# Computer Player Strategy Builder Guide

## AI Expert Documentation

Age of Empires II®: The Conquerors Expansion

**Important: Age of Empires II: The Conquerors Expansion allows you to create your own custom campaigns, scenarios, and computer player scripts. You may share these custom campaigns, scenarios, and computer player scripts for the purposes of gameplay, but you may not sell or make other commercial uses of the custom campaigns, scenarios, and computer player scripts. Microsoft reserves all other rights to the editors and files.**

## COPYRIGHT NOTICE

**Table of Contents**

# Introduction

Age of Empires II: The Conquerors Expansion uses an all-new Artificial Intelligence (AI) Expert System to act as the intelligence behind the computer players. This expert system uses a series of Rules that are tested to cause various actions to take place. In the following sections, you will learn how to make new rules for the AI to follow, how to check game facts to trigger those rules, and how to command the AI to take action based on your instructions.

AI Files are text files that have their extension changed to '.per' (for example, 'Henry Tudor.per'). These files contain the script commands that are used to create a new customized computer player. Use a text editor (one that displays line numbers is very helpful) to create your AI scripts, and copy them into the directory where you installed Age of Empires II, in the AI folder.
An empty text file with an .AI extension (for example, 'Marko.ai') should be created to make an entry appear in the list of computer players and in the scenario editor. The game will try to find a file with a matching .per extension that will be started when you pick that player. In your game's AI directory, you would then have 2 files:

C:\Program Files\Microsoft Games\Age of Empires II\AI\MyAI.ai (This is an empty file that makes an entry in the game's list)

C:\Program Files\Microsoft Games\Age of Empires II\AI\MyAI.per (This file contains your custom AI script of rules and facts)

*You must have both a .PER and an .AI file with the same name for your script to function!*

## *Rules*

Rules are the basis for the Expert System. There is a list of things we know about the game world, the other players, and so on. These are called *facts*. We check the *facts* with *rules* until a set of conditions exists that we need the computer player to act upon. *Actions* are what we call those commands that cause things to happen in the game. Examples might be training a unit, researching a technology, or sending a chat message.

### Defining Rules

Rules are defined in the script with the defrule instruction. If the conditions for the rule are met (True), the instructions in that rule are followed. If the conditions for the rule are not met (False), the rule is passed by.

A defrule example:

```
(defrule
      (cheats-enabled)
       =>
      (some action takes place)
)
```

Note that the parentheses around the rule are required – though the white-space formatting (spaces, tabs, etc.) is not important.

Rules continue to be evaluated until they are disabled. This is done with the disable-self command

```
(defrule
      (true)
       =>
```

```
            (disable-self)
      )
```

Going through the entire list of rules checking each one is what we call a rules pass. This system is very efficient, the rules may be checked as often as several times a second.

## Comment Lines

You will see comments throughout the AI Expert System script files. These comments start with a semicolon (;). For example:

```
      ;This line is a comment

      (defrule
            (food-amount greater-than 75)
             =>
            (train villager)   ;comments at end of line too!
      )
```

.

Any text that follows a semicolon on that line is a comment and is ignored by the AI script.

Once you define the rules, you can then use the all of the available facts and actions in combinations to do any action possible in the game.

## ***Facts***

Facts are those things that are tested in the rules. Player information such as how much gold you have, opponent information such as score, or game information such as time or victory conditions are some examples.

### Using Facts

Facts are used to enable rules. For example, this will train villagers whenever we have 50 food:

```
(defrule
      (food-amount greater-than 50)
       =>
      (train villager)
)
```

`See: Rules`

### Testing Facts

You will see <rel-op> associated with a lot of facts, this is a relative operator and allows you to test the relationship of the fact to another value, an example might be if you wanted the rule to be enabled when a certain amount of wood had been collected:

```
(defrule
      ( wood-amount greater-than 1000)
       =>
      ( chat-to-all “I have 1000 wood!”)
)
```

For more detailed information about parameters, see the "Parameters" section later in this document.

### Fact Parameters

Some facts can be tested just by checking them directly. If you want to see if cheats are enabled in the game, you can make a rule like this:

```
(defrule
      (cheats-enabled)
       =>
      (chat-to-all ”Let the cheating begin!” )
)
```

Other facts are more complex and require you to supply additional data that is used by the fact – information like the civilization <civ> or the map size <map-size> is required. Any fact that lists parameters <example> requires those parameters in order to work correctly.

## ***Actions***

Actions are those things you want the AI to do when it executes your rules. Actions can cause the AI player to build a building, train a unit, or send a chat message to a player, for example. Rules enable your computer player to take any action a human player could.

## Defrule Command

This command creates a new rule.

Example:

```
(defrule
      (game-time greater-than 30)
       =>
      (resign)
 )
```

## Defconst Command

This command creates a user-defined constant. For more information on defconst, see the "Conditional Loading and User-Defined Constants" section later in this document.

Syntax:

```
(defconst <constant-name> <value>)
```

`<constant-name>` is a user selected name. Use of the same format that the rest of the system uses is recommended but not required (for example, words-separated-with-dashes).

`<value>` is a valid integer value that will fit in a C++ type **short**. (For non-programmers, this means that the value can not be less than -32768 or greater than 32767.)

The following example defines a decided number of villagers in the Dark Age:

```
(defconst num-dark-age-villagers 22)
```

The following rule then uses it:

```
(defrule
        (civilian-population < num-dark-age-villagers)
        (can-train villager)
=>
        (train villager)
)
```

Constants are very handy for naming of goals, goal values, timers, taunts, etc.
Without constants all of these would be just nameless numbers.

Tip: If you group all of your defconsts together in one file, it makes it easy to customize your AI by changing the number that the defconst represents without having to change it everywhere in your file. In the example above, if you referred to num-dark-age-villagers in many places in your AI, you could easily change the defconst to be 12 villagers by changing it in just one place.

## Load Command

The Load command allows you to supply a filename of another script file within your main script file. This makes it easier to organize and re-use parts of your scripts in new ways.

Script language supports loading of script files from script files. Loaded files are in every aspect the same as original script files, so any script file can be loaded by any other script file.

Syntax:

```
(load "filename")
```

Load command can be inserted anywhere between the rules. For example:

```
(defrule ..........................)
(load "Dark Age Economy")
(defrule ..........................)
```

Notice that the filename does not have path or an extension. The script interpreter automatically adds a path and an extension. A script file being loaded should be placed in the same directory as the file that is loading it.

It is important to mention that the load command executes immediately. This means that when a load command is encountered, parsing of the current file is suspended until the load command finishes. At that point parsing resumes, starting with a rule immediately following the load command.

Load commands can be nested (a script that loads another script) up to 10 levels deep.

Loading multiple script files from a top-level script file makes computer players' knowledge modular. This approach has a benefit only if the script files loaded do not have overlapping areas of expertise.

## Load Random Command

A variation of the load command that allows for random loading of files. This command provides an option of randomizing AI strategies on the level higher than the rule level.

Syntax:

```
(load-random    20 "filename1"
                10 "filename2"
                40 "filename3"
                "filename4")
```

In the example above, "filename1" has a 20% chance of being loaded, "filename2" has a 10% chance, "filename3" has a 40% chance, and "filename4" is loaded if the first three files are not. "filename4" is called the default file.
Since all files share the same random number, one load-random command can load at most one file. Also, the sum of probabilities should never exceed 100.

Special Case 1:

```
(load-random
   20 "filename1"
   10 "filename2"
   40 "filename3"
)
```

No default file given. This is a valid syntax. There is a 30% chance that no file will be loaded.

Special Case 2:

(load-random "filename")

Only the default file is given. This is a valid syntax. The file is always loaded. This command is a slower version of the load command so its use is not recommended.

## Conditional Loading

Conditional loading allows you to load only rules that fit particular game settings. The system is somewhat similar to the C/C++ preprocessor. The main differences are:

a) The conditional loading mechanism works in the same pass as the parser that loads rules; therefore it is not a preprocessor.
b) The conditional loading mechanism makes a decision on whether a rule is loaded or not. It does not make a decision on whether a rule is parsed.

The latter is a design decision aimed to make syntax checking as painless as possible: single loading of a script in any game setting will check all rules for syntax errors.

The system automatically provides a set of symbols that reflect the game settings. These symbols can be checked to make decisions on which rules should be loaded.

Conditional loading provides three major benefits:

a) The ability to integrate rules for a wide variety of settings into a single AI personality.
b) Rules that are loaded are faster. For example if rules that work only with water maps are loaded, it is not necessary for those rules to keep checking the map type.
c) Rules that do not apply to given settings are not loaded, thus saving memory space.

Conditional loading recognizes four directives:

```
#load-if-defined  <system-defined-symbol>
#load-if-not-defined <system-defined-symbol>
#else,
#end-if
```

Together they are used to form the following constructs:

Construct #1:

```
#load-if-defined <system-defined-symbol>

...... define rules here

#end-if
```

Construct #2:

```
#load-if-not-defined <system-defined-symbol>

...... define rules here

#end-if
```

Construct #3:

#load-if-defined <system-defined-symbol>

...... define rules here

#else

...... define rules here

#end-if

Construct #4:

#load-if-not-defined <system-defined-symbol>

...... define rules here

#else

...... define rules here

#end-if

The following example loads only one rule based on the game difficulty setting:

```
#load-if-defined DIFFICULTY-EASIEST
(defrule
   (true)
   =>
   (chat-to-all "Easiest")
   (disable-self)
)
      #end-if
```

## *Technical Considerations*

Conditional loading directives can be nested 50 levels deep.

## *System-defined symbols*

There are two types of system-defined symbols:

1) Symbols that provide information on a game setting chosen from a drop-down list. In this case one symbol from a group of symbols will always be defined. A good example is map size. There will always be one map size chosen from a predefined set of sizes.

2) Symbols that provide information on a game setting chosen by selecting a check box. In this case a symbol is defined if a game setting is checked. Otherwise no symbol is defined. A good example is the Reveal Map setting.

Every system-defined symbol can have one of two possible scopes: global or local. Global symbols are shared by all computer players while local symbols are player specific. A good example of a global symbol would be DEATH-MATCH. If the game is set to be a Death Match game this is true for all computer players in the game. An example of local symbol would be JAPANESE-CIV.

System symbols available:

**Game Type**
(global, type 2)

DEATH-MATCH
REGICIDE
KING
WONDER RACE
DEFEND THE WONDER

**Starting Age**
(global, type 1)

DARK-AGE-START
FEUDAL-AGE-START
CASTLE-AGE-START
IMPERIAL-AGE-START
POST-IMPERIAL-AGE-START

**Starting Resources**
(global, type 1)

LOW-RESOURCES-START
MEDIUM-RESOURCES-START
HIGH-RESOURCES-START

**Map Size**
(global, type 1)

TINY-MAP
SMALL-MAP
MEDIUM-MAP
NORMAL-MAP
LARGE-MAP
GIANT-MAP

**Map Type**
(global, type 1)

ARABIA-MAP
ARCHIPELAGO-MAP
ARENA-MAP
BALTIC-MAP
BLACK-FOREST-MAP
COASTAL-MAP
CONTINENTAL-MAP
CRATER-LAKE-MAP
FORTRESS-MAP
GHOST-LAKE-MAP
GOLD-RUSH-MAP
HIGHLAND-MAP
ISLANDS-MAP
MEDITERRANEAN-MAP
MIGRATION-MAP

MONGOLIA-MAP
NOMAD-MAP
OASIS-MAP
REAL-WORLD [NAME] -MAP
RIVERS-MAP
SALT-MARSH-MAP
SCANDINAVIA-MAP
TEAM-ISLANDS-MAP
SCENARIO-MAP
YUCATAN-MAP

**Victory Type**
(global, type 1)

VICTORY-STANDARD
VICTORY-CONQUEST
VICTORY-TIME-LIMIT
VICTORY-SCORE
VICTORY-CUSTOM

**Difficulty**
(global, type 1)

DIFFICULTY-EASIEST
DIFFICULTY-EASY
DIFFICULTY-MODERATE
DIFFICULTY-HARD
DIFFICULTY-HARDEST

**Population Cap**
(global, type 1)

POPULATION-CAP-25
POPULATION-CAP-50
POPULATION-CAP-75
POPULATION-CAP-100
POPULATION-CAP-125
POPULATION-CAP-150
POPULATION-CAP-175
POPULATION-CAP-200

**Game Speed Lock**
(global, type 2)

GAME-SPEED-LOCKED

**Team Lock**
(global, type 2)

TEAMS-LOCKED

**Player's Civ**
(local, type 1)

GAIA

```
  AZTEC-CIV
  BRITON-CIV
  BYZANTINE-CIV
  CELTIC-CIV
  CHINESE-CIV
  FRANKISH-CIV
  GOTHIC-CIV
  HUN-CIV
  JAPANESE-CIV
  KOREAN-CIV
  MAYAN-CIV
  MONGOL-CIV
  PERSIAN-CIV
  SARACEN-CIV
  SPANISH-CIV
  TEUTONIC-CIV
  TURKISH-CIV
  VIKING-CIV
```

### *Examples*

```
#load-if-defined BRITON-CIV
(defrule
  (true)
  =>
  (chat-to-all "I am Briton")
  (disable-self)
)
#end-if

#load-if-defined TURKISH-CIV
(defrule
  (true)
  =>
  (chat-to-all "I am Turkish")
  (disable-self)
)
#end-if

#load-if-defined JAPANESE-CIV
(defrule
  (true)
  =>
  (chat-to-all "I am Japanese")
  (disable-self)
)
#end-if
```

### *Conditional Loading and User-Defined Constants*

A combination of conditional loading and user-defined constants can be very powerful. One of the common uses is scaling of parameters. Depending on the conditions, parameters with the same names are given different values. This technique reduces the number of needed rules and makes code more readable.

