```
function monitorClaudeUI() { // Check if we're on Claude.ai if (!window.location.href.includes('claude.ai')) { return; }

// Wait for Claude UI to be fully loaded const checkForClaudeUI = setInterval(() => { const claudeContainer = document.querySelector('.claude-container'); if (claudeContainer) { clearInterval(checkForClaudeUI); console.log('Claude UI detected, initializing sidebar'); domManipulator.renderProjects(); } }, 1000); }

// Start initialization initialize();
```

4. Document Creation Process Implementation

4.1 Document Context Injection

The core of our system is the ability to inject specialized context into Claude to guide the document creation process. Here's a detailed implementation:

```typescript
// src/content/document-creation.ts
import { DocumentType } from '../shared/types';

export class DocumentCreationManager {
  private currentDocumentId: string | null = null;
  private currentProjectId: string | null = null;
  private conversationState: 'asking_questions' | 'drafting' | 'refining' | 'completed' =
'asking_questions';
  private questionsAsked: number = 0;
  private responsesReceived: number = 0;

  // Start document creation process
  public startDocumentCreation(projectId: string, documentId: string, context: string): void
{
    this.currentProjectId = projectId;
    this.currentDocumentId = documentId;
    this.conversationState = 'asking_questions';
    this.questionsAsked = 1; // Initial context counts as first question
    this.responsesReceived = 0;

    // Inject initial context to Claude
    this.injectClaudePrompt(context);

    // Start monitoring for Claude's responses
    this.startResponseMonitoring();
  }

  // Inject prompt into Claude input field
  private injectClaudePrompt(prompt: string): void {
    // Find Claude's input field - actual selector may vary
    const inputField = document.querySelector('textarea.claude-input');
    if (!inputField) {
      console.error('Claude input field not found');
      return;
    }

    // Set value and dispatch input event
    (inputField as HTMLTextAreaElement).value = prompt;
    inputField.dispatchEvent(new Event('input', { bubbles: true }));

    // Find and click submit button
    const submitButton = document.querySelector('button.claude-submit-button');
    if (!submitButton) {
      console.error('Claude submit button not found');
      return;
    }

    (submitButton as HTMLButtonElement).click();
    this.questionsAsked++;
  }

  // Monitor for Claude's responses
  private startResponseMonitoring(): void {
    // Create a MutationObserver to watch for new messages
    const observer = new MutationObserver((mutations) => {
      // Check if a new Claude response has been added
      for (const mutation of mutations) {
        if (mutation.type === 'childList' && mutation.addedNodes.length > 0) {
          const responseElements = document.querySelectorAll('.claude-response-message');

          // If we have a new response
          if (responseElements.length > this.responsesReceived) {
            this.responsesReceived = responseElements.length;
            this.handleNewResponse(responseElements[responseElements.length - 1] as
HTMLElement);
          }
        }
      }
    });

    // Start observing the message container
    const messageContainer = document.querySelector('.claude-message-container');
    if (messageContainer) {
      observer.observe(messageContainer, { childList: true, subtree: true });
    }
  }

  // Handle a new response from Claude
  private handleNewResponse(responseElement: HTMLElement): void {
    const responseText = responseElement.textContent || '';

    // Update conversation state based on response content
    this.updateConversationState(responseText);

    // If the response contains a document draft
