

AI modding - Hearts of Iron 4 Wiki

This is a community maintained wiki. If you spot a mistake then you are welcome to fix it.

The artificial intelligence controlling countries in Hearts of Iron IV can be changed in many different aspects. This includes changing the focuses or technologies it pursues, changing how many units or buildings it should produce and of which type, where it should focus forces, which templates or variants it should aim for, and so on.

MTTH blocks[[编辑](#) | [编辑源代码](#)]

MTTH blocks, named in the game within /Hearts of Iron IV/common/mtth/, are a way to assign a dynamic value to some block, whether it's a [country = { ... } in a focus tree to assign which countries can get it](#), a dynamically-changing variable value (as in /Hearts of Iron IV/common/mtth/), a [mean-time-to-happen value for an event](#), et cetera. As MTTH blocks are spread throughout the entire game, it's incredibly important to know how one works. Its most common application, however, is assigning a base AI weight to a database entry such as a national focus or a technology.

In most cases, marked with ai_will_do, the approach taken by the game is generating a decimal number between 0 and the value of the ai_will_do block, and picking the entry with the highest value. There can be other modifiers applied after the ai_will_do value, such as the ones applied within [AI strategy plans](#) towards focuses and research or the ones that get defined within [Defines](#), which'll multiply the value you get. This can include different ai_will_do values such as what's done with country leader traits and with ideas or characters that use them.

Event options, instead, use ai_chance, which uses [probability-proportional-to-size sampling](#) with a virtual roll of 100 being the final deciding factor. Due to that, in the end, an event option cannot have a probability which isn't a multiple of 1/100 to be picked, although the total sum of each option's weights can be vastly different from 100. In other words, if one option has a value of 1, while the other has a value of 999, the probability for the first option to be picked will be 1% rather than 0.1%.

A MTTH block begins with an assumed value of 1. Further manipulations are done from that value as a starting point. There are 3 value-modifying arguments that can be done, add, base, and factor, done like `factor = 0.3`, `base = 10`, or `add = 5`. `add` adds the specified argument to the value, `factor` multiplies it, and `base`, typically done in the start before any changes, sets it to a different number entirely, akin to multiplying by 0 and adding the number.

[In order for an operation to apply conditionally, a modifier = { ... } block is used. This also functions as a trigger block, with the default scope \(thus also ROOT\) being the country for which the MTTH block is evaluated.](#) Depending on the MTTH block, [there may be additional scopes marked with FROM or FROM.FROM other than the default.](#) The value-modifying arguments function as regular arguments, they can be put in anywhere directly within the `modifier = { ... }` block with almost no difference, whether it's in the end, the beginning, or between triggers.

[Variables](#) can be used within the value-modifying arguments as well. If using a temporary variable that is defined within the `modifier = { ... }` trigger block, then the value-modifying argument of the modifier has to be after the definition of the variable in order for the variable to apply as defined. Variables can also be used outside of the `modifier = { ... }` block and directly inside of the MTTH block itself.

Example[[编辑](#) | [编辑源代码](#)]

```
ai_will_do = {
    base = 10.5

    # If the country is Germany, set the value to 0,
    # causing an early end of the evaluation.
    modifier = { tag = GER factor = 0 }
    modifier = { is_major = yes add = 1 }
    modifier = {
        factor = 3
        add = 2.5
        tag = FRA
    }
    factor = 2

    modifier = {
        set_temp_variable = { t = num_of_civilian_factories }
        add_to_temp_variable = { t = num_of_military_factories }
        divide_temp_variable = { t = 10 }
        round_temp_variable = t
        add = t
    }
}
```

Assuming that GER and FRA are major countries, the result is

0 for GER


74 for FRA

23 for other majors

21 for minors

After calculating that value, the total number of civilian and military factories within the country, divided by 10 and rounded, will get added to the score. If FRA has a total of 43 civilian and military factories as it does in base game, then this will result in 4 being added to the prior 74 for a total of 78, for example.

