

Technische Dokumentation: Operation Phase Six

KLASSIFIZIERUNG: ABSOLUT LETZTE RETTUNG

NUR FÜR ENTWICKLER MIT ABGESCHLOSSENER LEBENSVERSICHERUNG

1. Einleitung: Ein Blick in den Abgrund (des Codes)

Willkommen zur technischen Aufschlüsselung von "Phase Six". Im Gegensatz zum Handbuch für unsere... *Einweg*-Agenten, richtet sich dieses Dokument an Sie, den unglücklichen Techniker, der diesen digitalen Schrecken warten, erweitern oder einfach nur verstehen muss, bevor er Ihren Verstand verschlingt.

Dieses Projekt wurde nach dem Prinzip des "pragmatischen Minimalismus" entwickelt. Das ist unsere höfliche Umschreibung für "Wir hatten weder Zeit noch Budget, also ist alles in einer einzigen, riesigen Datei". Betrachten Sie es nicht als schlechte Praxis, sondern als eine in Code gegossene Metapher für die erdrückende, unentrinnbare Natur des Labyrinths selbst.

2. Architektur-Philosophie: Das Monolith-Prinzip

Das gesamte System ist in einer einzigen HTML-Datei (Phase6NexusV1_2beta.html) enthalten. Es gibt keine externen Skripte (außer Tailwind CSS, weil selbst der Wahnsinn Stil haben sollte), keine Build-Prozesse, keine Frameworks, keine Dependencies. Warum? Weil jede zusätzliche Datei eine weitere potenzielle Bruchstelle in der Realität darstellt.

- **HTML:** Dient als Knochengerüst und Container für die verschiedenen UI-Zustände (Intro, Erkundung, Kampf, Overlays). Divs werden munter ein- und ausgeblendet. Es ist das digitale Äquivalent dazu, unliebsame Dinge einfach unter den Teppich zu kehren.
- **CSS:** Eine Mischung aus Tailwind-Klassen für schnelles Prototyping und benutzerdefiniertem CSS im `<style>`-Block für alles, was ein bisschen mehr... *Persönlichkeit* braucht. Animationen für Treffer, Todesstöße und geistigen Verfall finden Sie hier.
- **JavaScript:** Der eigentliche Dämon. Der gesamte Code residiert innerhalb eines einzigen `<script>`-Tags, umschlossen von einem DOMContentLoaded-Listener.
- **Asset-Management:** Externe Assets sind eine Schwäche. Daher werden alle visuellen "Sprites" (Spieler, Gegner, Items) als rohe SVG-Strings direkt in den JavaScript-Datenobjekten (playerBase, enemyPool) gespeichert. Das vermeidet Ladezeiten und HTTP-Requests, macht die Datenobjekte aber zu einer Augenweide für Masochisten.

Diese monolithische Struktur sorgt dafür, dass die Anwendung eine in sich geschlossene, tragbare Realitätsblase ist. Sie können sie überallhin mitnehmen. Sie können ihr nicht entkommen.

3. Das Nervensystem: Globale Zustände & DOM-Management

An der Spitze der Nahrungskette unseres Codes stehen zwei globale Konstanten, die alles zusammenhalten.

DOM

Ein monolithisches Objekt, das Referenzen auf quasi jedes relevante DOM-Element zwischenspeichert. Dies geschieht einmal beim Start, um die ständige, seelenzerstörende Suche im DOM-Baum zu vermeiden (`document.getElementById(...)`). Die Performance-Vorteile sind real, ebenso wie die Tatsache, dass dieses Objekt eine tickende Zeitbombe im globalen Namespace ist.

GAME_MODES

Ein simples Objekt, das die drei Hauptzustände des Spiels definiert: EXPLORATION, COMBAT, und MENU. Der `currentMode` wird vom `GameManager` gesteuert und diktiert, welche UI-Elemente sichtbar sind und welche Benutzereingaben verarbeitet werden. Es ist der "Bitte warten, während das Universum über Ihr Schicksal entscheidet"-Zustand.

4. Modulare Gliederung des Wahnsinns

Obwohl alles in einem globalen Scope lebt, ist der Code in vier logische Hauptobjekte unterteilt, die sich gegenseitig aufrufen, wann immer es nötig ist, das Leid des Spielers zu maximieren.