    if (this.conversationState === 'drafting' || this.conversationState === 'refining') {
      this.captureDocumentDraft(responseText);
    }

    // If we need to transition to drafting
    if (this.questionsAsked >= 5 && this.conversationState === 'asking_questions') {
      this.transitionToDrafting();
    }
  }

  // Update conversation state based on response content
  private updateConversationState(responseText: string): void {
    // Check if the response contains drafting indicators
    if (responseText.includes('Here\'s a draft of your document') ||
        responseText.includes('Draft Document:') ||
        responseText.includes('# ') && responseText.includes('## ')) {
      this.conversationState = 'drafting';
    }

    // Check if refinement is happening
    if (responseText.includes('I\'ve updated the document') ||
        responseText.includes('Here\'s the revised')) {
      this.conversationState = 'refining';
    }
  }

  // Transition from question phase to drafting phase
  private transitionToDrafting(): void {
    const draftPrompt = "Based on our conversation so far, please create a comprehensive
draft of the document. Please format it as a Markdown document with appropriate headers,
sections, and formatting.";
    this.injectClaudePrompt(draftPrompt);
    this.conversationState = 'drafting';
  }

  // Capture document draft from Claude's response
  private captureDocumentDraft(responseText: string): void {
    // Extract the document content from the response
    // This is a simple approach - in practice, you'd want more sophisticated extraction
    let documentContent = responseText;

    // If the response has preamble text, try to find where the actual document starts
    const docMarkers = [
      'Here\'s a draft of your document:',
      'Draft Document:',
      '# ',
      '## '
    ];

    for (const marker of docMarkers) {
      const markerIndex = documentContent.indexOf(marker);
      if (markerIndex !== -1) {
        documentContent = documentContent.substring(markerIndex);
        break;
      }
    }

    // Send captured content to background script
    chrome.runtime.sendMessage({
      type: 'capture_document_content',
      payload: {
        projectId: this.currentProjectId,
        documentId: this.currentDocumentId,
        content: documentContent
      }
    }, (response) => {
      if (response.success) {
        this.showDocumentSavedNotification();
      } else {
        console.error('Failed to save document content:', response.error);
      }
    });
  }

  // Show notification that document was saved
  private showDocumentSavedNotification(): void {
    // Create notification element
    const notification = document.createElement('div');
    notification.className = 'gdds-notification';
    notification.innerHTML = `
      <div class="gdds-notification-content">
```

```
        <span class="gdds-icon success">✓</span>
        <span class="gdds-notification-text">Document draft saved!</span>
        <button class="gdds-button small">View</button>
      </div>
    `;

    // Add to page
    document.body.appendChild(notification);

    // Setup view button
    const viewButton = notification.querySelector('button');
    if (viewButton) {
      viewButton.addEventListener('click', () => {
        this.viewSavedDocument();
        document.body.removeChild(notification);
      });
    }

    // Auto-remove after 5 seconds
    setTimeout(() => {
      if (document.body.contains(notification)) {
        document.body.removeChild(notification);
      }
    }, 5000);
  }

  // View the saved document
  private viewSavedDocument(): void {
    // Implement view functionality
  }
}

// Export singleton instance
export const documentCreationManager = new DocumentCreationManager();
```

