

Battle for the Oceans - Development To-Do List

Last Updated: October 1, 2025 Game Bible Version: 3.3

✓ COMPLETED

Service Architecture (v2.3)

- ✓ UserProfileService.js - user profile CRUD operations
- ✓ GameStatsService.js - game completion tracking
- ✓ LeaderboardService.js - ranking calculations with guest/AI filtering
- ✓ GameContext.js architecture updates






Monetization & Rights System (v2.4)

- ✓ Database schema for user_rights and vouchers tables
- ✓ Voucher generation script (`scripts/vouchers.js v0.2.0`)
- ✓ Support for era unlocks and boost types
- ✓ RightsService.js implementation
- ✓ Purchase flow with Stripe payment processing
- ✓ Voucher redemption working (tested with Midway Island)

State Management Improvements

- ✓ Remove useEffect state transitions from PlayingPage and PlacementPage
- ✓ Era data consolidation through EraService
- ✓ Message system integration with fallbacks

CSS Architecture Modernization (v3.3)

-  BEM methodology implementation
 -  Modular CSS file structure created:
 - theme.css - CSS custom properties, typography
 - modal-overlay.css - Modal systems, overlays
 - buttons.css - Unified button system
 - forms.css - Form inputs, authentication
 - game-ui.css - Game-specific components
 - utilities.css - Badges, alerts, messages
 - responsive.css - All media queries consolidated
 -  Dynamic theme loading from era configs in App.js
 -  PurchasePage refactored to use CSS classes
 -  Removed hardcoded Midway theme from CSS files
-

HIGH PRIORITY (Current Sprint)

CSS Architecture Cleanup

☐ **Audit and consolidate duplicate CSS classes**

- Generic `.item` class instead of `.era-item` + `.opponent-item` when styling is identical
- Review all components for similar duplication patterns
- Maintain BEM where appropriate, but avoid purist over-complication
- Document decision criteria for when to use specific vs. generic classes

☐ **Eliminate remaining page-specific CSS files**

- Delete empty/redundant page CSS files (LaunchPage.css, etc.)
- Remove imports of deleted CSS files from page components
- Ensure all styling uses theme.css and component CSS files

☐ **Remove inline styles from all components**

- Audit all React components for `style={{...}}` usage
- Convert inline styles to CSS classes
- Update component versions after refactoring

☐ **CSS Documentation**

- Document when to use specific vs. generic classes
- Create CSS class usage guide for developers
- Add examples of proper BEM usage in project context

String/Message Elimination

- ☐ **Remove all hard-coded strings from code** (except console logs)
 - ☐ **Expand era config message coverage** - Ensure all user-facing text comes from era configs
 - ☐ **Update components to use `EraService.getMessage()`** - Replace remaining hard-coded strings
 - ☐ **Add missing message types to era configs** - Loading states, errors, confirmations, etc.
 - ☐ **Validation system** - Ensure era configs have required message types before deployment
-



MEDIUM PRIORITY (Next Sprint)

UI/UX Enhancements

- ☐ **Profile creation dialog** for new users without game names
- ☐ **Enhanced leaderboard** with multiple ranking types and recent champions
- ☐ **Game name usage throughout UI** (instead of emails where appropriate)
- ☐ **Boost visual indicators** in game UI (fire emoji decorations for attack/defense boosts)
- ☐ **Mobile responsiveness audit** - Test and fix mobile experience across all pages

Guest Player Experience

- ☐ **Guest player upgrade prompts** - Encourage registration after games
- ☐ **Guest statistics display** - Show what they're missing by not registering
- ☐ **Session persistence** - Save guest progress during session (if feasible)

Promotional System Improvements

- ☐ **Post-game promotional boxes** refinement
- ☐ **QR code voucher scanning** for mobile devices
- ☐ **Dynamic promotional content** from era configs (images, descriptions)

Animation & Visual Polish

- ☐ **Animation timing controls** - User-adjustable animation speeds
- ☐ **Sound effect integration** through action queue system
- ☐ **Visual effect expansion** - Particle effects, enhanced animations
- ☐ **Performance monitoring** - Animation frame rate optimization
- ☐ **Loading states** - Better feedback during async operations

LOWER PRIORITY (Future Releases)

New Era Development

- ☐ **Midway Island era completion** - Full testing and balancing
- ☐ **Pirates of the Gulf era** - Multiplayer alliances, irregular maps
- ☐ **Era-specific AI captain balancing** - Difficulty tuning per era
- ☐ **Additional historical eras** (Korean War, Gulf War, etc.)

