitmapFont. Supports anchoring, rotation, and mirroring (negative scale), and uses per-glyph drawing.

Variables:

- BitmapFont *m_font - Non-owning pointer to the bitmap font.*
- std::string m_text - The text string to render.
- TextAnchor m_anchor - Alignment rule (TopLeft or Center) relative to Transform world position.
- float m_extraScale - Extra multiplier applied on top of Transform scale.
- int m_layerOrder - Per-layer ordering inside a GameObject layer.

To Table of Contents

Methods:

- TextRenderer()
  - Return:
    - void - Creates a text renderer with default settings.

- void SetFont(BitmapFont *font)*
  - Return:
    - void - Assigns the font used for glyph rendering.
  - Parameters:
    - BitmapFont *font - Font asset pointer (non-owning).*

- void SetText(const std::string& text)
  - Return:
    - void - Sets the string to render.
  - Parameters:
    - const std::string& text - New text contents.

- void SetAnchor(TextAnchor a)
  - Return:
    - void - Sets the anchor mode.
  - Parameters:
    - TextAnchor a - TopLeft or Center.

- void SetExtraScale(float s)
  - Return:
    - void - Sets the extra scale multiplier.
  - Parameters:
    - float s - Multiplier applied on top of Transform scale.

- void SetLayerOrder(int order)
  - Return:
    - void - Sets per-layer draw order.
  - Parameters:
    - int order - Sub-order within the GameObject layer.

- int GetLayerOrder() const
  - Return:
    - int - Current layer order.

- void Render(Renderer& renderer) const
  - Return:
    - void - Renders the text string using the current Transform (position/rotation/scale).

To Table of Contents

- Steps:
  - Validate font and texture.
  - Read Transform world position as the anchor point.
  - Build final scale = Transform world scale *extraScale.*
  - If there is no rotation and no mirroring, measure the text and draw the full string through BitmapFont::Draw (fast path).
  - Otherwise:
    - Determine FlipMode from sign of the scale.
    - Measure the full text block size.
    - Compute the block's top-left relative to the anchor based on the chosen TextAnchor.
    - Iterate characters, compute each glyph's local center, apply mirroring, rotate around the anchor, convert to world top-left, and draw each glyph via Renderer::DrawTextureRotated.

- static void RenderAll(Renderer& renderer)
  - Return:
    - void - Legacy path: scans Object registry for TextRenderer, sorts them, and renders them.

- std::shared_ptr<Component> Clone() const override
  - Return:
    - std::shared_ptr<Component> - A new TextRenderer with copied font/text/anchor/scale/order settings.

RenderSystem:

- Files: RenderSystem.h/.cpp
- Summary: Keeps stable registries of active renderables (raw pointers). Registration happens when RenderableComponents are created/destroyed. Per frame, BuildQueue filters the registry by visibility and activation and pushes items into a RenderQueue.

Variables:

- std::vector<SpriteRenderer> *m_sprites - Registry of sprite renderers.*
- std::vector<TextRenderer> *m_texts - Registry of text renderers.*

Methods:

- static RenderSystem& Get()

To Table of Contents

    - Return:
        - RenderSystem& - Singleton access for the render registry.

- void Register(RenderableComponent *renderable)*
    - Return:
        - void - Adds the renderable to the correct registry based on its dynamic type.
    - Parameters:
        - RenderableComponent *renderable - Renderable component to register.*
    - Steps:
        - If renderable is a SpriteRenderer, push into m_sprites if not already present.
        - If renderable is a TextRenderer, push into m_texts if not already present.

- void Unregister(RenderableComponent *renderable)*
    - Return:
        - void - Removes the renderable from its registry.
    - Parameters:
        - RenderableComponent *renderable - Renderable component to unregister.*
    - Steps:
        - Swap-remove the pointer from the matching vector.

- void BuildQueue(RenderQueue& outQueue) const
    - Return:
        - void - Filters registry entries and adds them to the render queue.
    - Parameters:
        - RenderQueue& outQueue - Queue to append items into.
    - Steps:
        - For each registered SpriteRenderer:
            1. Skip null, invisible, or inactive-in-hierarchy owners.
            2. outQueue.Add(sprite)
        - Repeat for TextRenderer.

- void Clear()
    - Return:
        - void - Clears both registries (used on scene unload).

RenderQueue:

- Files: RenderQueue.h/.cpp

To Table of Contents

- Summary: Holds per-frame draw items and enforces deterministic ordering. Items are bucketed by (layer, layerOrder). Within each bucket, optional sprite axis sorting can be applied, followed by stable fallbacks (componentIndex then instanceId).

Variables:

- std::vector<Item> m_items - Flat list of queued render items for this frame.

- static SortOptions g_spriteSortOptions - Global sprite axis sort options used by RenderQueue.

- struct Item
    - ItemType type - Sprite or Text.
    - const void *ptr - Pointer to the renderable component (SpriteRenderer* or TextRenderer*).*
    - int layer - GameObject layer.
    - int layerOrder - Per-component order inside the layer.
    - float primaryAxis - Cached axis value (X or Y) for sprite tie-break sorting.
    - float secondaryAxis - Cached axis value (X or Y) for sprite tie-break sorting.
    - size_t componentIndex - Component index on the GameObject (stable fallback).
    - uint32_t instanceId - Object instance id (final deterministic fallback).

Methods:

- void Clear()
    - Return:
        - void - Clears the queue item list.

- static void SetSpriteSortOptions(const SortOptions& options)
    - Return:
        - void - Sets global sprite axis sorting preferences.
    - Parameters:
        - const SortOptions& options - Primary/secondary axis selections and ascending flags.

- static SortOptions GetSpriteSortOptions()
    - Return:
        - SortOptions - Returns current global sprite axis sorting preferences.

To Table of Contents

- void Add(const SpriteRenderer *sprite)*
    - Return:
        - void - Adds a sprite item into the queue with cached sort keys.
    - Parameters:
        - const SpriteRenderer *sprite - Sprite renderer to add.*
    - Steps:
        - Read GameObject layer and SpriteRenderer layerOrder.
        - Cache axis values from the sprite's Transform world position (if enabled).
        - Cache componentIndex and instanceId for deterministic fallbacks.
        - Push the item into m_items.

- void Add(const TextRenderer *text)*
    - Return:
        - void - Adds a text item into the queue with cached sort keys.
    - Parameters:
        - const TextRenderer *text* - *Text renderer to add.*
    - Steps:
        - Read GameObject layer and TextRenderer layerOrder.
        - Cache componentIndex and instanceId for deterministic fallbacks.
        - Push the item into m_items.

- void Execute(Renderer& renderer) const
    - Return:
        - void - Sorts and renders all queued items in deterministic order.
    - Parameters:
        - Renderer& renderer - The backend used to draw.
    - Steps:
        - Bucket items by (layer, layerOrder) using a packed 64-bit key.
        - Sort buckets by layer, then layerOrder.
        - For each bucket:
            - If bucket has more than one item:
                a. If both items are sprites and axis sorting is enabled:
                    i. Compare primaryAxis, then secondaryAxis.
                b. Fall back to componentIndex, then instanceId.
            - Call SpriteRenderer::Render or TextRenderer::Render for each item.

To Table of Contents

Animation system

- Files:
  - GameEngine/AnimationClip.h
  - GameEngine/AnimatorController.h
  - GameEngine/Animator.h
  - GameEngine/Animator.cpp

- Sumary: This system provides sprite-based animation using clips and a small controller-driven state machine. Animation is evaluated by the Animator component each frame and applied by updating the attached SpriteRenderer (texture, frame size, and frame index).

AnimationClip:

- Files:
  - GameEngine/AnimationClip.h

Sprite-only animation clip. A clip is a sequence of spritesheet frame indices played at a fixed FPS, with optional looping and optional frame events.

Variables:

- struct Event - Defines a named event that triggers when playback enters a specific local frame.
  - int localFrameIndex - Local clip frame index (0..frames.size-1) that triggers this event.
  - std::string name - Event identifier (used by gameplay code to react to animation timing).

- std::string name - Clip identifier (debug and tooling use).
- SpriteSheet *sheet - Spritesheet source used by this clip (texture + frame size).*
- std::vector<int> frames - Spritesheet frame indices (row-major) in playback order.
- float fps - Playback rate in frames per second.
- bool loop - If true, playback wraps around; if false, playback clamps to the last

To Table of Contents

frame.

- std::vector<Event> events - Optional list of frame events to fire as playback enters local frames.

Methods:

- bool IsValid() const
    - Parameters:
        - None
    - Return:
        - bool - True when the clip can be sampled (sheet exists, sheet is valid, frames are present, fps > 0).
    - Description:
        - Validates that the clip references a usable SpriteSheet and has a non-empty frame sequence.

- float GetLengthSeconds() const
    - Parameters:
        - None
    - Return:
        - float - Duration of the clip in seconds (frames.size / fps), or 0 if invalid.
    - Description:
        - Computes the clip duration from the number of frames and the FPS.

- int SampleFrameIndex(float t) const
    - Parameters:
        - float t - Time in seconds since the clip started.
    - Return:
        - int - The spritesheet frame index to display at time t, or -1 if invalid.
    - Description:
        - Converts time to a local clip frame using SampleLocalFrame, then maps that local frame to the actual spritesheet frame index stored in frames.

- int SampleLocalFrame(float t) const
    - Parameters:
        - float t - Time in seconds since the clip started.
    - Return:
        - int - Local frame cursor (0..frames.size-1), or -1 if invalid.

To Table of Contents

  - Description:
    - Converts time into a frame cursor using floor(t *fps).*
    - If loop is enabled, wraps the cursor into range using modulo.
    - If loop is disabled, clamps the cursor into range so the animation freezes on the first/last frame when out of bounds.

- float GetNormalizedTime(float t) const
  - Parameters:
    - float t - Time in seconds since the clip started.
  - Return:
    - float - Normalized progress in the range 0..1.
  - Description:
    - For looping clips, returns the fractional progress within the current loop.
    - For non-looping clips, clamps normalized progress into 0..1.

AnimatorController:

- Files:
  - GameEngine/AnimatorController.h

Immutable controller asset that defines animation parameters, states, and transitions. The Animator reads this graph at runtime to decide which clip to play and when to switch.

Types:

- enum class AnimParamType - Supported parameter storage types.
  - Float - Continuous numeric value.
  - Int - Integer value.
  - Bool - Boolean value.
  - Trigger - One-frame boolean pulse (consumed after evaluation).

- struct AnimParamDef - Parameter definition with defaults.
  - std::string name - Unique parameter name.
  - AnimParamType type - Parameter type.
  - float defaultFloat - Default value when type is Float.
  - int defaultInt - Default value when type is Int.
  - bool defaultBool - Default value when type is Bool.

To Table of Contents

- enum class AnimCondOp - Condition operator used by transition conditions.
  - BoolTrue - Parameter bool must be true.
  - BoolFalse - Parameter bool must be false.
  - FloatGreater - Float parameter must be greater than a threshold.
  - FloatLess - Float parameter must be less than a threshold.
  - FloatGreaterEq - Float parameter must be greater than or equal to a threshold.
  - FloatLessEq - Float parameter must be less than or equal to a threshold.
  - IntEquals - Int parameter must equal a value.
  - IntNotEquals - Int parameter must not equal a value.
  - TriggerSet - Trigger parameter must be set (true) for the transition to pass.

- struct AnimCondition - A single condition used on a transition.
  - std::string param - Name of the parameter to check.
  - AnimCondOp op - Operator used for the check.
  - float f - Float threshold used by float operators.
  - int i - Int value used by int operators.

- struct AnimTransition - Directed edge between two states.
  - int fromState - Source state id, or -1 for Any State transitions.
  - int toState - Target state id.
  - bool hasExitTime - If true, transition is only valid after reaching exitTimeNormalized.
  - float exitTimeNormalized - Normalized exit threshold (0..1) of the source clip.
  - std::vector<AnimCondition> conditions - All conditions that must pass for the transition.

- struct AnimState - Animation state entry.
  - int id - Unique state id.
  - std::string name - State name (used by Play and debugging).
  - AnimationClip *clip - Clip played while this state is active.*

Variables:

- std::vector<AnimParamDef> parameters - Parameter definitions and default values.
- std::vector<AnimState> states - State list (each state references an AnimationClip).
- std::vector<AnimTransition> transitions - Transition list (Any State and

To Table of Contents

per-state transitions).
- int entryState - State id to enter when a controller is assigned.

Methods:

- const AnimState *FindStateById(int id) const*
  - Parameters:
    - int id - State id to search for.
  - Return:
    - const AnimState - *Pointer to the matching state, or nullptr if not found.*
  - Description:
    - Iterates the states list and returns the first state with a matching id.

- int FindStateIdByName(const std::string& n) const
  - Parameters:
    - const std::string& n - State name to search for.
  - Return:
    - int - Matching state id, or -1 if not found.
  - Description:
    - Iterates the states list and returns the id of the first state with a matching name.

Animator:

- Files:
  - GameEngine/Animator.h
  - GameEngine/Animator.cpp

Runtime component that plays AnimationClips based on an AnimatorController. Each frame, it advances state time (unless SeekNormalized overrides it), evaluates transitions, and applies the current clip frame to the cached SpriteRenderer.

Variables:

- AnimatorController *m_controller - Active controller graph asset (not owned).*
- std::unordered_map<std::string, float> m_floats - Runtime float parameter values.

To Table of Contents

- std::unordered_map<std::string, int> m_ints - Runtime int parameter values.
- std::unordered_map<std::string, bool> m_bools - Runtime bool parameter values.
- std::unordered_map<std::string, bool> m_triggers - Runtime trigger parameter values (consumed after evaluation).

- int m_stateId - Current state id (matches a state in the controller).
- float m_stateTime - Time spent in the current state (seconds).