4.1 Der Architekt: MazeGenerator

- **API:**
 - `generate(width, height)`: Erwartet zwei Zahlen und gibt ein 2D-Array zurück.
- **Datenstruktur (Output):** Das Array besteht aus Objekten der Form `{ isWall: boolean, visited: boolean }`.
- **Algorithmus:** Ein klassischer *Recursive Backtracker*. Stellen Sie sich einen betrunkenen Programmierer vor, der durch ein Gitter stolpert und zufällig Wände einreißt, bis er sich selbst verirrt hat.

4.2 Der Reiseführer: Exploration

- **API / Schlüsselfunktionen:**
 - `init()`: Initialisiert den Canvas-Kontext.
 - `startLevel()`: Generiert ein neues Labyrinth, platziert Entitäten.
 - `gameLoop()`: Der `requestAnimationFrame`-Loop.
 - `draw()`: Zeichnet den aktuellen Zustand auf den Canvas.
 - `handlePlayerMove(e)`: Die primäre Eingabeschnittstelle.
- **Subsystem: Rendering-Details**
 - **Prozedurale Texturen:** Die Funktion `drawPixelArtTile` erzeugt eine texturierte Oberfläche ohne Bilddateien. Sie verwendet eine simple Formel, die auf den X/Y-Koordinaten der Kachel basiert, um eine von drei vordefinierten Farben auszuwählen. Das Ergebnis ist ein pseudo-zufälliges, aber deterministisches Muster, das den Wänden und Böden Tiefe verleiht. Effizient und absolut ausreichend, um Verwirrung zu stiften.
 - **Kamera-Glättung:** Die Kamera springt nicht abrupt, sondern folgt dem Spieler mit einer simplen Easing-Funktion: $camera.x += (targetX - camera.x) * 0.1$. Dies verhindert abrupte Bewegungen und reduziert die Wahrscheinlichkeit von Motion Sickness beim Spieler, bevor die Monster dies auf... traditionellere Weise tun.
 - **Render Culling:** Um nicht das gesamte Labyrinth bei jedem Frame zu zeichnen, berechnet die `draw()`-Funktion einen sichtbaren Bereich (`startX`, `endX`, `startY`, `endY`) basierend auf der Kameraposition und der Canvas-Größe. Nur Kacheln innerhalb dieses Rechtecks werden gerendert. Es ist der verzweifelte Versuch, Performance zu sparen, in einer Welt, in der jede Ressource endlich ist.
- **Subsystem: KI-Simulakrum**
 - Die "Intelligenz" der Monster in `moveMonsters()` folgt zwei simplen Regeln:
 1. **Sichtkontakt:** Ist der Spieler im Sichtfeld des Monsters (d.h. auf einer visible Kachel im `fogOfWar`), bewegt es sich stringent in seine Richtung (entweder horizontal oder vertikal, je nachdem, welche Distanz größer ist).
 2. **Ahnungslosigkeit:** Ist der Spieler nicht sichtbar, besteht eine 25%ige Chance, dass sich das Monster in eine zufällige, nicht blockierte Richtung bewegt.
 - Diese Logik erzeugt ein Verhalten, das von "ziellosem Umherirren" zu "unerbittlicher Verfolgung" wechselt, sobald der Agent entdeckt wurde.

4.3 Die Bürokratie der Gewalt: CombatSystem

- **API / Schlüsselfunktionen:**
 - `startCombat(playerData, enemyData)`: Initialisiert einen neuen Kampf.
 - `endCombat(...)`: Meldet das Ergebnis an den GameManager.
 - `rollDice(...)`: Die zentrale Würfel-Engine.
 - `enemyTurn()`: Enthält die tödliche, aber simple Gegner-KI.
- **Subsystem:** Die Mathematik des Leidens (Schadensberechnung)

Technische Dokumentation: Operation Phase Six

Der verursachte Schaden wird in `performAttack` und `resolvePlayerReaction` nach einer festen Formel berechnet:

Schaden = (Kritische_Treffer * Wunden) + max(0, Normale_Treffer - Schutz) * Wunden

- **Kritische Treffer:** Ignorieren den Schutz vollständig. Jeder kritische Erfolg (Würfelsumme ≥ 11) verursacht den vollen Wundschaden der Waffe.
- **Normale Treffer:** Werden vom Schutz des Ziels absorbiert. Nur die Treffer, die den Schutzwert übersteigen, verursachen Schaden.
- **Wunden:** Der Basis-Schadenswert einer Waffe.

Diese Formel macht Schutz zu einem wichtigen Wert gegen normale Angriffe, aber nutzlos gegen gut platzierte, kritische Treffer. Ein weiterer Grund zur Paranoia.

