ChatGPT

Technische Dokumentation: Modding in Hearts of Iron IV (HOI4)

In dieser Dokumentation werden sämtliche Modding-Aspekte von *Hearts of Iron IV* ausführlich erläutert, damit auch eine Künstliche Intelligenz alle Regeln, Strukturen und Mechaniken vollständig verstehen kann. Die Inhalte sind sachlich-technisch gehalten und umfassen alle relevanten Bereiche des HOI4-Moddings inklusive Dateipfade, Syntax und typischer Fallstricke.

Inhaltsverzeichnis

1. **Fokusbäume (national_focus)**
2. **Länderdefinitionen (countries, history/countries, states, tags)**
3. **Staaten und Provinzen (states, strategic_regions, provinces)**
4. **Einheiten-Templates und Technologien (units, technologies, equipment)**
5. **Events und Entscheidungen (events, decisions)**
6. **Ideen und Modifier (ideas, static_modifiers, dynamic_modifiers)**
7. **Traits und Charaktere (leaders, traits, advisors)**
8. **Künstliche Intelligenz (KI-Scripting, AI strategies)**
9. **Lokalisierung und Übersetzung (localisation-Dateien und -Regeln)**
10. **Grafik- und GUI-Modding (interface, GFX, Sprites, Icons)**
11. **Audio-Modding (Sounds, Musik)**
12. **Scripting-Regeln und Syntax (Befehle, Trigger, Effekte)**
13. **Kompatibilität, Fehlerdiagnose, Patch-Pflege**

---

1. Fokusbäume (national_focus)

Dateipfade und Struktur: Nationale Fokusbäume werden in HOI4 als Textdateien im Verzeichnis `/common/national_focus/` definiert [1]. Jede `.txt`-Datei repräsentiert einen Fokusbaum (z. B. `germany.txt` für den deutschen Baum). Der Dateiname ist dabei egal – das Spiel liest alle Dateien in diesem Ordner aus und integriert die darin definierten Fokusbäume [2]. Ein einzelner Fokusbaum wird durch den Block `focus_tree = { ... }` beschrieben, der eine eindeutige **ID** und weitere Attribute enthält. Beispielsweise beginnt der tschechoslowakische Fokusbaum so:

```
focus_tree = {
    id = czech_focus
    default = no
    continuous_focus_position = { x = 50 y = 1100 }

    country = {                    # Länderauswahl für diesen Baum
        factor = 0
        modifier = {
            add = 10
            tag = CZE
```

```
            has_dlc = "Death or Dishonor"
        }
    }
    ...
}
```

Beispiel: Oben definieren wir einen Fokusbaum mit der ID `czech_focus` (für die Tschechoslowakei). Der Eintrag `country = { ... }` legt fest, welche Länder diesen Fokusbaum verwenden: In diesem Fall standardmäßig keiner (`factor = 0`), außer das Land hat den Tag CZE und die DLC *Death or Dishonor* (dann wird additiv +10 gerechnet, was die Wahrscheinlichkeit erhöht, dass CZE diesen Baum erhält) [3] [4]. Falls `default = yes` gesetzt ist, würde dieser Baum für alle Länder ohne spezifischen Baum genutzt (häufig der generische Fokusbaum).

Foki-Einträge: Innerhalb des `focus_tree`-Blocks werden die einzelnen **Fokusse** als `focus = { ... }` Blöcke definiert. Jeder Fokus benötigt eine **ID** (einzigartig im gesamten Spiel), einen **Namen/Schlüssel** für die Lokalisierung, ein **Icon** und **Positionierungskoordinaten** im Baum (`x` / `y`) [5]. Optional können Voraussetzungen und Effekte definiert werden:

- `cost`: Kosten in politischen Machtpunkten für diesen Fokus. Beispiel: `cost = 10` [6].
- **Voraussetzungen:** `prerequisite = { focus = <ID> }` legt fest, welche Foki vorher abgeschlossen sein müssen (mehrere Einträge möglich für mehrere Voraussetzungen).
- **Effekte:** Was passiert beim Abschluss? Im Block `completion_reward = { ... }` werden Effekte ausgeführt (z. B. `add_ideas = CZE_fortification_focus` fügt eine Idee hinzu [7]).
- **Verfügbarkeits- und Abbruchbedingungen:** Mit `available = { ... }` kann man Bedingungen definieren, die erfüllt sein müssen, damit der Fokus überhaupt ausgewählt werden kann (z. B. bestimmtes Datum, Ideologie etc.). `cancel_if_invalid = yes` bewirkt, dass ein begonnener Fokus abbricht, falls er während des Fortschritts ungültig wird [8] (z. B. eine Voraussetzung fällt weg), und `continue_if_invalid = no` verhindert, dass er dennoch abschließt.
- `bypass`**:** Bedingungen, unter denen der Fokus **übersprungen** wird (bzw. automatisch als erledigt markiert), etwa wenn das Ziel bereits erreicht ist.
- **KI-Gewichtung:** `ai_will_do = { ... }` definiert, wie wahrscheinlich die KI diesen Fokus wählt. Intern wird ein Faktor berechnet (z. B. `factor = 200` im Beispiel) [9], oft kombiniert mit Triggern (siehe Abschnitt KI-Scripting). Eine höhere Zahl bedeutet höhere Priorität für die KI. (Details zur KI-Logik siehe Abschnitt 8.)

Gemeinsame und geteilte Bäume: Man kann Fokusbäume **teilen** oder **gemeinsam nutzen**, indem man im `country = { ... }`-Block mehrere Länder-Tags mit entsprechenden Faktoren/Modifikatoren angibt. Außerdem können mehrere Länder denselben Fokusbaum erhalten, wenn ihr Tag dort entsprechend berücksichtigt ist. Ein Fokusbaum kann auch als **Standard** (`default`) markiert werden, was typischerweise beim generischen Baum der Fall ist.

Typische Fehlerquellen: Häufige Probleme bei Fokusbäumen sind fehlende oder doppelte IDs, vergessene Kommas oder Klammern in der Syntax, sowie nicht vorhandene Lokalisierungstexte. Wenn ein Fokusbaum im Spiel nicht erscheint, sollte man überprüfen, ob die **ID eindeutig** ist und ob im entsprechenden Länderskript (country-Tag oder history) auf den richtigen Baum verwiesen wird. Außerdem müssen alle Fokus-Namen und -Beschreibungen in den Lokalisierungsdateien definiert sein, da sonst Platzhalter angezeigt werden. Die Icons für Foki müssen über die GFX-Dateien referenziert werden (siehe Grafik-Modding) – ein häufiger Fehler ist ein falscher Icon-Schlüssel, was zu einem fehlenden Bild im Baum führt.

2. Länderdefinitionen (countries, history/countries, states, tags)

Übersicht: Um ein neues Land ins Spiel zu bringen oder ein bestehendes zu modifizieren, müssen mehrere Dateien angepasst werden. Jedes Land in HOI4 wird über einen **3-letter Country Tag** identifiziert (z. B. `GER` für Deutschland). Alle existierenden Tags sind in der Datei `/common/country_tags/00_countries.txt` aufgelistet, wo jedem Tag eine Länderdefinitionsdatei zugeordnet ist [10] [11]. Neue Tags können auch in einer separaten Datei (z. B. `01_mycountries.txt`) im selben Ordner definiert werden, um die Originaldatei nicht zu überschreiben [12]. Eine Zeile dort hat das Format `TAG = "countries/<Name>.txt"` – damit verweist man auf die entsprechende Datei im Ordner `/common/countries/`.

Beispiel Tag-Eintrag: `ABC = "countries/MyCountry.txt"` würde einen neuen Tag `ABC` definieren, welcher seine Ländereigenschaften aus der Datei `MyCountry.txt` bezieht. Dieser Name ist frei wählbar, sollte aber sinnvoll den Landnamen beschreiben (im Originalspiel z. B. `GER - Germany.txt`).

Länderdefinitions-Datei (common/countries): Die Datei im Verzeichnis `common/countries/` enthält grundlegende Einstellungen des Landes. Klassischerweise stand hier früher u.a. die Farbcodierung des Landes auf der Karte. In aktuellen Versionen werden die Farben allerdings global in der Datei `common/countries/colors.txt` verwaltet – dort sind allen Tags RGB-Farbwerte zugewiesen [13]. Ebenso gibt es `cosmetic.txt` für alternative Farbschemata (Kosmetik-Tags, z. B. für formierbare Nationen) [14]. In der Länderdefinitionsdatei selbst findet man z.B.:

- **graphical_culture** (grafischer Kulturkreis für Einheitenmodelle),
- eventuell **party_name**-Definitionen pro Ideologie (teilweise aber auch über Lokalisierung geregelt),
- **long_name** und ähnliche Angaben, sofern benötigt,
- **match_picture**, **unit_locale**, etc., die z.B. beeinflussen, welche lokalen Bezeichnungen verwendet werden (je nach Version unterschiedlich genutzt).

Ein einfaches Beispiel-Inhalt für `MyCountry.txt` könnte sein:

```
graphical_culture = WesternEurope
color = { 38 119 175 }  # Falls nicht in colors.txt, RGB-Farbe
```

(Beachte: Viele dieser Angaben werden in neueren HOI4-Versionen zentralisiert verwaltet. Prüfe die aktuelle Vanilla-Dateistruktur, um zu sehen, welche Felder relevant sind.)

History-Datei (history/countries): Für jedes Land gibt es im Ordner `history/countries/` eine Datei, die die **Startbedingungen** im ausgewählten Szenario (1936 oder 1939) festlegt. Der Dateiname beginnt mit dem Tag, gefolgt vom Landnamen, z. B. `GER - Germany.txt`. In dieser Datei werden Startwerte wie Regierungsform, Regierungschef, Minister, Technologien, nationale Ideen, Startarmeen usw. festgelegt. Beispielauszüge aus einer solchen Datei für Frankreich [15]:

```
1936.1.1 = {
    set_politics = {
        ruling_party = democratic
        last_election = "1932.5.1"
```

```
        election_frequency = 48
        elections_allowed = yes
    }
    set_country_flag = FRA_legion_d_honneur
    set_research_slots = 3
    [...]
    oob = "FRA_1936_OOB.txt"
}
```

- Der Block unter dem Datum `1936.1.1` gibt den Zustand am Startdatum an (für 1939 gäbe es einen separaten 1939.9.1-Block). Im Beispiel wird die Politik gesetzt: Frankreich ist demokratisch, letzte Wahl am 1. Mai 1932, Wahlen alle 48 Monate erlaubt [15]. Außerdem setzt man Flags oder Variablen und die Zahl der **Forschungsplätze**, etc.
- `oob` **(Order of Battle):** Hier wird auf eine Datei im Ordner `history/units/` verwiesen, die die Starttruppen definiert. Im Beispiel `FRA_1936_OOB.txt` – diese enthält Divisionen, Armeen, Flotten etc., die das Land zu Spielbeginn besitzt. OOB-Dateien werden beim Laden des Landes ausgeführt [16], üblicherweise via solch einer Referenz.
- **Starttechnologien:** Oft sieht man Blöcke wie `set_technology = { ... }`, um anfangs erforschte Technologien zu definieren, oder `add_tech_bonus` für Forschungsmalus (z.B. UK beginnt mit Radar-Malus).
- **Nationale Geister und Ideen:** Über `add_ideas = my_idea` kann man Start-Nationalgeister vergeben. Ebenso werden hier historische Führer zugewiesen (siehe Charaktere) und eventuell Start-Verbündete oder Garantien über Effekte gesetzt (z.B. Guarantee- oder Relation-Effekte).

Wichtig: Ab Version **1.11 (Barbarossa)** wurde das **Charakter-System** eingeführt. Staatsoberhäupter, Feldherren, Berater etc. werden nun als *Charaktere* definiert (siehe Abschnitt 7), müssen aber in der History-Datei dem Land zugewiesen werden. Dies geschieht durch Zeilen wie `recruit_character = <Charakter-ID>` [17] innerhalb des Startdatumsblocks. Fehlende `recruit_character`-Einträge führen dazu, dass entsprechende Anführer oder Generäle im Spiel nicht auftauchen, selbst wenn sie in `/common/characters/` definiert wurden.

States-Zuweisung: Ein neues Land braucht auch Gebiete. In den State-History-Dateien (siehe Abschnitt 3) muss daher der **Besitzer (owner)** entsprechender Staaten auf den neuen Tag geändert werden und typischerweise `add_core_of = <TAG>` gesetzt werden, damit das Land Kerngebiete dort hat [18] [19]. Andernfalls hätte das Land keine Provinzen zu Spielstart oder keine Kernprovinzen (was zu Aufständen führen könnte). Für ein neues Land moddet man also häufig existierende Staaten um oder erstellt neue Staaten (siehe nächster Abschnitt).

