MANUALE TECNICO DI COLLABORAZIONE LLM PER THE SAFE PLACE (Revisione Completa per v0.7.23 "Data Unchained")

1. INTRODUZIONE E SCOPO DEL MANUALE

- **Scopo:** Questo documento serve come guida tecnica di riferimento per l'LLM (Cursor) durante lo sviluppo del videogioco "The Safe Place". Fornisce un contesto persistente sull'architettura del codice, le logiche implementate, le strutture dati chiave e lo stato attuale del progetto alla versione **v0.7.23 "Data Unchained"**.
- **Utilizzo da Parte dell'LLM:**
    - Consultare questo manuale PRIMA di proporre o applicare modifiche al codice.
    - Usarlo come base per comprendere le interdipendenze tra i moduli.
    - Fare riferimento a specifiche sezioni quando vengono richieste modifiche a funzionalità esistenti.
    - Aiutare a mantenere la coerenza con le convenzioni e le logiche già implementate.
    - Comprendere e rispettare la suddivisione dei file di dati introdotta con la v0.7.23.

2. PRINCIPI GUIDA PER L'LLM NELLO SVILUPPO

- **Mantenere la Modularità:** Rispettare la suddivisione dei file JS esistente, inclusa la nuova struttura dei file di dati (`game_data_misc.js`, `game_data_events.js`, `game_data_items.js`, `game_data_recipes.js`). Se si aggiungono nuove funzionalità complesse, discutere con il designer (Simone Pizzi) la possibile creazione di nuovi moduli.
- **Priorità ai Bug:** Nella fase attuale, la risoluzione dei bug noti e la stabilizzazione delle funzionalità esistenti hanno la priorità sulla creazione di nuove meccaniche complesse.
- **Modifiche Esplicite:** Quando si applicano modifiche, essere il più specifici possibile: indicare chiaramente il file, la funzione e, se possibile, le righe di codice interessate.
- **Conferma Visuale:** Dopo aver applicato una modifica, mostrare sempre al designer la porzione di codice modificata per una verifica umana.
- **Log di Debug:** Utilizzare `if (typeof DEBUG_MODE !== 'undefined' && DEBUG_MODE) console.log(...);` per i log diagnostici temporanei. Rimuoverli o commentarli una volta che il problema è risolto, su indicazione del designer.
- **Convenzioni:** Seguire le convenzioni di denominazione (`camelCase` per variabili e funzioni, `UPPER_CASE` per costanti) e lo stile di scrittura del codice esistente.
- **Non Reinventare:** Prima di scrivere nuovo codice per una funzionalità apparentemente mancante, verificare se esiste già una logica simile o una funzione helper che può essere riutilizzata o adattata, consultando questo manuale o chiedendo al designer.
- **Attenzione ai File di Dati:** Prestare particolare attenzione quando si modificano i file `game_data_*.js` per mantenere la corretta struttura degli oggetti e la coerenza dei

dati. La modifica di un `itemId` o di una `recipeKey` richiede attenzione per aggiornare tutti i riferimenti.

3. ARCHITETTURA GENERALE DEL CODICE (v0.7.23)

- **3.1. HTML (`index.html`):**
    - **Ruolo:** Definisce la struttura semantica dell'interfaccia utente (pannelli principali, overlay per eventi/azioni/crafting, popup, schermate di menu/gioco/fine). Carica tutti i file CSS e JavaScript.
    - **Ordine Caricamento JS (Critico e Aggiornato):**
        1. `js/game_constants.js` (Definizioni variabili globali e costanti di alto livello)
        2. `js/game_data_misc.js` (Dati statici: simboli mappa, descrizioni, lore, messaggi stato, ecc.)
        3. `js/game_data_events.js` (Dati statici: `EVENT_DATA`, `dilemmaEvents`)
        4. `js/game_data_items.js` (Dati statici: `ITEM_DATA`)
        5. `js/game_data_recipes.js` (Dati statici: `CRAFTING_RECIPES`)
        6. `js/game_utils.js` (Funzioni di utilità generiche)
        7. `js/dom_references.js` (Riferimenti agli elementi del DOM)
        8. `js/ui.js` (Logica di rendering dell'interfaccia utente)
        9. `js/player.js` (Logica del personaggio giocante, inventario, azioni)
        10. `js/events.js` (Logica di trigger e risoluzione degli eventi)
        11. `js/map.js` (Logica della mappa, movimento, ciclo giorno/notte)
        12. `js/game_core.js` (Orchestratore principale, inizializzazione, input)
    - *Nota: `js/game_data.js` è ora un file placeholder vuoto, il suo contenuto è stato distribuito nei nuovi file `game_data_*.js`.*

- **3.2. CSS (Cartella `css/`):**
    - **Modularità:** 9 file CSS (`base.css`, `layout.css`, `panels.css`, `map.css`, `controls.css`, `events.css`, `tooltip.css`, `utilities.css`, `responsive.css`). Ognuno gestisce aspetti stilistici specifici dell'interfaccia. La coerenza visiva retro-terminale è un obiettivo primario.