- SpriteRenderer *m_sprite - Cached SpriteRenderer on the same GameObject (not owned).*
- Texture *m_lastTexture - Last texture applied to the SpriteRenderer (used to detect changes).*
- Vector2i m_lastFrameSize - Last frame size applied to the SpriteRenderer (used to detect changes).

- bool m_timeOverriddenThisFrame - True when SeekNormalized was called this frame (prevents normal time advancement).
- int m_prevLocalFrame - Previously sampled local frame (used to avoid redundant work and to gate frame events).

Methods:

- Animator()
  - Parameters:
    - None
  - Return:
    - void - Constructs the component and sets its component name to Animator.
  - Description:
    - Initializes the base MonoBehaviour and labels the component for debugging.

- void Awake() override
  - Parameters:
    - None
  - Return:
    - void - Caches a SpriteRenderer pointer if one exists on the same GameObject.
  - Description:
    - Attempts to find a SpriteRenderer component and stores a raw pointer for fast per-frame updates.

- void Update() override
    - Parameters:
        - None
    - Return:
        - void - Advances animation state, evaluates transitions, applies the current clip frame, and clears triggers.
    - Description:
        - If no controller is assigned, clears triggers and exits.
        - Re-caches the SpriteRenderer if needed.
        - Advances m_stateTime by Time::DeltaTime unless SeekNormalized overrode time this frame.
        - Checks Any State transitions first, then transitions from the current state.
        - Applies the current clip's frame to the SpriteRenderer (texture, frame size, frame index).
        - Clears all triggers and resets the time override flag for the next frame.

- void SetController(AnimatorController *controller)*
    - Parameters:
        - AnimatorController *controller - Controller graph asset to use.*
    - Return:
        - void - Assigns the controller, ensures parameter defaults, and enters the entry state if defined.
    - Description:
        - Stores the controller pointer.
        - Ensures missing runtime parameters are initialized from controller defaults.
        - Switches to entryState (if valid), restarting time.

- AnimatorController *GetController() const*
    - Parameters:
        - None
    - Return:
        - AnimatorController - *The currently assigned controller pointer.*

- void SetFloat(const std::string& name, float v)
    - Parameters:
        - const std::string& name - Parameter name.
        - float v - New value.
    - Return:
        - void - Sets a float parameter value in the runtime map.

- void SetInt(const std::string& name, int v)
  - Parameters:
    - const std::string& name - Parameter name.
    - int v - New value.
  - Return:
    - void - Sets an int parameter value in the runtime map.

- void SetBool(const std::string& name, bool v)
  - Parameters:
    - const std::string& name - Parameter name.
    - bool v - New value.
  - Return:
    - void - Sets a bool parameter value in the runtime map.

- void SetTrigger(const std::string& name)
  - Parameters:
    - const std::string& name - Parameter name.
  - Return:
    - void - Sets a trigger parameter to true for evaluation during this frame.

- float GetFloat(const std::string& name) const
  - Parameters:
    - const std::string& name - Parameter name.
  - Return:
    - float - Parameter value if present, otherwise 0.

- int GetInt(const std::string& name) const
  - Parameters:
    - const std::string& name - Parameter name.
  - Return:
    - int - Parameter value if present, otherwise 0.

- bool GetBool(const std::string& name) const
  - Parameters:
    - const std::string& name - Parameter name.
  - Return:
    - bool - Parameter value if present, otherwise false.

- bool GetTrigger(const std::string& name) const
  - Parameters:
    - const std::string& name - Parameter name.
  - Return:
    - bool - True if the trigger exists and is currently set, otherwise

To Table of Contents

false.

- void Play(const std::string& stateName, bool restart = true)
    - Parameters:
        - const std::string& stateName - Target state name.
        - bool restart - If true, resets state time and frame tracking when switching.
    - Return:
        - void - Forces a state switch by name, ignoring transition rules.
    - Description:
        - Uses the controller to look up a state id by name.
        - Switches to that state if found.

- std::string GetCurrentStateName() const
    - Parameters:
        - None
    - Return:
        - std::string - Name of the current state, or an empty string if unknown.

- float GetStateTime() const
    - Parameters:
        - None
    - Return:
        - float - Time spent in the current state (seconds).

- void SeekNormalized(float targetN, float speedNormalizedPerSec)
    - Parameters:
        - float targetN - Target normalized position in the current clip (0..1).
        - float speedNormalizedPerSec - How fast to move toward the target (normalized units per second).
    - Return:
        - void - Adjusts state time toward a target normalized time without an abrupt jump.
    - Description:
        - Marks the frame as time-overridden so the normal time increment is skipped.
        - Clamps targetN to 0..1 and computes current normalized time.
        - Moves the normalized cursor toward targetN using Time::DeltaTime and the provided speed.
        - Converts the new normalized time back to seconds and stores it in m_stateTime.

To Table of Contents

- std::shared_ptr<Component> Clone() const override
    - Parameters:
        - None
    - Return:
        - std::shared_ptr<Component> - New Animator component with copied controller, parameters, and current state/time.
    - Description:
        - Creates a new Animator and copies controller pointer, parameter maps, state id, and state time.

Notes:

- Transition evaluation order:
    - Any State transitions are evaluated first (fromState == -1).
    - If none match, transitions from the current state are evaluated.
    - The first valid transition is taken, which keeps evaluation deterministic.

- Trigger handling:
    - Transitions can require a trigger to be set (TriggerSet condition).
    - Triggers used by a taken transition are immediately cleared.
    - At the end of Update, all triggers are cleared so triggers behave as one-frame pulses by default.

- Applying frames to SpriteRenderer:
    - The Animator ensures the SpriteRenderer uses the spritesheet referenced by the active clip.
    - Texture and frame size are only re-applied when they change (cached by m_lastTexture and m_lastFrameSize).
    - Frame index is sampled from the clip at m_stateTime and written into SpriteRenderer.

Assets and resource pipeline

Files:
- GameEngine/AssetManager.h
- GameEngine/AssetManager.cpp
- GameEngine/Surface.h
- GameEngine/Surface.cpp
- GameEngine/Texture.h

To Table of Contents

- GameEngine/Texture.cpp
- GameEngine/SpriteSheet.h
- GameEngine/AudioClip.h
- GameEngine/BitmapFont.h
- GameEngine/BitmapFont.cpp

Purpose: Loads game resources from disk, converts them into runtime-friendly objects (GPU textures, sprite sheets, bitmap fonts, audio buffers), and caches them so the same file is not loaded multiple times. This keeps I/O and resource lifetime management out of gameplay code and out of the rendering loop.

AssetManager:

Purpose: Central cache and factory for engine assets. Creates textures using the active Renderer, stores them in maps keyed by path (and optional color key), and returns stable pointers for reuse.

Variables:

- Renderer& m_renderer - Renderer reference used to create GPU textures from surfaces.
- std::string m_basePath - Prefix folder applied to relative paths when loading from disk.
- TextureScaleMode m_defaultTextureScaleMode - Default texture filtering (Nearest or Linear) used when no override is provided.
- std::unordered_map<std::string, std::unique_ptr<Texture>> m_textures - Cache of textures keyed by a computed string key (path plus optional color-key suffix).
- std::unordered_map<std::string, SpriteSheet> m_spriteSheets - Cache of sprite sheets keyed by sheet key (stores Texture *plus frame size).*
- std::unordered_map<std::string, std::unique_ptr<BitmapFont>> m_fonts - Cache of bitmap fonts keyed by font key (often equal to the texture path).
- std::unordered_map<std::string, std::unique_ptr<AudioClip>> m_audioClips - Cache of decoded WAV audio clips keyed by relative path.

Methods:

- AssetManager(Renderer& renderer)
    - Return:

        - AssetManager - Constructed manager instance.
    - Parameters:
        - Renderer& renderer - Renderer used for creating textures.
    - Description:
        - Initializes the manager and sets default base path and texture scale mode.

- void SetBasePath(const std::string& basePath)
    - Return:
        - void - No return value.
    - Parameters:
        - const std::string& basePath - New base path prefix (for example "Assets/").
    - Description:
        - Sets the base path used to build full file paths for all subsequent loads.

- void SetDefaultTextureScaleMode(TextureScaleMode mode)
    - Return:
        - void - No return value.
    - Parameters:
        - TextureScaleMode mode - Default filtering mode applied to newly loaded textures when no override is passed.
    - Description:
        - Controls whether textures default to nearest-neighbour or linear filtering.

- TextureScaleMode GetDefaultTextureScaleMode() const
    - Return:
        - TextureScaleMode - The manager's current default texture scale mode.
    - Parameters:
        - None
    - Description:
        - Retrieves the default filtering configuration.

- Texture *LoadTexture(const std::string& relativePath)*
    - Return:
        - Texture - *Cached or newly loaded texture pointer (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Path relative to base path (for example "Ship2.bmp").
    - Description:

To Table of Contents

        - Loads a BMP as a Texture using default settings, or returns the cached texture if already loaded.
    - Steps:
        - Build a cache key from relativePath (no color key).
        - If cached, return it.
        - Load the BMP and create an SDL_Texture through Texture's constructor.
        - Apply default scale mode.
        - Store in m_textures and return the stored pointer.

- Texture *LoadTexture(const std::string& relativePath, const Vector3i& colorKey)*
    - Return:
        - Texture - *Cached or newly loaded texture pointer (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Path relative to base path.
        - const Vector3i& colorKey - RGB key treated as transparent when converting Surface → Texture.
    - Description:
        - Same as LoadTexture(path), but applies a color key during loading to remove a background color.

- Texture *LoadTexture(const std::string& relativePath, TextureScaleMode textureScaleModeOverride)*
    - Return:
        - Texture - *Cached or newly loaded texture pointer (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Path relative to base path.
        - TextureScaleMode textureScaleModeOverride - Filtering mode to apply for this texture.
    - Description:
        - Loads or returns a cached texture, and forces the cached texture to use the requested scale mode.

- Texture *LoadTexture(const std::string& relativePath, const Vector3i& colorKey, TextureScaleMode textureScaleModeOverride)*
    - Return:
        - Texture - *Cached or newly loaded texture pointer (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Path relative to base path.
        - const Vector3i& colorKey - RGB transparency key.

To Table of Contents

        - TextureScaleMode textureScaleModeOverride - Filtering override for this texture.
    - Description:
        - Loads with color key and applies the filtering override.

- Texture *GetTexture(const std::string& relativePath) const*
    - Return:
        - Texture - *Cached texture pointer or nullptr if not loaded.*
    - Parameters:
        - const std::string& relativePath - Cache lookup key used at load time.
    - Description:
        - Returns a texture from the cache without loading.

- bool IsTextureLoaded(const std::string& relativePath) const
    - Return:
        - bool - True if a texture exists in the cache for the given key.
    - Parameters:
        - const std::string& relativePath - Cache key.
    - Description:
        - Checks whether a texture is present in the cache.

- void UnloadTexture(const std::string& relativePath)
    - Return:
        - void - No return value.
    - Parameters:
        - const std::string& relativePath - Cache key.
    - Description:
        - Removes a texture from the cache, releasing the GPU resource via RAII.

- void UnloadAllTextures()
    - Return:
        - void - No return value.
    - Parameters:
        - None
    - Description:
        - Clears the entire texture cache.

- SpriteSheet *LoadSpriteSheet(const std::string& textureRelativePath, const Vector2i& frameSize)*
    - Return:
        - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*

To Table of Contents

    - Parameters:
        - const std::string& textureRelativePath - Texture path relative to base path.
        - const Vector2i& frameSize - Per-frame size in pixels.
    - Description:
        - Loads the texture (using default settings), creates a SpriteSheet record, and caches it using a generated key.

- SpriteSheet *LoadSpriteSheet(const std::string& textureRelativePath, const Vector2i& frameSize, const Vector3i& colorKey)*
    - Return:
        - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*
    - Parameters:
        - const std::string& textureRelativePath - Texture path relative to base path.
        - const Vector2i& frameSize - Per-frame size in pixels.
        - const Vector3i& colorKey - Color key applied when loading the sheet texture.
    - Description:
        - Same as LoadSpriteSheet(path, frameSize) but loads the underlying texture with a color key.

- SpriteSheet *LoadSpriteSheet(const std::string& textureRelativePath, const Vector2i& frameSize, TextureScaleMode textureScaleModeOverride)*
    - Return:
        - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*
    - Parameters:
        - const std::string& textureRelativePath - Texture path relative to base path.
        - const Vector2i& frameSize - Per-frame size in pixels.
        - TextureScaleMode textureScaleModeOverride - Filtering override for the underlying texture.
    - Description:
        - Same as LoadSpriteSheet(path, frameSize) but forces the sheet texture scale mode.

- SpriteSheet *LoadSpriteSheet(const std::string& textureRelativePath, const Vector2i& frameSize, const Vector3i& colorKey, TextureScaleMode textureScaleModeOverride)*
    - Return:
        - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*

To Table of Contents

  - Parameters:
    - const std::string& textureRelativePath - Texture path relative to base path.
    - const Vector2i& frameSize - Per-frame size in pixels.
    - const Vector3i& colorKey - Color key applied to the underlying surface.
    - TextureScaleMode textureScaleModeOverride - Filtering override.
  - Description:
    - Loads a sprite sheet with both transparency and filtering override applied.

- SpriteSheet *LoadSpriteSheet(const std::string& sheetKey, const std::string& textureRelativePath, const Vector2i& frameSize)*
  - Return:
    - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*
  - Parameters:
    - const std::string& sheetKey - Explicit cache key for the sheet.
    - const std::string& textureRelativePath - Texture path relative to base path.
    - const Vector2i& frameSize - Per-frame size in pixels.
  - Description:
    - Same as auto-keyed LoadSpriteSheet, but the caller supplies the cache key (useful when many sheets share file names).