Zusammenfassung Schritt-für-Schritt für ein neues Land:

- Tag in `common/country_tags/` hinzufügen (neue Datei oder bestehende erweitern).
- Länderdefinitionsdatei in `common/countries/` erstellen mit korrektem Namen (wie im Tag-Eintrag referenziert) und Einstellungen (Farbe, Kultur etc.). Ggf. auch `colors.txt` ergänzen, damit das Land nicht weiß dargestellt wird (fehlende Farbe führt oft zu grell-weißen Ländern) [13].
- History-Datei in `history/countries/` anlegen (Dateiname beginnt mit Tag) und alle wichtigen Startwerte eintragen: Politik, Führer, Ideen, Technologien, OOB usw. Am einfachsten kopiert man eine ähnliche bestehende Datei und passt sie an.

- Staaten im Ordner `history/states/` dem neuen Land zuordnen: In den betreffenden State-Dateien `owner = TAG` und `add_core_of = TAG` setzen (sowie `controller = TAG`, falls das Startdatum nach einem Kriegsszenario relevant ist).
- Flaggen: In `/gfx/flags/` die Flaggenbilder für den neuen Tag hinzufügen (z.B. `ABC.tga`, `ABC_fascism.tga` etc. für die Ideologie-Varianten).
- Lokalisierung: Namen des Landes und ggf. der Fraktionen/Parteinamen übersetzen (dazu siehe Abschnitt 9).

Typische Fehlerquellen: Ein häufiger Fehler ist, einen neuen Tag zu definieren, aber die Referenzen nicht überall zu setzen – z.B. wird die Ländernamen-Lokalisierung vergessen, was dann als Platzhalter wie „ABC:0“ im Spiel erscheint. Oder man vergisst, im State die `owner`-Angabe zu ändern, sodass das Land keine Provinzen besitzt und sofort besiegt ist. Auch die Kodierung der Dateien ist wichtig (siehe Lokalisierung für Encoding). Außerdem gilt es zu beachten, dass **Änderungen an bestehenden Ländern** manchmal das Setzen von `replace_path` im Mod-Descriptor erfordern, insbesondere bei `history/countries/`. Da HOI4 diese Dateien *nicht* merged, sondern komplett lädt, kann es zu doppelten Einträgen kommen. Im Zweifel kann man im `.mod`-File der Mod angeben, dass `history/countries` komplett ersetzt wird, um Konflikte zu vermeiden (dadurch entfallen aber natürlich alle Vanilla-Länder, die man nicht selbst wiedereinfügt). Diese Methode sollte vorsichtig eingesetzt werden.

3. Staaten und Provinzen (states, strategic_regions, provinces)

State-Definitionen: Das Spiel unterteilt die Weltkarte in **Provinzen** (kleinste Einheit auf der Karte) und gruppiert diese zu **Staaten**. Ein *State* repräsentiert also eine Region, die aus mehreren benachbarten Provinzen besteht, und ist die Einheit, auf der Ressourcen, Fabriken und politische Kontrolle gehandhabt werden. Staaten werden durch Textdateien im Verzeichnis `history/states/` beschrieben [20]. Jede Datei dort definiert genau einen Staat mit einer eindeutigen **State-ID**. Der Inhalt einer State-Datei umfasst:

- `id`: die eindeutige Nummer des Staates. (Diese muss konsistent sein – sie wird z.B. in Savegames und anderen Referenzen verwendet.)
- `name`: ein Schlüssel für den Staatsnamen, z.B. `"STATE_1"` für Korsika [21] (die eigentliche Benennung erfolgt via Lokalisierung, z.B. in `state_names_l_german.yml`).
- `manpower`: Bevölkerungszahl, die für Rekrutierung zur Verfügung steht [22].
- `state_category`: Kategorie/Stufe (z.B. `wasteland`, `rural`, `town`, `large_city` etc.), was die maximalen Bauplätze beeinflusst.
- `history = { ... }`: In diesem Block stehen alle **Startzustände** des Staates:
- `owner = TAG` – der Besitzer bei Spielstart [18]. Falls kein Besitzer angegeben ist, gilt der Staat als unbesiedeltes Niemandsland [20].
- `controller = TAG` – in manchen Szenarien relevant, falls das kontrollierende Land vom Besitzer abweicht (z.B. besetzte Gebiete).
- `add_core_of = TAG` – definiert, dass das genannte Land einen Kernanspruch auf den Staat hat [19]. Mehrere `add_core_of`-Zeilen sind möglich, falls mehrere Länder Kernansprüche haben (z.B. um historische Gebietsansprüche abzubilden). Analog gibt es `add_claim_of = TAG` für bloße Besetzungsansprüche ohne Kernbevölkerung [23].
- `victory_points = { <Prov-ID> <Wert> }` – legt Siegpunkte in einzelnen Provinzen fest (im Beispiel hat Provinz 3838 einen Siegpunkt-Wert 1) [18]. Mehrere Einträge möglich.
- `buildings = { ... }` – Startgebäude und Infrastruktur: z.B. `infrastructure = 2`, `industrial_complex = 1` für 1 Zivilfabrik, `air_base = 1` etc. Man kann auch pro Provinz innerhalb des Staates Gebäude setzen, indem man die Prov-ID als Schlüssel nutzt [24] (im Beispiel hat Provinz 3838 einen Hafen der Stufe 3).

- Außerhalb des History-Blocks:
- `provinces = { ... }` – eine Liste aller Provinz-IDs, die zu diesem Staat gehören [25]. Diese IDs korrespondieren zu den Einträgen in der Kartendatei `map/definition.csv`.
- `local_supplies` – Basisversorgung in dem Staat (z.B. für infraarme Gebiete, oft 0.0).

Ein stark verkürztes Beispiel (aus State 1 – Korsika) sieht so aus [18] [25]:

```
state={
    id=1
    name="STATE_1" # Corsica
    manpower = 322900
    state_category = town
    history={
        owner = FRA
        victory_points = { 3838 1 }
        buildings = {
            infrastructure = 2
            industrial_complex = 1
            air_base = 1
            3838 = { naval_base = 3 }
        }
        add_core_of = FRA
    }
    provinces={ 3838 9851 11804 }
    local_supplies=0.0
}
```

Hier ist State-ID 1 mit drei Provinzen (3838, 9851, 11804) definiert [25]. Besitzer und Kernland ist Frankreich (FRA), es gibt einen Siegpunkt in Provinz 3838 (Ajaccio) und einige Startgebäude. Diese Struktur wiederholt sich für alle Staaten.

Strategische Regionen: Unabhängig von Staaten teilt HOI4 die Welt in **Strategische Regionen** ein. Diese sind wichtig für das Air- und Navy-Management (Luftregionen, Seezonen) und für Wettereffekte. Die strategischen Regionen werden im Verzeichnis `/map/strategicregions/` definiert (jede Datei dort eine Region). Eine strategische Region enthält u.a.: - eine **ID** und einen **Namen** (Lokalisierung ähnlich wie Staaten), - eine Liste von Provinzen, die zu dieser Region gehören, - optional Modifier für z.B. Schiffsortung (in Seezonen) oder Wetterzone.

Wichtig ist, dass die zugewiesenen Provinzen **zusammenhängend** sein sollten (zusammenhängendes Gebiet) [26], da z.B. Inseln meist zu separaten (See-)Regionen gehören. Außerdem überschneiden sich strategische Regionen nicht – jede Provinz gehört genau einer strategischen Region. Änderungen an den strategischen Regionen sind komplex und können KI-Verhalten beeinflussen (z.B. Luftmissionen). Paradox bietet ein **Nudge-Tool** im Spiel (Konsole `tdebug` bzw. im Launcher „Nudge“), mit dem man Regionen visuell bearbeiten kann [27].

Provinzen und Karte: Die eigentlichen Provinzen werden durch die Kartengrafiken definiert: - In `map/definition.csv` stehen für jede Provinz-ID die RGB-Farbwerte, Terrain-Typ etc. Jede Pixel-Farbe in der `provinces.bmp` Karte wird dort einer ID zugeordnet. - Neue Provinzen hinzufügen erfordert: eine neue, unbenutzte Farbwerte in `provinces.bmp` (korreliert zu einer neuen ID in der CSV), Anpassen von `default.map` (Anzahl der Provinzen), eventuelle Aktualisierung von Klimakarten, Flusskarten etc.

Dies ist sehr aufwendig und fehleranfällig – oft werden Mods stattdessen nur neue **Staaten** aus existierenden Provinzen zusammengesetzt (Grenzen verschieben, ohne neue Provinzen zu malen).

Typische Fehlerquellen: Das Map-Modding ist einer der schwierigsten Teile. Häufige Fehler: - **ID-Konflikte:** Zwei Staaten verwenden dieselbe State-ID oder eine Provinz ist in zwei State-Dateien aufgeführt – das führt zu Abstürzen oder Bugs. - **Vergessene Provinzen:** Wenn man Provinzen in der Liste eines States vergisst oder falsch angibt, können sie im Spiel unzugeordnet sein (grau dargestellt). - **Owner/Core Fehler:** Wird ein `owner` nicht gesetzt, bleibt der Staat unbesetzt. Ohne `core` hat das Land hohe Revoltrisiken dort. - **Strategic Region Mismatch:** Provinzen, die keiner strategischen Region zugewiesen sind, erzeugen Fehler. Jede neue Provinz muss auch einer strategischen Region hinzugefügt werden. - **Error Log prüfen:** Das Spiel protokolliert viele dieser Probleme in der `error.log`. Beispielsweise Warnungen wie *"Province X is duplicated in states"* oder *"State Y has no owner"*. Daher sollte man bei Kartenänderungen unbedingt im Debug-Modus das Fehlerprotokoll überwachen (siehe Abschnitt 13).

4. Einheiten-Templates und Technologien (units, technologies, equipment)

Division-Templates und Einheitentypen: In HOI4 werden Land- und Luftstreitkräfte als Divisionen bzw. Staffeln dargestellt, die aus **Brigaden/Bataillonen** bestehen, welche wiederum bestimmte **Ausrüstung** und **Manpower** benötigen. Die Gestaltung der Divisionen (Division-Designer) erfolgt im Spiel durch den Spieler, aber die **Basis-Einheitentypen** und Ausrüstungsarten, aus denen Templates bestehen, sind in den Spieldateien definiert.

- **Einheiten (units):** In älteren PDX-Titeln gab es z.B. `common/units/`-Dateien für Einheiten. In HOI4 sind die Landkampfeinheiten durch Kombination von Ausrüstung definiert. Dennoch existieren Dateien, die die *Kategorien* und Eigenschaften festlegen, z.B. in `common/units/` findet man Unterordner wie `equipment` (für Ausrüstungstypen) und ggf. vordefinierte Templates für bestimmte Länder (in `history/units/`, via OOB geladen).
- **Divisionstemplates für KI:** Es gibt in `common/units/` auch Dateien für Spezialeinheiten-Limits etc., aber spezifische Templates werden meist in KI-Skripten oder per OOB vordefiniert. Mods können vordefinierte Templates hinzufügen, indem sie sie in die History (Start-OOB) oder via Events hinzufügen.

- **Technologien:** Das Forschungssystem in HOI4 schaltet neue **Ausrüstungsklassen**, **Upgrades** und **Modifier** frei. Technologien werden im Ordner `/common/technologies/` definiert [28]. Es gibt mehrere Dateien, meist aufgeteilt nach Tech-Bäumen (Infanterie, Panzer, Luft, Marine, Doktrinen etc.). Jede Technologie hat eine eindeutige **ID**, einen **Namen** (Lokalisierung), **Kategorie** (z.B. infantry, armor, industry), oft eine grafische Position im Tech-Grid (x/y Koordinate) und Voraussetzungstechnologien. Außerdem sind Effekte angegeben, z. B. welche **Einheiten oder Ausrüstung** freigeschaltet werden oder welche **Modifier** gewährt werden. Beispiel: Die Erforschung von "Basic Tanks" könnte ein neues Ausrüstungs-Objekt `light_tank` freischalten, was durch `unlock_equipments = { light_tank_equipment_1 }` dargestellt wird (vereinfacht). Technologien können auch Einheitenboni geben (z.B. +5% Infantry Angriff), üblicherweise durch Modifier in der Technologiedatei.

– *Doktrinen* sind ebenfalls als Technologien definiert (z.B. Landdoctrines in einer separaten Datei). Sie folgen dem gleichen Prinzip [29].

Für die Darstellung im Forschungsbildschirm gibt es GUI-Dateien (meist unter `/interface/`), die die grafischen Tech-Bäume und Kategorien abbilden. Modder müssen diese anpassen, wenn sie neue Tech-Kategorien oder umfangreiche Änderungen vornehmen, damit Buttons korrekt angezeigt werden. Oft genügt es aber, innerhalb bestehender Bäume neue Techs einzufügen.

- **Ausrüstung (equipment):** Ausrüstung definiert die konkreten **Waffen, Fahrzeuge, Flugzeuge, Schiffe** etc., die produziert und von Einheiten genutzt werden. HOI4 unterscheidet **Archetypen** und **Modelle**: Ein *Archetyp* ist ein Oberbegriff (z.B. "Jäger" oder "Gewehr") und die einzelnen Modelle (z.B. "BF-109" oder "Kar98k") erben Eigenschaften vom Archetyp. Laut Paradox-Wiki: *"Equipment is split into two types, archetype and regular. Archetype equipment is used to assign more general attributes that regular equipment then inherits via the archetype attribute."* [30].