- **3.3. JavaScript (Cartella `js/`):**
    - **Modularità:** 12 file JS attivi (escluso `game_data.js` se completamente vuoto), ognuno con responsabilità logiche distinte come descritto sopra nell'ordine di caricamento. La comunicazione avviene principalmente tramite variabili globali (definite in `game_constants.js`) e chiamate di funzione dirette tra moduli.

(Parte 2: Dettaglio dei Moduli JavaScript)

4. DETTAGLIO DEI MODULI JAVASCRIPT (v0.7.23)

- **4.1. `js/game_constants.js`**
    - **Responsabilità Primaria:** Dichiara le variabili di stato globali del gioco (usando `let`) che verranno modificate durante il gameplay e definisce le costanti numeriche, stringhe e probabilistiche (`const`) di alto livello utilizzate da tutta la logica di gioco.
    - **Variabili Globali Chiave (esempi `let`):** `player`, `map`, `messages`, `gameActive`, `eventScreenActive`, `gamePaused`, `isDay`, `dayMovesCounter`, `nightMovesCounter`, `daysSurvived`, `currentEventContext`, `currentEventChoices`, `easterEggPixelDebhFound`, `uniqueEventWebRadioFound`.
    - **Costanti Fondamentali (esempi `const`):** `GAME_NAME`, `GAME_VERSION_NUMBER`, `GAME_VERSION_NAME`, `DEBUG_MODE`, `MAP_WIDTH`, `MAP_HEIGHT`, `MAX_MESSAGES`, `STARTING_FOOD`, `STARTING_WATER`, `DAY_LENGTH_MOVES`, `NIGHT_LENGTH_MOVES`, `MOVE_FOOD_COST`, `MOVE_WATER_COST`, `SEARCH_TIME_COST`, `SHELTER_TILES`, `MAX_INVENTORY_SLOTS`, `COMPLEX_EVENT_CHANCE`, `COMPLEX_EVENT_TYPE_WEIGHTS`, `TRACCE_LOOT_CHANCE`, `PREDATOR_LOOT_WEIGHTS`, `RANDOM_REWARD_POOLS`, `TIPO_ARMA_LABELS`, `WEAR_FROM_USAGE`, `SAVE_KEY`.
    - **Punti di Attenzione per l'LLM:**
        - Questo file è il primo script JS caricato dopo `index.html`. Le variabili globali qui definite sono il principale mezzo di condivisione dello stato dinamico tra i moduli.
        - Le costanti qui definite non devono essere modificate dinamicamente. Qualsiasi nuovo parametro di gioco globale o probabilità generale dovrebbe essere aggiunto qui.

- **4.2. `js/game_data_misc.js` (Nuovo da v0.7.23)**
    - **Responsabilità Primaria:** Contiene dati statici descrittivi e testuali vari, che non rientrano specificamente nelle categorie di eventi principali, oggetti o ricette.
    - **Costanti/Strutture Dati Fondamentali:** `TILE_SYMBOLS`, `TILE_DESC`, `STATO` (enum per stati giocatore), `STATO_MESSAGGI` (array di testi per stati giocatore), `loreFragments` (array di testi per frammenti di lore), `mountainBlockMessages`, `tipiBestie`, `descrizioniIncontroBestie`, `descrizioniTracce`, `descrizioniIncontroPredoni`, `descrizioniOrroreIndicibile`, `esitiPericoloAmbientaleEvitato`,

`esitiPericoloAmbientaleColpito`, `esitiFugaPredoniKo`, `esitiParlaPredoniKo`, `esitiSeguiTracceOkNulla`, `esitiEvitaAnimaleKo`, `EVENT_CHANCE` (oggetto con probabilità base eventi per tipo di tile), `ITEM_EFFECT_DESCRIPTIONS` (oggetto che mappa tipi di effetto a funzioni generatrici di testo descrittivo per tooltip).
- **Punti di Attenzione per l'LLM:** Molti dei testi usati da `game_utils.getRandomText()` provengono da array definiti in questo file. È cruciale per l'atmosfera e il feedback narrativo.