AI strategies[[编辑](#) | [编辑源代码](#)]

AI strategies are used in order to pursue AI to do or avoid something. This includes diplomatic actions the AI will do, where and how exactly AI should focus and use its land army and the navy, the production lines for buildings and equipment, how AI should handle the intelligence system within the  [La Résistance](#) DLC.

A value within an AI strategy can be either positive or negative, and it being negative will make AI desire to do it less.

Regular AI strategies are stored in /Hearts of Iron IV/common/ai_strategy/*.txt files, however, AI strategies may be defined outside of that file. The [add_ai_strategy effect](#) can add a permanent AI strategy within any effect block, and [AI strategy plans](#) can also include AI strategies defined within of themselves. Overall, /Hearts of Iron IV/common/ai_strategy/*.txt files describe AI strategies that would enable and disable themselves automatically.

Each entry within a /Hearts of Iron IV/common/ai_strategy/*.txt file is a block with the name of the AI strategy. This name can be anything, even allowing overlap between them: this will only appear in AI dumps and the player will never see it. If an AI strategy modifies the chance for AI to pick a diplomatic option with the player, it will be seen as `COUNTRY has strategic reasons to be ...` to the player when hovering over the option. This is an example of an AI strategy with the name of `BHR_invalidate_qatar`:

```
BHR_invalidate_qatar = {
    allowed = {
        original_tag = BHR
    }
    enable = {
```

```

country_exists = QAT
}
abort = {
    has_war_with = QAT
}
abort_when_not_enabled = yes
ai_strategy = {
    type = invade
    id = QAT
    value = 200
}
}

```

Arguments[\[编辑 | 编辑源代码\]](#)

These in particular are arguments for `/Hearts of Iron IV/common/ai_strategy/*.txt` entries, rather for AI strategy entries within them.

`allowed = { ... }` is a trigger block that's **checked only before the game's start**, permanently allowing or disallowing an AI strategy. Typically, only used for country checks (such as `tag = OMA`) or DLC checks that can never be unfulfilled once they are met.

`enable = { ... }` is a trigger block that must be met in order to enable the AI strategy. By default, this being unmet will *not* cancel the AI strategy. However, `abort_when_not_enabled = yes` can be used as a separate argument in order to make that be no longer the case. Unlike `allowed`, this is checked continuously.

`abort = { ... }` is a trigger block that must be met in order to *remove* the AI strategy. It being unmet also prevents it from being added in the first place.

`ai_strategy = { ... }` is, itself, the AI strategy that would be applied. The only argument shared for every AI strategy is `type = <AI strategy type>`. [The rest depends on the AI strategy.](#)



In addition, reversed AI strategies exist. These are used with AI strategies that use `id = TAG` to target towards a specific country to swap it around: that strategy will be enabled for TAG towards the country which meets `enable = { ... }`. In order to mark the AI strategy as reversed, `reversed = yes` is used. `enable_reverse = { ... }` is an additional trigger block required to *enable* this AI strategy. It does not have a default scope, so scoping into a country is required.

An example of a reversed AI strategy is the following:

```

BHR_support_neutrals = {
    allowed = {
        NOT = { original_tag = BHR }
    }
    enable = {
        has_government = neutrality
    }
    enable_reverse = {
        BHR = { has_government = neutrality }
    }
    reversed = yes
    abort_when_not_enabled = yes
    ai_strategy = {
        type = support
        id = BHR
        value = 100
    }
}

```

Without `reversed = yes`, this would make every  [non-aligned](#) country support BHR. Due to that argument, the strategy is reversed: instead, BHR supports every  [non-aligned](#) country.

Types[\[编辑 | 编辑源代码\]](#)

This list may be outdated. A list of every AI strategy can be found within base game's `/Hearts of Iron IV/common/ai_strategy/documentation.info` file.