- SpriteSheet *LoadSpriteSheet(const std::string& sheetKey, const std::string& textureRelativePath, const Vector2i& frameSize, const Vector3i& colorKey)*
  - Return:
    - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*
  - Parameters:
    - const std::string& sheetKey - Explicit cache key.
    - const std::string& textureRelativePath - Texture path relative to base path.
    - const Vector2i& frameSize - Per-frame size.
    - const Vector3i& colorKey - Color key for transparency.
  - Description:
    - Explicit-key variant with transparency.

- SpriteSheet *LoadSpriteSheet(const std::string& sheetKey, const std::string& textureRelativePath, const Vector2i& frameSize, TextureScaleMode textureScaleModeOverride)*
  - Return:

To Table of Contents

        - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*
    - Parameters:
        - const std::string& sheetKey - Explicit cache key.
        - const std::string& textureRelativePath - Texture path relative to base path.
        - const Vector2i& frameSize - Per-frame size.
        - TextureScaleMode textureScaleModeOverride - Filtering override.
    - Description:
        - Explicit-key variant with filtering override.

- SpriteSheet *LoadSpriteSheet(const std::string& sheetKey, const std::string& textureRelativePath, const Vector2i& frameSize, const Vector3i& colorKey, TextureScaleMode textureScaleModeOverride)*
    - Return:
        - SpriteSheet - *Cached or newly created sprite sheet (nullptr on failure).*
    - Parameters:
        - const std::string& sheetKey - Explicit cache key.
        - const std::string& textureRelativePath - Texture path relative to base path.
        - const Vector2i& frameSize - Per-frame size.
        - const Vector3i& colorKey - Color key for transparency.
        - TextureScaleMode textureScaleModeOverride - Filtering override.
    - Description:
        - Explicit-key variant with both transparency and filtering override.

- SpriteSheet *GetSpriteSheet(const std::string& sheetKey)*
    - Return:
        - SpriteSheet - *Cached sprite sheet pointer or nullptr if missing.*
    - Parameters:
        - const std::string& sheetKey - Cache key.
    - Description:
        - Returns a sprite sheet from the cache without loading.

- bool IsSpriteSheetLoaded(const std::string& sheetKey) const
    - Return:
        - bool - True if the sprite sheet exists in the cache.
    - Parameters:
        - const std::string& sheetKey - Cache key.
    - Description:
        - Checks whether a sprite sheet exists in the cache.

To Table of Contents

- void UnloadSpriteSheet(const std::string& sheetKey)
    - Return:
        - void - No return value.
    - Parameters:
        - const std::string& sheetKey - Cache key.
    - Description:
        - Removes a sprite sheet from the cache. This does not automatically unload its texture unless that texture is also removed from the texture cache.

- void UnloadAllSpriteSheets()
    - Return:
        - void - No return value.
    - Parameters:
        - None
    - Description:
        - Clears the sprite sheet cache.

- AudioClip *LoadAudioClip(const std::string& relativePath)*
    - Return:
        - AudioClip - *Cached or newly loaded AudioClip (throws on load failure).*
    - Parameters:
        - const std::string& relativePath - WAV path relative to base path.
    - Description:
        - Loads a WAV file via SDL_LoadWAV, copies the PCM bytes into a vector, caches the result, and returns a pointer.
    - Steps:
        - If cached, return cached pointer.
        - Build full path from basePath + relativePath.
        - Use SDL_LoadWAV to get AudioSpec, buffer pointer, and length.
        - Copy buffer bytes into AudioClip.pcm and free SDL buffer.
        - Store in m_audioClips and return pointer.

- AudioClip *GetAudioClip(const std::string& relativePath) const*
    - Return:
        - AudioClip - *Cached clip pointer or nullptr if missing.*
    - Parameters:
        - const std::string& relativePath - Cache key.
    - Description:
        - Returns a clip from cache without loading.

To Table of Contents

- bool IsAudioClipLoaded(const std::string& relativePath) const
    - Return:
        - bool - True if the clip exists in the cache.
    - Parameters:
        - const std::string& relativePath - Cache key.
    - Description:
        - Checks whether a clip exists in the cache.

- void UnloadAudioClip(const std::string& relativePath)
    - Return:
        - void - No return value.
    - Parameters:
        - const std::string& relativePath - Cache key.
    - Description:
        - Removes a clip from the cache (releasing its PCM buffer memory).

- void UnloadAllAudioClips()
    - Return:
        - void - No return value.
    - Parameters:
        - None
    - Description:
        - Clears the audio clip cache.

- BitmapFont *LoadFont(const std::string& relativePath, const Vector2i& glyphSize, unsigned char firstChar = 32)*
    - Return:
        - BitmapFont - *Cached or newly loaded font (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Texture path relative to base path.
        - const Vector2i& glyphSize - Width/height of each glyph cell.
        - unsigned char firstChar - ASCII code of the first glyph in the sheet (default 32).
    - Description:
        - Loads a texture and constructs a BitmapFont with sequential mapping starting at firstChar.

- BitmapFont *LoadFont(const std::string& fontKey, const std::string& relativePath, const Vector2i& glyphSize, unsigned char firstChar = 32)*
    - Return:
        - BitmapFont - *Cached or newly loaded font (nullptr on failure).*
    - Parameters:

To Table of Contents

        - const std::string& fontKey - Explicit cache key for the font.
        - const std::string& relativePath - Texture path relative to base path.
        - const Vector2i& glyphSize - Glyph cell size.
        - unsigned char firstChar - ASCII code of the first glyph (default 32).
    - Description:
        - Same as LoadFont(path, glyphSize), but the caller supplies the cache key.

- BitmapFont *LoadFont(const std::string& relativePath, const Vector2i& glyphSize, const Vector3i& colorKey, unsigned char firstChar = 32)*
    - Return:
        - BitmapFont - *Cached or newly loaded font (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Texture path relative to base path.
        - const Vector2i& glyphSize - Glyph cell size.
        - const Vector3i& colorKey - Transparency color key.
        - unsigned char firstChar - ASCII first glyph code.
    - Description:
        - Loads the font texture with a color key to support bitmap transparency.

- BitmapFont *LoadFont(const std::string& fontKey, const std::string& relativePath, const Vector2i& glyphSize, const Vector3i& colorKey, unsigned char firstChar = 32)*
    - Return:
        - BitmapFont - *Cached or newly loaded font (nullptr on failure).*
    - Parameters:
        - const std::string& fontKey - Explicit cache key.
        - const std::string& relativePath - Texture path relative to base path.
        - const Vector2i& glyphSize - Glyph cell size.
        - const Vector3i& colorKey - Transparency key.
        - unsigned char firstChar - ASCII first glyph code.
    - Description:
        - Convenience overload that uses the current default texture scale mode.

- BitmapFont *LoadFont(const std::string& relativePath, const Vector2i& glyphSize, unsigned char firstChar, const Vector3i& colorKey, TextureScaleMode textureScaleModeOverride)*
    - Return:

To Table of Contents

        - BitmapFont - *Cached or newly loaded font (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Texture path relative to base path.
        - const Vector2i& glyphSize - Glyph cell size.
        - unsigned char firstChar - ASCII first glyph code.
        - const Vector3i& colorKey - Transparency key.
        - TextureScaleMode textureScaleModeOverride - Filtering override.
    - Description:
        - Loads a font with explicit transparency and filtering override.

- BitmapFont *LoadFont(const std::string& fontKey, const std::string& relativePath, const Vector2i& glyphSize, unsigned char firstChar, const Vector3i& colorKey, TextureScaleMode textureScaleModeOverride)*
    - Return:
        - BitmapFont - *Cached or newly loaded font (nullptr on failure).*
    - Parameters:
        - const std::string& fontKey - Explicit cache key.
        - const std::string& relativePath - Texture path relative to base path.
        - const Vector2i& glyphSize - Glyph cell size.
        - unsigned char firstChar - ASCII first glyph code.
        - const Vector3i& colorKey - Transparency key.
        - TextureScaleMode textureScaleModeOverride - Filtering override.
    - Description:
        - Full-control overload that selects key, transparency, and filtering.

- BitmapFont *GetFont(const std::string& keyOrRelativePath) const*
    - Return:
        - BitmapFont - *Cached font pointer or nullptr if missing.*
    - Parameters:
        - const std::string& keyOrRelativePath - Cache key.
    - Description:
        - Returns a font from the cache without loading.

- bool IsFontLoaded(const std::string& keyOrRelativePath) const
    - Return:
        - bool - True if the font exists in the cache.
    - Parameters:
        - const std::string& keyOrRelativePath - Cache key.
    - Description:
        - Checks whether a font exists in the cache.

To Table of Contents

- void UnloadFont(const std::string& keyOrRelativePath)
    - Return:
        - void - No return value.
    - Parameters:
        - const std::string& keyOrRelativePath - Cache key.
    - Description:
        - Removes a font from the cache, releasing its texture reference and mapping.

- void UnloadAllFonts()
    - Return:
        - void - No return value.
    - Parameters:
        - None
    - Description:
        - Clears the font cache.

- void UnloadAll()
    - Return:
        - void - No return value.
    - Parameters:
        - None
    - Description:
        - Convenience method that clears all asset caches (textures, sprite sheets, fonts, audio clips).

- std::string BuildDefaultSpriteSheetKey(const std::string& textureRelativePath, const Vector2i& frameSize)
    - Return:
        - std::string - Generated cache key used by the auto-keyed sprite sheet loaders.
    - Parameters:
        - const std::string& textureRelativePath - Texture path.
        - const Vector2i& frameSize - Frame size used to disambiguate sheets that share a texture.
    - Description:
        - Creates a stable string key (texture path + frame size) so different frame sizes produce different cached sheets.

- Texture *LoadTextureInternal(const std::string& relativePath, bool useColorKey, const Vector3i& colorKey, TextureScaleMode textureScaleMode)*
    - Return:

        - Texture - *Loaded/cached texture pointer (nullptr on failure).*
    - Parameters:
        - const std::string& relativePath - Texture path relative to base path.
        - bool useColorKey - Whether the color key should be applied.
        - const Vector3i& colorKey - RGB transparency key (used only if useColorKey is true).
        - TextureScaleMode textureScaleMode - Filtering to apply to the created texture.
    - Description:
        - Shared implementation used by all LoadTexture overloads; computes a cache key and performs the actual disk load + texture creation.
    - Steps:
        - Build a cache key from relativePath, appending a “colorkey:r,g,b” suffix when useColorKey is true.
        - If that key already exists in m_textures, return the cached pointer.
        - Construct the full file path from m_basePath + relativePath.
        - Create a new Texture from the file path (with optional color key applied during Surface creation).
        - Set the texture’s scale mode (Nearest/Linear).
        - Store it in m_textures and return the stored pointer.

Surface:

Purpose: CPU-side image container used while loading BMPs. A Surface is primarily a staging step so the engine can apply a color key before creating a GPU Texture.

Variables:

- void *m_surface - Native SDL_Surface pointer stored opaquely (destroyed on Surface destruction).*

Methods:

- Surface(const std::string& filePath)

To Table of Contents

- Return:
  - Surface - Constructed surface object.
- Parameters:
  - const std::string& filePath - Full file path to a BMP on disk.
- Description:
  - Loads a BMP into an SDL_Surface (throws an EngineException if the file cannot be loaded).

- Surface(Surface&& other) noexcept
  - Return:
    - Surface - Moved surface.
  - Parameters:
    - Surface&& other - Source surface to move from.
  - Description:
    - Transfers ownership of the native surface pointer and nulls out the source.

- Surface& operator=(Surface&& other) noexcept
  - Return:
    - Surface& - This surface after move assignment.
  - Parameters:
    - Surface&& other - Source surface to move from.
  - Description:
    - Destroys any currently owned surface, then takes ownership of other's native pointer.

- ~Surface()
  - Return:
    - void - No return value.
  - Parameters:
    - None
  - Description:
    - Destroys the underlying SDL_Surface if valid.

- void SetColorKey(const Vector3i& color)
  - Return:
    - void - No return value.
  - Parameters:
    - const Vector3i& color - RGB color treated as transparent.
  - Description:
    - Enables surface color keying so the specified RGB becomes transparent when converted into a Texture.

- void ClearColorKey()

To Table of Contents

  - Return:
    - void - No return value.
  - Parameters:
    - None
  - Description:
    - Disables color keying on the surface.

- void *GetNative() const*
  - Return:
    - void - *Native SDL_Surface pointer.*
  - Parameters:
    - None
  - Description:
    - Returns the internal native pointer (used by Texture creation).

- Vector2i GetSize() const
  - Return:
    - Vector2i - Width and height of the surface in pixels.
  - Parameters:
    - None
  - Description:
    - Returns (0,0) if the surface is invalid.

- bool IsValid() const
  - Return:
    - bool - True if a surface is loaded and m_surface is non-null.
  - Parameters:
    - None
  - Description:
    - Validity guard used by loaders.

Texture:

Purpose: GPU-side image resource created from a Surface. Wraps an SDL_Texture and exposes size, validity, and texture filtering (scale mode).

Variables:

- std::unique_ptr<Impl> impl - PIMPL that stores the SDL_Texture pointer,

To Table of Contents

texture size, and current scale mode.

Methods:

- Texture(Renderer& renderer, const std::string& filePath)
    - Return:
        - Texture - Constructed texture object.
    - Parameters:
        - Renderer& renderer - Renderer used to create the native SDL_Texture.
        - const std::string& filePath - Full BMP file path.
    - Description:
        - Loads a BMP via Surface and creates a GPU texture without a color key.

- Texture(Renderer& renderer, const std::string& filePath, bool useColorKey, const Vector3i& colorKey)
    - Return:
        - Texture - Constructed texture object.
    - Parameters:
        - Renderer& renderer - Renderer used to create the native SDL_Texture.
        - const std::string& filePath - Full BMP file path.
        - bool useColorKey - Enables transparency color key processing.
        - const Vector3i& colorKey - RGB key treated as transparent when useColorKey is true.
    - Description:
        - Loads a BMP via Surface, optionally applies a color key, then creates a GPU texture.