- **4.3. `js/game_data_events.js` (Nuovo da v0.7.23)**

    - **Responsabilità Primaria:** Contiene le definizioni strutturate di tutti gli eventi di gioco specifici dei tile e degli scenari di dilemma morale.
    - **Costanti/Strutture Dati Fondamentali:**
        - `EVENT_DATA`: Oggetto principale. Le chiavi sono i nomi dei tipi di tile (es. `PLAINS`, `FOREST`, `CITY`, `REST_STOP`). Ogni valore è un array di oggetti evento. Ogni oggetto evento ha: `id` (stringa univoca per quel tipo di tile), `title` (stringa), `description` (stringa, può contenere `\n`), `isUnique` (booleano opzionale), `choices` (array di oggetti scelta). Ogni scelta ha: `text`, `actionKey` (stringa univoca per la logica di esito), e poi `skillCheck` (oggetto con `stat` e `difficulty`) con `successText`, `successReward`, `failureText`, `failurePenalty`, OPPURE `outcome` (stringa per risultato diretto), OPPURE `effect` (oggetto per effetto meccanico semplice). Possono esserci anche `isSearchAction: true`, `timeCost: numero`, `usesWeapon: true`.
        - `dilemmaEvents`: Array di oggetti, ciascuno rappresentante uno scenario di dilemma morale complesso, con una struttura simile agli eventi in `EVENT_DATA` (ma con `id` univoco globale per il dilemma).
    - **Punti di Attenzione per l'LLM:** La corretta struttura di `EVENT_DATA` e `dilemmaEvents` è fondamentale per il funzionamento di `triggerTileEvent`, `triggerComplexEvent` (per i dilemmi) e `handleEventChoice`. Ogni `actionKey`, `stat` per skill check, `itemId` in reward/penalty deve essere valido e coerente.

- **4.4. `js/game_data_items.js` (Nuovo da v0.7.23)**

    - **Responsabilità Primaria:** "Database" centrale di tutti i template degli oggetti esistenti nel gioco.
    - **Costanti/Strutture Dati Fondamentali:** `ITEM_DATA` (grande oggetto). La chiave di ogni entry è l'`itemId` (stringa univoca, es. `'canned_food'`, `'pipe_wrench'`).

    - **Struttura Oggetto Item (Riepilogo):** `id`, `name`, `nameShort` (opzionale), `description`, `type` (food, water, medicine, resource, tool, weapon, armor, ammo, blueprint, lore), `usable`, `stackable`, `weight`, `value`.
        - **Per `weapon`:** `weaponType`, `damage` (numero o `{min, max}`), `maxDurability`, `ammoType`, `ammoPerShot`, ecc.
        - **Per `armor`:** `armorValue`, `maxDurability`, `slot`.
        - **Per `ammo`:** `ammoType`.
        - **Per usabili:** `effects` (array di oggetti effetto, vedi Sezione 5.2 del manuale precedente per i tipi di effetto).
    - **Punti di Attenzione per l'LLM:** La coerenza di `itemId` è fondamentale. Ogni oggetto deve avere le proprietà appropriate per il suo `type`. `maxDurability` è per il template, `currentDurability` è sull'istanza dell'oggetto nel `player.inventory` o `player.equippedWeapon/Armor`.

- **4.5. `js/game_data_recipes.js` (Nuovo da v0.7.23)**

    - **Responsabilità Primaria:** Definisce tutte le ricette di crafting.
    - **Costanti/Strutture Dati Fondamentali:** `CRAFTING_RECIPES` (oggetto). La chiave di ogni entry è la `recipeKey` (stringa univoca, es. `'purify_water'`).
    - **Struttura Oggetto Ricetta:** `productName` (per UI), `productId` (deve esistere in `ITEM_DATA`), `productQuantity`, `ingredients` (array di `{itemId, quantity}`), `description` (opzionale), `successMessage` (opzionale).
    - **Punti di Attenzione per l'LLM:** Assicurarsi che tutti gli `itemId` usati in `productId` e `ingredients` esistano in `ITEM_DATA`.