AI strategies related to diplomatic actions: [展开](#)

AI strategies related to land army management: [展开](#)

AI strategies related to navy management: [展开](#)

AI strategies related to intelligence: [展开](#)

AI strategies related to production, construction, and recruitment: [展开](#)

Other AI strategies: [展开](#)

AI areas[\[编辑 | 编辑源代码\]](#)

AI areas are defined within `/Hearts of Iron IV/common/ai_areas/*.txt` files. These are *only* used in a variety of previously-listed AI strategies, such as [area priority](#). A province may be in several AI areas and it may be in none. Hovering over a province with the debug mode turned on will provide information in which AI areas the province is, if any.

Each AI area is a separate block within the file, with the name of the block being the name of the area. Within these blocks, 2 things can be added:

`continents = { ... }` is a list of continents that make up the AI area. Continents are defined within `/Hearts of Iron IV/map/continent.txt` and assigned to provinces within [their definitions](#) in `/Hearts of Iron IV/map/definition.csv`. The full name of the continent should be used within the AI area's definition, rather than the ID used in the province definition.

`strategic_regions = { ... }` is a list of strategic regions that make up the AI area, by their ID number.

If there are multiple defined, the province has to be in *any* of them.

An example of an AI area is the following:

```
my_ai_area = {
  continents = {
    europe
    africa
  }
  strategic_regions = {
    53 # Caribbean
    189 # Burma
  }
}
```

AI focuses[[编辑](#) | [编辑源代码](#)]

Main article: [AI focuses](#)

AI focuses, defined within `/Hearts of Iron IV/common/ai_focuses/*.txt` files, are used to tell the game which technology categories and focuses the AI should pick depending on its currently-pursued focuses.

AI peace[[编辑](#) | [编辑源代码](#)]

The peace conference behaviour for AI is defined within `/Hearts of Iron IV/common/ai_peace/*.txt` files.

When defining the peace conference AI that would be used by the country, the game picks the first-defined one that meets the prerequisites. In this case, files are loaded sorted by their filename using [ASCII character IDs](#).

When evaluating the peace conference, ROOT (the default scope) is used to represent the currently-evaluated winner and FROM is used to represent the currently-evaluated loser. Additionally, the following [temporary variables](#) exist within:

`taken_states@TAG` is an array of states that are annexed by the country TAG as a part of the peace conference.

`taken_by@123` is the country that took the specified state, in this case 123.

`current_states@TAG` is an array of states that aren't decided upon yet in the peace conference and are under the control of TAG that lost the war.

`subject_states@TAG` is an array of states that are, within the peace conference, set to be transferred to countries that are subjected by TAG.

`subject_countries@TAG` is an array of countries that are, within the peace conference, set to be subjected by TAG.

`subjected_by@123` is the overlord of the country that is set to have the specified state in the peace conference, in this case 123.

`subjected_by@TAG` is an array of countries that have TAG as their overlord.

`liberate_states@TAG` is an array of states that are going to countries that have been liberated by TAG.

`liberate_countries@TAG` is an array of countries that have been liberated by TAG.

The `enable = { ... }` trigger block is used to determine whether the AI should be enabled. If true and no other previously-loaded peace conference AI is true, this AI will be chosen.

There are the following peace options that are used in the game:

`annex` - The country FROM gets entirely annexed by ROOT. This only evaluates the countries.

`liberate` - The country FROM gets liberated by ROOT.

`puppet` - The country FROM gets puppeted by ROOT.

`puppet_all` - The country FROM gets puppeted by ROOT *and* is able to retain all of its states.

`puppet_state` - The state ROOT gets transferred to FROM.FROM, which became a subject of FROM within the peace conference.

`take_states` - The state FROM gets annexed by ROOT.

`force_government` - The country FROM gets its ideology forcefully changed by ROOT.

When evaluating peace conferences, the controller of a given state is the same one that occupied it before the start of the conference while the owner is the original owner.

Each peace conference option is a [MTTH block](#) within the file's definition.

Example:

AI strategy plans[[编辑](#) | [编辑源代码](#)]

AI strategy plans to tell AI what to prioritise depending on circumstances: which advisors to pick, which technologies to research, which AI strategies to apply, which focuses to pick, etc. These are more detailed than general AI strategies, primarily intended to be activated for most of the game to tell the overall plan of a country. Multiple AI strategy plans can be defined and executed at the same time for a country.