Die Ausrüstungsdefinitionen finden sich in Dateien unter `/common/units/equipment/` [31]. Beispielsweise könnte `infantry_equipment.txt` den Archetyp `infantry_equipment` definieren und verschiedene Stufen (Infantry Equipment I, II, III ...) mit zunehmenden Werten. Ein Eintrag enthält Parameter wie `type = infantry_equipment`, `archetype = infantry_equipment`, **Bonuseigenschaften** (Hard Attack, Soft Attack, Reliability etc.), **Produktionskosten**, benötigte Technologien etc. Schiffe und Panzer haben komplexere Definitionen (mit Modulen), was aber ebenfalls über Ausrüstungsdefinitionen und Module in `common/units/` realisiert wird (z.B. `tank_modules`).

Wichtig: Jedes Ausrüstungsteil hat einen eindeutigen internen Namen. Wenn man neue Ausrüstung hinzufügt, muss man auch sicherstellen, dass entsprechende **Technologien** existieren, um sie freizuschalten, und **Icons/Grafiken**, damit sie sichtbar sind. Ebenso sollten Lokalisierungseinträge für Namen und Beschreibungen angelegt werden.

Einheiten-Statistiken und Balance: Viele Einheiteneigenschaften (Geschwindigkeit, HP, Supplybedarf, Erfahrungsfaktoren) sind in den Definitionen festgelegt. Einige globale Modifikatoren befinden sich auch in `common/defines` (LUA-Skripte mit globalen Konstanten). Beim Modding neuer Einheiten sollte man auf Balance achten – z.B. Extremwerte können die KI überfordern oder das Spielgefühl kaputtmachen.

Typische Fehlerquellen: - **Falsche Vererbung:** Wenn eine Ausrüstung falsch als Archetyp deklariert wird oder umgekehrt, kann das Spiel die Upgrade-Pfade nicht verstehen (z.B. die Mechanik der Ausrüstungsumrüstung erfordert korrekte Archetypenketten [32]). - **fehlende Lokalisierung:** Neue Einheiten oder Techs ohne Lokalisierung werden als Platzhalter angezeigt. - **Icon-Probleme:** Für neue Ausrüstung müssen Icons definiert werden (siehe Grafik-Modding). Ein häufiges Problem ist, dass Mods zwar neue Panzer o.Ä. definieren, aber kein Icon in der `.gfx`-Datei hinterlegen – dann sieht man im Produktionsfenster kein Bild. - **Lade-Reihenfolge:** Die Dateien in `/common/ideas/`, `/common/technologies/` etc. werden alphabetisch geladen. Manchmal muss man via Dateinamen („00_xyz.txt" vs. „zzz_mod.txt") sicherstellen, dass z.B. eigene Techs nach Vanilla-Techs geladen werden, falls man Referenzen überschreibt [33]. - **Kompatibilität mit KI:** Wenn man neue Einheitentypen einfügt, sollte man überprüfen, wie die KI darauf reagiert. Gegebenenfalls müssen KI-Skripte (ai_templates, ai_strategy) angepasst werden, damit die KI die neuen Einheiten baut oder die Techs erforscht.

5. Events und Entscheidungen (events, decisions)

Events: Events sind Skripte, die unter bestimmten Bedingungen ausgelöst werden können, um dem Spieler eine Nachricht und Entscheidungsoptionen zu präsentieren oder direkte Effekte im Spiel auszuführen. Event-Skripte liegen im Ordner `/events/` (beliebig aufteilbar in mehrere Dateien, z.B. nach thematik). Ein Event hat typischerweise folgendes Format:

```
country_event = {
    id = mymod.1
    title = mymod.1.t
    desc = mymod.1.d
    picture = GFX_event_picture      # (optional)
    trigger = { [...] }
    mean_time_to_happen = { days = X }
    immediate = { [...] }
    option = {
        name = mymod.1.a
        ai_chance = { factor = 50 }
        effect = { [...] }
    }
    option = {
        name = mymod.1.b
        ai_chance = { factor = 50 }
        effect = { [...] }
    }
}
```

- **ID:** Jedes Event braucht eine eindeutige ID. Konvention ist *<ModTag>.<Nummer>*, z.B. `GER.5` für ein deutsches Event oder eigenes Kürzel `mymod.1`. Die ID dient auch der Referenz (zum Triggern oder für den Eventverlauf).
- **Titel und Beschreibung:** `title` und `desc` verweisen auf Lokalisierungsschlüssel (im Beispiel `mymod.1.t` und `mymod.1.d`). Diese müssen in den Lokalisierungsdateien mit dem entsprechenden Text hinterlegt sein. `picture` kann ein Bild (1280×720px meistens) einblenden.
- **trigger:** Bedingungen, unter denen das Event *passieren kann*. Ist ein Trigger definiert, prüft das Spiel periodisch (standardmäßig alle 20 Tage) [34] mittels *Mean Time to Happen*, ob die Bedingungen erfüllt sind, und löst dann entsprechend die Events aus. **Wichtig:** Ohne `trigger` (und ohne `is_triggered_only`) würde ein Event sofort beim Laden feuern – daher haben die meisten Events entweder einen Trigger+MTTH oder sie werden manuell aufgerufen.
- **mean_time_to_happen (MTTH):** Gibt die mittlere Zeit an, bis das Event eintritt (wenn Trigger erfüllt). Beispiel: `days = 30` bedeutet ~30 Tage im Durchschnitt. Intern rechnet das Spiel alle 20 Tage [35] die Chance aus. Ist keine MTTH angegeben, *wird der Trigger nie geprüft* [34] – das Event würde also nie spontan passieren. (Ausnahme: Events, die *direkt* durch Effekte oder Konsolenbefehle ausgelöst werden.)
- **immediate:** Dieser Block wird sofort ausgeführt, wenn das Event ausgelöst wird, *noch bevor* es dem Spieler angezeigt wird. Hier kann man Vor-Effekte skripten (z.B. Variablen setzen).
- **option:** Die Antwortmöglichkeiten. Jedes Event braucht mindestens eine Option. Jede Option hat einen `name` (wieder ein Lokalisierungsschlüssel) und enthält Effekte, die beim Anklicken ausgeführt werden. Man kann außerdem `ai_chance` oder detailliertere `trigger` innerhalb der Option angeben, um zu steuern, welche Option die KI wählt (z.B. 90% Option A, 10% Option B).
- **Effekte in Optionen:** Im Effekt-Block kann alles passieren, was auch sonst als Befehl geht – z.B. Politische Macht geben: `add_political_power = 100` [36], Kriege erklären, Flags setzen etc.
- **is_triggered_only:** Setzt man am Event `is_triggered_only = yes`, dann wird es **nie** von selbst durch Trigger+MTTH passieren, sondern **muss explizit via Skript aufgerufen** werden [37]. Solche Events werden typischerweise von Fokus-Belohnungen, Entscheidungen oder

On_Action-Hooks ausgelöst (siehe unten). Das Flag verhindert dabei, dass ein fehlender Trigger das Event sofort feuert.

Event-Chains und Timing: Oft wollen Modder Ereignisketten erstellen. Dabei kommt das **Verketten von Events** und **Verzögern** ins Spiel. Beispiel: Ein Fokus gewährt ein Event direkt nach Abschluss [38]. Oder Option C eines Events soll nach zufälliger Verzögerung ein Folgeevent auslösen. Dies erreicht man so:

```
# In einer Event-Option:
hidden_effect = {
    country_event = {
        id = followup_event.1
        days = 14
        random = 336
    }
}
```

Hier wird im Verborgenen (für den Spieler unsichtbar) ein Event `followup_event.1` geplant, welches in 14 Tagen + 0–336 Stunden (0–14 Tage) ausgelöst wird [39]. Durch die Kombination von `days` und `random` erhält man einen zufälligen Zeitkorridor [40]. Diese Methode ist effizienter als über Trigger und MTTH zu arbeiten, weil das Event genau eingeplant wird und die Engine es nicht ständig prüfen muss (Performance). Wichtig: Bei manuell ausgelösten Events sollte man *immer* `is_triggered_only = yes` setzen [37], sonst würde ein solches Event ohne Trigger sofort nach Spielstart feuern.

Events können auch direkt ohne Verzögerung von anderen Skripten aufgerufen werden, z.B. per Befehl `country_event = { id = XYZ }` in Foki oder Entscheidungen.

Global oder Lokal: HOI4 kennt **country_event** (für länderspezifische Events, häufig) und **news_event** (für Zeitungsticker-Stil Nachrichten). Es gibt auch **state_event**, aber das wird seltener genutzt. In der Praxis verwendet man fast immer `country_event`. Um ein Event für ein anderes Land auszulösen, kann man im Effekt den Scope wechseln (z.B. `GER = { country_event = { id = abc.1 } }` würde das Event abc.1 für Deutschland triggern). Siehe dazu auch Scripting/Syntax weiter unten.

Decisions (Entscheidungen): Entscheidungen sind interaktive Aktionen im entsprechenden Spielmenü, die der Spieler (oder KI) unter bestimmten Voraussetzungen starten kann. Das Decision-System wurde mit Patch 1.5 eingeführt und erlaubt zeitlich begrenzte und wiederholbare Aktionen (ähnlich Missionssystem). Modding dafür findet in `/common/decisions/` statt. Es gibt meist Unterordner: `decisions/*.txt` für die eigentlichen Entscheidungen und `decisions/categories/*.txt` für die Kategorisierung.

Eine Decision-Definition sieht vereinfacht etwa so aus:

```
country_decisions = {
    category = {
        id = my_category
        icon = GFX_my_category_icon
        allowed = { always = yes }  # Kategorie sichtbar? (z.B. nur für
bestimmte Länder)
        # ...
```

```
        decision = {
            id = my_decision
            icon = GFX_my_decision_icon
            days_remove = 30        # verschwindet nach 30 Tagen, wenn nicht
aktiviert
            cost = 50              # Politische Macht Kosten
            available = { stability < 0.5 }    # Bedingungen, damit Decision
erscheint
            visible = { has_idea = my_idea }   # Bedingungen, damit überhaupt
angezeigt (optional)
            target = FRA           # falls zielt auf Frankreich (für
zielgerichtete Entscheidungen)
            remove_trigger = { is_at_war = yes }  # falls vor Ablauf entfernt
werden soll, wenn Bedingung erfüllt
            complete_effect = { add_stability = -0.2 }  # Effekt bei
Durchführung
            ai_will_do = {             # KI-Bereitschaft, diese Entscheidung
zu wählen
                factor = 5
                modifier = { factor = 0.1 condition }  # KI nimmt selten
falls condition
            }
        }
    }
}
```

- Entscheidungen sind nach **Kategorien** gruppiert (z.B. *Politik*, *Gebäude*, *Geheimdienst* etc.). Kategorien können in separaten Dateien definiert sein (`decisions/categories/*.txt`). Sie haben Icons und Sichtbarkeitsregeln.
- Jeder **decision**-Block enthält Bedingungen:
- `available` : Bedingungen, dass die Entscheidung genommen werden *kann* (wenn nicht erfüllt, Decision ausgegraut).
- `visible` : Bedingungen, dass die Decision *überhaupt erscheint*. Oft nutzt man visible, um z.B. eine Entscheidung erst nach einem bestimmten Datum anzuzeigen.
- `targeted_decision` : Manche Entscheidungen können ein Ziel haben (anderes Land, Staat etc.). Dann werden sie im Spiel pro Ziel aufgelistet. Dafür nutzt man Felder wie `target_root_trigger` (Filter, wer als Ziel in Frage kommt). Im Beispiel oben ist `target = FRA` fix auf Frankreich – das wäre statisch; dynamische Ziel-Entscheidungen sind komplexer.
- `days_remove` : wie lange die Option verfügbar bleibt, wenn man sie nicht anklickt (Danach verschwindet sie, kann aber evtl. wieder auftauchen, wenn die Bedingungen weiterhin gelten).
- `remove_trigger` : Falls gesetzt, verschwindet die Entscheidung vorzeitig, sobald dieser Trigger erfüllt ist (z.B. Kriegsbeitritt macht Mobilisierungsentscheidung obsolet).
- `complete_effect` : Was passiert, wenn die Entscheidung durchgeführt wird (ähnlich wie Event-Effekte).
- `cost` : meist politische Macht (oder andere Ressourcen, wenn modifiziert – manche Entscheidungen kosten z.B. Command Power oder Equipment).
- **KI-Verhalten:** Standardmäßig zieht die KI Entscheidungen **nicht** ohne weiteres [41] . Man muss in `ai_will_do` festlegen, mit welcher Wahrscheinlichkeit die KI das tut. Diese `ai_will_do` -Blöcke funktionieren analog zu Event-MTTH: Das Spiel prüft periodisch, ob KI-Entscheidungen

verfügbar sind, und berechnet anhand der Faktoren, ob sie gezogen werden [41]. Ohne `ai_will_do` würde die KI die Entscheidung ignorieren.