- **4.6. `js/game_utils.js`**

    - **Responsabilità Primaria:** Funzioni di utilità generiche, pure o quasi pure.
    - **Funzioni Chiave:** `getRandomInt`, `getRandomText`, `addMessage`, `performSkillCheck`, `getSkillCheckLikelihood`, `isWalkable`, `getTipoArmaLabel`, `getItemEffectsText` (ora legge `ITEM_EFFECT_DESCRIPTIONS` da `game_data_misc.js`), `chooseWeighted`.
    - **Punti di Attenzione:** Verificare che `getItemEffectsText` e `getTipoArmaLabel` facciano riferimento correttamente alle costanti ora in `game_data_misc.js`.

- **4.7. `js/dom_references.js`**

    - **Responsabilità Primaria:** Centralizzazione dei riferimenti agli elementi del DOM.

- **Funzioni Chiave:** `assignAllDOMReferences()`.
- **Punti di Attenzione:** Mantenere aggiornato con `index.html`.

- **4.8. `js/ui.js`**

  - **Responsabilità Primaria:** Rendering dell'interfaccia utente. Legge stato e dati, aggiorna il DOM. Non modifica lo stato logico del gioco.
  - **Funzioni Chiave:** `renderStats`, `renderMessages`, `renderInventory`, `renderLegend`, `renderMap`, `showEventPopup`, `closeEventPopup`, `buildAndShowComplexEventOutcome`, `showItemTooltip`, `hideItemTooltip`, `getItemDetailsHTML`, `showMapTooltip`, `hideMapTooltip`, `disableControls`, `enableControls`, `showCraftingPopup`, `closeCraftingPopup`, `populateCraftingRecipeList`, `updateCraftingDetails`, `updateVersionDisplay`.
  - **Punti di Attenzione:**
    - **Indicatore "Notte":** La funzione `renderStats` è responsabile dell'applicazione della classe `.night-time-indicator` a `DOM.statDayTime`. Le istruzioni per correggere questo bug (inclusi i `console.log` di debug) devono essere applicate a questa funzione.
    - **Danno `[object Object]0`:** La funzione `getItemDetailsHTML` (usata per tooltip e popup azioni oggetto) gestisce la visualizzazione del danno. Se `describeWeaponDamage` (in `events.js`) chiama `getItemDetailsHTML` o una sua logica simile, il bug potrebbe essere qui o in `describeWeaponDamage`. È più probabile che `describeWeaponDamage` costruisca la sua stringa e debba essere corretta.
    - Le chiamate a `render*()` devono essere efficienti e avvenire solo quando necessario.

- **4.9. `js/player.js`**

  - **Responsabilità Primaria:** Logica del personaggio giocante (statistiche, inventario, azioni, crafting, durabilità).
  - **Funzioni Chiave:** `generateCharacter`, `addItemToInventory`, `removeItemFromInventory`, `useItem`, `equipItem`, `unequipItem`, `dropItem`, `applyWearToEquippedItem`, `showRepairItemTypePopup`, `applyRepair`, `checkAmmoAvailability`, `consumeAmmo`, `showItemActionPopup`, `closeItemActionPopup`, `checkIngredients`, `attemptCraftItem`, `checkAndLogStatusMessages`, `showTemporaryMessage`.
  - **Punti di Attenzione:**

        - Questo file è stato recentemente soggetto a correzioni di sintassi. Assicurarsi che sia stabile.
        - La corretta gestione di `player.inventory` con oggetti che hanno `currentDurability` vs oggetti stackabili è fondamentale.
        - La funzione `useItem` è molto complessa e gestisce molti tipi di effetti.

- **4.10. `js/events.js`**

    - **Responsabilità Primaria:** Trigger e risoluzione logica degli eventi.
    - **Funzioni Chiave:** `triggerTileEvent`, `triggerComplexEvent`, `handleEventChoice`, `describeWeaponDamage`, `applyChoiceReward`, `applyPenalty`, `performRestStopNightLootCheck`.
    - **Punti di Attenzione:**
        - `handleEventChoice` è il motore degli eventi.
        - **Danno `[object Object]0`:** Il bug risiede in `describeWeaponDamage`. Deve essere corretta per formattare correttamente il danno quando è un range (usando `getRandomInt`).
        - **Feedback Loot Eventi:** Le funzioni `applyChoiceReward` e `handleRandomRewardType` devono assicurare che venga chiamato `addMessage` con il nome specifico dell'oggetto trovato.
        - **Usura Arma in Combattimento:** Le chiamate a `applyWearToEquippedItem` devono essere presenti nei rami di successo dei combattimenti in `handleEventChoice`.