Example:

```
#load-if-defined DEATH-MATCH
  (defconst dark-age-villagers 6)
#else
  (defconst dark-age-villagers 22)
#end-if
```

In Death Match games the code sets the dark-age-villagers constant to 6. In other games the code sets it to 22.

Note that constants can not be redefined. The code

```
(defconst my-constant 1)
(defconst my-constant 2)
```

will cause the following error

```
ERR2012: Constant Already Defined: my-constant
```

The error will not appear if the constant is defined more than once with the same value:

```
(defconst my-constant 1)
(defconst my-constant 1)
```

## System Defined Constants

For every computer player the system defines a set of constants that make rule-writing easier and more efficient. Here they are:

**my-player-number**
(type <player-number>)

**my-civ**
(type <civ>)

**my-unique-unit**
(type <unit>)

**my-unique-unit-upgrade**
(type <research-item>)

**my-elite-unique-unit**
(type <unit>)

**my-unique-unit-line**
(type <unit>)

**my-unique-research**
(type <research-item>)

## Facts

### *Fact List*

## *Constant Facts*

**true**
**false**

## *Event Detection Facts*

**event-detected**
**taunt-detected**
**timer-triggered**

## *Game Facts*

**cheats-enabled**
**death-match-game**
**difficulty**
**game-time**
**map-size**
**map-type**
**player-computer**
**player-human**
**player-in-game**
**player-resigned**
**player-valid**
**population-cap**
**regicide-game**
**starting-age**
**starting-resources**
**victory-condition**

## *Commodity Trade Facts*

**can-buy-commodity**

**can-sell-commodity**
**commodity-buying-price**
**commodity-selling-price**

### *Tribute Detection Facts*

**players-tribute**
**players-tribute-memory**

### *Escrow Facts*

**can-build-gate-with-escrow**
**can-build-wall-with-escrow**
**can-build-with-escrow**
**can-research-with-escrow**
**can-spy-with-escrow**
**can-train-with-escrow**
**escrow-amount**

### *Computer Player Object Count Facts*

**attack-soldier-count**
**attack-boat-count**
**building-count**
**building-count-total**
**building-type-count**
**building-type-count-total**
**civilian-population**
**defend-soldier-count**
**defend-warboat-count**
**housing-headroom**
**idle-farm-count**
**military-population**
**population**
**population-headroom**
**soldier-count**
**unit-count**
**unit-count-total**
**unit-type-count**
**unit-type-count-total**
**warboat-count**

### *Computer Player Resource Facts*

**dropsite-min-distance**
**food-amount**
**gold-amount**
**resource-found**
**sheep-and-forage-too-far**
**stone-amount**
**wood-amount**

### *Regicide Facts*

**can-spy**

### *Computer Player Availability Facts*

**building-available**
**can-afford-building**
**can-afford-complete-wall**
**can-afford-research**
**can-afford-unit**
**can-build**
**can-build-gate**
**can-build-wall**
**can-research**
**can-train**
**research-available**
**research-completed**
**unit-available**
**wall-completed-percentage**
**wall-invisible-percentage**

### *Computer Player Miscellaneous Facts*

**civ-selected**
**current-age**
**current-age-time**
**current-score**
**doctrine**
**enemy-buildings-in-town**
**enemy-captured-relics**
**goal**
**player-number**
**random-number**
**shared-goal**
**stance-toward**
**strategic-number**
**town-under-attack**

### *Opponent Facts*

**players-building-count**
**players-building-type-count**
**players-civ**
**players-civilian-population**
**players-current-age**
**players-current-age-time**
**players-military-population**
**players-population**
**players-score**
**players-stance**
**players-unit-count**
**players-unit-type-count**

### *Cheating Facts*

**cc-players-building-count**
**cc-players-building-type-count**
**cc-players-unit-count**
**cc-players-unit-type-count**

## *Fact Details*

### true

This fact is always true. It is used for testing purposes.

### false

This fact is always false. It is used for testing purposes.

### attack-soldier-count <rel-op> <value>

The fact checks the computer player’s attack soldier count. An attack soldier is a land-based military unit currently assigned to attack groups.

### attack-warboat-count <rel-op> <value>

The fact checks the computer player’s attack warboat count. An attack warboat is a boat capable of attacking and currently assigned to attack groups.

### building-available <building>

The fact checks that the building is available to the computer player's civ, and that the tech tree prerequisites are met.
The fact does not check that there are enough resources to build the building.
The fact allows the use of building line wildcard parameters for the <building>.

### building-count <rel-op> <value>

This fact checks the computer player's building count. Only existing buildings are included.

### building-count-total <rel-op> <value>

This fact checks the computer player's total building count. The total includes existing and queued buildings.

### building-type-count <building> <rel-op> <value>

This fact checks the computer player's building count. Only existing buildings of the given type are included.
The fact allows the use of building line wildcard parameters for the <building>.

### building-type-count-total <building> <rel-op> <value>

This fact checks the computer player's total building count. The total includes existing and queued buildings of the given type.
The fact allows the use of building line wildcard parameters for the <building>.

### can-afford-building <building>

This fact checks whether the computer player has enough resources to build the given building.
The fact does not take into account escrowed resources.
The fact allows the use of building line wildcard parameters for the <building>.

### can-afford-complete-wall <perimeter> <wall-type>

This fact checks whether the computer player has enough resources to finish the given wall at the given perimeter. The fact does not take into account escrowed resources.
In particular it checks:

- That the wall type is available to the computer player’s civilization.

- That the tech tree prerequisites for using that wall type are met.
- That the resources needed for completing a wall line are available, not counting escrow amounts.

## can-afford-research <research-item>

This fact checks whether the computer player has enough resources to perform the given research. The fact does not take into account escrowed resources.

## can-afford-unit <unit>

This fact checks whether the computer player has enough resources to train given unit.
The fact does not take into account escrowed resources.
The fact allows the use of unit line wildcard parameters for the <unit>.

## can-build <building>

This fact checks whether the computer player can build the given building. In particular it checks:

- That the building is available to the computer player's civilization.
- That the tech tree prerequisites for building are met.
- That the resources needed for the building are available, not counting escrow amounts.

The fact allows the use of building line wildcard parameters for the <building>.

## can-build-gate <perimeter>

This fact checks whether construction of a gate as part of the given perimeter wall can start.
The fact does not take into account escrow resources.
In particular it checks:

- That the gate is available to the computer player's civilization.
- That the tech tree prerequisites for building a gate are met.
- That the resources needed for building a gate are available, not counting escrow amounts.
- That there is a location to build a gate.

## can-build-gate-with-escrow <perimeter>

This fact checks whether construction of a gate as part of the given perimeter wall can start.
The fact takes into account escrow resources.
In particular it checks:

- That the gate is available to the computer player's civilization.
- That the tech tree prerequisites for building a gate are met.
- That the resources needed for building a gate are available when using escrow amounts.
- That there is a location to build a gate.

## can-build-wall <perimeter> <wall-type>

This fact checks whether a wall line of the given wall type can be built at the given perimeter.
The fact does not take into account escrow resources.
In particular it checks:

- That the wall type is available to the computer player's civilization.
- That the tech tree prerequisites for using that wall type are met.
- That the resources needed for building a wall line are available, not counting escrow amounts.
- That there is a location to build a wall line.

The fact allows the use of wall line wildcard parameters for the <wall-type>.

## can-build-wall-with-escrow <perimeter> <wall-type>

This fact checks whether a wall line of the given wall type can be built at the given perimeter.
The fact takes into account escrow resources.
In particular it checks:

- That the wall type is available to the computer player’s civilization.
- That the tech tree prerequisites for using that wall type are met.
- That the resources needed for building a wall line are available when using escrow amounts.
- That there is a location to build a wall line.

The fact allows the use of wall line wildcard parameters for the <wall-type>.

## can-build-with-escrow <building>

This fact checks whether the computer player can build the given building. In particular it checks:

- That the building is available to the computer player’s civilization.
- That the tech tree prerequisites for building are met.
- That the resources needed for building are available when using escrow amounts.

The fact allows the use of building line wildcard parameters for the <building>.

## can-buy-commodity <commodity>

This fact checks whether the computer player can buy one lot of the given commodity.
The fact does not take into account escrowed resources.

## can-research <research-item>

This fact checks if the given research can start. In particular it checks:

- That the research item is available to the computer player's civilization.
- That the tech tree prerequisites for research are met.
- That resources needed for research are available, not counting escrow amounts.
- That there is a building that is not busy and is ready to start research.

## can-research-with-escrow <research-item>

This fact checks if the given research can start. In particular it checks:

- That the research item is available to the computer player's civilization.
- That the tech tree prerequisites for research are met.
- That resources needed for research are available when using escrow amounts.
- That there is a building that is not busy and is ready to start research.

## can-sell-commodity <commodity>

This fact checks whether the computer player can sell one lot of the given commodity.
The fact does not take into account escrowed resources.

## can-spy

This fact checks if the spy command can be executed.
The fact takes into account escrowed resources.

## can-spy-with-escrow

This fact checks if spy command can be executed.
The fact does not take into account escrowed resources.

## can-train <unit>

This fact checks if the training of the given unit can start. In particular it checks:

- That the unit is available to the computer player's civilization.
- That the tech tree prerequisites for training the unit are met.
- That resources needed for training the unit are available, not counting escrow amounts.
- That there is enough housing headroom for the unit.
- That there is a building that is not busy and is ready to start training the unit.

The fact allows the use of unit line wildcard parameters for the <unit>.

### can-train-with-escrow <unit>

This fact checks if the training of the given unit can start. In particular it checks:

- That the unit is available to the computer player's civilization.
- That the tech tree prerequisites for training the unit are met.
- That resources needed for training the unit are available when using escrow amounts.
- That there is enough housing headroom for the unit.
- That there is a building that is not busy and is ready to start training the unit.

The fact allows the use of unit line wildcard parameters for the <unit>.

### cc-players-building-count <player-number> <rel-op> <value>

A cheating version of players-building-count. For use in scenarios only.
The fact checks the given player's building count. Both existing buildings and buildings under construction are included regardless of whether they have been seen – fog is ignored.
The fact allows “any”/”every” wildcard parameters for the <player-number> and the use of building line wildcard parameters for the <building>.

### cc-players-building-type-count <player-number> <building> <rel-op> <value>

A cheating version of players-building-type-count. For use in scenarios only.
This fact checks the given player's building count. Both existing buildings and buildings under construction of the given type are included regardless of whether they have been seen – fog is ignored.
The fact allows “any”/”every” wildcard parameters for the <player-number> and the use of building line wildcard parameters for the <building>.

### cc-players-unit-count <player-number> <rel-op> <value>

A cheating version of players-unit-count. For use in scenarios only.
This fact checks the given player's unit count. Only trained units are included and fog is ignored.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### cc-players-unit-type-count <player-number> <unit> <rel-op> <value>

A cheating version of players-unit-type-count. For use in scenarios only.
This fact checks the given player's unit count. Only trained units of the given type are included and fog is ignored.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### cheats-enabled

This fact checks whether the cheats have been enabled.

### civ-selected <civ>

This fact checks the computer player’s civ.

### civilian-population <rel-op> <value>

This fact checks the computer player’s civilian population. The civilian population is villagers, trade vehicles, fishing boats, etc.

### commodity-buying-price <commodity> <rel-op> <value>

This fact checks the current buying price for the given commodity.

### commodity-selling-price <commodity> <rel-op> <value>

This fact checks the current selling price for the given commodity.

### current-age <rel-op> <age>

This fact checks computer player’s current age.

### current-age-time <rel-op> <value>

This fact checks the computer player’s current age time -- time spent in the current age.

### current-score <rel-op> <value>

This fact checks the computer player’s current score.

### death-match-game

This fact checks if the game is a Death Match game.

### defend-soldier-count <rel-op> <value>

This fact checks the computer player's defend soldier count.
A defend soldier is a land-based military unit not assigned to attack groups.

### defend-warboat-count <rel-op> <value>

This fact checks the computer player's defend warboat count.
A defend warboat is a boat capable of attacking and not assigned to attack groups.

### difficulty <rel-op> <difficulty>

This fact checks difficulty setting.

### doctrine <value>

This fact checks what the current doctrine is.

### dropsite-min-distance <resource-type> <rel-op> <value>

This fact checks computer player’s minimum dropsite walking distance for a given resource type.
Long walking distances indicate a need for a new dropsite.
It is not recommended to use this fact for building of first dropsites necessary for age advancement. If, at the beginning, the resources happen to be close enough to the Town Center, building of the first dropsites will be delayed, resulting in slower age progression.

### enemy-buildings-in-town

The fact checks for enemy buildings in a computer player's town.

### enemy-captured-relics

This fact checks if the enemy has captured all relics.
When this happens, tactical AI automatically starts targeting Monasteries and Monks. Use this fact to intensify attacks and combine it with the attack-now action to force attacks.

### escrow-amount <resource-type> <rel-op> <value>

This fact checks a computer player's escrow amount for a given resource type.

### event-detected <event-type> <event-id>

This fact checks if the given event has been detected. The fact stays true until the event is explicitly disabled by the acknowledge-event action.

### food-amount <rel-op> <value>

This fact checks a computer player's food amount.

### game-time <rel-op> <value>

This fact checks the game time. The game time is given in seconds. The fact can be used to make rules time-specific. For example, the computer can become more aggressive after 15 minutes of game time.

### goal <goal-id> <value>

This fact checks what the given goal is.