Entscheidungen können auch **dauerhafte Effekte** haben oder **wiederholbar** sein (durch Flags kann man steuern, ob eine Decision wieder erscheint). Zudem können Decisions andere Decisions aktivieren oder deaktivieren. Man kann mit `select_effect` Effekte definieren, die sofort beim Anklicken passieren, noch bevor die Ausführung bestätigt wird, aber das wird selten benötigt.

Typische Fehlerquellen: - **Falsche Kategoriezuteilung:** Wenn eine Decision keiner gültigen Kategorie zugewiesen ist, erscheint sie nicht. - **Bedingungslogik:** Verwechslung von `available` und `visible` – wenn `visible` nie true wird, sieht man die Option nie; wenn `available` nie true ist, sieht man sie grau, kann sie aber nie anklicken. Hier muss logisch sauber gearbeitet werden. - **Lokalisierung:** Jede Entscheidung (und Kategorie) braucht Namen und Beschreibungstexte in der Lokalisierung, sonst stehen Platzhalter da. - **KI ignoriert Decision:** falls `ai_will_do` vergessen wurde, nutzt die KI die Entscheidung *nie*, selbst wenn es sinnvoll wäre [41]. Umgekehrt kann eine schlecht gesetzte KI-Priorität zu unerwünschtem KI-Spam führen. - **Fehler in Effekten:** Ein häufiger Fehler ist, Effekte in Decisions anzugeben, die im falschen Scope sind. Z.B. `add_state_modifier` in einer country decision erfordert, dass man vorher auf einen State scope wechselt oder einen target definiert. Solche Probleme sieht man im Error-Log.

6. Ideen und Modifier (ideas, static_modifiers, dynamic_modifiers)

Ideen (Ideas): In HOI4 bezeichnet "Ideas" ein weites Feld – es umfasst Regierungsrichtlinien, Nationale Geister (National Spirits), Gesetzesänderungen, Berater und Theorien. All diese werden über das Idea-System verwaltet und in Dateien unter `/common/ideas/` definiert. Die Dateien sind oft nach Ländern oder Kategorien aufgeteilt (z.B. `germany.txt` für länderspezifische, `_laws.txt` für Gesetze etc.). Eine Idea-Definition sieht etwa so aus:

```
political_advisor = {  # Idee-Typ (Slot) z.B. Berater
    my_advisor_idea = {
        allowed = { original_tag = MYC }      # Nur verfügbar für bestimmtes
Land
        available = { has_government = democratic }
        cost = 150                            # Politische Macht Kosten
        picture = myc_advisor_portrait
        category = political_advisor
        traits = { ideology_driven }          # z.B. Attribut (optional,
meist für Country Leader relevant)
        allowed_civil_war = no
        modifier = {
            stability_factor = 0.05           # +5% Stabilität
            political_power_gain = 0.1        # +0.1 Macht/Tag
        }
    }
}
```

In diesem fiktiven Beispiel wird ein politischer Berater definiert. Wichtig sind die Felder: - **Idea-Typ:** Die oberste Ebene (`political_advisor`, `design_company`, `national_spirit` etc.) bestimmt, in

welchem Slot diese Idee wirkt (Berater, Minister, Designer, Geist). Es gibt vordefinierte Typen, die das Interface kennt. - **allowed:** Bedingungen, damit diese Idee grundsätzlich im Spiel vorhanden ist (wenn false, wird sie komplett ignoriert). Oft benutzt, um Ideen auf bestimmte Länder zu beschränken (wie im Beispiel `original_tag = MYC` – nur das Land mit Tag MYC hat diese Idee in seiner Liste). - **available:** Bedingungen, damit der Spieler sie tatsächlich auswählen kann. Z.B. ideologische Einschränkungen oder benötigte Technologien. - **cost:** die Politische Macht Kosten (oder Command Power bei Militärposten). - **picture:** Icon/Porträt, definiert in den GFX-Dateien. Ein falscher Verweis hier führt zu einem blanken Icon. - **category:** bestimmt, unter welcher Überschrift es im Menü erscheint (besonders relevant für National Spirits vs. Politische Berater etc.). - **allowed_civil_war:** ob man die Idee im Bürgerkrieg beibehalten darf (oft `no` für normale National Spirits, damit Rebellions-Staaten die nicht umsonst übernehmen). - **modifier:** die durch die Idee verliehenen Boni/Mali. Dies nutzt das allgemeine Modifier-System – z.B. wie oben Stability +5%. Hier können alle gültigen Modifier verwendet werden (siehe *List of modifiers* im Wiki für vollständige Liste). Diese Modifier gelten, solange die Idee aktiv ist.

Nationale Geister sind nichts anderes als Ideas vom Typ `national_spirit`. Sie werden meistens durch Events/Fokus vergeben (`add_ideas = idea_name`). Man definiert sie genauso unter einer entsprechenden Kategorie, oft in `ideas/*.txt`. Beispiel:

```
national_spirit = {
    MYC_recovery_spirit = {
        icon = GFX_goal_generic_consumer_goods      # man kann Fokus-Icons
verwenden
        allowed = { original_tag = MYC }
        removal_cost = 100       # kann für 100 PP entfernt werden (optional)
        modifier = {
            construction_speed_factor = -0.5   # -50% Baugeschwindigkeit
(Malus)
        }
    }
}
```

Hier wurde ein Nationaler Geist mit negativem Effekt definiert. `removal_cost` erlaubt dem Spieler, den Geist gegen Machtpunkte loszuwerden (falls sinnvoll).

Static vs. Dynamic Modifiers: *Modifier* sind die Zahlenwerte, die Ideen, Technologien, Events etc. dem Land geben (z.B. +10% Angriff). Das Spiel kennt hunderte verschiedener Modifier (für nahezu jede Variable). Diese sind fest im Code verankert, aber ihre Anwendung unterscheidet man in **statische** und **dynamische** Modifier: - **Statische Modifier** sind im Ordner `/common/static_modifiers/` definiert. Das sind vordefinierte Bündel an Effekten, die global oder unter bestimmten Umständen aktiv sind. Beispiele: Schwierigkeitseffekte (`VERY_HARD_PLAYER` etc.), Wettereffekte, ideologische Standardboni oder z.B. `victory_points` Modifier. Diese werden vom Spiel konstant angewendet, wenn die Bedingungen stimmen. Man moddet hier selten, außer man will z.B. die Boni für Schwierigkeitsgrade ändern. *(Anmerkung: In neueren Versionen hat Paradox einige static_modifiers in den Ordner* `common/modifiers/` *ausgelagert, teils doppelte Dateien – darauf achten, welche Datei die wirksame ist)* [42]. - **Dynamische Modifier** sind temporär vergebene Modifier-Blöcke, meist per Effekt. Beispielsweise kann ein Event sagen `enable_dynamic_modifier = { modifier = my_temp_mod days = 30 }`, woraufhin für 30 Tage ein definierter Modifier wirkt. Dynamische Modifier werden häufig für zeitlich begrenzte Effekte genutzt (z.B. “Kriegsmüdigkeit” ändert sich dynamisch oder ein Buff für 6 Monate nach einer Entscheidung). Sie werden in `/common/dynamic_modifiers/` definiert. Dort legt man im

Prinzip benannte Pakete von Modifiern fest, ähnlich wie static_modifiers, nur dass diese eben per Skript an- und ausgeschaltet werden können.

Unabhängig davon sind **Modifier** "dynamisch veränderbar innerhalb von Modifier-Blöcken" [43] – d.h. sie können in Events/Ideen an- und abgeschaltet oder geändert werden, im Gegensatz zu Defines (die nur beim Start fix sind). Wichtig ist: Ein Modifier-Eintrag allein macht noch nichts, er muss durch eine Idee, Tech oder anderes *getragen* werden [44]. Beispiel: Definiert man in static_modifiers einen Modifier `my_special_mod = { research_speed = 0.5 }`, passiert erst etwas, wenn z.B. ein Event `add_static_modifier = my_special_mod` für ein Land aufruft.

Typische Fehlerquellen: - **Vertauschung von Triggern und Modifiern:** In Ideen-Dateien dürfen nur Modifier im `modifier`-Block stehen. Modder versuchen manchmal, dort Trigger einzubauen (fälschlicherweise, in der Hoffnung, es würde dynamisch wirken). Das funktioniert nicht – Trigger gehören in `available` oder `allowed`, nicht in `modifier`. Umgekehrt können Modifier nicht in Trigger-Blöcken benutzt werden (sie sind kein Zustand, sondern Effekte) [45]. - **Kompatibilität mit KI:** Ideen wie Gesetze haben oft KI-Gewicht in separaten Dateien (z.B. ai_will_do Blöcke, oder hardcodiert). Wenn man neue Gesetz-Ideen hinzufügt, muss man ggf. KI-Prioritäten definieren, sonst wird die KI sie selten wechseln. Manche Ideen (z.B. Design Companies) tauscht die KI nie, es sei denn, man nutzt events, weil sie statisch vergeben werden. - **Fehler bei Removal/Upgrade:** Bei National Spirits mit `removal_cost` oder bei ideas, die andere disabled machen (via `allowed`/`cancel` Mechanismen), können logische Widersprüche auftreten. Testen ist wichtig – kann der Spieler in eine Sackgasse kommen, wenn eine Idee entfernt wird? - **Modifier Wirkung:** Nicht alle Modifier wirken in jedem Kontext. Z.B. wirken bestimmte Ausrüstungs-Modifier nur auf entsprechende Einheitentypen. Es gibt lange Listen, was was tut (das Wiki listet alle bekannten Modifier und ihre Wirkung [46]). Ein häufiger Stolperstein: z.B. `research_speed` vs `industrial_research_speed` – wenn man falschen verwendet, sieht man keinen Effekt. Hier hilft das Fehlerprotokoll nur begrenzt; es erfordert Nachschlagen.

7. Traits und Charaktere (Leaders, Traits, Advisors)

Überblick – Neues Charaktersystem: HOI4 hat mit neueren Updates (v.a. *La Résistance* und *No Step Back*) ein umfassendes Charakter-System eingeführt. Früher waren Staatsoberhäupter und Generäle einfach Einträge in History-Dateien; jetzt werden **Charaktere** zentral definiert und dann einzelnen Ländern zugewiesen. Charaktere umfassen Staatsführer, Generäle, Admiräle, Berater, Feldmarschälle, Spione etc. Jeder Charakter hat Eigenschaften (**Traits**), die bestimmte Boni verleihen.

Definition von Charakteren: Im Ordner `/common/characters/` befinden sich die Charakterdefinitionen. Dort kann man pro Land oder global Charaktere definieren. Eine Charakter-Definition hat in etwa diesen Aufbau:

```
characters = {
    MYC_john_doe = {
        name = "John Doe"
        description = "Genius Strategist"
        picture = myc_john_doe_portrait
        gender = male

        country_leader = {                          # Wenn er
Staatsoberhaupt ist
```

```
            ideology = democratic
            traits = { charismatic }                # z.B. hat den Trait
"Charismatisch"
        }
        advisor = {                                 # Wenn er ein
politischer Berater ist
            slot = political_advisor
            idea_token = MYC_doe_advisor            # Verweist auf eine Idea
in common/ideas
            availability = { is_exiled = no }
        }
        corps_commander = {                         # Wenn er ein General
ist
            skill = 3
            attack_skill = 4
            defense_skill = 2
            planning_skill = 3
            logistics_skill = 1
            trait = { desert_fox logistics_wizard }
        }
        # admiral = { ... } analog für Admiräle
    }
}
```

Das obige Beispiel skizziert einen fiktiven Charakter, der mehrere Rollen haben könnte – natürlich hat in der Realität eine Person meist entweder die Rolle *country_leader* oder *advisor* oder *commander*. Aber es ist möglich, dass z.B. ein General gleichzeitig Berater-Idee im Spiel ist (z.B. bestimmte Generäle als Kriegsminister). Schlüsselpunkte: - `name` und `description` sind Lokalisierungskeys (für Namen und ggf. Tooltip-Beschreibung). - `picture` ist der Porträt-GFX (definiert in einer interface/GFX Datei). - Je nach Rolle gibt es Unterblöcke: - **country_leader:** für Staatschef. Erfordert Angabe der Ideologie, damit das Spiel weiß, unter welcher Regierung der Charakter Staatsoberhaupt wird. *Nur einer* pro Ideologie pro Land aktiv. Traits hier sind besondere Country-Leader-Traits (z.B. *stahlhart* etc.), definiert in `common/country_leader/*.txt` oder `common/traits`. - **advisor:** für politische Berater/Designer. Der `idea_token` muss auf eine Idea verweisen [47], welche den eigentlichen Effekt definiert (siehe Abschnitt Ideen). Im Grunde "trägt" der Charakter nur das Porträt und den Namen, während der Idea-Eintrag die Kosten und Modifier hat. Dies ermöglicht es auch, Berater per Event auszutauschen (indem Idea entfernt/hinzugefügt wird). `slot` gibt an, welche Kategorie (political_advisor, military_high_command, theorist etc.). Man kann auch `available` / `availability` setzen, um die Verfügbarkeit dynamisch zu steuern (z.B. nur verfügbar, wenn eine bestimmte Regierung an der Macht ist). - **corps_commander / field_marshal:** für Generäle und Marschälle (Land) – analog `navy_commander` für Admiräle. Diese haben Attribute (Skill-Level, einzelne Fertigkeitswerte) und Listen von Traits (z.B. *Desert Fox*, *Logistics Wizard*). Die Traits sind in `common/units/units_leader/*.txt` definiert (oder `common/traits` – Paradox hat die Organisation geändert, aber es gibt Dateien wie `00_traits.txt` mit allen Generals-Traits). - Ein Charakter kann theoretisch mehrere dieser Blöcke haben (z.B. gleichzeitig General und Berater), aber dann muss man aufpassen, dass die `idea_token` unique ist und die Zuweisung logisch bleibt.