- **4.11. `js/map.js`**

    - **Responsabilità Primaria:** Generazione mappa, movimento, ciclo giorno/notte, flavor text.
    - **Funzioni Chiave:** `generateMap`, `movePlayer`, `consumeResourcesOnMove`, `applyPassiveStatusEffects`, `transitionToNight`, `transitionToDay`, `showRandomFlavorText`, `checkForLoreFragment`.
    - **Punti di Attenzione:** `movePlayer` orchestra molte azioni.

- **4.12. `js/game_core.js`**

    - **Responsabilità Primaria:** Orchestratore principale, inizializzazione, input, flusso schermate, salvataggio/caricamento.
    - **Funzioni Chiave:** `window.onload`, `showScreen`, `initializeStartScreen`, `initializeGame`, `handleGlobalKeyPress`, `handleEventKeyPress`, `handleChoiceContainerClick`, `setupInputListeners`, `endGame`, `saveGame`, `loadGame`.

- **Punti di Attenzione:** Gestione corretta degli stati `gameActive`, `gamePaused`, `eventScreenActive`.

(Parte 3: Strutture Dati Fondamentali (Dettaglio), Flussi Logici Chiave e Sincronizzazione Lavori)

5. STRUTTURE DATI FONDAMENTALI (DETTAGLIO v0.7.23)

- **5.1. Oggetto `player` (Variabile globale in `js/game_constants.js`, inizializzato in `js/player.js`)**

  - **Scopo:** Contiene tutte le informazioni dinamiche relative al personaggio giocante.
  - **Proprietà Chiave Rilevanti per Modifiche Recenti/Future:**
    - `equippedWeapon`: Oggetto `{ itemId: "id_arma", currentDurability: numero }` o `null`.
    - `equippedArmor`: Oggetto `{ itemId: "id_armatura", currentDurability: numero }` o `null`.
    - `inventory`: Array di oggetti.
      - Per oggetti con durabilità (armi/armature, da `js/game_data_items.js` con `maxDurability`): `{ itemId: "id_oggetto", quantity: 1, currentDurability: numero }`.
      - Per altri oggetti: `{ itemId: "id_oggetto", quantity: numero }`.
    - `knownRecipes`: Array di stringhe (`recipeKey` da `js/game_data_recipes.js`).
  - **Note per LLM:** La distinzione tra `maxDurability` (nel template dell'oggetto in `js/game_data_items.js`) e `currentDurability` (sull'istanza qui nel `player`) è fondamentale.

- **5.2. `ITEM_DATA` (Oggetto in `js/game_data_items.js`)**

  - **Scopo:** "Database" dei template per tutti gli oggetti. Chiave: `itemId`.
  - **Struttura Chiave Oggetto Item (Riepilogo):**
    - `id`, `name`, `nameShort`, `description`, `type` (food, water, ecc.), `usable`, `stackable`, `weight`, `value`.
    - Per `type: 'weapon'`: `weaponType`, `damage` (numero o `{min, max}`), `maxDurability`, `ammoType` (se applicabile), ecc.
    - Per `type: 'armor'`: `armorValue`, `maxDurability`, `slot`.
    - Per `type: 'blueprint'`: `effects: [{ type: 'learn_recipe', recipeKey: '...' }]`.

        - Per usabili con effetti diretti: `effects: [{ type: '...', ...}]` (es. `add_resource`, `cure_status`).
    - **Note per LLM:** Consultare questo file per le proprietà base di un oggetto quando si implementano logiche di loot, crafting o effetti.

- **5.3. `EVENT_DATA` (Oggetto in `js/game_data_events.js`)**

    - **Scopo:** Definizioni degli eventi specifici del tile. Chiavi: nomi dei tipi di tile (es. `PLAINS`). Valori: array di oggetti evento.
    - **Struttura Oggetto Evento:** `id`, `title`, `description`, `choices` (array). Ogni scelta: `text`, `actionKey`, e (`skillCheck` | `outcome` | `effect`), `successReward`, `failurePenalty`, opzionali `isSearchAction`, `timeCost`, `usesWeapon`.
    - **Note per LLM:** La coerenza di `actionKey` e la struttura delle ricompense/penalità sono essenziali per `handleEventChoice`.

- **5.4. `dilemmaEvents` (Array in `js/game_data_events.js`)**

    - **Scopo:** Array di oggetti evento che definiscono gli scenari di dilemma morale.
    - **Struttura:** Simile agli oggetti evento in `EVENT_DATA`.

