# Meatcorps Engine & Snake Game Development Status

This document contains a detailed overview of the current state of the Meatcorps custom RayLib-powered game engine and the Snake arcade game project, including completed features, architecture details, and planned features.

## 1. Engine Architecture Overview

The engine is built on RayLib for C#, chosen for its simplicity, cross-platform capabilities, and excellent support for ARM and lower-spec devices compared to MonoGame. The architecture is modular and supports dependency injection via an ObjectManager. Key systems include:  
- BaseScene management with sub-scene support  
- ObjectManager for game object registration and lookup  
- RenderService with strict scene and game object layers for deterministic rendering order  
- Camera abstraction with world/UI layers  
- Post-processing system with pluggable effects  
- Input mapping architecture with PlayerInputRouter for flexible player control mapping  
- Arduino arcade controller integration with per-button lighting and animations  
- Particle system with fluent builder API and mutators  
- Tweening utilities for animation, easing, and interpolation

## 2. Completed Features

- Core scene management and game loop  
- Strict rendering layer order (scene layers + game object layers)  
- PixelPerfectRenderTarget with post-processing pipeline  
- Input abstraction via IInputMapper and IInputMapperWithManager  
- PlayerInputRouter to assign mappers to different players  
- ArduinoControllerModule with fluent setup API, player enabling, and button animations (BlinkAnimation, FlashAnimation)  
- Particle system with support for custom mutators and velocity-based oscillations  
- Meat spawn logic ensuring placement only on free grid positions  
- Multiplayer support with Arduino double-player arcade controller  
- Modular architecture to allow Snake game logic to remain separate from engine internals  
- PlayerInputRouter has been implemented.

## 3. Planned Snake Game Features

- Wall spawning after a total number of meat slices eaten, placed once no snake tail is in the location  
- Warning system before walls spawn  
- Meat rotting over time, turning black and reducing score if eaten  
- Boid system for flies that target the oldest meat slices  
- Power-ups that grant temporary speed boosts, with timed expiration  
- Intro and outro screens for arcade mode

## 4. Planned Engine Improvements

- Implement PlayerInputRouter mapping for mixed control schemes (e.g., Player 1 on keyboard, Player 2 on controller)  
- Expanded controller support: keyboard, mouse, and standard gamepads  
- Optional benchmarking overlay for FPS, frame time, draw calls, and resolution scaling  
- Runtime toggling of post-processing effects  
- Low-spec optimization path for Raspberry Pi or weaker PCs (dynamic resolution, reduced effects)

## 5. Hardware Testing

- Current development and testing primarily on MacBook Pro M4 Max (high-end, baseline performance)  
- Planned tests on Ryzen 7 7840U mini PC with Radeon 780M  
- Steam Deck (Zen 2 APU + RDNA 2 GPU) as a mid-tier benchmark environment  
- Expected portability to Raspberry Pi 5 for lightweight builds (without heavy post-processing)

## 6. Next Immediate Steps

- Finalize wall spawning and warning logic  
- Implement meat rotting system with flies. The flies will be implemented as a boid system.  
- Add power-up spawning and logic  
- Create intro/outro screens   
- Optional: Build benchmarking overlay for hardware testing

# Quick audit checklist you can run through your codebase

* No new [] in any **Draw/Update/Apply** method (post‑FX, particles, AI).
* No ToList(), Select, Where inside the main loop.
* No string interpolation or Console.WriteLine during frames.
* All shader SetValue/Get\* use stackalloc.
* Any temporary arrays in gameplay AI use ArrayPool<T> and are returned same frame.
* Background services are cached as List<IBackgroundService> (you already do) and iterated directly.
* No lambdas created inside update/draw; anything callback‑based is registered once.