- **Subsystem: Verwaltung von Statuseffekten**

Statuseffekte wie blutend, geschockt oder korrodiert sind keine komplexen Objekte. Es sind simple Zähler (number) auf dem player-Objekt im `gameState`. Jede Runde werden sie in `startPlayerTurn()` dekrementiert und ihr Effekt wird durch simple `if (player.blutend > 0)`-Abfragen ausgelöst. Ein brutal einfacher, aber effektiver endlicher Automat.

4.4 Der Puppenmeister: GameManager

- **API / Schlüsselfunktionen:**

- `init()`: Der Startpunkt der gesamten Anwendung.
- `switchToMode(mode, data)`: Die wichtigste Funktion zur Steuerung des Anwendungszustands.
- `onCombatEnd(outcome)`: Verarbeitet das Ergebnis eines Kampfes.
- `unlockAchievement(id)`: Schaltet einen Erfolg frei.

- **Subsystem: Persistenz des Schmerzes (localStorage)**

Die Anwendung nutzt die `localStorage` API, um den Fortschritt zwischen den jämmerlichen Leben eines Agenten zu speichern.

- `nexusKillBonus`: Speichert die Gesamtzahl der Kills des letzten Laufs, die als HP-Bonus für den nächsten dienen.
- `nexusIncarnations`: Zählt, wie oft der Agent bereits versagt hat.
- `nexusAchievements`: Ein JSON-Array von IDs der freigeschalteten Erfolge.
- `nexusDefeatedEnemyTypes`: Ein Set von Gegner-IDs, um den "Kammerjäger"-Erfolg zu verfolgen.

- **Subsystem: Event-Listener-Voodoo**

In der Funktion `showMessage` wird ein gängiger Trick für Vanilla-JS-Anwendungen verwendet, um Event-Listener sicher zu entfernen und hinzuzufügen. Anstatt `removeEventListener` zu verwenden, wird der Button mit `cloneNode(true)` geklont und ersetzt. Der Klon hat keine der alten Listener. Der neue Listener wird dann an den frischen Klon gehängt. Das ist schmutzig, aber es funktioniert zuverlässig und verhindert Memory Leaks durch vergessene Listener.

Technische Dokumentation: Operation Phase Six

5. Klangsynthese des Schreckens: Das Audio-System

- **API:**
 - `setupAudio()`: Initialisiert den `AudioContext`.
 - `playSound(type)`: Spielt prozedural generierte Soundeffekte ab.
 - `playMusic(play)`: Steuert den prozeduralen Musik-Scheduler.
- **Web Audio API Nodes:**
 - Verwendet werden `OscillatorNode`, `GainNode`, `BiquadFilterNode`, und `BufferSourceNode` (für weißes Rauschen). Eine minimalistische, aber flexible Toolbox.
- **Musik-Scheduler (`scheduler`):**

Verwendet einen Lookahead-Scheduler (`setTimeout`), um Audio-Events präzise im `AudioContext` zu planen und so Timing-Probleme von `setInterval` zu umgehen.

6. Zukünftige Katastrophen (Bekannte Probleme)

- **Globaler Scope:** Alles lebt im globalen Scope. Jedes Modul kann die Interna eines anderen direkt manipulieren. Dies ist keine Funktion, es ist eine Drohung.
- **Kein Save-State:** `localStorage` speichert nur Metadaten. Ein Schließen des Tabs bedeutet den sofortigen, endgültigen Tod des aktuellen Laufs. Wir nennen das "Permadeath-Feature".
- **Balancing:** Das Balancing ist eine delikate Kunst, die auf Hoffnung, Gebeten und einer `Math.random()`-Funktion basiert.

7. Analyse der Sinnlosigkeit: Schlusswort

Diese Dokumentation bietet einen Einblick in die geordnete Anarchie, die "Phase Six" antreibt. Jedes Modul ist darauf ausgelegt, eine spezifische Form des Leidens zu verwalten. Das System ist robust genug, um zu funktionieren, aber fragil genug, um Sie daran zu erinnern, dass alles jederzeit zusammenbrechen kann.

Änderungen am Code sind möglich, aber nicht empfohlen. Wie das Labyrinth selbst, neigt auch diese Codebasis dazu, sich auf unvorhersehbare Weise zu verändern, wenn man an den falschen Stellen gräbt. Fügen Sie Ihr Feature hinzu, schließen Sie die Augen und hoffen Sie, dass Sie keine schlafenden Götter im globalen Scope geweckt haben.

Ende der technischen Einweisung.
Löschen Sie Ihren Browserverlauf.
Es wird Sie nicht retten, aber es ist ein Anfang.