Zuweisung im Spiel: Damit ein definierter Charakter im Spiel auch aktiv ist, muss er einem Land *zugewiesen* werden. Das geschieht (wie schon erwähnt) in der jeweiligen `history/countries/` Datei mit `recruit_character = <ID>` [17] [48]. Ohne diesen Eintrag existiert der Charakter zwar in den

Dateien, aber taucht nicht im Spiel auf. Alternativ können Charaktere auch per Event ins Spiel gebracht werden (Effekt `recruit_character = <ID>`). Beispiel: In `MYC - Mycountry.txt` (History) könnte man am Start: `recruit_character = MYC_john_doe` setzen, damit John Doe ab Spielstart im Land ist (sei es als General oder was definiert).

Trait-Definitionen: Traits (Eigenschaften) sind in den Dateien unter `common/traits/` bzw. `common/unit_leader/` und `common/country_leader/` definiert. Hier legt man fest: - Trait-Name (für interne Verwendung, z.B. `desert_fox`), - Anzeige-Name/Description (Lokalisierung, z.B. "Wüstenfuchs"), - Kategorie (z.B. `leader_trait`, `tank_designer_trait`, `army_high_command_trait` etc., um Einordnung im Spiel und Kompatibilität zu bestimmen), - Welche Boni der Trait gibt (z.B. `attack_skill_factor = 0.25` für +25% Angriffskillsteigerung für Generäle), - Gegebenenfalls Voraussetzungen (manche Traits sind *upgradeable*, oder können nur bestimmte Einheitenkommandanten zugewiesen werden).

Neue Traits hinzuzufügen ist möglich. Man muss sicherstellen, ein Icon dafür bereitzustellen: Charakter-Traits nutzen kleine 64×64 Icons in `/gfx/interface/traits/`. Modder können eigene Trait-Icons dort ablegen (Namenskonvention: `trait_mytrait.dds`) und müssen in einer GFX-Datei einen `spriteType` definieren mit `name = "GFX_trait_mytrait"` referenziert auf die Bilddatei [49]. Danach kann man den neuen Trait in den Definitionen verwenden und Charaktere damit ausstatten. Beachte: Harte Kodierung – manche Trait-Effekte sind fix im Code an bestimmte Namen gebunden, aber die meisten reinen Modifier-Traits funktionieren generisch.

Adviser Traits vs Leader Traits: Es gibt spezielle Traits, die direkt als Ideen fungieren (z.B. Chief of Army - "Offensive Doctrine" etc. sind im Prinzip Ideas mit modifiers, aber manchmal hat Paradox diese auch als Traits behandelt). Grundsätzlich muss man unterscheiden zwischen **Country Leader Traits** (für Staatsoberhäupter, wirken global wie z.B. Hitler's Demagoge-Trait) und **Unit Leader Traits** (für Generäle/Admiräle, betreffen die Armee/Marine unter ihrem Kommando). Berater haben in der Regel keine eigenen "Traits", ihr Effekt kommt vollständig aus der Idea-Definition (z.B. +10% Angriff für Infanterie wenn ein High Command mit Infanterist-Hintergrund eingestellt wird – das steht als Modifier in der Idea). Allerdings werden Berater-Kategorien selbst manchmal als Traits bezeichnet (z.B. *Infantry Expert* als Armeeberater – intern ist das ein Trait mit Bonus, zugewiesen über Idea).

Typische Fehlerquellen: - **Charakter nicht rekrutiert:** Man vergisst den `recruit_character` Eintrag – der Charakter bleibt unsichtbar. Das Error-Log kann hier z.B. Warnungen geben, wenn ein Advisor Idea definiert ist, aber der entsprechende Charakter nicht rekrutiert wurde (oder umgekehrt). - **Doppelvergabe:** Zwei Charaktere mit gleicher Ideologie als country_leader – das Spiel nimmt dann meist den zuerst rekrutierten. Wenn z.B. via Event ein zweiter democratic leader rekrutiert wird, ersetzt er den ersten. Man sollte alte ggf. entfernen (`remove_country_leader_role` Effekt). - **Trait funktioniert nicht:** Wenn ein neu erstellter Trait nicht greift, prüfen ob der Trait richtig kategorisiert wurde und ob der Charakter die richtige Rolle hat. Ein Admiraltrait auf einem General bringt nichts. Oder man hat den Trait zwar definiert, aber keinem Charakter gegeben – dann sieht man ihn nie. - **Porträt fehlt:** Falscher `picture`-Pfad in Charakterdefinition oder fehlender GFX-Eintrag -> resultiert im schwarzen Silhouettenbild. - **Idea-Token Inkonsistenz:** Wenn man Advisor-Charaktere macht, muss `advisor.slot` und das zugehörige `idea_token` in `common/ideas` übereinstimmen. Vergisst man die Idea oder schreibt anderen Token, hat man Geistercharaktere oder nicht klickbare Berater. - **Alte vs neue Methode mischen:** Manche älteren Mods behalten das alte System (z.B. setzen Führer nur über History ohne character-Eintrag). Das geht teilweise noch, aber viele neuere Mechaniken (z.B. Spionage oder Biographien) funktionieren nur mit dem Character-System. Es ist ratsam, konsistent das neue System zu verwenden.

8. Künstliche Intelligenz (KI-Scripting, AI strategies)

HOI4 bietet umfangreiche Möglichkeiten, das Verhalten der KI zu beeinflussen. Grob gibt es zwei Ansätze: 1. **Inline-Gewichtung in Game-Elementen (ai_will_do)** – viele definierbare Sachen wie Foki, Entscheidungen, Technologien, Ideen haben einen Block `ai_will_do = { ... }`, in dem per Faktor und Trigger die KI-Präferenz festgelegt wird. 2. **AI-Strategiepläne (AI strategy plans)** – separate Skripte, meist länderspezifisch, die der KI strategische Leitlinien vorgeben (welche Fokusse in welcher Reihenfolge, welche Nation anzugreifen, welche Tech-Priorität usw.).

ai_will_do und MTTH: In den meisten Fällen interpretiert das Spiel den `ai_will_do`-Block ähnlich wie einen MTTH (Mean time to happen) Mechanismus. Es wird zufällig eine Zahl von 0 bis zum angegebenen Wert gezogen und damit entschieden [50]. Höhere Faktoren machen eine Aktion also wahrscheinlicher. Man kann mit Unterbedingungen arbeiten:

Beispiel aus einem Fokus:

```
ai_will_do = {
    factor = 5
    modifier = {
        factor = 0    # KI ignoriert
        date < 1939.1.1
    }
}
```

Hier hätte die KI grundsätzlich Faktor 5, aber vor 1939 ignoriert sie den Fokus (Faktor 0). So lassen sich historisch plausiblere Abläufe gestalten. Entscheidend ist: **Ohne `ai_will_do` führt die KI eine Aktion meist nie aktiv aus.** Z.B. ein Fokus ohne ai_will_do wird von der historischen KI evtl. nur über die Default-Historic-Liste angegangen, oder gar nicht, falls kein Plan existiert. Bei Entscheidungen gilt sogar „ohne ai_will_do = nie“ [41].

AI-Strategiepläne: Im Ordner `/common/ai_strategy_plans/` (und ähnlichen, z.B. ai_peace, ai_focuses) sind vordefinierte Pläne. Besonders im historischen Modus folgt die KI diesen Plänen strikt. Ein AI-Plan kann z.B. sagen: *Wenn Weltfrieden <X, mache Fokus A, dann B, dann C…* oder *Nach Kriegsbeginn, konzentriere auf Panzer-Tech*. Diese Pläne modifizieren die Gewichtungen. Z.B. kann ein Plan focus-Faktoren setzen: *“Für Fokus XYZ factor = 0”* damit KI den nicht auswählt, oder *“focus_factors { ABC_focus = 2.0 }”*, um die Priorität zu verdoppeln [51]. Wenn ein AI-Plan aktiv ist, multipliziert er die internen `ai_will_do`-Werte mit diesen Faktoren [51].

Paradox hat für die historischen Abläufe viele dieser Pläne erstellt, insbesondere um alternative Pfade ebenfalls zu ermöglichen. Für Mods, die ahistorische Szenarien bieten, kann man eigene AI-Pläne schreiben. Diese werden in der Länder-History oder per Events aktiviert (Effekt `switch_ai_strategy_plan`). Außerdem gibt es *strategic region AI* Skripte, *invasions KI* Parameter etc., die alle modifizierbar sind.

AI Scripting allgemein: Man kann die KI mit Befehlen beeinflussen, z.B. via `add_ai_strategy = { type = invasion id = "TAG" value = 50 }` könnte man einer KI-Land sagen, es soll Invasion gegen Land X planen (die Zahl ist Priorität). Ebenso kann man via `ai_strategy` Einträge in der History vorgeben, wen ein Land ignorieren soll, welche Gebiete es anstrebt, etc. Diese tiefergehenden AI-Skripte liegen meist in `common/ai_strategies` und `common/ai_strategy_plans`. Der Umfang

ist groß; im Rahmen dieser Dokumentation sei gesagt: Mods können damit z.B. sicherstellen, dass die KI eines neuen Landes den gewünschten Pfad nimmt oder bestimmte Techs priorisiert.

KI-Prioritäten für Forschung: Standardmäßig hat jede Tech einen `ai_will_do` (teils generisch berechnet über Tech-Block und Nutzen) [52] . Mods mit neuen Technologien sollten diese Blöcke sinnvoll ergänzen, sonst forscht die KI evtl. nicht darauf hin. Alternativ kann man AI-Pläne nutzen (z.B. *in 1940, if not researched techX, add huge factor*).

Grenzen der KI: Wichtig zu verstehen: HOI4-KI hat Hardcodiertes Verhalten (z.B. Divisionen bauen nach Gewichtung Templates, Heeresgruppenführung etc.), das man nur indirekt steuern kann (Templates via define weight in AI, Prioritäten via *ai_equipment* files, etc.). Man kann KI nicht völlig neu "beibringen", aber man kann Parameter setzen. Zum Beispiel gibt es in `common/ai_equipment` Vorgaben, wie viele Divisionen welcher Art ein Land anstrebt. Für neue Einheitentypen müsste man diese verteilen.

Typische Fehlerquellen: - **Unvorhergesehene KI-Dummheiten:** Manchmal führt ein modifizierter KI-Plan zu blockiertem Verhalten. Z.B. wenn zwei Fokus gegenseitig factor 0 bekommen, hat KI evtl. nichts sinnvolles zu wählen. Immer KI im Testlauf beobachten! - **ai_will_do Syntaxfehler:** Vergessene geschweifte Klammern in `ai_will_do`-Blöcken crashen das Script-Parsing. - **Performance:** Zu viele oder zu komplexe Trigger in KI-Blöcken (etwa in jedem Fokus ein riesiger Block mit täglichen Checks) könnten das Spiel verlangsamen. AI-Pläne sind hier performanter, da sie seltener ausgewertet werden. - **Kompatibilität Off/On:** Falls man ahistorisches Gameplay unterstützt, bedenken, dass *historical AI focuses* Einstellung einiges überschreibt. Meist gelten dann starre Pfade aus den AI-Plänen. Mods sollten gegebenenfalls eigene Pfade dort definieren, sonst nimmt KI immer Vanilla-Fokuspfad auf historisch. - **Achtung bei Balance:** Wenn man KI-Faktoren zu hoch setzt, kann es sein, dass KI immer nur eine Sache tut (z.B. immer die gleiche Entscheidung spammt). Hier moderat justieren und testen.

9. Lokalisierung und Übersetzung (Localisation-Dateien und -Regeln)

Lokalisation in HOI4: Texte (Namen, Beschreibungen, Tooltipps) sind in Lokalisierungsdateien ausgelagert, damit Modding und Übersetzung einfacher ist. Diese Dateien liegen im Ordner `/localisation/` (bei Mods entsprechend im Mod-Verzeichnis). HOI4 verwendet **YAML**-Dateien mit der Endung `.yml` für die Texte. Es gibt separate Dateien pro Sprache, erkennbar am Suffix im Dateinamen, z.B. `_l_english.yml`, `_l_german.yml` etc.

