Starsector Rule Scripting

Rules can be used to power interaction dialogs, and may possibly be extended to other things. The rules are defined in the "Starsector Rules" spreadsheet, which goes into data/campaign.rules.csv.

In dev mode, the game checks for this file being modified and automatically reloads it when that happens. So, for example, it's possible to change rules on the fly without even leaving the currently-open interaction dialog that's using them.

Spreadsheet Data

Triggers

Each rule has a trigger, which defines what type of action may cause this rule to be executed. A trigger can be an arbitrary string. The following triggers are hardcoded:

OpenInteractionDialog - fired when a dialog is first opened
DialogOptionSelected - fired when a dialog option is selected
BeginFleetEncounter - fired when the fleet interaction dialog is opened
OpenCommLink - fired when the player opens a comm link (whether during a fleet encounter, or elsewhere)

Conditions

All conditions must match if a rule is to be executed.

When looking for the "best" matching rule (FireBest command), the rule with the highest score wins. If there's a tie, one is picked randomly. The score for a rule is equal to the number of its conditions, but in addition, each condition may have additional scoring weight specified by appending "score:<integer> to the condition. For example:

```
$respondingToCommRelayInterference score:100
```

Will give this condition a weight of 100.

When looking for "all" matching rules (FireAll command), all matching rules will be executed.

One condition per line.

Script

Expressions to run when the rule is executed. Consist of operations (basic memory manipulation etc) and commands (for more complex tasks, such as making the dialog show an appropriate illustration).

One expression per line.

Text

Text to add to the dialog when the rule is executed. Variations can be separated by an "OR" on a separate line; if this is done, one will be picked randomly. For example:

```
Text one
OR
Text two
```

Token replacement works the same way it does for reports - that is, the game will provide whatever $ tokens are necessary to make the text work. Variables from memory can be used as well; $ tokens provided by the game take precedence if there's a conflict.

Options

The options to show when the rule is executed. If a rule provides options, the current options will be cleared before adding the new ones.

Format:
<sort order>:<option id>:<option text>

The sort order is optional and is used to figure out the order to add options in when all matching rules are picked, as opposed to just one. The option id should be unique for the specific dialog, but doesn't have to be unique across the entire spreadsheet.

A "$option" variable will be set to the id of the option selected when the dialog fires a DialogOptionSelected event.

Scripting

Variables and Memory

Each rule runs in the context of the local memory of the entity that's the subject of the rule. For example, while interacting with a Comm Relay, the rule will by default write/read to/from that specific relay's memory.

There are also other memory contexts available for each rule. Currently, they are:
`global` - global memory accessible from any rule
`faction` - memory for the faction of the entity being interacted with
`market` - the market that this entity belongs to
`player` - similar to global, just used for organizational purposes

In dev mode, the variables can be viewed by interacting with the entity and choosing the "dump memory" option. New variables ("facts") can be added by a CampaignPlugin, using the various updateXXXFacts methods.
More contexts may eventually be added.

A variable token starts with a $, for example, "$menuState". A variable name can also have a prefix to indicate what memory context it's in. For example, "$global.menuState" means the "menuState" variable in the global memory. If there is no prefix (or if the prefix doesn't reference an existing memory context), the variable goes into the local memory by default.

For example:
`$menuState` -> goes into local memory
`$global.menuState` -> goes into global memory
`$blah.menuState` -> goes into local memory, under the name blah.menuState, because there's no memory context named "blah".

Memory Contents

The memory contains whatever is written to it by rules, and the core game can also write "facts" to it. What those are will be driven by what the rules need.

Currently, each entity's local memory contains the tags this entity has, in the form of:
`$tag:<tag>`

For example:
`$tag:comm_relay`
`$tag:station`
etc.

Other "facts" may include things like the faction of the entity, the total fleet points for a fleet, the player fleet points, faction relationships, other evaluations like whether the fleet is stronger than the player fleet, etc. Anything that's needed to make the rules work.

For debugging, when running the game in dev mode, there's an option to dump the memory contents in every rule-based interaction dialog.

Literals

If a token doesn't start with a $, it's considered to be a literal. There's no typing, the game will attempt to treat the literal (or the contents of a variable) as a string, float, or boolean, depending on how it's used.

If a string needs to contain spaces or characters found in operators, it can be quoted using double quotes. If a double quote needs to appear in a string, it can be escaped using a backslash (\). Some examples:
`main` -> the string *main*
`"main menu"` -> the string *main menu*
`"main \"menu\""` -> the string *main "menu"*

Handles

A variable can also be a "handle", which means its pointing to some other kind of Java object. The script doesn't manipulate these directly, but they're used to pass around parameters between different commands. For example, a "Wait" command stores a pointer into a variable, and the AbortWait takes that variable as a parameter to know which Wait to about.

Comments

A script line starting with a # is ignored.