- **5.5. `CRAFTING_RECIPES` (Oggetto in `js/game_data_recipes.js`)**

    - **Scopo:** Definizioni di tutte le ricette di crafting. Chiave: `recipeKey`.
    - **Struttura Oggetto Ricetta:** `productName`, `productId` (da `ITEM_DATA`), `productQuantity`, `ingredients` (array di `{itemId, quantity}`), `description`, `successMessage`.
    - **Note per LLM:** Gli `itemId` degli ingredienti e del prodotto devono esistere in `js/game_data_items.js`.

- **5.6. Array di Testo (principalmente in `js/game_data_misc.js`)**

    - `STATO_MESSAGGI`, `loreFragments`, `mountainBlockMessages`, ecc.
    - **Scopo:** Fornire varietà testuale. Usati con `game_utils.getRandomText()`.

6. FLUSSI LOGICI CHIAVE DEL GIOCO (Concettualmente Invariati, ma con riferimenti ai nuovi file di dati)

- **6.1. Inizializzazione (`js/game_core.js`):**

    - Caricamento script nell'ordine specificato.

    - `initializeGame()` chiama `generateCharacter()` (usa dati da `js/game_data_items.js` per equip e inventario iniziale) e `generateMap()` (usa `TILE_SYMBOLS` da `js/game_data_misc.js`).

- **6.2. Movimento Giocatore (`js/map.js` -> `movePlayer`):**

    - Utilizza `TILE_SYMBOLS` e `TILE_DESC` (da `js/game_data_misc.js`).
    - Chiama `triggerTileEvent` e `triggerComplexEvent` (da `js/events.js`).
    - `transitionToNight/Day` usano `STATO_MESSAGGI` (da `js/game_data_misc.js`).

- **6.3. Risoluzione Evento (`js/events.js` -> `handleEventChoice`):**

    - Usa `EVENT_DATA` e `dilemmaEvents` (da `js/game_data_events.js`).
    - Usa `ITEM_DATA` (da `js/game_data_items.js`) per ricompense/penalità.
    - `describeWeaponDamage` usa `ITEM_DATA`.
    - `applyChoiceReward` usa `RANDOM_REWARD_POOLS` (da `js/game_constants.js`) e `ITEM_DATA`.
    - `applyPenalty` può interagire con lo stato del `player`.

- **6.4. Interazione Oggetti (`js/player.js`):**

    - `useItem`, `equipItem`, ecc. operano su `ITEM_DATA` (da `js/game_data_items.js`) e modificano `player.inventory`/`equippedWeapon/Armor`.
    - `attemptCraftItem` usa `CRAFTING_RECIPES` (da `js/game_data_recipes.js`) e `ITEM_DATA`.

- **6.5. Salvataggio/Caricamento (`js/game_core.js`):**

    - Salva e carica le variabili di stato definite in `js/game_constants.js` (inclusi `player` e `map`).

7. SINCRONIZZAZIONE STATO LAVORI (RIFERIMENTO v0.7.23 "Data Unchained" - Aggiornamento 16/05/2025)

- **Principali Modifiche Introdotte con v0.7.23 "Data Unchained":**

    1. **Miglioramento Feedback Ricompense Eventi:** Le funzioni `applyChoiceReward` e `handleRandomRewardType` in `js/events.js` sono state modificate (come da istruzioni del 15/05) per includere chiamate a `addMessage` che specificano l'oggetto e la quantità trovati. *(L'efficacia di questa modifica è ancora da confermare pienamente nei test).*

2. **Suddivisione di `js/game_data.js`:** Il contenuto è stato modularizzato in `js/game_data_misc.js`, `js/game_data_events.js`, `js/game_data_items.js`, `js/game_data_recipes.js`.
3. **Aggiornamento `index.html`:** Inclusione dei nuovi file JS di dati nell'ordine corretto.
4. **Aggiornamento Informazioni di Versione:** A `v0.7.23 Data Unchained` (`GAME_VERSION_NUMBER`, `GAME_VERSION_NAME` in `js/game_constants.js`; titolo e meta in `index.html`; commenti nei file JS).