Jede YAML-Lokalisierungsdatei beginnt mit einer Sprachkennzeichnung als erster Zeile, z.B. `l_german:` für die deutsche Datei. Danach folgen Einträge pro Zeile im Format:

```
SCHLUESSEL:0 "Übersetzter Text"
```

Beispiel: `GER_fortification_studies:0 "Festungsstudien"` würde den Namen des Fokus `GER_fortification_studies` setzen. Für Beschreibungen konventionell `<key>_desc`: `GER_fortification_studies_desc:0 "Dieser Fokus verbessert die Befestigungen..."`. Die `:0` Syntax kommt aus Clausewitz-Engine und trennt den Schlüssel vom Text (in HOI4 wird der Teil vor dem Anführungszeichen ignoriert, man schreibt fast immer `:0`).

Wichtig – Encoding: Alle Lokalisierungsdateien **müssen im UTF-8 mit BOM** gespeichert sein [53] , damit Sonderzeichen korrekt erkannt werden. Insbesondere für nicht-lateinische Schriftzeichen oder Umlaute

im Deutschen ist BOM erforderlich. Viele Editoren (Notepad++) erlauben das explizite Konvertieren (Achtung: BOM ist ein Byte Order Mark am Anfang der Datei). Falls die Kodierung falsch ist, werden Texte nicht geladen. Die Fehlermeldung im Log lautet meist „Localization file must be in utf-8-bom encoding“ [54].

Lokalisierungsregeln: - Kommentare sind mit `#` am Zeilenanfang möglich (wird ignoriert). - Keine Tabs verwenden (nur Leerzeichen), YAML ist sensitive bei Einrückungen – aber HOI4 nutzt kaum verschachtelte Strukturen in Loc, meist flache Key:Value Paare. - **Platzhalter:** HOI4 unterstützt dynamische Werte in Texten, notiert als `£`-Zeichen mit Code (z.B. `£STABILITY` zeigt das Stabilitäts-Icon, oder `[GetName]` in Beschreibungen ruft z.B. Ländervariablen ab). Diese muss man genau richtig schreiben; das Wiki oder vorhandene Loc-Texte bieten Beispiele. Hier kann man leicht Fehler machen, die dann als wörtliche Platzhalter im Spiel sichtbar bleiben. - **Geschlecht/Plurals:** In HOI4 gibt es begrenzte Unterstützung für Variablenflexion, aber da es primär ländliche Bezeichnungen sind, wird das selten gebraucht (im Gegensatz zu CK2/CK3). Für Berater oder Generäle kann man je nach gender im Loc z.B. [He/She] Konstrukte verwenden, aber das geht zu sehr ins Detail.

Lokalisierungs-Workflow für Mods: - Man kann nur die gewünschten Sprachen ergänzen. Viele Mods liefern nur englische Texte (also nur `_l_english.yml`). Das Spiel zeigt dann englische Texte auch auf Deutsch an, falls keine deutsche Übersetzung existiert. Für einen vollständigen deutschen Mod sollte man analog `_l_german.yml` erstellen. - **Schlüssel konsistent halten:** Typos in den Keys führen zu „KEY:0“ Placeholder im Spiel. Daher bei Änderungen von Keys in Scripts nicht vergessen, die Loc anzupassen. - **Zeilenumbrüche:** Im Gegensatz zu EU4 verwendet HOI4 in Loc-Strings `\n` für Zeilenumbruch in Beschreibungstexten. - **Kein Semikolon-Format:** Wer von älteren Paradox-Spielen kommt: HOI4 nutzt nicht mehr das mehrsprachige Semikolonformat in einer Datei, sondern getrennte Dateien pro Sprache. Also niemals Semikolon in HOI4-Loc verwenden; das wird als Teil des Texts interpretiert. In HOI4 braucht jede Sprache ihre eigene Datei oder zumindest eigenen Abschnitt.

Typische Fehlerquellen: - **Falsche Kodierung:** Dies ist der häufigste Grund, warum Texte nicht angezeigt werden. Wenn plötzlich alle Loc fehlt, liegt fast immer ein BOM-Problem oder Syntaxfehler in einer .yml vor. Tipp: in Notepad++ sollte unten rechts "UTF-8 BOM" stehen. [55] - **Anführungszeichen/ Format:** Vergessene Anführungszeichen oder zusätzliche sind fatal. Wenn ein Anführungszeichen im Text selbst gebraucht wird, muss man es escapen (meist vermeiden oder `'` verwenden). YAML akzeptiert keine ungescannte Quotes. - **Schlüssel doppelt:** Wenn zwei verschiedene Mods (oder Dateien) denselben Lokalisierungsschlüssel definieren, überschreibt einer den anderen – je nach Ladeordnung. Das führt zu Inkonsistenzen. Daher möglichst eindeutige Keys (z.B. Prefix mit Mod-Kürzel: `MYMOD_event1_title` statt generisch `event1_title`). - **Umlaute in Keys:** Der Schlüssel selbst sollte ASCII bleiben (keine Umlaute), nur im Wert darf Unicode rein. - **Fehlende Lokalisierung für neue Sprachen:** Wenn man eine Mod beispielsweise nur in Deutsch schreibt und jemand spielt auf Englisch, sieht er die Keys. Daher nach Möglichkeit zumindest englische Default-Loc mitliefern, auch wenn Mod auf eine Sprache fokussiert ist.

10. Grafik- und GUI-Modding (Interface, GFX, Sprites, Icons)

Übersicht: Grafisches Modding in HOI4 umfasst das Ändern oder Hinzufügen von Icons, Porträts, Interface-Layouts und anderen visuellen Elementen. Die Assets sind aufgeteilt in Bilddateien (im Ordner `gfx/`) und Definitionen (im Ordner `interface/`). HOI4 verwendet ein eigenes Sprite-System: *spriteTypes* (für Grafiken) und *textSpriteTypes* (für Text-Grafiken).

Icons und Sprites: Um ein neues Icon (z.B. für einen Fokus, eine Idee, eine Einheit) hinzuzufügen, muss man zwei Dinge tun: 1. Die Bilddatei in den passenden gfx-Unterordner legen (Format meist DDS oder

TGA). Z.B. Fokus-Icons gehen nach `gfx/interface/goals/` als .tga 95×85 px. 2. Eine Definition in einer `.gfx` Datei (im `interface`-Ordner) hinzufügen, die dem Spiel den Bildpfad und einen internen Namen mitteilt.

Eine GFX-Definition (Ausschnitt) sieht so aus:

```
spriteTypes = {
    spriteType = {
        name = "GFX_my_new_icon"
        texturefile = "gfx/interface/my_folder/my_icon.dds"
        noOfFrames = 1
        mipmap = true
    }
}
```

Dies folgt einem festen Format [56]: Man definiert innerhalb eines `spriteTypes`-Blocks einzelne `spriteType`-Einträge mit eindeutigen **name**. Der Name muss immer mit `GFX_` beginnen und darf keine Leerzeichen oder Sonderzeichen enthalten [57]. `texturefile` gibt den Pfad zur Bilddatei an. `noOfFrames` ist relevant für animierte Sprites (z.B. Flaggen haben mehrere Frames in einem Strip); bei statischen Icons = 1. `mipmap = true` generiert verkleinerte Versionen für Zoom-Out (meist immer true für UI-Elemente).

Hat man das definiert, kann man diesen `name` in allen Scripts verwenden, wo ein Icon gebraucht wird (z.B. `icon = GFX_my_new_icon` in Focus oder Idea). Das Spiel lädt beim Start alle .gfx Dateien – falsch formatierte führen zu Crash oder Fehler.

Beispiel – Fokus-Icon: Fokus-Icons sind in `interface/goals.gfx` definiert [58]. Will man ein neues hinzufügen, kann man auch eine eigene .gfx Datei (z.B. `my_mod_icons.gfx`) erstellen, die vom Mod geladen wird. Wichtig ist nur, dass die Datei im mod mit dem gleichen Pfad unter interface liegt, damit das Spiel sie erkennt.

Portraits (Charakterbilder): Ähnlich: Bilder in `gfx/leaders/` oder `gfx/interface/ideas/` etc., und Def in `.gfx` mit `spriteType`. Für Country-Leader-Portraits gibt es oft Namenskonvention `GFX_portrait_<leadername>`.

Interface (.gui Dateien): Dies sind Textdateien, die das Layout von UI-Elementen definieren. Zu finden unter `interface/` mit diversen Namen (z.B. `countrytechtreeview.gui` für den Tech-Bildschirm, `nationalfocusview.gui` für Fokus-Baum, etc.). GUI-Modding ist **komplex**, da man Koordinaten und Widgets manuell anpassen muss. Dennoch ist es für manche Modding-Vorhaben nötig, z.B. zusätzliche Buttons einfügen, Fenster vergrößern, neue UI-Anzeigen schaffen (man denke an Mods, die z.B. Ressourcen extra anzeigen).

Die HOI4-GUI-Sprache ähnelt anderen Clausewitz-Spielen: Es gibt Fenster-Container, Labels, Buttons, die man positionieren kann. Beispiel-Snippet eines GUI-Elements:

```
containerWindowType = {
    name = "my_window"
    position = { x=100 y=200 }
```

```
    size = { width=300 height=200 }
    moveable = true
    Button = { ... }
    text = "..."
}
```

Man muss die existierenden GUI-Dateien überschreiben oder per Mod ersetzen, um Änderungen zu erzielen. Mods sollten *so wenig wie möglich in GUI-Dateien ändern*, da jeder Patch hier schnell Inkompatibilitäten erzeugt. Oft reicht es, via Sprites Änderungen vorzunehmen (z.B. Grafiken austauschen).

Grafikformate: HOI4 nutzt DDS (DXT5) für viele Icons, TGA für Fokus-Icons (wegen besseren Alphakanälen). Es unterstützt auch PNG, aber Paradox bleibt meist bei DDS/TGA. Wichtig: Größe/Seitenverhältnis möglichst wie Original, sonst kann das UI verzerren (es sei denn, man passt die GUI an).

Mapping und Map-Grafik: Neben UI gibt es auch Map-Modding (Terrain-Grafiken, Mapmode-Farben etc.), was aber seltener im „Modding-Guide“ gemeint ist.

Typische Fehlerquellen: - **Falscher Pfad oder Name:** Ein Buchstabendreher im Pfad oder Name im .gfx führt dazu, dass das Bild nicht geladen wird. Das Spiel zeigt dann statt Icon oft einen pinken Platzhalter oder gar nichts. - **SpriteName nicht übereinstimmend verwendet:** Wenn man z.B. ein Icon `GFX_my_tank` definiert, aber in der Techdatei `icon = GFX_my_tank_icon` schreibt, passiert nichts (und es gibt keine klare Fehlermeldung). Name muss exakt passen [59]. - **Unsupported Format:** JPEG oder andere Formate gehen nicht. Falls ein .png mit falschem Encoding vorliegt, kann das Probleme machen – DDS/TGA sind sicherer. - **Interface Konfiguration:** Schon eine fehlende Klammer in einer .gui Datei kann das UI des gesamten Spiels unbrauchbar machen (im schlimmsten Fall Crash beim Laden des Interfaces). Beim GUI-Modding daher sehr sorgfältig Syntax einhalten. Das Error.log weist GUI-Fehler aus (Zeilennummer), was die Fehlersuche erleichtert. - **Mod überschreibt ganze Interface-Datei:** Falls zwei Mods dieselbe .gui ändern, sind sie inkompatibel. Hier muss man solche Änderungen bewusst gestalten oder dokumentieren. - **Animierte Sprites:** Wenn `noOfFrames` und die tatsächliche Bilddatei nicht zusammenpassen (z.B. 10 Frames angegeben, aber Bild hat andere Auflösung), wird nur Mist angezeigt. - **Größenverhältnis:** Bei Portraits muss man auf aspect ratio achten. Wenn man sehr abweichende Bildabmessungen nutzt, werden sie im Spiel gequetscht. Empfohlene Größen: 156x210 px für Leader, 110x150 für Adviser (oder entsprechende 2x für high DPI). Fokus-Icons 95x85 px. Einheitensymbole 82x82 px, Tech-Icons 84x84 px. Abweichungen sollte man in der GUI einstellen, sonst leidet die Qualität.

11. Audio-Modding (Sounds, Musik)

Musik-Modding: HOI4 ermöglicht es, eigene Musikstücke hinzuzufügen, inklusive eigener Radiostationen (seit *Battle for the Bosporus* DLC gibt es Radiosender im Spielmenü). Die gängigsten Änderungen für Mods: - **Neue Musikstücke einfügen:** Man platziert die Audiodateien (OGG-Format empfohlen) im Mod-Verzeichnis, typischerweise in `music/` oder einem Unterordner. Dann erstellt man eine oder mehrere `.txt` Dateien (z.B. `mymod_music.txt`) im Ordner `music/` des Mods. Darin definiert man innerhalb eines `music = { ... }` Blocks die Tracks.