Ship Visual System

- ☐ **AssetManager class** - PNG asset loading and caching
- ☐ **ShipRenderer class** - Ship visualization with damage overlays
- ☐ **Ship PNG assets** - Create artwork for all ship types across eras
- ☐ **Damage overlay system** - Visual damage indicators
- ☐ **Asset storage setup** - Organized folder structure in Supabase

Advanced Features

- ☐ **Real-time multiplayer** with WebSocket synchronization
- ☐ **Tournament system** with bracket management
- ☐ **Ship upgrade mechanics** with experience systems
- ☐ **Environmental effects** (storms, fog, mine fields)
- ☐ **Advanced AI** with machine learning adaptation
- ☐ **Replay system** - Watch completed games
- ☐ **Achievement system** - Unlock badges and titles

Technical Improvements

- ☐ **Network synchronization** for multiplayer action queues
- ☐ **WebRTC** for peer-to-peer multiplayer option
- ☐ **Redis** for real-time game state synchronization
- ☐ **3D visualization option** with Three.js integration
- ☐ **Progressive Web App** (PWA) support
- ☐ **Offline mode** capabilities

CODE QUALITY & TESTING

Unit Testing

- ☐ **CoreEngine tests** - State machine transitions
- ☐ **Game.js tests** - Combat mechanics, turn management
- ☐ **Service layer tests** - All service classes
- ☐ **Board.js tests** - Spatial logic, collision detection
- ☐ **Player statistics tests** - Calculation accuracy

Integration Testing

- ☐ **Full game flow tests** - Launch to game completion
- ☐ **Purchase flow tests** - Stripe + voucher redemption
- ☐ **Guest player flow tests** - Session management
- ☐ **Alliance system tests** - Multi-player scenarios

Performance Optimization

- ☐ **React component profiling** - Identify unnecessary re-renders
- ☐ **Asset loading optimization** - Lazy loading, caching strategies
- ☐ **Database query optimization** - Leaderboard performance
- ☐ **Mobile performance** - Touch responsiveness, render optimization
- ☐ **Memory leak detection** - Long session stability

Documentation

- ☐ **Component documentation** - JSDoc for all classes
 - ☐ **API documentation** - Service layer interfaces
 - ☐ **Integration guide** - How to add new eras
 - ☐ **Developer onboarding** - Setup and contribution guide
 - ☐ **Asset guidelines** - Creating ship PNGs and terrain
-

DEFERRED (Working But Could Be Better)

Code Organization

- ☐ **Game.js refactoring** - Statistics extraction to service (HIGH RISK - working well currently)
 - ☐ **Alliance management extraction** - Separate class (not urgent)
 - ☐ **Component consolidation** - Where beneficial (careful review needed)
 - ☐ **Archive old/unused code** - Clean up deprecated files
-



NOTES & CONSIDERATIONS

Critical Reminders

- **NEVER initialize statistics outside Player.js constructor** - Single source of truth
- **Always use BEM CSS methodology** - But avoid purist over-complication
- **No inline styles** - All styling in CSS files
- **Always increment version numbers** when making changes
- **Check ID prefix for guest/AI detection** - `userId.startsWith('guest-')` pattern
- **Focus on 30-minute rich gameplay sessions** - Turn-based strategic depth

High Risk Areas (Approach with Caution)

- **CoreEngine state machine** - Currently working well, synchronous core is critical
- **Turn management** - Action queue system functioning properly
- **Hit resolution** - Alliance rules and scoring working correctly
- **Statistics system** - Single source of truth in Player.js, don't break it

CSS Architecture Principles

- **Mobile-first responsive design** - Base styles for mobile, enhance for larger screens
- **BEM methodology** - `.block__element--modifier` but be pragmatic
- **Generic over specific** - Use `.item` instead of `.era-item` + `.opponent-item` when appropriate
- **Theme-driven** - Era themes loaded dynamically from configs
- **No inline styles** - Exception: none allowed

Questions to Resolve

- ☐ Should boost mechanics (attack/defense bonuses) be implemented now or deferred?
 - ☐ What's the priority for 3D visualization vs. other features?
 - ☐ When to implement real-time multiplayer vs. async turn-based multiplayer?
 - ☐ Asset creation timeline - in-house or contract artist?
-

End of To-Do List