AI strategy plans are defined within `/Hearts of Iron IV/common/ai_strategy_plans/*.txt` files. Within these files, a new strategy plan is done as a new block, the name of which must be the same as the internal ID of the plan.

Within that plan, `name = "AI plan's name"` and `desc = "AI plan's description"` decide the name and the description of the strategy plan. *This is never intended to be shown to the player*, so localising it into different languages is never needed. Instead, this is used within the `aiview` console command, which tells info to the developer about what AI wants to prioritise.

`allowed = { ... }` is, similarly to decisions or ideas, is a trigger block **checked only at the game's start**. This is primarily used to tell which country and DLCs to restrict the strategy plan to.

`enable = { ... }` is checked each day if the allowed is met. If `enable = { ... }` is met, then the AI strategy plan will be assigned to the AI regardless of whether `enable = { ... }` turns false later or not. Commonly, [Triggers#has_game_rule](#) is used to make it work with custom game rules deciding what path the AI will pick. [is_historical_focus_on](#) is commonly used with the default AI game rule, and country flags can be used for randomisation, by setting up an [on_startup](#) to set a random one using [random_list](#)

`abort = { ... }` is checked every day in order to make the AI *stop* using this AI strategy plan if `enable = { ... }` is met. Additionally, it must be false in order for the AI strategy plan to be possible to be picked.

`ai_national_focuses = { TAG_focus_name_1 TAG_focus_name_2 }` is a list of national focuses, separated by whitespaces, in the order that the AI should take them. In this example, the AI will try to take `TAG_focus_name_1` first if possible. If it's already taken or `TAG_focus_name_1` is impossible to take, then AI will try to take `TAG_focus_name_2`. If both of the focuses are impossible to take due to being completed or unavailable, then it will move on to other focuses, taking `ai_will_do = { ... }` into consideration. While following a focus order, it ignores `ai_will_do = { ... }`

} values.

`focus_factors = { ... }` assigns a multiplier to `ai_will_do` values of the specified focus. An entry in this block looks like `TAG_focus_name = 3`. In this case, this will make the `ai_will_do` value of the focus be multiplied by 3, assuming AI strategy plan's weight of 1. If the focus has an `ai_will_do` value of 4 after applying modifiers, it'll become 12 if AI is following this strategy plan, and get treated as such. And, of course, a factor of 0 will make the focus be never picked without specification in `ai_national_focuses`. This can serve as a faster-to-write or a more randomised way to make AI follow a political path by making focuses it should never pick have a value of 0.

`research = { ... }` assigns a multiplier to `ai_will_do` values of the specified research categories. An entry in this block looks like `artillery = 3`. In this case, this will make the `ai_will_do` value of every technology within the category be multiplied by 3, assuming AI strategy plan's weight of 1. Other built-in modifiers still apply, but this will increase the likelihood.

Other blocks that also assign bonuses are `ideas = { ... }` and `traits = { ... }`, with the similar formatting. The ideas block is used for individual ideas (such as laws or designers) or advisors (using the `idea_token` in the entry), while traits are for country leader traits that are assigned to the ideas/advisors.

`ai_strategy = { ... }` allows an [AI strategy](#) to apply when the strategy plan is turned on.

`weight = { ... }` is a [MTTH block](#) assigning an overall weight to the plan. This multiplies each factor within the AI strategy plan by the weight before applying them. A weight of 1.25 will turn a focus factor of 4 into 5 before applying it, for instance. This can be used to make the AI follow the strategy plan more strictly in some cases and less strictly in others.

Example[\[编辑\]](#) [\[编辑源代码\]](#)

AI templates[\[编辑\]](#) [\[编辑源代码\]](#)

/Hearts of Iron IV/common/ai_templates/*.txt files are used in order to define the templates that AI would aim for within the division template window.