### gold-amount <rel-op> <value>

This fact checks a computer player's gold amount.

### housing-headroom <rel-op> <value>

This fact checks computer player's housing headroom. Housing headroom is the difference between current housing capacity and trained unit capacity. For example, a computer player has a Town Center (capacity 5), a House (capacity 5) and 6 villagers. In this case, housing headroom is 4.

### idle-farm-count <rel-op> <value>

This fact checks a computer player's idle farm count – the number of farms with no farmers. It should be used before a new farm is built to make sure it is needed.

### map-size <map-size>

This fact checks map size.

### map-type <map-type>

This fact checks map type.

### military-population <rel-op> <value>

This fact checks computer player's military population.

### player-computer <player-number>

This fact checks if the given player is a computer player.
The fact allows "any"/"every" wildcard parameters for the <player-number>.

### player-human <player-number>

This fact checks if the given player is a human.
The fact allows "any"/"every" wildcard parameters for the <player-number>.

### player-in-game <player-number>

This fact checks if the given player is a valid player and still playing.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### player-number <player-number>

This fact checks computer player’s player number.

### player-resigned <player-number>

This fact checks if the given player has lost by resigning. Note that a player can lose without resigning, so this fact should not be used to check whether a player has lost a game. To check whether a player has lost a game use (not (player-in-game <player-number)).
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### player-valid <player-number>

This fact checks if the given player is a valid player. In games with more than 2 players, players that lost before the game is over are considered to be valid players. This is because although the player is not in the game, their units/buildings can still be in the game. To check whether the given player is still in the game use the player-in-game fact.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-building-count <player-number> <rel-op> <value>

This fact checks the given player's building count. Both existing buildings and buildings under construction are included. The computer player relies only on what it has seen – no cheating.
The fact allows “any”/”every” wildcard parameters for the <player-number> and the use of building line wildcard parameters for the <building>.

### players-building-type-count <player-number> <building> <rel-op> <value>

This fact checks the given player's building count. Both existing buildings and buildings under construction of the given type are included. The computer player relies only on what it has seen – no cheating.
The fact allows “any”/”every” wildcard parameters for the <player-number> and the use of building line wildcard parameters for the <building>.

### players-civ <player-number> <civ>

This fact checks the given player’s civ.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-civilian-population <player-number> <rel-op> <value>

This fact checks a given player’s civilian population.
This is equivalent to a human player checking the timeline.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-current-age <player-number> <rel-op> <age>

This fact checks the given player’s current age.
This is equivalent to a human player checking the timeline.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-current-age-time <player-number> <rel-op> <value>

This fact checks the given player’s current age time -- time spent in the current age.
This is equivalent to a human player checking the timeline.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-military-population <player-number> <rel-op> <value>

This fact checks the given player’s military population.
This is equivalent to a human player checking the timeline.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-population <player-number> <rel-op> <value>

This fact checks the given player’s population.
This is equivalent to a human player checking the timeline.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-score <player-number> <rel-op> <score>

This fact checks the given player’s current score.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-stance <player-number> <diplomatic-stance>

This fact checks the given player’s diplomatic stance.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-tribute <player-number> <resource-type> <rel-op> <value>

This facts checks the player's tribute given throughout the game.
Only tribute for the given resource type is checked.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-tribute-memory <player-number> <resource-type> <rel-op> <value>

This facts checks a player's tribute given since the player's tribute memory was cleared.
Only tribute memory for the given resource type is checked.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-unit-count <player-number> <rel-op> <value>

This fact checks the given player's unit count. The computer player relies only on what it has seen – no cheating. For allies and self only trained units are included.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### players-unit-type-count <player-number> <unit> <rel-op> <value>

This fact checks the given player's unit count. The computer player relies only on what it has seen – no cheating. For allies and self only trained units of the given type are included.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### population <rel-op> <value>

This fact checks the computer player’s population.

### population-cap <rel-op> <value>

This fact checks population cap setting.

### population-headroom <rel-op> <value>

This fact checks the computer player’s population headroom. Population headroom is the difference between the game’s population cap and current housing capacity. For example, in a game with a population cap of 75, the computer player has a town center (capacity 5) and a house (capacity 5). In this case population headroom is 65.

### random-number  <rel-op>  <value>

This fact checks random number value.

### regicide-game

This fact checks if the game is a regicide game.

### research-available  <research-item>

The fact checks that the given research is available to the computer player's civ, and that the research is available at this time (tech tree prerequisites are met). The fact does not check that there are enough resources to start researching.

### research-completed  <research-item>

This fact checks that the given research is completed.

### resource-found  <resource-type>

This fact checks whether the computer player has found the given resource.
The facts should be used at the beginning of the game. Once it becomes true for a certain resource it stays true for that resource. In this context, food refers to a forage site and wood refers to forest trees (not lone trees).

### shared-goal <shared-goal-id> <value>

This fact checks a given shared goal -- a goal that is shared among computer players.
It is to be used only when all computer players are on the same team.

### sheep-and-forage-too-far

This fact checks whether the computer player has any forage site(s) and/or sheep within 8 tiles of the drop-off location (Mill or Town Center).

### soldier-count  <rel-op>  <value>

This fact checks the computer player’s soldier count. An attack soldier is a land-based military unit.

### stance-toward <player-number> <diplomatic-stance>

This fact checks the computer player’s stance toward a given player.
The fact allows “any”/”every” wildcard parameters for the <player-number>.

### starting-age <rel-op> <age>

This fact checks the game's starting age. In addition to the regular age parameters, post-imperial-age can be used.

### starting-resources <rel-op> <starting-resources>

This fact checks starting resources (low, medium, or high).

### stone-amount  <rel-op>  <value>

This fact checks the computer player’s stone amount.

### strategic-number  <strategic-number>  <rel-op>  <value>

This fact checks a strategic number’s value.

### taunt-detected <player-number> <taunt-id>

This fact detects a given taunt. The check can be performed any number of times until the taunt is explicitly acknowledged.
The fact allows “any”/”every” wildcard parameters for the <player-number>.
The following example detects a request for food by an enemy player, computer or human:

```
(defrule
   (taunt-detected any-enemy 3)
   =>
   (acknowledge-taunt this-any-enemy 3)
   (chat-to-player this-any-enemy "No food for you")
)
```

### timer-triggered <timer-id>

This fact checks whether a given timer has triggered. For disabled timers this fact is always false. The check can be performed any number of times until the timer is explicitly disabled.

### town-under-attack

This fact is set to true when a computer player’s town is under attack.

### unit-available <unit>

The fact checks that the unit is available to the computer player's civ, and that the tech tree prerequisites for training the unit are met.
The fact does not check whether the unit training can start. This depends on resource availability, housing headroom, and whether the building needed for training is currently used for research/training of another unit.
The fact allows the use of unit line wildcard parameters for the <unit>.

### unit-count <rel-op> <value>

This fact checks the computer player's unit count. Only trained units are included.

### unit-count-total <rel-op> <value>

This fact checks the computer player's total unit count. The total includes trained and queued units.

### unit-type-count <unit> <rel-op> <value>

This fact checks the computer player's unit count. Only trained units of the given type are included.
The fact allows the use of unit line wildcard parameters for the <unit>.

### unit-type-count-total <unit> <rel-op> <value>

This fact checks the computer player's total unit count. The total includes trained and queued units of the given type.
The fact allows the use of unit line wildcard parameters for the <unit>.

### victory-condition <victory-condition>

This fact checks the game victory condition.

### wall-completed-percentage <perimeter> <rel-op> <value>

This fact checks the completion percentage for a given perimeter wall. Trees and other destructible natural barriers are included and count as completed.

### wall-invisible-percentage <perimeter> <rel-op> <value>

This fact checks what percentage of the potential wall placement is covered with fog.

Example:

```
(defrule
 (wall-completed-percentage 1 < 100)     ; Not all of it finished
 (wall-invisible-percentage 1 == 0)      ; And we can see it all
      =>
 (chat-local "Found hole in the wall.")
)
```

Notice that if the invisible percentage is not equal to 0 we do not know if there is a hole or not. This is because the hidden tile(s) might have a tree(s).

### warboat-count <rel-op> <value>

The fact checks the computer player’s warboat count. A warboat is a boat capable of attacking.

### wood-amount <rel-op> <value>

This fact checks the computer player’s wood amount.

# Actions

## *Action List*

### *Input / Output Actions*

**chat-local**
**chat-local-using-id**
**chat-local-using-range**
**chat-local-to-self**
**chat-to-all**
**chat-to-all-using-id**
**chat-to-all-using-range**
**chat-to-allies**
**chat-to-allies-using-id**
**chat-to-allies-using-range**
**chat-to-enemies**
**chat-to-enemies-using-id**
**chat-to-enemies-using-range**
**chat-to-player**
**chat-to-player-using-id**
**chat-to-player-using-range**
**chat-trace**
**log**
**log-trace**
**taunt**
**taunt-using-range**

### *Rule Control Actions*

**disable-self**

### *Event Actions*

**acknowledge-event**

**acknowledge-taunt**
**disable-timer**
**enable-timer**
**set-signal**

### *Commodity Trade Actions*

**buy-commodity**
**sell-commodity**

### *Tribute Actions*

**clear-tribute-memory**
**tribute-to-player**

### *Escrow Actions*

**release-escrow**
**set-escrow-percentage**

### *Regicide Actions*

**spy**

### *Cheating Actions*

**cc-add-resource**

### *Other Actions*

**do-nothing**
**attack-now**
**build**
**build-forward**
**build-gate**
**build-wall**
**delete-building**
**delete-unit**
**enable-wall-placement**
**generate-random-number**
**research**
**resign**
**set-difficulty-parameter**
**set-doctrine**
**set-goal**
**set-shared-goal**
**set-stance**
**set-strategic-number**
**train**

### *Action Details*

#### do-nothing

This action does nothing at all. It is primarily used as a stub for testing purposes. Note that every rule needs at least one action.

#### acknowledge-event  <event-type> <event-id>

This action acknowledges a received event by resetting the associated flag.

## acknowledge-taunt <player-number> <taunt-id>

This action acknowledges the taunt (resets the flag). Like other event systems in the AI, taunt detection requests explicit acknowledgement.
The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.

## attack-now

This action forces attack with currently available attack units. Units are designated as attack units by using `sn-percent-attack-soldiers` or `sn-percent-attack-boats`.

### build <building>

This action builds the given building.
The action allows the use of building line wildcard parameters for the <building>.

### build-forward <building>

This action builds given building close to enemy.
The action allows the use of building line wildcard parameters for the <building>.

### build-gate <perimeter>

This action builds a gate as part of the given perimeter wall.

### build-wall <perimeter> <wall-type>

This action builds a wall line of the given wall type at the given perimeter.
The action allows the use of wall line wildcard parameters for the <wall-type>.

### buy-commodity <commodity>

This action buys one lot of the given commodity.

### cc-add-resource <resource-type> <amount>

This is a cheating action that adds the given resource amount to the computer player.
It is to be used in scenarios to avoid late game oddities such as computer player villagers going all over the map while looking for the last pile of gold.

### chat-local <string>

This action displays the given string as a local chat message.

### chat-local-using-id <string-id>

This action displays a string, defined by a string id, as a local chat message.
Ensemble Studios use only.

### chat-local-using-range <string-id-start> <string-id-range>

This action displays a random string as a local chat message. The random string is defined by a string id randomly picked out of a given string id range.
Ensemble Studios use only.

### chat-local-to-self <string>

This action displays a given string as local chat message. The message is displayed only if the user is the same player as the computer player sending the message. For debugging purposes only.

### chat-to-all <string>

This action sends a given string as a chat message to all players.

### chat-to-all-using-id <string-id>

This action sends a string, defined by a string id, as a chat message to all players.
Ensemble Studios use only.

### chat-to-all-using-range <string-id-start> <string-id-range>

This action sends a random string as chat message to all players. The random string is defined by a string id randomly picked out of a given string id range.

Ensemble Studios use only.

Example:
(chat-to-all-using-range 5020 5)

will send a random localized message with a string id between 5020 and 5024.

### chat-to-allies <string>

This action sends a given string as a chat message to allies.

### chat-to-allies-using-id <string-id>

This action sends a string, defined by a string id, as a chat message to allied players.
Ensemble Studios use only.

### chat-to-allies-using-range <string-id-start> <string-id-range>

This action sends a random string as a chat message to allied players. The random string is defined by a string id randomly picked out of a given string id range.
Ensemble Studios use only.

### chat-to-enemies <string>

This action sends a given string as a chat message to enemies and neutral players.

### chat-to-enemies-using-id <string-id>

This action sends a string, defined by a string id, as a chat message to enemies and neutral players.
Ensemble Studios use only.

### chat-to-enemies-using-range <string-id-start> <string-id-range>

This action sends a random string as a chat message to enemies and neutral players. The random string is defined by a string id randomly picked out of a given string id range.
Ensemble Studios use only.

### chat-to-player <player-number> <string>

This action sends a given string as a chat message to a given player.
The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.

### chat-to-player-using-id <player-number> <string-id>

This action sends a string, defined by a string id, as a chat message to a given player.
The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.
Ensemble Studios use only.

### chat-to-player-using-range <player-number> <string-id-start> <string-id-range>

This action sends a random string as a chat message to enemies and neutral players. The random string is defined by a string id randomly picked out of a given string id range.
The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.
Ensemble Studios use only.

### chat-trace <value>

This action displays the given value as a chat message. Used purely for testing to check when a rule gets executed.

### clear-tribute-memory <player-number> <resource-type>

This action clears the given player's tribute memory. Only tribute memory for the given resource type is cleared.
The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.