- ~Texture()
    - Return:
        - void - No return value.
    - Parameters:
        - None
    - Description:
        - Destroys the underlying SDL_Texture via RAII.

- Texture(Texture&& other) noexcept
    - Return:
        - Texture - Moved texture.
    - Parameters:

To Table of Contents

        - Texture&& other - Source texture.
    - Description:
        - Transfers ownership of the internal SDL_Texture.

- Texture& operator=(Texture&& other) noexcept
    - Return:
        - Texture& - This texture after move assignment.
    - Parameters:
        - Texture&& other - Source texture.
    - Description:
        - Replaces the owned texture with other’s owned texture.

- Vector2i GetSize() const
    - Return:
        - Vector2i - Texture width and height in pixels.
    - Parameters:
        - None
    - Description:
        - Returns a cached size captured from the source Surface at creation time.

- void *GetNative() const*
    - Return:
        - void - *Native SDL_Texture pointer.*
    - Parameters:
        - None
    - Description:
        - Returns the native texture handle for Renderer draw calls.

- bool IsValid() const
    - Return:
        - bool - True if the internal SDL_Texture exists.
    - Parameters:
        - None
    - Description:
        - Validity check used by renderers and asset loaders.

- void SetScaleMode(TextureScaleMode mode)
    - Return:
        - void - No return value.
    - Parameters:
        - TextureScaleMode mode - Nearest or Linear filtering.
    - Description:
        - Updates SDL texture filtering and stores the chosen mode.

To Table of Contents

- TextureScaleMode GetScaleMode() const
    - Return:
        - TextureScaleMode - Current filtering mode.
    - Parameters:
        - None
    - Description:
        - Returns the last set scale mode (defaults to Linear on creation).

SpriteSheet:

Purpose: Lightweight description of a grid-based spritesheet: a Texture pointer plus a per-frame size. This is used by animation and sprite rendering to compute source rectangles by frame index.

Variables:

- std::string name - Debug/cache name for the sheet.
- Texture *texture - Underlying texture (not owned by SpriteSheet; owned by AssetManager texture cache).*
- Vector2i frameSize - Frame width/height in pixels for indexing into the sheet.

Methods:

- bool IsValid() const
    - Return:
        - bool - True if texture exists, the texture is valid, and frameSize is positive.
    - Parameters:
        - None
    - Description:
        - Used as a guard before sampling frames or building animations.

AudioClip:

Purpose: Simple PCM audio container loaded from disk (currently WAV via SDL_LoadWAV). Stored and owned by AssetManager so multiple AudioSources can share the same buffer.

Variables:

- std::string name - Cache key and debug name (usually the relative path).
- SDL_AudioSpec spec - Audio format description for pcm data (rate, channels, sample format).
- std::vector<std::uint8_t> pcm - Raw PCM bytes in the format described by spec.

BitmapFont:

Purpose: Bitmap font that renders fixed-size glyphs from a texture atlas. It maps characters to atlas cells, measures strings, and draws glyphs using renderer texture draw calls.

Variables:

- Texture *m_texture - Texture atlas containing glyphs.*
- Vector2i m_glyphSize - Pixel size of each glyph cell in the atlas.
- Vector2i m_spacing - Extra spacing between glyphs (atlas spacing and visual tracking).
- int m_columns - Number of glyph columns in the atlas grid.
- int m_rows - Number of glyph rows in the atlas grid.
- int m_glyphCount - Total glyph slots (columns *rows).*
- unsigned char m_firstChar - ASCII code that maps to the first glyph slot under default sequential mapping.
- std::array<int, 256> m_charToIndex - Lookup table mapping character code > glyph index (-1 means unmapped).

Methods:

- BitmapFont(Texture *texture, const Vector2i& glyphSize, unsigned char firstChar = 32)*

To Table of Contents

- Return:
    - BitmapFont - Constructed font instance.
- Parameters:
    - Texture *texture - Texture atlas with glyph grid.*
    - const Vector2i& glyphSize - Size of each glyph cell in pixels.
    - unsigned char firstChar - First ASCII character for sequential mapping (default 32 for space).
- Description:
    - Initializes the font grid (columns/rows) from the texture size and builds a default sequential character map.

- void SetSpacing(const Vector2i& spacing)
    - Return:
        - void - No return value.
    - Parameters:
        - const Vector2i& spacing - Spacing between glyph cells in the atlas.
    - Description:
        - Controls how much the draw cursor advances between glyphs beyond glyphSize.

- Vector2i GetSpacing() const
    - Return:
        - Vector2i - Current spacing value.
    - Parameters:
        - None
    - Description:
        - Returns the current spacing configuration.

- Texture *GetTexture() const*
    - Return:
        - Texture - *Atlas texture pointer.*
    - Parameters:
        - None
    - Description:
        - Returns the atlas texture used for drawing glyphs.

- Vector2i GetGlyphSize() const
    - Return:
        - Vector2i - Glyph cell size.
    - Parameters:
        - None
    - Description:
        - Returns the configured glyph size.

To Table of Contents

- bool SetLayoutRows(const std::vector<std::string>& rows, char spacePlaceholder = '_', char emptyPlaceholder = '.')
    - Return:
        - bool - True if the layout matches the atlas grid and mapping succeeded.
    - Parameters:
        - const std::vector<std::string>& rows - String table describing each row of the atlas grid.
        - char spacePlaceholder - Character used to explicitly map a space cell (default '_').
        - char emptyPlaceholder - Character used to mark an unused cell (default '.').
    - Description:
        - Builds a custom mapping from characters to atlas cells. This is useful when the font sheet is not laid out in ASCII order.
    - Steps:
        - Validate that rows.size equals the atlas row count and each row string matches the atlas column count.
        - Clear the char-to-index table to “unmapped”.
        - For each atlas cell:
            1. If emptyPlaceholder, skip the cell.
            2. If spacePlaceholder, map the cell to the space character.
            3. Otherwise map the cell to the given character.
        - Return true if all rows were valid.

- bool MapCharToCell(char c, int col, int row)
    - Return:
        - bool - True if (col,row) is inside the atlas grid and mapping succeeded.
    - Parameters:
        - char c - Character to map.
        - int col - Column index in atlas grid.
        - int row - Row index in atlas grid.
    - Description:
        - Maps a specific character to a specific atlas cell index (manual override).

- bool GetGlyphSourceRect(char c, Vector2f& outSrcPos, Vector2f& outSrcSize) const
    - Return:
        - bool - True if the character is mapped and a valid source rect was produced.
    - Parameters:

To Table of Contents

        - char c - Character to query.
        - Vector2f& outSrcPos - Output top-left in texture pixels.
        - Vector2f& outSrcSize - Output size in texture pixels.
    - Description:
        - Converts a character into a texture source rectangle by using the character mapping and grid dimensions.

- void Draw(Renderer& renderer, const std::string& text, const Vector2f& worldTopLeft, const Vector2f& scale) const
    - Return:
        - void - No return value.
    - Parameters:
        - Renderer& renderer - Renderer used for draw calls.
        - const std::string& text - Text to draw.
        - const Vector2f& worldTopLeft - World-space top-left start position.
        - const Vector2f& scale - Scale applied to glyph size and spacing.
    - Description:
        - Draws each mapped glyph as a textured rectangle, advancing a cursor across the string. Unmapped characters are skipped.

- void DrawColored(Renderer& renderer, const std::string& text, const Vector2f& worldTopLeft, const Vector2f& scale, const Vector4i& color) const
    - Return:
        - void - No return value.
    - Parameters:
        - Renderer& renderer - Renderer used for draw calls.
        - const std::string& text - Text to draw.
        - const Vector2f& worldTopLeft - World-space top-left start position.
        - const Vector2f& scale - Scale applied to glyph size and spacing.
        - const Vector4i& color - RGBA tint applied to glyph draw calls.
    - Description:
        - Same as Draw, but applies a tint color to each glyph.

- Vector2f MeasureText(const std::string& text, const Vector2f& scale) const
    - Return:
        - Vector2f - Size of the rendered string in world units/pixels (depending on your world scale).
    - Parameters:
        - const std::string& text - Text to measure.
        - const Vector2f& scale - Scale applied to glyph size and spacing.
    - Description:
        - Computes how much space the string would occupy when

To Table of Contents

drawn by simulating the cursor advance across all mapped characters.

Utilities

- Files:
  - Logger.h/.cpp
  - EngineException.hpp
  - Types.hpp
  - ViewportUtils.h

- Summary: Shared helpers used by multiple subsystems (diagnostics, error reporting, math primitives, and viewport-bound checks).

Logger:

- Files: Logger.h/.cpp
- Summary: Thread-safe logger that writes timestamped messages to the console (with severity coloring) and can optionally mirror the same output into a log file.

Variables:

- std::mutex logMutex - serializes log writes to keep multi-threaded output readable.
- bool fileLoggingEnabled - toggles file output on/off.
- std::ofstream logFile - append-only stream used when file logging is enabled.
- static Logger *instance - legacy pointer member; the active singleton instance is provided by GetInstance().*

Methods:

- static Logger& GetInstance()
  - Return:
    - Logger& - singleton Logger instance.

    - Parameters:
        - None
    - Description:
        - Returns a single Logger object that is created the first time it is requested.

- void Init(const std::string& filename = "log.txt")
    - Return:
        - void - initializes optional file logging.
    - Parameters:
        - const std::string& filename - path of the log file to open in append mode.
    - Steps:
        - Locks the logger mutex.
        - Opens the file stream.
        - Enables file logging if the stream successfully opens.

- void Shutdown()
    - Return:
        - void - closes and flushes the log file if it is open.
    - Parameters:
        - None
    - Steps:
        - Locks the logger mutex.
        - Flushes and closes the file stream if needed.

- void Log(const std::string& message, int level)
    - Return:
        - void - writes a formatted line to console (and optionally file).
    - Parameters:
        - const std::string& message - message text.
        - int level - severity level (Trace, Debug, Info, Warning, Error, Critical).
    - Steps:
        - Locks the logger mutex.
        - Creates a prefix containing time and severity label.
        - Chooses a console color based on severity.
        - Prints the final line to the console.
        - If file logging is enabled, appends the same formatted line to the log file.

- void Trace(const std::string& message)
    - Return:
        - void - logs with Trace severity.

To Table of Contents

    - Parameters:
        - const std::string& message - message text.

- void Debug(const std::string& message)
    - Return:
        - void - logs with Debug severity.
    - Parameters:
        - const std::string& message - message text.

- void Info(const std::string& message)
    - Return:
        - void - logs with Info severity.
    - Parameters:
        - const std::string& message - message text.

- void Warning(const std::string& message)
    - Return:
        - void - logs with Warning severity.
    - Parameters:
        - const std::string& message - message text.

- void Error(const std::string& message)
    - Return:
        - void - logs with Error severity.
    - Parameters:
        - const std::string& message - message text.

- void Critical(const std::string& message)
    - Return:
        - void - logs with Critical severity.
    - Parameters:
        - const std::string& message - message text.

- bool IsFileLoggingEnabled() const
    - Return:
        - bool - true if file output is enabled.
    - Parameters:
        - None

- void SetFileLoggingEnabled(bool enabled)
    - Return:
        - void - toggles file logging on/off.
    - Parameters:
        - bool enabled - new file logging state.

To Table of Contents

- private std::string GetCurrentTime()
    - Return:
        - std::string - time string used in log prefixes.
    - Parameters:
        - None

- private std::string GetCurrentDate()
    - Return:
        - std::string - date string used for file naming or reporting.
    - Parameters:
        - None

- private std::string LevelToString(int level)
    - Return:
        - std::string - severity label for the provided level.
    - Parameters:
        - int level - severity value.

- private void WriteToFile(const std::string& formattedMessage)
    - Return:
        - void - appends a pre-formatted message into the log file.
    - Parameters:
        - const std::string& formattedMessage - full formatted line.

Macros:

- LOG_TRACE(message)
    - Description:
        - Convenience wrapper that calls Logger::Trace.

- LOG_DEBUG(message)
    - Description:
        - Convenience wrapper that calls Logger::Debug.

- LOG_INFO(message)
    - Description:
        - Convenience wrapper that calls Logger::Info.

- LOG_WARNING(message)
    - Description:
        - Convenience wrapper that calls Logger::Warning.

To Table of Contents

- LOG_ERROR(message)
    - Description:
        - Convenience wrapper that calls Logger::Error.

- LOG_CRITICAL(message)
    - Description:
        - Convenience wrapper that calls Logger::Critical.

EngineException:

- Files: EngineException.hpp
- Summary: Lightweight exception type plus throwing macros. Used when the engine wants a clear failure with a readable message and optional source context.

Variables:

- std::string m_message - stored error message returned by what().

Methods:

- EngineException(const std::string& message)
    - Return:
        - void - constructs an EngineException storing the provided message.
    - Parameters:
        - const std::string& message - error description.

- const char *what() const noexcept*
    - Return:
        - const char - *pointer to the stored message buffer.*
    - Parameters:
        - None

- template<typename T> EngineException& operator<<(EngineException& ex, const T& value)
    - Return:
        - EngineException& - same exception instance with its message

To Table of Contents

extended.
    - Parameters:
        - EngineException& ex - exception being built.
        - const T& value - appended value (formatted through a string stream).
    - Description:
        - Allows stream-like message building, for example by appending numbers or labels without manually formatting.

Macros:

- THROW_ENGINE_EXCEPTION(message)
    - Description:
        - Throws EngineException with the provided message.

- THROW_ENGINE_EXCEPTION_WITH_CONTEXT(message)
    - Description:
        - Throws EngineException with the provided message plus the current file and line.