Operations

= (Assignment operator)

Assignment operator. The left hand side must be a variable, the right hand side can be a variable or a literal. Takes an optional extra parameter that makes the variable being assigned to expire after the set time, expressed in game days.
Examples:

```
# the $menuState variable will not expire
$menuState = main

# the $menuState variable will expire immediately (i.e. upon exiting the dialog)
$menuState = main 0

# the $menuState variable will expire in 1.5 game days,
# or around 15 seconds of the game being unpaused
$menuState = main 1.5
```

++, -- (increment/decrement)

Increment/decrement the variable by one. Takes an optional extra parameter that resets the expire time of the variable after its value is updated.
Examples:

```
$timesSawPlayerDoingThing++
# same as above, but the variable will expire in 10 game days
$timesSawPlayerDoingThing++ 10
```

No-op (no operation)

That is, just a variable without an operator. True if and only if the variable's value is "true", not case sensitive. Note that a non-0 number will evaluate to false.
Example:

```
# checks whether $commSnifferInstalled == true
$commSnifferInstalled
```

==, !=, >,<,<=,>= (comparison operators)

Basic comparison operators. Take two parameters, either side can be a variable or a literal.
Examples:

```
$menuState == main
$global.pirateFleetsDefeated >= 10
```

! (not operator)

Not operator. Takes exactly one parameter, which has to be a variable.
Example:

```
!$commSnifferInstalled
```

Commands

A command points to an arbitrary piece of Java code that the script can pass parameters to. Usually, a command will write the results of its execution into memory so that different rules can execute depending on the command's results. For example, the Wait command will set different variables to "true" depending on whether the Wait finished successfully or was interrupted.

The name of the command corresponds to the name of a class found in:
starfarer.api/com/fs/starfarer/api/impl/campaign/rulecmd

The command names are case-sensitive.

More commands will be added as-needed to cover whatever special tasks the rules may need to do.

Wait

Closes the dialog and waits a set amount of time (in game days). The player fleet will be leashed to the entity for the duration (in case it's moving), but the player can override the leash by moving away. Waiting is interrupted by moving away from the entity, or by engaging in combat. Other scripts can also interrupt waiting using the **AbortWait** command.

If the wait finishes successfully, the dialog will be re-opened.

Usage:
```
Wait <handle> <duration> <wait finished> <wait interrupted> <wait in progress>
```

<handle>: a reference to the Wait object will be written to this variable. Used to be able to abort this Wait.
<duration>: in game days
<wait finished>: variable that will be set to true if the wait finishes successfully. Expires immediately
<wait interrupted>: variable that will be set to true if the wait is interrupted. Expires in 2 days.
<wait in progress>: variable that will be set to true while the wait is ongoing. Expires... quickly.

Example:
```
Wait $global.csWait 0.5 $global.csFinished $global.csInterrupted $global.csInProgress
```

Note that all the variables are global; this is needed to make sure that if an outside entity needs to cancel the wait (such as, say, a patrol fleet, during a dialog after catching the player), the have access to them. They wouldn't if the variables were local to the comm relay.

AbortWait

Cancels an in-progress wait and cleans up all of its related variables. Does NOT set the <wait interrupted> variable.
Usage:
```
AbortWait <handle>
```

<handle>: the variable that contains the handle for the Wait that's to be aborted

Example:
```
AbortWait $global.csWait
```

DismissDialog

Close the dialog and unpause the game. No parameters.

expire

Sets the expiration time (in game days) for the specified variable. The variable will be removed from memory after this time. Time will only count while the game is not paused.
Usage:
```
expire <variable> <duration>
```

Example:
```
expire $menuState 1
```

unset

Removes the variable from memory.
Usage:
```
unset <variable>
```

Example:
```
unset $menuState
```

FireAll

Execute all matching rules for the specified trigger.
Usage:
```
FireAll <trigger>
```

<trigger>: can be a variable or a literal.

Example:
```
FireAll PopulateOptions
```

FireBest

Execute the best-matching rule for the specified trigger.
Usage:
```
FireBest <trigger>
```

<trigger>: can be a variable or a literal.

Example:
```
FireBest InitCommRelayDialog
```

PrintDescription
Print the custom description for this entity, if there is one.
Usage:
```
PrintDescription <index>
```

<index>: 1, 2 or 3, determines which descriptions.csv column to print.

Example:
```
PrintDescription 1
```

ShowDefaultVisual
Shows the custom interaction dialog visual for the entity, if that's set. If not, will show the default graphic for the planet. Will be expanded as needed. No parameters.

AddSelector
Usage:
AddSelector <order> <result variable> <text> <color> <min> <max>

SetTooltip

SetShortcut

AddText

SetTextHighlightColors

SetTextHighlights

SetTooltipHighlightColors

SetTooltipHighlights

BroadcastPlayerAction

BroadcastPlayerWaitAction
Etc