### delete-building <building>

This action deletes exactly one building of a given type.

### delete-unit <unit>

This action deletes exactly one unit of a given type.

### disable-self

This action disables the rule that it is part of. Since disabling takes effect in the next execution pass, other actions in the same rule are still executed one last time.
Example:

```
(defrule
  (game-time greater-than 30)
   =>
  (disable-self)
)
```

## disable-timer <timer-id>

This action disables the given timer.

## enable-timer <timer-id>

This action enables the given timer and sets it to the given time interval.

## enable-wall-placement <perimeter>

This action enables wall placement for the given perimeter. Enabled wall placement causes the rest of the placement code to do some planning and place all structures at least one tile away from the future wall lines.

If you are planning to build a wall, you have to explicitly define which perimeter wall you plan to use when the game starts. This is a one-time action and should be used during the initial setup.

Example:

```
(defrule
  (enable-wall-placement 2)
   =>
  (disable-self)
)
```

## generate-random-number <value>

This action generates a player-specific integer random number within given range (1 to <value>). The number is stored and its value can be tested. Subsequent executions of this action generate new random numbers that replace existing ones. Example:

```
; For readability reasons define the constants

(defconst feudal-age-rush 1)
(defconst castle-age-rush 2)

; First roll the dice. This will generate number between 1 and 100 inclusive

(defrule
  (true)
  =>
  (generate-random-number 100)
  (disable-self)
)

; Based on the outcome we pick the strategy:
; 20% chance of feudal age rush
; 80% chance of castle age rush

(defrule
  (random-number > 80)
  =>
  (set-goal 1 feudal-age-rush)
  (disable-self)
)

(defrule
  (random-number < 81)
  =>
  (set-goal 1 castle-age-rush)
  (disable-self)
)
```

## log <string>

This action writes the given string to a log file. Used purely for testing purposes. Works only if logging is enabled.

## log-trace <value>

This action writes the given value to a log file. Used purely for testing to check when a rule gets executed. Works only if logging is enabled.

## release-escrow <resource-type>

This action releases the computer player's escrow for a given resource type.

## research <research-item>

This action researches the given item. To prevent cheating, this action uses the same criteria as the **can-research** fact to make sure the item can be researched.

### research <age>

This action researches the next age.

### resign

This action causes the computer player to resign.

```
(defrule
  (game-time equal 6000)
   =>
  (resign)
)
```

### sell-commodity <commodity>

This action sells one lot of a given commodity.

### set-difficulty-parameter <difficulty-parameter> <value>

This action sets a given difficulty parameter to a given value.

### set-doctrine <value>

This action sets the doctrine to the given value.

### set-escrow-percentage <resource-type> <value>

This action sets the computer player's escrow percentage for a given resource type.
Given values have to be in the range 0-100.

### set-goal <goal-id> <value>

This action sets a given goal to a given value.

### set-shared-goal <shared-goal-id> <value>

This action sets a given shared goal (a goal that is shared among computer players) to a given value.
To be used only when all computer players are on the same team.

### set-signal <signal-id>

This action sets a given signal that can be checked by the trigger system.

### set-stance <player-number> <diplomatic-stance>

This action sets the stance toward a given player.
The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.

### set-strategic-number <strategic-number> <value>

This action sets a given strategic number to a given value.

See Appendix A for information on strategic numbers.

### spy

This action executes a spy command.

### taunt <value>

This action triggers the taunt associated with the given value.

```
(defrule
    (game-time equal 6000)
     =>
    (taunt 10)
    (disable-self)
  )
```

## taunt-using-range <taunt-start> <taunt-range>

This action triggers a random taunt that is picked from a given taunt range.

Example:
(taunt-using-range 50 10)

will use a random taunt between 50 and 59.

## train <unit>

This action trains the given unit. To prevent cheating, this action uses the same criteria as the **can-train** fact to make sure the unit can be trained.
The fact allows the use of unit line wildcard parameters for the <unit>.

```
  (defrule
    (food-amount greater-than 100)
     =>
    (train villager)
    (train swordsman)
  )
```

## tribute-to-player <player-number> <resource-type> <value>

This action tributes the given amount of the given resource type to the player defined by the player-number parameter. Implementation specifics:

- If the computer player does not have a Market no tribute is given.
- In the case when the value parameter specifies an amount larger than available, only the available resources of the given type are tributed. If, for example, there is only 60 food and the tribute action specifies 100 food, only 60 food will be tributed.
- The tribute action is ignored when there are no resources of the given type.
- Tribute fees are paid and deducted from the tribute amount (if applicable).

The action allows “any”/”every” wildcard parameters for the <player-number>.
It also allows the use of rule variables for the <player-number>.

# Parameters

## *Parameter List*

## *Parameter Details*

### <age>

is one of the following:

dark-age feudal-age
castle-age
imperial-age

starting age facts also use:

post-imperial-age

### <building>

is one of the following:

archery-range
barracks
blacksmith
bombard-tower
castle
dock
farm
fish-trap
guard-tower
house
keep
lumber-camp
market
mill
mining-camp
monastery
outpost
siege-workshop
stable
town-center
university
watch-tower
wonder

**For a description of the wildcard parameters accepted as <building> parameters, see the "Wildcard Parameters" section later in this document.**

### <civ>

is one of the following:

briton
byzantine
celtic

chinese
frankish
gothic
japanese
mongol
persian
saracen
teutonic
turkish
viking

## <commodity>

is one of the following:

food
stone
wood

## <difficulty>

is one of the following:

easiest
easy
moderate
hard
hardest

**The ordering of difficulty settings is the opposite of what one would expect!**
**Make sure that this is taken in account when using facts to compare difficulties.**

**easiest > easy > moderate > hard > hardest**

## <difficulty-parameter>

is one of the following:

ability-to-dodge-missiles
ability-to-maintain-distance

Description:

ability-to-dodge-missiles
Chance of a computer player’s unit dodging a missile. Valid range 0-100. Default value is 100.

ability-to-maintain-distance
Chance that a computer player’s ranged unit will maintain the distance. Valid range 0-100. Default value is 100.

## <diplomatic-stance>

is one of the following:

ally
neutral
enemy

## <event-id>

is a valid event id. Event ids have a range that depends on the event type.
For trigger events the id is in the range from 0 to 255.

## <event-type>

is a one of the following:

trigger

## <goal-id>

is a valid goal id. Goal ids have a range from 1 to 40.

## <map-size>

is one of the following:

tiny
small
medium
normal
large
giant

## <map-type>

is one of the following:

arabia
archipelago
baltic
black-forest
coastal
continental
crater-lake
fortress
gold-rush
highland
islands
mediterranean
migration
rivers
team-islands
scenario-map

## <perimeter>

is a valid wall perimeter. Allowed values are 1 and 2, with 1 being closer to the Town Center than 2.

Perimeter 1 is usually between 10 and 20 tiles from the starting Town Center.
Perimeter 2 is usually between 18 and 30 tiles from the starting Town Center.

## <player-number>

is a valid player number or one of the wildcard parameters (if explicitly allowed by the fact/action):

any-ally
any-computer
any-computer-ally
any-computer-enemy
any-computer-neutral
any-enemy
any-human
any-human-ally
any-human-enemy

any-human-neutral
any-neutral

every-ally
every-computer
every-enemy
every-human
every-neutral

**For a detailed description of wildcard parameters, see the "Wildcard Parameters" section later in this document.**

**Note: Wildcard parameters applying to allies do not apply to self.**

## <rel-op>

is one of the following (two versions, either can be used):

**full:**

less-than
less-or-equal
greater-than
greater-or-equal
equal
not-equal

**short:**

<
<=
>
>=
==
!=

## <research-item >

is one of the following (grouped by building):
**Note: You must include ri before the item for research items.**

**archery range research items:**

ri-arbalest
ri-crossbow
ri-elite-skirmisher
ri-hand-cannon
ri-heavy-cavalry-archer

**barracks research items:**

ri-champion
ri-elite-eagle-warrior
ri-halberdier
ri-long-swordsman
ri-man-at-arms
ri-parthian-tactics
ri-pikeman
ri-squires
ri-thumb-ring
ri-tracking
ri-two-handed-swordsman

**blacksmith research items:**

ri-blast-furnace
ri-bodkin-arrow
ri-bracer
ri-chain-barding
ri-chain-mail
ri-fletching
ri-forging
ri-iron-casting
ri-leather-archer-armor
ri-padded-archer-armor
ri-plate-barding
ri-plate-mail
ri-ring-archer-armor
ri-scale-barding
ri-scale-mail

**castle research items:**

ri-conscription
ri-hoardings
ri-sappers
ri-elite-berserk
ri-elite-cataphract
ri-elite-chu-ko-nu
ri-elite-huskarl
ri-elite-janissary
ri-elite-longbowman
ri-elite-mameluke
ri-elite-mangudai
ri-elite-samurai
ri-elite-teutonic-knight
ri-elite-throwing-axeman
ri-elite-war-elephant
ri-elite-woad-raider
ri-my-unique-elite-unit
ri-my-unique-research

**dock research items:**

ri-cannon-galleon
ri-careening
ri-deck-guns
ri-dry-dock
ri-elite-longboat
ri-fast-fire-ship
ri-galleon
ri-heavy-demolition-ship
ri-shipwright
ri-war-galley

**lumber camp research items:**

ri-bow-saw
ri-double-bit-axe
ri-two-man-saw

**market research items:**

ri-banking
ri-caravan
ri-cartography
ri-coinage
ri-guilds

**mill research items:**

ri-crop-rotation
ri-heavy-plow
ri-horse-collar

**mining camp research items:**

ri-gold-mining
ri-gold-shaft-mining
ri-stone-mining
ri-stone-shaft-mining

**monastery research items:**

ri-atonement
ri-block-printing
ri-faith
ri-fervor
ri-herbal-medicine
ri-heresy
ri-illumination
ri-redemption
ri-sanctity
ri-theocracy

**siege workshop research items:**

ri-bombard-cannon
ri-capped-ram
ri-heavy-scorpion
ri-onager
ri-scorpion
ri-siege-onager
ri-siege-ram

**stable research items:**

ri-bloodlines
ri-cavalier
ri-heavy-camel
ri-husbandry
ri-hussar
ri-light-cavalry
ri-paladin

**town center research items:**

ri-hand-cart
ri-loom
ri-town-patrol
ri-town-watch

ri-wheel-barrow

**university research items:**

ri-architecture
ri-ballistics
ri-bombard-tower
ri-chemistry
ri-fortified-wall
ri-guard-tower
ri-heated-shot
ri-keep
ri-masonry
ri-murder-holes
ri-siege-engineers
ri-stonecutting

**Note: The reason for having a “ri-“ prefix is to avoid duplicate symbols. Without the prefix, the current parser cannot distinguish between some research items and associated units/buildings. For example, trebuchet could mean unit or research item.**

### <resource-type>

is one of the following:

food
gold
stone
wood

### <shared-goal-id>

is a valid shared goal id. Shared goal ids have a range from 0 to 255.

### <signal-id>

is a valid signal id. Signal ids have a range from 0 to 255.

### <starting-resources>

is one of the following:

low-resources
medium-resources
high-resources

### <strategic-number>

is one of the following:

sn-percent-civilian-explorers
sn-percent-civilian-builders
sn-percent-civilian-gatherers
sn-cap-civilian-explorers
sn-cap-civilian-builders
sn-cap-civilian-gatherers
sn-minimum-attack-group-size
sn-total-number-explorers
sn-percent-enemy-sighted-response
sn-enemy-sighted-response-distance
sn-sentry-distance

sn-relic-return-distance
sn-minimum-defend-group-size
sn-maximum-attack-group-size
sn-maximum-defend-group-size
sn-minimum-peace-like-level
sn-percent-exploration-required
sn-zero-priority-distance
sn-minimum-civilian-explorers
sn-number-attack-groups
sn-number-defend-groups
sn-attack-group-gather-spacing
sn-number-explore-groups
sn-minimum-explore-group-size
sn-maximum-explore-group-size
sn-gold-defend-priority
sn-stone-defend-priority
sn-forage-defend-priority
sn-relic-defend-priority
sn-town-defend-priority
sn-defense-distance
sn-number-boat-attack-groups
sn-minimum-boat-attack-group-size
sn-maximum-boat-attack-group-size
sn-number-boat-explore-groups
sn-minimum-boat-explore-group-size
sn-maximum-boat-explore-group-size
sn-number-boat-defend-groups
sn-minimum-boat-defend-group-size
sn-maximum-boat-defend-group-size
sn-dock-defend-priority
sn-sentry-distance-variation
sn-minimum-town-size
sn-maximum-town-size
sn-group-commander-selection-method
sn-consecutive-idle-unit-limit
sn-target-evaluation-distance
sn-target-evaluation-hitpoints
sn-target-evaluation-damage-capability
sn-target-evaluation-kills
sn-target-evaluation-ally-proximity
sn-target-evaluation-rof
sn-target-evaluation-randomness
sn-camp-max-distance
sn-mill-max-distance
sn-target-evaluation-attack-attempts
sn-target-evaluation-range
sn-defend-overlap-distance
sn-scale-minimum-attack-group-size
sn-scale-maximum-attack-group-size
sn-attack-group-size-randomness
sn-scaling-frequency
sn-maximum-gaia-attack-response
sn-build-frequency
sn-attack-separation-time-randomness
sn-attack-intelligence