Types:

- Files: Types.hpp
- Summary: Header-only math primitives used across the engine, including numeric helpers, vector types, rectangles, and a 3x3 matrix used to build 2D transform matrices.

Variables:

- template<typename T> struct Math::Constants
    - static constexpr T Pi - pi constant.
    - static constexpr T TwoPi - 2*pi.*
    - static constexpr T HalfPi - pi/2.
    - static constexpr T Epsilon - numeric epsilon for T.
    - static constexpr T Deg2Rad - degrees-to-radians multiplier.
    - static constexpr T Rad2Deg - radians-to-degrees multiplier.

- template<typename T> class Vector2
    - T x - X component.

To Table of Contents

    - T y - Y component.

- template<typename T> class Vector3
    - T x - X component.
    - T y - Y component.
    - T z - Z component.

- template<typename T> class Vector4
    - T x - X component.
    - T y - Y component.
    - T z - Z component.
    - T w - W component.

- template<typename T> class Rect
    - T x - left coordinate.
    - T y - top coordinate.
    - T width - width.
    - T height - height.

- class Matrix3x3f
    - float m[9] - matrix data stored in column-major order.

- Common typedefs
    - using Vector2f = Vector2<float> - float 2D vector.
    - using Vector2i = Vector2<int> - int 2D vector.
    - using Vector2u = Vector2<unsigned int> - unsigned int 2D vector.
    - using Vector2d = Vector2<double> - double 2D vector.
    - using Vector3f = Vector3<float> - float 3D vector.
    - using Vector3i = Vector3<int> - int 3D vector.
    - using Vector3u = Vector3<unsigned int> - unsigned int 3D vector.
    - using Vector3d = Vector3<double> - double 3D vector.
    - using Vector4f = Vector4<float> - float 4D vector.
    - using Vector4i = Vector4<int> - int 4D vector.
    - using Vector4u = Vector4<unsigned int> - unsigned int 4D vector.
    - using Vector4d = Vector4<double> - double 4D vector.
    - using Recti = Rect<int> - int rectangle.
    - using Rectf = Rect<float> - float rectangle.
    - using Rectu = Rect<unsigned int> - unsigned int rectangle.
    - using Rectd = Rect<double> - double rectangle.

ViewportUtils

To Table of Contents

- File: GameEngine/ViewportUtils.h

- Purpose: Utility helpers for working with the visible world area defined by the game's virtual resolution. Provides world bounds, side bitmasks, and point/rectangle tests and clamps so gameplay code can easily detect when something leaves the screen (or keep it inside).

Types:

- Viewport::Side
    - Type: enum class Side : uint8_t - Bitmask used to describe one or more sides of the viewport.
    - Values:
        - None - No side selected.
        - Left - Left edge.
        - Right - Right edge.
        - Top - Top edge.
        - Bottom - Bottom edge.
        - All - Combination of Left, Right, Top, Bottom.

- Viewport::Bounds
    - Type: struct Bounds - World-space bounds of the visible game area.
    - Fields:
        - float left - World X coordinate of the left edge.
        - float right - World X coordinate of the right edge.
        - float bottom - World Y coordinate of the bottom edge.
        - float top - World Y coordinate of the top edge.

- Viewport::WorldRect
    - Type: struct WorldRect - Rectangle expressed as a top-left point plus a positive size.
    - Fields:
        - Vector2f topLeft - World-space top-left corner.
        - Vector2f size - Width and height (size.x >= 0, size.y >= 0).
    - Notes:
        - The helpers treat top as larger Y and bottom as smaller Y (Y is up). The bottom edge is computed as topLeft.y - size.y.

Methods:

- operator|(Side a, Side b)
    - Return:
        - Side - A combined side mask that contains all bits present in a or b.

To Table of Contents

- Parameters:
    - Side a - Left operand.
    - Side b - Right operand.

- operator&(Side a, Side b)
  - Return:
    - Side - The intersection of both masks (only the bits present in both).
  - Parameters:
    - Side a - Left operand.
    - Side b - Right operand.

- operator|=(Side& a, Side b)
  - Return:
    - Side& - Reference to a after OR-ing in b.
  - Parameters:
    - Side& a - Mask to be updated.
    - Side b - Bits to add into a.

- Any(Side s)
  - Return:
    - bool - True if the mask contains at least one side bit.
  - Parameters:
    - Side s - Mask to test.

- GetWorldBounds(const Vector2i& gameSize)
  - Return:
    - Bounds - World-space bounds for the visible area.
  - Parameters:
    - const Vector2i& gameSize - Virtual resolution in pixels (width, height).
  - Steps:
    - Computes half-width and half-height.
    - Returns bounds centered at world origin:
      - left = -width/2, right = +width/2
      - bottom = -height/2, top = +height/2

- OutsideSidesPoint(const Vector2f& p, const Vector2i& gameSize)
  - Return:
    - Side - Mask of which sides the point is outside of.
  - Parameters:
    - const Vector2f& p - Point in world space.
    - const Vector2i& gameSize - Virtual resolution used to compute bounds.

To Table of Contents

    - Steps:
        - Builds world bounds from gameSize.
        - Tests p against each boundary:
            - If p.x is left of bounds.left, sets Left.
            - If p.x is right of bounds.right, sets Right.
            - If p.y is above bounds.top, sets Top.
            - If p.y is below bounds.bottom, sets Bottom.

- IsPointOutside(const Vector2f& p, const Vector2i& gameSize, Side sides = Side::All)
    - Return:
        - bool - True if the point is outside any of the requested sides.
    - Parameters:
        - const Vector2f& p - Point in world space.
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.
        - Side sides - Which sides to consider (default: all).
    - Steps:
        - Computes OutsideSidesPoint(p, gameSize).
        - Intersect the result with sides.
        - Returns true if any bit remains.

- ClampPoint(const Vector2f& p, const Vector2i& gameSize, Side sides = Side::All)
    - Return:
        - Vector2f - Clamped point that is forced inside the chosen sides.
    - Parameters:
        - const Vector2f& p - Point in world space.
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.
        - Side sides - Which sides are allowed to clamp (default: all).
    - Steps:
        - Computes world bounds from gameSize.
        - For each side included in sides:
            - Left clamps x up to bounds.left.
            - Right clamps x down to bounds.right.
            - Top clamps y down to bounds.top.
            - Bottom clamps y up to bounds.bottom.

- RectLeft(const WorldRect& r)
    - Return:
        - float - Left edge x coordinate.
    - Parameters:
        - const WorldRect& r - Rectangle.

To Table of Contents

- RectRight(const WorldRect& r)
    - Return:
        - float - Right edge x coordinate.
    - Parameters:
        - const WorldRect& r - Rectangle.

- RectTop(const WorldRect& r)
    - Return:
        - float - Top edge y coordinate.
    - Parameters:
        - const WorldRect& r - Rectangle.

- RectBottom(const WorldRect& r)
    - Return:
        - float - Bottom edge y coordinate.
    - Parameters:
        - const WorldRect& r - Rectangle.
    - Notes:
        - Computed as r.topLeft.y - r.size.y.

- OutsideSidesRect(const WorldRect& r, const Vector2i& gameSize)
    - Return:
        - Side - Mask of which sides the rectangle is fully outside of.
    - Parameters:
        - const WorldRect& r - Rectangle to test.
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.
    - Steps:
        - Computes world bounds from gameSize.
        - Marks a side only if the rectangle is completely on that side:
            - Left if rect.right < bounds.left
            - Right if rect.left > bounds.right
            - Top if rect.bottom > bounds.top
            - Bottom if rect.top < bounds.bottom

- OverhangSidesRect(const WorldRect& r, const Vector2i& gameSize)
    - Return:
        - Side - Mask of which sides the rectangle is partially outside of.
    - Parameters:
        - const WorldRect& r - Rectangle to test.
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.
    - Steps:

To Table of Contents

        - Computes world bounds from gameSize.
        - Marks a side if any part crosses that boundary:
            - Left if rect.left < bounds.left
            - Right if rect.right > bounds.right
            - Top if rect.top > bounds.top
            - Bottom if rect.bottom < bounds.bottom

- IsRectOutside(const WorldRect& r, const Vector2i& gameSize, Side sides = Side::All)
    - Return:
        - bool - True if the rectangle is fully outside any of the requested sides.
    - Parameters:
        - const WorldRect& r - Rectangle to test.
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.
        - Side sides - Which sides to consider (default: all).
    - Steps:
        - Computes OutsideSidesRect(r, gameSize).
        - Intersects with sides and returns true if any bit remains.

- IsRectOverhanging(const WorldRect& r, const Vector2i& gameSize, Side sides = Side::All)
    - Return:
        - bool - True if the rectangle is partially outside any of the requested sides.
    - Parameters:
        - const WorldRect& r - Rectangle to test.
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.
        - Side sides - Which sides to consider (default: all).
    - Steps:
        - Computes OverhangSidesRect(r, gameSize).
        - Intersects with sides and returns true if any bit remains.

- ClampRect(WorldRect r, const Vector2i& gameSize, Side sides = Side::All)
    - Return:
        - WorldRect - A rectangle shifted so it fits fully inside the chosen sides.
    - Parameters:
        - WorldRect r - Rectangle to clamp (passed by value so it can be modified and returned).
        - const Vector2i& gameSize - Virtual resolution used to compute bounds.

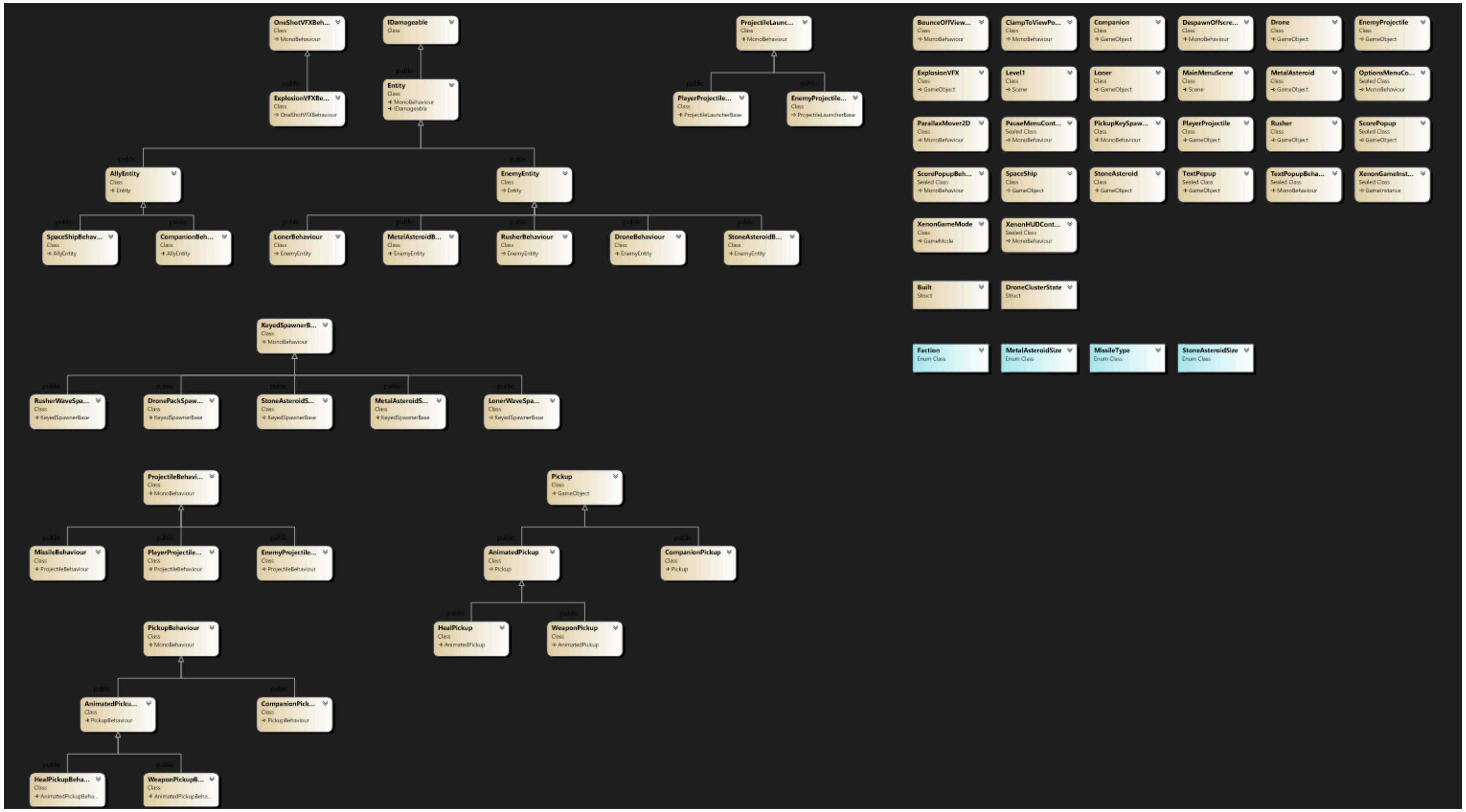
To Table of Contents

        - Side sides - Which sides are allowed to clamp (default: all).
    - Steps:
        - Computes bounds from gameSize.
        - Horizontal:
            1. If left edge is past bounds.left, shifts rect right until aligned.
            2. If right edge is past bounds.right, shifts rect left until aligned.
        - Vertical:
            1. If top edge is above bounds.top, shifts rect down until aligned.
            2. If bottom edge is below bounds.bottom, shifts rect up until aligned.
        - Returns the adjusted rectangle.