- **Funzionalità Implementate e Stato dei Test Recenti (Feedback del 15-16/05/2025):**
    - **Struttura UI di Base:** Funzionante.
    - **Generazione Mappa e Movimento:** Funzionanti.
    - **Ciclo Giorno/Notte:** Transizioni base OK.
    - **Sopravvivenza (HP, Fame, Sete), Inventario, Multi-Stato:** Meccaniche di base operative.
    - **Sistema Eventi (Generale):** Flusso delle scelte in `handleEventChoice` sembra ripristinato dopo correzioni di sintassi. Molti eventi specifici e complessi testati con successo a livello di attivazione/scelta.
    - **Durabilità Individuale Oggetti e Usura:** Confermata funzionante per azioni con `usesWeapon:true` (es. "Forza il passaggio"). Il salvataggio/caricamento della durabilità è OK.
    - **Crafting e Apprendimento Ricette:** OK.
    - **Popup Azioni Oggetto e Tooltip:** OK.
    - **Rifugio 'R' (Diurno e Notturno):** Logica di base OK.

- **Bug Noti / Problemi Attualmente Sotto Investigazione o da Risolvere (Post v0.7.23):**
    1. **Indicatore "Notte" non cambia colore (VERDE):**
        - **Sintomo Persistente:** La scritta "Notte" non diventa blu. I log di debug specifici inseriti in `renderStats` (tramite sostituzione completa della funzione da parte di Cursor) NON sono apparsi nella console nell'ultimo test.
        - **Sospetto Fortissimo:** La modifica a `js/ui.js` (sostituzione di `renderStats`) **NON è stata effettivamente salvata/applicata da Cursor o persiste un problema di cache del browser non risolto da hard refresh.**
        - **Stato:** NON RISOLTO. (PRIORITÀ MASSIMA DI DEBUG TECNICO).
    2. **Messaggio danno in combattimento: "Infliggi [object Object]0 danni..." (ROSSO):**

- **Sintomo:** Errore di formattazione nel log quando si infligge danno.
- **Diagnosi:** Problema in `describeWeaponDamage()` in `js/events.js` che non gestisce correttamente `weaponData.damage` quando è un oggetto `{min, max}`.
- **Stato:** NON RISOLTO. (PRIORITÀ ALTA DI CORREZIONE).

3. **Feedback Loot da Eventi (es. Rifugio Diurno) nel Log Eventi (GIALLO):**
    - **Sintomo:** Incertezza se gli oggetti specifici trovati vengano sempre elencati nel log eventi.
    - **Diagnosi:** Le modifiche per aggiungere `addMessage` specifici in `applyChoiceReward` e `handleRandomRewardType` (Task B del 15/05) potrebbero non essere state applicate da Cursor o potrebbero non coprire tutti i casi.
    - **Stato:** DA VERIFICARE APPROFONDITAMENTE.
4. **Mancanza Incontri con Predoni (GIALLO/ROSSO):**
    - **Sintomo:** Impossibilità di incontrare Predoni.
    - **Stato:** NON RISOLTO. (RICHIEDE TEST CON FORZATURA EVENTO).
5. **Usura Arma in *Combattimento Effettivo* (GIALLO):**
    - **Sintomo:** Non osservata usura durante combattimenti con Animali (i Predoni non appaiono).
    - **Diagnosi:** La chiamata a `applyWearToEquippedItem` in `handleEventChoice` per i successi in combattimento (Task C del 15/05) potrebbe non essere stata applicata da Cursor o non viene raggiunta.
    - **Stato:** DA VERIFICARE (dopo fix danno e accesso a combattimenti).
6. **Bilanciamento Fame/Sete e Disponibilità Risorse (GIALLO - Design).**
7. **Popolamento Testi Placeholder (GIALLO - Contenuto).**

- **Roadmap Prossimi Passi (Immediati e a Breve Termine):**
    1. **RISOLVERE DEFINITIVAMENTE BUG INDICATORE NOTTE (Massima Priorità):**
        - **Azione per Designer (Simone):** Procedere con la **sostituzione manuale** dell'intera funzione `renderStats()` in `js/ui.js` con il codice corretto fornito precedentemente (quello che include i `console.log("[DEBUG UI NOTTE RENDERSTATS]...")`). Salvare il file.