sn-initial-attack-delay
sn-save-scenario-information
sn-special-attack-type1

sn-special-attack-influence1

sn-minimum-water-body-size-for-dock
sn-number-build-attempts-before-skip
sn-max-skips-per-attempt
sn-food-gatherer-percentage
sn-gold-gatherer-percentage
sn-stone-gatherer-percentage
sn-wood-gatherer-percentage
sn-target-evaluation-continent
sn-target-evaluation-siege-weapon
sn-group-leader-defense-distance
sn-initial-attack-delay-type
sn-blot-exploration-map
sn-blot-size

sn-intelligent-gathering
sn-task-ungrouped-soldiers
sn-target-evaluation-boat
sn-number-enemy-objects-required
sn-number-max-skip-cycles
sn-retask-gather-amount
sn-max-retask-gather-amount

sn-max-build-plan-gatherer-percentage
sn-food-dropsite-distance
sn-wood-dropsite-distance
sn-stone-dropsite-distance
sn-gold-dropsite-distance
sn-initial-exploration-required
sn-random-placement-factor
sn-required-forest-tiles

sn-attack-diplomacy-impact
sn-percent-half-exploration
sn-target-evaluation-time-kill-ratio
sn-target-evaluation-in-progress
sn-attack-winning-player

sn-coop-share-information
sn-attack-winning-player-factor
sn-coop-share-attacking
sn-coop-share-attacking-interval
sn-percentage-explore-exterminators
sn-track-player-history
sn-minimum-dropsite-buffer
sn-use-by-type-max-gathering
sn-minimum-boar-hunt-group-size

sn-minimum-amount-for-trading
sn-easiest-reaction-percentage

sn-easier-reaction-percentage
sn-hits-before-alliance-change
sn-allow-civilian-defense
sn-number-forward-builders
sn-percent-attack-soldiers
sn-percent-attack-boats
sn-do-not-scale-for-difficulty-level
sn-group-form-distance
sn-ignore-attack-group-under-attack
sn-gather-defense-units
sn-maximum-wood-drop-distance
sn-maximum-food-drop-distance
sn-maximum-hunt-drop-distance
sn-maximum-fish-boat-drop-distance
sn-maximum-gold-drop-distance
sn- maximum-stone-drop-distance

sn-gather-idle-soldiers-at-center
sn-garrison-rams

### <string>

is a sequence of characters in double quotes.

### <string-id>

is a valid string id from a localized string table.
Ensemble Studios use only.

### <string-id-range>

is the size of a string id range.
Ensemble Studios use only.

### <string-id-start>

is a valid string id (from a localized string table) that defined beginning of a string id range.
Ensemble Studios use only.

### <taunt-id>

is a valid taunt id.

### <taunt-range>

is the size of a taunt range.

### <taunt-start>

is a taunt that defines a beginning of a taunt range.

### <timer-id>

is a valid timer id (range 1-10).

### <unit>

is one of the following (grouped by building):

**archery range units:**

arbalest
archer
cavalry-archer
crossbowman
elite-skirmisher
hand-cannoneer
heavy-cavalry-archer
skirmisher

**barracks units:**

champion
eagle-warrior
elite-eagle-warrior
halberdier
long-swordsman
man-at-arms
militiaman
pikeman
spearman
two-handed-swordsman

**castle units:**

berserk
cataphract
chu-ko-nu
conquistador
elite-berserk
elite-cataphract
elite-chu-ko-nu
elite-conquistador
elite-huskarl
elite-jaguar-warrior
elite-janissary
elite-longbowman
elite-mameluke
elite-mangudai
elite-plumed-archer
elite-samurai
elite-tarkan
elite-teutonic-knight
elite-throwing-axeman
elite-war-elephant
elite-war-wagon
elite-woad-raider
huskarl
jaguar-warrior
janissary
longbowman
mameluke
mangudai
petard
plumed-archer
samurai
tarkan
teutonic-knight

throwing-axeman
trebuchet
war-elephant
war-wagon
woad-raider

**dock units:**
cannon-galleon
demolition-ship
elite-cannon-galleon
elite-longboat
elite-turtle-ship
fast-fire-ship
fire-ship
fishing-ship
galleon
galley
heavy-demolition-ship
longboat
trade-cog
transport-ship
turtle-ship
war-galley

**market units:**
trade-cart

**monastery units:**
missionary
monk

**siege-workshop units:**
battering-ram
bombard-cannon
capped-ram
heavy-scorpion
mangonel
onager
scorpion
siege-onager
siege-ram

**stable units:**
camel
cavalier
heavy-camel
hussar
knight
light-cavalry
paladin
scout-cavalry

**town center units:**
villager

**For a description of the wildcard parameters accepted as <unit> parameters see the "Wildcard Parameters" section later in this document.**

### <value>

is a signed 16-bit integer.

### <victory-condition>

is one of the following:

standard
conquest
time-limit
score
custom

### <wall>

is one of the following:

fortified-wall
palisade-wall
stone-wall

or the following wildcard character:

stone-wall-line

## *Wildcard Parameters*

### <player-number> wildcard parameters:

any-ally
any-computer
any-computer-ally
any-computer-enemy
any-computer-neutral
any-enemy
any-human
any-human-ally
any-human-enemy
any-human-neutral
any-neutral

every-ally
every-computer
every-enemy
every-human
every-neutral

Usage of <player-number> wildcard parameters:

1. Wildcard parameters of the form "any-..." used in facts

The fact is true if it is true for at least one player that satisfies the given criteria.

Example:

```
 (defrule
        (players-current-age any-enemy == imperial-age)
        =>
        (chat-to-allies "At least one enemy in imperial")
)
```

2. Wildcard parameters of the form "every-..." used in facts

The fact is true if it is true for every player that satisfies the given criteria.

Example:

```
(defrule
        (players-current-age every-enemy == imperial-age)
        =>
  (chat-to-allies "All enemies are in imperial")
)
```

3. Wildcard parameters of the form "any-..." used in actions

The action executes for the first player that satisfies the given criteria.

Example:

```
(defrule
        (gold-amount > 10000)
        =>
  (tribute-to-player any-ally gold 1000)
)
```

4. Wildcard parameters of the form "every-..." used in actions

The action executes for every player that satisfies the given criteria.

Example:

```
(defrule
        (true)
        =>
  (set-stance every-human enemy)
        (set-stance every-computer ally)
        (chat-to-all "All computer players are my allies, all humans are my enemies")
        (disable-self)
)
```

**Note: Wildcard parameters applying to allies do not apply to self.**

## <building> wildcard parameters:

watch-tower-line

## <unit> wildcard parameters:

Grouped by building:

**archery range units:**

archer-line
cavalry-archer-line
skirmisher-line

**barracks units:**

eagle-warrior-line
militiaman-line
spearman-line

**castle units:**
berserk-line
cataphract-line
chu-ko-nu-line
conquistador-line
huskarl-line
jaguar-warrior-line
janissary-line
longbowman-line
mameluke-line
mangudai-line
plumed-archer-line
samurai-line
tarkan-line
teutonic-knight-line
throwing-axeman-line
war-elephant-line
war-wagon-line
woad-raider-line

**dock units:**
cannon-galleon-line
demolition-ship-line
fire-ship-line
galley-line
longboat-line
turtle-ship-line

**siege-workshop units:**
battering-ram-line
mangonel-line
scorpion-line

**stable units:**
camel-line
knight-line
scout-cavalry-line

These parameters are sometimes referred to as **unit line** or **line parameters**.

**For the use of <unit> wildcard parameters, see the "Usage of line parameters" section below.**

## <wall> wildcard parameters:

stone-wall-line

## Usage of line parameters:

Line parameters are interpreted by facts/actions in two ways:

a) They are translated into a currently available unit/building from a given line.
For example, action *(train knight-line)* will train a unit from knight-line
that is currently available. Which unit is trained (Knight, Cavalier, or Paladin)
is determined by the current state of research upgrades.

The following facts use line parameters in this fashion:

building-available
can-afford-building
can-afford-unit
can-build
can–build-wall
can-build-wall-with-escrow
can-build-with-escrow
can-train
can-train-with-escrow
unit-available

The same is true for the following actions:

build
build-forward
build-wall
train

b) They cause iteration on all buildings/units in the given line.
All unit/building count facts use this interpretation of line parameters.
For example *(unit-type-count militiaman-line > 5)* will take into account all units in the militia line: Militia, Man-at-Arms, etc.
It is rare to see more than one type of unit from a line at the same time. Exceptions are scenarios where any combination of units can be placed in the editor, and cases when units are converted.
The following facts use line parameters in an iterative fashion:

building-type-count
building-type-count-total
cc-players-building-type-count
cc-players-unit-type-count
players-building-type-count
players-unit-type-count
unit-type-count
unit-type-count-total

The implementation of line parameters is very fast so use it whenever needed.
All line parameters are named after the first unit/building in the line.

## *Difficulty Parameters*

Difficulty parameters are tactical parameters that should be adjusted according to the game’s difficulty setting.

### ability-to-dodge-missiles

Chance of a computer player’s unit dodging a missile. Valid range 0-100. Default value is 100.

For example, if an opponent shoots at your units with a cannon, this is the percentage chance that your units will try to avoid the area where the cannon ball will hit. Dodging makes the computer player harder to kill; when set to 100 the computer player will try to dodge every incoming shot.

### ability-to-maintain-distance

Chance that computer player's ranged unit will maintain the distance. Valid range 0-100. Default value is 100.

If a computer player is using an archer to attack an enemy knight, this is the percent chance that the archer will back up and fire if the knight advances toward it.  If 100, the archer will always back up.

## *AI Player Guidelines for The Conquerors Expansion*

### Level of Difficulty - Random Map Game

These are guidelines for the AI Player on a random map scaled to the level of difficulty as defined in The Age of Kings.

| AI Behaviors | EASIEST | EASY | MODERATE | HARD | HARDEST |
|---|---|---|---|---|---|
| | | | | | |
| Advances through the ages | After any human player has advanced to that age | Slowly as a novice player, or as Easiest does | As an experienced player | As fast as possible | As fast as possible; computer players cooperate to slingshot through the ages |
| Attacks First | Never | Never | Occasionally | Yes | Yes |
| Breaks alliances | No | Rarely | Rarely | Yes | Yes |
| Builds a Castle | Rarely | Seldom | Yes | Yes | Yes |
| Builds a Wonder | Never | Seldom | If possible | If possible | Seldom |
| Expansion and resource gathering | Slowly selects nearby resources; abandons contested resources | Slowly | Fast | Aggressively defends resources | Aggressively defends resources, destroys enemy resources |
| Monks used to convert buildings | Never | Rarely | Seldom | Yes | Yes |
| Monks used to convert units | Rarely and with very few Monks | Seldom or slowly | Yes | Yes | Yes |
| Starting diplomatic stance | Neutral | Neutral | Mix of Neutral and Enemy | Enemy | Enemy |
| Town siege | Never | Seldom | Yes | Yes | Yes |
| Walls and towers | Never | Sometimes defensive towers | Yes | Yes | Yes; may build offensive towers |
| Will ally with human players | Yes (unless game would end) | Yes (unless game would end) | With one human only (unless game would end) | Never | No |
| Will ally with humans | Depends on personality | Depends on personality | Sometimes | Never | Never |
| Will ally with other computer AIs | No | Yes | Yes | Preferred | Preferred |
| Will trade | Yes | Yes | Sometimes | No | No |
| x- Comments | | | | | AI is given additional resources at the start of the game |
| Cooperates against human player | No | Sometimes | Sometimes | Yes; coordinated attacks, optimized building strategies, etc. | Yes |

### *Age of Empires II: The Conqueror's Expansion Level of Difficulty - Current Operation*

Distance an enemy unit must be within when the computer player unit looks for a new target:

| | |
|---|---|
| easiest: | LOS (can be modified by sn-easiest-reaction-percentage) |
| easy: | LOS (can be modified by sn-easier-reaction-percentage) |
| moderate: | LOS * 2 |
| hard: | LOS * 2 |
| hardest: | LOS * 2 |

Computer players ignore relics on the easiest level.

If a non-exploring computer unit gets attacked, the computer player's attack delay is modified:

| | |
|---|---|
| easiest: | allow attacking one minute earlier |
| easy: | allow attacking two minutes earlier |
| moderate: | allow attacking immediately |
| hard: | allow attacking immediately |
| moderate: | allow attacking immediately |

After a wolf kills a unit, have it gorge itself (not attack again) for:

| | |
|---|---|
| easiest: | 35 seconds |
| easy: | 30 seconds |
| moderate: | 25 seconds |
| hard: | 20 seconds |
| hardest: | 15 seconds |

Distance a unit must be within when a wolf looks for a new target:

| | |
|---|---|
| easiest: | LOS * 0.5 |
| easy: | LOS * 0.75 |
| moderate: | LOS * 2 |
| hard: | LOS * 2 |
| hardest: | LOS * 2 |

### *Difficulty differences for Wonder Race*

Easiest:

Only creates 5 villagers.

Does not research Wheelbarrow, Stone Mining, Gold Mining, Double Bit Axe, Bow Saw, Horse Collar, Heavy Plow, Hand Cart, Stone Shaft Mining, Gold Shaft Mining, Two Man Saw, or Crop Rotation.

Easy:

Only creates 10 villagers.

Does not research the same technologies listed in Easiest.

Moderate:

Creates 35 villagers in the Dark Age, 40 villagers in the Feudal Age, 55 villagers in the Castle Age, and no limit in the Imperial Age beyond the pop cap (these numbers only apply to pop cap 75 - the numbers differ for other pop caps).
Does not research Hand Cart, Stone Shaft Mining, Gold Shaft Mining, Two Man Saw, or Crop Rotation.

Hard:

Creates the same number of villagers as Moderate.

Hardest:

Same as Hard, but every 20 minutes, gets 500 extra of each resource.