To Table of Contents

Xenon Game Demo

To Table of Contents


OneShotVFXBeh... Class MonoBehaviour
IDamageable Class
ProjectileLaunc... Class MonoBehaviour
ExplosionVFXBe... Class OneShotVFXBehaviour
Entity Class MonoBehaviour IDamageable
PlayerProjectile... Class ProjectileLauncherBase
EnemyProjectile... Class ProjectileLauncherBase
AllyEntity Class Entity
EnemyEntity Class Entity
SpaceShipBehav... Class AllyEntity
CompanionBeh... Class AllyEntity
LonerBehaviour Class EnemyEntity
MetalAsteroidB... Class EnemyEntity
RusherBehaviour Class EnemyEntity
DroneBehaviour Class EnemyEntity
StoneAsteroidB... Class EnemyEntity
KeyedSpawnerB... Class MonoBehaviour
RusherWaveSpa... Class KeyedSpawnerBase
DronePackSpaw... Class KeyedSpawnerBase
StoneAsteroidS... Class KeyedSpawnerBase
MetalAsteroidS... Class KeyedSpawnerBase
LonerWaveSpa... Class KeyedSpawnerBase
ProjectileBehavi... Class MonoBehaviour
MissileBehaviour Class ProjectileBehaviour
PlayerProjectile... Class ProjectileBehaviour
EnemyProjectile... Class ProjectileBehaviour
Pickup Class GameObject
AnimatedPickup Class Pickup
CompanionPickup Class Pickup
HealPickup Class AnimatedPickup
WeaponPickup Class AnimatedPickup
PickupBehaviour Class MonoBehaviour
AnimatedPicku... Class PickupBehaviour
CompanionPick... Class PickupBehaviour
HealPickupBeha... Class AnimatedPickupBeha...
WeaponPickupB... Class AnimatedPickupBeha...
public
BounceOffView... Class MonoBehaviour
ClampToViewPo... Class MonoBehaviour
Companion Class GameObject
DespawnOffscre... Class MonoBehaviour
Drone Class GameObject
EnemyProjectile Class GameObject
ExplosionVFX Class GameObject
Level1 Class Scene
Loner Class GameObject
MainMenuScene Class Scene
MetalAsteroid Class GameObject
OptionsMenuCo... Sealed Class MonoBehaviour
ParallaxMover2D Class MonoBehaviour
PauseMenuCont... Sealed Class MonoBehaviour
PickupKeySpaw... Class MonoBehaviour
PlayerProjectile Class GameObject
Rusher Class GameObject
ScorePopup Sealed Class GameObject
ScorePopupBeh... Sealed Class MonoBehaviour
SpaceShip Class GameObject
StoneAsteroid Class GameObject
TextPopup Sealed Class GameObject
TextPopupBeha... Sealed Class MonoBehaviour
XenonGameInst... Sealed Class GameInstance
XenonGameMode Class GameMode
XenonHUDCont... Sealed Class MonoBehaviour
Built Struct
DroneClusterState Struct
Faction Enum Class
MetalAsteroidSize Enum Class
MissileType Enum Class
StoneAsteroidSize Enum Class

To Table of Contents

1. Bootstrapping and global state

Main entry point:

- Files:
    - Xenon/Main.cpp
- Purpose:
    - Creates the engine Config used by the game, wires the Xenon GameInstance type, selects the first Scene, and starts the engine loop.

Functions:

- main(int argc, char **argv)**
    - Return:
        - int — process exit code returned to the OS.
    - Parameters:
        - int argc — number of command-line arguments.
        - char **argv — array of argument strings.**
    - Description:
        - Builds a Config with the target FPS and fixed delta time settings used by the engine.
        - Sets the asset base path (where the engine will look for bmp and wav files).
        - Creates the initial Scene (MainMenuScene) and gives it to the engine.
        - Sets the GameInstance factory to XenonGameInstance so persistent game state exists before the first scene starts.
        - Calls the engine Run loop, then performs clean shutdown when the game exits.

XenonGameInstance:

- Files:
    - Xenon/XenonGameInstance.hpp
- Inherits:
    - GameInstance (GameEngine)
- Purpose:
    - Stores game-wide configuration and paths that must persist across

To Table of Contents

Scene changes (for example, the file used to save high scores).

Variables:

- std::string scoreFilename
    - Location of the game's high-score file (written on shutdown by the active GameMode).

Methods:

- XenonGameInstance()
    - Return:
        - constructor — creates the instance with default settings.
    - Parameters:
        - none
    - Description:
        - Sets scoreFilename to a default save file path inside the assets folder.

- std::string GetScoreFile() const
    - Return:
        - std::string — the currently configured save file path.
    - Parameters:
        - none
    - Description:
        - Exposes the score file path so menus and game modes can read and write the top-10 score table.

2. Scene flow and per-scene rules

XenonGameMode:

- Files:
    - Xenon/XenonGameMode.hpp
- Inherits:
    - GameMode (GameEngine)
- Purpose:
    - Defines per-run rules for score, lives, health, and high-score persistence.
    - Acts as the "single source of truth" for run statistics so UI and gameplay systems don't directly depend on each other.

To Table of Contents

Variables:

- int m_playerLives
  - Remaining lives for the player in the current run.
- float m_playerHealth
  - Current health in the [0..1] range (normalized), used directly by UIProgressBar.
- int m_currentScore
  - Score accumulated in the current run.
- std::vector<int> m_topScores
  - Loaded top-10 score table (kept sorted, trimmed to 10).

Methods:

- void OnAttach(Scene& scene)
  - Return:
    - void — no return value.
  - Parameters:
    - Scene& scene — the Scene that owns this GameMode.
  - Description:
    - Loads the current top-10 score list from disk (if available) so UI can show it.
    - Initializes per-run counters (lives, health, score) to defaults.

- void OnStart()
  - Return:
    - void — no return value.
  - Parameters:
    - none
  - Description:
    - Called once after the Scene becomes active.
    - Typically used to reset UI and ensure initial run state is consistent before gameplay begins.

- void OnDestroy()
  - Return:
    - void — no return value.
  - Parameters:
    - none
  - Description:
    - Saves the current run score to the top-10 list (if it qualifies), then writes the table to disk.

To Table of Contents

            - This guarantees high scores persist even if the player quits directly from gameplay.

- void AddScore(int amount)
    - Return:
        - void — no return value.
    - Parameters:
        - int amount — points to add to the run score.
    - Description:
        - Adds amount to m_currentScore.
        - Keeps score operations centralized so enemy and pickup code does not need to know about high-score persistence.

- void SetHealth01(float health01)
    - Return:
        - void — no return value.
    - Parameters:
        - float health01 — new normalized health value (0 = dead, 1 = full).
    - Description:
        - Clamps health into [0..1] and stores it.
        - Normalized health simplifies UI because the progress bar can render directly from this value.

- void LoseLife()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Decrements m_playerLives and applies the run's death policy:
            - If there are remaining lives, health is restored and the player can respawn.
            - If there are no lives left, gameplay will return to the main menu or end the run (depending on the current scene's flow).

- int GetScore() const
    - Return:
        - int — current run score.
    - Parameters:
        - none
    - Description:
        - Read-only accessor for UI (score label) and other systems (for

To Table of Contents

example, end-of-run screens).

- int GetLives() const
    - Return:
        - int — current remaining lives.
    - Parameters:
        - none
    - Description:
        - Read-only accessor for UI (lives icons) and logic gates (for example, whether respawn is allowed).

- float GetHealth01() const
    - Return:
        - float — current health in [0..1].
    - Parameters:
        - none
    - Description:
        - Used by XenonHUDController to drive the health progress bar.

- const std::vector<int>& GetTopScores() const
    - Return:
        - const std::vector<int>& — reference to the top-10 list.
    - Parameters:
        - none
    - Description:
        - Used by UI code to display the persistent high score list.

MainMenuScene:

- Files:
    - Xenon/MainMenuScene.hpp
- Inherits:
    - Scene (GameEngine)
- Purpose:
    - Builds the main menu UI (buttons and navigation) and routes the player into gameplay scenes.
    - Hosts the Options overlay so options can be reused later (for example, during pause) without duplicating UI.

Variables:

- std::shared_ptr<UICanvas> m_canvas

To Table of Contents

    - Root canvas for main menu UI.
- std::shared_ptr<UILabel> m_title
    - Game title label.
- std::vector<std::shared_ptr<UIButton>> m_buttons
    - List of menu buttons (used for easy enabling/disabling and navigation wiring).
- std::shared_ptr<OptionsMenuController> m_options
    - Options overlay controller instance (spawned when entering options).
- std::shared_ptr<UILabel> m_hiScoreLabel
    - Label used to show the current top score on the main menu.

Methods:

- void OnStart()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Creates UI elements for the menu (canvas, labels, buttons).
        - Wires each button's callback:
            1. Start Game creates Level1 (or equivalent gameplay scene) and switches the engine to it.
            2. Options toggles the OptionsMenuController overlay.
            3. Exit requests engine quit.
        - Sets up navigation neighbors so the menu works with keyboard/gamepad focus.

- void OnUpdate()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Updates any dynamic labels (for example, refreshing the displayed high score if the table changed).
        - Delegates to options overlay controller if it is active.

Level1:

- Files:
    - Xenon/Level1.hpp

To Table of Contents

- Inherits:
    - Scene (GameEngine)
- Purpose:
    - Implements the first gameplay level.
    - Responsible for spawning the player, setting up spawners, and connecting the HUD to the active XenonGameMode.

Variables:

- std::shared_ptr<GameObject> m_player
    - Player ship root object.
- std::shared_ptr<XenonHUDController> m_hud
    - HUD controller for health, lives, and score UI.
- std::vector<std::shared_ptr<MonoBehaviour>> m_spawners
    - Collection of enemy/pickup spawners used by this level.

Methods:

- void OnStart()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Ensures this Scene has a XenonGameMode instance attached.
        - Spawns the player SpaceShip and sets its initial position.
        - Creates the HUD and binds it to the current XenonGameMode.
        - Creates spawners for enemies and pickups and places them in the level.

- void OnUpdate()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Handles debug hotkeys for test spawning (when enabled in code).
        - Runs any level-specific rules (such as manual wave triggers or test spawns).

To Table of Contents

3. HUD and menu overlays

XenonHUDController:

- Files:
  - Xenon/XenonHUDController.hpp
- Inherits:
  - MonoBehaviour (GameEngine)
- Purpose:
  - Owns all on-screen HUD widgets (health bar, lives, score, hi-score) and updates them from XenonGameMode every frame.

Variables:

- std::shared_ptr<UICanvas> m_canvas
  - HUD canvas.
- std::shared_ptr<UIProgressBar> m_healthBar
  - Bottom-left health bar.
- std::shared_ptr<UILabel> m_playerLabel
  - Top-left "Player One" text.
- std::shared_ptr<UILabel> m_scoreLabel
  - Top-left score value.
- std::shared_ptr<UILabel> m_hiScoreLabel
  - Top-center "Hi Score" label.
- std::shared_ptr<UILabel> m_hiScoreValue
  - Top-center hi-score number.
- std::vector<std::shared_ptr<UIImage>> m_lifeIcons
  - UI images used to represent lives.
- XenonGameMode *m_mode*
  - Non-owning pointer to the active XenonGameMode.

Methods:

- void BindToGameMode(XenonGameMode *mode)*
  - Return:
    - void — no return value.
  - Parameters:
    - XenonGameMode *mode — game mode providing score, health, and lives.*
  - Description:
    - Stores the mode pointer used by Update.
    - Allows this HUD to be created before the game mode is fully

configured, then bound later.

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Creates the HUD canvas and all widgets.
        - Loads fonts and UI textures via the engine AssetManager.
        - Applies consistent UI colors (magenta color key where required by the textures).

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Pulls health, score, and lives from XenonGameMode.
        - Updates:
            - Health bar fill value.
            - Score label text.
            - Life icons visibility count.
            - Hi-score label/value from the top-10 list.

OptionsMenuController:

- Files:
    - Xenon/OptionsMenuController.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Creates an options overlay that can be used from the main menu (and reused later from pause).
    - Contains the control scheme selection logic (keyboard vs gamepad assignment).

Variables:

- std::shared_ptr<UICanvas> m_canvas
    - Root canvas of the options overlay.

To Table of Contents

- std::shared_ptr<UIPanel> m_panel
    - Backdrop panel behind the options widgets.
- std::vector<std::shared_ptr<UIButton>> m_buttons
    - Option buttons (player 1 controls, player 2 controls, back).
- int m_p1ControlMode
    - Player 1 input mode selection.
- int m_p2ControlMode
    - Player 2 input mode selection.
- std::function<void()> m_onClose
    - Callback invoked when the overlay should close.

Methods:

- void SetOnClose(std::function<void()> onClose)
    - Return:
        - void — no return value.
    - Parameters:
        - std::function<void()> onClose — action to run when leaving the options overlay.
    - Description:
        - Allows the calling Scene or controller to decide what “close” means (return to menu, return to pause, etc.).

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Builds the overlay UI (canvas, panel, labels, buttons).
        - Sets up focus navigation and button callbacks.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Updates text labels to reflect the current selections (keyboard or gamepad).
        - Accepts submit/back actions to close the overlay.

To Table of Contents

PauseMenuController:

- Files:
  - Xenon/PauseMenuController.hpp
- Inherits:
  - MonoBehaviour (GameEngine)
- Purpose:
  - Provides an in-game pause overlay and controls pause/resume behavior.
  - Uses Time::SetPaused so gameplay freezes without destroying scene state.

Variables:

- std::shared_ptr<UICanvas> m_canvas
  - Pause overlay canvas.
- std::shared_ptr<UIPanel> m_panel
  - Background panel.
- std::vector<std::shared_ptr<UIButton>> m_buttons
  - Buttons (resume, options, quit to menu).
- std::shared_ptr<OptionsMenuController> m_options
  - Options overlay for pause context.
- bool m_visible
  - Whether pause overlay is currently visible.