A role template is defined as a block within any file in the folder with the name of the block being the same as the name of the role template.

These arguments are used for role templates themselves:

`roles = { ... }` is a list of roles that the templates would have. These can be anything, getting used for the [role_ratio AI strategy](#). AI will try to have one template for each role it has.

`available_for = { ... }` restricts the countries that use this role template to the tags in the list. If unspecified, every country uses them.

`blocked_for = { ... }` restricts the countries that use this role template from being the tags in the list. This is only needed if there is no `available_for = { ... }` block.

`match_to_count = 0.3` is a decimal on the scale of 0-1 that decides how much a division template should fulfill one of the templates within the role template in order to count as one. By default set to 0.5.

`upgrade_prio = { ... }` is a [MTTH block](#) that decides the 'importance' of the role template compared to other role templates for spending experience on improving. If there are several role templates with the same roles, the one with the highest priority gets used, otherwise this decides likelihood compared to role templates with different roles.

Additionally, within role templates each individual template is defined as a block with the name of the template being the name of the block. It may be anything, as long as there is no overlap. These arguments are used within a template definition:

`upgrade_prio = { ... }` is a [MTTH block](#) that decides the 'importance' of the template compared to other templates within the role template for spending experience on improving.

`production_prio = { ... }` is a [MTTH block](#) that decides the 'importance' of the template compared to other templates within the role template for actually producing.

`replace_with = my_other_template` assigns a different template that serves as a more modern version of this one. When it becomes possible to create that template, AI will try to move towards using that one.

`can_upgrade_in_field = { ... }` is a trigger block that decides when AI will try to upgrade divisions that are already assigned this template to the one that it's set to be replaced with.

`custom_icon = 10` assigns the icon that AI would assign to this template. This example would assign the template icon using the sprites `GFX_div_templ_10_large` and `GFX_div_templ_10_small`. Optional, defaults to the majority brigade within the template.

`reinforce_prio = 2` assigns the reinforce priority that AI would set on this template. The default priority is 1, appearing in-game as 'regular'. 0 is 'reserve', and 2 is 'elite'. Defaults to 1 if unset.

`target_width = 20` assigns the combat width that the AI aims for on this template.

`width_weight = 2.5` assigns how *much* the AI should focus on aiming towards the target width. The higher the weight, the more AI would avoid deviating from the target width.

`column_swap_factor = 0.3` assigns a likelihood for AI to swap entire columns within the template to a different subunit group to meet the template.

`stat_weights = { ... }` is a list of decimal values. Each one applies to different stats and decides what exactly AI should try to prioritise and what it should try to avoid. As within `NDefines.NAI.DIVISION_DESIGN_WEIGHTS` in [Defines](#), these stats are first the 17 army values, then `air_attack`, then common values and special values. In other words, Air and Navy values get skipped aside from `air_attack`, which is used by anti-air equipment. The value would be added to the default in defines: a 0.00 would mean it's the same priority as default.

`allowed_types = { ... }` is a list of sub-units that the AI can add to the template. If one is omitted, then the AI would never add it. This can be used to make AI not put units of different varieties in the template, such as putting infantry in a mobile light tank division, slowing it down.

`target_template = { ... }` assigns the template that AI should aim for. In particular, these arguments go inside of it:

`weight = 0.8` is how much the AI should aim towards having that template.

`match_value = 5000` is a value that decides how much the template is worth to AI if it's matched.

`support = { ... }` is a list of support battalions within the target template. A single definition within looks like `artillery = 1`. The first decides the unit type, the second decides the amount. Since there is a limit of only one of the same support battalion being allowed, the number can't be anything other than 1.

`regiments = { ... }` is a list of non-support, regular battalions within the target template. A single definition within looks like `artillery_brigade = 4`. The first decides the unit type, the second decides the amount.

Each template assigned to a role will form a group for the role, and then the fitness score of each available template is used to determine which is used at a specific moment for a specific role.