## *Difficulty differences for Defend the Wonder*

Easiest:

(set-strategic-number sn-percent-enemy-sighted-response 10)
(set-strategic-number sn-consecutive-idle-unit-limit 60)
(set-strategic-number sn-easiest-reaction-percentage 20)
(set-difficulty-parameter ability-to-maintain-distance 100)
(set-difficulty-parameter ability-to-dodge-missiles 100)
Builds few houses

Easy:

(set-strategic-number sn-percent-enemy-sighted-response 25)
(set-strategic-number sn-consecutive-idle-unit-limit 20)
(set-strategic-number sn-easier-reaction-percentage 20)
(set-strategic-number sn-hits-before-alliance-change 50)
(set-difficulty-parameter ability-to-maintain-distance 75)
(set-difficulty-parameter ability-to-dodge-missiles 75)
Builds few houses, but not as few as on Easiest.

Moderate:

(set-strategic-number sn-percent-enemy-sighted-response 75)
(set-strategic-number sn-consecutive-idle-unit-limit 5)
(set-strategic-number sn-hits-before-alliance-change 25)
(set-difficulty-parameter ability-to-maintain-distance 50)
(set-difficulty-parameter ability-to-dodge-missiles 50)

Hard:

(set-strategic-number sn-percent-enemy-sighted-response 99)
(set-strategic-number sn-consecutive-idle-unit-limit 1)
(set-strategic-number sn-hits-before-alliance-change 10)
(set-difficulty-parameter ability-to-maintain-distance 0)
(set-difficulty-parameter ability-to-dodge-missiles 0)

Hardest:

Same as Hard, but gets extra resources.

## *Difficulty differences for King of the Hill*

Exactly the same as for the regular random game maps, because it uses the same AI.

## Variables

### *Rule Variables*

Rule variables are variables with a rule scope. This means that the variables are set within a rule and can be used only within the same rule.
Only implicit variables are supported - the variables set by the system.

Here is a list of currently available variables:

this-any-ally
this-any-computer
this-any-computer-ally
this-any-computer-enemy
this-any-computer-neutral
this-any-enemy
this-any-human
this-any-human-ally
this-any-human-enemy
this-any-human-neutral
this-any-neutral

The variables have to be used as <player-number> argument in one of the following actions:

chat-to-player
chat-to-player-using-id
clear-tribute-memory
set-stance
tribute-to-player

The variables are set by the associated wildcard parameters used in facts. For example:

*any-enemy* wildcard that is successful will store its result in *this-any-enemy*

Here is an example of how variables can be used in rules:

```
(defrule
   (players-civ any-enemy gothic)
   =>
   (chat-to-player this-any-enemy "I know you are a Goth")
   (disable-self)
)
```

In this example the fact *players-civ* looks for an enemy player that is a Goth. If the enemy with that civilization is found, the fact is true causing the rule to trigger. At the same time, the result of the wildcard search is stored in *this-any-enemy.* The action *chat-to-player* is executed and uses *this-any-enemy* variable to send a message to the appropriate enemy that has chosen to be a Goth.
It is important to remember that the variables have a rule scope. This means that once the rule has executed the value in the variable becomes invalid.
For example, if the following rule followed the one above, the message "Hi, my Goth enemy" would not be sent.

```
(defrule
  (true)
  =>
  (chat-to-player this-any-enemy "Hi, my Goth enemy") ; this is never sent
  (disable-self)
)
```

## Timers

Timers provide a mechanism to trigger one or more rules after a given time interval. Timers have player scope. Each computer player gets 10 timers.

### *Timer Facts:*

**timer-triggered**

### *Timer Actions:*

**enable-timer**
**disable-timer**

### *Examples:*

Example 1

Here is an example of how tribute demand can be handled using timers. After 10 minutes of playing the computer player asks for tribute and waits 5 minutes to get it.

```
; After 10 minutes of playing ask for tribute, start 5 minute timer

(defrule
  (game-time greater-than 600)
  =>
  (chat-to-player 1 "Give me 500 gold in the next 5 minutes")
  (clear-tribute-memory 1 gold)
  (enable-timer 1 300)
  (disable-self))

; Tribute not received in time, declare player 1 to be an enemy
; Note that explicit disabling of timer is necessary even after it
;  triggers

(defrule
  (timer-triggered 1)
  =>
  (disable-timer 1)
  (chat-to-player 1 "Time is up. We are enemies now")
  (set-stance 1 enemy))

; Tribute received in time. Disable the timer

(defrule
  (players-tribute-memory 1 gold greater-or-equal 500)
  =>
  (disable-timer 1)
  (clear-tribute-memory 1 gold)
  (chat-to-player 1 "Thanks"))
```

Example 2

Here is even better example that uses two timers. Every 15 minutes throughout the game the computer player asks for tribute and waits 5 minutes to get it.

```
; Schedule 15 minute timer for the first time

(defrule
  (true)
  =>
  (enable-timer 2 900)
  (disable-self))

; Every 15 minutes ask for tribute and wait 5 minutes to get it.
; Restart 15 minute timer.

(defrule
  (timer-triggered 2)
  =>
  (disable-timer 2)
  (enable-timer 2 900)
  (chat-to-player 1 "Give me 500 gold in the next 5 minutes.")
  (clear-tribute-memory 1 gold)
  (enable-timer 1 300))

; Tribute not received in time, declare player 1 to be an enemy.
; No need to ask for tribute again - disable both timers.

(defrule
  (timer-triggered 1)
  =>
  (disable-timer 1)
  (disable-timer 2)
  (chat-to-player 1 "Time is up. We are enemies now")
  (set-stance 1 enemy))

; Tribute received in time. Disable the 5 minute timer

(defrule
  (players-tribute-memory 1 gold greater-or-equal 500)
  =>
  (disable-timer 1)
  (clear-tribute-memory 1 gold)
  (chat-to-player 1 "Thanks"))
```

## Error Messages

Errors will be reported in an error dialog when the game starts. Only one error at a time will be reported.

### *Error reporting format*

*Player number*
*File name*
*Line number: error code: description*

For some errors the line numbers are not relevant and can be omitted
(for example, file open failed).

## Description of error codes

ERR2xxx Syntax errors
ERR3xxx Preprocessor errors
ERR5xxx File errors
ERR6xxx Memory allocation errors
ERR8xxx Miscellaneous errors
ERR9xxx Undocumented errors

Undocumented errors are a safeguard for errors that fall through logic without being handled.

## List of errors

ERR2001: Missing opening parenthesis
ERR2002: Missing keyword
ERR2003: Invalid keyword
ERR2004: Missing identifier
ERR2005: Invalid identifier
ERR2006: Missing file name
ERR2007: Missing left-hand side (LHS) of the rule
ERR2008: Missing arrow
ERR2009: Missing right-hand side (RHS) of the rule
ERR2010: Missing closing quote
ERR2011: Missing closing parenthesis
ERR2012: Constant already defined
ERR2013: Unexpected end-of-file

ERR3001: Invalid preprocessor directive
The given directive is not one of the following:
#load-if-defined
#load-if-not-defined
#else
#end-if

ERR3002: Missing preprocessor symbol
Preprocessor directive is expecting a preprocessor symbol to follow.

ERR3003: Preprocessor nesting too deep
Preprocessor directives are nested more than 50 levels deep.

ERR3004: Unexpected #else
Found #else without matching #load-if-… directive.

ERR3005: Unexpected #end-if
Found #end-if without matching #load-if-… directive.

ERR3006: Missing #end-if
End-of-file reached with outstanding #load-if-… directive and no matching #end-if.

ERR5001: File open failed
ERR5002: File read failed
ERR6001: List full
ERR6002: Rule too long

ERR6003: String table full
ERR8001: No rules
ERR9000: Undocumented error
ERR9001: Unexpected error

### *Hints for efficient testing and debugging of scripts*

When an error is detected, do the following:

- Press OK to close the error dialog.
- Tab out of the game. Do not stop the game!
- Use a text editor to edit script that has the error.
- Tab back into the game.
- Use the game menu to restart the game.

Repeat until the script has no errors.

You can specify your AI script to run in a game scenario by selecting its name in the Players section. Making a custom scenario from a random map will allow you to try out your AI on a known map so it is easier to evaluate its performance.

You can use the team number setting on the pre-game settings screen to have your AI ally with other players (human or computer) or force it to fight against the computer AI.

## Data Types

### *String*

A string is a set of characters that starts with double quotes (“) and is followed by zero or more printable characters. A string ends with double quotes.

### *Symbol*

A symbol is any sequence of characters that starts with any printable ASCII character and is followed by zero or more printable ASCII characters. When a delimiter is found the symbol is ended. The following characters act as delimiters: space, tab, carriage return, line feed, and opening and closing parentheses “(“ and “)”. A semicolon “;”, which is used to start a comment, also acts as a delimiter. Symbols are case sensitive.

# Appendix A - Internal Strategic Number (SN) parameter documentation

## Age of Empires Personality Types

### File names

These rules are initialized out of files with a .per extension. These files reside in the Data folder where you installed Age of Empires. They must have a .per extension to be recognized by the scenario builder and the AI loading mechanism.

### File content

The .per files simply contain a list of number pairs. The first number is keyed to the strategic number, the second is the value for that strategic number. The pairs do not need to be ordered. There are minimal facilities for bounds checking during the read process, but there are no checks for values that cause bad AI behavior or game slowdowns.

### Strategic number description

The following is a list of the strategic numbers. The number on the left is the key for the strategic number on the right (this is the key that must be referenced in the VC file). Below each pair is a description of what that strategic number represents.

*CIVILIAN NUMBERS*
`sn-percent-civilian-explorers`
Controls the normal, formula-based percentage of civilian explorers allocated. Must be >= 0.

`sn-percent-civilian-builders`
Controls the normal, formula-based percentage of builders allocated. Must be >= 0.

`sn-percent-civilian-gatherers`
Controls the normal, formula-based percentage of gatherers allocated. Must be >= 0.

`sn-cap-civilian-explorers`
Caps the number of civilian explorers allocated.
Factored in after the percentage is calculated. Ignored when set to –1. Must be >= -1.

`sn-cap-civilian-builders`
Caps the number of builders allocated. Factored in after the percentage is calculated. Ignored when set to –1. Must be >= -1.

`sn-cap-civilian-gatherers`
Caps the number of gatherers allocated. Factored in after the percentage is calculated. Ignored when set to –1. Must be >= -1.

`sn-total-number-explorers`
Caps the total number of explorers/explorer groups allocated. Factored in after the percentage of civilian explorers is calculated and the soldier groups are set up. Ignored when set to –1.

`sn-minimum-civilian-explorers`
Sets a minimum number of civilian explorers. Must be >= 0.

`sn-food-gatherer-percentage`
The required percentage of food gatherers. Must be >= 0 and <= 100. This is applied before the normal calculation formula takes effect.

`sn-gold-gatherer-percentage`
The required percentage of gold gatherers. Must be >= 0 and <= 100. This is applied before the normal calculation formula takes effect.

`sn-stone-gatherer-percentage`
The required percentage of stone gatherers. Must be >= 0 and <= 100. This is applied before the normal calculation formula takes effect.

`sn-wood-gatherer-percentage`
The required percentage of wood gatherers. Must be >= 0 and <= 100. This is applied before the normal calculation formula takes effect.

`sn-number-enemy-objects-required`
The count of the number of enemy objects the computer player must see before dropping the number of civilian explorers down to the minimum level. Number must be >= 0.

`sn-retask-gather-amount`
The minimum amount that a gatherer must gather before the computer player allows him to be retasked to another resource type. Some code may override this. Must be >= 0.

`sn-max-retask-gather-amount`
The maximum amount that a gatherer can be told to gather before being allowed to be retasked. Some code may override this. Must be >= 0.

`sn-initial-exploration-required`
The percentage of the map that must be explored by a computer player before any building is allowed. Must be >= 0 and <= 100.

`sn-use-by-type-max-gathering`
Controls whether or not logical, type-specific gatherer requirements are placed on the quantity of resources gatherers must collect before being allowed to be retasked. Must be 0 or 1.

`sn-percent-half-exploration`
The percentage of map exploration that allows the computer player to task down to half the number of explorers. Must be >= 0.

`sn-minimum-boar-hunt-group-size`
The number of civilians a computer player must collect before allowing boars to be hunted for food. Must be >= 1.

`sn-number-forward-builders`
The number of civilians a computer player uses to build outside of an enemy town. Forward builders refer specifically to those villagers that must board a Transport to cross over water that cannot otherwise be pathed, either because players are on islands, or because other forms of access have been walled-off. It is not necessary to specify forward builders, unless the villagers need to board a Transport. Must be >=0.  Default is 0.

*GROUP-RELATED NUMBERS*
`sn-group-commander-selection-method`
Sets the method by which group commanders are selected. 0 selects the unit with the most hit points. 1 selects the unit with the fewest hit points. 2 selects the unit with the most range. The commander is set once during a group’s creation and is only reset when the commander dies. Must be >= 0 and <= 2.

`sn-consecutive-idle-unit-limit`
Sets the number of consecutive seconds that pass before a group is set to idle if all of its units are idle. This is only used during attack and retreat phases. Must be >= 0.

`sn-task-ungrouped-soldiers`
Controls whether or not ungrouped computer player soldiers get tasked to spread out and guard the computer player's general town area. Must be 0 or 1.

*ATTACK GROUP NUMBERS*
`sn-number-attack-groups`
Sets the desired number of land-based attack groups. Must be >= 0. `Sn-percent-attack-soldiers` generally works better.

`sn-minimum-attack-group-size`
Sets the minimum size of land-based attack groups. A group must meet its minimum size as one of the tasking prerequisites. Must be >= 0.