Methods:

- void SetVisible(bool visible)
  - Return:
    - void — no return value.
  - Parameters:
    - bool visible — show or hide pause overlay.
  - Description:
    - Toggles the canvas visibility and sets Time paused/unpaused accordingly.
    - Ensures input focus is set to the first button when opened.

- bool IsVisible() const
  - Return:
    - bool — true when pause overlay is visible.
  - Parameters:
    - none
  - Description:

To Table of Contents

        - Used by gameplay code to avoid re-opening pause repeatedly.

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Builds pause UI widgets and sets callbacks:
            - Resume: closes the overlay and resumes time.
            - Options: opens OptionsMenuController in pause context.
            - Quit: switches back to MainMenuScene.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Listens for pause input (typically start/escape) and toggles visibility.
        - If options overlay is open, routes back action to close it first.

4. Gameplay base types

Faction:

- Files:
    - Xenon/Faction.hpp
- Purpose:
    - Provides a shared enum used by projectiles and entities to decide whether collisions should deal damage.

Types:

- enum class Faction
    - Player
    - Enemy

To Table of Contents

IDamageable:

- Files:
    - Xenon/IDamageable.hpp
- Purpose:
    - Small interface used to apply damage without requiring a hard dependency on a concrete entity type.

Methods:

- void ApplyDamage(float amount)
    - Return:
        - void — no return value.
    - Parameters:
        - float amount — damage amount to subtract from health.
    - Description:
        - Implementations reduce health and trigger death/feedback behavior.

Entity:

- Files:
    - Xenon/Entity.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Shared base class for “things that can take damage and die”.
    - Provides a consistent component layout: SpriteRenderer, Rigidbody2D, Collider2D, Animator, AudioSource.

Variables:

- SpriteRenderer *sprite*
    - Cached sprite renderer used for visuals.
- Rigidbody2D *rigidbody*
    - Cached rigidbody controlling movement.
- Collider2D *collider*
    - Cached collider for hits.
- Animator *animator*
    - Cached animator for sprite animation (optional depending on entity).
- AudioSource *audioSource*
    - Cached audio source (optional depending on entity).

To Table of Contents

- float maxHealth
  - Maximum health value (used to compute normalized health).
- float health
  - Current health.
- bool invulnerable
  - Whether this entity ignores incoming damage.

Methods:

- void Awake()
  - Return:
    - void — no return value.
  - Parameters:
    - none
  - Description:
    - Locates required components on the GameObject and caches them.
    - Sets baseline physics state (body type, gravity scale, rotation policy) if needed by derived classes.

- void ApplyDamage(float amount)
  - Return:
    - void — no return value.
  - Parameters:
    - float amount — amount of damage to apply.
  - Description:
    - Early exits when invulnerable is true.
    - Reduces health and calls OnDeath when health reaches zero.

- virtual void OnDeath()
  - Return:
    - void — no return value.
  - Parameters:
    - none
  - Description:
    - Hook for derived entities to spawn VFX, award score, or trigger respawn logic.

AllyEntity:

- Files:
  - Xenon/AllyEntity.hpp

To Table of Contents

- Inherits:
    - Entity (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Shared base for player-aligned entities (player ship, companions).
    - Provides team identity and common overlap handling for friendly collisions.

Variables:

- Faction faction
    - Set to Player for allies.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Calls Entity::Awake, then assigns faction and baseline ally physics settings.

EnemyEntity:

- Files:
    - Xenon/EnemyEntity.hpp
- Inherits:
    - Entity (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Shared base for enemy-aligned entities, including score award logic and overlap behavior against allies.

Variables:

- int scoreValue
    - Points awarded when this enemy dies.
- Faction faction
    - Set to Enemy for enemies.

To Table of Contents

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Calls Entity::Awake, then assigns faction and enemy defaults.

- virtual void HandleOverlap(GameObject *other)*
    - Return:
        - void — no return value.
    - Parameters:
        - GameObject *other — the object that overlapped this enemy's collider.*
    - Description:
        - Resolves whether the overlap should cause damage based on other's components (projectile vs ally).
        - Triggers damage, death, and score award through XenonGameMode.

5. Player, movement, and player weapons

SpaceShip:

- Files:
    - Xenon/SpaceShip.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Player ship prefab expressed as a C++ class.
    - Builds the full component stack in the constructor so scenes can spawn the player with a single CreateGameObject call.

Variables:

- (none)
    - This type mainly exists to bundle construction and attach SpaceShipBehaviour.

To Table of Contents

Methods:

- SpaceShip(Scene *scene)*
    - Return:
        - constructor — creates the ship object and all components.
    - Parameters:
        - Scene *scene — owning scene (used by some pooled creation paths).*
    - Description:
        - Adds SpriteRenderer using Ship2 texture and sets frame size.
        - Adds Rigidbody2D and a BoxCollider2D configured as a trigger.
        - Adds Animator and AudioSource.
        - Adds SpaceShipBehaviour (script) that implements movement, firing, damage, and respawn policy.
        - Adds viewport helper components so the ship stays inside screen bounds.

SpaceShipBehaviour:

- Files:
    - Xenon/SpaceShip.hpp
- Inherits:
    - AllyEntity (Xenon) which inherits Entity which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Implements player control:
        - Reads input, applies velocity, handles shooting cooldowns.
        - Tracks invulnerability windows and death animations.
        - Talks to XenonGameMode to update health/lives and score.

Variables:

- float m_speed
    - Base movement speed.
- float m_shootCooldown
    - Minimum time between shots.
- float m_lastShotTime
    - Last shot timestamp (used to enforce cooldown).
- float m_invulnTime
    - Duration of post-hit invulnerability.
- float m_lastHitTime
    - Timestamp of most recent hit.
- int m_weaponLevel

To Table of Contents

    - Upgrade level that changes the projectile pattern.
- bool m_dead
    - Whether the ship is currently in death state.
- bool m_respawning
    - Whether the ship is in a respawn transition.
- XenonGameMode *m_mode*
    - Cached pointer to current game mode.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Calls AllyEntity::Awake to cache core components.
        - Retrieves XenonGameMode from the Scene and caches it.
        - Initializes weapon state and synchronizes health to the game mode for UI.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Reads movement input and builds a 2D direction vector.
        - Writes velocity to the Rigidbody2D so physics is authoritative for position.
        - Handles firing:
            - Checks cooldown by comparing current Time::Now() with m_lastShotTime.
            - Spawns player projectiles using the current weapon level pattern.
        - Applies invulnerability logic:
            - If recently hit, enables invulnerable and optionally flashes/changes animation state.
        - When dead:
            - Plays death animation, then asks XenonGameMode whether to respawn or return to menu.

- void ApplyDamage(float amount)

To Table of Contents

    - Return:
        - void — no return value.
    - Parameters:
        - float amount — damage dealt to the player.
    - Description:
        - Enforces invulnerability window.
        - Reduces health, updates XenonGameMode normalized health.
        - When health reaches zero:
            - Calls XenonGameMode::LoseLife.
            - Starts death state and triggers VFX/audio.

- void SetWeaponLevel(int level)
    - Return:
        - void — no return value.
    - Parameters:
        - int level — new weapon upgrade level.
    - Description:
        - Stores the current weapon level.
        - Weapon level influences projectile spawn pattern (single, spread, missiles, etc).

6. Enemies and enemy spawning

Rusher:

- Files:
    - Xenon/Rusher.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Enemy prefab that rushes the player with simple steering and deals damage on overlap.

Methods:

- Rusher(Scene *scene)*
    - Return:
        - constructor — builds enemy components and attaches RusherBehaviour.
    - Parameters:

To Table of Contents

        - Scene *scene — owning scene.*
    - Description:
        - Creates sprite/physics components and installs RusherBehaviour.

RusherBehaviour:

- Files:
    - Xenon/Rusher.hpp
- Inherits:
    - EnemyEntity (Xenon) which inherits Entity which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Implements “rush straight towards player” behavior and handles death + score.

Variables:

- float m_speed
    - Movement speed.
- float m_turnRate
    - How quickly the enemy rotates to face target.
- GameObject *m_target*
    - Pointer to the player object.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Finds and caches the player target from the scene.
        - Configures rigidbody body type and collision policies.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:

To Table of Contents

        - Computes direction from enemy to player.
        - Rotates gradually toward the direction using turn rate.
        - Applies forward velocity so the enemy closes distance.

Loner:

- Files:
    - Xenon/Loner.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Enemy prefab that enters screen, pauses, then fires a spread of bullets.

LonerBehaviour:

- Files:
    - Xenon/Loner.hpp
- Inherits:
    - EnemyEntity (Xenon) which inherits Entity which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Implements Loner's "arrive then shoot" state machine.
    - Uses a small internal timer to swap states without needing scene-level scripting.

Variables:

- enum State { Entering, Waiting, Shooting, Leaving }
    - Current behavior phase.
- float m_stateTime
    - Time spent in current state.
- float m_enterSpeed
    - Movement speed used while entering.
- float m_waitDuration
    - How long to wait before firing.
- int m_bulletsPerShot
    - Spread size.
- float m_fireCooldown
    - Time between shots (if multiple shots are used).
- EnemyProjectileLauncher *m_launcher*

To Table of Contents

    - Cached launcher used to spawn bullets.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Configures kinematic movement and disables gravity.
        - Creates or finds an EnemyProjectileLauncher and caches it.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Advances m_stateTime using Time::DeltaTime.
        - Switches state when timers complete:
            - Entering: move into a visible zone.
            - Waiting: hold still for m_waitDuration.
            - Shooting: fire a spread (bulletsPerShot) via launcher.
            - Leaving: move out (commonly to the left) and despawn when offscreen.

Drone:

- Files:
    - Xenon/Drone.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Enemy prefab that spawns in packs, flies in an S-curve, and awards increasing points within a cluster.

DroneBehaviour:

- Files:
    - Xenon/Drone.hpp

To Table of Contents

- Inherits:
    - EnemyEntity (Xenon) which inherits Entity which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Implements pack spawning, looping animation, and sin-wave motion.
    - Uses a shared DroneClusterState so each kill in the same pack is worth more.

Variables:

- BoxCollider2D *boxCol*
    - Collider reference used for size and overlap.
- Animator *animator*
    - Animator driving looping frames.
- SpriteSheet *m_sheet*
    - Drone sprite sheet.
- AnimationClip m_clip
    - “Loop all frames” clip built at runtime.
- AnimatorController m_ctrl
    - Controller with a single looping state.
- float m_forwardSpeed, m_sinAmplitude, m_sinHz, m_phase
    - Parameters controlling S-curve motion.
- int m_packIndex
    - Index of this drone inside its pack (0..N-1).
- float m_packSpacing
    - Spacing along the line spawn.
- bool m_usePackLine
    - Whether pack line spawning is enabled.
- Vector2f m_startPos
    - Spawn origin used as the base point for the sin offset.
- float m_birthTime
    - Spawn timestamp used to compute sin phase over time.
- std::shared_ptr<DroneClusterState> m_cluster
    - Shared kill state for scoring escalation.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:

To Table of Contents

        - Sets rigidbody to kinematic and disables gravity.
        - Loads drone.bmp as a sprite sheet and assigns it to SpriteRenderer.
        - Builds a looping animation clip that plays all frames.
        - Applies visual variation by seeking to a pack-index-dependent normalized time so drones don't animate in sync.

- void Start()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Captures m_startPos and m_birthTime.
        - Applies "perfect line" offset if m_usePackLine is enabled (packIndex *packSpacing).*

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Computes forward motion along -LocalUp and side motion along LocalRight.
        - Uses a sine wave with amplitude and Hz to get the side offset.
        - Applies final velocity and updates transform via rigidbody.
        - Despawns only when leaving the screen on the left side (so it can fly in from other sides without instant despawn).

- void OnDeath()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Increments the shared cluster kill counter.
        - Computes score using basePoints plus an escalation based on killsSoFar.
        - Spawns a score popup and VFX, then destroys the drone object.

To Table of Contents

StoneAsteroid:

- Files:
    - Xenon/StoneAsteroids.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Asteroid prefab for the “stone” theme.
    - Spawns with random rotation speed and drifts; breaks into smaller asteroids when destroyed.

StoneAsteroidBehaviour:

- Files:
    - Xenon/StoneAsteroids.hpp
- Inherits:
    - EnemyEntity (Xenon) which inherits Entity which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Implements asteroid movement and split-on-death behavior.

Variables:

- float m_spinDegPerSec
    - Rotation speed in degrees per second.
- Vector2f m_linearVel
    - Constant drift velocity applied each frame.
- int m_sizeTier
    - Tier controlling sprite frame size and whether it can split further.
- float m_spawnTime
    - Used for time-based effects if needed.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Configures rigidbody as kinematic and sets initial spin and drift.
        - Chooses a sprite frame and scale based on m_sizeTier.

To Table of Contents

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Applies constant drift and spin.
        - Uses viewport despawn helpers to remove the asteroid when it leaves the active play area.

- void OnDeath()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Awards score through XenonGameMode.
        - Spawns explosion VFX.
        - If size tier allows, spawns two smaller asteroids with adjusted velocities, then destroys the original.

MetalAsteroid:

- Files:
    - Xenon/MetalAsteroids.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - A tougher asteroid variant (metal theme) with different stats and visuals.

MetalAsteroidBehaviour:

- Files:
    - Xenon/MetalAsteroids.hpp
- Inherits:
    - EnemyEntity (Xenon) which inherits Entity which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Similar to StoneAsteroidBehaviour but with different health/score values and potentially different split rules.

To Table of Contents

EnemySpawners:

- Files:
  - Xenon/EnemySpawners.hpp
- Purpose:
  - Contains spawner behaviours used by Level1 to generate enemies, pickups, and waves with consistent timing and placement.

KeyedSpawnerBase:

- Files:
  - Xenon/EnemySpawners.hpp
- Inherits:
  - MonoBehaviour (GameEngine)
- Purpose:
  - Common spawner base that supports enabling/disabling and debug key triggers.