4.2 Document Export Implementation

typescript

```typescript
// src/shared/document-exporter.ts
import { saveAs } from 'file-saver';
import { marked } from 'marked';
import { pdf } from 'pdf-lib';

export class DocumentExporter {
  // Export document as Markdown
  public exportMarkdown(title: string, content: string): void {
    const blob = new Blob([content], { type: 'text/markdown;charset=utf-8' });
    saveAs(blob, `${this.sanitizeFilename(title)}.md`);
  }

  // Export document as plain text
  public exportText(title: string, content: string): void {
    // Strip Markdown formatting for plain text
    const plainText = content
      .replace(/#{1,6}\s/g, '') // Remove headers
      .replace(/\*\*/g, '')      // Remove bold
      .replace(/\*/g, '')        // Remove italic
      .replace(/\[([^\]]+)\]\(([^)]+\)/g, '$1') // Replace links with just text
      .replace(/```[^`]*```/g, '') // Remove code blocks
      .replace(/`([^`]+)`/g, '$1'); // Remove inline code

    const blob = new Blob([plainText], { type: 'text/plain;charset=utf-8' });
    saveAs(blob, `${this.sanitizeFilename(title)}.txt`);
  }

  // Export document as PDF
  public async exportPDF(title: string, content: string): Promise<void> {
    // Convert Markdown to HTML
    const html = marked(content);

    // Create PDF document with basic styling
    const pdfDoc = await pdf.create();
    const page = pdfDoc.addPage();

    // Note: This is a simplified approach
    // In a real implementation, you would want to use a more sophisticated
    // PDF generation library that can properly render HTML to PDF
    // This might require a server component or a more advanced library

    // For now, this is a placeholder that would just add basic text
    page.drawText(`${title}\n\n${content}`, {
```

```
      x: 50,
      y: page.getHeight() - 50,
      size: 12
    });

    const pdfBytes = await pdfDoc.save();
    const blob = new Blob([pdfBytes], { type: 'application/pdf' });
    saveAs(blob, `${this.sanitizeFilename(title)}.pdf`);
  }

  // Export to Google Docs (requires Google Drive integration)
  public async exportToGoogleDocs(title: string, content: string): Promise<string> {
    // This would require Google Drive API integration
    // Here's a simplified outline of what this would involve

    // 1. Get OAuth token
    const token = await this.getGoogleAuthToken();

    // 2. Convert content to Google Docs format
    // Google Docs API expects a specific format for document content
    const docContent = this.convertToGoogleDocsFormat(content);

    // 3. Make API request to create a new document
    const response = await fetch('https://docs.googleapis.com/v1/documents', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        title: title,
        body: {
          content: docContent
        }
      })
    });

    const data = await response.json();

    // 4. Return the document URL
    return `https://docs.google.com/document/d/${data.documentId}/edit`;
  }

  // Helper method to get Google auth token
  private async getGoogleAuthToken(): Promise<string> {
    return new Promise((resolve, reject) => {
      chrome.identity.getAuthToken({ interactive: true }, (token) => {
        if (chrome.runtime.lastError) {
          reject(chrome.runtime.lastError);
        } else {
          resolve(token);
        }
      });
    });
  }

  // Helper method to convert Markdown to Google Docs format
  private convertToGoogleDocsFormat(markdown: string): any {
    // This would be a complex conversion process
    // For a real implementation, you might use a library or service

    // Simplified placeholder that would need to be expanded
    return {
      // Google Docs API-specific format
    };
  }

  // Helper method to sanitize filename
  private sanitizeFilename(filename: string): string {
    return filename
      .replace(/[/\\?%*:|"<>]/g, '-')
      .trim();
  }
}

// Export singleton instance
export const documentExporter = new DocumentExporter();
```