`sn-maximum-attack-group-size`
Sets the maximum size of land-based attack groups.
Must be >= 0 and >= sn-minimum-attack-group-size.

`sn-group-form-distance`
Sets the distance over which attack soldiers will group. Set this value high if buildings that train military units are far apart. Default = 20.

`sn-scale-minimum-attack-group-size`
The scaling factor for the minimum attack group size. Added to SNMinimumAttackGroupSize when the tactical AI does its scaling.

`sn-scale-maximum-attack-group-size`
The scaling factor for the maximum attack group size. Added to SNMinimumAttackGroupSize when the tactical AI does its scaling.

`sn-attack-group-size-randomness`
The randomness factor in the attack group size. This sets a cap on the amount of randomness in the minimum attack group size. The randomness factor is set once (when the group is created) and will be between 0 and this number.

`sn-attack-group-gather-spacing`
Controls the relative proximity (to the group gather point) that grouped units must be in before the group is considered gathered. Must be >= 1.

`sn-group-leader-defense-distance`
Sets the defense distance for defenders of an important attack group leader. Must be >= 1.

`sn-percent-attack-soldiers`
Sets the percentage of defense soldiers that will be sent into battle (modified for difficulty level) the next time attack-now is issued. All newly created soldiers are defense soldiers by default, and will remain defense soldiers until attack-now is issued. For example, if 10 soldiers were defending a town, and `sn-percent-attack-soldiers` was set to 50, then 5 soldiers will form an attack group and attack. This SN only needs to be set once, but it can be changed as needed. `sn-percent-attack-soldiers` works best when not using `sn-number-defend-groups.`
Must be >= 0 and <= 100. Default = 0.

`sn-percent-attack-boats`
Sets the percentage of defense boats that will be sent into battle (modified for difficulty level) the next time attack-now is issued. All newly created boats are defense boats by default, and will remain defense boats until attack-now is issued. Both attack soldiers and attack boats will attack when attack-now is issued. This SN only needs to be set once, but it can be changed as needed.
Must be >= 0 and <= 100. Default = 0.

*MISCELLANEOUS ATTACK NUMBERS*

`sn-percent-enemy-sighted-response`
Sets the percentage of idle troops that will respond to another unit being attacked. Must be >= 0 and <= 100.

`sn-enemy-sighted-response-distance`
Sets the distance inside of which units will be candidates for response to an enemy attack. Must be >= 0 and <= 144.

`sn-blot-exploration-map`
This controls whether or not the computer player re-explores previously explored regions. A value of 1 has the computer player re-explore, a value of 0 does not.

`sn-blot-size`
This controls the size of the area that a computer player marks for re-exploration. Must be > 0 and < the map size.

`sn-maximum-gaia-attack-response`
The maximum number of civilians that respond to another civilian getting attacked by a Gaia animal. Must be >= 0.

`sn-attack-separation-time-randomness`
The amount of randomness incorporated into the attack separation time.

`sn-attack-intelligence`
Specifies whether the intelligent attack system is used. The intelligent attack system tries to avoid enemy units when attacking and tries to attack from different sides.

`sn-initial-attack-delay`
The forced, initial delay before any computer player attacks (in seconds). Must be >= 0.

`sn-initial-attack-delay-type`
The type of initial attack delay. A value of 1 denotes a delay ended by the build list. A value of 2 uses the `sn-initial-attack-delay` timeout. A value of 3 allows the computer player to attack after he has been attacked by a non-Gaia player. A value of 0 allows any of the three occurrences to enable attacks.

`sn-garrison-rams`
Defaults to 1. Set to 0 to turn off. When on, the CP AI *tries* (but doesn't always succeed) to put infantry units into rams before the attack group departs.

*DEFEND GROUP NUMBERS*

`sn-number-defend-groups`
Sets the desired number of land-based defend groups. Must be >= 0.

`sn-minimum-defend-group-size`
Sets the minimum size of land-based defend groups. A group must meet its minimum size as one of the tasking prerequisites. Must be >= 0.

`sn-maximum-defend-group-size`
Sets the maximum size of land-based defend groups.
Must be >= 0 and >= sn-minimum-defend-group-size.

`sn-gold-defend-priority`
Sets the priority of defending gold. A priority of 0 means that gold will not be defended. A priority of 1 means that gold has the highest defend priority. Must be >= 0 and <= 7.

`sn-stone-defend-priority`
Sets the priority of defending stone. Must be >= 0 and <= 7.

`sn-forage-defend-priority`
Sets the priority of defending forage sites. Must be >= 0 and <= 7.

`sn-relic-defend-priority`
Sets the priority of defending relics. Must be >= 0 and <= 7.

`sn-town-defend-priority`
Sets the priority of defending the computer player's town. Must be >= 0 and <= 7.

`sn-defense-distance`
Sets the distance at which items (town excluded) are defended. Must be >= 0.

`sn-sentry-distance`
Sets the distance at which the town is defended. Must be >= 0.

`sn-sentry-distance-variation`
Sets the amount of allowable variation in the defense distances. Must be >= 0.

`sn-defend-overlap-distance`
Sets the amount of influence that a defend group has. Defend groups will be assigned so that their circles of influence do not overlap. Must be >= 0.

`sn-gather-idle-soldiers-at-center`
Defaults to 0. When set to 1, it will "move" the town defense gather point to the "center" (randomized +-6 tiles) of the map. No provision is made if the center is in an unreachable spot. When it's set, all idle and retreating units will try to go to the center. Useful for King of the Hill and similar variants to get the CP to group near the middle.

*EXPLORE GROUP NUMBERS*
`sn-number-explore-groups`
Sets the desired number of land-based soldier exploration groups. Must be >= 0.

`sn-minimum-explore-group-size`
Sets the minimum size of land-based soldier exploration groups. A group must meet its minimum size as one of the tasking prerequisites. Must be >= 0.

`sn-maximum-explore-group-size`
Sets the maximum size of land-based soldier exploration groups.
Must be >= 0 and >= sn-minimum-explore-group-size.

*BOAT ATTACK GROUP NUMBERS*
`sn-number-boat-attack-groups`
Sets the desired number of boat attack groups. Setting `sn-percent-attack-boat` usually works better. Must be >= 0.

`sn-minimum-boat-attack-group-size`
Sets the minimum size of boat attack groups. A group must meet its minimum size as one of the tasking prerequisites. Must be >= 0.

`sn-maximum-boat-attack-group-size`
Sets the maximum size of boat attack groups.
Must be >= 0 and >= sn-minimum-boat-attack-group-size.

*BOAT DEFEND GROUP NUMBERS*

`sn-number-boat-defend-groups`
Sets the desired number of boat defend groups. Must be >= 0.

`sn-minimum-boat-defend-group-size`
Sets the minimum size of boat defend groups. Must be >= 0.

`sn-maximum-boat-defend-group-size`
Sets the maximum size of boat defend groups.
Must be >= 0 and >= sn-minimum-boat-defend-group-size.

`sn-dock-defend-priority`
Sets the priority of defending a Dock. 0 does not protect Docks, 1 does. Must be either 0 or 1.

*BOAT EXPLORE GROUP NUMBERS*
`sn-number-boat-explore-groups`
Sets the desired number of boat exploration groups. Must be >= 0.

`sn-minimum-boat-explore-group-size`
Sets the minimum size of boat exploration groups. Must be >= 0.

`sn-maximum-boat-explore-group-size`
Sets the maximum size of boat exploration groups.
Must be >= 0 and >= sn-minimum-boat-explore-group-size.

*TOWN BUILDING NUMBERS*
`sn-minimum-town-size`
Sets the minimum size of a computer player town. Must be >= 0.

`sn-maximum-town-size`
Sets the maximum size of a computer player town.
Must be >= 0 and >= sn-minimum-town-size.

`sn-camp-max-distance`
Sets the maximum distance that lumber camp and mining camp may be placed from a Town Center. Must be >= 0.

`sn-mill-max-distance`
Sets the maximum distance that mills may be placed from a Town Center. Must be >= 0.

`sn-minimum-water-body-size-for-dock`
The minimum number of tiles (in surface area) that a body of water must be for a Dock to be placed on it. Must be >= 10.

`sn-max-build-plan-gatherer-percentage`
The maximum percentage of gatherers that a computer player will task based on the pregame requirements of the build plan. Must be >= 0 and <= 100.

`sn-food-dropsite-distance`
The maximum number of tiles a computer player likes to walk to drop off its food. Must be >= 3.

`sn-wood-dropsite-distance`
The maximum number of tiles a computer player likes to walk to drop off its wood. Must be >= 3.

`sn-stone-dropsite-distance`
The maximum number of tiles a computer player likes to walk to drop off its stone. Must be >= 3.

`sn-gold-dropsite-distance`
The maximum number of tiles a computer player likes to walk to drop off its gold. Must be >= 3.

`sn-minimum-dropsite-buffer`
Controls how far away a computer player will keep dropsites in relation to enemy Town Centers. Must be 0 or 1.

`sn-random-placement-factor`
A number that gets added into the placement of the computer player to randomize building placement (off of the calculated ideal). Must be >= 0.

`sn-required-forest-tiles`
The minimum number of forest tiles that a computer player must uncover before placing its first lumber camp. Must be >= 0.

*TARGET EVALUATION NUMBERS*

`sn-target-evaluation-distance`
Sets the multiplier used for the distance rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-hitpoints`
Sets the multiplier used for the hit point rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-damage-capability`
Sets the multiplier used for the damage capability rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-kills`
Sets the multiplier used for the kill rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-ally-proximity`
Sets the multiplier used for the ally proximity (number of allies in range) rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-rof`
Sets the multiplier used for the rate of fire rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-randomness`
Sets the multiplier used for the randomness factor in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-attack-attempts`
Sets the multiplier used for the attack attempts rating in computer player target evaluation. Must be >= 0.

`sn-target-evaluation-range`
Sets the multiplier used for the range rating in computer player target evaluation. Must be >= 0.

`sn-special-attack-type1`
Sets the type of object (first slot) that the computer player particularly wants to attack. Must be a valid master object ID or –1 if no special attack type is desired.

`sn-special-attack-influence1`
Sets the multiplier used for the special attack type 1 rating in computer player target evaluation. Must be > 0 to influence the computer player toward attacking the special type 1, < 0 to influence the computer player away from attacking the special type 1.

`sn-target-evaluation-continent`

Sets the additive value used for the targets on the same continent as the attack group commander. Must be > 0 to influence the computer player toward attacking the units on the same continent or 0 for no special influence.

`sn-target-evaluation-siege-weapon`
Sets the additive value used for influencing siege weapons to attack stationary targets (and influencing non-siege weapons not to attack those stationary targets). Must be > 0 to influence the computer player to use siege weapons to attack stationary targets or 0 for no special influence.

`sn-target-evaluation-boat`
Sets the additive value used for influencing land units to attack or not attack boats. Must be > 0 to influence land units to attack boats, 0 for no special influence, and less than 0 for aversion.

`sn-target-evaluation-time-kill-ratio`
The amount of influence the time to kill a target has in deciding what to attack. Must be >= 0.

`sn-target-evaluation-in-progress`
The amount of influence of continuing to attack a target already under attack. Must be >= 0.

### *MISCELLANEOUS NUMBERS*

`sn-do-not-scale-for-difficulty-level`
Disables the automatic difficulty-scaling. See Level of Difficulty - Random Map Game. Default = 0. Must be 0 or 1.

`sn-relic-return-distance`
Sets the distance that relics must be within to be considered returned to the Town Center. Must be >= 0.

`sn-minimum-peace-like-level`
Sets the level at which computer players must like another player before allying with that player. Must be >= 0 and <= 100.

`sn-percent-exploration-required`
Sets the minimum amount of exploration that a computer player must have accomplished before being allowed to retask civilian explorers. Must be >= 0 and <= 100.

`sn-zero-priority-distance`
Sets the distance at which a computer player’s order for a unit is set to a priority of 0. Must be >= 0 and <= 144.

`sn-scaling-frequency`
Sets the number of minutes that pass between each scaling inside the tactical AI. Must be >= 0.

`sn-build-frequency`
Sets the number of tactical AI updates that pass between each training or research attempt. Must be >= 0.

`sn-save-scenario-information`
Controls whether the learning information is saved at the end of the scenario for a given computer player. Must be 0 (to turn off) or 1 (to turn on).

`sn-number-build-attempts-before-skip`
The maximum number of build attempts a build plan can go through before being put into skip mode. Must be >= 1.

`sn-max-skips-per-attempt`

The maximum number of unbuilt items that can be skipped during any build plan processing before giving up (for being too far ahead of the current position in the plan). Must be >= 1.

`sn-minimum-amount-for-trading`
Controls how much of a resource a computer player must have before using it for trading. Must be >= 0.

`sn-hits-before-alliance-change`
Sets the number of times a computer player will allow his units to be hit by an ally before allowing his diplomacy to be changed. Must be >= 0.

`sn-attack-diplomacy-impact`
The impact (positive or negative) that a computer player injects into his diplomacy system when attacked. Must be >= 0 and <= 100.

`sn-easiest-reaction-percentage`
Sets the effective reaction percentage (of normal LOS) a computer player unit will use in single-player Easiest level scenario or campaign games. Must be >= 0 and <= 100.

`sn-easier-reaction-percentage`
Sets the effective reaction percentage (of normal LOS) a computer player unit will use in single-player easier scenario or campaign games. Must be >= 0 and <= 100.

`sn-track-player-history`
Decides whether or not a human player's tendencies are tracked or not. Must be 0 or 1.