Variables:

- bool m_enabled
  - Whether the spawner is currently active.
- Key m_debugKey
  - Optional key used to force a spawn during debugging.
- float m_cooldown
  - Minimum delay between spawns.

Methods:

- void Update()
  - Return:
    - void — no return value.
  - Parameters:
    - none
  - Description:
    - If enabled, checks timers and spawns according to the derived policy.
    - If a debug key is configured, allows immediate spawn on key press.

To Table of Contents

WaveSpawner:

- Files:
    - Xenon/EnemySpawners.hpp
- Inherits:
    - KeyedSpawnerBase (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Spawns a timed sequence of enemies (waves) rather than continuous random spawns.

Methods:

- void StartWave()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Resets wave counters and starts spawning the wave pattern.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Advances a wave timer and spawns wave entries at specific times.

7. Projectiles and combat objects

Projectile common types:

- Files:
    - Xenon/ProjectileCommon.hpp
- Purpose:
    - Defines the shared projectile behaviour base classes for both player and enemy projectiles, plus collision rules using Faction.

To Table of Contents

ProjectileBehaviour:

- Files:
    - Xenon/ProjectileCommon.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Base script for a projectile that moves with a constant velocity and despawns when leaving the viewport.

Variables:

- Faction faction
    - Owner faction used to decide what can be damaged.
- float speed
    - Projectile forward speed.
- float damage
    - Damage applied on hit.
- float lifetime
    - Maximum age before auto-despawn.
- float birthTime
    - Spawn timestamp used to compute age.
- Vector2f direction
    - Normalized direction vector.
- XenonDespawnOffscreen2D *despawn*
    - Viewport helper used to despawn when outside.

Methods:

- void Awake()
    - Return:
        - void — no return value.
    - Parameters:
        - none
    - Description:
        - Captures spawn time and ensures despawn helper exists.
        - Initializes default movement direction based on transform orientation.

- void Update()
    - Return:
        - void — no return value.
    - Parameters:

To Table of Contents

        - none
    - Description:
        - Moves the projectile forward at speed.
        - Destroys the projectile after lifetime seconds or when offscreen.

- void OnTriggerEnter(Collider2D *other)*
    - Return:
        - void — no return value.
    - Parameters:
        - Collider2D *other — collider that was hit.*
    - Description:
        - Resolves hit target.
        - Applies damage to IDamageable targets that belong to the opposite faction.
        - Spawns hit VFX and destroys the projectile on successful hit.

PlayerProjectileBehaviour:

- Files:
    - Xenon/ProjectileCommon.hpp
- Inherits:
    - ProjectileBehaviour (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Specializes defaults for player shots (faction, damage, speed) and any unique hit effects.

EnemyProjectileBehaviour:

- Files:
    - Xenon/ProjectileCommon.hpp
- Inherits:
    - ProjectileBehaviour (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Specializes defaults for enemy shots and ensures they can damage the player.

ProjectileLaunchers:

- Files:

To Table of Contents

    - Xenon/ProjectileLaunchers.hpp
- Purpose:
    - Provides reusable launcher components that spawn projectiles with a consistent API.

ProjectileLauncherBase:

- Files:
    - Xenon/ProjectileLaunchers.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Provides common spawn helpers and cooldown handling.

Variables:

- float m_cooldown
    - Minimum time between shots.
- float m_lastShotTime
    - Timestamp of last spawn.

Methods:

- bool CanShoot(float now) const
    - Return:
        - bool — true when enough time has passed since last shot.
    - Parameters:
        - float now — current timestamp (Time::Now).
    - Description:
        - Compares now with m_lastShotTime and m_cooldown.

- void MarkShot(float now)
    - Return:
        - void — no return value.
    - Parameters:
        - float now — current timestamp.
    - Description:
        - Updates m_lastShotTime to enforce cooldown.

To Table of Contents

PlayerProjectileLauncher:

- Files:
  - Xenon/ProjectileLaunchers.hpp
- Inherits:
  - ProjectileLauncherBase (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
  - Spawns player bullets with spread patterns depending on weapon level.

EnemyProjectileLauncher:

- Files:
  - Xenon/ProjectileLaunchers.hpp
- Inherits:
  - ProjectileLauncherBase (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
  - Spawns enemy bullets aimed at the player or in fixed spread arcs.

ProjectileObjects:

- Files:
  - Xenon/ProjectileObjects.hpp
- Purpose:
  - Defines GameObject wrappers (prefabs) that attach the correct projectile behaviour and render components.

EnemyProjectiles:

- Files:
  - Xenon/EnemyProjectiles.hpp
- Purpose:
  - Defines enemy bullet prefab(s) and helper container object(s) used to batch enemy shots.

Missiles:

- Files:

To Table of Contents

- Xenon/Missiles.hpp
- Purpose:
    - Defines missile prefabs and behaviour:
        - missiles accelerate, seek targets, and explode on hit.

8. Pickups and upgrades

Pickup:

- Files:
    - Xenon/Pickup.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Base pickup prefab that uses a trigger collider and a PickupBehaviour to detect overlap with the player.

PickupBehaviour:

- Files:
    - Xenon/Pickup.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Detects player overlap and triggers the pickup effect.

Variables:

- bool m_consumed
    - Prevents double-consumption when multiple overlaps happen in the same frame.

Methods:

- void OnTriggerEnter(Collider2D *other)*
    - Return:
        - void — no return value.
    - Parameters:
        - Collider2D *other — collider that entered.*

To Table of Contents

    - Description:
        - Checks if other belongs to the player ship.
        - Calls ApplyToPlayer on derived behaviours.
        - Marks consumed and destroys the pickup GameObject.

AnimatedPickup:

- Files:
    - Xenon/AnimatedPickup.hpp
- Inherits:
    - Pickup (Xenon) which inherits GameObject (GameEngine)
- Purpose:
    - Pickup base class that includes an Animator and a looping sprite animation so derived pickups share the same visual setup.

AnimatedPickupBehaviour:

- Files:
    - Xenon/AnimatedPickup.hpp
- Inherits:
    - PickupBehaviour (Xenon) which inherits MonoBehaviour (GameEngine)
- Purpose:
    - Same overlap logic as PickupBehaviour, but also ensures the looping animation is configured.

HealPickup:

- Files:
    - Xenon/HealPickup.hpp
- Inherits:
    - AnimatedPickup (Xenon) which inherits GameObject (GameEngine)
- Purpose:
    - Restores player health.

WeaponPickup:

- Files:
    - Xenon/WeaponPickup.hpp
- Inherits:

To Table of Contents

    - AnimatedPickup (Xenon) which inherits GameObject (GameEngine)
- Purpose:
    - Upgrades player weapon level and changes projectile pattern.

CompanionPickup:

- Files:
    - Xenon/CompanionPickup.hpp
- Inherits:
    - AnimatedPickup (Xenon) which inherits GameObject (GameEngine)
- Purpose:
    - Spawns a Companion ally entity that fights alongside the player.

Companion:

- Files:
    - Xenon/Companion.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Ally ship that orbits/follows the player and fires automatically.
    - Implemented as a prefab + CompanionBehaviour script.

9. VFX and on-screen feedback

VFX helpers:

- Files:
    - Xenon/VFX.hpp
- Purpose:
    - Prefabs and behaviours for explosions and one-shot effects.

OneShotVFXBehaviour:

- Files:
    - Xenon/VFX.hpp
- Inherits:
    - MonoBehaviour (GameEngine)

To Table of Contents

- Purpose:
    - Plays an animation clip once and destroys the object when finished.

ExplosionVFX:

- Files:
    - Xenon/VFX.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - Explosion prefab used by enemy deaths and projectile impacts.

ScorePopup:

- Files:
    - Xenon/ScorePopup.hpp
- Inherits:
    - GameObject (GameEngine)
- Purpose:
    - A short-lived text popup showing points gained at the enemy's position.

ScorePopupBehaviour:

- Files:
    - Xenon/ScorePopup.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Moves the popup upward, fades it out, and destroys it after a small lifetime.

10. Environment helpers

ParallaxMover2D:

- Files:
    - Xenon/ParallaxBackground.hpp

To Table of Contents

- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Moves background objects at a slower rate than the camera/player to create parallax depth.
    - Designed to work with rotated sprites and “edge hugging” placement.

ClampToViewport2D:

- Files:
    - Xenon/XenonViewportComponents.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Clamps a GameObject’s position so it stays inside the visible world bounds.

DespawnOffscreen2D:

- Files:
    - Xenon/XenonViewportComponents.hpp
- Inherits:
    - MonoBehaviour (GameEngine)
- Purpose:
    - Destroys a GameObject when it leaves the viewport on configured sides.
    - Used by projectiles, asteroids, and enemies to keep gameplay objects from existing forever offscreen.

11. Asset keys and asset organization

XenonAssetKeys:

- Files:
    - Xenon/XenonAssetKeys.h
- Purpose:
    - Central list of string keys used when loading cached assets via AssetManager.
    - Prevents typos and keeps asset lookups consistent across all gameplay and UI code.

To Table of Contents

Types:

- struct XenonAssetKeys::Sheets
  - Contains keys for sprite sheet assets (Ship, enemies, pickups, VFX).
- struct XenonAssetKeys::Textures
  - Contains keys for single textures (backgrounds, UI sprites).
- struct XenonAssetKeys::Fonts
  - Contains keys for bitmap fonts (8x8 and 16x16).
- struct XenonAssetKeys::Audio
  - Contains keys for wav clips (shots, pickups, explosions).

Conclusion

I really enjoyed working on this project this year. I felt rewarded for my efforts even with the occasional frustrations that surged. There is a lot that I would improve. Making the logger more accessible, putting the object pool actually working in the game. Find a better way to serialize Strings. And possibly even try to make this into an editor based game engine. I probably won't touch this project for a while, but I hope to come back to it one day.

References:

- Unity documentation (Scripting API + Manual)
  - https://docs.unity3d.com
  - https://docs.unity3d.com/6000.3/Documentation/ScriptReference/MonoBehaviour.html
  - https://docs.unity3d.com/6000.3/Documentation/ScriptReference/Time.html
  - https://docs.unity3d.com/6000.3/Documentation/Manual/execution-order.html
  - https://docs.unity3d.com/6000.5/Documentation/Manual/script-execution-order.html

- Unreal Engine documentation (Game framework concepts)
  - https://dev.epicgames.com/documentation/en-us/unreal-engine/game-mode-and-game-state-in-unreal-engine
  - https://dev.epicgames.com/documentation/en-us/unreal-engine/python-

To Table of Contents

api/class/GameInstance?application_version=4.27

- Fixed timestep and accumulator model
    - https://gafferongames.com/post/fix_your_timestep/

- SDL3 documentation (rendering, surfaces/textures, events, audio)
    - https://wiki.libsdl.org/SDL3
    - https://wiki.libsdl.org/SDL3/CategoryRender
    - https://wiki.libsdl.org/SDL3/CategoryEvents
    - https://wiki.libsdl.org/SDL3/SDL_PollEvent
    - https://wiki.libsdl.org/SDL3/SDL_Surface
    - https://wiki.libsdl.org/SDL3/SDL_Texture
    - https://wiki.libsdl.org/SDL3/SDL_CreateTexture
    - https://wiki.libsdl.org/SDL3/SDL_CreateTextureFromSurface
    - https://wiki.libsdl.org/SDL3/SDL_RenderTexture
    - https://wiki.libsdl.org/SDL3/SDL_LoadBMP_IO
    - https://wiki.libsdl.org/SDL3/SDL_SetSurfaceColorKey
    - https://wiki.libsdl.org/SDL3/SDL_AudioStream
    - https://wiki.libsdl.org/SDL3/SDL_PutAudioStreamData
    - https://wiki.libsdl.org/SDL3/SDL_GetAudioStreamData

- Box2D documentation (world, bodies, shapes, contacts)
    - https://box2d.org/documentation/

- C++ standard library references (algorithms, containers, time, threading, I/O)
    - https://en.cppreference.com/w/cpp/17.html
    - https://en.cppreference.com/w/cpp/utility/functional/invoke.html
    - https://en.cppreference.com/w/cpp/chrono/steady_clock
    - https://en.cppreference.com/w/cpp/chrono/time_point
    - https://en.cppreference.com/w/cpp/thread/sleep_for
    - https://en.cppreference.com/w/cpp/io/basic_ifstream
    - https://en.cppreference.com/w/cpp/io/basic_ofstream
    - https://en.cppreference.com/w/cpp/container/vector
    - https://en.cppreference.com/w/cpp/memory/shared_ptr
    - https://en.cppreference.com/w/cpp/container/unordered_map
    - https://en.cppreference.com/w/cpp/container/unordered_set
    - https://en.cppreference.com/w/cpp/algorithm/stable_sort
    - https://en.cppreference.com/w/cpp/algorithm/remove

- Game architecture / patterns used conceptually (loop, object pool, dirty flags)
    - https://gameprogrammingpatterns.com/game-loop.html
    - https://gameprogrammingpatterns.com/update-method.html
    - https://gameprogrammingpatterns.com/object-pool.html
    - https://gameprogrammingpatterns.com/dirty-flag.html

To Table of Contents

- File I/O basics (quick reference)
    - https://www.w3schools.com/CPP/cpp_files.asp
    - https://www.w3schools.com/cpp/cpp_ref_reference.asp

- Visual Studio diagrams / tooling
    - https://stackoverflow.com/questions/17191218/generate-a-class-diagram-from-visual-studio
    - https://www.doxygen.nl/manual/docblocks.html
    - https://www.doxygen.nl/download.html

- Misc
    - https://stackoverflow.com/questions/4053837/colorizing-text-in-the-console-with-c
    - https://stackoverflow.com/questions/2789481/problem-calling-stdmax

To Table of Contents