3D Dungeon Ecosystem Engine - Development Roadmap

Phase 1: Foundation Layer (Weeks 1-2)

Step 1: Project Setup and Build System

1. **Initialize Project Structure**
    - Create folder structure as defined
    - Set up `package.json` with dependencies (webpack, babel, testing framework)
    - Configure webpack for development and production builds
    - Set up basic HTML entry point with canvas element
2. **Core Math Library**
    - Implement `Vector3.js` with basic operations (add, subtract, normalize, dot, cross)
    - Build `Matrix4.js` with transformation operations (translate, rotate, scale, multiply)
    - Create `Quaternion.js` for rotation handling
    - Add `MathUtils.js` with common functions (clamp, lerp, random ranges)
    - Write unit tests for all math operations
3. **Basic WebGL Context**
    - Create `WebGLContext.js` to initialize WebGL2 context
    - Implement basic error handling and capability detection
    - Set up viewport management and context loss handling
    - Add basic clear and drawing setup

Step 2: Minimal Rendering Pipeline

1. **Shader System Foundation**
    - Build `ShaderManager.js` to compile and cache shaders
    - Create basic vertex shader (transform vertices to screen space)
    - Create basic fragment shader (solid color output)
    - Implement shader program linking and uniform management
2. **Geometry System**
    - Implement `Mesh.js` for vertex buffer management
    - Create `PrimitiveGenerator.js` with cube, sphere, plane generation
    - Build vertex array object (VAO) management

   - Add basic mesh rendering functionality

3. **Camera System**
   - Implement `Camera.js` with perspective projection
   - Add view matrix calculation (look-at functionality)
   - Create basic camera controls (orbit, pan, zoom)
   - Integrate camera with shader uniforms

Step 3: Scene Management

1. **Scene Graph**
   - Build `Scene.js` with hierarchical node structure
   - Implement transform inheritance (parent-child relationships)
   - Add node addition/removal and traversal
   - Create basic frustum culling
2. **Render Queue**
   - Implement `RenderQueue.js` for draw call batching
   - Add depth sorting for transparent objects
   - Create material-based sorting for efficiency
   - Build basic render state management

Milestone 1: Render a spinning colored cube

Phase 2: Advanced Rendering (Weeks 3-4)

Step 4: Materials and Lighting

1. **Material System**
   - Create `Material.js` with diffuse, specular, normal properties
   - Implement material-shader binding
   - Add uniform parameter management
   - Build material presets for different surface types
2. **Lighting Framework**
   - Implement `Light.js` with point, directional, spot lights
   - Add Phong/Blinn-Phong lighting in shaders
   - Create shadow mapping basic setup

   - Integrate lighting with material system

3. **Texture Management**
   - Build `Texture.js` with loading and caching
   - Implement texture atlasing for creatures
   - Add normal mapping and specular mapping
   - Create procedural texture generation utilities

Step 5: Spatial Systems

1. **Spatial Partitioning**
   - Implement `Octree.js` for 3D space subdivision
   - Create `SpatialHash.js` for fast neighbor queries
   - Add insertion, removal, and query operations
   - Build collision detection framework
2. **Collision and Physics**
   - Create `AABB.js` for bounding box collisions
   - Implement `Ray.js` for ray casting
   - Add basic physics integration (position, velocity)
   - Build collision response system

Milestone 2: Render a lit dungeon room with textured walls

Phase 3: Procedural Generation (Weeks 5-6)

Step 6: Dungeon Generation

1. **Room Generation**
   - Implement `RoomGenerator.js` with rectangular rooms
   - Add room size variation and environmental parameters
   - Create wall, floor, ceiling geometry generation
   - Build room decoration placement (rocks, water, debris)
2. **Corridor System**
   - Build `CorridorGenerator.js` for room connections
   - Implement corridor pathfinding and geometry
   - Add corridor width variation and branching

- Create seamless room-corridor transitions

3. **Environmental Zones**
   - Implement `EnvironmentalZones.js` for temperature/humidity
   - Create gradient calculation between zones
   - Add water source placement and flow simulation
   - Build organic matter distribution

Step 7: Terrain and Surfaces

1. **Organic Terrain**
   - Create `TerrainGenerator.js` with height maps
   - Implement moss, slime, and debris surface generation
   - Add surface normal calculation for lighting
   - Build texture blending for surface transitions
2. **Water System**
   - Implement basic water surface geometry
   - Create water shader with reflection/refraction
   - Add water flow visualization
   - Build humidity calculation from water sources

Milestone 3: Generate a complete dungeon with environmental zones

---

Phase 4: Core Simulation (Weeks 7-9)

Step 8: Ecosystem Foundation

1. **Species Definition System**
   - Implement `Species.js` with all ecological parameters
   - Create species JSON configuration loading
   - Build trophic level and food web relationships
   - Add environmental requirement definitions
2. **Population Dynamics**
   - Build `Population.js` with birth/death calculations
   - Implement carrying capacity calculations
   - Add population pressure effects

- Create genetic diversity tracking

3. **Environmental Simulation**
    - Implement `Environment.js` with all environmental factors
    - Create environmental update loops
    - Add environmental gradient calculations
    - Build resource depletion and renewal

