Offline 2D Gamified Classroom - Development Guidelines

1. Development Environment Setup

1.1 Prerequisites

- **Node.js (v16+)**: Core JavaScript runtime
- **npm/yarn**: Package management
- **Git**: Version control
- **VS Code** (recommended): IDE with extensions:
  - ESLint
  - Prettier
  - Phaser Editor
  - Tiled Map Editor integration

1.2 Initial Setup

bash

```bash
# Create project directory
mkdir offline-classroom-game
cd offline-classroom-game

# Initialize project
npm init -y

# Install core dependencies
npm install phaser@3.55.2 localforage@1.10.0

# Install development dependencies
npm install --save-dev webpack webpack-cli webpack-dev-server typescript ts-loader html-webpack

# Initialize Git repository
git init
echo "node_modules/\ndist/\n.DS_Store" > .gitignore
```

1.3 Configuration Files

- **package.json**: Define scripts for build, dev, and test
- **webpack.config.js**: Configure module bundling
- **tsconfig.json**: TypeScript configuration (if using TS)
- **.eslintrc**: Code linting rules
- **.prettierrc**: Code formatting rules

2. Coding Standards

2.1 General Principles

- **Modularity**: Use modules and components for reusability
- **Separation of Concerns**: Keep game logic, data management, and UI separate
- **Progressive Enhancement**: Build core functionality first, then enhance
- **Error Handling**: Gracefully handle errors, especially for offline operations
- **Documentation**: Document all methods, components, and complex logic

2.2 File and Folder Structure

- Follow the project structure provided in the Project Directory Structure document
- Use consistent naming conventions:
  - **PascalCase**: Classes, Components, Types
  - **camelCase**: Variables, functions, instances
  - **kebab-case**: Files, directories
  - **UPPER_SNAKE_CASE**: Constants

2.3 JavaScript/TypeScript Standards

- Use ES6+ features but ensure compatibility with target browsers
- Prefer `const` over `let` when variables won't be reassigned
- Use async/await for asynchronous operations
- Implement proper error handling with try/catch
- Use JSDoc comments for all functions and classes
- Avoid global variables and namespace pollution

2.4 Phaser Specific Guidelines

- Structure game as multiple scenes extending `Phaser.Scene`
- Use Phaser's built-in physics, input, and animation systems
- Preload assets in dedicated preload scenes
- Properly destroy objects when scenes are shut down
- Use Phaser's event system for communication between objects
- Avoid direct DOM manipulation where possible, use Phaser's API

3. Asset Management

3.1 Asset Organization

- Organize assets by type (images, audio, maps)

- Further categorize by purpose (characters, tiles, UI, etc.)
- Use consistent naming conventions for all assets
- Create an asset manifest for easier management

3.2 Asset Optimization

- Optimize images for web (compression, appropriate format)
- Use sprite sheets for related images
- Configure texture atlases for efficient rendering
- Keep audio files small and use appropriate formats
- Consider lazy loading for non-essential assets

3.3 Asset Loading

- Implement loading screens with progress indicators
- Prioritize essential assets for initial load
- Use asset packs for organized loading
- Cache assets appropriately for offline use
- Implement error handling for failed asset loads

4. Offline Functionality

4.1 Service Worker Implementation

- Register service worker in main entry point
- Configure caching strategies:
  - Cache-first for static assets
  - Network-first with cache fallback for dynamic content
  - Cache-only for offline-specific resources
- Implement versioning for cache updates
- Handle service worker updates and notifications

4.2 IndexedDB Usage

- Use localForage as wrapper for IndexedDB
- Create separate stores for different data types
- Implement proper error handling for all database operations
- Use versioning for schema upgrades
- Monitor storage usage and implement cleanup strategies

4.3 Network Status Management

- Detect online/offline status changes

- Update UI based on connectivity status
- Queue operations when offline for later sync
- Implement timeout and retry mechanisms for network requests
- Provide clear feedback to users about sync status

5. User Interface Development

5.1 UI Components

- Create reusable UI components when possible
- Use consistent styling and interaction patterns
- Implement responsive layouts for different screen sizes
- Consider accessibility in all UI design
- Use Phaser's built-in UI components or HTML overlays as appropriate

5.2 Game World Interface

- Implement clear visual cues for interactive objects
- Design intuitive player movement and interaction
- Use consistent feedback for player actions
- Implement mini-maps or navigation aids as needed
- Balance game aesthetics with educational clarity

5.3 Educational Content Display

- Create flexible, reusable resource viewers
- Implement progress indicators for resources
- Design engaging quiz interfaces with clear feedback
- Support multiple content formats (text, video, interactive)
- Ensure content is accessible and properly formatted

6. Testing

6.1 Unit Testing

- Write tests for core functionality and utilities
- Mock dependencies for isolated testing
- Test offline functionality in controlled environments
- Use testing framework like Jest with appropriate mocks
- Automate testing in build pipeline

6.2 Integration Testing

- Test interactions between major components
- Validate data flow across the application
- Test online/offline transitions
- Verify sync functionality works correctly
- Test with various network conditions

6.3 User Testing

- Conduct usability testing with target audience
- Test on various devices and screen sizes
- Validate educational effectiveness
- Gather feedback on user experience
- Iterate based on testing results

7. Performance Optimization

7.1 Rendering Performance

- Use appropriate rendering techniques (WebGL vs Canvas)
- Implement object pooling for frequently created/destroyed objects
- Use sprite batching where possible
- Implement culling for off-screen objects
- Monitor and optimize frame rate

7.2 Memory Management

- Properly destroy objects and clear references
- Monitor memory usage with dev tools
- Implement texture cleanup for unused assets
- Optimize storage usage in IndexedDB
- Handle low memory situations gracefully

7.3 Battery and Network Efficiency

- Minimize network requests
- Batch database operations
- Implement sleep mode when app is inactive
- Reduce animations and updates when appropriate
- Optimize sync operations to minimize data transfer

8. Security Considerations

8.1 Data Security

- Don't store sensitive information in localStorage
- Encrypt sensitive data in IndexedDB if necessary
- Implement proper authentication for sync operations
- Validate all data before storage and usage
- Handle user data according to privacy best practices

8.2 Content Security

- Validate and sanitize user-generated content
- Implement appropriate content restrictions for educational context
- Secure any communications between users
- Use Content Security Policy for web resources
- Validate external resources before loading

9. Deployment

9.1 Build Process

- Configure production build with optimizations
- Minimize and bundle JavaScript code
- Optimize and compress assets
- Generate appropriate manifests and service worker files
- Run automated tests before deployment

9.2 Distribution

- Configure proper caching headers for server
- Set up CDN for static assets if applicable
- Implement version control in distribution
- Create update mechanism for existing installations
- Prepare offline distribution package if needed

9.3 Monitoring

- Implement error logging and reporting
- Track key performance metrics
- Monitor user engagement and educational metrics
- Set up alerts for critical issues
- Plan for ongoing maintenance and updates