`sn-attack-winning-player`
Controls whether or not the computer player will attack the winning player (if there is more than one to choose from). Must be 0 or 1.

`sn-coop-share-information`
Controls whether or not allied computer players share information about what they uncover (this is not like Cartography; instead, it's analogous to two humans chatting). Must be 0 or 1.

`sn-attack-winning-player-factor`
The influence the `SNAttackWinningPlayer` will have on deciding who to attack if it's set to 1. Must be >= 0 and <= 100.

`sn-coop-share-attacking`
Controls whether allied computer players can attack to defend each other. Must be 0 or 1.

`sn-coop-share-attacking-interval`
Controls how often this computer player can ask another for help (in seconds). Must be >= 0.

`sn-percentage-explore-exterminators`
Determines how many of the computer player's soldier explore groups are set as extermination groups. Must be >= 0 and <= 100.

`sn-maximum-wood-drop-distance`
`sn-maximum-food-drop-distance`
`sn-maximum-hunt-drop-distance`
`sn-maximum-fish-boat-drop-distance`
`sn-maximum-gold-drop-distance`
`sn-maximum-stone-drop-distance`
The parameters control how far from a dropsite a given resource type can be before the CP ignores it. -1 indicates a "don't care" -- i.e. it can be any distance (as it used to be).
All parameters are by default set to –1.

By setting the parameters to the appropriate value it is possible to avoid having villagers walk all over the map to gather resources.

## Appendix B - SN Parameter Defaults

(from the internal codes)

| | |
|---|---|
| 34 | sn-percent-civilian-explorers |
| 0 | sn-percent-civilian-builders |
| 66 | sn-percent-civilian-gatherers |
| 2 | sn-cap-civilian-explorers |
| 2 | sn-cap-civilian-builders |
| -1 | sn-cap-civilian-gatherers |
| 4 | sn-minimum-attack-group-size |
| 4 | sn-total-number-explorers |
| 50 | sn-percent-enemy-sighted-response |
| 50 | sn-enemy-sighted-response-distance |
| 12 | sn-sentry-distance |
| 10 | sn-relic-return-distance |
| 1 | sn-minimum-defend-group-size |
| 10 | sn-maximum-attack-group-size |
| 4 | sn-maximum-defend-group-size |
| 85 | sn-minimum-peace-like-level |
| 100 | sn-percent-exploration-required |
| 50 | sn-zero-priority-distance |
| 0 | sn-minimum-civilian-explorers |
| 3 | sn-number-attack-groups |
| 0 | sn-number-defend-groups |
| 4 | sn-attack-group-gathers-pacing |
| 0 | sn-number-explore-groups |
| 1 | sn-minimum-explore-group-size |
| 3 | sn-maximum-explore-group-size |
| 0 | sn-gold-defend-priority |
| 0 | sn-stone-defend-priority |
| 0 | sn-forage-defend-priority |
| 0 | sn-relic-defend-priority |
| 7 | sn-town-defend-priority |
| 3 | sn-defense-distance |
| 2 | sn-number-boat-attack-groups |
| 1 | sn-minimum-boat-attack-group-size |
| 5 | sn-maximum-boat-attack-group-size |
| 1 | sn-number-boat-explore-groups |
| 1 | sn-minimum-boat-explore-group-size |
| 2 | sn-maximum-boat-explore-group-size |
| 0 | sn-number-boat-defend-groups |
| 0 | sn-minimum-boat-defend-group-size |
| 0 | sn-maximum-boat-defend-group-size |
| 0 | sn-dock-defend-priority |
| 2 | sn-sentry-distance-variation |
| 12 | sn-minimum-town-size |
| 20 | sn-maximum-town-size |
| 3 | sn-group-commander-selection-method |
| 15 | sn-consecutive-idle-unit-limit |
| 50 | sn-target-evaluation-distance |
| 0 | sn-target-evaluation-hitpoints |
| 0 | sn-target-evaluation-damage-capability |
| 0 | sn-target-evaluation-kills |
| 0 | sn-target-evaluation-ally-proximity |

```
0     sn-target-evaluation-rof
0     sn-target-evaluation-randomness
25    sn-camp-max-distance
100   sn-mill-max-distance
-25   sn-target-evaluation-attack-attempts
0     sn-target-evaluation-range
5     sn-defend-overlap-distance
1     sn-scale-minimum-attack-group-size
0     sn-scale-maximum-attack-group-size
1     sn-attack-group-size-randomness
10    sn-scaling-frequency
3     sn-maximum-gaia-attack-response
1     sn-build-frequency
15    sn-attack-separation-time-randomness
0     sn-attack-intelligence
0     sn-initial-attack-delay
0     sn-save-scenario-information
-1    sn-special-attack-type1

0     sn-special-attack-influence1
300   sn-minimum-water-body-size-for-dock
25    sn-number-build-attempts-before-skip
10    sn-max-skips-per-attempt
0     sn-food-gatherer-percentage
0     sn-gold-gatherer-percentage
0     sn-stone-gatherer-percentage
0     sn-wood-gatherer-percentage
100   sn-target-evaluation-continent
0     sn-target-evaluation-siege-weapon
3     sn-group-leader-defense-distance
0     sn-initial-attack-delay-type
1     sn-blot-exploration-map
15    sn-blot-size
0     sn-intelligent-gathering
1     sn-task-ungrouped-soldiers
0     sn-target-evaluation-boat
10    sn-number-enemy-objects-required
50    sn-number-max-skip-cycles
20    sn-retask-gather-amount
40    sn-max-retask-gather-amount

50    sn-max-build-plan-gatherer-percentage
7     sn-food-dropsite-distance
10    sn-wood-dropsite-distance
25    sn-stone-dropsite-distance
7     sn-gold-dropsite-distance
2     sn-initial-exploration-required
50    sn-random-placement-factor
10    sn-required-forest-tiles
10    sn-attack-diplomacy-impact
30    sn-percent-half-exploration
20    sn-target-evaluation-time-kill-ratio
50    sn-target-evaluation-in-progress
1     sn-attack-winning-player
1     sn-coop-share-information
25    sn-attack-winning-player-factor
1     sn-coop-share-attacking
120   sn-coop-share-attacking-interval
50    sn-percentage-explore-exterminators
0     sn-track-player-history
```

```
25    sn-minimum-dropsite-buffer
1     sn-use-by-type-max-gathering
5     sn-minimum-boar-hunt-group-size

50    sn-minimum-amount-for-trading
100   sn-easiest-reaction-percentage
100   sn-easier-reaction-percentage
3     sn-hits-before-alliance-change

1     sn-allow-civilian-defense

0     sn-do-not-scale-for-difficulty-level
20    sn-group-form-distance
0     sn-ignore-attack-group-under-attack
0     sn-gather-defense-units

-1    sn-maximum-wood-drop-distance
-1    sn-maximum-food-drop-distance
-1    sn-maximum-hunt-drop-distance
-1    sn-maximum-fish-boat-drop-distance
-1    sn-maximum-gold-drop-distance
-1    sn-maximum-stone-drop-distance


0     sn-gather-idle-soldiers-at-center
1     sn-garrison-rams
```

## Facts, Actions, and Parameters (combined list)

# Some Examples

## *Controlling Villager Distribution*

### Dark Age Distribution

**Note: Separate logic sets resource-needed goal.**

```
(defrule
        (goal resource-needed NO)
        (current-age == dark-age)
        (civilian-population < 10)
        (not (strategic-number sn-wood-gatherer-percentage == 10) )
=>
        (set-strategic-number sn-wood-gatherer-percentage 10)
        (set-strategic-number sn-food-gatherer-percentage 90)
        (set-strategic-number sn-gold-gatherer-percentage 0)
        (set-strategic-number sn-stone-gatherer-percentage 0)
)

(defrule
        (goal resource-needed WOOD)
        (current-age == dark-age)
        (civilian-population < 10)
        (not (strategic-number sn-wood-gatherer-percentage == 20) )
=>
        (set-strategic-number sn-wood-gatherer-percentage 20)
        (set-strategic-number sn-food-gatherer-percentage 80)
        (set-strategic-number sn-gold-gatherer-percentage 0)
        (set-strategic-number sn-stone-gatherer-percentage 0)
)

(defrule
        (goal resource-needed NO)
        (current-age == dark-age)
        (civilian-population >= 10)
        (not (strategic-number sn-wood-gatherer-percentage == 30) )
=>
        (set-strategic-number sn-wood-gatherer-percentage 30)
        (set-strategic-number sn-food-gatherer-percentage 70)
)

(defrule
        (goal resource-needed WOOD)
        (current-age == dark-age)
        (civilian-population >= 10)
        (not (strategic-number sn-wood-gatherer-percentage == 40) )
=>
        (set-strategic-number sn-wood-gatherer-percentage 40)
        (set-strategic-number sn-food-gatherer-percentage 60)
)

(defrule
        (goal resource-needed FOOD)
        (current-age == dark-age)
        (civilian-population >= 10)
        (not (strategic-number sn-wood-gatherer-percentage == 20) )
=>
        (set-strategic-number sn-wood-gatherer-percentage 20)
        (set-strategic-number sn-food-gatherer-percentage 80)
)
```

```
(defrule
        (goal resource-needed GOLD)
        (current-age == dark-age)
=>
        (set-strategic-number sn-wood-gatherer-percentage 25)
        (set-strategic-number sn-food-gatherer-percentage 65)
        (set-strategic-number sn-gold-gatherer-percentage 10)
        (disable-self)
)
```

## *How to trade*

### Selling excess resources

```
 (defrule
        (wood-amount > 1200)
        (or
                (food-amount < 1600)
                (or
                        (gold-amount < 1200)
                        (stone-amount < 650)
                )
        )
        (can-sell-commodity wood)
=>
        (chat-local-to-self "excess wood")
        (release-escrow wood)
        (sell-commodity wood)
)

(defrule
        (food-amount > 1700)
        (or
                (wood-amount < 1100)
                (or
                        (gold-amount < 1200)
                        (stone-amount < 650)
                )
        )
        (can-sell-commodity food)
=>
        (chat-local-to-self "excess food")
        (release-escrow food)
        (sell-commodity food)
)

(defrule
        (stone-amount > 1400)
        (or
                (wood-amount < 1100)
                (or
                        (food-amount < 1600)
                        (gold-amount < 1200)
                )
        )
        (can-sell-commodity stone)
=>
        (chat-local-to-self "excess stone")
        (release-escrow stone)
        (sell-commodity stone)
)
```

### Using excess gold to buy cheap resources

```
(defrule
        (gold-amount > 1250)
        (wood-amount < 1100)
        (can-buy-commodity wood)
        (commodity-buying-price wood < 50)

=>
        (chat-local-to-self "excess gold; buy wood")
        (release-escrow gold)
        (buy-commodity wood)
)

(defrule
        (gold-amount > 1250)
        (food-amount < 1600)
        (can-buy-commodity food)
        (commodity-buying-price food < 50)
=>
        (chat-local-to-self "excess gold; buy food")
        (release-escrow gold)
        (buy-commodity food)
)

(defrule
        (gold-amount > 1400)
        (stone-amount < 650)
        (can-buy-commodity stone)
        (commodity-buying-price stone < 200)
=>
        (chat-local-to-self "excess gold; buy stone")
        (release-escrow gold)
        (buy-commodity stone)
)
```

## *How to resign gracefully*

### Detecting the resign conditions

The following code detects resign conditions and sets goal 1 to 19 as a signal to a different group of rules to start resigning. The resign condition is difficulty-dependent.

```
(defrule
        (difficulty >= easy)
        (game-time > 300)
        (soldier-count == 0)
        (unit-type-count villager < five-percent-pop)
        (nand
                (players-stance any-human ally)
                (stance-toward any-human ally)
        )
=>
        (set-goal 1 19)
        (disable-self)
)

(defrule
        (difficulty == moderate)
        (game-time > 300)
        (building-type-count wonder < 1)
        (soldier-count == 0)
        (unit-type-count villager < five-percent-pop)
        (nand
                (players-stance any-human ally)
                (stance-toward any-human ally)
        )
        (not (can-train villager) )
=>
        (set-goal 1 19)
        (disable-self)
)

(defrule
        (difficulty <= hard)
        (game-time > 300)
        (building-type-count wonder < 1)
        (soldier-count == 0)
        (unit-type-count cannon-galleon-line == 0)
        (unit-type-count villager == 0)
        (nand
                (players-stance any-human ally)
                (stance-toward any-human ally)
        )
        (not (can-train villager) )
=>
        (set-goal 1 19)
        (disable-self)
)
```

## Tributing to allies and deleting all buildings before resigning

**Note: Goal 1 set to 19 signals a resign condition.**

```
; Tribute all resources to ally that is still in the game
(defrule
        (goal 1 19)
        (players-population any-ally > 10)
=>
        (release-escrow wood)
        (release-escrow food)
        (release-escrow gold)
        (release-escrow stone)
        (tribute-to-player this-any-ally wood 10000)
        (tribute-to-player this-any-ally food 10000)
        (tribute-to-player this-any-ally gold 10000)
        (tribute-to-player this-any-ally stone 10000)
        (disable-self)
)

;*****************************
;delete all military buildings one at the time
(defrule
        (goal 1 19)
=>
        (delete-building watch-tower)
        (delete-building guard-tower)
        (delete-building keep)
        (delete-building bombard-tower)
        (delete-building castle)
)

; When all military buildings are deleted, resign
(defrule
        (goal 1 19)
        (building-type-count watch-tower == 0)
        (building-type-count guard-tower == 0)
        (building-type-count keep == 0)
        (building-type-count bombard-tower == 0)
        (building-type-count castle == 0)
=>
        (resign)
        (disable-self)
```

)