Step 9: Room-Level Simulation

1. **Individual Room Logic**
    - Implement `Room.js` with complete ecosystem simulation
    - Add predator-prey interaction calculations
    - Create disease outbreak modeling
    - Build resource competition systems
2. **Migration System**
    - Build `Migration.js` with population pressure triggers
    - Implement inter-room movement calculations
    - Add migration path optimization
    - Create migration visualization data

Step 10: Web Worker Integration

1. **Simulation Worker**
    - Create `SimulationWorker.js` for ecosystem calculations
    - Implement message passing between main thread and worker
    - Add worker-safe data serialization
    - Build simulation state synchronization
2. **Multi-threaded Architecture**
    - Separate simulation timing from rendering timing
    - Implement simulation result caching
    - Add worker error handling and recovery
    - Create performance monitoring for workers

Milestone 4: Complete ecosystem simulation running in background

---

Phase 5: Creature Visualization (Weeks 10-11)

Step 11: Creature Rendering

1. **Entity System**
   - Implement `Entity.js` base class
   - Create `Creature.js` with position, scale, rotation
   - Build creature model loading and instancing
   - Add creature color variation based on genetics
2. **Animation Framework**
   - Build `ProceduralAnimation.js` for creature movement
   - Implement basic walking, feeding, fleeing animations
   - Create animation blending between states
   - Add inverse kinematics for leg placement
3. **Instanced Rendering**
   - Implement GPU instancing for creature populations
   - Create instance data buffers (position, scale, color)
   - Add level-of-detail switching based on distance
   - Build creature population culling

Step 12: Behavior Visualization

1. **Movement and Flocking**
   - Implement `Flocking.js` with boids algorithm
   - Create pathfinding visualization
   - Add territorial behavior display
   - Build predator-prey chase sequences
2. **Life Cycle Events**
   - Create birth/death particle effects
   - Implement creature aging visualization
   - Add feeding behavior animations
   - Build mating behavior displays

Milestone 5: Animated creatures moving through the dungeon

Phase 6: Advanced Features (Weeks 12-13)

Step 13: Data Visualization

1. **Population Graphs**
   - Implement `PopulationGraphs.js` with 3D bar charts
   - Create population trend line displays
   - Add species composition pie charts
   - Build interactive graph controls
2. **Environmental Overlays**
   - Create `EnvironmentalOverlay.js` with heat maps
   - Implement temperature/humidity gradient visualization
   - Add resource availability displays
   - Build migration path trail rendering

Step 14: Interactive Tools

1. **Ecosystem Manipulation**
   - Create creature placement/removal tools
   - Implement environmental modification tools
   - Add food source dropping functionality
   - Build disease introduction tools
2. **Time Controls**
   - Implement time scaling (pause, fast-forward, rewind)
   - Create time-lapse recording and playback
   - Add ecosystem state save/load functionality
   - Build historical data browsing

Milestone 6: Complete interactive ecosystem visualization

Phase 7: Polish and Optimization (Weeks 14-16)

Step 15: Performance Optimization

1. **Rendering Optimization**
   - Implement frustum culling optimization
   - Add occlusion culling for hidden rooms

  - Create texture atlasing and compression
  - Build mesh optimization and LOD generation

2. **Simulation Optimization**
   - Optimize spatial queries with better data structures
   - Implement simulation result caching
   - Add adaptive simulation quality based on performance
   - Create memory pool management for creatures

Step 16: User Experience

1. **UI/UX Polish**
   - Create comprehensive debug panel
   - Build intuitive camera controls
   - Add helpful tooltips and information displays
   - Implement ecosystem health indicators
2. **Audio Integration**
   - Add spatial audio for creatures and environment
   - Create procedural creature sounds
   - Implement ambient dungeon atmosphere
   - Build audio cues for ecological events

Step 17: Documentation and Testing

1. **Comprehensive Testing**
   - Write unit tests for all simulation components
   - Create integration tests for ecosystem stability
   - Add performance regression tests
   - Build automated ecosystem health validation
2. **Documentation**
   - Write API documentation for all systems
   - Create user guides for ecosystem interaction
   - Build developer tutorials for extending the system
   - Document ecological model assumptions and limitations

Final Milestone: Complete, polished 3D dungeon ecosystem simulation

Development Guidelines

Daily Development Routine

1. **Morning**: Write/review code for current step
2. **Midday**: Test and debug current implementation
3. **Afternoon**: Document progress and plan next steps
4. **Evening**: Commit code and update milestone tracking

Quality Gates

- Each step must pass unit tests before proceeding
- Performance benchmarks must be met at each milestone
- Code review and documentation required for each major component
- Ecosystem stability testing required before advancing phases

Risk Mitigation

- Keep simulation and rendering completely separate to avoid coupling
- Build debugging tools early to identify problems quickly
- Create fallback options for complex features (simplified models)
- Regular performance testing to avoid late-stage optimization crises

Success Metrics

- Stable ecosystem with realistic population dynamics
- Smooth 60fps rendering with thousands of creatures
- Intuitive user interaction and ecosystem manipulation
- Extensible architecture for adding new species/behaviors
- Comprehensive documentation and testing coverage

This roadmap balances building a solid technical foundation with creating visible progress milestones that demonstrate the unique ecological simulation features.