4.3 Sidebar CSS Implementation

css

```css
/* src/content/sidebar.css */
.gdds-sidebar {
  position: fixed;
  top: 0;
  right: 0;
  width: 300px;
  height: 100vh;
  background-color: #2a2a2a;
  color: #f0f0f0;
  box-shadow: -2px 0 10px rgba(0, 0, 0, 0.2);
  z-index: 1000;
  transition: transform 0.3s ease;
  display: flex;
  flex-direction: column;
  font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
}

.gdds-sidebar.collapsed {
  transform: translateX(290px);
}

.gdds-sidebar-header {
  padding: 15px;
  display: flex;
  align-items: center;
  justify-content: space-between;
  border-bottom: 1px solid #3e3e3e;
}

.gdds-sidebar-header h3 {
  margin: 0;
  font-size: 16px;
  font-weight: 600;
}

.gdds-content {
  flex: 1;
  overflow-y: auto;
  padding: 10px 0;
}

.gdds-projects-list {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.gdds-project-item {
  background-color: #363636;
  border-radius: 6px;
  margin: 0 10px;
  overflow: hidden;
}

.gdds-project-header {
  padding: 12px 15px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  cursor: pointer;
}

.gdds-project-header h4 {
  margin: 0;
  font-size: 14px;
  font-weight: 500;
}

.gdds-project-documents {
  padding: 10px;
  background-color: #2f2f2f;
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.gdds-document-item {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 8px 12px;
  border-radius: 4px;
  font-size: 13px;
  background-color: #404040;
}

.gdds-document-item.status-not_started {
```

```css
  opacity: 0.7;
}

.gdds-document-item.status-in_progress {
  border-left: 3px solid #ffc107;
}

.gdds-document-item.status-completed {
  border-left: 3px solid #4caf50;
}

.gdds-document-info {
  display: flex;
  flex-direction: column;
  gap: 2px;
}

.gdds-document-title {
  font-weight: 500;
}

.gdds-document-status {
  font-size: 11px;
  opacity: 0.7;
}

.gdds-document-actions {
  display: flex;
  gap: 5px;
}

.gdds-button {
  background-color: #525252;
  color: #ffffff;
  border: none;
  border-radius: 4px;
  padding: 8px 12px;
  font-size: 13px;
  cursor: pointer;
  transition: background-color 0.2s;
}

.gdds-button:hover {
  background-color: #676767;
```

```css
}

.gdds-button.primary {
  background-color: #8c52ff;
}

.gdds-button.primary:hover {
  background-color: #9d6aff;
}

.gdds-button.secondary {
  background-color: transparent;
  border: 1px solid #525252;
}

.gdds-button.secondary:hover {
  background-color: rgba(255, 255, 255, 0.1);
}

.gdds-button.small {
  padding: 4px 8px;
  font-size: 12px;
}

.gdds-button.icon-only {
  padding: 4px;
  display: flex;
  align-items: center;
  justify-content: center;
}

.gdds-icon {
  font-size: 14px;
  line-height: 1;
}

.gdds-empty-state {
  padding: 20px;
  text-align: center;
  opacity: 0.7;
  font-size: 14px;
}

.gdds-dialog {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 2000;
}

.gdds-dialog-content {
  background-color: #2a2a2a;
  border-radius: 8px;
  padding: 20px;
  width: 400px;
  max-width: 90%;
}

.gdds-dialog-content h3 {
  margin-top: 0;
  font-size: 18px;
}

.gdds-form-group {
  margin-bottom: 15px;
}

.gdds-form-group label {
  display: block;
  margin-bottom: 5px;
  font-size: 14px;
}

.gdds-form-group input {
  width: 100%;
  padding: 8px 10px;
  border-radius: 4px;
  border: 1px solid #4a4a4a;
  background-color: #333333;
  color: #f0f0f0;
  font-size: 14px;
}
```

```css
.gdds-form-actions {
  display: flex;
  justify-content: flex-end;
  gap: 10px;
  margin-top: 20px;
}

.gdds-notification {
  position: fixed;
  bottom: 20px;
  right: 320px;
  background-color: #333333;
  border-radius: 6px;
  padding: 12px 15px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);
  z-index: 1500;
  transition: transform 0.3s, opacity 0.3s;
  animation: slideIn 0.3s;
}

.gdds-notification-content {
  display: flex;
  align-items: center;
  gap: 10px;
}

.gdds-icon.success {
  color: #4caf50;
}

.gdds-notification-text {
  font-size: 14px;
}

@keyframes slideIn {
  from {
    transform: translateX(50px);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
    }
  }
```

5. Google Drive Integration

For cloud storage and sharing capabilities, we'll implement Google Drive integration:

typescript

```
// src/shared/google-drive.ts
export class GoogleDriveManager {
  // Get OAuth token
  private async getAuthToken(): Promise<string> {
    return new Promise((resolve, reject) => {
      chrome.identity.getAuthToken({ interactive: true }, (token) => {
        if (chrome.runtime.lastError) {
          reject(chrome.runtime.lastError);
        } else {
          resolve(token);
        }
      });
    });
  }

  // Check if user is authenticated
  public async isAuthenticated(): Promise<boolean> {
    try {
      await this.getAuthToken();
      return true;
    } catch (error) {
      return false;
    }
  }

  // Create folder for project
  public async createProjectFolder(projectName: string): Promise<string> {
    const token = await this.getAuthToken();

    const response = await fetch('https://www.googleapis.com/drive/v3/files', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        name: `Game Docs - ${projectName}`,
        mimeType: 'application/vnd.google-apps.folder'
      })
    });

    const data = await response.json();
    return data.id;
```

```
}

// Create or update document in Google Drive
public async saveDocument(
  documentTitle: string,
  content: string,
  folderId: string,
  existingFileId?: string
): Promise<string> {
  const token = await this.getAuthToken();

  // If updating existing file
  if (existingFileId) {
    await fetch(`https://www.googleapis.com/upload/drive/v3/files/${existingFileId}`, {
      method: 'PATCH',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'text/markdown'
      },
      body: content
    });

    return existingFileId;
  }

  // Create metadata for new file
  const metadataResponse = await fetch('https://www.googleapis.com/drive/v3/files', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      name: `${documentTitle}.md`,
      parents: [folderId],
      mimeType: 'text/markdown'
    })
  });

  const fileData = await metadataResponse.json();
  const fileId = fileData.id;

  // Upload content to the file
  await fetch(`https://www.googleapis.com/upload/drive/v3/files/${fileId}?uploadType=media`,
    method: 'PATCH',
    headers: {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'text/markdown'
    },
    body: content
  });

  return fileId;
}

// Convert document to Google Docs format
public async convertToGoogleDocs(fileId: string): Promise<string> {
  const token = await this.getAuthToken();

  // Copy the file and convert to Google Docs format
  const response = await fetch('https://www.googleapis.com/drive/v3/files', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      name: null, // Use same name as source file
      parents: null, // Use same parent as source file
      mimeType: 'application/vnd.google-apps.document'
    })
  });

  const data = await response.json();
  return data.id;
}

// Get sharing link for document
public async getShareableLink(fileId: string): Promise<string> {
  const token = await this.getAuthToken();

  // Create a sharing permission
  await fetch(`https://www.googleapis.com/drive/v3/files/${fileId}/permissions`, {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json'
    },
      body: JSON.stringify({
        role: 'reader',
        type: 'anyone'
      })
    });

    // Get the file's web view link
    const response = await fetch(`https://www.googleapis.com/drive/v3/files/${fileId}?fields=we
      headers: {
        'Authorization': `Bearer ${token}`
      }
    });

    const data = await response.json();
    return data.webViewLink;
  }
}

// Export singleton instance
export const googleDriveManager = new GoogleDriveManager();
```

6. Testing and Debugging

6.1 Debugging Chrome Extension

typescript

```typescript
// src/shared/logger.ts
export enum LogLevel {
  Debug = 0,
  Info = 1,
  Warning = 2,
  Error = 3
}

export class Logger {
  private level: LogLevel = LogLevel.Info;

  // Set Logging Level
  public setLevel(level: LogLevel): void {
    this.level = level;
  }

  // Debug Log
  public debug(message: string, ...data: any[]): void {
    if (this.level <= LogLevel.Debug) {
      console.debug(`[GDDS Debug] ${message}`, ...data);
    }
  }

  // Info Log
  public info(message: string, ...data: any[]): void {
    if (this.level <= LogLevel.Info) {
      console.info(`[GDDS Info] ${message}`, ...data);
    }
  }

  // Warning Log
  public warn(message: string, ...data: any[]): void {
    if (this.level <= LogLevel.Warning) {
      console.warn(`[GDDS Warning] ${message}`, ...data);
    }
  }

  // Error Log
  public error(message: string, ...data: any[]): void {
    if (this.level <= LogLevel.Error) {
      console.error(`[GDDS Error] ${message}`, ...data);
    }
  }
  // Log DOM structure for debugging
  public logDOMSnapshot(selector: string): void {
    if (this.level <= LogLevel.Debug) {
      const element = document.querySelector(selector);
      if (element) {
        console.log(`[GDDS DOM Snapshot] ${selector}:`, element.cloneNode(true));
      } else {
        console.log(`[GDDS DOM Snapshot] Element not found: ${selector}`);
      }
    }
  }
}

// Export singleton instance
export const logger = new Logger();
```