        - Testare con **hard refresh** e controllare la console per i log di debug quando è notte.
        - Se i log appaiono e la classe `.night-time-indicator` viene aggiunta (verificabile dai log `Classi DOPO aver applicato toggle`), ma il colore non cambia, allora il problema è una specificità CSS. In tal caso, istruire Cursor a rendere la regola `.night-time-indicator { color: cornflowerblue; font-weight: bold; }` più specifica, ad esempio: `section#game-info ul li span.night-time-indicator { color: cornflowerblue !important; font-weight: bold !important; }` (usando `!important` come ultima risorsa se necessario).
    2. **CORREGGERE BUG VISUALIZZAZIONE DANNO `[object Object]0` (Alta Priorità):**
        - Istruire Cursor a modificare `describeWeaponDamage()` in `js/events.js` come da istruzioni dettagliate del 15/05 (gestire `weaponData.damage` come oggetto `{min, max}` usando `getRandomInt`, assicurare che `playerDamage` sia un numero).
    3. **VERIFICARE E FINALIZZARE FEEDBACK LOOT EVENTI:**
        - Istruire Cursor a mostrare le funzioni `applyChoiceReward` e `handleRandomRewardType` in `js/events.js`. Verificare se le chiamate `addMessage(\`Hai trovato: ${ITEM_DATA[itemId].name} (x${quantity}).`, 'success');sono presenti dopo`addItemToInventory`. Se mancano, farle aggiungere.
    4. **TESTARE EVENTO PREDONI E USURA ARMA IN COMBATTIMENTO:**
        - Una volta risolti i punti 1-3, forzare l'evento Predatore e testare a fondo.
    5. **BILANCIAMENTO RISORSE.**
    6. **PULIZIA CODICE E CONTENUTI.**

7.6 Log Modifiche v0.7.24 "Blue Night" (da v0.7.23 "Data Unchained")

- **Scopo della Versione:** Stabilizzazione dell'interfaccia utente, correzione di bug critici nel feedback visivo e testuale, e verifica di meccaniche di combattimento fondamentali.

- **Principali Bug Corretti e Funzionalità Verificate:**

    1. **Indicatore "Notte" (UI):** RISOLTO.
        - Problema: L'indicatore testuale dell'ora non cambiava colore in blu durante la notte a causa di un problema di specificità CSS.
        - Soluzione: Aumentata la specificità della regola CSS per `.night-time-indicator` in `css/panels.css`.
    2. **Visualizzazione Danno Numerico (Events/UI):** RISOLTO.
        - Problema: Il messaggio di danno mostrava `[object Object]0` per armi con danno a range.

        - Soluzione: Modificata `describeWeaponDamage()` in `js/events.js` per calcolare correttamente un valore numerico da range di danno usando `getRandomInt()`.
    3. **Feedback Loot Eventi (Events/Player/UI):** RISOLTO.
        - Problema: Il log eventi non specificava gli oggetti ottenuti e la console mostrava errori di "Fallimento aggiunta".
        - Soluzione: `addItemToInventory()` in `js/player.js` ora restituisce `true` al successo. `applyChoiceReward()` in `js/events.js` ora usa questo valore e chiama `addMessage()` direttamente per il feedback del loot, non restituendo più una stringa.
    4. **Test Evento Predoni (Events):** MECCANICA BASE CONFERMATA.
        - Problema: Gli incontri con i Predoni erano rari o assenti.
        - Verifica: Forzando l'evento "PREDATOR" in `triggerComplexEvent()` (bypassando `COMPLEX_EVENT_CHANCE` e la selezione pesata), l'evento si è attivato correttamente. La frequenza normale è ora dipendente dal bilanciamento. La forzatura è stata rimossa.
    5. **Usura Arma in Combattimento (Events/Player):** CONFERMATA FUNZIONANTE.
        - Verifica: Assicurato che le opzioni di combattimento negli eventi complessi dinamici (Predatore, Animale) avessero `usesWeapon: true` in `triggerComplexEvent()`. La logica in `handleEventChoice()` che chiama `applyWearToEquippedItem()` funziona come previsto.
    6. **Errori Testo Esiti Evento (Content/Events):** RISOLTO.
        - Problema: `ReferenceError` per variabili di testo mancanti (es. `descrizioniTracceOkLoot`, `esitiOrroreIndicibileAffrontaOk`).
        - Soluzione: Aggiunte le definizioni delle costanti mancanti in `js/game_data_misc.js`.

- **Pulizia Codice:**

    - Rimossi i `console.log` di debug specifici usati per la diagnosi dei problemi sopra elencati, una volta confermata la soluzione.
    - Rimossa una riga di commento duplicata in `js/player.js`.

- **Stato Attuale (v0.7.24):** Il gioco è significativamente più stabile in termini di feedback all'utente e meccaniche di base testate. I prossimi passi si concentreranno sul bilanciamento delle risorse e sulla progressiva aggiunta di contenuti e meccaniche più complesse.