Beispiel Musikdefinition in der Textdatei:

```
music = {
    song = {
        name = "my_mod_song"
        file = "music/my_mod_song.ogg"
        volume = 0.8
        chance = {
            modifier = {
                factor = 0    # nicht spielen, wenn im Krieg
                has_war = yes
            }
        }
    }
}
```

- **name:** Ein interner Bezeichner. - **file:** Pfad zur Musikdatei. - **volume:** Optional, Lautstärkeanpassung. - **chance:** Man kann Bedingungen setzen, wann das Lied gespielt wird. Oft nutzen Mods `chance` mit `has_war` oder bestimmten Ländern/Ideologien, um thematische Musik richtig einzusetzen. (Im Beispiel würde die Chance auf 0 gehen, wenn im Krieg, sodass das Lied nur in Friedenszeiten läuft.) - **historical_year / priority:** Es gibt Parameter, um Musik ab/nach bestimmten Jahren häufiger zu spielen, sodass z.B. Vorkriegszeit andere Musik hat als Nachkriegszeit.

- **Radio-Stationen:** Um einen eigenen Sender hinzuzufügen, muss man eine Datei unter `music` erstellen, die Station definiert, z.B.:

  ```
  radio_station = {
      id = "RADIO_STATION_MYMOD"
      name = "MYMOD Radio"
      icon = GFX_radio_mymod
      song = "my_mod_song"
      song = "my_mod_song2"
  }
  ```

  So eine Station kann mit mehreren Songs befüllt werden. Zusätzlich braucht man ein Icon in gfx (128x128) und Lokalisierung für den Namen des Senders (Key in `music_station_l_english.yml` z.B.). Man kann auch bestehende Stationen erweitern statt neue zu machen.

- **Soundeffekte:** Soundeffekte (Waffensounds, GUI-Sounds) sind schwieriger zu modden, da HOI4 auf das **WWise** Audiosystem setzt. Vieles ist in `.asset` Dateien und Audiobanken definiert, was außerhalb des einfachen Edits liegt. Im Prinzip kann man existierende Sounds ersetzen, indem man eine Datei mit gleichem Namen in `sound/` bereitstellt. Aber neue Sound-Trigger einzubauen erfordert das Bearbeiten von .asset files (z.B. `sound.sfx` oder so ähnlich), was kaum dokumentiert ist. Meist beschränkt sich Audio-Modding auf Musik.

Typische Fehlerquellen: - **Falsches Format:** Die Musik muss als `.ogg` vorliegen (mp3 geht oft nicht, oder führt zu Absturz wenn lizenziertes Format ohne Unterstützung)[60]. - **Pfad nicht übereinstimmend:** Wenn `file = "music/abc.ogg"` angegeben ist, muss die Datei genau dort liegen. Groß/Kleinschreibung spielt unter Windows keine Rolle, unter Linux schon. - **Kein Eintrag**

in .txt: Ein häufiges Problem: Man fügt zwar .ogg Dateien hinzu, vergisst aber die `.txt` Definition – dann kennt das Spiel die Musik nicht. - **Lautstärke und Reihenfolge:** Mods, die sehr viele Songs hinzufügen, müssen bedenken, dass das Spiel alle Songs mehr oder minder random abspielt (nach Station getrennt). Ohne passende `chance`-Modifikatoren kann es passieren, dass z.B. thematisch unpassende Musik in Kriegszeiten läuft. Hier muss man sorgfältig die Chance-Blöcke setzen. - **Mehrere Mods ändern Musik:** Musikmods addieren sich meist problemlos (Songs landen einfach zusätzlich in der Liste). Konflikte gibt es nur, wenn zwei Mods denselben Radio-Sender mit gleichen ID ändern – dann überschreibt einer den anderen. - **Audio Output:** Das Spiel mischt alle Songs in einen *continuous soundtrack*. Falls ein Lied nicht spielt, kann man Debug-Ausgabe schwer bekommen – am besten testet man, ob es in der In-Game Songliste (über das Radio-UI) auftaucht. Wenn nicht, ist was an der Definition falsch. - **Soundeffekte austauschen:** Wenn man z.B. andere Waffensounds will, könnte man die Bank-Dateien überschreiben, aber das ist riskant (kann zu Multiplayer-Inkompatibilität führen). Einfacher: vorhandene .ogg in `sound/` ersetzen (z.B. `Artillery_fire.ogg` durch eigenen Sound). Muss aber exakt gleicher Name und Länge sein, sonst asynchron.

12. Scripting-Regeln und Syntax (Befehle, Trigger, Effekte)

Dieser Abschnitt behandelt die *allgemeine Skriptsprache* von HOI4, die in Fokusbäumen, Events, Entscheidungen, AI etc. überall zum Einsatz kommt. Wichtig ist, Konzepte wie **Trigger** (Bedingungen) und **Effekte** (Kommandos) zu unterscheiden:

- **Trigger** sind Bedingungen, die entweder *wahr* oder *falsch* ergeben. Sie werden verwendet in `trigger = { }` Blöcken, in `available = { }`, `visible = { }`, als Auslöser für Events, Entscheidungen, KI etc. Beispiele: `has_war = yes` (ist im Krieg), `num_of_factories > 50` (mehr als 50 Fabriken) [61], `tag = GER` (Land ist Deutschland) [61]. Trigger können kombiniert werden mit logischen Operatoren wie `AND` (implizit, wenn mehrere Bedingungen in einer Klammer stehen, müssen alle zutreffen), `OR = { ... }` (mindestens eine muss wahr sein), `NOT = { ... }` (Bedingung invers) [62], `NOR`, `XOR` etc. sowie Vergleichen `<`, `>`, `<=`, `>=`. Auch Verschachtelung ist möglich. Beispiel:

  ```
  trigger = {
      tag = GER
      NOT = { has_government = communism }  # Nicht kommunistisch
      OR = {
         num_of_factories > 50
         has_nuclear_reactors = yes
      }
  }
  ```

  Dieser Trigger erfordert: Land Tag GER, nicht kommunistisch und (Fabriken >50 oder besitzt Kernreaktoren).

- **Effekte** sind Befehle, die das Spiel verändern, z.B. `add_political_power = 100`, `declare_war_on = { target = FRA type = conquest }`, `set_country_flag = XYZ`. Sie werden in `effect = { }` Blöcken oder als direkte Folge von Aktionen verwendet. Effekte haben oft Parameter. Beispiel: `white_peace = POL` – bewirkt, dass der aktuelle Krieg zwischen Aktor und POL mit Weißem Frieden endet [63]. Oder `give_guarantee = POL` – aktives Land gibt Polen eine Garantie [63]. Man sieht: Manche Effekte verlangen, dass man ein

Ziel angibt (hier den Tag POL). Andere sind allgemeiner (z.B. `add_stability = -0.2` braucht kein Ziel, es wirkt auf das aktuell gescopte Land).

Scopes (Kontext): HOI4 hat verschiedene Wirkungskreise. Die wichtigsten sind *Country-Scope*, *State-Scope*, *Unit/Character-Scope*. Wenn man z.B. in einer `country_event`-Definition ist, sind Trigger standardmäßig auf das Land bezogen, das das Event bekommt (**ROOT = Land**). Will man innerhalb eines Effekts auf ein anderes Land referenzieren, nutzt man den Tag oder andere Scope-Wechsel. Beispiel:

```
GER = { add_ideas = democratic_drift }
```

aus einem Fokus-Effekt würde dem Land mit Tag GER eine Idee geben [64]. Innerhalb der geschweiften Klammer wechselt der Scope auf Deutschland (GER) [64], führt den Befehl dort aus, dann kehrt er zurück. Alternativ kann man `FROM` oder `ROOT` in manchen Konstellationen nutzen, z.B. in einem Event-Option innerhalb eines `FROM = { }` Blocks bezieht sich `FROM` auf den vorherigen Scope (etwa der Aggressor in einem Friedensevent). Das *Scope-Management* ist eine der schwierigsten Aspekte für Neulinge.

Kurz: - `ROOT` = das aktuelle Haupt-Subjekt (z.B. Land, das den Fokus/Entscheidung/Event hat). - `THIS` = wird selten in HOI4 verwendet, meist analog zu ROOT. - `FROM` = das Subjekt, von dem man in einem komplexen Trigger/Effekt kommt (z.B. in einem Event, FROM kann das auslösende Land sein wenn Event per Befehl kam). - Genaue Verwendung würde den Rahmen sprengen; im Modding-Wiki gibt es dazu Erklärungen.

If/Else Strukturen: Man kann bedingt Effekte ausführen mit der `if = { limit = { <Trigger> } <Effekte> } else = { <Effekte> }` Syntax. Beispiel:

```
if = {
    limit = { has_idea = party_unity }
    remove_ideas = party_unity
}
else = {
    add_political_power = 50
}
```

Heißt: Wenn das Land den Nationalgeist *party_unity* hat, dann entferne ihn, andernfalls gebe 50 PP. `limit` fungiert als die IF-Bedingung. Nested ifs sind auch möglich. Das ist nützlich in Events oder Fokus-Belohnungen, um unterschiedliche Pfade umzusetzen.

Loops: HOI4 ermöglicht begrenzte Schleifen, z.B. `every_country = { limit = { ... } <Effekte> }` um für jedes Land etwas zu tun, was die Bedingung erfüllt. Oder `random_owned_state = { <Effekte> }` um einen zufälligen Staat auszuwählen. Diese müssen mit Bedacht eingesetzt werden (Performance!).

Gültigkeit und Scope von Befehlen: Einige Befehle funktionieren nur in passendem Scope. Z.B. `add_core_of = TAG` kann nur im State-Scope stehen (denn es fügt dem Staat einen Kern hinzu). Versucht man das im Country-Scope, passiert nichts und es gibt einen Fehler im Log. Um so etwas im Kontext eines Landes zu tun, muss man erst in den State-Scope wechseln, z.B.:

```
state = 123 {
    add_core_of = GER
}
```

Das würde im Land-Kontext State 123 nehmen und GER als Kern geben. Umgekehrt gibt es Effekte, die nur im Länderscope Sinn machen (z.B. `join_faction` geht nur für ein Land).

Validierungs-Hilfen: - Paradox liefert in der Installation unter `Hearts of Iron IV/documentation/` oft Dateien `trigger_documentation.txt` und `effects_documentation.txt`, in denen *alle* Trigger und Effekte mit kurzer Erklärung aufgelistet sind [65]. Diese sind Gold wert, um nachzuschlagen, welche Befehle existieren. Für Mods wichtig: es gibt keinen „Neuen Befehl“ erfinden – man muss aus diesem Pool nutzen. - Das **Error Log** (im Userordner unter Logs) listet ungültige Trigger/Effekte beim Laden. Wenn man z.B. `add_unit` im Falschen Scope nutzt, gibt’s eine Warnung "*Invalid Scope*". - Das Wiki hat Seiten *Triggers* und *Effects* mit Erklärungen [66].

Beispiele für Kombinationen: - Man kann Trigger/Effekte auch mit Variablen nutzen (z.B. Zählen von Ereignissen). HOI4 hat *country scope variables* und *global variables*, die man mit `set_variable` und in Triggern mit `check_variable` nutzen kann – fortgeschritten, aber erwähnenswert. - Zuweisungen: `=`, `<`, `>` funktionieren auch auf viele Trigger (Zahlenvergleiche). - Zeit-Trigger: `date > 1940.1.1` prüft Datum. `has_dlc` prüft ob DLC aktiv.

Typische Fehlerquellen: - **Klammern nicht geschlossen:** Die häufigste Syntaxsünde – führt zu Paradox Error am Start ("Unexpected token" oder Ähnliches). Hier hilft ein Texteditor mit Klammermatching. - **Falscher Scope:** Der Effekt wird im Spiel einfach nichts tun, oder der Trigger immer false. Beispiel: `owns_state = 5` muss im Country-Scope stehen (Frage: Land besitzt Staat 5?), im State-Scope würde es bedeuten „Besitzt der Staat 5?“ was unlogisch ist. Die Engine merkt das und loggt einen Error. - **Zahlen als Strings:** In Loc-Schlüsseln oder Variablennamen kann man keine Zahlen am Anfang haben (Beispiel Key "1936_start" ist problematisch, besser "start_1936"). In Script-Triggern kann es egal sein, aber bei Variablen streng drauf achten. - **Groß/Kleinschreibung:** Trigger/Effektnamen sind idR. case-sensitive, Tags jedoch nicht unbedingt (GER vs ger funktioniert meist, aber man schreibt Tags immer groß). Bei File-Path und Loc-Keys gilt genau schreiben. - **Operator Missverständnisse:** `OR = { A B }` heißt A oder B muss *true* sein. Manchmal sieht man Modder `OR = { A = X B = Y }` schreiben in der Annahme, das würde "A gleich X oder B gleich Y" bedeuten – das ist falsch formatiert. Korrekt:

```
OR = {
   A = X
   B = Y
}
```

Jede Zeile innerhalb OR ist wieder ein Trigger. - **Limit/Trigger in Effekten:** In on_actions z.B. nutzt man oft:

```
on_peaceconference_ended = {
    effect = {
       if = { limit = { FROM = GER } ... }
```

```
    }
}
```

Hier ist `limit` innerhalb eines Effekts! Das kann verwirren, aber in solchen Hooks ist der Code im Effekte-Kontext mit eingebetteten Triggern in limit-Blöcken.