6.2 Testing Extension Development Build

bash

```bash
# To build the extension for testing:
npm run build

# To watch for changes during development:
npm run watch
```

To load the extension for testing:

1. Open Chrome and navigate to `chrome://extensions/`
2. Enable "Developer mode" (toggle in top-right)
3. Click "Load unpacked" and select the `dist` folder
4. Navigate to Claude.ai to test the extension

7. Deployment Preparation

7.1 Production Build Configuration

javascript

```js
// webpack.config.prod.js
const path = require('path');
const CopyPlugin = require('copy-webpack-plugin');
const TerserPlugin = require('terser-webpack-plugin');
const CssMinimizerPlugin = require('css-minimizer-webpack-plugin');

module.exports = {
  mode: 'production',
  entry: {
    background: './src/background/background.ts',
    content: './src/content/content.ts',
    popup: './src/popup/popup.ts'
  },
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        use: 'ts-loader',
        exclude: /node_modules/
      },
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader']
      }
    ]
  },
  resolve: {
    extensions: ['.tsx', '.ts', '.js']
  },
  output: {
    filename: '[name].js',
    path: path.resolve(__dirname, 'dist')
  },
  optimization: {
    minimize: true,
    minimizer: [
      new TerserPlugin({
        terserOptions: {
          format: {
            comments: false,
          },
        },
        extractComments: false,
      }),
      new CssMinimizerPlugin(),
    ],
  },
  plugins: [
    new CopyPlugin({
      patterns: [
        { from: 'manifest.json', to: '.' },
        { from: 'src/popup/popup.html', to: '.' },
        { from: 'src/assets', to: '.' },
        { from: 'src/content/sidebar.css', to: '.' }
      ]
    })
  ]
};
```

7.2 Chrome Web Store Preparation

```
# Package extension for Chrome Web Store
zip -r game-doc-system.zip dist/
```

Required assets for Chrome Web Store submission:

1. Extension zip file
2. Icon images (128x128, 48x48, 16x16)
3. Screenshots of the extension in action (1280x800 or 640x400)
4. Promotional images (optional):
   - Small promo tile (440x280)
   - Large promo tile (920x680)
   - Marquee promo tile (1400x560)

8. Additional Resources

8.1 Chrome Extension Development Resources

- Chrome Extension Documentation
- Chrome Extension Manifest V3
- Chrome Identity API

8.2 Google Drive API Resources

- Google Drive API Documentation
- Google Drive REST API Reference

8.3 Recommended Libraries

- Marked.js - Markdown parsing and rendering
- PDF-lib - PDF generation
- FileSaver.js - Client-side file saving
- DOMPurify - HTML sanitization

9. Troubleshooting Common Issues

9.1 Claude.ai DOM Structure Changes

Claude.ai may update their UI structure, breaking the extension. If this happens:

1. Update the DOM selectors in `dom-manipulator.ts`
2. Check for changes in the message container structure
3. Update the response monitoring logic if needed

9.2 Google Drive API Quota Limits

Google Drive API has usage quotas. If you hit limits:

1. Implement rate limiting in the `GoogleDriveManager` class
2. Add retry logic with exponential backoff
3. Consider batch operations for multiple document updates

9.3 Chrome Extension Permissions

If the extension isn't working properly:

1. Check required permissions in `manifest.json`
2. Ensure host permissions include `https://claude.ai/*`
3. Verify OAuth scopes for Google Drive integration

10. Next Steps and Future Enhancements

After implementing the core functionality described in this technical guide, consider these enhancements:

1. **Template Management** - Allow users to create and save document templates
2. **Advanced Export Options** - Support for additional export formats

3. **Collaborative Editing** - Real-time collaboration features
4. **Analytics Dashboard** - Track document creation metrics and progress
5. **Custom Document Types** - Allow users to define their own document types
6. **Integration with Game Engines** - Export documentation for Unity, Unreal, etc.

This technical implementation guide provides a comprehensive blueprint for building the Game Development Document System as a Chrome extension that enhances Claude.ai with game documentation capabilities.