Example[\[编辑\]](#) [\[编辑源代码\]](#)

AI equipment[\[编辑\]](#) [\[编辑源代码\]](#)

/Hearts of Iron IV/common/ai_equipment/*.txt files are used in order to define the equipment variants that the AI should aim for when assigning modules to tank or ship variants.

A role template is defined as a block within any file in the folder with the name of the block being the same as the name of the role template.

These arguments are used for role templates themselves:

`category = <land|naval>` decides whether the template is used for tanks or ships respectively.

`roles = { ... }` is a list of roles that the templates would have. These can be anything, getting used for the [role_ratio AI strategy](#). AI will try to have one variant for each role it has.

`available_for = { ... }` restricts the countries that use this role template to the tags in the list. If unspecified, every country uses them.

`blocked_for = { ... }` restricts the countries that use this role template from being the tags in the list. This is only needed if there is no `available_for = { ... }` block.

`priority = { ... }` is a [MTTH block](#) that decides the 'importance' of the role template compared to other role templates for spending experience on improving. If there are several role templates with the same roles, the one with the highest priority gets used, otherwise this decides likelihood compared to role templates with different roles.

Additionally, within role templates each individual design is defined as a block with the name of the design being the name of the block. It may be anything, as long as there is no overlap. These arguments are used within a design definition:

`role_icon_index = 2` is used to assign a specific role icon to a *ship*. This does not work within land equipment. The icons are defined as a part of `naval_equipment_role = { ... }` within `/Hearts of Iron IV/gfx/army_icons/army_icons.txt`.

`priority = { ... }` is a [MTTH block](#) that decides the 'importance' of the design compared to other templates within the role template for spending experience on improving.

`enable = { ... }` is a trigger block that decides when AI should aim towards the design.

`allowed_types = { ... }` is a list of sub-units that the AI can add to the design. If one is omitted, then the AI would never add it. This can be used to make AI not put units of different varieties in the design, such as putting infantry in a mobile light tank division, slowing it down.

`target_variant = { ... }` assigns the variant that AI should aim for. In particular, these arguments go inside of it:

`match_value = 5000` is a value that decides how much the template is worth to AI if it's matched.

`type = light_tank_chassis_0` is the specific equipment type that must be used by the design.

`modules = { ... }` is a list of modules that the equipment should have, in particular:

`main_armament_slot = tank_flamethrower` decides requirements for a specified module slot, in first place. The requirement may be a module category, a specific module, or the word `empty`. If specifying a module, the greater-than or lesser-than signs can be used in order to require greater or lesser modules: this is decided by the year defined within the module. If lesser is specified, AI aims towards oldest modules, if greater is specified, AI aims towards newest modules. If specifying empty, greater than can be used to ensure it's *not* empty.

`main_armament_slot = { ... }` allows specifying more details for a module slot:

`module = tank_flamethrower` decides a module requirement for the slot. This is the exact same formatting as previous `main_armament_slot = tank_flamethrower`: module categories, modules, or empty are allowed, and the equality signs can be used in the same way.

`any_of = { ... }` is a list of modules or module categories. The module slot must have at least one of them.

`upgrade = current` ensures that, when upgrading a variant to match this design, it must use the same one as on the existing equipment. If the 'greater than' sign is used (`>`), then it would require AI to upgrade this slot as well.

`upgrades = { ... }` is a list of upgrades that the design should have, in particular:

`tank_nsb_engine_upgrade = 3` decides the AI priority that the specified upgrade should have to a fixed number.

`tank_nsb_engine_upgrade = { ... }` is a [MTTH block](#) that assigns the AI priority to the specified upgrade dynamically.

`requirements = { ... }` is a list of modules that the AI *must* have. This follows the same formatting as a specified module slot within a target variant's modules block: `module =`

`tank_flamethrower`, `any_of = { ... }`, etc. However, this is not tied to a specific slot.

`allowed_modules = { ... }` is a list of modules that the AI can use after the requirements in the target variant are met. If a module isn't here, it'll never be picked. The modules specified first take priority over the later ones.

Example[\[编辑\]](#) [\[编辑源代码\]](#)