- **Unsortierte Listen:** Manche Trigger liefern Listen zurück (z.B. alle Staaten mit xy); die Engine kann daraus keine eindeutige Auswertung machen, das wird aber im Regelfall vermieden oder es existieren separate count-Triggers.
- **Entwicklerkommandos im Release:** Es gibt Debug-Effekte (z.B. `log = "text"` schreibt ins Log) [67] . Diese sollte man in veröffentlichten Mods rausnehmen, da sie ggf. Performance kosten oder ungewollt Infos leaken.

13. Kompatibilität, Fehlerdiagnose, Patch-Pflege

Mod-Struktur und .mod-Datei: Eine HOI4-Mod wird durch eine Descriptor-Datei (meist `.mod` im mod Verzeichnis oder via Steam Workshop) definiert. Darin stehen u.a. `supported_version` und möglicherweise `dependencies` oder `replace_path` Angaben. **Kompatibilität** bedeutet hier vor allem, dass die `supported_version` im Mod-Descriptor auf die aktuelle Spielversion eingestellt ist, sonst markiert das Spiel die Mod als veraltet (was Spieler zwar ignorieren können, aber es ist ein Hinweis). Nach jedem Patch sollte man diese Versionsnummer anheben, sobald die Mod kompatibel getestet ist [68] .

Patch-Pflege: Paradox-Patches können Mod-Inhalte brechen. Daher empfiehlt es sich: - **Changelog studieren:** Was wurde in Vanilla geändert? Wenn z.B. neue Staaten hinzugefügt wurden und die Mod überschreibt `00_states.txt`, muss man diese integrieren. Oder wenn neue Ideen eingeführt wurden, auf ID-Konflikte achten. - **Merge vs Override:** HOI4 merged einige Dateien nach Keys (z.B. Ideas, National Focus – mod hinzugefügte Items werden zusätzlich geladen, sofern keine ID-Kollision). Andere Ordner wiederum werden nicht gemerged – notably `history/*` Dateien werden alle geladen, aber zwei History-Dateien für selben Tag summieren ihre Effekte potenziell (bzw. beide werden ausgeführt – was zu doppelten Divisionen führen könnte). Speziell `history/countries` ist tricky: zwei Dateien mit gleichem Tag werden beide angewandt, man nutzt also eher override per replace_path wenn man einen Länder-Start komplett ersetzen will [69] . Bei Patch-Updates sollte man prüfen, ob die Mod irgendwo replace_path nutzt und ob das noch nötig ist oder angepasst werden muss.

- **Kompatibilitätstests:** Nach einem Patch empfiehlt sich, die Mod im Debug-Modus zu starten und ins Fehlerlog zu schauen. Falls hunderte Zeilen Error (z.B. "Unexpected token in ..."), sind vermutlich Änderungen in den Vanilla-Files, die Mod-Einträge ungültig machen. So entdeckt man auch überschriebenene Defines oder Parameteränderungen.

Fehlerdiagnose (Debugging): HOI4 hat ein sehr hilfreiches Fehlerlogging. Startet man das Spiel mit dem Parameter `-debug`, wird im Spielverzeichnis unter `logs/` die `error.log` ausführlich geschrieben. Auch ohne Debug werden viele Fehler geloggt, aber im Debug mehr. Typische Vorgehensweise: - **error.log prüfen:** Bei Crash oder fehlenden Dingen immer dort schauen. Oft steht genau, was fehlt oder wo ein Script nicht gelesen werden konnte. - **instantError nach Laden:** HOI4 zeigt im Hauptmenü unten rechts einen kleinen Hund (“error dog”) an, wenn Fehler erkannt wurden. Man kann darauf klicken, um das Fehlerlog direkt ingame zu sehen [70] . - **Gezielt Filtern:** Das Log kann man nach Keywords durchsuchen (z.B. eigenen Mod-Präfix). - **Crash ohne Log:** Kommt vor bei z.B. Grafikproblemen oder ganz frühen Loading-Fehlern (falsche Syntax in critical files). Hier mal `exceptions.log` ansehen oder schrittweise Mod-Teile deaktivieren, um Eingrenzung zu machen. -

Syntax-Check: Nutzen von Tools oder Editor mit Paradox-Script-Highlight hilft sehr. Es gibt Community-Tools (z.B. CWTools, oder Validate Script in Visual Studio Code), die HOI4-Syntax prüfen und Logiken validieren, was enorm bei großen Mods hilft.

Häufige Fehler und ihre Lösung: - *"Unexpected Token: XYZ"* im Error – weist meist auf eine kaputte Klammer-Struktur hin. Mit Editor Klammern zählen, ggf. Datei halbieren um Abschnitt zu finden. - *"Invalid Scope or trigger"* – Ein Trigger/Effekt wird am falschen Ort genutzt. Prüfen mit Doku, ggf. Scope wechseln oder Trigger ändern. - *"Undefined Localization key"* – Ein Lokalisierungsschlüssel fehlt. Log zeigt den Key. Suchen wo er sein sollte und in Loc-Datei ergänzen. - *Grafik nicht gefunden:* Im Log steht manchmal, wenn ein Texturefile-Pfad falsch ist. Sonst bemerkt man es visuell. - *Crash bei Init ohne Fehlermeldung:* Oftmals ein Problem mit `interface/*.gfx` oder .gui Datei. Hier einzeln prüfen, oder probeweise die letzte Änderung rückgängig machen. - *Stuck on loading:* Hängt das Laden ewig an z.B. *"Initializing Maplogic"* oder so, kann es an einer fehlerhaften State/Province-Zuordnung liegen. Das Log verrät oft: z.B. *"State X contains province Y multiple times"* oder *"Province Z assigned to invalid region"*. Solche Meldungen suchen.

Kompatibilität zwischen Mods: - HOI4 Mods können konfliktreich sein, wenn sie dieselben Dateien bearbeiten. Anders als bei EU4 fehlt ein Prioritätssystem (außer man manuell mit dependencies arbeitet). Als KI kann man das Konzept verstehen: Der *letzte geladene Mod* mit einer Änderung an derselben Datei wird gewinnen (Load-Reihenfolge kann man in `mod/load_order.txt` beeinflussen über Launcher). Wenn eine Mod also z.B. die `common/technology/armor.txt` verändert und eine andere auch, gewinnt die mit späterer Priorität. In praktischer Nutzung kann man davon ausgehen: Workshop Mods sind alphabetisch nach Mod-Name geladen, aber das kann variieren. - Für KI/AI Mods: man kann mehrere AI-Strategie Mods kombinieren, solange sie unterschiedliche Länder betreffen oder additive Pläne haben. - Grafikmods sind meist kompatibel, sofern sie unterschiedliche Aspekte anfassen (eine Mod ändert Division-Sprites, die andere Flaggen – kein Problem). - Total Conversion Mods sind untereinander nicht kompatibel (laden komplett eigene Files).

Patch-Pflege konkret: - Nach jedem HOI4 Patch am besten die eigenen mod-Dateien mit den neuen Vanilla-Dateien vergleichen (Tools wie WinMerge). Besonders in `common/` und `history/`. Anpassungen dann manuell einpflegen. - Paradox ist relativ mod-freundlich, kündigt Breaking Changes oft in Dev Diaries an, aber Überraschungen gibt es immer. - Mods sollten eine Versionsnummer haben und Changelog für User, sodass diese wissen, ob mod mit neuem Patch X läuft.

Abschließend: Fehlerdiagnose ist ein kontinuierlicher Prozess beim Modden. Es empfiehlt sich, regelmäßig das eigene Mod-Log zu **säubern** – also alle bekannten Fehler fixen – bevor man nach neuen Fehlern sucht, da manchmal harmlose Warnungen echte Probleme überdecken können [71]. Und wenn das Spiel abstürzt, immer erst schauen, ob vorher im error.log schon etwas Schwerwiegendes stand (oft crash das Spiel *nach* dem ersten fatalen Fehler). In diesem Sinne: Happy Modding!

Quellen: Die oben genannten Informationen basieren auf der offiziellen Paradox-Interactive HOI4 Modding-Dokumentation [18] [28], Community-Wikis und Modding-Erfahrungswerten [72] [73]. Wichtige Referenzen und Beispiele wurden aus dem Hearts of Iron IV Wiki und Foren entnommen [11] [61], um technische Genauigkeit zu gewährleisten. Bitte konsultieren Sie die verlinkten Quellen für detaillierte Listen von Befehlen und weitere Modding-Tipps.

---

[1] [3] [4] [5] [6] [7] [8] [9] [72] Common/national focus | Hoi4 modding Wiki | Fandom
https://hoi4-modding.fandom.com/wiki/Common/national_focus

2 National focus - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/National_focus

10 12 Common/country tags | Hoi4 modding Wiki | Fandom
https://hoi4-modding.fandom.com/wiki/Common/country_tags

11 63 64 Tag | Hoi4 modding Wiki | Fandom
https://hoi4-modding.fandom.com/wiki/Tag

13 r/hoi4modding on Reddit: When changing country colour in my mod ...
https://www.reddit.com/r/hoi4modding/comments/p070d9/when_changing_country_colour_in_my_mod_the/

14 Cosmetic tag modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Cosmetic_tag_modding

15 Ideology modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Ideology_modding

16 History/units - Hoi4 modding Wiki - Fandom
https://hoi4-modding.fandom.com/wiki/History/units

17 48 57 Character modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Character_modding

18 19 21 22 23 24 25 History/states | Hoi4 modding Wiki | Fandom
https://hoi4-modding.fandom.com/wiki/History/states

20 State modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/State_modding

26 Strategic region modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Strategic_region_modding

27 How do you mod Air/Strategic Regions? : r/hoi4modding - Reddit
https://www.reddit.com/r/hoi4modding/comments/5bw263/how_do_you_mod_airstrategic_regions/

28 29 52 Technology modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Technology_modding

30 Equipment modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Equipment_modding

31 Division modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Division_modding

32 Equipment - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Equipment

33 Idea modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Idea_modding

34 35 36 37 38 39 40 61 62 73 Events folder | Hoi4 modding Wiki | Fandom
https://hoi4-modding.fandom.com/wiki/Events_folder

41 Decision modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Decision_modding

42 HoI 4 - Double "static_modifiers.txt" and "00_static_modifiers.txt"
https://forum.paradoxplaza.com/forum/threads/hoi-4-double-static_modifiers-txt-and-00_static_modifiers-txt.1078742/

43 Modifiers - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Modifiers

44 Modifiers | Modding Cooperative Hoi4 Wiki - Fandom
https://mod-coop-hoi4.fandom.com/wiki/Modifiers

45 Static/Difficulty Modifiers | Paradox Interactive Forums
https://forum.paradoxplaza.com/forum/threads/static-difficulty-modifiers.942364/

46 List of modifiers - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/List_of_modifiers

47 How do I make a custom trait? - HOI4 Modding
https://edge.hoi4modding.com/forum/general-chat/2147-how-do-i-make-a-custom-trait

49 Custom Commander Traits Tutorial : r/hoi4modding - Reddit
https://www.reddit.com/r/hoi4modding/comments/o5ij0i/custom_commander_traits_tutorial/

50 AI modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/AI_modding

51 National focus modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/National_focus_modding

53 Localisation - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Localisation

54 55 Localization file should be in in utf-8-bom encoding : r/hoi4modding
https://www.reddit.com/r/hoi4modding/comments/e4jaca/localization_file_should_be_in_in_utf8bom_encoding/

56 Graphical asset modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Graphical_asset_modding

58 Where can i find the National Focus Icons in the game's files? - Reddit
https://www.reddit.com/r/hoi4modding/comments/c8lu54/where_can_i_find_the_national_focus_icons_in_the/

59 Discussions - Steam Community
https://steamcommunity.com/workshop/discussions/18446744073709551615/530969757695877640/?appid=394360

60 Guide :: How To Make Custom Radio Stations (Possibly Outdated)
https://steamcommunity.com/sharedfiles/filedetails/?id=2145878376

65 Effect - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Effect

66 Triggers - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Triggers

67 Country creation - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Country_creation

68 Updating Hoi4 mod to 1.6 and MtG compatibility : r/hoi4modding
https://www.reddit.com/r/hoi4modding/comments/b07qyk/updating_hoi4_mod_to_16_and_mtg_compatibility/

69 70 Modding - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Modding

71 Troubleshooting - Hearts of Iron 4 Wiki
https://hoi4.paradoxwikis.com/Troubleshooting